Groupe de travail Réseau
Request for Comments : 1321
Traduction Claude Brière de L'Isle

R. Rivest
MIT Laboratory for Computer Science et RSA Data Security, Inc.
avril 1992

Algorithme de résumé de message MD5

Statut du présent mémoire
Le présent mémoire donne des informations pour la communauté de l'Internet. Il ne spécifie pas une norme de l'Internet. La distribution du présent mémoire n'est soumise à aucune restriction.

Remerciements
Nous tenons à remercier Don Coppersmith, Burt Kaliski, Ralph Merkle et Noam Nisan pour leurs nombreux commentaires et suggestions utiles.

Table des matières

1. Résumé

Le présent document décrit l'algorithme de résumé de message MD5. L'algorithme prend en entrée un message de taille arbitraire et produit en sortie une "empreinte" ou "résumé de message de 128 bits de l'entrée. On fait l'hypothèse qu'il est impossible par le calcul de produire deux messages qui ont le même résumé de message, ou de produire un message ayant un résumé de message cible pré spécifié donné. L'algorithme MD5 est destiné aux applications de signature numérique, où un gros fichier doit être "compressé" d'une façon sûre avant d'être chiffré avec une clé privée (secrète) sous un système de chiffrement à clé publique tel que RSA.

L'algorithme MD5 est conçu pour être assez rapide sur les machines à 32 bits. De plus, l'algorithme MD5 n'exige aucun grand tableau de substitution ; l'algorithme peut être codé de façon assez compacte.

L'algorithme MD5 est une extension de l'algorithme de résumé de message MD4 [1], [2]. MD5 est légèrement plus lent que MD4, mais est plus "conservateur" dans sa conception. MD5 a été conçu parce qu'il était estimé que MD4 serait peut-être adopté plus rapidement par les utilisateurs qu'il n'était justifié par les analyses critiques existantes ; comme MD4 a été conçu pour être exceptionnellement rapide, il est "à la marge" en termes de risques d'attaque réussie d'analyse cryptographique. MD5 sauvegarde un bit, abandonne un peu de vitesse au profit d'une meilleure probabilité de sécurité définitive. Il incorpore certaines suggestions faites par divers réviseurs, et contient des optimisations supplémentaires. L'algorithme MD5 est mis dans le domaine public pour être examiné en vue de son adoption possible comme norme.

Pour les applications fondées sur OSI, l'identifiant d'objet de MD5 est

md5 OBJECT IDENTIFIER ::= iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 5}

Dans le type X.509 AlgorithmIdentifier [3], les paramètres pour MD5 devraient avoir le type NULL.

2. Terminologie et notation

Dans le présent document un "mot" est une quantité de 32 bits et un "octet" est une quantité de huit bits. Une séquence de bits peut être interprétée da façon naturelle comme une séquence d'octets, où chaque groupe consécutif de huit bits est interprété comme un octet avec le bit d'ordre élevé (de poids fort) de chaque octet figurant en premier. De même, une séquence d'octets peut être interprétée comme une séquence de mots de 32 bits, où chaque groupe consécutif de quatre octets est interprété comme un mot avec l'octet d'ordre inférieur (de moindre poids) est donné en premier.

Soit x_i qui note "x indice i". Si l'indice est une expression, on l'entoure d'accolades, comme dans x_{i+1}. De même, on utilise ^ pour indiquer l'élévation à la puissance (exponentiation), de sorte que x^i note x à la puissance i.

Soit le symbole "+" qui note l'addition de mots (c'est-à-dire, l'addition modulo 2^32). Soit X <<< s qui note la valeur de 32 bits obtenue par glissement circulaire (rotation) de X à gauche de s positions binaires. Soit not(X) qui note le complément de X, au bit près, et soit X v Y qui note l'opération OU au bit près de X et Y. Soit X xor Y qui note la combinaison par opérateur OUX de X et Y, et soit XY qui note l'opération ET de X et Y.

3. Description de l'algorithme MD5

On commence par supposer qu'on a en entrée un message de b bits, et qu'on souhaite trouver son résumé de message. Ici b est un entier arbitraire non négatif ; b peut être zéro, il n'est pas nécessaire qu'il soit un multiple de huit, et il peut être arbitrairement grand. On imagine que les bits du message sont écrits comme suit :

m_0 m_1 ... m_{b-1}

Les cinq étapes qui suivent sont effectuées pour calculer le résumé de message du message.

3.1 Étape 1. Ajout des bits de bourrage

Le message est "bourré" (étendu) de telle sorte que sa longueur (en bits) soit congruente à 448, modulo 512. C'est à dire que le message est étendu de telle sorte qu'il soit à 64 bits près un multiple d'une longueur de 512 bits. Le bourrage est toujours effectué, même si la longueur du message est déjà congruente à 448, modulo 512.

Le bourrage est effectué comme suit : un seul bit "1" est ajouté au message, puis des bits "0" sont ajoutés de telle sorte que la longueur en bits du message bourré devienne congruente à 448, modulo 512. En tout, au moins un bit et au plus 512 bits sont ajoutés.

3.2 Étape 2. Ajout de la longueur

Une représentation sur 64 bits de b (la longueur du message avant l'ajout des bits de bourrage) est ajoutée au résultat de l'étape précédente. Dans le cas peu vraisemblable où b serait supérieur à 2^64, seuls les 64 bits de moindre poids de b sont utilisés. (Ces bits sont ajoutés comme deux mots de 32 bits et il sont ajoutés avec le mot de moindre poids en premier conformément aux conventions précédentes.)

À ce point, le message résultant (après bourrage de bits et avec b) a une longueur qui est un exact multiple de 512 bits. Autrement dit, ce message a une longueur qui est un exact multiple de 16 mots (de 32 bits). Soit M[0 ... N-1] qui note les mots du message résultant, où N est un multiple de 16.

3.3 Étape 3. Initialiser la mémoire tampon MD

Une mémoire tampon de quatre mots (A,B,C,D) est utilisée pour calculer le résumé de message. Ici, chacun de A, B, C, D est un registre de 32 bits. Ces registres sont initialisés aux valeurs suivantes en hexadécimal, les octets de moindre poids en premier) :

mot A : 01 23 45 67
mot B : 89 ab cd ef
mot C : fe dc ba 98
mot D : 76 54 32 10

3.4 Étape 4. Traitement du message en blocs de 16 mots

On définit d'abord quatre fonctions auxiliaires qui prennent chacune en entrée ces mots de 32 bits et produisent en sortie un mot de 32 bits.

F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))

Dans chaque position binaire F agit comme un conditionnel : si X alors Y autrement Z. La fonction F pourrait avoir été définie en utilisant + à la place de v car XY et not(X)Z n'auront jamais de 1 dans la même position binaire.) Il est intéressant de noter que si les bits de X, Y, et Z sont indépendants et non biaisés, chaque bit de F(X,Y,Z) sera alors indépendant et non biaisé.

Les fonctions G, H, et I sont similaires à la fonction F, en ce qu'elles agissent en "parallèle au bit près" pour produire leur résultat à partir des bits de X, Y, et Z, de telle sorte que si les bits correspondants de X, Y et Z sont indépendants et non biaisés, chaque bit de G(X,Y,Z), H(X,Y,Z) et I(X,Y,Z) sera alors indépendant et non biaisé. Noter que la fonction H est la fonction "oux" ou "de parité" au bit près de ses entrées.

Cette étape utilise un tableau de 64 éléments T[1 ... 64] construit à partir de la fonction sinus. Soit T[i] qui note le i ème élément du tableau, qui est égal à la partie entière de 4 294 967 296 fois abs(sin(i)), où i est en radians. Les éléments du tableau sont donnés dans l'appendice.

Faire ce qui suit :

```
/* Traiter chaque bloc de 16 mots. */
Pour i = 0 à N/16-1 faire

/* Copie le bloc i dans X. */
Pour j = 0 à 15 faire
Régler X[j] à M[i*16+j].
Fin /* de la boucle sur j */

/* Sauvegarder A comme AA, B comme BB, C comme CC et D comme DD. */
AA = A
BB = B
CC = C
DD = D

/* Tour 1. */
/* Soit [abcd k s i] qui note l'opération a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Faire les 16 opérations suivantes. */
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

/* Tour 2. */
/* Soit [abcd k s i] qui note l'opération a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Faire les 16 opérations suivantes. */
[ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
[ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
[ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
[ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]
```

/* Tour 3. */
/* Soit [abcd k s i] qui note l'opération a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* Faire les 16 opérations suivantes. */
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

/* Tour 4. */
/* Soit [abcd k s i] qui note l'opération a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* Faire les 16 opérations suivantes. */
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]

/* Effectuer ensuite les additions suivantes. (Elles incrémentent chacun des quatre registres de la valeur qu'il avait avant le commencement de ce bloc.) */
A = A + AA
B = B + BB
C = C + CC
D = D + DD
Fin /* de la boucle sur i */

3.5 Étape 5. Résultat

Le résumé de message produit en sortie est A, B, C, D. C'est à dire qu'on commence par l'octet de moindre poids de A, et qu'on termine par l'octet de poids fort de D.

Cela achève la description de MD5. Une mise en œuvre de référence en C est donnée dans l'appendice.

4. Résumé

L'algorithme de résumé de message MD5 est simple à mettre en œuvre, et produit une "empreinte" ou résumé de message d'un message de longueur arbitraire. On a fait l'hypothèse que la difficulté d'arriver à ce que deux messages aient le même résumé de message est de l'ordre de 2^64 opérations, et que la difficulté d'arriver à ce qu'un message ait un résumé de message donné est de l'ordre de 2^128 opérations. L'algorithme MD5 a été étudié avec soin à la recherche de faiblesses. Il est cependant un algorithme relativement nouveau et d'autres analyses de sa sécurité sont bien sûr justifiées, comme c'est le cas avec toute nouvelle proposition de cette sorte.

5. Différences entre MD4 et MD5

Les différences entre MD4 et MD5 sont énumérées ci-après :

1. Un quatrième tour a été ajouté.
2. Chaque étape a maintenant une constante additive unique.
3. La fonction g dans le tour 2 a changé de (XY v XZ v YZ) à (XZ v Y not(Z)) pour rendre g moins symétrique.
4. Chaque étape ajoute maintenant le résultat de l'étape précédente. Cela provoque un "effet d'avalanche" plus rapide.
5. L'ordre dans lequel on accède aux mots d'entrée dans les tours 2 et 3 est changé, pour rendre ces schémas moins semblables les uns aux autres.
6. Les glissements de quantités dans chaque tour ont été approximativement optimisés, pour donner un "effet d'avalanche" plus rapide. Les glissements dans les différents tours sont distincts.

Références

[1] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1320, MIT and RSA Data Security, Inc., April 1992.

[2] Rivest, R., "The MD4 message digest algorithm", in A.J. Menezes and S.A. Vanstone, editors, Advances in Cryptology - CRYPTO '90 Proceedings, pages 303-311, Springer-Verlag, 1991.

[3] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework."

APPENDICE A – Mise en œuvre de référence

Cet appendice contient les fichiers suivants tirés de RSAREF : une trousse à outils cryptographique pour la messagerie à confidentialité améliorée :
global.h – fichier d'en-tête global
md5.h – fichier d'en-tête pour MD5
md5c.c -- code source pour MD5

Pour des informations complémentaires sur RSAREF, envoyer un message à <rsaref@rsa.com>.

L'appendice comporte aussi le fichier suivant :
mddriver.c – pilote d'essais pour MD2, MD4 et MD5

Le pilote compile pour MD5 par défaut mais peut compiler pour MD2 ou MD4 si le symbole MD est défini sur la ligne de commande de compilateur C comme 2 ou 4.

La mise en œuvre est portable et devrait fonctionner sur différentes plates-formes. Cependant, il n'est pas difficile d'optimiser la mise en œuvre sur des plates-formes particulières, exercice laissé aux soins du lecteur. Par exemple, sur des plates-formes "petites boutiennes" où le dernier octet adressé dans un mot de 32 bits est celui de moindre poids et où il n'y a pas de restriction d'alignement, l'invocation du décodage de la transformation MD5Transform peut être remplacée par un "typecast".

A.1 global.h

```
/* Types et constantes de GLOBAL.H - RSAREF */

/* PROTOTYPES devrait être réglé à un si et seulement si le compilateur prend en charge le prototypage d'argument de fonction. Ce qui suit fait revenir PROTOTYPES à 0 par défaut si il n'a pas déjà été défini avec les fanions de compilateur C. */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER définit un type de pointeur générique. */
typedef unsigned char *POINTER;

/* UINT2 définit un mot de deux octets. */
typedef unsigned short int UINT2;

/* UINT4 définit un mot de quatre octets. */
typedef unsigned long int UINT4;

/* PROTO_LIST est défini en fonction de la façon dont PROTOTYPES est défini ci-dessus. Si on utilise PROTOTYPES, PROTO_LIST retourne alors la list, autrement, il retourne une liste vide. */
#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
```

A.2 md5.h

```
/* MD5.H – fichier d'en-tête pour MD5C.C */
```

```
/* Copyright (C) 1991-2, RSA Data Security, Inc. Créée en 1991. Tous droits réservés.

Licence de copier et utiliser ce logiciel est accordée pourvu qu'il soit identifié comme "algorithme de résumé de message
MD5 de RSA Data Security, Inc." dans tous les matériaux mentionnant ou faisant référence à ce logiciel où à sa fonction.

Licence est aussi accordée de faire et utiliser des travaux dérivés pourvu que de tels travaux soient identifiés comme
"dérivés de l'algorithme de résumé de message MD5 de RSA Data Security, Inc." dans tous les matériaux mentionnant ou
faisant référence à ces travaux dérivés.

RSA Data Security, Inc. n'offre aucune garantie concernant la possibilité de commercialiser le présent logiciel ou celle de
l'adapter à aucun objet particulier. Il est fourni "en l'état" sans garantie expresse ou implicite d'aucune sorte.

Ces notices doivent être conservées dans toutes copies de toute partie de cette documentation et/ou logiciel. */

/* Contexte MD5. */
typedef struct {
UINT4 state[4];                 /* état (ABCD) */
UINT4 count[2];                 /* nombre de bits, modulo 2^64 (lsb en premier) */
unsigned char buffer[64];       /* mémoire tampon d'entrée */
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
((MD5_CTX *, unsigned char *, unsigned int));
void MD5Final PROTO_LIST ((unsigned char [16], MD5_CTX *));
```

A.3 md5c.c

```
/* MD5C.C - algorithme de résumé de message MD5 de RSA Data Security, Inc. */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Créée en 1991. Tous droits réservés.

Licence de copier et utiliser ce logiciel est accordée pourvu qu'il soit identifié comme "algorithme de résumé de message
MD5 de RSA Data Security, Inc." dans tous les matériaux mentionnant ou faisant référence à ce logiciel où à sa fonction.

Licence est aussi accordée de faire et utiliser des travaux dérivés pourvu que de tels travaux soient identifiés comme
"dérivés de l'algorithme de résumé de message MD5 de RSA Data Security, Inc." dans tous les matériaux mentionnant ou
faisant référence à ces travaux dérivés.

RSA Data Security, Inc. n'offre aucune garantie concernant la possibilité de commercialiser le présent logiciel ou celle de
l'adapter à aucun objet particulier. Il est fourni "en l'état" sans garantie expresse ou implicite d'aucune sorte.

Ces notices doivent être conservées dans toutes copies de toute partie de cette documentation et/ou logiciel. */

#include "global.h"
#include "md5.h"

/* Constantes pour le sous-programme MD5Transform. */

#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
```

```
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST ((UINT4 *, unsigned char *, unsigned int));
static void MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = { 0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H et I sont les fonctions MD5 de base. */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT fait une rotation de x de n bits à gauche. */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* Transformations FF, GG, HH et II pour les tours 1, 2, 3 et 4. La rotation est séparée de l'addition pour empêcher le
recalcul. */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \

 (a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* Initialisation de MD5. Commence une opération MD5, en écrivant un nouveau contexte. */
void MD5Init (context)
MD5_CTX *context;                   /* contexte */
{
context->count[0] = context->count[1] = 0;
/* Charge les constantes magiques d'initialisation. */
context->state[0] = 0x67452301;
context->state[1] = 0xefcdab89;
context->state[2] = 0x98badcfe;
context->state[3] = 0x10325476;
}

/* OPération de mise à jour de bloc MD5. Continue une opération de résumé de message MD5, traite un autre bloc de
message, et met à jour le contexte. */
```

```
void MD5Update (context, input, inputLen)
MD5_CTX *context;                /* contexte. */
unsigned char *input;            /* bloc d'entrée. */
unsigned int inputLen;           /* longueur du bloc d'entrée. */
{
unsigned int i, index, partLen;

/* Calcule le nombre d'octets mod 64 */
index = (unsigned int)((context->count[0] >> 3) & 0x3F);

/* Met à jour le nombre de bits. */
if ((context->count[0] += ((UINT4)inputLen << 3))  < ((UINT4)inputLen << 3)) context->count[1]++;
context->count[1] += ((UINT4)inputLen >> 29);
partLen = 64 - index;

/* Transforme autant de fois que possible. */
if (inputLen >= partLen) {
MD5_memcpy ((POINTER)&context->buffer[index], (POINTER)input, partLen);
MD5Transform (context->state, context->buffer);

for (i = partLen; i + 63 < inputLen; i += 64) MD5Transform (context->state, &input[i]);
index = 0;
  }
else
i = 0;

/* Entrée restante de la mémoire tampon. */
MD5_memcpy ((POINTER)&context->buffer[index], (POINTER)&input[i],
inputLen-i);
}

/* Finalisation de MD5. Termine une opération de résumé de message MD5, écrit le résumé de message et met le contexte
à zéro. */
void MD5Final (digest, context)
unsigned char digest[16];        /* résumé de message. */
MD5_CTX *context;                /* contexte. */
{
unsigned char bits[8];
unsigned int index, padLen;

/* Sauvegarde le nombre de bits. */
Encode (bits, context->count, 8);

/* Bourre à 56 mod 64.*/
index = (unsigned int)((context->count[0] >> 3) & 0x3f);
padLen = (index < 56) ? (56 - index) : (120 - index);
MD5Update (context, PADDING, padLen);

/* Ajoute la longueur (avant bourrage). */
MD5Update (context, bits, 8);

/* Mémorise l'état dans le résumé. */
Encode (digest, context->state, 16);

/* Met à zéro les informations sensibles.*/
MD5_memset ((POINTER)context, 0, sizeof (*context));
}

/* Transformation MD5 de base. Transforme l'état sur la base du bloc. */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
```

```
UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

Decode (x, block, 64);

/* Tour 1 */
FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

/* Tour 2 */
GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

/* Tour 3 */
HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

/* Tour 4 */
II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
```

```
II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

state[0] += a;
state[1] += b;
state[2] += c;
state[3] += d;

/* Mise à zéro des informations sensibles.*/
MD5_memset ((POINTER)x, 0, sizeof (x));
}

/* Code l'entrée (UINT4) en sortie (caractères non signés). Suppose que la longueur est un multiple de 4. */
static void Encode (output, input, len) unsigned char *output;
UINT4 *input;
unsigned int len;
{
unsigned int i, j;

for (i = 0, j = 0; j < len; i++, j += 4) {
output[j] = (unsigned char)(input[i] & 0xff);
output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
}
}

/* Décode l'entrée (caractères non signés) en sortie (UINT4). Suppose que la longueur est un multiple de 4. */
static void Decode (output, input, len) UINT4 *output;
unsigned char *input;
unsigned int len;
{
unsigned int i, j;

for (i = 0, j = 0; j < len; i++, j += 4)
output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) | (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Note : Remplacer "for loop" par memcpy standard si possible. */

static void MD5_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
unsigned int i;

for (i = 0; i < len; i++)
output[i] = input[i];
}

/* Note : Remplacer "for loop" par memcpy standard si possible. */
static void MD5_memset (output, value, len)
```

```
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 ((char *)output)[i] = (char)value;
}
```

A.4 mddriver.c

```
/* MDDRIVER.C – pilote d'essais pour MD2, MD4 et MD5 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. Créée en 1990. Tous droits réservés.
RSA Data Security, Inc. n'offre aucune garantie concernant la possibilité de commercialiser le présent logiciel ou celle de
l'adapter à aucun objet particulier. Il est fourni "en l'état" sans garantie expresse ou implicite d'aucune sorte.

Ces notices doivent être conservées dans toutes copies de toute partie de cette documentation et/ou logiciel. */

/* Ce qui suit fait revenir MD par défaut à MD5 si il n'a pas déjà été défini avec les fanions de compilateur C. */
#ifndef MD
#define MD5
#endif

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "global.h"
#if MD == 2
#include "md2.h"
#endif
#if MD == 4
#include "md4.h"
#endif
#if MD == 5
#include "md5.h"
#endif

/* Longueur du bloc d'essai, nombre de blocs d'essai. */
#define TEST_BLOCK_LEN 1000
#define TEST_BLOCK_COUNT 1000

static void MDString PROTO_LIST ((char *));
static void MDTimeTrial PROTO_LIST ((void));
static void MDTestSuite PROTO_LIST ((void));
static void MDFile PROTO_LIST ((char *));
static void MDFilter PROTO_LIST ((void));
static void MDPrint PROTO_LIST ((unsigned char [16]));

#if MD == 2
#define MD_CTX MD2_CTX
#define MDInit MD2Init
#define MDUpdate MD2Update
#define MDFinal MD2Final
#endif
#if MD == 4
#define MD_CTX MD4_CTX
#define MDInit MD4Init
#define MDUpdate MD4Update
#define MDFinal MD4Final
```

```
#endif
#if MD == 5
#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
#endif

/* Pilote principal.
Arguments (peut être une combinaison quelconque) :
-sstring        - chaîne de résumés
-t              - donne la durée de l'essai
-x              - donne le desscriptif de l'essai
filename        - fichier des résumés
(none)          - entrée standard des résumés */
int main (argc, argv)
int argc;

char *argv[];
{
  int i;

  if (argc > 1)
 for (i = 1; i < argc; i++)
   if (argv[i][0] == '-' && argv[i][1] == 's')
     MDString (argv[i] + 2);
   else if (strcmp (argv[i], "-t") == 0)
     MDTimeTrial ();
   else if (strcmp (argv[i], "-x") == 0)
     MDTestSuite ();
   else
     MDFile (argv[i]);
  else
 MDFilter ();

  return (0);
}

/* Résume une chaîne et imprime le résultat. */
static void MDString (string)
char *string;
{
MD_CTX context;
unsigned char digest[16];
unsigned int len = strlen (string);

MDInit (&context);
MDUpdate (&context, string, len);
MDFinal (digest, &context);

printf ("MD%d (\"%s\") = ", MD, string);
MDPrint (digest);
printf ("\n");
}

/* Mesures le temps pour résumer les blocs TEST_BLOCK_COUNT TEST_BLOCK_LEN-byte. */
static void MDTimeTrial ()
{
MD_CTX context;
time_t endTime, startTime;
unsigned char block[TEST_BLOCK_LEN], digest[16];
unsigned int i;
```

```
 printf
("MD%d time trial. Digesting %d %d-byte blocks ...", MD, TEST_BLOCK_COUNT, TEST_BLOCK_LEN);

/* Initialise le bloc. */
for (i = 0; i < TEST_BLOCK_LEN; i++)
 block[i] = (unsigned char)(i & 0xff);

/* Lance le temporisateur. */
time (&startTime);

/* Blocs de résumé. */
MDInit (&context);
for (i = 0; i < TEST_BLOCK_COUNT; i++) MDUpdate (&context, block, TEST_BLOCK_LEN);
MDFinal (digest, &context);

/* Arrête le temporisateur */
time (&endTime);

printf (" done\n");
printf ("Digest = ");
MDPrint (digest);
printf ("\nTime = %ld seconds\n", (long)(endTime-startTime));
printf
("Speed = %ld bytes/second\n",
(long)TEST_BLOCK_LEN * (long)TEST_BLOCK_COUNT/(endTime-startTime));
}

/* Résume une suite de référence de chaînes et imprime le résultat. */
static void MDTestSuite ()
{
printf ("MD%d test suite:\n", MD);

MDString ("");
MDString ("a");
MDString ("abc");
MDString ("message digest");
MDString ("abcdefghijklmnopqrstuvwxyz");
MDString ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
MDString ("12345678901234567890123456789012345678901\12345678901234567890123456789012345678901234567890");
}

/* Résume un fichier et imprime le résultat. */
static void MDFile (filename)
char *filename;
{
FILE *file;
MD_CTX context;
int len;
unsigned char buffer[1024], digest[16];

if ((file = fopen (filename, "rb")) == NULL)
 printf ("%s can't be opened\n", filename);

else {
MDInit (&context);
while (len = fread (buffer, 1, 1024, file))
MDUpdate (&context, buffer, len);
MDFinal (digest, &context);

fclose (file);

printf ("MD%d (%s) = ", MD, filename);
MDPrint (digest);
```

```
printf ("\n");
}
}
/* Résume l'entrée standard et imprime le résultat. */
static void MDFilter ()
{
MD_CTX context;
int len;
unsigned char buffer[16], digest[16];

MDInit (&context);
while (len = fread (buffer, 1, 16, stdin))
MDUpdate (&context, buffer, len);
MDFinal (digest, &context);

MDPrint (digest);
printf ("\n");
}
/* Imprime un résumé de message en hexadécimal. */
static void MDPrint (digest)
unsigned char digest[16];
{

  unsigned int i;

  for (i = 0; i < 16; i++)
 printf ("%02x", digest[i]);
}
```

A.5 Suite d'essais

La suite d'essais MD5 (option de pilote "-x") devrait imprimer le résultat suivant :

```
MD5 test suite:
MD5 ("") = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
d174ab98d277d9f5a5611c2c9f419d9f
MD5 ("12345678901234567890123456789012345678901234567890123456789012345678
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

Considérations pour la sécurité

Le niveau de sécurité exposé dans le présent mémoire est considéré comme suffisant pour la mise en œuvre de schémas de signatures numériques hybrides de très haute sécurité fondés sur MD5 et un système de chiffrement à clé publique.

Adresse de l'auteur

Ronald L. Rivest
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-324
545 Technology Square
Cambridge, MA 02139-1986
téléphone : (617) 253-5880
mél : rivest@theory.lcs.mit.edu