NORME INTERNATIONALE

ISO 8731-2

Deuxième édition
1992-09-15

Banque — Algorithmes approuvés pour l'authentification des messages —

Partie 2: Algorithme authentificateur de message

Banking — Approved algorithms for message authentication —

Part 2: Message authenticator algorithm

Numéro de référence
ISO 8731-2:1992(F)

Sommaire

Page

1 Domaine d'application **1**

2 Références normatives **1**

3 Brève description **1**

3.1 Généralités **1**

3.2 Aspects techniques **1**

4 L'algorithme applicable à un segment **2**

4.1 Définitions des fonctions utilisées dans l'algorithme **2**

4.2 Spécification de l'algorithme **3**

5 Spécification du mode opératoire **4**

Annexes

A Exemples d'essais pour la mise en œuvre de l'algorithme **6**

B Spécification de MAA en VDM **12**

Organisation internationale de normalisation
Case Postale 56 • CH-1211 Genève 20 • Suisse
Version française tirée en 1993

Imprimé en Suisse

Avant-propos

L'ISO (Organisation internationale de normalisation) est une fédération mondiale d'organismes nationaux de normalisation (comités membres de l'ISO). L'élaboration des Normes internationales est en général confiée aux comités techniques de l'ISO. Chaque comité membre intéressé par une étude a le droit de faire partie du comité technique créé à cet effet. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'ISO participent également aux travaux. L'ISO collabore étroitement avec la Commission électrotechnique internationale (CEI) en ce qui concerne la normalisation électrotechnique.

Les projets de Normes internationales adoptés par les comités techniques sont soumis aux comités membres pour vote. Leur publication comme Normes internationales requiert l'approbation de 75 % au moins des comités membres votants.

La Norme internationale ISO 8731-2 a été élaborée par le comité technique ISO/TC 68, *Banque et services financiers liés aux opérations bancaires*, sous-comité SC 2, *Opérations et procédures*.

Cette deuxième édition annule et remplace la première édition (ISO 8731-2:1987), dont elle constitue une révision technique.

L'ISO 8731 comprend les parties suivantes, présentées sous le titre général *Banque — Algorithmes approuvés pour l'authentification des messages*:

— *Partie 1: DEA*

— *Partie 2: Algorithme authentificateur de message*

Les annexes A et B de la présente partie de l'ISO 8731 sont données uniquement à titre d'information.

Page blanche

Banque — Algorithmes approuvés pour l'authentification des messages —

Partie 2:
Algorithme authentificateur de message

1 Domaine d'application

L'ISO 8731, spécifie, en plusieurs parties distinctes, les algorithmes d'authentification approuvés, c'est-à-dire ceux qui ont été approuvés comme étant conformes aux spécifications d'authentification fixées dans l'ISO 8730. La présente partie de l'ISO 8731 traite de l'algorithme authentificateur de message à utiliser dans le calcul du code d'authentification de message (MAC – Message Authentication Code).

L'algorithme authentificateur de message (MAA – Message Authenticator Algorithm) est spécialement conçu pour réaliser une authentification très rapide en utilisant un ordinateur de grande capacité. Cet algorithme est spécialement conçu pour être utilisé lorsqu'il y a des volumes importants de données et lorsque la mise en œuvre efficace par logiciel et souhaitée. Le MAA est aussi adapté pour être utilisé avec un calculateur programmable.

Les exemples d'essais sont donnés en annexe A qui ne fait pas partie intégrante de la présente partie de l'ISO 8731. Un exemple d'essai complémentaire est donné dans une annexe de l'ISO 8730.

Une spécification du MAA en VDM est donnée en annexe B, qui ne fait pas partie intégrante de la présente partie de l'ISO 8731.

2 Références normatives

Les normes suivantes contiennent des dispositions qui, par suite de la référence qui en est faite, constituent des dispositions valables pour la présente partie de l'ISO 8731. Au moment de la publication, les éditions indiquées étaient en vigueur. Toute norme est sujette à révision et les parties prenantes des accords fondés sur la présente partie de l'ISO 8731 sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des normes indiquées ci-après. Les membres de la CEI et de l'ISO possèdent le registre des Normes internationales en vigueur à un moment donné.

ISO 7185:1990, *Technologies de l'information — Langages de programmation — Pascal (Publiée actuellement en anglais seulement).*

ISO 8730:1990, *Opérations bancaires — Spécifications liées à l'authentification des messages (service aux entreprises).*

3 Brève description

3.1 Généralités

L'algorithme authentificateur de message (MAA) repose sur le principe du code d'authentification de message (MAC), qui est un nombre envoyé avec le message, de telle sorte que le destinataire du message puisse contrôler que celui-ci n'a fait l'objet d'aucune modification depuis son envoi par l'expéditeur.

3.2 Aspects techniques

Tous les nombres manipulés par cet algorithme doivent être considérés comme des entiers non signés de 32 bits sauf indication contraire. Pour un tel nombre N, $0 < N < 2^{32}$. Cet algorithme peut être mis en œuvre de façon aisée et efficace dans un calculateur comportant des mots de 32 bits de long ou plus.

Les messages à authentifier peuvent être formés d'une chaîne de bits de n'importe quelle longueur. Ils devront être introduits dans l'algorithme comme une suite de nombre de 32 bits, M_1, $M_2 - M_n$ au nombre de n, appelés «blocs de messages». La façon détaillée dont il convient de compléter le dernier bloc M_n jus-

qu'à une longueur de 32 bits ne fait pas partie de l'algorithme mais doit être définie dans toute application. Cet algorithme ne doit pas être utilisé pour authentifier des messages comportant plus de 1 000 000 de blocs, c'est-à-dire $n < 1\ 000\ 000$.

La clé doit être composée de 2 nombres de 32 bits J et K et a donc une taille de 64 bits.

Le résultat de l'algorithme est une valeur d'authentification de 32 bits. Le calcul peut être effectué sur des messages ne comportant qu'un seul bloc ($n = 1$).

Les messages comportant plus de 256 blocs de messages doivent être divisés en segments de 256 blocs, excepté le dernier segment qui peut comporter moins de 256 blocs de messages.

L'article 4 donne la spécification de l'algorithme applicable à un segment. Si le message complet n'est composé que d'un seul segment, cet algorithme permet d'effectuer le calcul complet, et le résultat du calcul (Z) est la valeur d'authentification. S'il y a plus de 256 blocs de messages, il conviendra d'utiliser le mode d'opération spécifié dans l'article 5.

L'algorithme de segment comporte trois parties:

a) le «prélude» doit être un calcul effectué avec les seules parties de clés (J et K) et il doit en résulter 6 nombres X_0, Y_0, V_0, W, S et T qui seront utilisés dans les calculs suivants. Ce calcul n'a pas besoin d'être répété tant qu'une nouvelle clé n'est pas mise en œuvre;

b) la «boucle principale» est un calcul qui doit être répété pour chaque bloc de message M, et en conséquence constitue la part la plus importante du calcul dans le cas des messages longs;

c) la «coda» doit consister en deux opérations de la «boucle principale» utilisant comme «blocs de message» les deux nombres S et T à tour de rôle, suivi d'un simple calcul de Z.

L'utilisation de la boucle principale (voir article 5) est un élément essentiel de l'application de cet algorithme.

La figure 1 montre le flux de données de façon schématique.

4 L'algorithme applicable à un segment

4.1 Définitions des fonctions utilisées dans l'algorithme

4.1.1 Définitions générales

Un certain nombre de fonctions sont utilisées dans la description de l'algorithme. Dans ce qui suit, X et Y sont des nombres de 32 bits ainsi que le résultat, sauf spécification contraire.

CYC(X) est le résultat d'une rotation circulaire d'un bit vers la gauche de X.

ET(X,Y) est le résultat de l'opération logique ET effectuée sur chacun des 32 bits.

OU(X,Y) est le résultat de l'opération logique OU effectuée sur chacun des 32 bits.

X/OU(X,Y) est le résultat de l'opération OU exclusif (addition modulo 2) effectuée sur chacun des 32 bits.

ADD(X,Y) est le résultat de l'addition de X et Y sans tenir compte de la retenue du 32e bit, c'est-à-dire l'addition modulo 2^{32}.

RET(X,Y) est la valeur de la retenue du 32e bit quand X est ajouté à Y. Elle peut prendre la value 0 ou 1.

MUL1(X,Y), MUL2(X,Y) et MUL2A(X,Y) sont les trois formes différentes de multiplication dont le résultat est sur 32 bits.

[X||Y] est le résultat de la concaténation des nombres binaires X et Y, avec cadrage à gauche ou sur la position la plus significative. La notation est étendue pour concaténer plus de deux nombres et est appliquée aussi à des octets et à des nombres plus longs que 32 bits.

4.1.2 Définitions des fonctions de multiplication

Pour expliquer ces multiplications, prenons le produit sur 64 bits de X et de Y soit [U||L]. En conséquence, U est la moitié supérieure (poids fort) du produit et L la moitié inférieure (poids faible).

4.1.2.1 Définition de MUL1(X,Y)

Multiplier X et Y pour obtenir [U||L]. S et C sont ici des variables locales.

$S := ADD(U,L);$ . . . (1)

$C := RET(U,L);$ . . . (2)

$MUL1(X,Y) := ADD(S,C).$ . . . (3)

C'est-à-dire que U doit être ajouté à L avec une retenue finale.

Numériquement, le résultat est congru à X*Y, le produit de X par Y, modulo $(2^{32} - 1)$. Cela n'est pas nécessairement le plus petit résidu car il peut être égal à $2^{32} - 1$.

4.1.2.2 Définition de MUL2(X,Y)

Cette forme de multiplication ne doit pas être utilisée dans la boucle principale, mais seulement dans le prélude. D,E,F,S, et C sont ici des variables locales.

$$D := ADD(U,U); \quad \ldots (4)$$

$$E := RET(U,U); \quad \ldots (5)$$

$$F := ADD(D,2E); \quad \ldots (6)$$

$$S := ADD(F,L); \quad \ldots (7)$$

$$C := RET(F,L); \quad \ldots (8)$$

$$MUL2(X,Y) := ADD(S,2C). \quad \ldots (9)$$

Numériquement, le résultat est congru à X*Y, le produit de X par Y, modulo $(2^{32} - 2)$. Ce n'est pas nécessairement le plus petit résidu car il peut être égal à $2^{32} - 1$ ou à $2^{32} - 2$.

4.1.2.3 Définition de MUL2A(X,Y)

Il s'agit d'une forme simplifiée de MUL2(X,Y) utilisée dans la boucle principale, qui donne le bon résultat seulement dans le cas où au moins un des nombres X ou Y comporte un zéro dans son bit de poids fort.

Cette forme de multiplication est employée pour faire des économies dans le traitement. D, S, et C sont ici des variables locales.

$$D := ADD(U,U); \quad \ldots (10)$$

$$S := ADD(D,L); \quad \ldots (11)$$

$$C := RET(D,L); \quad \ldots (12)$$

$$MUL2A(X,Y) := ADD(S,2C). \quad \ldots (13)$$

Ce résultat est congru à X*Y modulo $(2^{32} - 2)$ dans les conditions spécifiées car dans la notation de MUL2(X,Y) ci-dessus, la retenue E = 0.

4.1.3 Définition des fonctions BYT[X||Y] et PAT[X||Y]

Une procédure est utilisée dans le prélude pour conditionner à la fois les parties de la clé et les résultats de façon à éviter les longues chaînes de «un» ou de «zéro». Elle donne deux résultats qui sont les valeurs conditionnées de X et Y et un nombre PAT[X,Y] qui enregistre les modifications qui ont été effectuées. PAT[X,Y] < 255 de façon à être essentiellement un nombre de 8 bits.

X et Y sont considérés comme des chaînes d'octets.

En utilisant la notation (X,Y...) pour concaténer,

$$[X||Y] = [B_0||\ B_1||\ B_2||\ B_3||\ B_4||\ B_5||\ B_6||\ B_7]$$

Donc les octets B_0 à B_3 sont dérivés de X et les octets B_4 à B_7 sont dérivés de Y.

La procédure est mieux décrite par un programme où chaque octet B_i est considéré comme étant un nombre entier d'une longueur de 8 bits.

```
P := 0
for i := 0 to 7 do
begin
   P := 2*P;
   if B[i]= 0 then
   begin
      P := P + 1;
      B'[i] := P
   end
   else
      if B[i]= 255 then
      begin
         P : = P + 1;
         B'[i] := 255 - P
      end
      else
         B'[i] := B[i];
end
end;
```

NOTE 1 La procédure est écrite en langage de programmation PASCAL (voir ISO 7185), sauf que l'identificateur non normalisé B' a été utilisé pour maintenir la continuité du texte. Les symboles B[i] et B'[i] correspondent à B_i et B'_i dans le texte.

Les résultats sont

$$BYT[X,Y] = [B'_0||\ B'_1||\ B'_2||\ B'_3||\ B'_4||\ B'_5||\ B'_6||\ B'_7]$$

et

$$PAT[X,Y] = P$$

4.2 Spécification de l'algorithme

4.2.1 Le prélude

$$[J_1||K_1] := BYT[J||K];$$

$$P := PAT[J||K];$$

$$Q := (1 + P)*(1 + P). \quad \ldots (14)$$

En premier lieu, un calcul utilisant J_1 donne H_4, H_6, et H_8 desquels on déduit X_0, V_0 et S.

$$J1_2 := MUL1(J_1,J_1); \quad J2_2 := MUL2(J_1,J_1);$$

$$J1_4 := MUL1(J1_2,J1_2); \quad J2_4 := MUL2(J2_2,J2_2);$$

$$J1_6 := MUL1(J1_2,J1_4); \quad J2_6 := MUL2(J2_2,J2_4);$$

$$J1_8 := MUL1(J1_2,J1_6); \quad J2_8 := MUL2(J2_2,J2_6). \quad \ldots (15)$$

$H_4 := X/OU(J1_4,J2_4);$

$H_6 := X/OU(J1_6,J2_6);$

$H_8 := X/OU(J1_8,J2_8).$ . . . (16)

Un calcul similaire utilisant K_1 donne H_5, H_7 et H_9 desquels Y_0, W et T sont dérivés.

$K1_2 := MUL1(K_1,K_1);$ $K2_2 := MUL2(K_1,K_1);$

$K1_4 := MUL1(K1_2,K1_2);$ $K2_4 := MUL2(K2_2,K2_2);$

$K1_5 := MUL1(K_1,K1_4);$ $K2_5 := MUL2(K_1,K2_4);$

$K1_7 := MUL1(K1_2,K1_5);$ $K27 := MUL2(K2_2,K2_5);$

$K1_9 := MUL1(K1_2,K1_7);$ $K2_9 := MUL2(K2_2,K2_7).$ . . . (17)

$H' := X/OU(K1_5,K2_5);$

$H_5 := MUL2(H',Q);$

$H_7 := X/OU(K1_7,K2_7);$

$H_9 := X/OU(K1_9,K2_9).$ . . . (19)

En fin de compte, les résultats doivent être conditionnés par la fonction BYT.

$[X_0||Y_0] := BYT[H_4||H_5];$
$[V_0||W] := BYT[H_6||H_7];$
$[S||T] := BYT[H_8||H_9].$ . . . (20)

4.2.2 La boucle principale

Cette boucle doit être réitérée pour chacun des blocs de message M_i. En plus de M_i, les valeurs principales employées doivent être X et Y et les résultats principaux sont les nouvelles valeurs de X et Y. Elle doit utiliser également V et W et modifier V à chaque itération. Notons que X, Y et V sont initialisés avec les valeurs fournies par le prélude.

Pour pouvoir réutiliser les mêmes clés, les valeurs initiales de X, Y et V doivent être conservées, en conséquence, elles sont appelées X_0, Y_0 et V_0 et il doit y avoir un pas d'initialisation $X := X_0$, $Y := Y_0$, $V := V_0$, après lequel la boucle principale doit être effectuée pour la première fois.

NOTE 2 Le programme est présenté en colonnes pour clarifier son opération «parallèle» mais il devrait être lu dans l'ordre normal de lecture, de gauche à droite de chaque ligne.

$V := CYC(V);$

$E := X/OU(V,W);$ . . . (21)

$X := X/OU(X,M_i);$ $Y := X/OU(Y,M_i);$ . . . (22)

$F := ADD(E,Y);$ $G := ADD(E,X);$

$F := OU(F,A);$ $G := OU(G,B);$

$F := ET(F,C);$ $G := ET(G,D);$ . . . (23)

$X := MUL1(X,F);$ $Y := MUL2A(Y,G).$ . . . (24)

Les nombres A, B, C, D sont des constantes qui sont, en notation hexadécimale:

Constante A: 0204 0801
Constante B: 0080 4021
Constante C: BFEF 7FDF
Constante D: 7DFE FBFF

NOTE 3 Les lignes (21) sont communes aux deux chemins. La ligne (22) introduit le bloc de message M_i. Les lignes (23) préparent les multiplicateurs et la ligne (24) génère de nouvelles valeurs de X et Y. Seuls X, Y et V sont modifiés pour leur utilisation dans le cycle suivant. F et G sont des variables locales. Comme la constante D a zéro pour bit de poids fort, $G < 2^{31}$, cela assure que MUL2A à la ligne (24) donnera le bon résultat.

4.2.3 La coda

Après que le dernier bloc de message du segment ait été traité, la coda doit être réalisée en appliquant la boucle principale au bloc de message S, puis au bloc T. Après cela le résultat Z = X/OU(X,Y) doit être calculé. Ceci termine la coda. Si le message ne contient pas plus de 256 blocs de messages, Z est la valeur du MAC. Sinon la valeur de Z doit être utilisée dans le mode opératoire spécifié dans l'article 5.

NOTE 4 Les valeurs X_0, Y_0, V_0, W, S et T devraient être mémorisées de façon à calculer des valeurs ultérieures de Z sans avoir à répéter le prélude (calcul à l'aide de la clé) tant que la clé n'aura pas été changée.

5 Spécification du mode opératoire

Les messages comportant plus de 256 blocs de messages doivent être divisés en segments SEG_1, SEG_2, ... SEG_s de 256 blocs chacun sauf le dernier segment qui peut avoir de 1 à 256 blocs. Le nombre de segments est s.

Le résultat Z d'un algorithme applicable à un segment, tel que spécifié dans l'article 4, appliqué aux éléments de clé J, K et au message M, doit être noté Z(J,K,M).

Pour calculer le MAC d'un message excédant 256 blocs, le mode opératoire doit utiliser l'algorithme ci-dessus une fois pour chaque segment. L'algorithme spécifié dans l'article 4 doit être appliqué au premier segment pour produire:

$Z_1 = Z(J,K,SEG_1).$

Z_1 doit être concaténé avec le second segment pour produire $[Z_1 \| SEG_2]$ auquel l'algorithme doit être appliqué:

$Z_2 = Z(J,K,[Z_1 \| SEG_2]).$

Noter que Z_1 est traité comme un bloc de message qui est préfixé à SEG_2 pour former un segment pouvant aller jusqu'à 257 blocs.

S'il n'y a plus de segments, Z_2 sera le MAC résultant pour le message complet. Si ce n'est pas le cas, la procédure devra continuer, et pour le $l^{ème}$ segment:

$$Z_i = Z(J,K,[Z_{i-1}||SEG_i]).$$

Le message étant composé de s segments, Z_s sera le MAC résultant pour tout le message.

NOTE 5 Il est nécessaire de n'effectuer le prélude qu'une seule fois et ses résultats (ligne 20) peuvent être mémorisés pour être utilisés sur chaque calcul Z_i. La boucle principale est réalisée une fois pour chaque bloc de message, y compris les blocs Z_i, préfixés. La coda est réalisée à la fin de chaque segment puisqu'elle fait partie de l'algorithme spécifié dans l'article 4.

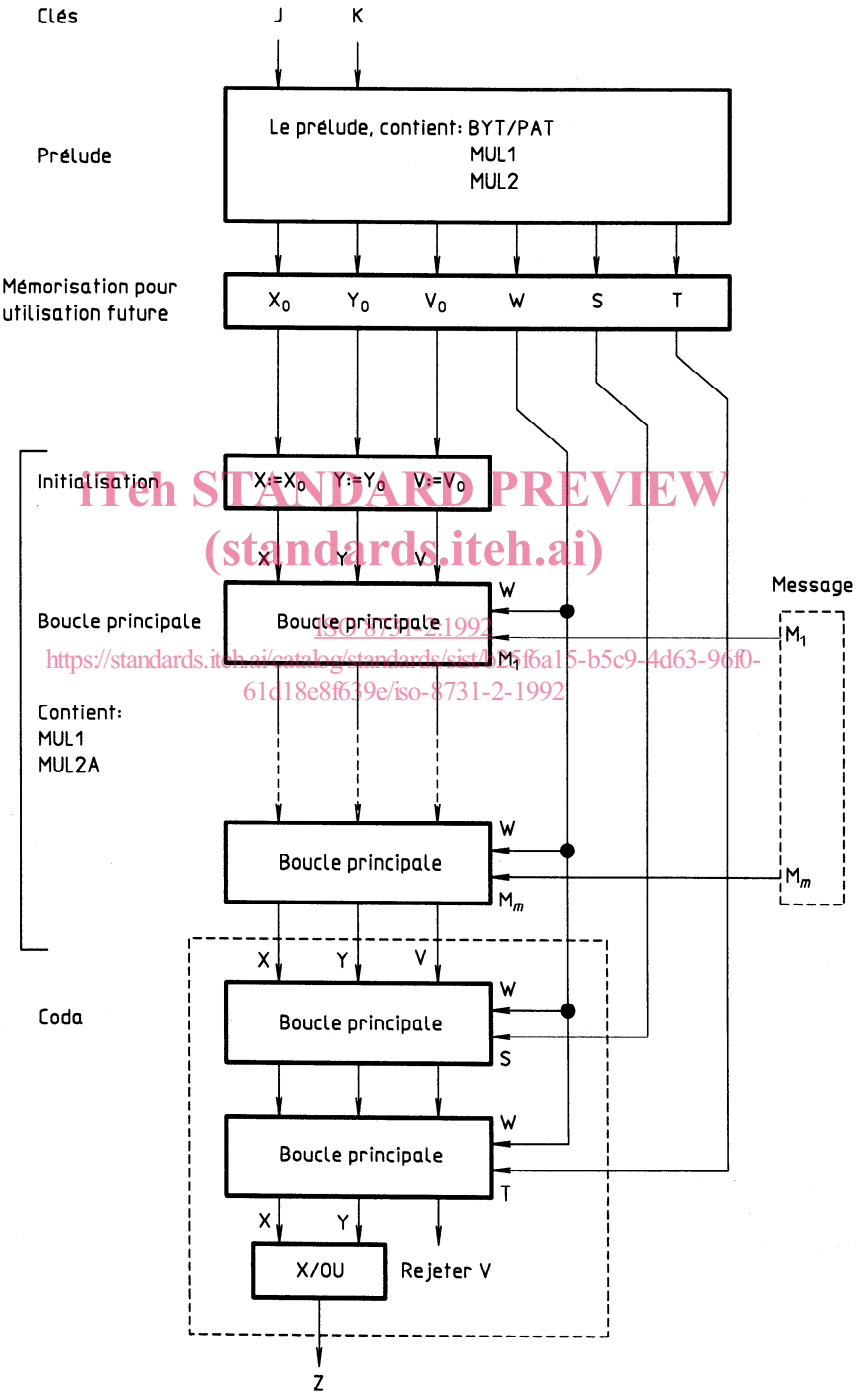

Figure 1 — Schéma montrant le flux de données pour l'algorithme de segment de *m* blocs de messages

Annexe A
(informative)

Exemples d'essais pour la mise en œuvre de l'algorithme

A.1 Généralités

Pour la plupart des parties de l'algorithme, des exemples d'essais simples sont fournis. Les données utilisées ne sont pas toujours réalistes, c'est-à-dire que ce ne sont pas des valeurs qui pourraient résulter des parties antérieures de l'algorithme, et des valeurs artificielles de constantes sont utilisées. Ceci est fait afin que les cas d'essais présentés restent suffisamment simples pour pouvoir être vérifiés par des calculs effectués manuellement et, en conséquence, pour que la vérification des mises en œuvre de l'algorithme ne consiste pas à comparer une mise en œuvre par machine avec une autre. En conséquence, les parties qui font l'objet d'essais sont:

— MUL1, MUL2, MUL2A;

— BYT[X,Y] et PAT[X,Y];

— le prélude, excepté l'opération initiale BYT[J,K];

— la boucle principale.

La «coda» ne fait pas l'objet d'essai séparé car elle utilise uniquement la boucle principale et une fonction X/OU. Pour procéder à l'essai de l'algorithme dans son entier, des résultats d'une expérience de mise en œuvre sont donnés.

A.2 Exemples d'essai de MUL1, MUL2, MUL2A

Il est proposé que les multiplications soient essayées avec des nombres très petits et des nombres très grands. Pour représenter un grand nombre, ces exemples utilisent le complément à un. Donc, si a est un petit nombre (disons inférieur à 4096), on utilise la notation $\bar{a}$ pour signifier son complément, c'est-à-dire $2^{32} - 1 - a$.

Pour les petits nombres a et b, les trois fonctions de multiplication donnent leur produit réel a^*b. Lorsque des grands nombres sont utilisés, les fonctions peuvent donner des résultats différents. Elles peuvent être essayées des deux manières à tour de rôle, avec MUL(x,y) et MUL(y,x) afin de vérifier que ces dernières donnent le même résultat.

A.2.1 Cas d'essai de MUL1

En arithmétique modulo $(2^{32} - 1)$, $\bar{a}$ est effectivement $-a$, donc les résultats sont très simples.

$\text{MUL1}(\bar{a},b) = \text{MUL1}(a,\bar{b}) = \overline{a^*b}$

$\text{MUL1}(\bar{a},\bar{b}) = a^*b$

Des exemples d'essais sont donnés dans le tableau A.1.

A.2.2 Cas d'essai de MUL2

$\text{MUL2}(\bar{a},b) = \overline{a^*b - b + 1}$

$\text{MUL2}(a,\bar{b}) = \overline{a^*b - a + 1}$

$\text{MUL2}(\bar{a},\bar{b}) = a^*b - a - b + 1$

Des exemples d'essais sont donnés dans le tableau A.1.

A.2.3 Cas d'essai de MUL2A

Cela donnera le même résultat que MUL2 si elle est essayée avec des nombres pris dans la même série. Pour des essais avec des grands nombres, a et $b - 2^{31}$ doivent être utilisés.

$$\mathrm{MUL2A}(\bar{a},b) = \overline{a^{*}b - b + 1}$$

$$\mathrm{MUL2A}(a,\bar{b}) = \overline{a^{*}b - a + 1}$$

$$\mathrm{MUL2A}(\bar{a},\bar{b} - 2^{31}) = 2^{31*}(1 - p) + a^{*}b + p - b - 1$$

où p est la parité de a, la valeur de son bit le moins significatif.

Cela donne, pour les valeurs paires de a le résultat suivant: $2^{31} + a^{*}b - b - 1$, et pour les valeurs impaires de a: $a^{*}b - b$.

Des exemples d'essais sont donnés dans le tableau A.1.

Tableau A.1 — Cas d'essais des fonctions de multiplication (hexadécimal)

Fonction	a	b	Résultat
MUL1	0000 000F FFFF FFF0 FFFF FFF0	0000 000E 0000 000E FFFF FFF1	0000 00D2 FFFF FF2D 0000 00D2
MUL2	0000 000F FFFF FFF0 FFFF FFF0	0000 000E 0000 000E FFFF FFF1	0000 00D2 FFFF FF3A 0000 00B6
MUL2A	0000 000F FFFF FFF0 7FFF FFF0 FFFF FFF0	0000 000E 0000 000E FFFF FFF1 7FFF FFF1	0000 00D2 FFFF FF3A 8000 00C2 0000 00C4

A.3 Exemples d'essais de BYT et PAT

Le tableau A.2 présente trois cas d'essais de ces fonctions.

Tableau A.2 — Cas d'essais des fonctions BYT et PAT

Fonction	X	Y
[X‖Y] BYT[X‖Y] PAT[X‖Y]	00 00 00 00 01 03 07 0F FF	00 00 00 00 1F 3F 7F FF
[X‖Y] BYT[X‖Y] PAT[X‖Y]	FF FF 00 FF FE FC 07 F0 FF	FF FF FF FF E0 C0 80 00
[X‖Y] BYT[X‖Y] PAT[X‖Y]	AB 00 FF CD AB 01 FC CD 6A	FF EF 00 01 F2 EF 35 01