68

NORME INTERNATIONALE

ISO 8731-2

Première édition
1987-12-15

Corrigée et réimprimée
1988-03-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ

Banque — Algorithmes approuvés pour l'authentification des messages —

Partie 2 :
Algorithme d'authentification des messages

Banking — Approved algorithm for message authentication —

Part 2 : Message authenticator algorithms

Numéro de référence
ISO 8731-2 : 1987 (F)

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 normalement 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.

Les projets de Normes internationales adoptés par les comités techniques sont soumis aux comités membres pour approbation, avant leur acceptation comme Normes internationales par le Conseil de l'ISO. Les Normes internationales sont approuvées conformément aux procédures de l'ISO qui requièrent 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*.

L'attention des utilisateurs est attirée sur le fait que toutes les Normes internationales sont de temps en temps soumises à révision et que toute référence faite à une autre Norme internationale dans le présent document implique qu'il s'agit, sauf indication contraire, de la dernière édition.

Imprimé en Suisse

Sommaire

Banque — Algorithmes approuvés pour l'authentification des messages —

Partie 2 : Algorithme d'authentification des messages

1 Objet et 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'authentifications contenues 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).

L'algorithme d'authentification des messages (MAA) a été spécialement conçu pour réaliser une authentification très rapide des messages grâce à l'utilisation d'un ordinateur.

Cet algorithme doit être utilisé lorsqu'il y a des volumes importants de données et lorsque la mise en œuvre efficace par logiciel est souhaitée. Le MAA est aussi adapté pour être utilisé avec une calculatrice programmable.

Les exemples d'essais sont donnés en annexe, qui ne fait pas partie intégrante de la présente Norme internationale.

2 Référence

ISO 8730, *Opérations bancaires — Spécifications liées à la normalisation de l'authentification des messages.*

3 Brève description

3.1 Généralités

L'algorithme authentificateur de message (MAA) repose sur le principe du code d'Authentification de Message (MAC), c'est-à-dire un nombre envoyé avec le message, afin 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 seront considérés comme des entiers non signés de 32 bits à moins qu'il n'en soit explicitement défini autrement. Pour un nombre N, $0 \leqslant N < 2^{32}$. Cet algorithme peut être implémenté 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 seront 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 jusqu'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 un message comportant plus de 1 000 000 blocs, c'est-à-dire $n \leqslant 1\,000\,000$.

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

Le résultat de l'algorithme est un authentificateur de 32 bits dont la valeur est appelée Z. Le calcul peut être effectué sur des messages ne comportant qu'un seul bloc ($n = 1$).

Le calcul est effectué en 3 parties :

a) le «calcul préalable» doit être un calcul effectué avec les seules clés (J et K) et il en résulte 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 effectué pour chaque bloc de message M_i 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'authentificateur.

Le mode d'opération (voir chapitre 5) est un élément essentiel de l'application de cet algorithme.

La figure montre le flux de données de façon schématique.

4 L'algorithme

4.1 Définition des fonctions utilisées dans l'algorithme

4.1.1 Généralités

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 d'Y sans tenir compte de toute 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 valeur 0 ou 1

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

4.1.2 Définition des fonctions de multiplication

Pour expliquer ces multiplications, prenons le produit sur 64 bits de X et de Y soit [U,L]. Les crochets signifient que les valeurs figurant entre ces crochets sont «concaténées», U à la gauche de L. En conséquence, U est la partie supérieure (poids forts) du produit et L la partie inférieure.

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)

U est donc 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 n'est pas utilisée dans la boucle principale, mais seulement dans le calcul préalable. D, E, F, S et C sont ici des variables locales.

D := ADD(U,U); ... (4)

E := RET(U,U); ... (5)

F := ADD(D,2E); ... (6)

S := ADD(F,L); ... (7)

C := RET(F,L); ... (8)

MUL2(X,Y) := ADD(S,2C). ... (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 et 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); ... (10)

S := ADD(D,L); ... (11)

C := RET(D,L); ... (12)

MUL2A(X,Y) := ADD(S,2C). ... (13)

Le résultat est congru à X*Y modulo ($2^{32} - 2$) sous 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 calcul préalable pour conditionner à la fois les clés et les résultats de façon à éviter les longues chaînes de «un» ou de «zéro». Elle donne deux résultats qui sont les valeurs conditionnées X et Y et un nombre PAT[X,Y] qui enregistre les modifications qui ont été effectuées. PAT[X,Y] $\leqslant$ 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 la mieux décrite par un programme où chaque octet B_i est considéré comme étant un nombre entier d'une longueur de 8 bits.

```
begin
P := 0;
for i := 0 to 7 do
begin
   P := 2*P;
   Si 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 — 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 Calcul préalable

$$[J_1,K_1] := BYT[J,K];$$
$$P := PAT[J,K];$$
$$Q := (1 + P) * (1 + P). \quad \ldots (14)$$

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

$$J1_2 := MUL1(J_1,J_1); \qquad J2_2 := MUL2(J_1,J_1);$$
$$J1_4 := MUL1(J1_2,J1_2); \qquad J2_4 := MUL2(J2_2,J2_2);$$
$$J1_6 := MUL1(J1_2,J1_4); \qquad J2_6 := MUL2(J2_2,J2_4);$$
$$J1_8 := MUL1(J1_2,J1_6); \qquad J2_8 := MUL2(J2_2,J2_6). \quad \ldots (15)$$

$$H_4 := X/OU(J1_4,J2_4);$$
$$H_6 := X/OU(J1_6,J2_6);$$
$$H_8 := X/OU(J1_8,J2_8). \quad \ldots (16)$$

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

$$K1_2 := MUL1(K_1,K_1); \qquad K2_2 := MUL2(K_1,K_1);$$
$$K1_4 := MUL1(K1_2,K1_2); \qquad K2_4 := MUL2(K2_2,K2_2);$$
$$K1_5 := MUL1(K_1,K1_4); \qquad K2_5 := MUL2(K_1,K2_4);$$
$$K1_7 := MUL1(K1_2,K1_5); \qquad K2_7 := MUL2(K2_2,K2_5);$$
$$K1_9 := MUL1(K1_2,K1_7); \qquad K2_9 := MUL2(K2_2,K2_7). \quad \ldots (17)$$

$$H' := X/OU(K1_5,K2_5);$$
$$H_5 := MUL2(H',Q);$$
$$H_7 := X/OU(K1_7,K2_7);$$
$$H_9 := X/OU(K1_9,K2_9). \quad \ldots (19)$$

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

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

4.2.2 La boucle principale

Cette boucle doit être réalisée à tour de rôle pour chacun des blocs de message M_i. En plus de M_i, les valeurs principales employées sont 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 calcul. Notons que X, Y et V sont initialisés avec les valeurs fournies par le calcul préalable. De façon à 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. La «coda», utilisée après que tous les blocs de message aient été traités par n cycles de la boucle, est décrit dans la section suivante.

NOTE — 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 sur chaque ligne.

$$V := CYC(V);$$
$$E := X/OU(V,W); \quad \ldots (21)$$
$$X := X/OU(X,M_i); \qquad Y := X/OU(Y,M_i); \quad \ldots (22)$$
$$F := ADD(E,Y); \qquad G := ADD(E,X);$$
$$F := OU(F,A); \qquad G := OU(G,B);$$
$$F := ET(F,C); \qquad G := ET(G,D); \quad \ldots (23)$$
$$X := MUL1(X,F); \qquad Y := MUL2A(Y,G). \quad \ldots (24)$$

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 — 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 ici des variables locales. Compte tenu que la constante D a zéro pour chiffre le plus significatif, $G < 2^{31}$, cela assure que MUL2A à la ligne (24) donnera le bon résultat.

4.2.3 La coda

Après que le dernier bloc de message M_n ait été traité, la boucle principale doit être réalisée avec le «bloc de message» S puis avec le bloc T à nouveau, c'est-à-dire $M_{n+1} = S$, $M_{n+2} = T$.

Après cela le code d'authentification de message (MAC) doit être calculé par $Z = X/OU(X,Y)$ et l'algorithme doit être alors terminé.

NOTE — Les valeurs X_0, Y_0, V_0, W, S et T devront être mémorisées de façon à calculer des valeurs ultérieures de l'authentificateur sans avoir à répéter le calcul préalable (calcul de la clé) tant que les clés n'auront pas été changées.

5 Spécification du mode d'opération

Les messages excédant 1 024 octets doivent être divisés en blocs de 1 024 octets et chaîner comme suit.

Pour le premier bloc de 1 024 octets le MAC (4 octets) doit être créé. La valeur du MAC doit être placée avant le second bloc (mais non transmise avec lui) et les 1 028 octets résultants doivent être authentifiés. Cette procédure doit se poursuivre, le MAC de chaque bloc étant placé avant le suivant, et ce jusqu'au dernier bloc qui n'est pas tenu d'atteindre les 1 024 octets et le MAC final doit être utilisé en tant que MAC du message complet.

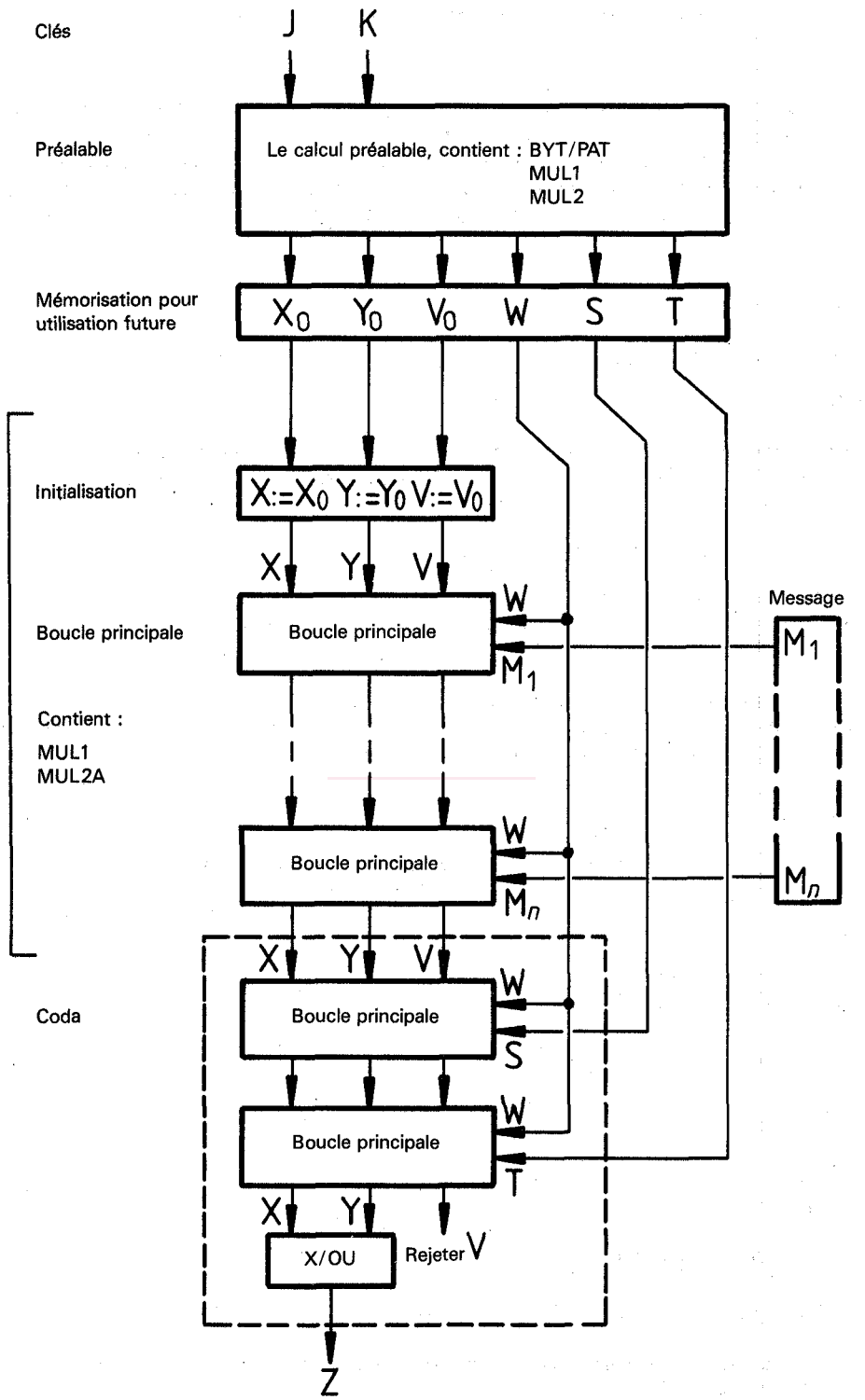

Figure — Schéma montrant le flux de données

Annexe

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

(Cette annexe ne fait pas partie intégrante de la norme.)

Pour la plupart des parties de l'algorithme, des exemples d'essais simples sont donnés. 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; 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, 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];
- Calcul préalable 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.1 Exemples d'essais de MUL1, MUL2 et MUL2A

Nous proposons que les multiplications soient essayées avec des nombres très petits et des nombres très grands. Pour représenter un grand nombre, on utilise le complément à un. Donc, si a est un petit nombre (disons inférieur à 4 096), on utilise la notation $\overline{a}$ 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.1.1 Cas d'essais de MUL1

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

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

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

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

A.1.2 Cas d'essais de MUL2

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

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

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

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

A.1.3 Cas d'essais 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, $\overline{a}$ et $\overline{b} - 2^{31}$ doivent être utilisés

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

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

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

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