

INTERNATIONAL STANDARD

ISO
8731-2

Second edition
1992-09-15

Banking — Approved algorithms for message authentication —

Part 2: Message authenticator algorithm

*Banque — Algorithmes approuvés pour l'authentification des
messages —*

Partie 2: Algorithme d'authentification des messages



Contents	Page
1 Scope	1
2 Normative references	1
3 Brief description.	1
3.1 General.	1
3.2 Technical	1
4 The segment algorithm	1
4.1 Definition of the functions used in the algorithm.	1
4.2 Specification of the algorithm	3
5 Specification of the mode of operation	3
Annexes	
A Test examples for implementation of the algorithm.	5
B Specification of MAA in VDM.	9

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

International Standard ISO 8731-2 was prepared by Technical Committee ISO/TC 68, *Banking and related financial services*, Sub-Committee SC 2, *Operations and procedures*.

This second edition cancels and replaces the first edition (ISO 8731-2:1987), of which it constitutes a technical revision.

ISO 8731 consists of the following parts, under the general title *Banking — Approved algorithms for message authentication*:

- *Part 1: DEA*
- *Part 2: Message authenticator algorithm*

Annexes A and B of this part of ISO 8731 are for information only.

This page intentionally left blank

Banking -- Approved algorithms for message authentication --

Part 2 : Message authenticator algorithm

1 Scope

ISO 8731 specifies, in individual parts, approved authentication algorithms i.e. approved as meeting the authentication requirements specified in ISO 8730. This part of ISO 8731 deals with the Message Authenticator Algorithm for use in the calculation of the Message Authentication Code (MAC).

The Message Authenticator Algorithm (MAA) is specifically designed for high-speed authentication using a mainframe computer. This is a special purpose algorithm to be used where data volumes are high, and efficient implementation by software a desirable characteristic. MAA is also suitable for use with a programmable calculator.

Test examples are given in annex A, which does not form part of this part of ISO 8731. A further test example is given as an Annex in ISO 8730.

A specification of MAA in VDM is given in Annex B, which does not form part of this part of ISO 8731.

2 Normative references

The following standards contain provisions which, through references in this text, constitute provisions of this part of ISO 8731. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO 8731 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 7185 : 1990, *Information technology - Programming languages - PASCAL*.

ISO 8730 : 1990, *Banking - Requirements for message authentication (wholesale)*.

3 Brief description

3.1 General

The Message Authenticator Algorithm works on the principle of a Message Authentication Code (or MAC), a number sent with a message, so that a check can be made by the receiver of the message that it has not been altered since it left the sender.

3.2 Technical

All numbers manipulated in this algorithm shall be regarded as 32-bit unsigned integers, unless otherwise stated. For such a number N , $0 < N < 2^{32}$. This algorithm can be implemented conveniently and efficiently in a computer with a word length of 32 bits or more.

Messages to be authenticated may originate as a bit string of any length. They shall be input to the algorithm as a sequence of 32 bit numbers, $M_1, M_2 \dots M_n$, of which there are n , called message blocks. The detail of how to pad out the last block M_n to 32 bits is not part of the algorithm but shall be defined in any application. This algorithm shall not be used to authenticate messages with more than 1 000 000 blocks, i.e. $n < 1\,000\,000$.

The key shall comprise two 32 bit numbers J and K and thus has a size of 64 bits.

The result of the algorithm is a 32 bit authentication value. The calculation can be performed on messages as short as one block ($n = 1$).

Messages longer than 256 message blocks shall be divided into segments of 256 blocks, except that the last segment may have less than 256 message blocks.

Clause 4 specifies the segment algorithm. If the whole message is within one segment this completes the calculation and its output (Z) is the value of the authenticator. If there are more than 256 message blocks, the mode of operation specified in clause 5 shall be used.

The segment algorithm has three parts.

- The prelude shall be a calculation made with the key parts (J and K) alone and it shall generate six numbers X_0, Y_0, V_0, W, S and T which shall be used in the subsequent calculations. This part need not be repeated until a new key is installed.
- The main loop is a calculation which shall be repeated for each message block M , and therefore, for long messages, dominates the calculation.
- The coda shall consist of two operations of the main loop, using as its message blocks the two numbers S and T in turn, followed by a simple calculation of Z .

The mode of operation (see clause 5) is an essential feature of the implementation of this algorithm.

Figure 1 shows the data flow in schematic form.

4 The segment algorithm

4.1 Definition of the functions used in the algorithm

4.1.1 General definitions

A number of functions are used in the description of the algorithm. In the following, X and Y are 32 bit numbers and the result is a 32 bit number except where stated otherwise.

CYC(X)	is the result of a one-bit cyclic left shift of X.
AND(X,Y)	is the result of the logical AND operation carried out on each of 32 bits.
OR(X,Y)	is the result of the logical OR operation carried out on each of 32 bits.
XOR(X,Y)	is the result of the XOR operation (modulo 2 addition) carried out on each of 32 bits.
ADD(X,Y)	is the result of adding X and Y discarding any carry from the 32nd bit, that is to say, addition modulo 2^{32} .
CAR(X,Y)	is the value of the carry from the 32nd bit when X is added to Y; it has the value of 0 or 1.
MUL1(X,Y), MUL2(X,Y) and MUL2A(X,Y)	are three different forms of multiplication, each with a 32 bit result.
[X Y]	is the result of concatenating the binary numbers X and Y, in the left of most significant position. The notation is extended to concatenate more than two numbers and is applied also to 8 bit bytes and numbers longer than 32 bits.

4.1.2 Definition of multiplication functions

To explain the multiplications, let the 64 bit product of X and Y be [U||L]. Hence U is the upper (most significant) half of the product and L the lower (least significant) half.

4.1.2.1 To calculate MUL1(X,Y)

Multiply X and Y to produce [U||L] with S and C as local variables,

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

C := CAR(U,L); ... (2)

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

That is to say, U shall be added to L with end around carry.

Numerically the result is congruent to $X*Y$, the product of X and Y, modulo $(2^{32} - 1)$. It is not necessarily the smallest residue because it may equal $2^{32} - 1$.

4.1.2.2 To calculate MUL2(X,Y)

This form of multiplication shall not be used in the main loop, only in the prelude. With D, E, F, S and C as local variables,

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

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

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

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

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

MUL2(X,Y) := ADD(S,2C). ... (9)

Numerically the result is congruent to $X*Y$, the product of X and Y, modulo $(2^{32} - 2)$. It is not necessarily the smallest residue because it may equal $2^{32} - 1$ or $2^{32} - 2$.

4.1.2.3 To calculate MUL2A(X,Y)

This is a simplified form of MUL2(X,Y) used in the main loop, which yields the correct result only when at least one of the numbers X and Y has a zero in its most significant bit.

This form of multiplication is employed for economy in processing. D, S, C are local variables,

D := ADD(U,U); ... (10)

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

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

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

The result is congruent to $X*Y$ modulo $(2^{32} - 2)$ under the conditions stated because, in the notation of MUL2(X,Y) above, the carry E = 0.

4.1.3 Definition of the functions BYT[X||Y] and PAT[X||Y]

A procedure is used in the prelude to condition both the key parts and the results in order to prevent long strings of ones or zeros. It produces two results which are the conditioned values of X and Y and a number PAT[X,Y] which records the changes that have been made. PAT[X,Y] < 255 so it is essentially an 8 bit number.

X and Y are regarded as strings of bytes.

[X||Y] = [B₀|| B₁|| B₂|| B₃|| B₄|| B₅|| B₆|| B₇]

Thus bytes B₀ to B₃ are derived from X and B₄ to B₇ from Y.

The procedure is best described by a procedure where each byte B_i is regarded as an integer of length 8 bits.

begin

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 The procedure is written in the programming language PASCAL (see ISO 7185), except that the non-standard identifier B' has been used to maintain continuity with the text. The symbols B[i] and B'[i] correspond to B_i and B'_i in the text.

The results are

$$\text{BYT}[X||Y] = [B_0||B'_1||B_2||B'_3||B_4||B'_5||B_6||B'_7]$$

and

$$\text{PAT}[X||Y] = P$$

4.2 Specification of the algorithm

4.2.1 The prelude

$$[J_1||K_1] := \text{BYT}[J||K];$$

$$P := \text{PAT}[J||K];$$

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

First, by means of a calculation using J₁, produce H₄, H₆, and H₈ from which X₀, V₀ and S are derived.

$$J_{12} := \text{MUL1}(J_1, J_1); \quad J_{22} := \text{MUL2}(J_1, J_1);$$

$$J_{14} := \text{MUL1}(J_{12}, J_{12}); \quad J_{24} := \text{MUL2}(J_{22}, J_{22});$$

$$J_{16} := \text{MUL1}(J_{12}, J_{14}); \quad J_{26} := \text{MUL2}(J_{22}, J_{24});$$

$$J_{18} := \text{MUL1}(J_{12}, J_{16}); \quad J_{28} := \text{MUL2}(J_{22}, J_{26}). \quad \dots (15)$$

$$H_4 := \text{XOR}(J_{14}, J_{24});$$

$$H_6 := \text{XOR}(J_{16}, J_{26});$$

$$H_8 := \text{XOR}(J_{18}, J_{28}). \quad \dots (16)$$

From a similar calculation using K₁, produce H₅, H₇ and H₉, from which Y₀, W and T are derived.

$$K_{12} := \text{MUL1}(K_1, K_1); \quad K_{22} := \text{MUL2}(K_1, K_1);$$

$$K_{14} := \text{MUL1}(K_{12}, K_{12}); \quad K_{24} := \text{MUL2}(K_{22}, K_{22});$$

$$K_{15} := \text{MUL1}(K_1, K_{14}); \quad K_{25} := \text{MUL2}(K_1, K_{24});$$

$$K_{17} := \text{MUL1}(K_{12}, K_{15}); \quad K_{27} := \text{MUL2}(K_{22}, K_{25});$$

$$K_{19} := \text{MUL1}(K_{12}, K_{17}); \quad K_{29} := \text{MUL2}(K_{22}, K_{27}). \quad \dots (17)$$

$$H' := \text{XOR}(K_{15}, K_{25});$$

$$H_5 := \text{MUL2}(H', Q);$$

$$H_7 := \text{XOR}(K_{17}, K_{27});$$

$$H_9 := \text{XOR}(K_{19}, K_{29}). \quad \dots (19)$$

Finally, condition the results using the BYT function

$$[X_0||Y_0] := \text{BYT}[H_4||H_5];$$

$$[V_0||W] := \text{BYT}[H_6||H_7];$$

$$[S||T] := \text{BYT}[H_8||H_9]. \quad \dots (20)$$

4.2.2 The main loop

This loop shall be performed in turn for each of the message blocks M_i. In addition to M_i, the principal values employed shall be X and Y and the main results shall be the new values of X and Y. It shall also use V and W and modify V at each performance. X, Y and V shall be initialized with the values provided by the prelude. In order to use the same keys again, the initial values of X, Y and V shall be preserved, therefore they shall be denoted X₀, Y₀ and V₀ and there shall be an initializing step X := X₀, Y := Y₀, V := V₀, after which the main loop shall be entered for the first time.

NOTE 2 The program is shown in columns to clarify its parallel operation but it should be read in normal reading order, left to right on each line.

$$V := \text{CYC}(V);$$

$$E := \text{XOR}(V, W); \quad \dots (21)$$

$$X := \text{XOR}(X, M_i); \quad Y := \text{XOR}(Y, M_i); \quad \dots (22)$$

$$F := \text{ADD}(E, Y); \quad G := \text{ADD}(E, X);$$

$$F := \text{OR}(F, A); \quad G := \text{OR}(G, B);$$

$$F := \text{AND}(F, C); \quad G := \text{AND}(G, D); \quad \dots (23)$$

$$X := \text{MUL1}(X, F); \quad Y := \text{MUL2A}(Y, G). \quad \dots (24)$$

The numbers A, B, C, D are constants which are, in hexadecimal notation:

Constant A: 0204 0801

Constant B: 0080 4021

Constant C: BFEF 7FDF

Constant D: 7DFE FBFF

NOTE 3 Lines (21) are common to both paths. Line (22) introduces the message block M_i. Lines (23) prepare the multipliers and line (24) generates new X and Y values. Only X, Y and V are modified for use in the next cycle. F and G are local variables. Since the constant D has its most significant digit zero, G < 2³¹ and this ensures that MUL2A in line (24) will give the correct result.

4.2.3 The coda

The coda shall be performed after the last message block of the segment has been processed, by applying the main loop to message block S, then again to message block T. Then the result Z = XOR(X, Y) shall be calculated. This completes the coda. If the message contains no more than 256 message blocks, Z is the value of the MAC. Otherwise the value of Z shall be used in the mode of operation specified in clause 5.

NOTE 4 In order to calculate further Z values without repeating the prelude (key calculation) until the key is changed the values X₀, Y₀, V₀, W, S and T should be retained.

5 Specification of the mode of operation

Messages longer than 256 message blocks shall be divided into segments SEG₁, SEG₂...SEG₅ each of 256 blocks except that the last segment may have from 1 to 256 blocks. The number of segments is s.

The result Z of the segment algorithm specified in clause 4, when applied to key J, K and a message M shall be denoted Z(J, K, M).

The mode of operation for calculating the MAC for a message of more than 256 blocks shall employ the above algorithm once for each segment. The algorithm specified in clause 4 shall be applied to the first segment to produce:

$$Z_1 = Z(J, K, \text{SEG}_1).$$

Z_1 shall be concatenated with the second segment to produce $[Z_1 || \text{SEG}_2]$, to which the algorithm shall be applied:

$$Z_2 = Z(J, K, [Z_1 || \text{SEG}_2]).$$

Note that Z_1 is treated as a message block which is prefixed to SEG_2 to form a segment of up to 257 blocks.

If there are no more segments, Z_2 shall be the resultant MAC for the whole message, otherwise the procedure shall continue, and for the i th segment:

$$Z_i = Z(J, K, [Z_{i-1} || \text{SEG}_i]).$$

There are in total s segments; then Z_s shall be the resultant MAC for the whole message.

NOTE 5 The prelude need be performed only once and its results (line 20) may be retained for use on each Z_i calculation. The main loop is performed once for each message block, including the prefixed Z_i blocks. The coda is performed at the end of each segment, since it is part of the segment algorithm specified in clause 4.

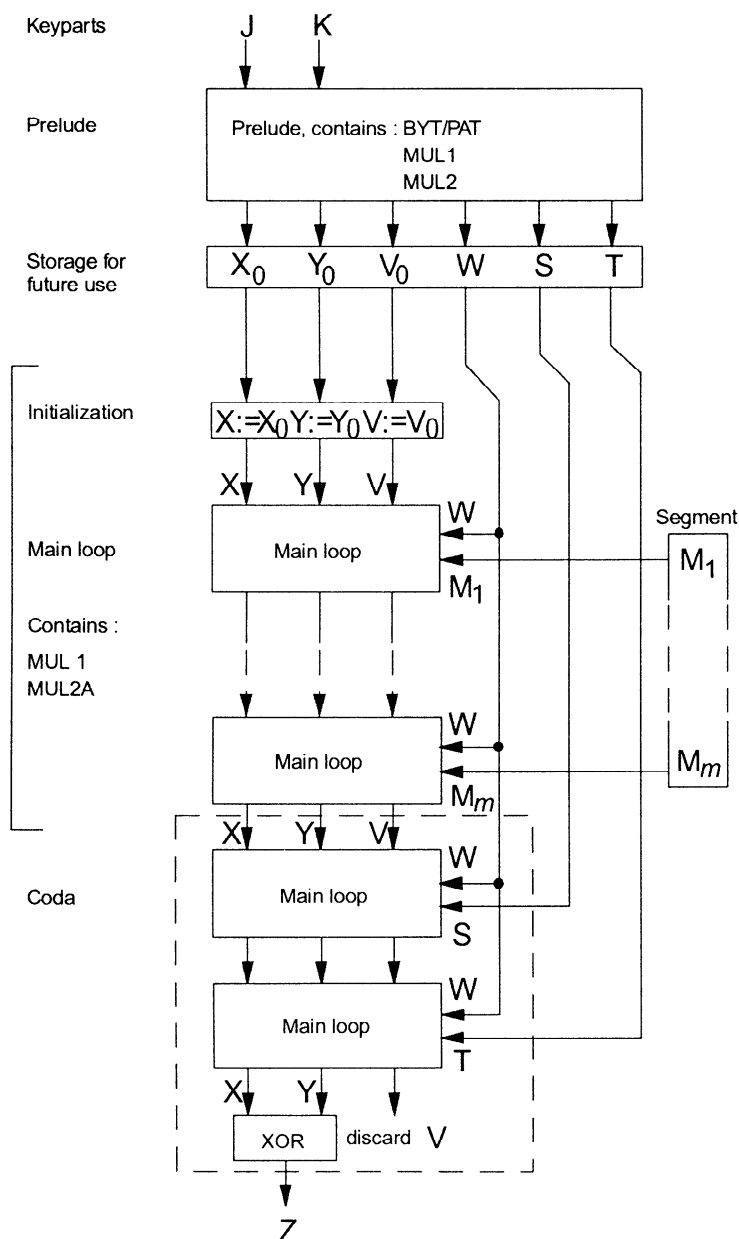


Figure 1 - Schematic showing data flow for the segment algorithm applied to a segment of m message blocks