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

Reference number
ISO 8731-2:1992(E)

Contents

Annexes

International Organization for Standardization
Case Postale 56 • CH-1211 Genève 20 • Switzerland
Printed in Switzerland

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 -- 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.

a) 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.

b) The main loop is a calculation which shall be repeated for each message block M, and therefore, for long messages, dominates the calculation.

c) 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.

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

Thus bytes B_0 to B_3 are derived from X and B_4 to B_7 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

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

and

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

4.2 Specification of the algorithm

4.2.1 The prelude

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

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

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

First, by means of a calculation using J_1, produce H_4, H_6, and H_8 from which X_0, V_0 and S are derived.

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

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

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

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

$H_4 := XOR(J1_4,J2_4);$

$H_6 := XOR(J1_6,J2_6);$

$H_8 := XOR(J1_8,J2_8).$ ... (16)

From a similar calculation using K_1, produce H_5, H_7 and H_9, from which Y_0, W and T are derived.

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

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

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

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

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

$H' := XOR(K1_5,K2_5);$

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

$H_7 := XOR(K1_7,K2_7);$

$H_9 := XOR(K1_9,K2_9).$ ...(19)

Finally, condition the results using the BYT function

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

4.2.2 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 X_0, Y_0 and V_0 and there shall be an initializing step $X := X_0$, $Y := Y_0$, $V := V_0$, 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.

$V := CYC(V);$

$E := XOR(V,W);$ ...(21)

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

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

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

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

$X := MUL1(X,F);$ $Y := MUL2A(Y,G).$ ...(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, $G < 2^{31}$ 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 X_0, Y_0, V_0, 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 $SEG_1,SEG_2...SEG_5$ 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:

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

Z_1 shall be concatenated with the second segment to produce $[Z_1||SEG_2]$, to which the algorithm shall be applied:

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

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 ith segment:

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

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.

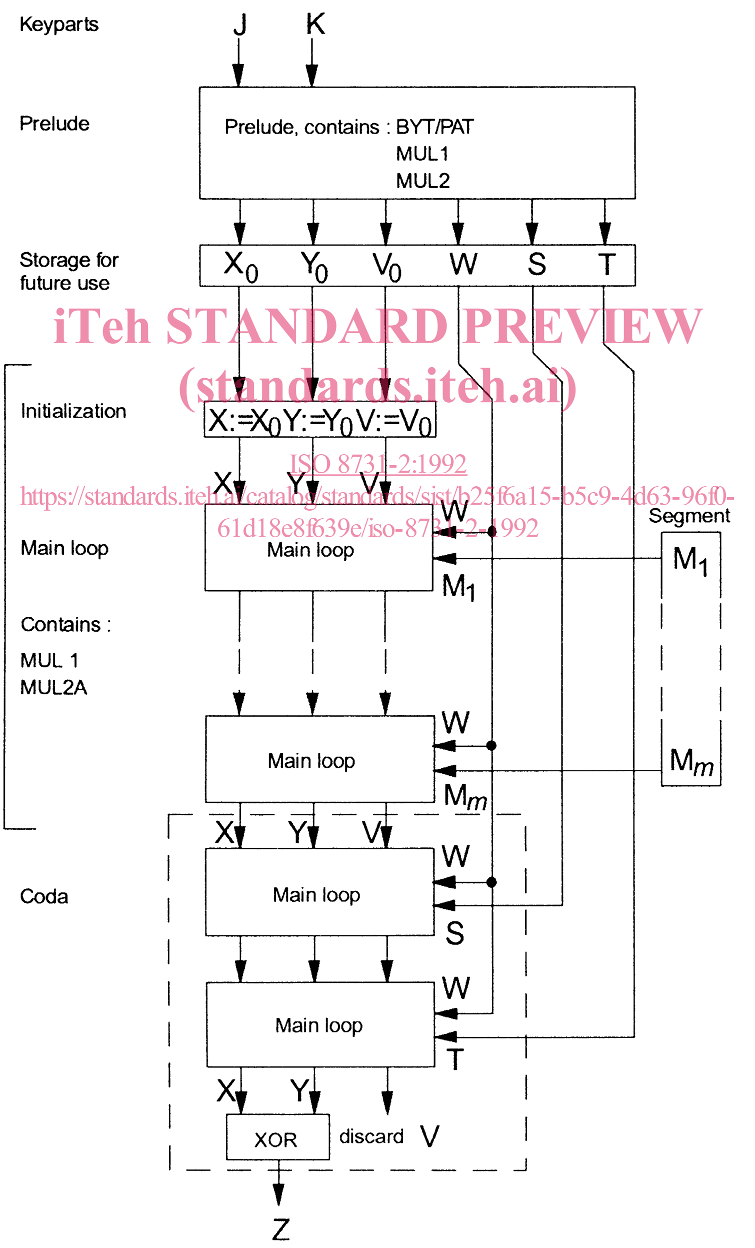

Figure 1 - Schematic showing data flow for the segment algorithm applied to a segment of m message blocks

Annex A

(informative)

Test examples for implementation of the algorithm

A.1 General

For most parts of the algorithm, simple test examples are given. The data used are not always realistic, i.e. they are not values which could be produced by earlier parts of the algorithm, and artificial values of constants are used. This is done to keep the test cases so simple that they can be verified by a pencil and paper calculation and thus the verification of the algorithm's implementations does not consist of comparing one machine implementation with another. The parts thus tested are:

- MUL1, MUL2, MUL2A;
- BYT[X,Y] and PAT[X,Y];
- Prelude, except the initial BYT[J,K] operation;
- Main loop.

The coda is not tested separately because it uses only the main loop and one XOR function. For testing the whole algorithm, some results from a trial implementation are given.

A.2 Test examples for MUL1, MUL2, MUL2A

It is suggested that the multiplication operations should be tested with very small numbers and very large numbers. To represent a large number these examples use the ones complement. Thus if a is a small number (say less than 4 096) the notation $\overline{a}$ is used to mean its complement, i.e. $2^{32} - 1 - a$.

For small numbers a and b, all three multiplication functions produce their true product $a*b$. When large numbers are used the functions can give different results. They should be tested both ways round, with MUL(x,y) and MUL(y,x) to verify that these are equal.

A.2.1 Test cases for MUL1

In modulo $(2^{32} - 1)$ arithmetic $\overline{a}$ is effectively $-a$, therefore the results are very simple

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

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

Examples for testing are given in table 1.

A.2.2 Test cases for MUL2

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

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

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

Examples for testing are given in table 1.

A.2.3 Test cases for MUL2A

This will give the same result as MUL2 when tested with numbers within its range. For testing with large numbers, $\overline{a}$ and $\overline{b} - 2^{31}$ shall be used

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

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

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

where p is the parity of a, the value of its least significant bit.

That is, for even values of a the result is $2^{31} + a*b - b - 1$ and for odd values of a the result is $a*b - b$.

Examples for testing are given in table A.1.

Table A.1 - Test cases for multiplication functions (hexadecimal)

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

A.3 Test examples for BYT and PAT

Three cases for testing these functions are listed in table A.2.

Table A.2 - Test cases for the BYT and PAT functions

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

A.4 Test examples for the prelude

An example is given in table A.3. The initial BYT[J||K] operation is not tested. It is assumed that the results from lines (14) are

$$J_1 = 0000\ 0100, \qquad K_1 = 0000\ 0080, \qquad P = 1.$$

Table A.3 - Test cases for lines (15) to (20) of the prelude

$J1_2$	0001 0000	$J2_2$	0001 0000
$J1_4$	0000 0001	$J2_4$	0000 0002
$J1_6$	0001 0000	$J2_6$	0002 0000
$J1_8$	0000 0001	$J2_8$	0000 0004
H_4		0000 0003	
H_6		0003 0000	
H_8		0000 0005	
$K1_2$	0000 4000	$K2_2$	0000 4000
$K1_4$	1000 0000	$K2_4$	1000 0000
$K1_5$	•0000 0008	$K2_5$	0000 0010
$K1_7$	0002 0000	$K2_7$	0004 0000
$K1_9$	8000 0000	$K2_9$	0000 0002
H′		0000 0018	
H_5		0000 0060 (Q = 4)	
H_7		0006 0000	
H_9		8000 0002	
$[X_0\|\|Y_0]$	0103 0703 1D3B 7760	PAT$[X_0\|\|Y_0]$	EE (1110 1110)
$[V_0\|\|W]$	0103 050B 1706 5DBB	PAT$[V_0\|\|W]$	BB (1011 1011)
[S||T]	0103 0705 8039 7302	PAT[S||T]	E6 (1110 0110)

The PAT values obtained from conditioning the results of the prelude are quoted above for checking purposes but are not used in the algorithm.

A.5 Test examples for the main loop

In table A.4, three examples of single block messages are given, using small and large numbers with the convention that $\bar{a}$ *is* 2^{32} - 1 - *a*. In the third example there are two cases of large numbers which must have zero in the 32nd bit, shown as $\bar{2}$ - 2^{31} and $\bar{3}$ - 2^{31} respectively. They could have been written 2^{31} - 3 and 2^{31} - 4 respectively. In order to keep the numbers small, artificial values of the constants A, B, C and D are used. Three single block examples are followed by a message of three blocks, in order to check that the implementation correctly retains the value of X, Y and W. The final S and T cycles of the coda are not included in this table.

Table A.4 - Test cases for the main loop (decimal)

		Single block messages						Three-block message						
A	B	4	1	1	4	1	2	2	1	2	1	2	1	
C	D	$\overline{8}$	$\overline{4}$	$\overline{6}$	$\overline{3}$	$\overline{1}$	$\overline{2*}$	$\overline{4}$	$\overline{4}$	$\overline{4}$	$\overline{4}$	$\overline{4}$	$\overline{4}$	
V	W	3	3	3	3	7	7	1	1	2	1	4	1	
X_0	Y_0	2	3	$\overline{2}$	$\overline{3}$	$\overline{2}$	$\overline{3}$	1	2	3	2	20	9	
M		5		1		8		0		1		2		
V		6		6		14		2		4		8		CYC
E		5		5		9		3		5		9		XOR
X	Y	7	6	$\overline{3}$	$\overline{2}$	$\overline{10}$	$\overline{11}$	1	2	2	3	22	11	XOR
F	G	11	12	2	1	$\overline{2}$	$\overline{1}$	5	4	8	7	20	31	ADD
F	G	15	13	3	5	$\overline{2}$	$\overline{1}$	7	5	10	7	22	31	OR
F	G	7	9	1	4	$\overline{3}$	$\overline{3*}$	3	1	10	3	18	27	AND
X	Y	49	54	$\overline{3}$	$\overline{5}$	30	30	3	2	20	9	396	297	MUL
Z		7		6		0		1		29		165		XOR

* - 2^{31}