A MESSAGE AUTHENTICATOR ALGORITHM SUITABLE FOR A MAINFRAME COMPUTER

Donald Watts Davies
Independent Consultant,
Sunbury on Thames,
Middx, UK.

INTRODUCTION

Authenticators are widely used to protect payment messages against active attack. They produce a number, sometimes called a 'MAC' which is a function of the whole message and a secret key. the earlier name for them in banking was 'test-key', but this obsolescent term is confusing to cryptographers.

Several algorithms now in use, such as that of S.W.I.F.T. and the Data Seal are not revealed to the public. Authenticators based on the DEA 1 and the DSA algorithms (decimal shift and add - for decimal calculators) are public but neither is well adapted to mainframe computers.

Bankers Automated Clearing Services (BACS) suggested the need for a 'mainframe authenticator' and together, with a colleague, David Clayden, we developed this one, known in banking circles as 'MAA'.

The algorithm attracted the attention of the 'Test Key Working Party' of the CLCB (Committee of the London Clearing Banks) who arranged for independent testing of the algorithm. It is also being considered by an ISO working group.

G.R. Blakley and D. Chaum (Eds.): Advances in Cryptology - CRYPTO '84, LNCS 196, pp. 393-400, 1985.

DESCRIPTION

The definition of the algorithm is contained in an NPL Report DITC 17/83 dated February 1983 with the same title as this paper, by D. W. Davies and D. O. Clayden. All I can do here is to sketch out its structure. Serious Study requires a copy of the definition. NPL is the UK National Physical Laboratory at Teddington, Middlesex, TW1I OLW, UK.

The key has two numbers, J and K, each of 32 bits. All words used in the algorithm are 32 bits long. When a new key is installed, a key calculation called the 'Prelude' produces 6 numbers X_0, Y_0, V_0, W, S, T which are used in the rest of the algorithm. The choice of J, K is unrestricted.

Multiplication is the principal tool of this algorithm and is used in two varieties, modulo $2^{32} - 1$ and modulo $2^{32} - 2$. The prelude is mainly the following calculation:-

$$X_0 = J^4 (1) \text{ XOR } J^4 (2)$$

$$Y_0 = \left[K^5 (1) \text{ XOR } K^5 (2)\right](1 + P)^2$$

$$V_0 = J^6 (1) \text{ XOR } J^6 (2)$$

$$W = K^7 (1) \text{ XOR } K^7 (2)$$

$$S = J^8 (1) \text{ XOR } J^8 (2)$$

$$T = K^9 (1) \text{ XOR } K^9 (2)$$

Where 1 and 2 refer to the two moduli $2^{32} - 1$ and $2^{32} - 2$ respectively and XOR is bit-wise on a 32 bit word.

The eight bytes J,K are first treated by a procedure to replace any byte which is 0000,0000 or 1111,1111. A resultant number P records the changes made and its use in calculating Yo avoids reducing the key space. The pairs X_0, Y_0 ; V_0, W and S,T are similarly treated to remove runs of zeros or ones before they are used in the body of the algorithm.

The main part of the calculation (we considered calling it the Fugue) takes in the message in blocks M_i of size 4 bytes and, for each one repeats the steps in Figure 1. The variables X, Y, V are initialised to X_0, Y_0 and V_0 respectively. For each block, V is cyclic shifted left one bit and XORed with W to produce E. The + operations are modulo 2^{32}. The constants A, B, C, D are used in the logical operations to set 8 bits of each numbers (F and G) to fixed values. The aim is to avoid bytes of all zeros or all ones in the multipliers F and G, as well as to introduce non-linearity. The two multiplications with different moduli complete the round.

The authenticator value to be produced at the end of the calculation is simply Z = X XOR Y, but after the last message block has been used, the numbers S and T are used as message blocks for two rounds (as if appended to the message) before the final XOR operation. This last part, producing Z, is called the Coda.

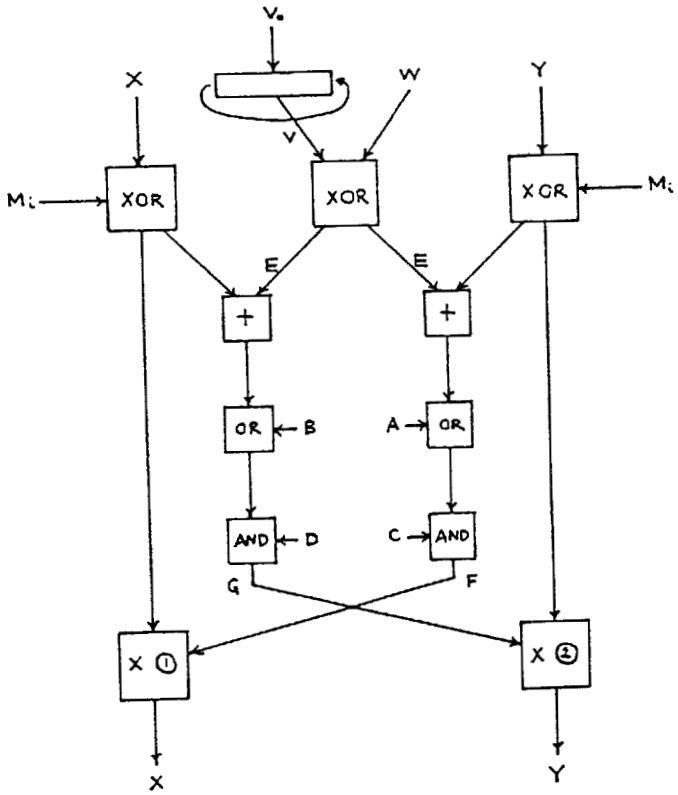

FIGURE 1

PERFORMANCE

Since this algorithm is designed for a 32 computer containing a multiplier, the performance figure, for a typical IBM configuration would be of interest. But in the time that has passed since the report was published, no such measurements of performance have been reported to us. An assembly language program for a microcomputer (2 MHz 6502 = BBC Micro) takes 47 ms for the prelude and coda and 5.92 ms for each block of message (675 byte/s or 5405 bit/s). Since this uses a programmed multiplication it is not the way that MAA was designed to be used.

TESTING

We have no positive reason for confidence in the security of the algorithm but at each stage of testing we tried all the input/output dependancies and statistical distributions we could think of. We also used a zero message and some constant messages (such as all ones) and looked for loops.

Most of the testing was done with altered versions of the algorithm deliberately weakened in someway. For example, we demanded in most cases that both the X and the Y values should show good statistical properties (and confirmed the results with Z). We reduced the number of fixed bits in A, B, C, D and removed E or S and T, though not all these at the same time. For sensitivity to key changes we varied separately the six outputs of the prelude, before testing with the prelude in place.

At several stages of development, problems were found and fixed, but we found the fixes had to be carefully thought through to avoid bringing back old problems. When all our weakened tests were passed we tested it again in its complete form.

PROBLEMS

Two problems have been pointed out. If X becomes zero and M_i remains zero then X remains zero. If you know X you could make M_i = X and engineer this zero value. If both X and Y becomes zero and M_i remains zero then X and Y remain zero. In this last case any set of consecutive zero message blocks can be inserted without changing the value of the authenticator. This is indeed a flaw but can anyone suggest how an opponent would use it, not knowing when X = Y = 0, a very rare event?

The second problem was posed by H. Block of SAK Data AB in 'File Authentication - A rule for constructing algorithms', at Eurocrypt 84. If all the M_i are fixed, each round of the main loop maps (X,Y) into (X,Y) with a mapping that is injective. For an approximation, assume that these are random mappings. Now imagine that 3-4 early blocks in a very long sequence are varied so that all 2^{64} states of X,Y are attained at some point in the sequence of rounds. With constant M_i values thereafter, each mapping reduces the number of attainable states. When it falls below about 2^{34}, there is a significant risk that values of Z will be missing from the set of 2^{32}. Eventually, the 'memory' of these early changes of M_i will be lost. Block concludes that injective functions should never be used. He thinks that the problem may be worse than we see when random mappings are assumed.

During the testing of the algorithm with 'toy' examples this effect was detected and (though with only a few cases to estimate from) its magnitude agreed with the theoretical value for random mappings, so we are content to rely on that theory. If we used the argument that 'it might be much worse' this would disqualify all but provably secure algorithms, of which there is a shortage.

ANALYSIS OF THE 'LOSS OF MEMORY' PROBLEM

Suppose that the number of states is N and that a set X_i of these is mapped by a random mapping into the same domain, giving X_{i+1} distinct states. Then approximately (Poisson distribution)

$$X_{i+1}/N = 1 - \exp(-X_i/N)$$

If the sequence is evaluated, starting at $X_o/N = 1$, it follows, to a close approximation:

$$X_i/N = 2/\{i + 1/3\ln(i) + 9/5\}$$

In the example of the MAA, $N = 2^{64}$ and $x_i = 2^{32}$ is reached approximately when $i = 2^{33}$. With, say, 10^9 blocks of data in the message (4×10^9 bytes), there should be no perceptible effect. In fact, to measure the effect would require a sample of much more than 2^{33} blocks.

We have suggested an arbitary, but very safe upper limit of 10^6 blocks for any one message. Other considerations (error control and recovery) usually set a lower limit then this.

Acknowledgement is made to the National Physical Laboratory for supporting this work and to Open Computer Security for help with the presentation of this paper at Crypto '84.