Security Analysis of the Message Authenticator Algorithm (MAA)[1]

Bart Preneel[2] Vincent Rijmen[2] Paul C. van Oorschot[3]

April 15, 1997

Abstract

The security of the ISO banking standard Message Authenticator Algorithm (ISO 8731-2), also known as MAA, is considered. The attacks presented herein, which exploit the internal structure of the algorithm, are the first computationally feasible attacks on MAA. First a MAC forgery attack is presented that requires 2^{17} messages of 256 Kbytes or 2^{24} messages of 1 Kbyte; the latter circumvents the special MAA mode for long messages defined in the standard. Next a key recovery attack on MAA is described which requires 2^{32} chosen texts consisting of a single message block. The number of off-line multiplications for this attack varies between 2^{44} for one key in 1000 to about 2^{51} for one key in 50. This should be compared to about $3 \cdot 2^{65}$ multiplications for an exhaustive key search. Finally it is shown that MAA has 2^{33} keys for which it is rather easy to create a large cluster of collisions. These keys can be detected and recovered with 2^{27} chosen texts. From these attacks follows the identification of several classes of weak keys for MAA.

1 Introduction

Message authentication code (MAC) algorithms are commonly used for data integrity and data origin authentication, e.g. in banking applications [12]. They are functions parameterized by a relatively short (e.g. 64-bit) secret key K which allow one to associate with a string x of arbitrary length a short string of fixed length denoted by $\mathrm{MAC}_K(x)$. This latter string is a complex function of its two inputs.

MACs are used in a setting where a sender and a receiver share a secret key K. In order to protect a message x, the sender Alice computes $\mathrm{MAC}_K(x)$, and appends this to the message. On receipt of x, the receiver Bob recomputes $\mathrm{MAC}_K(x)$ and verifies that it corresponds to the transmitted MAC value. If so, Bob accepts that the message came from Alice and was not modified (see also figure 1). An active eavesdropper Eve may try to fool Bob into accepting $x' \neq x$ as authentic. However, she does not know K, and if the MAC function satisfies certain security properties, she will be unable to predict $\mathrm{MAC}_K(x')$ better than by just guessing.

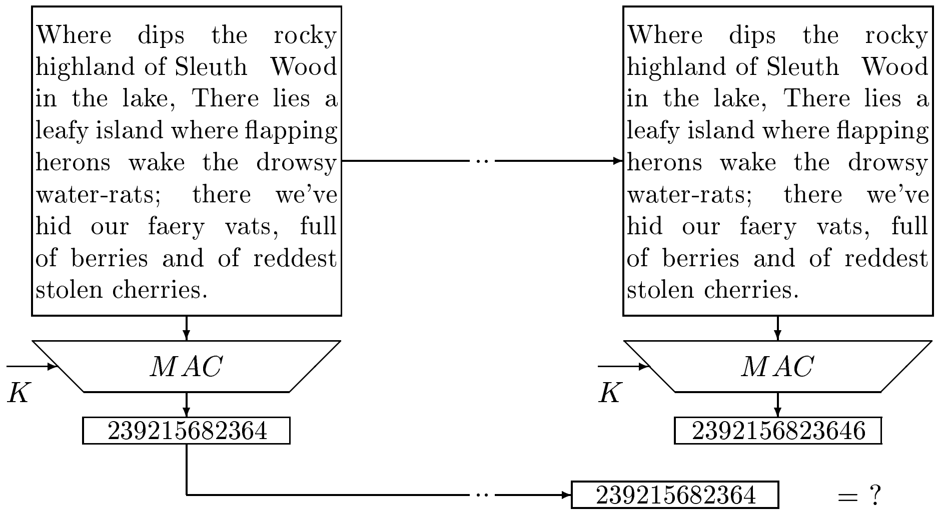
Until recently, only a few early MAC proposals were standardized and used in practice, and very little published research was available regarding their evaluation. These proposals

[1]A subset of these results was presented at Eurocrypt'96.

[2]Bart Preneel and Vincent Rijmen are with the Katholieke Universiteit Leuven, Dept. Electrical Engineering-ESAT, Kardinaal Mercierlaan 94, B–3001 Heverlee, Belgium;
email {bart.preneel},{vincent.rijmen}@esat.kuleuven.ac.be. Bart Preneel is a postdoctoral researcher and Vincent Rijmen is a research assistant; both are sponsored by the Fund for Scientific Research – Flanders (Belgium) (F.W.O.).

[3]Paul C. van Oorschot is with Entrust Technologies, P.O. Box 3511 Station C, Ottawa, Ontario K1Y 4H7, Canada; email paulv@entrust.com.

Figure 1: Using a Message Authentication Code for data integrity.



are the Cipher Block Chaining (CBC) and the Cipher FeedBack (CFB) modes of a block cipher [1, 2, 16, 17] on the one hand, and the Message Authenticator Algorithm (MAA) [10, 11, 16] on the other hand.

During the past few years, MACs have received more attention. Several papers have discussed the security of the CBC-MAC: a proof of security (a lower bound to break the system under certain specific assumptions) was given by Bellare et al. [3], and an almost matching upper bound by Preneel and van Oorschot [24]. Other recent attacks include that of Knudsen [20], which improved a forgery attack for short outputs, and that of Preneel and van Oorschot [26] in which a flaw was pointed out in the double key length variant of CBC-MAC, which is known as the ANSI retail MAC X9.19 [2, 17].

Several new practical MAC schemes have been proposed according to two approaches. Computationally secure schemes have been developed in several papers [4, 5, 6, 24]. For so-called "provably secure" schemes, major improvements have been found both in terms of speed of computation and the key size [18, 19, 21, 28]. These schemes, which follow the model of Simmons [29] and Wegman-Carter [31] still impose the restriction of one-time use of the key. They can be reduced to a potentially efficient computationally secure scheme by generating the keys using a cryptographically strong pseudo-random string generator.

While the attacks of Preneel and van Oorschot [24] apply to a large class of MAC algorithms, no cryptanalytic results were published specific for the ISO 8731-2 standard MAA[4], which dates back to 1983 [10, 11]. It is quite a remarkable algorithm: after Lucifer and DES, it is probably the first custom-designed cryptographic algorithm to be published (with the exception of stream cipher designs); it is also the first algorithm after DES to be included in a published standard; and it withstood cryptanalysis for more than twelve years.

Designed to run on mainframes in the mid 1980's, MAA is remarkably fast and efficient on present day PCs and workstations with 32-bit processors. On such machines MAA is about 40% faster than RIPEMD-160 [9] and SHA-1 [15], and about five times faster than

[4]Among several banks, a letter appears to have been circulated in the 1980's which claims the existence of a cryptanalytic attack on MAA, but no details were ever published.

DES-based MACs. The basic operations of MAA are modular multiplication, addition and exclusive or; in this respect MAA is quite similar to IDEA [22].

The current paper presents a detailed security analysis of MAA, including a forgery attack and its optimizations, and a key recovery attack. Several new classes of weak keys are presented, and it is shown that for certain keys MAA exhibits an undesirable behavior.

The remainder of this paper is organized as follows. §2 provides background definitions and reviews a generic forgery attack on MACs. §3 gives a detailed description of MAA. §4 develops various optimizations of this forgery attack based on the internal structure of MAA and defines two classes of weak keys. §5 presents the new key recovery attack and compares this to an exhaustive key search. §6 proposes two new classes of weak keys, for which it is easy to find many collisions. Each collision is produced by two messages that differ in only one block; the difference depends on the key, but it is constant for all messages of the so-called "collision cluster." For some of these key recovery is much easier. §7 concludes the paper, discussing the impact of these attacks on current applications and suggesting several improvements to the basic design.

2 Background Definitions and Review

A hash function h map bitstrings of arbitrary finite length into strings of fixed length (say m bits). A message authentication code (MAC) is a hash function with a secret key as a secondary input, that satisfies some additional security properties. These are formulated in terms of resistances against two types of attacks.

forgery attack: this type of attack consists of predicting the value of $\mathrm{MAC}_K(x)$ for a message x without initial knowledge of K. If there exists a message for which the adversary can do this, then he is said to be capable of *existential forgery*. If the adversary is able to determine the MAC for a message of his choice, he is said to be capable of *selective forgery*. Ideally, existential forgery is computationally infeasible; a less demanding requirement is that only selective forgery is so. Practical attacks often require that a forgery is *verifiable*, i.e. that the forged MAC is known to be correct (e.g. before attempting to use it to advantage) with probability near 1.

key recovery attack: this type of attack consists of finding the key K itself from a number of message/MAC pairs. Such an attack is more devastating than forgery, since it allows for arbitrary selective forgeries. Ideally, any attack allowing key recovery requires about $2^{|K|}$ operations, where $|K|$ denotes the bitlength of K. Verification of such an attack requires $|K|\,/m$ text-MAC pairs.

In addition, one requires that computing $\mathrm{MAC}_K(x)$ is computationally efficient, given the specification of the MAC algorithm, a specific value K, and an input x.

The attacks can be further classified according to the type of control an adversary has over the device computing the MAC value. In a *chosen-text attack*, an adversary may request and receive MACs corresponding to a number of messages of his choice, before completing his (forgery or key recovery) attack. For forgery, the forged MAC must be on a message different than any for which a MAC was previously obtained. In an *adaptive* chosen-text attack, requests may depend on the outcome of previous requests. In contrast, in *known-text attacks*, the adversary has access (only) to some number of text-MAC pairs. The most threatening types of attacks in practice are known-text attacks in which the number of text-MAC pairs required is very small.

Iterative MACs process inputs in successive fixed-size b-bit blocks. A message or text input x is divided into blocks x_1 through x_t, the last of which is padded appropriately if required for completeness. The MAC involves a *compression function* f and an n-bit ($n \geq m$) *chaining variable* H_i between stage $i-1$ and stage i:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, x_i), \qquad 1 \leq i \leq t \\ \mathrm{MAC}_K(x) &= g(H_t)\,. \end{aligned}$$

Here g denotes the *output transformation*, and IV denotes some initial value (e.g. a constant). The secret key may be employed in the IV, in f, and/or in g. For an input pair (x, x') with $\mathrm{MAC}_K(x) = g(H_t)$ and $\mathrm{MAC}_K(x') = g(H'_t)$, a collision is said to occur if $\mathrm{MAC}_K(x) = \mathrm{MAC}_K(x')$. This collision is called an *internal* collision if $H_t = H'_t$, and an *external* collision if $H_t \neq H'_t$ but $g(H_t) = g(H'_t)$.

A general forgery attack has been described that is applicable to all iterated MACs [24]. Its feasibility depends on the bitsizes n of the chaining variable and m of the hash-result, and the number s of common trailing blocks of the known texts ($s \geq 0$). The basic version is a known-text attack; if the message length is an input to the output transformation, all messages must have equal length. Two results are summarized here for convenience.

Lemma 1 ([24]) *An internal collision for an iterated MAC allows a verifiable MAC forgery, through a chosen-text attack requiring a single chosen text.* ∎

This follows since for an internal collision (x, x'), $\mathrm{MAC}_K(x \,\|\, y) = \mathrm{MAC}_K(x' \,\|\, y)$ for any single block y; thus a requested MAC on the chosen text $x \,\|\, y$ provides a forged MAC (the same) for $x' \,\|\, y$. Note this assumes that the MAC algorithm is deterministic.

Proposition 1 indicates how difficult it is to find an internal collision for a given MAC algorithm [24, 27].

Proposition 1 *Let* MAC() *be an iterated MAC with n-bit chaining variable, m-bit result, a compression function f which behaves like a random function (for fixed x_i), and output transformation g. An internal collision for* MAC *can be found using u known text-MAC pairs, where each text has the same substring of $s \geq 0$ trailing blocks, and v chosen texts. The expected values for u and v are: $u = \sqrt{2/(s+1)} \cdot 2^{n/2}$; $v = 0$ if g is a permutation or $s + 1 \geq 2^{n-m+6}$, and otherwise*

$$v \approx 2\left(\frac{2^{n-m}}{s+1} \cdot \left(1 - \frac{1}{e}\right) + \left\lfloor \frac{n - m - \log_2(s+1)}{m-1} \right\rfloor + 1\right). \tag{1}$$

∎

The details of the proof (see [27] for complete details) are quite involved, but for the case $s = 0$, a sketch of the main steps is included here because these results will be needed later in this paper.

A first lemma used in the proof of Proposition 1, is based on elementary probability theory (see for example [13]).

Lemma 2 *When drawing a sample of size r from a set of N elements with replacements, where $r \to \infty, N \to \infty$ and $r^2/(2N) \to \lambda$, the distribution of the number of coincidences converges to a Poisson distribution with expected value λ or*

$$\Pr(\#\ \textit{coincidences}\ = c) = e^{-\lambda} \cdot \frac{\lambda^c}{c!}, \qquad c \geq 0\,. \tag{2}$$

The probability that there is at least one coincidence is given by

$$1 - \exp(-\lambda)\,. \tag{3}$$

An identical result holds when one draws two samples of sizes r and s from a set of N elements with replacements, where $r, s, N \to \infty$ and $rs/N \to \lambda$.

This lemma implies that for $s = 0$, an internal collision will occur with high probability when $r = 2^{(n+1)/2}$ text-MAC pairs are available.

The second element of the proof is that if g is a random function (rather than a permutation), one expects about $r^2/2^{m+1}$ external collisions, given r text-MAC pairs. For external collisions, the forgery attack of Lemma 1 will not work. This means that an attacker has to separate internal and external collisions. This can be done by *simulating* the forgery attack of Lemma 1 as follows: given x and x' with $\mathrm{MAC}_K(x) = \mathrm{MAC}_K(x')$, the attacker chooses a random block y and asks for *both* values $\mathrm{MAC}_K(x\|y)$ and $\mathrm{MAC}_K(x'\|y)$. If (x, x') is an internal collision, these two values will always be equal; if (x, x') is an external collision, the two values are independent, and will only be equal with probability $1/2^m$. ■

Note that in MAA, the compression function f is considered to behave as a random mapping for fixed x_i, and the output transformation is not a permutation (cf. §3).

3 The Message Authenticator Algorithm

MAA was designed by D. Davies; it was presented at Crypto'84 [10, 11] and has been included in the ISO 8731-2 banking standard [16]. The algorithm was intended for mainframe computers circa 1983, which implies that it is also very efficient on current 32-bit PCs and workstations. The algorithm consists of three parts. The *prelude* is a key expansion from the 64-bit key K to 192 bits (six 32-bit words). First K is written as the concatenation of two 32-bit words, in this paper denoted by J_1 and J_2. From the first key word (J_1) one derives the parameters X_0, V_0, S; from the second one (J_2) one computes the parameters Y_0, W, T. The prelude, which needs only be executed during installation of a new key, also eliminates "weaker bytes" $\mathtt{00_x}$ or $\mathtt{FF_x}$ in keys and parameters[5] (this is achieved by the BYT procedure). More details are provided in Appendix B. The two words of the chaining variable (X_i, Y_i) are initialized with (X_0, Y_0). The *main loop* mixes the chaining variable with message word x_i $(0 \leq i \leq t-1)$ and key dependent parameters V and W. It consists of two inter-dependent parallel branches with logical operations, addition modulo 2^{32} ($\boxplus$), and multiplication modulo $2^{32}-1$ ($\otimes_1$) and modulo $2^{32}-2$ ($\otimes_2$); due to the peculiar definition (see [16]), the result may actually be equal to $2^{32}-1$ respectively $2^{32}-2$. In the *coda*, S and T are introduced as message blocks to be processed as per the main loop. The 32-bit MAC result is computed using addition mod 2: $\mathrm{MAA}_K(x) = X_{t+2} \oplus Y_{t+2}$.

A single iteration i of the main loop can be described as follows:

[5]Notation: $\mathtt{FF_x}$ denotes a hexadecimal byte, or 8-bit value FF of 8 ones.

Step 1: $V_i := \mathrm{rol}(V_{i-1})$; $Q_i := V_i \oplus W$; % rol denotes 1 bit cyclic shift left
Step 2: $t_1 := X_{i-1} \oplus x_i$; $t_2 := Y_{i-1} \oplus x_i$;
$X_i := t_1 \otimes_1 M_1(Q_i \boxplus t_2)$; $Y_i := t_2 \otimes_2 M_2(Q_i \boxplus t_1)$;

Here $M_1(\cdot)$ and $M_2(\cdot)$ are masking operations that each fix 8 bits (4 to zero and 4 to one):

$$M_1(z) = (z \vee A) \wedge C \qquad\qquad M_2(z) = (z \vee B) \wedge D\,,$$

where $A = \texttt{02040801}_\texttt{x}$, $B = \texttt{00804021}_\texttt{x}$, $C = \texttt{BFEF7FDF}_\texttt{x}$, and $D = \texttt{7DFEFBFF}_\texttt{x}$.

The output transformation g consists of the coda iterations (where the key-dependent S and T play the role of x_i) and the final XORing as noted above.

ISO 8731-2 [16] limits the size of the messages to $4 \cdot 10^6$ bytes (≈ 3.8 Mbyte). Also, the standard defines a special mode for messages longer than 1024 bytes (256 blocks). In this mode, MAA is applied to the first 1024 bytes, and the corresponding 4-byte MAC is prefixed to the next 1024 bytes of the message to form the new input of MAA. This procedure is repeated with the next 1024-byte block, until the end of the message is reached.

4 New Forgery Attacks

This section discusses how the basic attack of Proposition 1 may be customized for MAA, and optimized using "special messages". An extension of these ideas to the special mode of MAA for long messages is then presented.

4.1 Basic MAC Forgery

The basic attack of Proposition 1 can be applied to MAA with parameters $n = 64$, $m = 32$. However, an internal collision (i.e. a collision for X and Y) yields an external collision only if V is in the same rotational position (the number of bits over which V is rotated effectively adds 5 bits to the 64-bit MAA internal memory). This effect can be countered by using messages of the same length modulo 32, or messages for which the common trailing blocks start at the same position modulo 32. Note however that this is not strictly necessary: if the number of known texts given by Proposition 1 is multiplied by about 32, the condition on the length can be omitted. A second observation is that the output transformation consists of two iterations, providing two ways to obtain an external collision; consequently one expects two additional external collisions after processing S and T (for $2^{32.5}$ messages), which accounts for (an expected) four additional chosen texts to eliminate them. A third observation is that in the second chain, the modulus $(2^{32} - 2)$ is even, while the second multiplier is odd (the least significant bit of D is 1). This implies that the least significant bit of Y_i, denoted $\mathrm{LSB}(Y_i)$, is equal to the $\mathrm{LSB}(Y_{i-1} \oplus x_i)$. Consequently, $\mathrm{LSB}(Y_t) = LSB(Y_0) \oplus \bigoplus_{i=1}^{t} \mathrm{LSB}(x_i)$, and the 64-bit internal memory can be reduced to 63 bits by making the second term of the right hand side a constant. This assumes that the attacker can choose texts for which the value of this sum is constant.

Table 1 gives the parameters of the attack for various values s. Here $n = 63$, $m = 32$, and the last column counts the total number of bytes of the known and chosen texts. Increasing s allows a reduction in the number of known and chosen texts (the 'known' messages begin to resemble 'chosen' messages), but increases the total number of bytes which must be processed with the known key. This situation can be improved (as subsequently explained) by selecting messages with a special structure. Since fast MAA implementations process 3–4 Megabytes per second, processing 2^{32} bytes requires about 20 minutes.

Table 1: Parameters for basic forgery attack on MAA

# com. blocks s	# known texts u	length (bytes) $4(s+1)$	# chosen texts $v+1$	length (bytes) $4(s+2)$	total size (bytes)
0	2^{32}	≥ 4	$2^{31.3}$	≥ 8	$2^{35.2}$
2^8-1	2^{28}	$\geq 2^{10}$	$2^{23.3}$	$\geq 2^{10}+4$	$2^{38.1}$
$2^{16}-1$	2^{24}	$\geq 2^{18}$	$2^{15.3}$	$\geq 2^{18}+4$	$2^{42.0}$
$2^{32}-1$	2^{16}	$\geq 2^{34}$	4	$\geq 2^{34}+4$	$2^{50.0}$

4.2 Optimized MAC Forgery using Special Messages and Weak Keys

As noted by the designer of MAA, $2^{32}-1$ has several small prime factors (in fact $2^{32}-1 = 3 \cdot 5 \cdot 17 \cdot 257 \cdot 65537$), and if one of these appears in X_i, it might remain there due to multiplicative properties. The fact that x_i is added in every iteration should destroy this property; however if all x_i's for $i \geq i_0$ are chosen equal to 0, it is shown that such a factor would remain nonetheless.

Lemma 3 *Let $p > 1$ be a proper divisor of $2^{32}-1$. If $i = \alpha \cdot p - 1$ zero blocks are inserted, starting with x_{i_0}, then the probability that p divides X_{i_0+i} is at least $1 - e^{-\alpha}$.* ■

Proof: Assuming the X_i's are uniformly distributed (which has been confirmed by computer experiments), the probability that p divides X_{i_0} is equal to $1/p$. From Step 2 of the MAA description, it follows that if $x_{i_0} = 0$, X_{i_0+1} can be written as $X_{i_0} \cdot M_1(q) - q'(2^{32}-1)$, with q, q' integers. Thus, if p is a divisor of $2^{32}-1$ and p divides X_{i_0}, p will also divide X_{i_0+1}. The argument can be repeated (assuming independence), to show that after i zero blocks, the probability that p divides X_{i_0+i} is equal to $1-(1-\frac{1}{p})^{i+1}$. For $i = \alpha \cdot p - 1$, the probability becomes

$$1 - \left(1 - \frac{1}{p}\right)^{\alpha \cdot p} \geq 1 - e^{-\alpha}\,.$$

The latter inequality follows from the fact that the function $(1-\frac{1}{x})^x$ is a non-decreasing function of x for $x > 1$, with limit value $1/e$. Note that for $x \geq 257$, the function approximates its limit value with an error smaller than 0.2%. ■

Lemma 3 implies that after about 2^{16} iterations (each with $x_i = 0$), with probability 0.63, X_i will be divisible by all its prime factors and thus equal to $\texttt{FFFFFFFF}_\texttt{x}$ (and not $\texttt{00000000}_\texttt{x}$, due to the special definition of $\otimes_1$). This property can be exploited in an existential forgery attack as per Proposition 2.

Proposition 2 *An existential forgery on MAA is possible which requires a single chosen text of about 2^{18} bytes to forge the MAC for a text of length about 2^{38} bytes.* ■

Proof: The existential forgery is obtained by appending about 2^{18} zero bytes to an arbitrary message x and requesting the MAC for this message. The MAC for the same message with about 2^{38} appended zero bytes is then the same with high probability. It follows from Lemma 3 that after a sufficiently large number of zero blocks, the chaining variable X_i becomes constant. Once X_i is constant, finding a collision for Y_i yields an internal collision. Consider the second chain. For $X_i = \texttt{FFFFFFFF}_\texttt{x}$ and $x_i = \texttt{00000000}_\texttt{x}$, $Y_i = Y_{i-1} \otimes_2 U_i$, where

$U_i = ((\mathtt{FFFFFFFF_x} \boxplus Q_i) \vee B) \wedge D$. Note the value of U_i depends only on $i \bmod 32$. To make this equation independent of i, first define $\tilde{U} = \prod_{i=0}^{31} U_i$ (with multiplication $\bmod 2^{32}-2$). Then note $Y_{i+31} = Y_{i-1} \otimes_2 \tilde{U}$. Since $LSB(D) = 1$, and $2^{31}-1$ is a Mersenne prime, all U_i's are elements of the cyclic subgroup (of order $2^{31}-1$) of $\mathbb{Z}_{2^{32}-2}$. This implies that the same holds for $\tilde{U}$, and thus by Fermat's (little) theorem $\tilde{U}^{2^{31}-2} \bmod (2^{32}-2) = 1$. This may be used directly in a forgery attack as follows. A message ending with 2^{16} zero blocks (denoted $X \| 0^{2^{16}}$; here 0^i denotes i blocks of 32 zero-bits) has high probability of having $X_i = \mathtt{FFFFFFFF_x}$. If the MAC for such a message is requested, with high probability $X \| 0^{2^{16}} \| 0^{32 \cdot (2^{31}-2)}$ will have the same MAC. ∎

The existential forgery in the proof of Proposition 2, while not likely of practical utility to an adversary (the message ends in about 2^{38} zero bytes), illustrates a certificational weakness of MAA. Note that unlike many other attacks presented herein, this attack is prevented by the special long message mode of MAA.

The above assumptions are however worst case (for the attacker): for certain values of V and W, the U_i's will have a shorter period. This leads to the definition of weak keys for MAA. A first type of weak keys are the *rotational weak keys*: these are external keys which result in an internal key V of rotational period < 32. For such keys, rotating V over 2, 4, 8, or 16 positions will yield V again (there are no keys V of period 1 since the all zero and all one values are eliminated). There are respectively 2, 14, 254, and 64 516 values of V for which this holds.[6] One can find by exhaustive examination which values of the first 32-bit word of the input key yield such values of V. Therefore the number of rotational weak keys is independent of the second input key word, and thus 2^{32} times larger. If V has period r, the forgery above requires $r \cdot (2^{31}-2)$ zero blocks. Verifying the forgery allows an attacker to obtain information on V, which is undesirable since it leaks partial key bits. It is relatively easy (see §5) to detect whether a key is weak, leading to a key recovery attack on these keys. *These observations imply that the dependence of V on only 32 bits of the input key is a design weakness in MAA.*

A second class of weak keys are keys for which $\tilde{U}$ has small order. These are called *low order weak keys*. The order of $\tilde{U}$ must divide $\mathcal{D}$, where $\mathcal{D}$ is the order of $\mathbb{Z}_M^*$ for $M = 2^{32}-2$; in some cases the order of $\tilde{U}$ is considerably smaller than $\mathcal{D}$. More specifically, $\mathcal{D} = \phi(M) = \phi(2) \cdot \phi(2^{31}-1) = 2^{31}-2 = 2 \cdot 3^2 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331$, where $\phi()$ is Euler's totient function. For each divisor d of $\mathcal{D}$, the number of elements of order exactly d is $\phi(d)$, and the total number of elements whose order divides d is exactly d. For example, from $d = \mathcal{D}/331 = 6\,487\,866$, it follows that 1 key in 331 will yield a forgery after appending about $6\,487\,866 \cdot 32 \cdot 4 = 2^{29.6}$ zero bytes.

It is possible to obtain an internal collision using shorter messages (cf. Table 1) by exploiting the properties of zero blocks presented in Lemma 3. This requires chosen texts rather than known texts. As discussed above, for a text with a sufficiently large number of trailing zero blocks, X_t becomes constant (i.e. $\mathtt{FFFFFFFF_x}$) with high probability; if about $\sqrt{2} \cdot 2^{16}$ such texts are available, one expects by the birthday paradox (see Lemma 2) to find two texts with colliding values for Y_t as well. The third observation of §4.1 implies that 2^{16} such texts will suffice if the sum of the least significant bits of the message blocks is kept constant. It will be assumed that this condition is satisfied. Note that the attack can be extended easily to the case where the zero blocks are followed by some arbitrary common trailing blocks.

[6] Recall that bytes equal to $\mathtt{00_x}$ or $\mathtt{FF_x}$ are eliminated from V in the prelude.

The exact number of chosen texts for this improved variant can be computed as follows. Lemma 3 (with $p = 2^{16}+1$) implies that among $r = 2^{16}/(1-e^{-\alpha})$ messages each containing $\alpha(2^{16}+1)-1$ trailing zero blocks for some α, one expects to find about 2^{16} messages for which the first chaining variable X_t becomes equal to $\texttt{FFFFFFFF}_\texttt{x}$ (for simplicity it is assumed that $\alpha \geq 1/32$, since otherwise the second prime factor 257 has to be taken into account in the calculation). Lemma 2 implies that, among r such messages, the expected number of collisions for the second chaining variable Y_t, which corresponds to a (complete) internal collision is equal to $(2^{16})^2/2^{32} = 1$. The expected number of external collisions is equal to $r^2/2^{33} = 1/(2(1-e^{-\alpha})^2)$; these collisions can be eliminated by simulating the attack of Lemma 1 as discussed in section 2. Two additional chosen texts are sufficient to identify an external collision with high probability; the expected value is about $2(1-1/e)$ (for details, see [27]). The total number of blocks in the chosen texts is then approximately

$$2^{32}\frac{\alpha}{(1-e^{-\alpha})} + 2^{16}\left(1-e^{-1}\right)\frac{\alpha}{(1-e^{-\alpha})^2}\,. \tag{4}$$

If $\alpha \to 0$, the first term approaches 2^{32}, but the second term increases quickly. The sum is minimized for $\alpha \approx 1/228$ (but this violates the constraint on α), and for $\alpha > 1/228$ it increases monotonically with α. The number of blocks is thus minimized for $\alpha = 1/32$ (for this value the first term in equation (4) dominates and is approximately equal to $2^{32}+2^{26}$), while the number of chosen texts can be reduced by increasing α.

It is possible to use even shorter messages, but then it is assumed that X_t has become with high probability a multiple of $(2^{32}-1)/65\,537 = 65\,535 = \texttt{FFFF}_\texttt{x}$. A complete internal collision requires then a collision in a set of size about 2^{47}, which implies that more chosen messages are needed. In this case α must exceed 1/4 to avoid interference of the third prime factor 17.

An overview of the trade-offs is given in Table 2. These trade-offs are more realistic (cf. Table 1) with respect to the total number of bytes; this number increases only slightly while reducing the number of chosen texts. However, chosen texts are more difficult to obtain than known texts.

Table 2: Parameters for improved forgery attack on MAA. Attacks with < 256 zero blocks avoid the 'long message mode' (see §4.3).

	α	# 0 blocks	# chosen texts	total size (bytes)
$X_t = \texttt{FFFFFFFF}_\texttt{x}$	1/32	2 047	$2^{21.0}$	$2^{34.0}$
	1/4	16 383	$2^{18.2}$	$2^{34.2}$
	1/2	32 768	$2^{17.3}$	$2^{34.3}$
	1	65 636	$2^{16.7}$	$2^{34.7}$
	2	131 073	$2^{16.2}$	$2^{35.2}$
$X_t =$ multiple of $\texttt{FFFF}_\texttt{x}$	1/4	63	$2^{26.2}$	$2^{34.3}$
	1/2	128	$2^{25.3}$	$2^{34.4}$
	1	256	$2^{24.7}$	$2^{34.7}$
	2	513	$2^{24.2}$	$2^{35.2}$

4.3 Long Message MAC Forgery

For a fixed key and message block x_i, the compression function of MAA is not a permutation. This causes the "loss of memory" problem, as was pointed out by Block [8], and mentioned by Davies [10]. If a large number of variations of the first blocks are chosen, all 2^n states will be reached at some point. However, if the next message blocks are kept constant, it can be shown that the fraction of states $y[i]$ at stage i can be approximated by $2/(i + \frac{1}{3}\ln i + \frac{9}{5})$, for $i \geq 1$. To control this effect, ISO 8731-2 [16] limits the size of the messages to $4 \cdot 10^6$ bytes, and defines a special mode for messages longer than 1024 bytes (256 blocks) as described at the end of §3. The long message mode may be interpreted as the definition of a "meta" compression function based on MAA which compresses 1028 bytes to 4 bytes. This thwarts attacks using more than 256 zero blocks, including the forgery attack of Proposition 2 which requires a single chosen text.

However, it follows from Table 2 that the attack with zero blocks can be done with about $2^{24.7}$ messages of about 1000 bytes, independent of the special mode. Ironically the basic attack (using Proposition 1) of §4.1 works even slightly better with the special mode: in the case that $s \geq 256$, an additional non-bijective mapping exists, resulting in an additional opportunity for an internal collision. Consequently s may be replaced by $s + \lfloor s/256 \rfloor$.

5 Key Recovery Attacks

A key recovery attack on a MAC is considerably more serious than a forgery, as key recovery allows MAC forgery on arbitrary messages and without additional work, whereas forgeries are often existential only (and then of questionable practical use) and often chosen texts are required for each additional forged MAC. After discussing exhaustive key search parameters it is explained how an internal collision for MAA, under certain circumstances, allows key recovery.

5.1 Exhaustive Key Search

If three text-MAC pairs are available for MAA, the 64-bit user key can be recovered by trying all 2^{64} possibilities. The prelude is quite complex, but since the processing of J_1 and J_2 is almost independent (the only interaction is through the BYT operation), this represents only a relatively small computation ($\leq 2^{37}$ multiplications). Verifying a key on a single text-MAC pair is the main part of the effort: for a t-block text this requires about $2t+4$ multiplications. The total expected number of multiplications for $t = 1$ is then $3 \cdot 2^{64}$. This effort should be compared to a DES key search which requires 2^{55} encryptions; using dedicated hardware about 3 hours are sufficient to find the key with an investment of US$ 1 million [32] (in 1993 dollars and technology). A similar investment in 1996 will find an MAA key in about 1 week.

5.2 Key Recovery from Collisions

A first case occurs when the key is a rotational weak key, i.e., a weak key for which the period of $V < 32$ (see §4.2). One can verify the period of V by simulating the attack of Lemma 1, using two sets of $2^{17.3}$ chosen texts containing 131 072 trailing zero blocks each. In one set all messages have length 0 mod 32; in the other all have length 16 mod 32. One expects 9 internal collisions between messages of these two sets and 6 external collisions, but these cannot yet be distinguished from each other. The simulated attack of Lemma 1 then (with high probability) filters the external collisions. If V has period < 32, this attack will work for all the internal collisions, and it will fail otherwise. The key recovery attack will fail if there are no internal collisions; since the number of internal collisions is Poisson distributed, the probability of this event, for 9 internal collisions as above, is $e^{-9} \approx 0.01\%$. In this way it is evident if V belongs to this special set of 2^{48} keys. If so, finding the key then requires an exhaustive search over only 2^{48} keys (rather than 2^{64}). This attack has success probability 2^{-16} (since 2^{48} of 2^{64} keys are weak). Note that in this case the long message mode can be thwarted by using two sets of $2^{25.8}$ messages with about 256 trailing zero blocks; this will yield about 9 internal collisions, and about $2^{19.6}$ external collisions. The rest of the attack is similar; the second step requires slightly more work. These observations can be partially summarized as follows.

Proposition 3 *For MAA, one can detect, using 2^{27} chosen messages of about 1 Kbyte each, whether a key belongs to a subclass of 2^{48} weak keys.* ∎

A second attack is based on an internal collision which is created by the message only, i.e. the chaining variables that enter the round are identical. This can be achieved by trying all 2^{32} possible values for the first message block, or similarly by doing this for the first block after a number of common leading blocks. Two messages which have the first $t-1$ blocks in common and for which $x'_t = x_t \oplus d$ form an internal collision if and only if both equations (5a) and (5b) hold:

$$\begin{aligned}
&(X_{t-1} \oplus x_t) \otimes_1 M_1((V_t \oplus W) + (Y_{t-1} \oplus x_t)) = \\
&\qquad (X_{t-1} \oplus x_t \oplus d) \otimes_1 M_1((V_t \oplus W) + (Y_{t-1} \oplus x_t \oplus d)) \qquad (5a)\\
&(Y_{t-1} \oplus x_t) \otimes_2 M_2((V_t \oplus W) + (X_{t-1} \oplus x_t)) = \\
&\qquad (Y_{t-1} \oplus x_t \oplus d) \otimes_2 M_2((V_t \oplus W) + (X_{t-1} \oplus x_t \oplus d))\,. \qquad (5b)
\end{aligned}$$

In the following it will be assumed that $t = 1$, and the subscripts from x, X, Y, and V will be omitted. The problem now is to solve the equations (5a) and (5b) for the key.

If the main loop of MAA behaves as a random mapping, Lemma 2 implies that the number c of internal collisions between all 2^{32} 1-block messages is a Poisson distributed random variable with parameter $\lambda = 1/2$. Because of the LSB property, the same expression holds if only all the 2^{31} even (or odd) blocks are considered. The probabilities to have at least one internal collision are indicated in Table 3.

Table 3: Probability that there is at least one internal collision after a single block, given that messages of the same parity are used whenever possible.

# messages	$\Pr(c \geq 1)$
2^{29}	1.6%
2^{30}	6.0%
2^{31}	22.1%
2^{32}	39.3%

If the BYT operation on X, Y, V, and W is neglected, then X and V are determined by the first key register J_1, and Y and W by the second key register J_2. A random word is left unchanged by the BYT operation with probability $(1-2^{-7})^4 = 0.97$. Thus the fraction of keys where the BYT operation has no influence can be estimated as $(0.97)^4 = 0.88$.

Solving the two equations (5a) and (5b) for J_1 and J_2 is not straightforward. The remainder of this section explains how this can be done. The following strategy is used. First equation (5b) is rewritten by defining two additional variables a and z. Then all (a, z) pairs that are solutions of the equation are generated. On average 2^{32} solutions are expected. In the next step candidates for J_1 and J_2 are determined. In the last step it is checked whether these candidates are also solutions of equation (5a).

5.3 Determination of a and z

To facilitate notation, define:

$$\begin{aligned} a &= Y_0 \oplus x \\ b &= X_0 \oplus x \\ z &= Q + b \end{aligned}$$

Using this notation, equation (5b) can be rewritten as:

$$a \otimes_2 M_2(z) = (a \oplus d) \otimes_2 M_2(z \oplus d \oplus \delta)\,, \tag{6}$$

where δ is a correction term which accounts for the interchanging of the order of addition modulo 2^{32} and exor:

$$\delta = (Q + (b \oplus d)) \oplus (Q + b) \oplus d\,.$$

In the next subsection it will be explained how δ can be predicted accurately.

By the definition of $\otimes_2$, equation (6) is:

$$a \times M_2(z) = ((a \oplus d) \times M_2(z \oplus d \oplus \delta)) \bmod 2^{32} - 2\,.$$

Since $2^{32} - 2$ factors to $2 \times (2^{31} - 1)$, this can be rewritten as

$$\begin{aligned} a \times M_2(z) &= (a \oplus d) \times M_2(z \oplus d \oplus \delta) \bmod 2 && \text{(7a)} \\ a \times M_2(z) &= (a \oplus d) \times M_2(z \oplus d \oplus \delta) \bmod 2^{31} - 1\,. && \text{(7b)} \end{aligned}$$

Since $M_2(\cdot)$ is odd, d must be even. For even d every (a, z) pair is a solution of equation (7a). Equation (7b) is rewritten as:

$$a \times (a \oplus d)^{-1} = ((M_2(z))^{-1} \times M_2(z \oplus d \oplus \delta)) \bmod 2^{31} - 1\,.$$

Here "$^{-1}$" denotes "inverse modulo $2^{31} - 1$" (the case $a \oplus d = 0 \bmod 2^{31} - 1$ has to be considered separately). The left hand side of this equation only depends on a, the right hand side only depends on z. For given d and guessed δ, now two tables are created: one that lists all possible left hand sides and their corresponding values for a, the other lists all possible right hand values and the corresponding values for z. Each match in these tables gives an (a, z) pair that is a solution of the equation. On average 2^{32} matching pairs are expected. In the next steps J_1 and J_2 are determined from a and z and it is checked whether these are also solutions of the first equation.

5.4 Prediction of δ

The value of δ depends on the values of $V \oplus W$ and $X_0 \oplus x$. A class of weak keys can be defined, where $V \oplus W = 0$ or $V \oplus W = \mathtt{8000_x}$. For this class of keys, δ is always 0. The expected number of keys in this class is 2^{33}.

In the more general case the most probable values of δ are calculated, using only knowledge of d. If Q and b are considered as random variables, a statistical model for the bits of δ can be built. Recall the definition of δ from §5.3:

$$\delta = (Q + (b \oplus d)) \oplus (Q + b) \oplus d\,.$$

The carry bits of the first addition are denoted with $c[i]$, the carry bits of the second addition with $e[i]$. The equation leads to the following equations at the bit level, where the bits of a word are numbered from the right to the left (the least significant bits get number 0, the most significant number 31):

$$\begin{aligned}
\delta[i] &= c[i] \oplus e[i] \\
c[i+1] &= c[i]Q[i] \oplus c[i](b[i] \oplus d[i]) \oplus Q[i](b[i] \oplus d[i]) \\
e[i+1] &= e[i]Q[i] \oplus e[i]b[i] \oplus Q[i]b[i]\,,
\end{aligned}$$

where $c[0]$ and $e[0]$ are equal to 0, or

$$\delta[i+1] = (Q[i] \oplus b[i])\delta[i] \oplus (Q[i] \oplus c[i])d[i]\,, \tag{8}$$

where $\delta[0] = 0$. The probability for δ can be described with a Markov model: given the probabilities P_i for the state i, P_{i+1} can be calculated by using the transition probability matrices T_0 ($d[i] = 0$) and T_1 ($d[i] = 1$):

$$P_{i+1} = \begin{bmatrix} \Pr(\delta[i+1] = 0) \\ \Pr(\delta[i+1] = 1) \end{bmatrix} = T_{d[i]} \cdot P_i\,.$$

The transition matrices can easily be calculated:

$$T_0 = \begin{bmatrix} 1 & 0.5 \\ 0 & 0.5 \end{bmatrix}, \qquad T_1 = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}.$$

In the attack on MAA, for each value guess for δ a new table must be calculated. To reduce the expected workload, the most probable values of δ are tried first. Alternatively, the maximum number of operations to be performed can be determined in advance, the δ values are chosen to maximize the success probability. The set of δ's that have maximal probability forms a vector space. From the transition matrices, one concludes that if $d[i] = 1$, $\delta[i+1]$ can take either value with equal probability. Each 1-bit of d divides the maximal probability of δ by a factor of 2 (except for the most significant bit of d, which has no influence on the value of δ). If $d[i] = 0$, the value of $\delta[i]$ determines the probability of $\delta[i+1]$ to take a value. Therefore if $d[i]$ is zero, or $d[i+1]$ is zero, $\delta[i+1]$ has to be zero for a δ with maximal probability. The other bits can be chosen at random. An exception to this rule is the most significant bit: if $d_{30} = 1$, δ_{31} can be chosen at random.

Table 4 shows the results of a simulation to determine the expected value of the summed probability of 2^t δ's (optimally chosen). It can be seen that the work factor of the attack

Table 4: Expected value of the summed probability of success p after an optimal selection of 2^t values for δ (for random values of d). Values are given for the full MAA (operating on 32-bit words), and for two reduced versions (operating on 14- or 18-bit words).

t	$\log_2 p$		
	14 bit	18 bit	32 bit
0	-4.0	-5.4	-10.3
2	-2.6	-3.8	-8.4
4	-1.8	-2.8	-7.0
6	-1.4	-2.3	-5.9

depends on the key that is being used. For instance, if $Q = 0$, δ is always equal to zero and the key is found immediately. For other values of Q the success probability is smaller. Keys which yield a value of Q with a low Hamming weight are thus easier to recover; they will be called *low Hamming weight* weak keys.

Note that the masking operation in MAA plays here to the advantage of the attacker: values of δ that differ only in bits that are masked, can be considered at the same time without additional effort.

5.5 Determination of J_1 and J_2 Candidates

J_2 is easily found from a table that lists for all $Y_0(J_2) = a(J_2) \oplus x$ the corresponding J_2. The relation between J_1 and z is less clear:

$$z = Q(J_1, J_2) + b(J_1) = (V(J_1) \oplus W(J_2)) + (X_0(J_1) \oplus x)\,. \tag{9}$$

And also

$$M_2(z \oplus d \oplus \delta) = M_2((V \oplus W) + (X_0 \oplus x \oplus d))\,. \tag{10}$$

To determine J_1 first a table is created with all possible values of $V(J_1) \oplus X_0(J_1)$, and the corresponding J_1-values. Then all possible values of z^* are determined,

$$z^* = k \oplus b = (V(J_1) \oplus W(J_2)) \oplus (X_0(J_1) \oplus x)\,.$$

Using equation (9) this becomes (the carry bits are denoted with $e[i]$):

$$\begin{aligned}
z^*[0] &= Q[0] \oplus b[0] = z[0] \\
z^*[i] &= Q[i] \oplus b[i] = z[i] \oplus e[i] = z[i] \oplus Q[i-1]b[i-1] \oplus (Q[i-1] \oplus b[i-1])e[i-1] \\
&= z[i] \oplus Q[i-1]b[i-1] \oplus z^*[i-1](1 \oplus z[i-1])\,.
\end{aligned}$$

It follows that:

$$\begin{aligned}
z^*[i-1] = 1 \quad &\Rightarrow \quad z^*[i] = z[i] \oplus 1 \oplus z[i-1] \\
z^*[i-1] = 0 \quad &\Rightarrow \quad z^*[i] = z[i] \oplus Q[i-1]\,.
\end{aligned}$$

In the last case equation (10) is used to determine $Q[i-1]$. If bit i is not masked, and $d[i-1] \neq 0$, equation (8) can be used:

$$\delta[i] = \delta[i-1](Q[i-1] \oplus b[i-1]) \oplus (Q[i-1] \oplus c[i-1])d[i-1]$$

and thus

$$\delta[i] = Q[i-1] \oplus c[i-1] = Q[i-1] \oplus \delta[i-1] \oplus e[i-1]\,.$$

Since

$$e[i-1] = z[i-1] \oplus z^*[i-1] = z[i-1]\,,$$

$Q[i-1]$ can be written as:

$$Q[i-1] = \delta[i] \oplus \delta[i-1] \oplus z[i-1]\,.$$

If bit i is masked or $d[i-1] = 0$, $z^*[i]$ cannot be determined, and both possibilities (the value 0 or 1) have to be checked.

Experimental results suggest that the average number of possible values is less than 2^{11}. For each value of z', J_1 is found by a search for the J_1-values corresponding with $z' \oplus W \oplus x$.

In the last step, J_1 and J_2 are checked. (J_1, J_2) pairs that produce an internal collision are output as possible key values.

5.6 Extensions

The same analysis can be applied starting from equation (5a) instead of (5b). Thus if starting from one equation the key is not found after a predetermined amount of operations, the other equation can be used, with hopefully more luck.

Solving the equations will become easier when two or more simultaneous internal collisions are known. For 2^{32} known texts, this event has a probability of 9%.

5.7 Experimental Results

The attack was implemented in C for several reduced versions of MAA. An l-bit reduced version uses l-bit working variables and key registers; it operates on l-bit message blocks. Table 5 gives the results. For the full version of MAA, the attack was executed with a reduced search space and the results were extrapolated. Because of the large effort to produce collisions for arbitrary keys, only a few 32-bit collisions were generated. This means a large value of t had to be used, and thus a large number of (J_1, J_2)-candidates were found.

Table 5: Experimental results for the key recovery attack on MAA.

version	t	success probability	number of (a, z)-pairs	number of (J_1, J_2)-candidates
14 bit words	2	21.6 %	$2^{16.9}$	$2^{21.8}$
18 bit words	2	2.6 %	$2^{21.0}$	$2^{26.9}$
18 bit words	4	8.1 %	$2^{22.8}$	$2^{28.2}$
32 bit words	6		$2^{38.9}$	$2^{49.5}$

6 Collision Clusters

In this section it is shown that a subset of MAA keys, called "collision cluster keys", can be detected and subsequently recovered much more easily than arbitrary MAA keys.

Consider two messages x and x' which differ only in a single word; for simplicity, it will be assumed that the messages contain only one word, i.e., $t = 1$. The expected number of internal collisions between all 2^{32} 1-block messages is equal to 1/2 (cf. §5). For a given value of $x \oplus x' = d$,

$$\Pr\{(X_1, Y_1) = (X_1', Y_1')\} \approx 1/2^{64}\,.$$

In this section it will be shown that there exists a large class of keys ($\geq 2^{33}$ elements) for which between 2^{11} and 2^{19} collisions occur (all with the same value of $d = x \oplus x'$). These weak keys will be called *collision cluster keys.* Thirteen bits of d must be equal to zero, with the other 19 bits being key dependent. This bounds the internal collision probability:

$$1/2^{40} \leq \Pr\{(X_1, Y_1) = (X_1', Y_1')\} \leq 1/2^{32}\,.$$

When d is known,

$$1/2^{21} \leq \Pr\{(X_1, Y_1) = (X_1', Y_1')\} \leq 1/2^{13}\,.$$

This cannot be used for a MAC forgery for an unknown key, because the probabilities are too small (compared to $1/2^{32}$). But, it will be shown that such collision cluster keys can be detected and subsequently recovered using about 2^{23} chosen texts.

A collision cluster key is defined as a key for which the number of collisions between all possible 1-block messages is substantially larger than two. Several classes of such keys can be distinguished. First a rather obvious class is described in detail. Then other types of collision cluster keys are mentioned.

6.1 Simple Collision Cluster Keys

In the following ϵ_i denotes $X_i \oplus Y_i$. Without loss of generality it is assumed that the collision occurs after the first iteration, and again the subscripts of Q, ϵ, X, Y, V, and W are omitted. An obvious sufficient set of conditions for a collision is:

$$\begin{aligned}
X \oplus x &= M_1(Q + (X \oplus x \oplus \epsilon \oplus d)) && \text{(11a)}\\
X \oplus x \oplus d &= M_1(Q + (X \oplus x \oplus \epsilon)) && \text{(11b)}\\
X \oplus x \oplus \epsilon &= M_2(Q + (X \oplus x \oplus d)) && \text{(11c)}\\
X \oplus x \oplus \epsilon \oplus d &= M_2(Q + (X \oplus x)) && \text{(11d)}
\end{aligned}$$

The equations (11) can be written at bit level (the least significant bit is number 0). The carry bits of the addition modulo 2^{32} ("+") into bit i are denoted with $u[i]$, $v[i]$, $w[i]$, and $t[i]$. The equations for the carry bits become (with $u[0] = v[0] = w[0] = t[0] = 0$):

$$\begin{aligned}
u[i+1] &= u[i](Q[i] \oplus X[i] \oplus x[i] \oplus \epsilon[i] \oplus d[i]) \oplus Q[i](X[i] \oplus x[i] \oplus \epsilon[i] \oplus d[i])\\
v[i+1] &= v[i](Q[i] \oplus X[i] \oplus x[i] \oplus \epsilon[i]) \oplus Q[i](X[i] \oplus x[i] \oplus \epsilon[i])\\
w[i+1] &= w[i](Q[i] \oplus X[i] \oplus x[i] \oplus d[i]) \oplus Q[i](X[i] \oplus x[i] \oplus d[i])\\
t[i+1] &= t[i](Q[i] \oplus X[i] \oplus x[i]) \oplus Q[i](X[i] \oplus x[i])
\end{aligned}$$

The masking of bit i determines the rest of the bit level equations. Four cases can be distinguished:

0. When both M_1 and M_2 leave bit i untouched:

$$u[i] = v[i] = w[i] = t[i] = Q[i] \oplus \epsilon[i] \oplus d[i]\,.$$

1. When M_1 sets or clears bit i, but M_2 does not:

$$\begin{aligned} w[i] = t[i] &= Q[i] \oplus \epsilon[i] \\ d[i] &= 0 \\ X[i] \oplus x[i] &= M_{1i}\,. \end{aligned}$$

2. When M_2 sets or clears bit i, but M_1 does not:

$$\begin{aligned} u[i] = v[i] &= Q[i] \oplus \epsilon[i] \\ d[i] &= 0 \\ X[i] \oplus x[i] \oplus \epsilon[i] &= M_{2i}\,. \end{aligned}$$

3. When both M_1 and M_2 determine bit i:

$$\begin{aligned} d[i] &= 0 \\ \epsilon[i] &= M_{1i} \oplus M_{2i} \\ X[i] \oplus x[i] &= M_{1i}\,. \end{aligned}$$

The next step is to write down these equations for the 32 bits (cf. Appendix A). The result is three sets of equations:

(i) A set of conditions on Q and ϵ: these conditions determine the class of collision cluster keys. The class of keys where Q_1 and ϵ_0 satisfy the equations is called the base class of collision cluster keys. If Q_1 and ϵ_0 do not satisfy the equations, it is still possible that Q_i and ϵ_{i-1} $(1 < i)$ do. These keys can be attacked by using messages consisting of $i+1$ blocks, where the first i blocks are common. This means that by doing more work the probability of success can be enhanced. In the generic case there are 32 different values for Q (because of the rotation operation). Varying the choice of common blocks, all values of ϵ can be created (but the values cannot be controlled).

(ii) Equations that give d as a function of Q and ϵ.

(iii) Conditions on $X \oplus x$ that determine for given X a vector space of messages.

These equations were solved for a reduced version of MAA that operates on 14-bit words instead of 32-bit words. For this version of MAA there are about $2^{14.6}$ keys in the base weak key class. Each weak key has an associated difference d and an associated vector space. Two messages that have difference d will produce a collision with a probability of 2^{-4}, 2^{-5}, or 2^{-6}. The probability depends on the size of the vector space of messages associated with the weak key. The equations for the full MAA were written down and the expected number of basic collision cluster keys equals 2^{33}. For each key there exists an associated difference d such that between 2^{11} and 2^{19} block messages with this difference will collide.

To demonstrate the existence of weak keys, a key that produces a Q_2 satisfying the equations was searched. Afterwards a first message block x_1 such that ϵ_1 is also a solution was searched.

Example 1 *Let* $J_1 = \texttt{e8813bb2}_{\texttt{x}}$, $J_2 = \texttt{45cfb69c}_{\texttt{x}}$, *and* $x_1 = \texttt{56e2}_{\texttt{x}}$. *Then there exist* 2^{18} *pairs of two block messages with the first message block equal to* x_1, *and the second message blocks differing by* $d = \texttt{1c081098}_{\texttt{x}}$ *that produce an internal collision.*

6.1.1 The Use of a Collision Cluster in an Attack

The existence of a collision cluster of size 2^p can be used in a differential attack to recover the key. Two messages with difference d will produce a collision with probability 2^{p-32}. The first step is to recover d. 13 bits of d are known to be zero. The message space can be divided in 2^{13} subspaces with constant values for these 13 bits. The collision cluster will be situated in one of these subspaces. In this subspace the difference will have a collision probability equal to 2^{p-19}. Since it is not known beforehand which subspace is the correct one, the attack has to be repeated for all of them. The 19 unknown bits of d can be found by comparing the MAC results for messages from the same subspace. With 2^n texts 2^{2n-1} pairs can be generated. A pair with the correct difference d will generate a collision with probability 2^{p-19}. A collision will occur with large probability when

$$\begin{array}{rcl} 2^{2n-1} & \geq & \left(2^{-19} \times 2^{p-19}\right)^{-1} \\ & \Updownarrow & \\ n & \geq & \dfrac{39-p}{2} . \end{array}$$

For the largest clusters, $p = 19$; only $2^{13} \cdot 2^{10} = 2^{23}$ texts are needed. When $p = 11$, 2^{27} texts are needed. Once d is known the bit equations can be used to determine bits of $X_0, Y_0, X_0 \oplus Y_0$, and Q_0. Off-line built tables of $X_0(J_1), V_1(J_1), Y_0(J_2)$, and $W(J_2)$ allow to determine J_1 and J_2. The following proposition summarizes this result.

Proposition 4 *A key that has an associated difference with a collision cluster of size 2^p, can be detected with a differential attack using $2^{13} \times 2^{(39-p)/2} = 2^{(65-p)/2}$ chosen messages. The value of the difference and the colliding messages gives enough information for recovery of the key with a lookup table.*

The collision cluster can be used for forging messages that lie in the subspace of the cluster. As explained above, a message m^* will produce the same MAC as the message m with probability $2^{p-19} \cdot 2^{-19} = 2^{p-38}$. This probability varies between 2^{-19} and 2^{-27}. However, which of the 2^{13} subspaces is forgeable, depends on the actual key value.

The enhanced collision probability also allows one to optimize the basic forgery attack on MAA. The required number of known messages u reduces from $\sqrt{2/(s+1)} \cdot 2^{32}$ to $2^{(65-p)/2}$. Since in this section we have only considered collisions starting with equal chaining variables, the known messages should differ in only one block. Also the number of common trailing blocks s is not relevant.

6.2 Involved Weak Keys

The set of collisions with the same difference d that occurs for weak keys, can be seen as a "burst" of collisions. Knowledge of one collision enables an attacker to create very easily the whole vector space of collisions. The interaction between masking and the two modular multiplications causes many other "bursts" of collisions. Below one other example is presented.

Example 2 *Consider a reduced version of MAA, operating on 14-bit words instead of 32-bit words. Let $J_1 = \texttt{72d}_\texttt{x}$, $J_2 = \texttt{3a39}_\texttt{x}$. Then there exist 2^9 one block message pairs $(x, x \oplus \texttt{31d8}_\texttt{x})$ that produce a collision.*

To explain the phenomenon, the example is analyzed in some detail. Equation (5a) gives:

$$(\texttt{1dce}_\texttt{x} \oplus x) \otimes_1 M_1(1 + (\texttt{13cb}_\texttt{x} \oplus x)) = (\texttt{2c16}_\texttt{x} \oplus x) \otimes_1 M_1(1 + (\texttt{2233}_\texttt{x} \oplus x)) \,. \tag{12}$$

$x = \texttt{c22}_\texttt{x}$ is a solution, called the base solution. Filling in gives:

$$\begin{aligned} \texttt{11ec}_\texttt{x} \times M_1(1 + \texttt{1fe9}_\texttt{x}) &= \texttt{11ec}_\texttt{x} \times \texttt{1fcb}_\texttt{x} = \texttt{130b}_\texttt{x} + \texttt{8e7}_\texttt{x} \times (2^{14} - 1) \\ \texttt{2034}_\texttt{x} \times M_1(1 + \texttt{2e11}_\texttt{x}) &= \texttt{2034}_\texttt{x} \times \texttt{2e13}_\texttt{x} = \texttt{130b}_\texttt{x} + \texttt{172f}_\texttt{x} \times (2^{14} - 1) \end{aligned}$$

Consider now the slightly modified equation (12):

$$(\texttt{11ec}_\texttt{x} + x') \otimes_1 (\texttt{1fcb}_\texttt{x} + x'') = (\texttt{2034}_\texttt{x} - x'') \otimes_1 (\texttt{2e13}_\texttt{x} - x') \,. \tag{13}$$

This can be rewritten as

$$\texttt{11ec}_\texttt{x} \times \texttt{1fcb}_\texttt{x} - \texttt{2034}_\texttt{x} \times \texttt{2e13}_\texttt{x} + (\texttt{1fcb}_\texttt{x} + \texttt{2034}_\texttt{x})x' + (\texttt{11ec}_\texttt{x} + \texttt{2e13}_\texttt{x})x'' = 0 \bmod (2^{14} - 1) \,. \tag{14}$$

The base solution of (12) corresponds with the solution $x' = x'' = 0$ of (14). Because $\texttt{11ec}_\texttt{x} + \texttt{2e13}_\texttt{x} = 2^{14} - 1$ and $\texttt{1fcb}_\texttt{x} + \texttt{2034}_\texttt{x} = 2^{14} - 1$, all values of x', x'' will satisfy equation (13). It can be concluded that every x^* for which an x' and an x'' can be found such that

$$\begin{aligned} \texttt{1dce}_\texttt{x} \oplus x^* &= \texttt{11ce}_\texttt{x} + x' \\ M_1(1 + (\texttt{13cb}_\texttt{x} \oplus x^*)) &= (\texttt{1fcb}_\texttt{x} + x'') \\ \texttt{2c16}_\texttt{x} \oplus x^* &= \texttt{2034}_\texttt{x} - x'' \\ M_1(1 + (\texttt{2233}_\texttt{x} \oplus x^*)) &= \texttt{2e13}_\texttt{x} - x' \end{aligned}$$

will be a solution of (12). A similar equivalence between addition and the interaction of M_2, Q, and exor must exist for equation (5b). The example shows that there exist keys and x-values in practice with a large cluster of x^* values.

7 Conclusion

Two improved variations of a general MAC forgery attack have been presented, tailored for MAA. In addition it has been shown that the internal structure of MAA may be exploited to reduce the total number of bytes of known text-MAC pairs required for the attack.

Two new key recovery attacks have been presented. The first attack shows how the secret key can be recovered after about 2^{31} chosen texts. The required effort depends on the Hamming weight of a value computed by two subkeys, $Q = V_0 \oplus W$. The second attack is based on the fact that for certain weak keys, MAA exhibits clusters of collisions; for some keys 2^{23} chosen texts suffice to recover the key. The attacks are summarized in Table 6 and Table 7.

These attacks have exposed several weaknesses of the MAA algorithm:

(i) The fact that three outputs of the prelude stage (see §3) depend on one half of the key, and the other three on the other half, makes the key recovery attack much easier. This problem could be avoided in part by imposing the constraint that messages be at least two blocks long, or by prepending one or two fixed non-zero blocks to each message, e.g., S and T as defined in §3.

(ii) The specific value of Q may critically affect the security of MAA. It is recommended to check it for weak values; applying the BYT function to it would be a good start. It would be welcome to update Q in a more complex way, e.g., by using also S and T and by using completely different values for Q in the equations for X_i and Y_i. The collision cluster attack can be eliminated by a stronger BYT function ensuring that Q is not a solution of the equations in Appendix A.

(iii) The main loop is vulnerable to message inputs equal to 0; this could be addressed partially by replacing the computation $x_i \oplus X_{i-1}$ by $x_i \oplus X_{i-1} \oplus E_i$ for a non-zero constant E_i, depending on the key and possibly the number of the iteration.

(iv) The effective size of the internal memory (63 bits) is too small to preclude a forgery attack. A solution to this problem is to insert an additional block between x_t and the coda blocks S and T. The additional block could be either a sequence number or a (pseudo)-random number.

A general measure would be to change the key frequently, for example after 2^{16} texts, and to check for certain classes of weak keys.

The attacks discussed in this paper can be further optimized; for example, the key recovery attack can be made much more powerful if two simultaneous internal collisions are known. There are probably additional ways to exploit the non-random behavior which we have observed in MAA.

While the key size of MAA probably limits its lifetime to at most five more years beyond 1997, this paper gives an indication that its use even in current applications should be re-evaluated carefully. Moreover, when MAA is replaced by a successor not only should the key size be enlarged, but also the described weaknesses should be eliminated.

Table 6: Selective summary of the forgery attacks.

attack	# known texts	# chosen texts	# common blocks	total size
basic attack (§4)	2^{28}	$2^{23.3}$	255 (trailing)	$2^{18.1}$ Mbyte
collision cluster (§6)	2^{27}		all but one	
improved attack (§4)		$2^{24.7}$	256 (trailing)	$2^{14.7}$ Mbyte

Table 7: Selective summary of the key recovery attacks.

attack	key fraction	# messages	**k**nown or **c**hosen	work
exhaustive search	1	3	k	2^{63}
rotationally weak keys (§4)	2^{-16}	2^{27}	c	2^{48}
collision (§5)	2^{-12}	2^{29}	c	2^{50}
collision cluster (§6)	2^{-31}	2^{27}	c	negligible

Acknowledgements

The authors would like to thank Erik De Win for assisting with the software implementation of the collision search and the anonymous referees for their helpful comments.

References

[1] ANSI X9.9 (revised), "Financial institution message authentication (wholesale)," American Bankers Association, April 7, 1986.

[2] ANSI X9.19, "Financial institution retail message authentication," American Bankers Association, August 13, 1986.

[3] M. Bellare, J. Kilian, P. Rogaway, "The security of cipher block chaining," *Proc. Crypto'94, LNCS 839*, Springer-Verlag, 1994, pp. 341–358.

[4] M. Bellare, R. Guérin, P. Rogaway, "XOR MACs: new methods for message authentication using block ciphers," *Proc. Crypto'95, LNCS 963*, Springer-Verlag, 1995, pp. 15–28.

[5] M. Bellare, R. Canetti, H. Krawczyk, "How to key Merkle–Cascaded pseudorandomness and its concrete security," 10 November 1995, http:// www.research.ibm.com/security/.

[6] M. Bellare, R. Canetti, H. Krawczyk, "Keying hash functions for message authentication," *Proc. Crypto'96, LNCS 1109*, Springer-Verlag, 1996, pp. 1–15. Full version: http:// www.research.ibm.com/security/.

[7] E. Biham, A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard,* Springer-Verlag, 1993.

[8] H. Block, "File authentication: A rule for constructing algorithms," *SÄKdata Report*, October 12, 1983.

[9] A. Bosselaers, H. Dobbertin, B. Preneel, "The RIPEMD-160 cryptographic hash function," *Dr. Dobb's Journal,* Vol. 22, No. 1, January 1997, pp. 24–28.

[10] D. Davies, "A message authenticator algorithm suitable for a mainframe computer," *Proc. Crypto'84, LNCS 196*, Springer-Verlag, 1985, pp. 393–400.

[11] D. Davies, D.O. Clayden, "The message authenticator algorithm (MAA) and its implementation," *NPL Report DITC 109/88*, Feb. 1988.

[12] D. Davies, W. Price, *Security for Computer Networks*, 2nd ed., Wiley, 1989.

[13] W. Feller, *"An Introduction to Probability Theory and Its Applications, Vol. 1,"* Wiley & Sons, 1968.

[14] FIPS 46, *Data encryption standard,* NBS, U.S. Department of Commerce, Washington D.C., Jan. 1977.

[15] FIPS 180-1, *Secure hash standard,* NIST, US Department of Commerce, Washington D.C., April 1995.

[16] ISO 8731:1987, *Banking – approved algorithms for message authentication, Part 1, DEA,* IS 8731-1, *Part 2, Message Authentication Algorithm (MAA),* IS 8731-2.

[17] ISO/IEC 9797:1993, *Information technology – Data cryptographic techniques – Data integrity mechanisms using a cryptographic check function employing a block cipher algorithm.*

[18] T. Johansson, G. Kabatianskii, and B. Smeets, "On the relation between A-codes and codes correcting independent errors," *Proc. Eurocrypt'93, LNCS 765*, Springer-Verlag, 1994, pp. 1–11.

[19] S. Halevi, H. Krawczyk, "MMH: software message authentication in the Gbit/second rates," *Fast Software Encryption 1997, LNCS*, Springer-Verlag, to appear.

[20] L.R. Knudsen, "A new chosen-text attack on CBC-MAC," *Electronics Letters,* Vol. 33 No. 1, January 1997, pp. 48–49.

[21] H. Krawczyk, "LFSR-based hashing and authentication," *Proc. Crypto'94, LNCS 839*, Springer-Verlag, 1994, pp. 129–139.

[22] X. Lai, J.L. Massey, S. Murphy, "Markov ciphers and differential cryptanalysis," *Proc. Eurocrypt'91, LNCS 547*, Springer-Verlag, 1991, pp. 17–38.

[23] M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard," *Proc. Crypto'94, LNCS 839*, Springer-Verlag, 1994, pp. 1–11.

[24] B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Proc. Crypto'95, LNCS 963*, Springer-Verlag, 1995, pp. 1–14.

[25] B. Preneel, P.C. van Oorschot, "On the security of two MAC algorithms," *Proc. Eurocrypt'96, LNCS 1070*, Springer-Verlag, 1996, pp. 19–32.

[26] B. Preneel, P.C. van Oorschot, "A key recovery attack on the ANSI X9.19 retail MAC," *Electronics Letters*, Vol. 32, No. 17, pp. 1568–1569, 1996.

[27] B. Preneel, P.C. van Oorschot, "On the security of iterated message authentication codes," submitted, June 1996.

[28] P. Rogaway, "Bucket hashing and its application to fast message authentication," *Proc. Crypto'95, LNCS 963*, Springer-Verlag, 1995, pp. 29–42.

[29] G.J. Simmons, "A survey of information authentication," in *'Contemporary Cryptology: The Science of Information Integrity,'* G.J. Simmons, Ed., IEEE Press, 1991, pp. 381–419.

[30] O. Staffelbach, W. Meier, "Cryptographic significance of the carry for ciphers based on integer addition," *Proc. Crypto'90, LNCS 537*, Springer-Verlag, 1991, pp. 601–615.

[31] M.N. Wegman, J.L. Carter, "New hash functions and their use in authentication and set equality," *J. Computer Sys. Sciences*, vol. 22, no. 3, pp. 265–279, 1981.

[32] M.J. Wiener, "Efficient DES key search," *Technical Report TR-244*, School of Computer Science, Carleton University, Ottawa, Canada, May 1994. Presented at the rump session of Crypto'93.

A Bit Level Equations for Collisions

This appendix contains the bit equations that are to be solved for a collision cluster key and the corresponding collisions (see §6).

$$
\begin{aligned}
(\epsilon[4] \oplus 1)(Q[4] \oplus Q[3]) &= 0\\
(Q[4] \oplus Q[3])Q[5] \oplus Q[5] \oplus (Q[4] \oplus Q[3])(X[4] \oplus x[4] \oplus \epsilon[4]) \oplus Q[4]Q[3] &= 0\\
Q[11] \oplus \epsilon[11] \oplus (Q[9] \oplus Q[8])(Q[10] \oplus \epsilon[10]) \oplus \epsilon[10] \oplus Q[10](1 \oplus \epsilon[10]) &= 0\\
(Q[9] \oplus Q[8])(1 \oplus \epsilon[9]) &= 0\\
Q[10] \oplus \epsilon[10] \oplus (Q[9] \oplus Q[8])(X[9] \oplus x[9] \oplus \epsilon[9]) \oplus Q[9]Q[8] &= 0\\
Q[12] \oplus \epsilon[11] \oplus Q[11](1 \oplus \epsilon[11]) &= 0\\
((Q[9] \oplus Q[8])(Q[10] \oplus \epsilon[10]) \oplus \epsilon[10])(Q[11] \oplus 1) \oplus \epsilon[11](Q[11] \oplus Q[10](1 \oplus \epsilon[10])) &= 0\\
Q[15] \oplus \epsilon[15] \oplus (Q[13] \oplus Q[12])(Q[14] \oplus \epsilon[14] \oplus 1) \oplus Q[14](1 \oplus \epsilon[14]) &= 0\\
(Q[13] \oplus Q[12])(1 \oplus \epsilon[13]) &= 0\\
Q[14] \oplus \epsilon[14] \oplus (Q[13] \oplus Q[12])(X[13] \oplus x[13] \oplus \epsilon[13]) \oplus Q[13]Q[12] &= 0\\
Q[16] \oplus \epsilon[16] \oplus ((Q[13] \oplus Q[12])(Q[14] \oplus \epsilon[14] \oplus 1) \oplus \epsilon[14])Q[15] &\\
\oplus\epsilon[15](Q[15] \oplus \epsilon[14] \oplus Q[14](1 \oplus \epsilon[14])) \oplus Q[15](1 \oplus \epsilon[15]) &= 0\\
(((Q[13] \oplus Q[12])(Q[14] \oplus \epsilon[14] \oplus 1) \oplus \epsilon[14])Q[13] &\\
\oplus\epsilon[15](Q[15] \oplus \epsilon[14] \oplus Q[14](1 \oplus \epsilon[14])))(Q[16] \oplus \epsilon[16]) \oplus \epsilon[16] &= 0\\
(\epsilon[17] \oplus 1)(Q[17] \oplus Q[16](1 \oplus \epsilon[16])) &= 0\\
Q[18] \oplus \epsilon[18] \oplus (Q[17] \oplus Q[16](1 \oplus \epsilon[16])) &\\
\oplus(Q[17] \oplus Q[16](1 \oplus \epsilon[16]))(X[17] \oplus x[17] \oplus \epsilon[17]) \oplus Q[17]Q[16](1 \oplus \epsilon[16]) &= 0\\
(Q[17] \oplus Q[16](1 \oplus \epsilon[16]))(Q[18] \oplus 1) \oplus \epsilon[18](\epsilon[18] \oplus (Q[17] \oplus Q[16](1 \oplus \epsilon[16]))) &= 0\\
(\epsilon[19] \oplus 1)(Q[19] \oplus \epsilon[18] \oplus Q[18](1 \oplus \epsilon[18])) &= 0\\
Q[20] \oplus \epsilon[20] \oplus (Q[19] \oplus \epsilon[18] \oplus Q[18](1 \oplus \epsilon[18])) &\\
\oplus(Q[19] \oplus \epsilon[18] \oplus Q[18](1 \oplus \epsilon[18]))(X[19] \oplus x[19] \oplus \epsilon[19]) \oplus Q[19](\epsilon[18] \oplus Q[18](1 \oplus \epsilon[18])) &= 0\\
Q[21] \oplus Q[20](1 \oplus \epsilon[20]) &= 0\\
(Q[19] \oplus \epsilon[18] \oplus Q[18](1 \oplus \epsilon[18]))Q[20] \oplus \epsilon[20](Q[20] \oplus (Q[19] \oplus \epsilon[18] \oplus Q[18](1 \oplus \epsilon[18])) &\\
(X[19] \oplus x[19] \oplus \epsilon[19]) \oplus Q[19](\epsilon[18] \oplus Q[18](1 \oplus \epsilon[18]))) &= 0\\
(Q[22] \oplus Q[21])(1 \oplus \epsilon[22]) &= 0\\
Q[23] \oplus \epsilon[23] \oplus (Q[22] \oplus Q[21])(X[22] \oplus x[22] \oplus \epsilon[22]) \oplus Q[22]Q[21] &= 0\\
(\epsilon[22] \oplus 1)(Q[22] \oplus Q[21])(Q[23] \oplus \epsilon[23] \oplus 1) &= 0\\
(Q[22] \oplus Q[21])(Q[23] \oplus \epsilon[23] \oplus 1) \oplus \epsilon[23] &= 0\\
(\epsilon[24] \oplus 1)(Q[24] \oplus \epsilon[23] \oplus Q[23](1 \oplus \epsilon[23])) &= 0\\
(Q[24] \oplus \epsilon[23] \oplus Q[23](1 \oplus \epsilon[23]))(Q[25] \oplus 1) \oplus (Q[25] \oplus (Q[24] \oplus \epsilon[23] \oplus Q[23](1 \oplus \epsilon[23])) &\\
(X[24] \oplus x[24] \oplus \epsilon[24]) \oplus Q[24](\epsilon[23] \oplus Q[23](1 \oplus \epsilon[23]))) &= 0\\
((Q[24] \oplus \epsilon[23] \oplus Q[23](1 \oplus \epsilon[23]))(X[24] \oplus x[24] \oplus \epsilon[24]) &\\
\oplus Q[24](\epsilon[23] \oplus Q[23](1 \oplus \epsilon[23])))Q[25] \oplus Q[26] &= 0\\
(\epsilon[28] \oplus 1)(Q[28] \oplus Q[27]) &= 0\\
(\epsilon[29] \oplus 1)(Q[28] \oplus Q[29]) &= 0\\
Q[30] \oplus \epsilon[30] \oplus (Q[28] \oplus Q[29])(1 \oplus X[29] \oplus x[29] \oplus \epsilon[29]) \oplus Q[28]Q[29] &= 0\\
(Q[28] \oplus Q[29] \oplus 1 \oplus \epsilon[30])Q[30] &\\
\oplus\epsilon[30](Q[30] \oplus (Q[28] \oplus Q[29])(X[29] \oplus x[29] \oplus \epsilon[29]) \oplus Q[29]Q[28]) \oplus Q[31] \oplus \epsilon[31] &= 0
\end{aligned}
$$

$$
\begin{aligned}
Q[1] &= Q[0]\\
Q[2] &= Q[1]\\
Q[3] &= Q[2]\\
Q[6] &= Q[5]
\end{aligned}
$$

$$\begin{aligned}
Q[7] &= Q[6] \\
Q[8] &= Q[7] \\
Q[27] &= Q[26] \\
Q[28] &= Q[27] \\
\epsilon[0] &= 0 \\
\epsilon[5] &= 1 \\
\epsilon[25] &= 1
\end{aligned}$$

$$\begin{aligned}
X[0] \oplus x[0] &= 1 \\
X[5] \oplus x[5] &= 0 \\
X[10] \oplus x[10] &= \epsilon[10] \\
X[11] \oplus x[11] \oplus 1 &= 0 \\
X[14] \oplus x[14] \oplus 1 &= \epsilon[14] \\
X[15] \oplus x[15] &= 0 \\
X[16] \oplus x[16] &= \epsilon[16] \\
X[18] \oplus x[18] \oplus 1 &= 0 \\
X[20] \oplus x[20] &= 0 \\
X[23] \oplus x[23] \oplus 1 &= \epsilon[23] \\
X[25] \oplus x[25] \oplus 1 &= 0 \\
X[30] \oplus x[30] &= 0 \\
X[31] \oplus x[31] &= \epsilon[31]
\end{aligned}$$

$$\begin{aligned}
d[0] &= 0 \\
d[1] &= \epsilon[1] \\
d[2] &= \epsilon[2] \\
d[3] &= \epsilon[3] \\
d[4] &= \epsilon[4] \oplus Q[4] \oplus Q[3] \\
d[5] &= 0 \\
d[6] &= \epsilon[6] \\
d[7] &= \epsilon[7] \\
d[8] &= \epsilon[8] \\
d[9] &= \epsilon[9] \oplus Q[9] \oplus Q[8] \\
d[10] &= 0 \\
d[11] &= 0 \\
d[12] &= \epsilon[12] \\
d[13] &= \epsilon[13] \oplus Q[13] \oplus Q[12] \\
d[14] &= 0 \\
d[15] &= 0 \\
d[16] &= 0 \\
d[17] &= \epsilon[17] \oplus Q[17] \oplus Q[16](1 \oplus \epsilon[16]) \\
d[18] &= 0 \\
d[19] &= \epsilon[19] \oplus Q[19] \oplus \epsilon[18] \oplus Q[18](1 \oplus \epsilon[18]) \\
d[20] &= 0 \\
d[21] &= \epsilon[21] \\
d[22] &= \epsilon[22] \oplus Q[22] \oplus Q[21] \\
d[23] &= 0 \\
d[24] &= \epsilon[24] \oplus Q[24] \oplus \epsilon[23] \oplus Q[23](1 \oplus \epsilon[23])
\end{aligned}$$

$$
\begin{aligned}
d[25] &= 0\\
d[26] &= \epsilon[26]\\
d[27] &= \epsilon[27]\\
d[28] &= \epsilon[28] \oplus Q[28] \oplus Q[27]\\
d[29] &= \epsilon[29] \oplus Q[29] \oplus Q[28]\\
d[30] &= 0\\
d[31] &= 0
\end{aligned}
$$

B Prelude of MAA

The prelude derives X_0, Y_0, V_0, W, S and T from the key words J_1 and J_2, using the following operations: exor ('$\oplus$'), multiplication modulo $2^{32}-1$ ('$\otimes_1$'), multiplication modulo $2^{32}-2$ ('$\otimes_2$') and the BYT procedure.

B.1 The BYT Procedure

The BYT procedure takes two 32-bit words as input. The two words are regarded as eight consecutive bytes

$$\{X, Y\} = \{B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7\}.$$

This conversion is done big endian, i.e.,

$$X = B_0 \cdot 2^{24} + B_1 \cdot 2^{16} + B_2 \cdot 2^8 + B_3.$$

The output consists of the updated versions of the eight input bytes and one extra byte P. The algorithm goes as follows.

```
P = 0;
for i = 0 to 7 do {
        P = 2 × P;
        if (B_i == 0) then {
                P = P + 1;
                B_i = P;
                }
        else if (B_i == 255) then {
                P = P + 1;
                B_i = 255 − P;
                }
        }
```

B.2 The Prelude

The prelude starts with an application of BYT.

$$
\begin{aligned}
\{J_1, J_2, P\} &= \text{BYT}\{J_1, J_2\}\\
Q &= (1+P) \times (1+P)
\end{aligned}
$$

X_0, V_0 and S are derived from J_1.

$$\begin{array}{lcl} J_1^2 & = & J_1 \otimes_1 J_1 \\ J_1^4 & = & J_1^2 \otimes_1 J_1^2 \\ J_1^6 & = & J_1^2 \otimes_1 J_1^4 \\ J_1^8 & = & J_1^2 \otimes_1 J_1^6 \end{array} \qquad \begin{array}{lcl} J_1^{2*} & = & J_1 \otimes_2 J_1 \\ J_1^{4*} & = & J_1^{2*} \otimes_2 J_1^{2*} \\ J_1^{6*} & = & J_1^{2*} \otimes_2 J_1^{4*} \\ J_1^{8*} & = & J_1^{2*} \otimes_1 J_1^{6*} \end{array}$$

$$\begin{array}{lcl} X_0 & = & J_1^4 \oplus J_1^{4*} \\ V_0 & = & J_1^6 \oplus J_1^{6*} \\ S & = & J_1^8 \oplus J_1^{8*} \end{array}$$

Y_0, W and T are derived from J_2, Y_0 also depends on Q.

$$\begin{array}{lcl} J_2^2 & = & J_2 \otimes_1 J_2 \\ J_2^4 & = & J_2^2 \otimes_1 J_2^2 \\ J_2^5 & = & J_2 \otimes_1 J_2^4 \\ J_2^7 & = & J_2^2 \otimes_1 J_2^5 \\ J_2^9 & = & J_2^2 \otimes_1 J_2^7 \end{array} \qquad \begin{array}{lcl} J_2^{2*} & = & J_2 \otimes_2 J_2 \\ J_2^{4*} & = & J_2^{2*} \otimes_2 J_2^{2*} \\ J_2^{5*} & = & J_2 \otimes_2 J_2^{4*} \\ J_2^{7*} & = & J_2^{2*} \otimes_2 J_2^{5*} \\ J_2^{9*} & = & J_2^{2*} \otimes_2 J_2^{7*} \end{array}$$

$$\begin{array}{lcl} Y_0 & = & (J_2^5 \oplus J_2^{5*}) \otimes_2 Q \\ W & = & J_2^7 \oplus J_2^{7*} \\ T & = & J_2^9 \oplus J_2^{9*} \end{array}$$

The prelude concludes with three applications of BYT.

$$\begin{array}{lcl} \{X_0, Y_0, P\} & = & \text{BYT}\{X_0, Y_0\} \\ \{V_0, W, P\} & = & \text{BYT}\{V_0, W\} \\ \{S, T, P\} & = & \text{BYT}\{S, T\} \end{array}$$