On the Security of Two MAC Algorithms

Bart Preneel[1]* Paul C. van Oorschot[2]

[1] Katholieke Universiteit Leuven, Dept. Electrical Engineering-ESAT
Kardinaal Mercierlaan 94, B–3001 Heverlee, Belgium
bart.preneel@esat.kuleuven.ac.be

[2] Bell-Northern Research, Box 3511 Station C
Ottawa, Ontario K1Y 4H7, Canada
paulv@bnr.ca

Abstract. The security of two message authentication code (MAC) algorithms is considered: the MD5-based envelope method (RFC 1828), and the banking standard MAA (ISO 8731–2). Customization of a general MAC forgery attack allows improvements in both cases. For the envelope method, the forgery attack is extended to allow key recovery; for example, a 128-bit key can be recovered using 2^{67} known text-MAC pairs and time plus 2^{13} chosen texts. For MAA, internal collisions are found with fewer and shorter messages than previously by exploiting the algorithm's internal structure; consequently, the number of chosen texts (each 256 Kbyte long) for a forgery can be reduced by two orders of magnitude, e.g. from 2^{24} to 2^{17}. This attack can be extended to one requiring only short messages (2^{24} messages shorter than 1 Kbyte) to circumvent the special MAA mode for long messages. Moreover, certain internal collisions allow key recovery, and weak keys for MAA are identified.

1 Introduction

Message authentication code (MAC) algorithms are symmetric-key techniques which provide data origin authentication and data integrity. They have received widespread use in many practical applications, e.g. banking [9]. The primary MAC algorithms used historically have been CBC-MAC and MAA. CBC-MAC [10, 11] is derived from the cipher-block chaining (CBC) mode of block ciphers such as DES; some theoretical support for this method has been given [1]. The Message Authenticator Algorithm (MAA) is an ISO standard [10] which dates back to 1984 [7]. The so-called envelope-based MACs of Tsudik [18] and others have also received recent attention (see e.g. Kaliski and Robshaw [12]). Recently several new practical MAC algorithms were proposed: XOR-MAC by Bellare et al. [2], HMAC by Bellare et al. [4], MDx-MAC by Preneel and van Oorschot [14], and the bucket-hashing MAC of Rogaway [16].

* N.F.W.O. postdoctoral researcher, sponsored by the National Fund for Scientific Research (Belgium).

Envelope-based MACs offer speed and simplicity. The basic technique involves using a secret key as part of the input to an unkeyed hash function. The envelope method of RFC 1828 [12, 17], arising from the IPSEC working group of the Internet Engineering Task Force (IETF), prepends and appends a secret key K to the message input: $\mathrm{MAC}(x) = h(K\|p\|x\|K)$. Here $\|$ denotes concatenation, and p denotes some padding bits. This construction was supported by a security proof under assumptions regarding the pseudo-randomness of MD5 [3]. An alternative HMAC [4], involving two invocations of $h(\cdot)$, is defined as $\mathrm{MAC}(x) = h(K\|p_1\|h(K\|p_2\|x))$, where p_1 and p_2 are strings of padding bits which pad K out to a full block; a version without padding was proposed earlier in [12]. The security of HMAC can be proven based on the assumptions that $h(\cdot)$ is collision resistant for a random and secret IV, and that the complete output of the compression function is hard to predict when its first input is random and secret.

Recent systematic analysis of MACs includes a new general attack by Preneel and van Oorschot [14] which applies to all iterated MACs. It involves a birthday attack using known text-MAC pairs, after which additional chosen text-MAC pairs (e.g. a single text for the most common version of CBC-MAC) allows MAC forgery. Overall this requires about $2^{n/2}$ known text-MAC pairs, where n is the bitlength of the internal memory (chaining variable) of the MAC algorithm.

The current paper adapts and refines this attack schema specifically for the envelope method (e.g. RFC 1828) and for MAA, yielding significant improvements and new results in each case. The sequel is organized as follows. §2 reviews definitions and the general attack on iterated MACs. §3 presents a new key recovery attack on the envelope method, while §4 gives an optimized forgery attack plus a new key recovery attack on MAA. §5 concludes the paper.

2 Background Definitions and Review

A hash function h map bitstrings of arbitrary finite length into strings of fixed length (say m bits). An *unkeyed hash function* does not involve secret parameters. Such a function is said to be *one-way* if it is both *preimage-resistant* (it is computationally infeasible to find any input which hashes to any pre-specified output); and *second-preimage resistant* (it is computationally infeasible to find any second input which has the same output as any specified input). Ideally, finding a preimage or second preimage requires about 2^m operations. A one-way hash function is said to be *collision resistant* if it is furthermore computationally infeasible to find a collision (i.e. two distinct inputs that hash to the same result). For ideal such functions, no collisions may be found more efficiently than by a birthday-like attack of about $2^{m/2}$ operations.

A *keyed* hash function h has a secret k-bit key K as a secondary input; when used for message authentication, such a function is called a message authentication code (MAC). Computing $h_K(x)$ (denoted simply $h(x)$ when K is understood) must be easy, given h, K, and an input x. An adversary able, without initial knowledge of K, to find a corresponding MAC for any single message, is said to be capable of *existential forgery*. An adversary able to determine the MAC for a message of his choice 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. A *key recovery* attack is more devastating than forgery – here an adversary is able to recover K itself, and thus carry out arbitrary selective forgeries. Ideally, any attack allowing key recovery requires about 2^k operations. Verification of such an attack requires k/m text-MAC pairs.

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.

Iterative hash functions and MACs h 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. h involves a *compression function* f and an n-bit ($n \geq m$) *chaining variable* H_i between stage $i-1$ and stage i: $H_0 = IV$; $H_i = f(H_{i-1}, x_i), 1 \leq i \leq t$; $h(x) = H_t$.

MACs often involve an output transformation g applied to H_t, yielding a MAC result $h(x) = g(H_t)$. The secret key may be employed in the IV, in f, and/or in g. For an input pair (x, x') with $h(x) = g(H_t)$ and $h(x') = g(H'_t)$, a collision $h(x) = h(x')$ is 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 [14] is applicable to all iterated MACs. 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 cited for convenience.

Lemma 1 [14]. *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'), $h(x \,\|\, y) = h(x' \,\|\, y)$ for any single block y; thus a requested MAC on the chosen text $x \,\|\, y$ provides a forged MAC (the same) for $x' \,\|\, y$. The general forgery attack (Proposition 2) depends on the nature of the compression function f. In MAA and in envelope methods based on MD5 [15], the compression function f is considered to behave as a random mapping for fixed x_i. The parameters of the original attack [14] may be improved slightly, as stated in Proposition 2.

Proposition 2. *Let h 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 h 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*

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

∎

3 New Key Recovery Attack on the Envelope Method

The *envelope method* as proposed by Tsudik [18] consists of respectively prepending and appending secret keys K_1 and K_2 to the message input: $\mathrm{MAC}(x) = h(K_1\|x\|K_2)$. An Internet proposed standard recommended by the IP Security (IPSEC) working group for authentication of IP datagrams, namely RFC 1828 [17], specifies a variant of this using MD5 and a single key K: $\mathrm{MAC}(x) = h(K\|p\|x\|K)$. Here p denotes some padding bits chosen such that $K\|p$ fills the first block, and allows for a security proof assuming pseudo-randomness of MD5 [3]. RFC 1828 allows a variable length key, and mandates support for bitlengths up to 128 bits. An important consideration motivating use of envelope MACs is that they require minimal implementation and deployment effort: code for the underlying unkeyed hash function can be used without modification.

A divide and conquer key recovery attack on the envelope method with distinct keys $K_1 \neq K_2$ was given by Preneel and van Oorschot [14]. Rather than an exhaustive search over $k_1 + k_2$ bits (where $k_i = |K_i|$ and $k_1 = n$ is the bitlength of the chaining variable), first K_1 and then K_2 are found. For an MD5-based MAC, the attack uses Proposition 2 with about 2^{64} known text-MAC pairs (assume $s = 0$ for simplicity) to find an internal collision. An off-line exhaustive search involving 2^{k_1} operations results in a small set of possible keys K_1 from which the correct key can be determined with a few chosen texts, reducing security to an appended secret key alone. K_2 is then determined by off-line exhaustive search. $K_1 \neq K_2$ thus offers substantially less additional security than one would hope, relative to k_1 bits of security for $K_1 = K_2$, although the former nonetheless requires an extremely large number of known text-MAC pairs. In any case, for $k_1 = 128$, a search requiring 2^{k_1} operations is completely infeasible.

We describe now a new divide and conquer key recovery attack. It applies to the method of RFC 1828 (also proposed in [12]), and exploits the padding procedure of MD5, which was not designed to conceal secret keys. This attack again requires a very large number of known text-MAC pairs (variable depending on choices made, but on the order of 2^{64}); however, the work complexity for key recovery is decreased dramatically from previous numbers (from 2^{128} to 2^{66}).

The new key recovery attack is applicable to the basic envelope method including the MD5-based RFC 1828 (IPSEC) variant and other hash functions using MD5-like padding. It can also recover the trail key of the variation with distinct keys. First recall the padding procedure for MD5 for a message input y of bitlength $b = |y|$. A single '1' bit is appended to y, followed by z '0' bits ($0 \leq z \leq 511$), where z is chosen to make the sum of b and the bitlength of the padding equal $448 \bmod 512$. The 64-bit integer representation of b is then appended to complete the last block. For the special case of the IPSEC envelope method with a 128-bit key, the data, after padding, processed by the compression function of MD5 has the form: $K||p||x||K||1000\ldots000||b$. Here x is the message on which a MAC is desired, $y = K||p||x||K$, and $b = 512 + 128 + |x|$. Defining $r = |x| \bmod 512$,

$$z = \begin{cases} 319 - r & \text{if} \quad 0 \leq r \leq 319 \\ 831 - r & \text{if} \ \ 320 \leq r \leq 511 \end{cases}$$

If $z \in [0, 319]$, the key K will lie completely in the last block, and the number of message bits in the last block is r. For $z \in [320, 446]$, $z - 319$ bits of the key K will be in the second last block, with the remaining key bits in the last block. For $z \in [447, 511]$, K falls completely in the second last block. In the latter two cases, there will be r message bits in the second last block (see Figure 1).

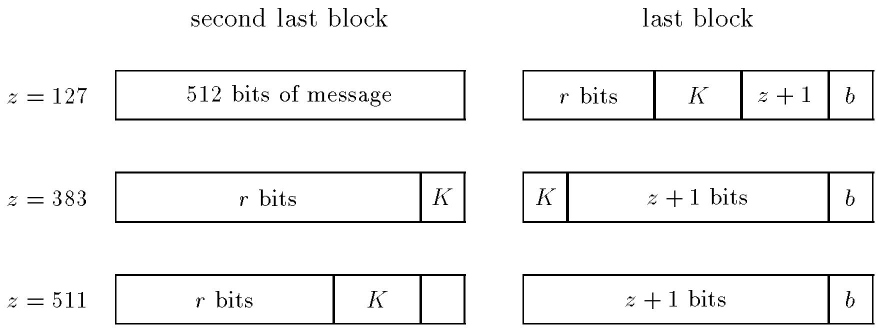

Fig. 1. Message, key, padding, and length fields in final blocks of envelope method.

Define an internal collision as a pair of inputs (x, x') which produce the same MAC output, and for which the internal chaining variables collide just before the block containing the key (or any partial key). Since such a collision is detectable only through a collision for the MAC, all blocks following the internal collision must be identical in the two members of the colliding input pair. Therefore the attack of Proposition 2 requires the lengths of all the messages to be equal, and the last r message bits (which are either in the last or in the second last block) to be the same. If $r = 0$ (i.e. $|x| = 0 \bmod 512$), there is no condition on the last message bits.

Consider the case $r = 511$ (i.e. $z = 320$). There is a single key bit in the second last block. Therefore 511 message bits in the second last block must be identical to allow for identification of an internal collision. However, if we simply guess that key bit, the unknown key is restricted to the last block, and collisions after the second last block are again internal collisions (or *almost internal* collisions). A first observation is that this reduces the constraint on the message. A more significant consequence is that by using the attack of Lemma 1, one can actually verify the guess for that key bit. This leads to a powerful divide and conquer attack for extracting the key, which may be illustrated as follows.

Let x be a 480-bit message. Then r=480, z=351, and the first block contains the padded key K. The second block contains 480 message bits and 32 key bits. The last block contains the 96 remaining key bits, a '1' bit followed by 351 '0' bits, and the 64-bit length field b=1120 (i.e. $512 + 128 + 480$). If the MACs for about $2^{64.5}$ such messages x are known, one may expect (by Proposition 2 with n=128=m, s=0) about two collisions: one after the second block (an almost internal collision), and one after the last block (an external collision). Denote the almost internal colliding pair (x, x'). Construct 2^{32} message pairs of the form $(x\|k_i\|y,\ x'\|k_i\|y)$, where k_i is 32 bits and ranges over all 2^{32} possible values, and y is now an arbitrary block. Request the 2^{33} corresponding MACs. When k_i takes on the value of the correct partial key, the two MACs agree; moreover, with probability $\approx 1 - 1/2^{96}$, no other pairs of MACs will be equal. This reveals 32 key bits. For the external collision, with overwhelming probability none of the pairs gives the same MAC.

It is easy to extend the attack to find further key bits. One possibility is to repeat the above procedure using messages of length 448 bits, yielding the next 32 key bits. The remaining 64 key bits are then most efficiently found (off-line) exhaustively. Alternatively, one could begin with messages of bitlength 448, which would require 2^{66} chosen texts, but reveal 64 bits of the key immediately. This reasoning allows the following general result:

Proposition 3. *There exists a key recovery attack on one-key envelope methods such as that of RFC 1828, which requires $q = \lceil 64/t \rceil$ steps ($1 \le t \le 64$) to find 64 bits of the key. Step i ($1 \le i \le q$) requires $\sqrt{2} \cdot 2^{64}$ known texts of bitlength $c_i \cdot 512 - t \cdot i$ for some fixed $c_i > 1$, and 2^{t+2} chosen texts.* ■

Table 1 summarizes the complexity to find 64 key bits in t-bit slices, for different values of t. If a 128-bit key is used with the remaining bits found by exhaustive search, the overall time complexity is on the order of the number of known texts. The attack is easily modified for keys exceeding 128 bits; e.g. recovering a 256-bit key in three 64-bit slices requires about 2^{66} known text-MAC pairs and the same order of chosen texts. Thus relative to this attack, this MAC design makes very poor use of key bits beyond 128. For context, recall that linear cryptanalysis of DES [13], viewed as a tremendous breakthrough, requires 2^{43} known texts against a 56-bit key, while differential cryptanalysis requires 2^{47} chosen texts [5]. The above key recovery attack, relative to its larger 128-bit key, requires substantially fewer known texts (and time); this indicates that the general construction fails to make good use of key bits.

The above attack requires that $|x| \bmod 512 \in [448, 511]$, because the number of bits of K in the penultimate block must be between 1 and 64 (the attack becomes less feasible if more than 64 bits need be guessed); and that the known texts have the same number of blocks, because the value of b must be the same for the two messages forming the internal collision. However, if a set of about 2^{73} "short" (say ten or fewer blocks) known messages is available, we expect to find

Table 1. Complexity of key recovery attack on envelope method (128-bit key)

t	# known texts	# chosen texts
4	$2^{68.5}$	2^{10}
8	$2^{67.5}$	2^{13}
16	$2^{66.5}$	2^{20}
32	$2^{65.5}$	2^{35}
64	$2^{64.5}$	2^{66}

among those a sufficient number of messages suitable for the attack (without fixing t in advance); the attack will still require a much smaller number (less than 2^{20}) of chosen texts to identify the key bits.

The attack relies on the key being split across blocks. While it is not practical, vulnerability to it represents a certificational weakness, and indicating an architectural flaw. One concludes it is more secure to isolate the entire trailing key in a separate block (together with the message length and possibly a pseudo-random string). However, this requires changing the padding procedure for MD5, contravening an original motivating factor – being able to call the underlying hash function directly. Nonetheless, customized MACs (as suggested in [12, 14]) appear to offer a more secure alternative to constructions relying directly on unkeyed hash functions. Note also that no one has yet evaluated functions such as MD5 with respect to pseudo-randomness properties (especially if only part of the input is replaced by a secret key).

4 New Forgery and Key Recovery Attacks on MAA

In this section, a preliminary description of MAA is followed by a discussion of how the basic attack of Proposition 2 may be customized for MAA, and optimized using special messages. An extension of these to the special mode of MAA for long messages is then presented, followed by a new key recovery attack.

4.1 Description of MAA

Designed by Davies, MAA was presented at Crypto'84 [7, 8] and is in the ISO 8731-2 banking standard [10]. On PCs and workstations it runs only 40% slower than MD5. The algorithm consists of three parts. The *prelude* is a key expansion from 64 bits (two 32-bit words) to 192 bits (six 32-bit words). From the first key word are derived the parameters $H1_0$, V, S; from the second, the parameters $H2_0$, 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. The two words of the chaining variable $(H1_i, H2_i)$ are initialized with $(H1_0, H2_0)$. 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 [10]), 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: $H_{t+2} = H1_{t+2} \oplus H2_{t+2}$.

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

Step 1: $V := \mathrm{rol}(V)$; $K_i := V \oplus W$; % rol denotes 1 bit cyclic shift left
Step 2: $t_1 := H1_{i-1} \oplus x_i$; $t_2 := H2_{i-1} \oplus x_i$;
$H1_i := t_1 \otimes_1 (((K_i \boxplus t_2) \vee A) \wedge C)$; $H2_i := t_2 \otimes_2 (((K_i \boxplus t_1) \vee B) \wedge D)$;

Here $A = \mathtt{02040801_x}$, $B = \mathtt{00804021_x}$, $C = \mathtt{BFEF7FDF_x}$, and $D = \mathtt{7DFEFBFF_x}$. These constants fix 8 bits of the second factor (four to 0, and four to 1). The output transformation g consists of the coda iterations (where the key-dependent S and T play the role of x_i) and final XORing as noted above.

4.2 Basic MAC Forgery on MAA

The basic attack of Proposition 2 can be applied to MAA with parameters $n = 64$, $m = 32$. However, an internal collision (i.e. a collision for $H1$ and $H2$) yields an external collision only if V is in the same 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 2 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; we expect 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 $H2_i$, denoted $\mathrm{LSB}(H2_i)$, is equal to the $\mathrm{LSB}(H2_{i-1} \oplus x_i)$. Consequently, $\mathrm{LSB}(H2_t) = LSB(H2_0) \oplus \bigoplus_{i=1}^{t} \mathrm{LSB}(x_i)$, 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 2 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 2. 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	$\geq 2^{34}+4$	$2^{50.0}$

4.3 Optimized MAC Forgery using Special Messages

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 $H1_i$, 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, such a factor would remain nonetheless. If p is a prime factor of $2^{32}-1$, the probability that i is the smallest index for which $H1_{i_0+i}$ is divisible by p is given by $\frac{1}{p}(1-\frac{1}{p})^i$, a geometric distribution with expected value $i = p-1$ for success (note this assumes that the $H1_i$'s are uniformly distributed; this assumption has been confirmed by some computer experiments). Thus, after about 2^{16} iterations (each with $x_i = 0$), $H1_i$ 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 $\otimes_1$). It is easy to prove the following:

Lemma 4. *Let p be a prime divisor of $2^{32}-1$. If $i = \alpha(p-1)$ zero blocks are inserted, starting with x_{i_0}, the probability that p divides $H1_{i_0+i}$ is $1-e^{-\alpha}$.* ∎

Once $H1_i$ is constant, finding a collision for $H2_i$ yields an internal collision. Consider the second chain. For $H1_i = \texttt{FFFFFFFF}_\texttt{x}$ and $x_i = \texttt{00000000}_\texttt{x}$, $H2_i = H2_{i-1} \otimes_2 U_i$, where $U_i = ((\texttt{FFFFFFFF}_\texttt{x} \boxplus K_i) \vee B) \wedge D$. 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 mod $2^{32}-2$). Then note $H2_{i+32} = H2_{i-1} \otimes_2 \tilde{U}$. 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 we have that $\tilde{U}^{2^{31}-2} \bmod (2^{32}-2) = 1$. 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 $H1_i = \texttt{FFFFFFFF}_\texttt{x}$. 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. This existential forgery, while not likely of practical use (the message ends in about 2^{38} zero bytes), illustrates a certificational weakness of MAA.

Proposition 5. *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.* ∎

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 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.[1] 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 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 (§4.5) to detect whether a key is weak, leading to a key recovery attack on these keys. *These observations suggest that the fact that V depends 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. The order of $\tilde{U}$ must divide D, where 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 D. More specifically, $D = \phi(M) = \phi(2) \cdot \phi(2^{31} - 1) = 2^{31} - 2 = 2 \cdot 3^2 \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331$, where $\phi()$ is Euler's totient function. For each divisor d of 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 $d = D/331 = 6\,487\,866$, 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 2) by exploiting the properties of zero blocks presented in Lemma 4. This requires chosen texts rather than known texts. As discussed above, for a text with a sufficiently large number of trailing zero blocks, $H1_t$ 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 to find two texts with colliding values for $H2_t$ as well. The third observation of §4.2 implies that 2^{16} such texts will suffice if the sum of the least significant bits of the message blocks is kept constant. We will assume 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.

The exact number of chosen texts for this improved variant can be computed as follows. Lemma 4 (with $p = 2^{16} + 1$) implies that among $r = 2^{16}/(1 - e^{-\alpha})$ messages each containing $\alpha 2^{16}$ trailing zero blocks for some α, we expect to find about 2^{16} messages for which the first chaining variable $H1_t$ becomes equal to $\mathtt{FFFFFFFF_x}$ (for simplicity it is assumed that $\alpha \geq 1/32$, since otherwise the second prime factor ($p = 257$) has to be taken into account in the calculation). By the birthday paradox, the expected number of collisions for the second chaining variable $H2_t$, which corresponds to a (complete) internal collision is then 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 (see [14]) the

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

attack of Lemma 1. Two additional chosen texts are sufficient to identify an external collision with high probability; the expected value is about $2(1-1/e)$ as will be shown in the full paper. The total number of blocks in the chosen texts is then approximately

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

If $\alpha \rightarrow 0$, 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 (2) 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 we assume that $H1_t$ 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 second prime factor (257).

An overview of the trade-offs is given in Table 3. These trade-offs are more realistic (cf. Table 2) 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 3. Parameters for improved forgery attack on MAA. Attacks with <256 zero blocks avoid the 'long message mode' (see §4.4).

	α	# 0 blocks	# chosen texts	total size (bytes)
$H1_t =$ $\texttt{FFFFFFFF}_\texttt{x}$	1/32	2 048	$2^{21.0}$	$2^{34.0}$
	1/4	16 384	$2^{18.2}$	$2^{34.2}$
	1/2	32 768	$2^{17.3}$	$2^{34.3}$
	1	65 636	$2^{16.7}$	$2^{34.7}$
	2	131 072	$2^{16.2}$	$2^{35.2}$
$H1_t =$ multiple of $\texttt{FFFF}_\texttt{x}$	1/4	64	$2^{26.2}$	$2^{34.3}$
	1/2	128	$2^{25.3}$	$2^{34.4}$
	1	256	$2^{24.7}$	$2^{34.7}$
	2	512	$2^{24.2}$	$2^{35.2}$

4.4 Long Message MAC Forgery on MAA

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 [6], and mentioned by Davies [7]. 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 [10] limits the size of the messages to $4 \cdot 10^6$ bytes ($\approx$ 3.8 Megabytes). 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 concatenated 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. This 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 5 which requires a single chosen text.

However, it follows from Table 3 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 2) of §4.2 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 $s + \lfloor s/256 \rfloor$.

4.5 Key Recovery Attack on MAA

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. Under certain circumstances an internal collision for MAA allows key recovery.

A first case is when we have a weak key for which the period of $V < 32$ (see §4.3). 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. We partially summarize these observations as follows.

Proposition 6. *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. The expected number of internal collisions is then Poisson distributed with $\lambda = 1/2$, with the probability that there are two internal collisions is given by $e^{-1/2}/4 = 0.152$. The unknowns affecting the internal collision are the parameters $(H1_0, V)$, and $(H2_0, W)$; see §4.1. A single internal collision ($H1_1 = H1'_1$, $H2_1 = H2'_1$) gives 64 bits of information about these parameters. Therefore two internal collisions yield enough information to find a solution.

We have developed an algorithm which can then solve for the 64 bits of the key inputs (via the 128 bits of the unknown parameters), starting from the least significant bit (to control the propagation of the carries). Also, the algorithm exploits the fact that $(H1_0, V)$ and $(H2_0, W)$ depend on different halves of the input key. A preliminary estimate is that the key can be recovered in at most 2^{50} operations (cf. 2^{64} for exhaustive key search). The expected number of chosen texts required is about 2^{35} to guarantee a high success probability. More details will be provided in the full paper.

5 Concluding Remarks

Two improved variations of a general MAC forgery attack have been presented, tailored for specific algorithms. For both the envelope method as discussed herein and the MAA, the forgery attack may be adapted to allow key recovery. In addition the internal structure of MAA may be exploited to reduce the total number of bytes of known text-MAC pairs required for the attack. An important conclusion is that the standard padding of a hash function should be modified when it is transformed into a MAC. This should not be surprising, since unkeyed hash functions are not typically designed for use as MACs.

Acknowledgements

We would like to thank Vincent Rijmen for help on the MAA attack.

References

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

2. 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.
3. 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/.
4. M. Bellare, R. Canetti, H. Krawczyk, "Keying hash functions for message authentication," 25 January 1996, http:// www.research.ibm.com/security/.
5. E. Biham, A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard,* Springer-Verlag, 1993.
6. H. Block, "File authentication: A rule for constructing algorithms," *SÄKdata Report*, October 12, 1983.
7. D. Davies, "A message authenticator algorithm suitable for a mainframe computer," *Proc. Crypto'84, LNCS 196*, Springer-Verlag, 1985, pp. 393–400.
8. D. Davies, D.O. Clayden, "The message authenticator algorithm (MAA) and its implementation," *NPL Report DITC 109/88*, Feb. 1988.
9. D. Davies, W. Price, *Security for Computer Networks*, 2nd ed., Wiley, 1989.
10. ISO 8731:1987, *Banking – approved algorithms for message authentication, Part 1, DEA,* IS 8731-1, *Part 2, Message Authentication Algorithm (MAA),* IS 8731-2.
11. ISO/IEC 9797:1993, *Information technology – Data cryptographic techniques – Data integrity mechanisms using a cryptographic check function employing a block cipher algorithm.*
12. B. Kaliski, M. Robshaw, "Message authentication with MD5," *CryptoBytes (RSA Laboratories Technical Newsletter)*, Vol. 1, No. 1, Spring 1995, pp. 5–8.
13. M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard," *Proc. Crypto'94, LNCS 839*, Springer-Verlag, 1994, pp. 1–11.
14. 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.
15. R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC) 1321*, Internet Activities Board, Internet Privacy Task Force, April 1992.
16. P. Rogaway, "Bucket hashing and its application to fast message authentication", *Proc. Crypto'95, LNCS 963*, Springer-Verlag, 1995, pp. 29–42.
17. P. Metzger, W. Simpson, "IP Authentication using Keyed MD5", Internet Request for Comments 1828, August 1995.
18. G. Tsudik, "Message authentication with one-way hash functions," *ACM Computer Communications Review*, Vol. 22, No. 5, 1992, pp. 29–38.