MDx-MAC and Building Fast MACs from Hash Functions*

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, P.O. Box 3511 Station C, Ottawa, Ontario, K1Y 4H7, Canada
paulv@bnr.ca

Abstract. We consider the security of message authentication code (MAC) algorithms, and the construction of MACs from fast hash functions. A new forgery attack applicable to all iterated MAC algorithms is described, the first known such attack requiring fewer operations than exhaustive key search. Existing methods for constructing MACs from hash functions, including the secret prefix, secret suffix, and envelope methods, are shown to be unsatisfactory. Motivated by the absence of a secure, fast MAC algorithm not based on encryption, a new generic construction (*MDx*-MAC) is proposed for transforming any secure hash function of the MD4-family into a secure MAC of equal or smaller bitlength and comparable speed.

1 Introduction

Hash functions play a fundamental role in modern cryptography. One main application is their use in conjunction with digital signature schemes; another is in conventional techniques for message authentication. In the latter, it is preferable that a hash function take as a distinct secondary input a secret key. Such hash functions, commonly known as *message authentication codes* (MACs), have received widespread use in practice for data integrity and data origin authentication, e.g. in banking applications (see [7, 17]).

Compared to the extensive work on the design and analysis of hash functions, little attention has been given to the design of efficient MACs [22] (although see [2]). One apparent reason is that the first proposals for MAC algorithms were quickly turned into standards and proved adequate in practice. The first constructions are based on the Cipher Block Chaining (CBC) and Cipher FeedBack (CFB) modes of a block cipher [14, 15]. Most standards and applications use the CBC mode (CBC-MAC); theoretical support for this construction was given recently in [1]. Another proposal dating back to 1984 is the Message Authenticator Algorithm (MAA) [5, 6, 14], for which no significant weaknesses have

* To appear in Proceedings Crypto'95, Springer-Verlag LNCS, August 1995.
** N.F.W.O. postdoctoral researcher, sponsored by the National Fund for Scientific Research (Belgium).

previously been identified. MAA is a current ISO standard, and is relatively fast in software (about 40% slower than MD5). Its main disadvantage is that the result, being 32 bits, is considered unacceptably short for many applications. Recent research on authentication codes has resulted in very fast, scalable, and information theoretically secure constructions [16, 19, 28], which require relatively short keys. The disadvantage is that a different key must be used for every message. If this is not acceptable, one can generate the key using a cryptographically strong pseudo-random string generator, but the resulting scheme is then (at most) computationally secure.

Around 1991, Rivest proposed two very fast hash functions, namely MD4 [24] and MD5 [25]. Later RIPEMD [23] and SHA [12] were introduced by other research groups. In software, these hash functions may be as much as one order of magnitude faster than DES. Several factors have motivated their adoption as the basis for MAC algorithms: system designers quickly realized that MACs based on these outperform other available options; the additional implementation effort required to use these as MACs is very small; and the fact that such MACs do not involve encryption algorithms has favorable export implications. Because of these factors, MAC constructions based on these hash functions were adopted in Kerberos [20] and SNMP [13] and are proposed for IPSEC [18].

In this paper, a new general attack is proposed which applies to all iterated MACs, including MAA and CBC-MAC. It is a birthday attack on known text-MAC pairs, which with a few additional chosen text-MAC pairs, allows MAC forgery. An extension of the attack is also given. The best previous general attack on MAC algorithms was an exhaustive search for the key. The new attack requires a number of known text-MAC pairs which is $O(2^{n/2})$, where n is the bitlength of the internal memory (chaining variable) of the MAC algorithm.

We then analyze three proposals for MAC algorithms based on hash functions: prepending a secret key to the message input of the hash function (secret prefix method), appending a secret key to the input (secret suffix method), and combining both operations (envelope method). For the secret prefix and secret suffix method, a systematic analysis is given which generalizes the known attacks. For the envelope method, the new general attack applies and illustrates that the proof outline in [26] for the security of this method is incorrect. Our conclusion is that these approaches do not achieve the security level suggested by the size of the parameters. Moreover, some variants of these methods are susceptible to more serious attacks.

In addition to the concerns this raises about these constructions for MAC algorithms from hash functions, the new attack calls into question the strength of MAA and CBC-MAC. This creates a need for a new fast MAC algorithm offering security substantially better than those which succumb to attacks requiring on the order of 2^{32} known or chosen text-MAC pairs.

Motivated by the above reasons, and the lack of an acceptable, non-encryption based fast MAC algorithm providing more than 32-bit results, we propose a new generic construction (MDx-MAC), suitable for application to MD4-family hash functions including MD5, RIPEMD, and SHA, and yielding comparable throughput. The security of the proposed construction is also examined.

The remainder of this paper is organized as follows. §2 reviews the definitions of a hash function and a MAC. §3 discusses the new general attack on MACs. §4 analyzes three previous proposals for constructing a MAC based on a secure hash function. §5 describes and examines the new MAC construction. §6 concludes the paper.

2 Definitions and Background

Hash functions are functions that map bitstrings of arbitrary finite length into strings of fixed length. Given h and an input x, computing $h(x)$ must be easy. First we give definitions of hash functions which do not involve secret parameters. A *one-way hash function* must satisfy the following properties:

– **preimage resistance**: it is computationally infeasible to find any input which hashes to any pre-specified output.
– **second preimage resistance**: it is computationally infeasible to find any second input which has the same output as any specified input.

For an *ideal* one-way hash function with an m-bit result, finding a preimage or a second preimage requires $O(2^m)$ operations. A *collision resistant hash function* is a one-way hash function that satisfies an additional condition:

– **collision resistance**: it is computationally infeasible to find a collision, i.e. two distinct inputs that hash to the same result.

For an *ideal* collision resistant hash function with an m-bit result, no attack finding a collision betters a birthday or square root attack of $O(2^{m/2})$ operations.

A MAC is a hash function with a secondary input, the secret key K. Given h, an input x, and the secret key K, computing $h(x)$ must be easy. (Note K here is assumed to be an implicit parameter of $h(x)$.) The strongest condition one may impose on a MAC is that for someone who does not know the secret key, it be computationally infeasible to perform an *existential forgery*, i.e. to find an arbitrary message and its corresponding MAC. This should be contrasted to a *selective forgery*, where an opponent can determine the MAC for a message of his choice. For a practical attack, one often requires that the forgery is *verifiable*, which means that the MAC is correct with probability close to 1. Here we assume that the opponent is capable of performing a *chosen text attack*, i.e. may obtain MACs corresponding to a number of messages of his choice. We allow in fact a stronger notion, namely an *adaptive* chosen text attack, in which his requests may depend on the outcome of previous requests. To be meaningful, a forgery must be for a message different than any for which a MAC was previously obtained.

For an *ideal* MAC, any method to find the key is as expensive as an exhaustive search of $O(2^k)$ operations for a k-bit key. The number of text-MAC pairs required for verification of such an attack is k/m. An opponent who has identified the correct key can compute the MAC for any message (i.e. key recovery allows selective forgery). If the opponent knows no text-MAC pairs, or if $m < k$, his best strategy may be to simply guess the MAC corresponding to a chosen message; the probability of success is $1/2^m$. The disadvantage of a guessing attack is that it is not verifiable. A further desirable property of an ideal MAC is that finding a second preimage should require $O(2^m)$ known text-MAC pairs. In some settings (e.g. multi-destination electronic mail [21]) it may be desirable that this requires $O(2^m)$ off-line MAC computations even for someone who knows the key.

Most hash functions, including MACs, are designed as iterative processes which hash inputs of arbitrary length by processing successive fixed-size b-bit blocks of the input. The input x is divided into t blocks x_1 through x_t. If the total length is not a multiple of b, the input is padded using an unambiguous padding rule. The hash function h can be described as follows:

$$H_0 = IV; \qquad H_i = f(H_{i-1}, x_i), 1 \leq i \leq t \qquad h(x) = H_t \,.$$

Here f is the *compression function* of h, and H_i is the *chaining variable* between stage $i-1$ and stage i with bitlength n ($n \geq m$).

In the case of a MAC, one often applies an output transformation g to H_t to obtain the hash-result, i.e. $h(x) = g(H_t)$. For example, in the CBC-MAC as specified in [14, 15] the output transformation g consists of selecting the leftmost m bits. The secret key can be introduced in the IV, in the compression function f, and in the output transformation g.

3 A New General Attack on MAC Algorithms

We describe a new attack which is applicable to all iterated MACs. The parameters depend only on the bitsize n of the chaining variables and on the bitsize m of the hash-result. We first make no assumptions about the texts being hashed, and further below give an optimization in the case that texts have a common sequence of s trailing blocks.

To facilitate Proposition 2 below, we first make two simple definitions followed by a lemma. Consider the pair (x, x') with $h(x) = g(H_t)$ and $h(x') = g(H'_t)$, where g is the output transformation as defined above. Given a collision $h(x) = h(x')$, it may have arisen in one of two ways. An *internal* collision is said to have occurred if $H_t = H'_t$. An *external* collision is said to have occurred if $H_t \neq H'_t$ but $g(H_t) = g(H'_t)$.

Lemma 1. *An internal collision for an iterated MAC algorithm can be used to obtain a verifiable MAC forgery with a chosen text attack requiring only one requested MAC.*

Proof: For an internal collision (x, x'), note $h(x \,\|\, y) = h(x' \,\|\, y)$ for any single block y. Thus requesting a MAC for the single chosen text $x \,\|\, y$, permits forgery – the MAC for $x' \,\|\, y$ is the same (here $\|$ denotes concatenation). ∎

Note that this observation has been made independently by others including H. Krawczyk (e.g. see [18]). It follows that a security requirement for MACs is that it be infeasible for an adversary to find internal collisions (cf. collision resistance for hash functions).

Proposition 2. *Let h be an iterated MAC with n-bit chaining variable and m-bit result. An internal collision for h can be found using u known text-MAC pairs and v chosen texts. The expected values for u and v are as follows: $u = \sqrt{2} \cdot 2^{n/2}$ and*[1] *$v = 0$ if the output transformation g is a permutation; otherwise, v is approximately $2 \cdot 2^{n-m} + 2\left\lceil \frac{n}{m} \right\rceil$.*

Proof: If the number of known texts is $r = \sqrt{2} \cdot 2^{n/2}$, a single internal collision is expected by the birthday paradox (note $\binom{r}{2}/2^n \approx 1$). If g is a permutation (e.g. the identity mapping), all collisions are internal and the results follows by Lemma 1. If g behaves as a random function[2], $\binom{r}{2}/2^m \approx r^2/2^{m+1} = 2^{n-m}$ external collisions are expected and additional work is required – for a verifiable forgery – to distinguish the internal collision from the external collisions. (Note Lemma 1 requires internal collisions.) This may be done by appending a string y to both elements of each collision pair and checking whether the corresponding MACs are equal. This requires $2(1 + 2^{n-m})$ chosen text-MAC requests. For an internal collision both results will always be equal, while for an external collision this will be so with probability[3] $1/2^m$. Discard collision pairs corresponding to unequal MACs. The expected number of remaining collision pairs after this stage is 2^{n-2m} external plus one internal (but these cannot yet be distinguished). If the (total) number of remaining collision pairs is 2 or more (e.g. $n - 2m > 0$), further external collisions must be discarded by appending a different y, and continuing in this manner until only a single collision remains; with high probability this is an internal collision. This may require a small number of additional chosen texts and a total number $2 \cdot 2^{n-m} \cdot 2^m/(2^m - 1) + 2\left\lceil \frac{n}{m} \right\rceil$. ∎

Note that creating t MAC forgeries by this method requires one internal collision and t chosen-text MAC requests. The cost of this one internal collision is given by Proposition 2.

The attack outlined in the proof of Proposition 2 yields an internal collision (x, x'). If x and x' have a common sequence of s trailing blocks and if the compression function f is a permutation (for fixed x_i), the collision must occur at H_{t-s}, i.e. just before the common blocks. After deleting the s common blocks in x and x', one still has an internal collision. In this case the attack can be enhanced since this provides additional freedom in the choice of the forged text by Lemma 1. In particular, if x and x' have the same length one can obtain a forgery on a text of that length. As a significant consequence, *in this case the attack cannot be precluded by prepending the length of the input before the MAC calculation or by fixing the length of the input.*

If all the texts in the known text-MAC pairs of Proposition 2 have a common sequence of s trailing blocks, and if the compression function behaves as a random

[1] For this choice of u the probability of the attack failing is $1/e$; however, such failure can be detected in which case selecting additional known text-MAC pairs is required. By doubling the number of known text-MAC pairs the probability of failure decreases to $1/e^4$. The same probabilities hold for the value u given in Proposition 4.

[2] This is formalized in the full paper. The effective random mapping is from n bits to m bits; if the image is smaller, the collision probability increases.

[3] If y has i blocks, the probability increases by a factor i (see Lemma 3 with $r = 2$).

mapping, fewer known and chosen texts are required. In order to prove this we use a generalization of the birthday attack (proof to be given in the full paper):

Lemma 3. *Let h be an iterated MAC with n-bit chaining variable, a compression function f which behaves like a random function (for fixed x_i), and an output transformation g that is a permutation. Consider a set of $r \geq 2$ distinct messages which have the last s blocks in common, with $r^2 s = O(2^n)$. The probability that the set contains at least two messages that collide under h is approximately*

$$1 - \exp\left(-\frac{r^2(s+1)}{2^{n+1}}\right) .$$

∎

(Note this reduces to the well known birthday attack for $s = 0$.) This lemma can be extended easily to the case where g is a random function. It yields an optimization of Proposition 2 as follows (proof to be given in the full paper):

Proposition 4. *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 as follows: $u = \sqrt{2/(s+1)} \cdot 2^{n/2}$; $v = 0$ if g is a permutation or $s + 1 \geq 2^{n-m+6}$ (the expected number of external collisions is sufficiently small), and otherwise v is approximately*

$$2 \cdot \frac{2^{n-m}}{(s+1)} + 2\left\lceil \frac{n - \log_2(s+1)}{m} \right\rceil .$$

∎

If we have an internal collision with $s \geq 1$, the probability that it occurs before the last w blocks equals $1 - w/(s+1)$. This event can be checked with a small number of additional chosen texts. Again the attack still works if one appends an arbitrary block y after the internal collision rather than at the end. This means that an attacker can replace or delete $w \leq s$ trailing blocks, and that *the attack is applicable even if the input is of fixed length or if the length is prepended to the input* (cf. [1]). A non-verifiable version of the attack requires $\sqrt{2/(s+1)} \cdot 2^{n/2}$ known texts and only a single chosen text, with success probability approximately $1/(2^{n-m}/(s+1)+1)$.

Applying Proposition 4 to MAA, $n = 64$, $m = 32$, and $s \geq 2$ (since two key-dependent blocks are always appended). For $s = 2$, the attack of Lemma 1 requires $0.82 \cdot 2^{32}$ known text-MAC pairs and $0.67 \cdot 2^{32}$ chosen text-MAC pairs. For $s = 2^{16}+2$ (corresponding to a fixed but arbitrary 256 Kbyte trailing block), $2^{24.5}$ known texts and 131 071 chosen texts are required. Note that the designer of MAA realized that its compression function not being a bijection might lead to weaknesses, motivating a special mode in [14] for messages longer than 1024 bytes. However, it turns out that the above attack is applicable to this mode as well. This is the first attack on MAA (that we are aware of) which is more efficient than an exhaustive key search or guessing the MAC.

For CBC-MAC with $m = n = 64$, Proposition 2 requires $2^{32.5}$ known text-MAC pairs and one chosen text; for $m = 32$, about 2^{33} additional chosen texts are required. The attack of Proposition 4 fails for CBC-MAC and CFB-MAC with maximal feedback, since for these the compression function is bijective on the chaining variable for fixed x_i (e.g. $H_i = f(H_{i-1}, x_i) = E_K(H_{i-1} \oplus x_i)$, where $E_K(H_{i-1})$ denotes the encryption of H_{i-1} with key K). However, it does apply to CFB-MAC with feedback shorter than one block and to RIPE-MAC [23]. In §4 we discuss other specific schemes to which the attacks apply. Proposition 4 answers in part an open question arising in the discussion of CBC-MAC [1] on whether a bijective compression function (for fixed input x_i) allows stronger security claims. Note that Lemma 3 is of independent interest for parallelizing a collision search when the constraint is the number of hash function evaluations rather than the number of evaluations of the round function.

4 Three Previous MACs Based on Hash Functions

In this section we discuss the security of three proposals to construct a MAC based on a hash function: the secret prefix, secret suffix, and envelope methods.

4.1 The Secret Prefix Method

The *secret prefix* method consists of prepending a secret key K_1 to the message x before the hashing operation: $\text{MAC}(x) = h(K_1 \| x)$ for h an unkeyed hash function. If the key consists of a complete block, this corresponds to a hash function with a secret IV. This method was suggested for MD4 independently by Tsudik [26] and by the Internet Security and Privacy Working Group for use in the Simple Network Management Protocol (SNMP) [13]. In the 1980s this was already proposed for at least two other schemes (for example [3]). As has been pointed out in several papers, this MAC is insecure: a single text-MAC pair contains information essentially equivalent to the secret key, independent of its size. An attacker may append any blocks to the message and update the MAC accordingly, using the old MAC as the initial chaining variable for the update. The messages for which an attacker can compute the MAC are restricted to those having known texts as prefix, but this is only a very weak restriction. The appending attack may be precluded if only a subset of the hash output bits are used as the MAC (e.g. $m = n/2$ as for MD2.5 below), or by prepending the length of the message before hashing [26]. However, relying on a prepended length for security appears to make additional demands on the properties of the hash function and the attack discussed following Proposition 4 still applies for $s \geq 1$. Indeed, the compression function in MD4-based hash functions is of the form $H_i = E_{x_i}(H_{i-1}) + H_{i-1}$, which behaves as a random function (for fixed x_i); the addition here is modulo 2^{32}.

A variation of the prefix method with MD5 is used in Kerberos V5, under the name MD2.5 [20]. The 128-bit key K_1 is derived from a 56-bit DES key K by using DES as a keystream generator in Output Feedback Mode (OFB) with

$IV = 0$. The MAC consists of the leftmost 64 bits of the 128-bit hash-result. While the expansion does not thwart an exhaustive search for the DES key, it appears to provide some benefit. Moreover, revealing only 64 bits of the hash-result makes it hard to append one or more blocks and update the MAC. However, the attack of Proposition 4 still applies if $s \geq 1$ and if we have an internal collision before the last block ($w \geq 1$). Also, it remains conceivable that one could append carefully chosen blocks at the end in such a way that the new MAC depends only on the 64 known bits, implying that choosing $m < n$ imposes additional conditions on the hash function beyond those for which it was designed or has yet been analyzed. An unfortunate additional drawback is that, while one advantage of an MD5-based MAC over a DES-CBC MAC is avoidance of block ciphers and associated possible export issues, DES is nonetheless required for key expansion in MD2.5. Another concern is that a 56-bit key does not offer sufficient protection against an exhaustive key search [29].

4.2 The Secret Suffix Method

A second proposal is to append a secret key K_2 to the message: $\text{MAC}(x) = h(x\|K_2)$. This approach, proposed for SNMP [13], is called the *secret suffix* method in [26]. A concern with this method is that an off-line collision attack on the hash function may be used to obtain an internal collision; therefore by a birthday attack, finding a pair (x, x') such that $h(x) = h(x')$ requires about $O(2^{n/2})$ off-line operations. (Note that the candidates for the collision search can be chosen from a controlled set.) Lemma 1 may then be applied. Moreover, this method is weak if an (off-line) second preimage attack on the underlying hash function is feasible – given one known text-MAC pair, a second hash function preimage (for that text) allows an existential MAC forgery. If t text-MAC pairs are known, finding a MAC second preimage requires $2^n/t$ rather than 2^n off-line trials; here, if the length of the message is not appended, t is the total number of blocks rather than the number of messages. Finally, it should be noted that an attacker can remove the feedforward in the last iteration, since the the chaining variable entering this iteration can be computed using x only.

4.3 The Envelope Method

The *envelope method* [26] combines the prefix and suffix methods. One prepends a secret key K_1 and appends a secret key K_2 to the message input: $\text{MAC}(x) = h(K_1\|x\|K_2)$. It is claimed in [26] (along with a sketch of proof) that a divide and conquer attack against K_1 and K_2 is not possible, and that breaking this method requires exhaustive search for a key of $k_1 + k_2$ bits (with $k_i = |K_i|$ and $k_1 = n$, the size of the chaining variable). We now show this statement is false.

The approach suggested in §3 can be used in a divide and conquer key recovery attack on K_1 and K_2. Once an attacker finds an internal collision for the chaining variables, he can perform an exhaustive search for K_1, eliminating all trial key values which do not yield a collision before the last block (i.e. an internal collision). This requires 2^{k_1} off-line operations. Slightly more than k_1/n internal collisions are required to determine K_1 uniquely [22]; if $k_1 = n$, two collisions are certainly sufficient. After this, the envelope method is effectively reduced to the secret suffix method. Once K_1 is determined, an exhaustive search can be used to find K_2. This disproves the claim of [26]. Note that choosing $K_1 \neq K_2$ does not offer nearly as much additional security as one might think relative to $K_1 = K_2$ (which provides k_1 bits of security from exhaustive search); an attack on the former however does require a large number of known text-MAC pairs.

The envelope method used with MD4-based hash functions is also subject to the forgery as discussed in §4.1; the MAC forgery possible by Proposition 4 applies with $m = n = 128$ (regardless of k_i) and $s \geq 1$ (assume the last block consists of K_2 only). For $s = 2^{16}$, $2^{56.5}$ known text-MAC pairs are required and one chosen text. The result is that the security of this scheme is significantly less than that suggested by the key size $k_1 + k_2$.

Three MD5-based MAC proposals for the IPSEC working group are made in [18]: one is the envelope method with $K_1 = K_2$ and $k_1 = 128$ (K_1 is padded to a complete block), the other two are $\mathrm{MAC}(x) = h(K_1 \| h(K_2 \| x))$ and $\mathrm{MAC}(x) = h(K_1 \| h(K_1 \| x))$. It is suggested that the best known attack on these schemes requires 2^{64} chosen messages; however, Proposition 4 shows that $2^{56.5}$ known text-MAC pairs are sufficient (if $s = 2^{16}$). Also, the second scheme is vulnerable to the divide and conquer attack described above.

4.4 Summary of Results on the Three Previous Proposals

The weaknesses of the three existing proposals discussed above are summarized in Table 1. Storage requirements (e.g. for known pairs) have been omitted, as well as the potential improvements due to common trailing blocks as discussed in §3. The tabulated values, corresponding to the best known attacks, give *upper* bounds on the security of these constructions. Depending on the parameters, finding a second preimage may be easier by first obtaining the key with an exhaustive search; this type of attack is not noted in the table.

If the underlying hash function is collision resistant (implying n is sufficiently large), the figures in Table 1 (aside from the secret prefix method without additional precautions) indicate that the corresponding attacks are only *certificational* – breaking these schemes is easier than breaking an ideal MAC with the same parameters, but the attacks are still not feasible in practice. In particular, the number of known or chosen texts required is much smaller than one would expect, and known texts can be replaced by off-line computations. It is however clear from Table 1 that if the hash function is only a one-way hash function (with n typically between 64 and 80 bits), then both the suffix and envelope methods are vulnerable as well. Also, it follows that in case of the envelope method k_1 must not be too small.

Even if keys are chosen sufficiently large that these attacks are computationally infeasible, one should keep in mind the attacks are independent of possible weaknesses of the hash function. More sophisticated attacks might be found which exploit such weaknesses.

Table 1. Security of 3 proposals to build n-bit MACs ($n = m$) from hash functions. "#MAC" is the number of known text-MAC pairs; "C" the number of chosen texts; "#opn" the number of off-line compression function operations required for best known attacks; t is the number of messages (or blocks) available to an attacker; k, k_1, k_2 are key bitlengths.

	ideal MAC (k)		secret prefix (k_1)		secret suffix (k_2)		envelope ($k_1 + k_2$)	
	#MAC	#opn	#MAC	#opn	#MAC	#opn	#MAC	#opn
key recovery	$\lceil \frac{k}{n} \rceil$	2^k	1	0‡	$\lceil \frac{k_2}{n} \rceil$	2^{k_2}	$\lceil \frac{k_1+k_2}{n} \rceil$	$2^{k_1+k_2}$
							$2^{n/2}$	$2^{k_1} + 2^{k_2}$
MAC forgery	$\lceil \frac{k}{n} \rceil$	2^k	1	1	1C	$2^{n/2}$	5C+$2^{n/2}$	0
							$2^{n/2}$	2^{k_1} †
2nd preim.	2^n	0	t	$2^n/t$	t	$2^n/t$	2^n	0

†This attack reduces the envelope method to the secret suffix method only.
‡Information essentially equivalent to the secret key is known.

5 A New MAC Construction: MDx-MAC

From the previous section it may be concluded that extreme care must be exercised in constructing a MAC from a hash function. With this in mind, we propose a new construction, with the following design goals:

1. The secret key should be involved at the beginning, at the end, and in every iteration of the hash function (cf. MAA [5, 6, 14]).
2. The deviation from the original hash function should be minimal (to minimize implementation effort and maximize on confidence previously gained).
3. The performance should be close to that of the hash function.
4. The additional memory requirements should be minimized (keeping smart card implementations in mind).
5. The approach should be generic, i.e. should apply to any hash function based on the same principles as MD4.

The new construction converts the hash function *MDx* into the MAC algorithm *MDx*-MAC with a key K up to 128 bits in length. Here *MDx* may be any of MD5, RIPEMD, SHA, or similar algorithms. (We omit MD4 itself from recommendation because of weaknesses identified in [8] and [27].)

MDx-MAC uses three 16-byte constants T_0, T_1, T_2, which define three further 96-byte constants U_0, U_1, U_2 (see below). The first run-time step is key expansion. If K is shorter than 128 bits, concatenate K to itself a sufficient number of times, and select the leftmost 128 bits. Let $\overline{MDx}$ denote algorithm *MDx* with both padding and appended length omitted. The 16-byte secret key K is expanded to three 16-byte (or for SHA, 20-byte) subkeys K_0, K_1, and K_2:

$$for\ i := 0\ to\ 2 \qquad K_i := \overline{MDx}(K \,\|\, U_i \,\|\, K)$$

The lefmost 16 bytes of the derived key K_1 are split into four 32-bit substrings denoted $K_1[i]$ $(0 \le i \le 3)$. Also, only the leftmost 16 bytes of K_2 are retained. Note for SHA, $\overline{MDx}$ produces a 20-byte output versus 16 bytes for e.g. MD5.

MDx-MAC is then obtained from *MDx* with modifications as follows:

1. The initial value *IV* of *MDx* is replaced by K_0.
2. $K_1[i \bmod 4]$ is added mod 2^{32} to the constants which are used in round i of each iteration of *MDx*.[4]
3. Following the block containing the padding and appended length as defined by *MDx* (i.e. the last block after normal post-processing), an additional complete 64-byte block is appended which has the following form:

$$K_2 \,\|\, K_2 \oplus T_0 \,\|\, K_2 \oplus T_1 \,\|\, K_2 \oplus T_2$$

4. The MAC result is the leftmost m bits of the hash value. In view of the attack of Proposition 4, $m = n/2$ is recommended for most applications.

The interpretation of strings as integers is defined to match that used in *MDx*.

The computational overhead of the MAC construction is 6 block operations for the key expansion (2 for each K_i); a single block operation is required for each 64 bytes of message. The additional storage requirements are 16 bytes for K, 48 bytes for the T_i, and 16 bytes for K_2; K_0 may be computed when required, and K_1 may be added immediately to the constants. Software implementations indicate MD5-MAC is 5-20% slower than MD5 (one factor is that the modified constants must be read from memory): on a 33 MHz 80486, MD5-MAC runs at 11.3 Mbit/s, while MD5 achieves 14.3 Mbit/s; on a HP 715–80, MD5-MAC runs at 43.6 Mbit/s compared to 46.9 Mbit/s for MD5. For RIPEMD and SHA, the performance difference is smaller since the modified constants can be stored in a register. Below we give the hex constants T_i and three test vectors (x, MD5-MAC(x)) for hex key K = `00112233445566778899aabbccddeeff`:

```
T0:   97 ef 45 ac 29 0f 43 cd 45 7e 1b 55 1c 80 11 34
T1:   b1 77 ce 96 2e 72 8e 7c 5f 5a ab 0a 36 43 be 18
T2:   9d 21 b4 21 bc 87 b9 4d a2 9d 27 bd c7 5b d7 c3
("",                             1f1ef2375cc0e0844f98e7e811a34da8)
("abc",                          e8013c11f7209d1328c0caa04fd012a6)
("abcdefghijklmnopqrstuvwxyz", 9172867eb60017884c6fa8cc88ebe7c9)
```

The 16-byte constants T_i and 96-byte constants U_i are defined as follows. The definition of T_i involves the 62-byte constant R = "ab. . . yzAB. . . YZ01. . . 89" and 2-byte constants S_0, S_1, S_2, where S_i is the 16-bit string formed by repeating twice the 8-bit hexadecimal representation of i (e.g. S_1 = `3131`).

$$for\ i := 0\ to\ 2 \qquad T_i := \overline{MDx}(S_i \,\|\, R) \text{ (leftmost 16 bytes of)}$$
$$for\ i := 0\ to\ 2 \qquad U_i := T_i \,\|\, T_{i+1} \,\|\, T_{i+2} \,\|\, T_i \,\|\, T_{i+1} \,\|\, T_{i+2}$$

[4] For RIPEMD the rounds in the two independent iterations are numbered 0-2 and 3-5 respectively. For MD5 and SHA, the rounds are 0-3.

where the subscripts in T_i are taken modulo 3. The constants T_i and U_i are fixed for all time, given MDx.

The idea of constructing the T_i in this manner is to obtain "random" (in the sense of un-contrived) bit strings which are easy to compute if an implementation of MDx is available (as opposed to the constants $\sqrt{2}$ and $\sqrt{3}$ in MD4, and the sine constants in MD5). The U_i are defined in terms of the T_i, so that it suffices to store 48 bytes to define the three 96-byte strings U_i. The key expansion makes use of two compression functions in order to preclude recovering K from any of the K_i. (If only a single iteration were used, a cryptanalyst could remove the feedforward in MDx, which would reduce the strength.) The three derived keys K_0, K_1, and K_2 are computed from K by applying a one-way function, implying that even if two of the three values are known, it is computationally infeasible to compute the third. Also, the relation between these derived keys is hard to predict. While the mapping from K to K_i is not bijective, the expected reduction in entropy for each K_i is negligible.

The role of K_0 and K_2 here is similar to that of the secret keys K_1 and K_2 in the envelope method, but with the difference that a divide-and-conquer attack now provides no advantage. The intention is that the use of K_1 in MDx-MAC provides additional protection over the envelope method in the case that weaknesses of the hash function become known. (Use of the iteration-invariant K_1 is not very strong by itself: for MD5 and RIPEMD this is almost equivalent to an offset in each message block.) Finally, an exhaustive search for each of the K_i is as hard as an exhaustive search for K. An advantage of the overall approach is that it minimizes the difference between MDx and MDx-MAC, reducing the probability of introducing new weaknesses.

If, as recommended, the bitsize of the chaining variable is equal to twice that of the MAC (i.e. $n = 2m$), a forgery attack on the new scheme requires $O(2^m/(s+1))$ chosen text-MAC pairs and $O(2^m/\sqrt{(s+1)})$ known texts (Proposition 4), and thus MDx-MAC is better than the envelope method as given in [26] for which $m = n$. In addition, MDx-MAC does not succumb to the divide and conquer key search attack. Moreover, the key size of 128 bits gives a better idea of the actual strength of MDx-MAC than the 128+512=640 bits previously proposed for the envelope method. We believe the new scheme would also be stronger against attacks exploiting the internal structure of MDx. Based on the above remarks, we state the following conjecture:

Conjecture 1 *If* MDx *is a secure hash function of the MD4-family, the best attacks on* MDx-*MAC are exhaustive (with respect to key search), and the attack of Proposition 4 (with respect to forgery).*

Although it was shown in [9] that the compression function of MD5 is not collision resistant, we do not expect this results in a weakness of MD5-MAC.

For SHA, a more natural construction might be to use a 160-bit key K, construct subkeys K_i of the same length, and make appropriate modifications. However, a 128-bit key appears adequate against all plausible key search attacks, and as defined (with $n = 2m = 160$), the forgery attack of Proposition 2 against SHA-MAC requires about 2^{80} chosen text-MAC pairs and 2^{80} known texts. Note that the corresponding numbers of known and chosen texts for MD5-MAC and RIPEMD-MAC are about 2^{64}.

6 Concluding Remarks

The new forgery attack on iterated MACs requires $O(2^{n/2})$ known text-MAC pairs and $O(2^{n-m})$ chosen texts, where m is the bitlength of the hash-result and n is that of the chaining variable. A naive non-verifiable attack always succeeds with probability 2^{-k} by guessing the k-bit key and computing the MAC, or 2^{-m} by guessing the MAC. These attack scenarios differ, but nonetheless suggest using $n = 2m = k$.

The new attack may pose a serious threat to certain applications of CBC-MAC (e.g. when $n = m = 64$). Its implications for the security of MAA are also serious. There are disadvantages with MAC functions such as DES CBC-MAC built from block ciphers (including speed and exportability). The analysis of existing proposals indicates that one must exercise care in designing MACs based on hash functions. The new MAC construction addresses all of these concerns and appears to be the first such proposal based on existing hash functions. It differs from previous such proposals in that it involves a secret key in each iteration of the hash function and yet is sufficiently generic to apply to any function of the MD4-family. As with all new proposals, we caution that it would be imprudent to employ the new MAC algorithm in practice prior to adequate peer review.

Acknowledgements

We would like to thank Antoon Bosselaers and Roland Lockhart for efficient independent software implementations confirming the test vectors in §5.

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* (this volume).
3. F. Cohen, "A cryptographic checksum for integrity protection," *Computers & Security*, Vol. 6, No. 5, 1987, pp. 505–510.
4. I.B. Damgård, "A design principle for hash functions," *Proc. Crypto'89, LNCS 435*, Springer-Verlag, 1990, pp. 416–427.
5. D. Davies, "A message authenticator algorithm suitable for a mainframe computer," *Proc. Crypto'84, LNCS 196*, Springer-Verlag, 1985, pp. 393–400.
6. D. Davies, D.O. Clayden, "The message authenticator algorithm (MAA) and its implementation," *NPL Report DITC 109/88*, Feb. 1988.
7. D. Davies, W. Price, *Security for Computer Networks*, 2nd ed., Wiley, 1989.
8. B. den Boer, A. Bosselaers, "An attack on the last two rounds of MD4," *Proc. Crypto'91, LNCS 576*, Springer-Verlag, 1992, pp. 194–203.

9. B. den Boer, A. Bosselaers, "Collisions for the compression function of MD5," *Proc. Eurocrypt'93, LNCS 765*, Springer-Verlag, 1994, pp. 293–304.
10. FIPS 46, *Data encryption standard,* NBS, U.S. Department of Commerce, Washington D.C., Jan. 1977.
11. FIPS 81, *DES modes of operation,* NBS, US Department of Commerce, Washington D.C., Dec. 1980.
12. FIPS 180-1, *Secure hash standard,* NIST, US Department of Commerce, Washington D.C., April 1995.
13. J.M. Galvin, K. McCloghrie, J.R. Davin, "Secure management of SNMP networks," *Integrated Network Management, II*, North Holland, 1991, pp. 703–714.
14. ISO 8731:1987, *Banking – approved algorithms for message authentication, Part 1, DEA,* IS 8731-1, *Part 2, Message Authentication Algorithm (MAA),* IS 8731-2.
15. ISO/IEC 9797:1993, *Information technology - Data cryptographic techniques - Data integrity mechanisms using a cryptographic check function employing a block cipher algorithm,*.
16. T. Johansson, G. Kabatianskii, B. Smeets, "On the relation between A-codes and codes correcting independent errors," *Proc. Eurocrypt'93, LNCS 765*, Springer-Verlag, 1994, pp. 1–11.
17. R.R. Jueneman, S.M. Matyas, C.H. Meyer, "Message authentication with Manipulation Detection Codes," *Proc. 1983 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, 1983, pp. 33–54.
18. B. Kaliski, M. Robshaw, "Message authentication with MD5," *CryptoBytes (RSA Laboratories Technical Newsletter)*, Vol. 1, No. 1, Spring 1995, pp. 5–8.
19. H. Krawczyk, "LFSR-based hashing and authentication," *Proc. Crypto'94, LNCS 839*, Springer-Verlag, 1994, pp. 129–139.
20. J. Linn, "The Kerberos Version 5 GSS-API Mechanism," *Internet Draft*, Feb. 1995.
21. C. Mitchell, M. Walker, "Solutions to the multidestination secure electronic mail problem," *Computers & Security*, Vol. 7, No. 5, 1988, pp. 483–488.
22. B. Preneel, *Cryptographic Hash Functions*, Kluwer Academic Publishers, 1995 (to appear).
23. RIPE, *Race Integrity Primitives Evaluation (RIPE-RACE 1040): Final Report,* LNCS, Springer-Verlag, 1995 (to appear).
24. R.L. Rivest, "The MD4 message digest algorithm," *Proc. Crypto'90, LNCS 537*, Springer-Verlag, 1991, pp. 303–311.
25. R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC) 1321*, Internet Activities Board, Internet Privacy Task Force, April 1992.
26. G. Tsudik, "Message authentication with one-way hash functions," *ACM Computer Communications Review*, Vol. 22, No. 5, 1992, pp. 29–38.
27. S. Vaudenay, "On the need for multipermutations: cryptanalysis of MD4 and SAFER," *Fast Software Encryption, LNCS*, Springer-Verlag, 1995 (to appear).
28. 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, 1981, pp. 265–279.
29. 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.