New (Two-Track-)MAC

Based on the Two Trails of RIPEMD

Efficient, Especially on Short Messages and for Frequent Key-Changes

Bert den Boer^{1,*}, Bart Van Rompay^{2,**}, Bart Preneel², and Joos Vandewalle²

¹ TNO/TPD, Stieltjesweg 1, 2628 CK Delft, The Netherlands
`bdenboer@tpd.tno.nl`

² Katholieke Universiteit Leuven, ESAT-COSIC,
Kasteelpark Arenberg 10, 3001 Leuven-Heverlee, Belgium
`bart.vanrompay@esat.kuleuven.ac.be`

Abstract. We present a new message authentication code. It is based on a two trail construction, which underlies the unkeyed hash function RIPEMD-160. It is in comparison with the MDx-MAC based on RIPEMD-160, much more efficient on short messages (that is on messages of 512 or 1024 bits) and percentage-wise a little bit more efficient on long messages. Moreover, it handles key-changes very efficiently. This positive fact remains if we compare our Two-Track-MAC with HMAC based on RIPEMD-160.

1 Introduction

Message Authentication Codes (MACs) are symmetric-key cryptographic primitives used to provide data integrity and symmetric data origin authentication. Given a message M to be authenticated and a secret key K (shared between two parties), the MAC algorithm computes an authentication tag $A = MAC(K, M)$ for the message. The pair (M, A) is passed from sender to receiver who can verify the authentication tag by computing the MAC of the message himself (as he knows the key).

The goal of an adversary (who does not know the key) is to forge a MAC for a message of his choice (selective forgery), or for an arbitrary message (existential forgery). Here it is assumed that the adversary has knowledge of a number of messages M^i and their corresponding authentication tags $A^i = MAC(K, M^i)$. In the case of a chosen-text attack the opponent is even able to request the MAC for a number of messages of his choice (before forging a MAC on a new, and different, message).

* Algorithm invented while working at debis Information Security Services – Bonn, Germany.

** The work described in this paper has been supported in part by the Commission of the European Communities through the IST Programme under Contract IST-1999-12324 and by the Concerted Research Action (GOA) Mefisto-666.

It is a common approach to construct MAC algorithms from existing cryptographic hash functions, as such schemes require little additional implementation effort. They are also generally faster than MACs which are based on block ciphers. A cryptographic hash function is a function which compresses an input of arbitrary length into a hash value of fixed length, while also satisfying some additional cryptographic properties (preimage resistance and collision resistance). A hash function usually works by iteration of a compression function, which has a fixed-length message input operating on an internal state variable. The final value of this internal state then serves as hash value.

To build a MAC algorithm from a hash function it is necessary to include a secondary input, the secret key, in the computation. Early proposals such as the envelope method [6], where the key material is simply prepended and appended to the message input to the hash function, were shown to have significant weaknesses [4,5]. MDx-MAC and HMAC have emerged as the most secure alternatives. MDx-MAC [4], which can be based on MD5, RIPEMD, SHA or similar algorithms, makes some small changes to the hash function used, while HMAC [1] is a black box construction that can be based on any hash function.

In this paper we will present a new MAC algorithm, called Two-Track-MAC (or TTMAC in short). It has been submitted as a candidate algorithm for the NESSIE project [3]. The algorithm is based on the RIPEMD-160 hash function [2] (making only small changes to the hash function). We will show that the structure of RIPEMD, which consists of two parallel trails, has been exploited to double the size of the internal state, and that this allows to significantly reduce the overhead in the computation of the MAC for short messages, compared to the other MAC constructions. Another advantage of our proposal is better efficiency in the case of frequent key changes. These properties are very useful in applications, e.g., banking applications, where many short messages need to be authenticated (with frequent key changes). Although there is no formal proof of security for our construction, based on the heuristic arguments presented in Section 3, we believe it is very unlikely that an attack can be found on Two-Track-MAC, which would not also breach the security of RIPEMD-160.

The remainder of this paper is organized as follows. In Section 2 we present our new MAC. Section 3 discusses the security, and Section 4 the efficiency of our proposal. In Section 5 we suggest a more general construction method that can be used to construct new schemes. Section 6 concludes the paper. Pseudo-code for our algorithm is given in the Appendix.

2 Presentation of Two-Track-MAC

The unkeyed hash function RIPEMD-160 (for a description we refer to [2]) uses two trails in its compression function. If we separate those two trails then each trail can be seen as a transformation of a 160-bit input I , controlled by a message M , consisting of sixteen words of 32 bits. Those 160 bits of the input I (and of the output) consist of five words of 32 bits. Call the output of the different trails $L(I, M)$ and $R(I, M)$ (left respectively right trail output for an input I and a

message M), then our proposal for a MAC on a relative short message M (of 512 bits) and a key K of 160 bits is (in short notation)

$$R(K, M) - L(K, M).$$

Or as $R(K, M)$ can be viewed as five words A_i of 32 bits : $(A_0, A_1, A_2, A_3, A_4)$, and similarly the value $L(K, M)$ as $(B_0, B_1, B_2, B_3, B_4)$, we get an output $E = (E_0, E_1, E_2, E_3, E_4)$ of five 32-bit words. Here

$$E_i = A_i - B_i \text{ (subtraction modulo } 2^{32}) \text{ for } i = 0, 1, 2, 3, 4.$$

Then the 160-bit string E is the MAC of the 512-bit message M . Figure 1 gives a schematic view of this computation.

Fig. 1. High level view of TTMAC for a message of a single block.

If the message is longer, i.e. $M = M_1 M_2 M_3 \cdots M_n$ where each M_i is of length 512 bits, we define, using a new operation L^* and a new operation R^* , the 160-bit quantity A , respectively the 160-bit quantity B . $A = (A_0, A_1, A_2, A_3, A_4) = L^*(K, M_1)$, where each A_i is a 32 bit word. And $B = (B_0, B_1, B_2, B_3, B_4) = R^*(K, M_1)$ as the result of the right trail. The operation L^* is based on the operation L , which had a straightforward inverse operation on the first (160 bits long) argument. This new operation L^* has a simple feedback with the first argument, i.e.

$$L^*(I, M) = L(I, M) - I$$

(this is five times a subtraction modulo 2^{32}). Similarly the operation R^* is defined in shorthand as

$$R^*(I, M) = R(I, M) - I$$

(this is five times a subtraction modulo 2^{32}). Now we introduce two 160-bit blocks C and D of five 32-bit words, $C = (C_0, C_1, C_2, C_3, C_4)$ and $D = (D_0, D_1, D_2, D_3, D_4)$, which are defined as follows:

$$C_2 = A_3 - B_0,$$

$$\begin{aligned}
C_3 &= A_4 - B_1, \\
C_4 &= A_0 - B_2, \\
C_0 &= (A_1 + A_4) - B_3, \\
C_1 &= A_2 - B_4, \\
D_1 &= (A_4 + A_2) - B_0, \\
D_2 &= A_0 - B_1, \\
D_3 &= A_1 - B_2, \\
D_4 &= A_2 - B_3, \\
D_0 &= A_3 - B_4.
\end{aligned}$$

All subtractions and additions are modulo 2^{32} . These 160-bit blocks C and D are the starting values for the left, respectively, right trail to incorporate the next 512-bit message block M_2 . If there are more message blocks M_i the iteration is the same. So we have

$$\begin{aligned}
HL(1) &= A = L^*(K, M_1), \\
HR(1) &= B = R^*(K, M_1),
\end{aligned}$$

and then iteratively (for $i = 2, \dots, n-1$) the three operations

$$\begin{aligned}
(A, B) &\rightarrow (C, D), \\
HL(i) &= A = L^*(C, M_i), \\
HR(i) &= B = R^*(D, M_i).
\end{aligned}$$

For the last message block M_n however, the role of the left and right trails is interchanged:

$$\begin{aligned}
(A, B) &\rightarrow (C, D), \\
HL(n) &= A = R^*(C, M_n), \\
HR(n) &= B = L^*(D, M_n).
\end{aligned}$$

Once we have $HL(n)$ and $HR(n)$ we define our MAC as $TTMAC(K, M)$ by $HL(n) - HR(n)$ (five times a subtraction modulo 2^{32}). In Figure 2 a schematic view of the computation is given for a message consisting of two blocks.

The same preprocessing rules as in the RIPEMD-160 hash function are used to format the message input to the algorithm (the message is padded to a bitlength which is a multiple of 512). An additional output transformation can be used to reduce the length of the MAC result. This transformation calculates the necessary number of output words, in such a manner that all of the normal output words are used. Let the normal 160-bit result be $E = (E_0, E_1, E_2, E_3, E_4)$, and denote the final (shortened) MAC result with F , consisting of t 32-bit words F_i ($t = 1, 2, 3$, or 4). For a 32-bit MAC we compute (using addition modulo 2^{32})

$$F_0 = E_0 + E_1 + E_2 + E_3 + E_4.$$

Fig. 2. High level view of TTMAC for a message of two blocks.

For a MAC result of 64, 96 or 128 bits we compute respectively the first two, the first three or all four of the following values (all additions are modulo 2^{32}):

$$F_0 = E_0 + E_1 + E_3,$$

$$F_1 = E_1 + E_2 + E_4,$$

$$F_2 = E_2 + E_3 + E_0,$$

$$F_3 = E_3 + E_4 + E_1.$$

3 Security of Our Proposal

3.1 Philosophy on the Security

The idea for the security is simple: Now we have an internal state variable ($HL(i), HR(i)$) of 320 bits. This is twice as long as for other MAC constructions (e.g., MDx-MAC and HMAC) based on RIPEMD-160 (or on SHA-1). Only in

the case of very weak transformations a cryptanalyst is allowed to hope on so-called internal collisions. In almost all attacks, which do not attack the very heart of the MAC (in our case the two trails of RIPEMD-160), forgery is based on internal collisions.

Another attack is possible if the MAC on a message contains all the information (or lacks “only” 32 bits of information) of the internal variable on a longer message, containing the first message as a prefix. In our case we have an internal state of 320 bits, so we can use 160 bits of information (the difference between the left and the right state variable, this depends on both trails) as the MAC output without compromising the internal state. Furthermore, we have used the idea of interchanging the left and right trails for the last message block as a free extra defence against such extension attacks. Note that other MAC constructions need to apply the compression function with some secret key material at the end of the computation in order to prevent these attacks. In our case, the secret key is only used as initial value for the two trails.

So now the worry for the cryptographer are the two trails of RIPEMD-160 itself. A single trail has one important weakness: it is a bijective operation, where the attacker can choose the bijection, which is parameterized by the 512-bit quantity M_i . But as long as two trails are used, parametrized by the same 512-bit quantity M_i , and only a sum will come out in the open, there is no danger that an attacker can invert the operation. Moreover, we have used feedback to counter a straightforward inverse operation. (We do not use feedback on messages of 512 bits, because there the feedback from the left trail would cancel out the feedback from the right trail, in other words we do not need feedback there). All this makes the transformation of a new 512-bit message block on the 320-bit internal variable a one-way operation.

Suppose a cryptanalyst discovers a message N , such that the function $L^*(\cdot, N)$, from a 160-bit leftside argument to a 160-bit output, has only a few short cycles (and many relative short tails ending in those cycles). Such a discovery is useless because we have chosen to mix the outputs of the two trails, as soon as the functions L^* and R^* have outputted their results. (Otherwise it might be possible for the cryptanalyst to generate collisions for the left trail by appending blocks to the message, and separately trying to find collisions for the right trail.) This mixing is thorough in the following sense: Denote the outcome of the left trail by A , the outcome of the right trail by B (as we did before), denote the MAC, in case we are done (in case this was the last message block), with E . Now denote C and D as the starting values for the new trails (in case it was not the last block). Then each pair out of the five values A, B, C, D and E has (just) enough information to determine the other three values. This ensures all kinds of injectivity properties.

The information theoretic uncertainty about the pair (C, D) can, of course, not be larger than the uncertainty about the key. That is 160 bits. But the goal of the design is that RIPEMD-160 is so “complicated”, that the “virtual” uncertainty about (C, D) is 320 bits. The construction is such that the information in any pair of the five tuple (A, B, C, D, E) is 320 bits. So if the uncertainty

about (C, D) is 320 bits then the uncertainty about for example the pair (A, E) is also 320 bits. This means that given all information about E , the uncertainty about A is still 160 bits. In other words the attacker has “virtually” no information about the starting value for the new left trail. Of course he has “virtual” information about the pair (A, B) , but that is more “complicated” than having information about a single trail.

3.2 Resistance against General Attacks

The resistance of Two-Track-MAC against forgery attacks which are generally applicable, depends on the following parameters: the keylength k which is 160 bits, the output length m which can be between 32 and 160 bits (in 32 bit steps), and the length l of the internal state which is 320 bits.

A first possible approach for an adversary is trying all possible keys (once he recovers the key he is able to forge the MAC for any message he chooses). For a key length k and output length m , such an attack requires about 2^k trials and k/m known text-MAC pairs (for verification of the attack).

Alternatively, the adversary can just guess the MAC corresponding to a chosen message. His success probability will be $1/2^m$ although this attack is not verifiable. The parameter m should be chosen long enough according to the needs of the application.

The forgery attack based on internal collisions requires about $2^{l/2}$ known text-MAC pairs to find an internal collision (with a birthday attack), and 2^{l-m} chosen texts to distinguish the internal collision from the external ones (this is shown in [4]). Table 1 below summarizes the difficulty of these general attacks applied to Two-Track-MAC.

Table 1. Resistance of Two-Track-MAC against general attacks. The output length m can take the following values: 32, 64, 96, 128 or 160 bits.

attack	trials	success prob.	known pairs	chosen texts
key search	2^{160}		$160/m$	
guessing the MAC		$1/2^m$		
internal collision			2^{160}	2^{320-m}

4 Short Comparison on the Efficiency of Two-Track-MAC

Our MAC uses only a few percent more operations on a message as RIPEMD-160 would do to get an unkeyed hash of the message (about 97% of the speed of RIPEMD-160 is achieved). This is already the case for the shortest possible

message of 512 bits. In contrast, both MDx-MAC and HMAC require an extra computation of the underlying compression function (using a secret key) at the end of the MAC computation. So that is relatively costly on messages of just one (512-bit) block (less than 50% of the speed of unkeyed hashing is achieved).

Also, since the secret key only serves as an initial value for the computation, a key-change will not slowdown the speed of the computation of Two-Track-MAC. In the case of HMAC or MDx-MAC a keychange costs respectively two or six extra computations with the underlying compression function.

5 General Construction

Our new MAC construction is not dependent on RIPEMD-160 alone. It just needs two operations L and $R : (T1 \times T2) \mapsto T1$. The set $T1$ should be big enough to make collisions improbable, for example $GF(2^{160})$. The size of the set $T2$ should be chosen big if messages are expected to be long. The operations L and R are allowed to be invertible if the second argument is fixed, but the operations L^* and R^* (including feedback from the first input) should be infeasible to invert. The operations L and R might be bijective in the first argument, but they should behave unpredictable on changes in the second argument, if the first argument is unknown (but perhaps fixed). It would even be better if the change in the output of the function L , say, is unpredictable with known first argument, i.e., the only way to know the effect of a change is to compute the new function value. Based on the experience that a first version of RIPEMD was partially broken, it is recommended that L and R should be as different as possible. In the case that $T1$ contains all 160-bit strings, one can use the same transitions $(A, B) \rightarrow (C, D)$ as we use for Two-Track-MAC based on RIPEMD-160. One of course needs to define also a padding rule as the message-length needs to be a multiple of some fixed quantity. With those transitions and a padding rule one can define the MAC on a message of any length.

6 Conclusion

We have presented a new message authentication code based on the two trail construction which underlies RIPEMD-160. The main advantage of the scheme is that it is more efficient than other schemes based on RIPEMD-160, especially in the case of short messages and frequent key-changes. We also suggested a more general construction method which can be used to construct new schemes.

A Pseudo-code for Two-Track-MAC

The TTMAC algorithm computes a 160-bit MAC value for an arbitrary message, under a 160-bit key. The result can be transformed into a shorter value by use of the output transformation (not reflected in the pseudo-code below). First we define all the constants and functions.

TTMAC: definitions

nonlinear functions at bit level: exor, mux, -, mux, -

$$f(j, x, y, z) = x \oplus y \oplus z \quad (0 \leq j \leq 15)$$

$$f(j, x, y, z) = (x \wedge y) \vee (\neg x \wedge z) \quad (16 \leq j \leq 31)$$

$$f(j, x, y, z) = (x \vee \neg y) \oplus z \quad (32 \leq j \leq 47)$$

$$f(j, x, y, z) = (x \wedge z) \vee (y \wedge \neg z) \quad (48 \leq j \leq 63)$$

$$f(j, x, y, z) = x \oplus (y \vee \neg z) \quad (64 \leq j \leq 79)$$

added constants (hexadecimal)

$$c(j) = 00000000_{\text{x}} \quad (0 \leq j \leq 15)$$

$$c(j) = 5A827999_{\text{x}} \quad (16 \leq j \leq 31) \quad \lfloor 2^{30} \cdot \sqrt{2} \rfloor$$

$$c(j) = 6ED9EBA1_{\text{x}} \quad (32 \leq j \leq 47) \quad \lfloor 2^{30} \cdot \sqrt{3} \rfloor$$

$$c(j) = 8F1BBCDC_{\text{x}} \quad (48 \leq j \leq 63) \quad \lfloor 2^{30} \cdot \sqrt{5} \rfloor$$

$$c(j) = A953FD4E_{\text{x}} \quad (64 \leq j \leq 79) \quad \lfloor 2^{30} \cdot \sqrt{7} \rfloor$$

$$c'(j) = 50A28BE6_{\text{x}} \quad (0 \leq j \leq 15) \quad \lfloor 2^{30} \cdot \sqrt[3]{2} \rfloor$$

$$c'(j) = 5C4DD124_{\text{x}} \quad (16 \leq j \leq 31) \quad \lfloor 2^{30} \cdot \sqrt[3]{3} \rfloor$$

$$c'(j) = 6D703EF3_{\text{x}} \quad (32 \leq j \leq 47) \quad \lfloor 2^{30} \cdot \sqrt[3]{5} \rfloor$$

$$c'(j) = 7A6D76E9_{\text{x}} \quad (48 \leq j \leq 63) \quad \lfloor 2^{30} \cdot \sqrt[3]{7} \rfloor$$

$$c'(j) = 00000000_{\text{x}} \quad (64 \leq j \leq 79)$$

selection of message word

$$r(j) = j \quad (0 \leq j \leq 15)$$

$$r(16..31) = 7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8$$

$$r(32..47) = 3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12$$

$$r(48..63) = 1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2$$

$$r(64..79) = 4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13$$

$$r'(0..15) = 5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12$$

$$r'(16..31) = 6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2$$

$$r'(32..47) = 15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13$$

$$r'(48..63) = 8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14$$

$$r'(64..79) = 12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11$$

amount for rotate left (rol)

$$s(0..15) = 11, 14, 15, 12, 5, 8, 7, 9, 11, 13, 14, 15, 6, 7, 9, 8$$

$$s(16..31) = 7, 6, 8, 13, 11, 9, 7, 15, 7, 12, 15, 9, 11, 7, 13, 12$$

$$s(32..47) = 11, 13, 6, 7, 14, 9, 13, 15, 14, 8, 13, 6, 5, 12, 7, 5$$

$$s(48..63) = 11, 12, 14, 15, 14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12$$

$$s(64..79) = 9, 15, 5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6$$

$$s'(0..15) = 8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14, 14, 12, 6$$

$$s'(16..31) = 9, 13, 15, 7, 12, 8, 9, 11, 7, 7, 12, 7, 6, 15, 13, 11$$

$$s'(32..47) = 9, 7, 15, 11, 8, 6, 6, 14, 12, 13, 5, 14, 13, 13, 7, 5$$

$$s'(48..63) = 15, 5, 8, 11, 14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8$$

$$s'(64..79) = 8, 5, 12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11$$

It is assumed that the message after padding consists of n 16-word blocks that will be denoted with $M_i[j]$, with $1 \leq i \leq n$ and $0 \leq j \leq 15$. The key used and the MAC value obtained consist of five words each, respectively $(K_0, K_1, K_2, K_3, K_4)$ and $(E_0, E_1, E_2, E_3, E_4)$. The symbols $\boxplus$ and $\boxminus$ denote respectively addition and subtraction modulo 2^{32} ; rol_s denotes cyclic left shift (rotate) over s positions. The pseudo-code for TTMAC is then given below.

TTMAC: pseudo-code

```

 $C_0 := K_0; C_1 := K_1; C_2 := K_2; C_3 := K_3; C_4 := K_4;$ 
 $D_0 := K_0; D_1 := K_1; D_2 := K_2; D_3 := K_3; D_4 := K_4;$ 
for  $i := 1$  to  $n$  {
   $A_0 := C_0; A_1 := C_1; A_2 := C_2; A_3 := C_3; A_4 := C_4;$ 
   $B_0 := D_0; B_1 := D_1; B_2 := D_2; B_3 := D_3; B_4 := D_4;$ 
  if ( $i \neq n$ ) for  $j := 0$  to 79 {
     $T := \text{rol}_{s(j)}(A_0 \boxplus f(j, A_1, A_2, A_3) \boxplus M_i[r(j)] \boxplus c(j)) \boxplus A_4;$ 
     $A_0 := A_4; A_4 := A_3; A_3 := \text{rol}_{10}(A_2); A_2 := A_1; A_1 := T;$ 
     $T := \text{rol}_{s'(j)}(B_0 \boxplus f(79 - j, B_1, B_2, B_3) \boxplus M_i[r'(j)] \boxplus c'(j)) \boxplus B_4;$ 
     $B_0 := B_4; B_4 := B_3; B_3 := \text{rol}_{10}(B_2); B_2 := B_1; B_1 := T;$ 
  }
  else for  $j := 0$  to 79 {
     $T := \text{rol}_{s'(j)}(A_0 \boxplus f(79 - j, A_1, A_2, A_3) \boxplus M_i[r'(j)] \boxplus c'(j)) \boxplus A_4;$ 
     $A_0 := A_4; A_4 := A_3; A_3 := \text{rol}_{10}(A_2); A_2 := A_1; A_1 := T;$ 
     $T := \text{rol}_{s(j)}(B_0 \boxplus f(j, B_1, B_2, B_3) \boxplus M_i[r(j)] \boxplus c(j)) \boxplus B_4;$ 
     $B_0 := B_4; B_4 := B_3; B_3 := \text{rol}_{10}(B_2); B_2 := B_1; B_1 := T;$ 
  }
   $A_0 := A_0 \boxminus C_0; A_1 := A_1 \boxminus C_1; A_2 := A_2 \boxminus C_2; A_3 := A_3 \boxminus C_3;$ 
   $A_4 := A_4 \boxminus C_4;$ 
   $B_0 := B_0 \boxminus D_0; B_1 := B_1 \boxminus D_1; B_2 := B_2 \boxminus D_2; B_3 := B_3 \boxminus D_3;$ 
   $B_4 := B_4 \boxminus D_4;$ 
  if ( $i \neq n$ ) {
     $C_2 := A_3 \boxminus B_0; C_3 := A_4 \boxminus B_1; C_4 := A_0 \boxminus B_2; C_0 := (A_1 \boxplus A_4) \boxminus B_3;$ 
     $C_1 := A_2 \boxminus B_4;$ 
     $D_1 := (A_4 \boxplus A_2) \boxminus B_0; D_2 := A_0 \boxminus B_1; D_3 := A_1 \boxminus B_2;$ 
     $D_4 := A_2 \boxminus B_3; D_0 := A_3 \boxminus B_4;$ 
  }
}
 $E_0 := A_0 \boxminus B_0; E_1 := A_1 \boxminus B_1; E_2 := A_2 \boxminus B_2; E_3 := A_3 \boxminus B_3;$ 
 $E_4 := A_4 \boxminus B_4.$ 

```

For a short message of up to 512 bits (one 16-word block after padding), no feedback or mixing is required and the following simplified pseudo-code can be used.

TTMAC (one message block): pseudo-code

$A_0 := K_0; A_1 := K_1; A_2 := K_2; A_3 := K_3; A_4 := K_4;$
 $B_0 := K_0; B_1 := K_1; B_2 := K_2; B_3 := K_3; B_4 := K_4;$
 for $j := 0$ to 79 {
 $T := \text{rol}_{s'(j)}(A_0 \boxplus f(79 - j, A_1, A_2, A_3) \boxplus M[r'(j)] \boxplus c'(j)) \boxplus A_4;$
 $A_0 := A_4; A_4 := A_3; A_3 := \text{rol}_{10}(A_2); A_2 := A_1; A_1 := T;$
 $T := \text{rol}_{s(j)}(B_0 \boxplus f(j, B_1, B_2, B_3) \boxplus M[r(j)] \boxplus c(j)) \boxplus B_4;$
 $B_0 := B_4; B_4 := B_3; B_3 := \text{rol}_{10}(B_2); B_2 := B_1; B_1 := T;$
 }
 $E_0 := A_0 \boxminus B_0; E_1 := A_1 \boxminus B_1; E_2 := A_2 \boxminus B_2; E_3 := A_3 \boxminus B_3;$
 $E_4 := A_4 \boxminus B_4.$

References

1. Mihir Bellare, Ran Canetti and Hugo Krawczyk, "Keying hash functions for message authentication," *Proc. Crypto'96, LNCS 1109*, Springer-Verlag, 1996, pp 1-15.
2. Hans Dobbertin, Antoon Bosselaers and Bart Preneel, "RIPEMD-160: a strengthened version of RIPEMD," *Fast Software Encryption, LNCS 1039*, Springer-Verlag, 1996, pp 71-82.
3. NESSIE Project – New European Schemes for Signatures, Integrity and Encryption – home page: <http://cryptonessie.org>.
4. Bart Preneel and Paul van Oorschot, "MDx-MAC and building fast MAC's from hash functions," *Proc. Crypto'95, LNCS 963*, Springer-Verlag, 1995, pp 1-14.
5. Bart Preneel and Paul van Oorschot, "On the security of two MAC algorithms," *Proc. Eurocrypt'96, LNCS 1070*, Springer-Verlag, 1996, pp 19-32.
6. Gene Tsudik, "Message authentication with one-way hash functions," *ACM Computer Communications Review*, Vol. 22, No. 5, 1992, pp. 29-38.