Tail-MAC: A Message Authentication Scheme for Stream Ciphers

Bartosz Zoltak

http://www.vmpcfunction.com
bzoltak@vmpcfunction.com

Abstract. Tail-MAC, A predecessor to the VMPC-MAC, algorithm for computing Message Authentication Codes for stream ciphers is described along with the analysis of its security. The proposed algorithm was designed to employ some of the data already computed by the underlying stream cipher in the purpose of minimizing the computational cost of the operations required by the MAC algorithm. The performed analyses indicate several problems with the security of the scheme and lead to a new design which described in a paper "VMPC-MAC: A Stream Cipher Based Authenticated Encryption Scheme". The new scheme solves all the problems found at a cost of some compromise in the performance.

Keywords: authenticated encryption, MAC, stream cipher, VMPC

1 Introduction

The interest in message authentication algorithms has been concentrated mostly on modes of operation of block ciphers in the past few years. Examples of some recent designs include OCB [4], OMAC [7], XCBC [6], EAX [8], CWC [9]. Parallely a growing interest in stream cipher design can be observed, however there appears to be a shortage of message authentication schemes for these algorithms. Two recent proposals in this area include Helix and Sober-128 stream ciphers with built-in MAC functionality. However a powerful attack against the MAC algorithm of Sober-128 [10] and two weaknesses of Helix [12] have been recently presented at FSE'04. This motivates that the field of message authentication schemes for stream ciphers is open for further investigation.

The approach here proposed was designed to minimize the computational cost of the additional-to-encryption MAC-related operations by employing some of the internal-state data, already computed by the cipher. This approach allows to achieve both simplicity of the design and high performance in software implementations.

Sections 2-4 discuss the general features and definitions of the Tail-MAC scheme, Section 5 outlines the VMPC stream cipher, sections 6-8 present an example application of the Tail-MAC integrated with VMPC stream cipher along with test vectors and performance rates and sections 9-10 discuss the security features of the Tail-MAC scheme.

2 General characteristics of the Tail-MAC scheme

The scheme is based on an internal state being transformed along with the progress of the encryption process in a manner determined by ciphertext-, key- and Initialization Vector (IV)-derived data.

The details of the implementation of the scheme, integrated with a selected stream cipher, will vary depending on what cipher is chosen. This results from the fact that the scheme employs some of the cipher-specific internal-state-data. This feature, when applied carefully, allows to significantly reduce the excessive computational work required by the MAC algorithm besides the work already performed by the underlying cipher.

Such approach results in the simplification of the complete authenticated encryption process, obtaining high efficiency in software implementations and, according to the further discussed analyses, ensuring a sufficient level of security.

3 Definition of a d-level Tail-MAC scheme

The definition assumes existence of a *cipher* generating a stream of b-bit words.

The internal state of the scheme consists of variables T and x_w:

T: $(8 \times d)$-element table of b-bit words. Let $T[n]$ denote n-th element of T.

$x_1, x_2, \ldots, x_d$: b-bit variables. Let $[x_1, x_2, \ldots, x_d]$ denote a $(b \times d)$-bit word combining $x_1, x_2, \ldots, x_d$.

Let f denote a bijective function and $iK(m)$ denote a part of the internal state of the *cipher* in time m; f and iK will be specified differently, depending on what cipher the Tail-MAC scheme is integrated with. Section 6 gives a proposed f function and $iK(m)$ for the VMPC stream cipher.

Let h denote a function combining T with the cryptographic key (K), with the message-unique Initialization Vector (V) and compressing them into a tag (MAC); h will vary depending on what cipher is chosen to be integrated with the Tail-MAC.

Let $g, m; i, j; q, r$ be temporary variables
Let $Pt[m]$ denote m-th b-bit word of plaintext
Let $Ct[m]$ denote m-th b-bit word of ciphertext

Table 1. Tail-MAC scheme

1. Set g, m, r to 0; Set q to 1 2. Encrypt $Pt[m]$ and store output in $Ct[m]$ 3. For j from d down to 2: execute step 4: *[1]   4. $x_j = (x_j + x_{j-1})$ modulo 2^b 5. $x_1 = f(x_1, iK(m), Ct[m] \times q)$ 6. xor $T[g, g+1, \ldots, g+d-1]$ with $[x_1, x_2, \ldots, x_d]$ *[2] 7. $g = (g + d)$ modulo $(8 \times d)$ 8. $r = r + 1 - q$ 9. if ($r \geq$ (*size of MAC in words*)) and ($g = 0$): Go to step 14 10. Increment m 11. If m is lower than total number of plaintext words: Go to step 2 12. Set q to 0 13. Go to step 3 14. $MAC = h(K, V, T)$
*[1] In the following $(d-1)$ steps j will take on values $d, (d-1), (d-2), \ldots, 2$. *[2] xor $T[g]$ with x_1; xor $T[g+1]$ with x_2; ...; xor $T[g+d-1]$ with x_d.

4 General security features of the scheme

The primary goal of the Tail-MAC scheme is to keep a sufficiently long record of the information derived from ciphertext-, key- and IV-data in the *tail* comprising a set of variables $x_1, x_2, \ldots, x_d$ and mark the T table with the *tail* in an extent sufficient to make it infeasible to change the ciphertext in a way which could produce collisions in the T table.

A mechanism of propagating any change of the ciphertext onto the elements of the T table is ensured by the fact that f is bijective, followed by the addition operations as they are performed on the *tail*-variables and further by the post-processing phase (the final iterations, executed for q=0). Because f is bijective, a change of one ciphertext word ($Ct[m]$) will cause an unconditional change in x_1. The addition operations performed on the *tail*-variables will unconditionally change the values of x_2 in iteration $m+1$, x_3 in iteration $m+2$,... and x_d in iteration $m+d-1$. In each of the iterations d elements of the T table will be updated by the values of $x_1, x_2, \ldots, x_d$. This mechanism shows in further analyses to provide a propagation of changes of the ciphertext onto T sufficient to thwart attacks aimed at obtaining collisions in the T table. The general purpose of the post-processing phase is to magnify the propagation effect. A more detailed analysis of the role of this phase can be found in Section 9.

Selection of iK and the f function should be carried out in such way as to ensure corruption of any predictable patterns of the ciphertext which could be conveyed onto the *tail* and successively onto the T table in a chosen ciphertext attack. iK and f should be based on secret non-linear parameters and can be derived from the selected parts of the internal state of the underlying stream cipher. The problem of the choice of iK and f is closely related to the characteristics of a given cipher and for the sake of clarity it will be discussed on a specific example found in Section 6, where a concrete Tail-MAC based authenticated encryption scheme is described.

The value of the d parameter, defining the length of the *tail*, indirectly implies the computational effort which would be required to forge the MAC in a chosen ciphertext attack. Further analysis of the scheme suggests that $d = 4$ provides a well sufficient security level (analysed in detail in sections 9 and 10) and enables a comfortable implementation of the system.

Applying the compression function h as the final step of the process (h can be based either on the internals of the employed cipher or on a hash-function) is aimed primarily at preventing a possible leakage of key-information. A dedicated analysis of this aspect is required for a specific implementation of the Tail-MAC scheme with a selected encryption algorithm. Further discussion on the choice of h will follow in Section 9.

The purpose of this work is not to give a set of complete and universal criteria for selecting the iK, f and h functions, but rather to outline the general requirements and the roles of each of the components and leave the choice and analysis of specific implementations to the designers. The Tail-MAC scheme is presented in this section as a general approach with general security requirements for its components, while in sections 6-10 a specific Tail-MAC based scheme employing the VMPC stream cipher is described along with a detailed discussion of its security.

5 Description of the VMPC stream cipher and its KSA

VMPC was introduced at FSE 2004 as a simple and software-efficient stream cipher with a specified Key Scheduling Algorithm (KSA) and Initialization Vector management routine. The internals of VMPC can be comfortably employed to construct an efficient authenticated encryption system based on the Tail-MAC scheme. Following [13], the VMPC stream cipher generates a stream of 8-bit words, as specified in Table 2.

Variables:

P : 256-byte table storing a permutation initialized by the VMPC KSA
s : 8-bit variable initialized by the VMPC KSA
n : 8-bit variable
L : desired length of the keystream in bytes
$+$: addition modulo 256

Table 2. VMPC stream cipher

1. $n = 0$
2. Repeat steps 3-6 L times:
3. $s = P[s + P[n]]$
4. Output $P[P[P[s]] + 1]$
5. $Temp = P[n]$
$P[n] = P[s]$
$P[s] = Temp$
6. $n = n + 1$

The VMPC Key Scheduling Algorithm (Table 3) transforms a cryptographic key (K) and (optionally) an Initialization Vector (V) into the 256-element permutation P and initializes variable s.

Variables as for VMPC stream cipher, with:

c : fixed length of the cryptographic key in bytes, $16 \leq c \leq 64$
K : c-element table storing the cryptographic key
z : fixed length of the Initialization Vector in bytes, $16 \leq z \leq 64$
V : z-element table storing the Initialization Vector
m : 16-bit variable
$+$: addition modulo 256 According to [13] there are no known security problems

Table 3. VMPC Key Scheduling Algorithm

1. $s = 0$
2. for n from 0 to 255: $P[n] = n$
3. for m from 0 to 767: execute steps 4-6:
4. $n = m$ modulo 256
5. $s = P[s + P[n] + K[m \text{ modulo } c]]$
6. $Temp = P[n]$
$P[n] = P[s]$
$P[s] = Temp$
7. If Initialization Vector is used: execute step 8:
8. for m from 0 to 767: execute steps 9-11:
9. $n = m$ modulo 256
10. $s = P[s + P[n] + V[m \text{ modulo } z]]$
11. $Temp = P[n]$
$P[n] = P[s]$
$P[s] = Temp$

regarding this cipher and its KSA. The cipher is claimed to generate keystreams undistinguishable from a truly random source and to have a number of security advantages over the popular RC4 keystream generator. The KSA of VMPC is reported to provide an undistinguishable from random diffusion of changes of one bit or byte of the cryptographic key of size up to 64 bytes onto the generated P permutation and onto output generated by the cipher. The algorithm is claimed to perform at a rate of about 12.7 clock-cycles per byte on a Pentium 4 processor. These features make the described cipher a plausible candidate to illustrate a practical application of the Tail-MAC scheme on.

The diffusion effect of the KSA has important consequences for the design of the h compression function of the further analysed VMPC-Tail-MAC scheme. Section 5.1. outlines the analyses of this aspect of the KSA following [13].

5.1 Diffusion effect of the VMPC KSA

The VMPC Key Scheduling Algorithm was tested for diffusion of changes of the cryptographic key onto the generated permutation and onto the VMPC cipher's output. A change of one byte of the cryptographic key of size 128, 256 and 512 bits appears to cause an undistinguishable from random change in the generated permutation and in the cipher's output.

In other words the analyses indicate that relations between two permutations or two keystreams generated from keys K and K' differing in only one bit or byte will be undistinguishable from relations between, accordingly, two random permutations or two random data-streams.

The KSA was designed to provide the diffusion without the use of the Initialization Vector and the tests were run without the IV (only steps 1-6 of the KSA (Table 3) were used in the tests).

Given numbers of equal permutation elements probabilities Frequencies of occurrence of situations where in two permutations, generated from keys differing in one byte, there occurs a given number (0, 1, 2, 3, 4, 5) of equal elements in the corresponding positions and the average number of equal elements in the corresponding positions – showed no statistically significant deviation from their expected values in samples of $2^{33.2}$ pairs of 128-, 256- and 512-bit keys.

Given numbers of equal Cipher's outputs probabilities Frequencies of occurrence of situations where in two 256-byte streams generated by the VMPC stream cipher directly after running the KSA for keys differing in one byte, there occurs a given number (0, 1, 2, 3, 4, 5) of equal values in the corresponding byte-positions and the average number of equal values in the corresponding byte-positions – showed no statistically significant deviation from their expected values in samples of $2^{33.2}$ pairs of 128-, 256- and 512-bit keys.

Equal corresponding permutation elements probabilities Frequencies of occurrence of situations where the elements in the corresponding positions of permutations generated from keys differing in one byte are equal (for each of the 256 positions) – showed no statistically significant deviation from their expected value in samples of $2^{33.2}$ pairs of 128-, 256- and 512-bit keys.

6 A d-level VMPC-Tail-MAC scheme

This section describes a specific scheme, where the Tail-MAC is integrated with the VMPC stream cipher. The security of this particular scheme is investigated in Section 9 and 10.

Let $x_1, x_2, \ldots, x_d$, T, Pt, Ct, m, g, q be defined as in Section 3

Let P, s, n, L, V, z be defined as in Section 5

Let "(+)" denote addition modulo 256

Let $b = 8$ and $d = 4$

Table 4. VMPC-Tail-MAC scheme

1. Run the VMPC Key Scheduling Algorithm
2. Set T, $x_1, x_2, x_3, x_4, m, g, n$ to 0; Set q to 1
3. $s = P[s\ (+)\ P[n]]$
4. if $q = 1$: $Ct[m] = Pt[m]$ xor $P[P[P[s]]\ (+)\ 1]$
5. For j from 4 down to 2: execute step 6:
6. $x_j = x_j\ (+)\ x_{j-1}$
7. $x_1 = P[x_1\ (+)\ s\ (+)\ Ct[m] \times q]$
8. For j from 0 to 3: execute step 9:
9. xor $T[g + j]$ with x_j
10. $Temp = P[n]$; $P[n] = P[s]$; $P[s] = Temp$
11. $g = (g + 4)$ modulo (8×4)
12. $n = n\ (+)\ 1$
13. Increment m
14. If $m = L$: Set q to 0
15. If $m < (L + 8 \times 3)$: Go to step 3
16.1. Store table T in table V
16.2. Set z to (8×4)
16.3. Execute step 8 of the VMPC Key Scheduling Algorithm (Table 3)
17.1. Set L to 20
17.2. Execute steps 1 and 2 of the VMPC stream cipher (Table 2) and save the 20 generated outputs as a 160-bit MAC

7 Test values of the VMPC-Tail-MAC scheme

Table 5 gives an example 20-byte tag generated by the VMPC-Tail-MAC scheme for a given 16-byte key (K), a given 16-byte Initialization Vector (V) and for a 256-byte plaintext $Message$ consisting of consecutive numbers from 0 to 255 ($Message[x] = x$ for $x \in \{0, 1, \ldots, 255\}$).

Table 5. Test vectors of VMPC-Tail-MAC

K; $c = 16$ [hex]	96, 61, 41, 0A, B7, 97, D8, A9, EB, 76, 7C, 21, 17, 2D, F6, C7
V; $z = 16$ [hex]	4B, 5C, 2F, 01, 3E, 67, F3, 95, 57, A8, D2, 6F, 3D, A2, B1, 55
$Message$ [dec]	0, 1, 2, 3, ..., 253, 254, 255
MAC [hex]	5C, 04, 54, 7E, C9, 47, 63, 62, 72, E3, C1, BB, 90, 71, 78, 1E, BA, F7, EA, F6

8 Performance of the VMPC-Tail-MAC scheme

Performance of a moderately optimized 32-bit assembler implementation of the Tail-MAC scheme integrated with the VMPC stream cipher, measured on an Intel Pentium 4, 2.66 GHz processor, is given in Table 6. Table 7 gives a performance rate of the bare Tail-MAC scheme, computed as a difference between the speed of the VMPC-Tail-MAC and the bare VMPC stream cipher.

Table 6. Performance rates of VMPC-Tail-MAC

MBytes/s	MBits/s	cycles/byte
127	1016	20.9

Table 7. Performance rates of bare Tail-MAC scheme

MBytes/s	MBits/s	cycles/byte
324	2592	8.2

9 Security of the VMPC-Tail-MAC scheme

The scheme described in Section 6 was designed to conform the general security requirements discussed in Section 4. It makes use of the diffusion effect provided by the VMPC KSA in the construction of the h function (steps 16.1-17.2) to make h corrupt any possible patterns that might occur in the T table. The h function this way magnifies the avalanche effect ensured by the post-processing phase in steps 3-15 for $q = 0$, which yields a hard to control or predict correlation of the ciphertext-, key- and IV-data with the resulting MAC.

The h function inherits the diffusion effect of the VMPC KSA and the undistinguishability from randomness of the output of the VMPC stream cipher, which provides a significant level of resistance against attempts of deducing any information about the cipher's internal state, (the P permutation), from the resulting

MAC. Even in a chosen ciphertext attack model the information about the internal state passed onto x_1 in step 7 will be corrupted by the post-processing phase and by the h function, which ensures that any information about the internal state will undergo an undistinguishable from random transformation before being revealed as the 20-byte MAC.

Constructions of the f function and $iK(m)$ (step 7) fulfill their roles described in Section 4. The f function is bijective because P is a permutation, which ensures that x_1 will undergo an unconditional change in consequence of a change of a ciphertext word, as discussed in Section 4. As a result of the pseudo-randomness, key- and IV-dependence and secrecy of the P permutation and the s variable, any possible pattern an attacker might want to convey from a chosen ciphertext onto x_1 and consecutively onto x_2, x_3, x_4 and T will be corrupted by P and s.

The scheme was intended to make it computationally unfeasible to obtain two identical T tables at any moment when processing two different messages encrypted with the same key and the same Initialization Vector. The extent of this difficulty is partly established by the value of the d parameter (here $d = 4$) determining the length of the $tail$, and magnified by the fact that T keeps record of (8×4) past values of x_1, x_2, x_3, x_4. A forgery attack would need to revert all the changes of the $x_1, \ldots, x_4$ variables and of the T table. The size of T (32 words) and a proposed length of the $tail$, $d = 4$, appear to provide a comfortable resistance to attacks aimed at changing the message in a way to revert these changes and obtain collisions in T, as discussed in detail in Section 10.

Observation of other messages encrypted with the same key and IV (in practical applications the IV should be message-unique for messages encrypted with the same key) would not give a noticeable advantage to an attacker capable of performing authenticating and verifying queries. The advantage the attacker might acquire (e.g. by trying to learn about the behavior of the f function by introducing different ciphertexts, observing the resulting MACs and trying to use this knowledge to revert the changes of $x_1, \ldots, x_4$ and T for an intercepted message or to produce a new valid message) would be practically negligible mostly because of the use of the h compression function in the final step. Following the analyses of the VMPC stream cipher quoted in Section 5, the h function produces undistinguishable from random outputs and thus corrupts any regularities of T, from observing which the attacker might benefit.

The post-processing of the T table (the final (8×3) iterations initialized in step 14) is intended to prevent possible forgery attempts through processing very short messages (one or a few bytes long), through manipulating only one or a few first or last bytes of the message or through appending or prepending attacker-chosen data to the message, which might be aimed at obtaining two messages with minor differences but producing the same T tables. The post-processing phase propagates all the changes of the message or data appended/prepended to the message onto variables x_1, x_2, x_3, x_d and onto all the (8×4) elements of the T table, which makes these changes hard to control. This effect is further magnified by the h compression function transforming T into an undistinguishable from random tag, MAC.

The choice of the number of iterations of the post-processing phase (here $8 \times 3 = 24$) can be determined by the desired length of the MAC (here 20 words). The number of post-processing iterations being greater than the length of the MAC ensures a comfortable feature, that the probability - that all inputs to the post-processing phase (all values of x_1 set in step 7) will set all the elements of the T table in a desired way (256^{-24}) - is lower than the probability of a correct random guess of the MAC (256^{-20}).

A desirable feature of an authenticated encryption scheme is a proof of security. However such proofs are mostly obtained for MAC algorithms based on primitives, which are assumed secure (block ciphers, for example in the EAX mode or hash functions, as for example in the HMAC scheme). Assuming that the underlying primitives are random functions or permutations, it is possible to build a proof of a given notion of security of the MAC algorithm.

Here the MAC algorithm is not based on such primitive, instead it reuses selected parts of the cipher's internal state to produce a tag of the encrypted message. It is rather unlikely that a formal proof of security of a similar scheme can be constructed. However the same applies to the security of any block or stream cipher, the security of which lies in the belief in the practical impossibility of finding feasible attacks, rather than in formal proofs of their security.

Following this observation we believe that the fact that Tail-MAC does not have a formal proof of its security is not a meaningful disadvantage of the scheme. This is additionally compensated by the high level of simplicity of the scheme, which may lead to accepting the Tail-MAC as a secure and easy to analyse scheme of authenticated encryption with the VMPC stream cipher or with other ciphers if such proposals appear in the future.

10 A chosen-ciphertext attack against the Tail-MAC scheme

The attack model described here is not the fastest method of breaking the Tail-MAC scheme. The new scheme VMPC-MAC (described in a paper "VMPC-MAC: A Stream Cipher Based Authenticated Encryption Scheme") , which is an evolution of the Tail-MAC, was redesigned so that no attack other than the described in this section could be applied to it, according to all the analyses which were carried out by the author.

The success probability of the attack is 2^{-144}, which can be considered a well-sufficient security level in any practical applications in the possible to predict future. In case a higher level of resistance to the discussed attack was required - it can be obtained by increasing the d parameter.

The attack assumes that the adversary has full passive and active access to the ciphertext and can use an unlimited number of verification queries for the new message. The purpose of the attacker is to introduce a new valid ciphertext (a ciphertext that was not MACed through an authenticating query, but which is deemed valid in the verification query). The most efficient attack approach found assumes that the attacker bases on a valid message (which was intercepted or obtained through an authenticating query) and attempts to change it in such way as to make the message generate the same MAC as the original message did.

The attack model begins with a random (or intended by the attacker) change of one bit (or byte) of the ciphertext - $Ct[m]$. The purpose of the attacker is to hide this change by manipulating the remaining part of the ciphertext in such way as to leave the resulting MAC unchanged.

The attack is illustrated on an example of the system described in Section 6, for $d = 4$ and $b = 8$, however analogous approach would apply for different values of d and b parameters and different choice of ciphers.

Let $x_w(m)$ denote the value of the x_w variable of the *tail* in iteration m;
$w \in \{1, 2, 3, 4\}$
Let $n = (m \text{ modulo } 8)\times 4$
Let "(+)" denote addition modulo (8×4)

A change of $Ct[m]$ unconditionally causes a change of $x_1(m)$, since P is a permutation.

Because $x_1(m)$ and only $x_1(m)$ directly updates $x_2(m+1)$ and indirectly updates $x_3(m+2)$ and $x_4(m+3)$, the variables $x_2(m+1)$, $x_3(m+2)$ and $x_4(m+3)$ will be unconditionally changed too.
The following elements of table T will be updated and unconditionally changed by those variables: $T[n]$ changed by $x_1(m)$, $T[n(+)5]$ changed by $x_2(m+1)$, $T[n(+)10]$ changed by $x_3(m+2)$ and $T[n(+)15]$ changed by $x_4(m+3)$.

The most efficient method of reverting these changes found forces the attacker to perform the following changes of the ciphertext:

1. Change $Ct[m+1]$ in such way as to make $x_4(m+4)$ return to its original value. The unavoidable cost of this is a change of $x_1(m+1)$, $x_2(m+2)$ and

$x_3(m+3)$.[1] [$x_3(m+3)$ must be changed in such way as to make $x_4(m+4) = (x_4(m+3) + x_3(m+3))$ modulo 256 return to its original value[2]].
As a result $T[n(+)4]$ is changed by $x_1(m+1)$, $T[n(+)9]$ is changed by $x_2(m+2)$ and $T[n(+)14]$ is changed by $x_3(m+3)$. $T[n(+)19]$ remains unchanged because the change of $x_4(m+4)$ was reverted.

2. Change $Ct[m+2]$ in such way as to make $x_3(m+4)$ return to its original value. The unavoidable cost of this is a change of $x_1(m+2)$ and $x_2(m+3)$.
As a result $T[n(+)8]$ is changed by $x_1(m+2)$ and $T[n(+)13]$ is changed by $x_2(m+3)$. $T[n(+)18]$ remains unchanged because the change of $x_3(m+4)$ was reverted.

3. Change $Ct[m+3]$ in such way as to make $x_2(m+4)$ return to its original value. The unavoidable cost of this is a change of $x_1(m+3)$.
As a result $T[n(+)12]$ is changed by $x_1(m+3)$. $T[n(+)17]$ remains unchanged because the change of $x_2(m+4)$ was reverted.

4. Change $Ct[m+4]$ in such way as to make $x_1(m+4)$ return to its original value. As a result $T[n(+)16]$ remains unchanged.

At this moment the attacker succeeded in stopping the avalanche of changes of elements of T, resulting from a change of $Ct[m]$, by reverting the changes of x_1, x_2, x_3, x_4 in the earliest possible iteration $m+4$. The cost of this is an unavoidable change of 10 elements of the T table ($T[n$, $n(+)4$, $n(+)5$, $n(+)8$, $n(+)9$, $n(+)10$, $n(+)12$, $n(+)13$, $n(+)14$, $n(+)15]$).

To complete a successful forgery, the attacker needs to revert the changes of these elements of T, too. Operations similar to steps 1-4 need to be performed to refrain x_1, x_2, x_3, x_4 from causing more damage to T and the additional requirement - to revert the already caused changes to T - needs to be satisfied. The most efficient approach found achieves that in the following steps 5-9:

5. Change $Ct[m+8]$ in such way as to change $x_1(m+8)$ in such way as to revert the change of $T[n]$, make $x_2(m+9)$ change in such way as to revert the change of $T[n(+)5]$, make $x_3(m+10)$ change in such way as to revert the change of $T[n(+)10]$, and make $x_4(m+11)$ change in such way as to revert the change of $T[n(+)15]$.

6. Change $Ct[m+9]$ in such way as to make $x_4(m+12)$ return to its original

[1] The algorithm can be varied into making some of the variables (e.g. $x_2(m+2)$) remain unchanged, which yields an apparent improvement, however further analysis shows that this actually leads to higher complexity of the complete attack.

[2] The approach by which the first variable to return to its original value is x_4, rather than e.g. x_1 or x_2, in further analysis shows to lead to much lower complexities of the complete attack.

value, make $x_1(m+9)$ change in such way as to revert the change of $T[n(+)4]$, make $x_2(m+10)$ change in such way as to revert the change of $T[n(+)9]$, make $x_3(m+11)$ change in such way as to revert the change of $T[n(+)14]$. $T[n(+)19]$ remains unchanged because the change of $x_4(m+12)$ was reverted.

7. Change $Ct[m+10]$ in such way as to make $x_3(m+12)$ return to its original value, make $x_1(m+10)$ change in such way as to revert the change of $T[n(+)8]$, make $x_2(m+11)$ change in such way as to revert the change of $T[n(+)13]$. $T[n(+)18]$ remains unchanged because the change of $x_3(m+12)$ was reverted.

8. Change $Ct[m+11]$ in such way as to make $x_2(m+12)$ return to its original value, make $x_1(m+11)$ change in such way as to revert the change of $T[n(+)12]$. $T[n(+)17]$ remains unchanged because the change of $x_2(m+12)$ was reverted.

9. Change $Ct[m+12]$ in such way as to make $x_1(m+12)$ return to its original value. As a result $T[n(+)16]$ remains unchanged.

The success probability of the described attack is determined by the total number of changes to variables x_1, x_2, x_3, x_4 and $T[0, 1, \ldots, 31]$, which need to be reverted. Steps 1-9 determine this probability, for the assumed $d = 4$ and $b = 8$, to $256^{-18} = 2^{-144}$.

Extending the length of the *tail* to $d = 5$ would, in an analogous attack, yield a success probability of $256^{-25} = 2^{-200}$ (which would also imply an increase of the size of the MAC to 25 or more bytes), however the implementation of the scheme would not be as comfortable as for $d = 4$ (while still easily achievable) which, given the fact that 2^{-144} is a well out of reach security level, encourages to propose $d = 4$ as sufficient for possible practical applications of the Tail-MAC scheme.

11 A 2^{-32} break of the Tail-MAC scheme

The main attack finds collisions in T. It asumes appending one byte to the cipherptext message and relies on the fact that the post-processing phase and the encryption phase of the MAC algorithm are equivalent (in a sense that they generate the same output from a given input) and on the fact that the probability that x1,x2,x3,x4 will not change T in the last step of processing the one-byte-longer-message is 2^{-32}.

The solution to this problem is introducing a variable R in the post-processing phase in the new VMPC-MAC scheme.

The other problem with the Tail-MAC is that if for a new message x1 grows by 128 (or any other integer D=256/integer), then the propagation of changes of the variables x1,x2,x3,x4 is significantly limited due to their linearity (only the addition operation x(n)=x(n)+x(n-1)). As a result in even numbers of overlapping loops the x variables do not change at all since the addition is modulo 256 (128+128 mod 256=0). This significantly limits the number of elements of T which are affected by the change of the one (or more) bytes of the message.

The solution to this problem is modification of step 6 of the scheme from $x_j = x_j\ (+)\ x_{j-1}$ into $x_j = P[x_j\ (+)\ x_{j-1}]$. The P permutation corrupts the linearity of the addition operation and avoids the problem.

12 Conclusions

This paper presents a predecessor to the VMPC-MAC scheme, which is a Message Authentication Scheme dedicated only to the VMPC Stream Cipher. The aim of the more specialised approach applied in the new scheme was to gain clarity. Security analysis of a MAC scheme based on a stream cipher's internal state without actually defining the cipher, as was the primary goal of the Tail-MAC, is too much of a hypothetical approach to provide it with a proper security analysis in the belief of the author.

The new paper "VMPC-MAC: A Stream Cipher Based Authenticated Encryption Scheme" is available at the ePrint archive of year 2004 and also at the VMPC website at http://www.VMPCfunction.com.

References

1. Federal Information Processing Standards Publication 198: The Keyed-Hash Message Authentication Code (HMAC), 2002 http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf
2. Mihir Bellare, Ran Canetti, Hugo Krawczyk: Message Authentication using Hash Functions the HMAC Construction, CryptoBytes, Vol 2, No. 1, RSA Laboratories, 1996
3. Mihir Bellare, Chanathip Namprempre: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm, Proceedings of ASIACRYPT 2000, LNCS vol. 1976 Springer-Verlag, 2000
4. Phillip Rogaway, Mihir Bellare, John Black, Ted Krovetz: OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption (2001), Eighth ACM Conference on Computer and Communications Security (CCS-8) (August 2001), ACM Press.
5. Mihir Bellare, Roch Guerin, Philip Rogaway: XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions, Proceedings of CRYPTO 1995, LNCS vol. 963, Springer-Verlag, 1995.
6. V. Gligor, P. Donescu: Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes, 2nd NIST Workshop on AES Modes of Operation, Santa Barbara, USA, 2001.
7. T. Iwata, K. Kurosawa: OMAC: One-key CBC MAC, Proceedings of Fast Software Encryption 2003, LNCS vol. 2887, Springer-Verlag 2003.
8. Mihir Bellare, Philip Rogaway, David Wagner: The EAX Mode of Operation Pre-proceedings of Fast Software Encryption 2004, pages 367-384.
9. Tadayoshi Kohno, John Viega, Doug Whiting: CWC: A High-Performance Conventional Authenticated Encryption Mode, Pre-proceedings of Fast Software Encryption 2004, pages 385-402.
10. Dai Watanabe, Soichi Furuya: A MAC forgery attack on SOBER-128, Pre-proceedings of Fast Software Encryption (FSE) 2004, pages 448-458.
11. Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, Tadayoshi Kohno: Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive, Proceedings of FSE 2003, LNCS vol. 2887, Springer-Verlag 2003.
12. Fredric Muller: Differential Attacks Against the Helix Stream Cipher, Pre-proceedings of Fast Software Encryption 2004, pages 75-88.
13. Bartosz Zoltak: VMPC One-Way Function and Stream Cipher, Pre-proceedings of Fast Software Encryption 2004, pages 190-204.
14. NESSIE consortium: Performance of Optimized Implementations of the NESSIE Primitives, 2003 www.cryptonessie.org
15. Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, Sven Verdoolaege: Analysis Methods for (Alleged) RC4. Proceedings of ASIACRYPT 1998, LNCS, vol. 1514, Springer-Verlag, 1998.
16. Scott R. Fluhrer, David A. McGrew: Statistical Analysis of the Alleged RC4 Keystream Generator. Proceedings of FSE 2000, LNCS, vol. 1978, Springer-Verlag, 2001.
17. Itsik Mantin, Adi Shamir: A Practical Attack on Broadcast RC4. Proceedings of FSE 2001, LNCS, vol. 2355, Springer-Verlag, 2002.
18. Scott Fluhrer, Itsik Mantin, Adi Shamir: Weaknesses in the Key Scheduling Algorithm of RC4. Proceedings of SAC 2001, LNCS, vol. 2259, Springer-Verlag 2001.
19. Jovan Dj. Golic: Linear Statistical Weakness of Alleged RC4 Keystream Generator. Proceedings of EUROCRYPT 1997, LNCS, vol. 1233, Springer-Verlag 1997.