A new MAC Construction ALRED and a Specific Instance ALPHA-MAC

Joan Daemen[1] and Vincent Rijmen[2,3⋆]

[1] STMicroelectronics Belgium
joan.daemen@st.com
[2] IAIK, Graz University of Technology
vincent.rijmen@iaik.tugraz.at
[3] Cryptomathic A/S

Abstract. We present a new way to construct a MAC function based on a block cipher. We apply this construction to AES resulting in a MAC function that is a factor 2.5 more efficient than CBC-MAC with AES, while providing a comparable claimed security level.

1 Introduction

Message Authentication Codes (MAC functions) are symmetric primitives, used to ensure authenticity of messages. They take as input a secret key and the message, and produce as output a short tag.

Basically, there are three approaches for designing MACs. The first approach is to design a new primitive from scratch, as for instance MAA [15] and, more recently, UMAC [8]. This approach allows to optimize the security-performance trade-off. The second approach is to define a new *mode of operation* for existing primitives. In this category, we firstly have numerous variants based on the CBC encryption mode for block ciphers, e.g. CBC-MAC [5], OMAC [16], TMAC [22], XCBC [9], and RMAC [17]. Secondly, there are the designs based on an unkeyed hash function: NMAC, HMAC [7, 4]. Finally, one can design new MACs using components from existing primitives, e.g. MDx-MAC [24] and Two-Track MAC [10].

In this paper, we propose a new MAC design method which belongs in the third category, cf. Section 3. We also present a concrete construction in Section 5. Before going there, we start with a discussion of security requirements for MACs and we present a new proposal for MAC security claims in Section 2. We discuss internal collisions for the new model in Section 4, and for the concrete construction in Section 6. Section 7 contains more details on extinguishing differentials, a special case of internal collisions. We briefly discuss performance in Section 8 and conclude in Section 9.

⋆ This researcher was supported financially by the A-SIT, Austria

2 Iterative MAC functions and security claims

A MAC function maps a key-message pair to a tag. The basic property of a MAC function is that it provides an unpredictable mapping between messages and the tag for someone who does not know, or only partially knows the key. Usually, one defines a number of design objectives that a cryptographic primitive of a given type must satisfy in order to be considered as secure. For MAC functions, we find the following design objectives in [23, Table 9.2]:

- *Key non-recovery*: the expected complexity of any key recovery attack is of the order of 2^{ℓ_k} MAC function executions.
- *Computation resistance*: there is no forgery attack with probability of success above $\max(2^{-\ell_k}, 2^{-\ell_m})$.

Here ℓ_k is the key length and ℓ_m the tag length. By forgery one means the generation of a message-tag pair (m, t) using only information on pairs (m_i, t_i) with $m \neq m_i$ for all i.

2.1 Iterative MAC functions

Most practical MAC functions are *iterative*. An iterative MAC function operates on a working variable, called the *state*. The message is split up in a sequence of message blocks and after a (possibly keyed) initialization the message blocks are sequentially injected into the state by a (possibly keyed) iteration function. Then a (possibly keyed) final transformation may be applied to the state resulting in the tag.

Iterative MAC functions can be implemented in hardware or software with limited amount of working memory, irrespective of the length of the input messages. They have the disadvantage that different messages may be found that lead to the same value of the state before the final transformation. This is called an *internal collision* [26].

2.2 Internal collisions

Internal collisions can be used to perform forgery. Assume we have two messages m_1 and m_2 that result in an internal collision. Then for any string m_3 the two messages $m_1\|m_3$ and $m_2\|m_3$ have the same tag value. So given the tag of any message $m_1\|m_3$, one can forge the tag of the message $m_2\|m_3$. Internal collisions can often be used to speed up key recovery as well [25]. If the number of bits in the state is n, finding an internal collision takes at most $2^n + 1$ known pairs. If the state transformation can be modeled by a random transformation, one can expect to find a collision with about $2^{n/2}$ known pairs due to the birthday paradox. One may consider to have a final transformation that is not reversible to make the detection of internal collisions infeasible. However, as described in Appendix A, this is impossible.

The presence of internal collisions makes that even the best iterative MAC function cannot fulfill the design objectives given above: if the key is used to

generate tags over a very large number of messages, an internal collision is likely to occur and forgery is easy.

For many MAC schemes based on CBC-MAC with the DES as underlying block cipher, internal collisions can be used to retrieve the key: the ISO 9797 [5] schemes are broken in [11, 18]. More sophisticated variants like Retail MAC [1] and MacDES [19] are broken in [25], respectively [12, 13].

One approach to avoid the upper limit due to the birthday paradox in iterative MAC functions is *diversification*. The MAC function has next to the message and the key a third input parameter that serves to diversify the MAC computation to make the detection of internal collisions impossible. For the proofs of security that accompany these schemes, the implementation of a tag generating device must impose that its value is non-repeating or random. This method has several important drawbacks. First of all, not only the tag must be sent along with the message, but also this third parameter, typically doubling the overhead. In case of a random value, this puts the burden on the developer of the tag generating device to implement a cryptographic random generator, which is a non-trivial task. Moreover the workload of generating the random value should be taken into account in the performance evaluation of the primitive. In case of a non-repeating value the MAC function becomes stateful, i.e., the tag generation device must keep track of a counter or multiple counters and guarantee that these counters cannot be reset. But in some cases even the randomization mechanism itself introduces subtle flaws. The best known example of a randomized MAC is RMAC [17] cryptanalyzed in [21].

Another way to avoid internal collisions is to impose an upper bound on the number of tags that may be generated with a given key. If this upper bound is large enough it does not impose restrictions in actual applications. This is the approach we have adopted in this paper.

2.3 A proposal for new security claims

We formulate a set of three orthogonal security claims that an iterative MAC function should satisfy to be called secure.

Claim 1 *The probability of success of any forgery attack not involving key recovery or internal collisions is* $2^{-\ell_m}$.

Claim 2 *There are no key recovery attacks faster than exhaustive key search, i.e. with an expected complexity less than* 2^{ℓ_k-1} *MAC function executions.*

We model the effect of internal collisions by a third dimension parameter, the *capacity* ℓ_c. The capacity is the size of the internal memory of a random map with the same probability for internal collisions as the MAC function.

Claim 3 *The probability that an internal collision occurs in a set of* A *((adaptively) chosen message, tag) pairs, with* $A < 2^{\ell_c/2}$*, is not above* $1-\exp(-A^2/2^{\ell_c+1})$.

Note that for $A < 1/4 \times 2^{\ell_c/2}$ we have $1 - \exp(-A^2/2^{\ell_c+1}) \approx A^2/2^{\ell_c+1}$. In the best case the capacity ℓ_c is equal to the number of bits of the state. It is up to the designer to fix the value of the capacity ℓ_c used in the security claim.

3 The ALRED construction

We describe here a way to construct a MAC function based on an iterated block cipher. The key length of the resulting MAC function is equal to that of the underlying block cipher, the length of the message must be a multiple of ℓ_w bits, where ℓ_w is a characteristic of a component in the MAC function. In our presentation, we use the term *message word* to indicate ℓ_w-bit message blocks and call ℓ_w the *word length*.

3.1 Definition

The ALRED construction consists of a number of steps:

1. Initialization:
   (a) Initialize the state with the all-zero block.
   (b) Apply the *block cipher* to the state.
2. Chaining: for each message word perform an iteration:
   (a) Map the bits of the message word to an *injection input* that has the same dimensions as a sequence of r round keys of the block cipher. We call this mapping the *injection layout*.
   (b) Apply a sequence of r *block cipher round functions* to the state, with the injection input taking the place of the round keys.
3. Final transformation:
   (a) Apply the *block cipher* to the state.
   (b) Truncation: the tag is the first ℓ_m bits of the state.

Let the message words be denoted by x_i, the state after iteration i by y_i, the key by k and the tag by z. Let f denote the iteration function, which consists of the combination of the injection layout and the sequence of r block cipher round functions. Then we can write:

$$y_0 = \mathrm{Enc}_k(0) \tag{1}$$

$$y_i = f(y_{i-1}, x_i), \quad i = 1, 2, \ldots, q \tag{2}$$

$$z = \mathrm{Trunc}(\mathrm{Enc}_k(y_q)) \tag{3}$$

The construction is illustrated in Figure 1 for the case $r = 1$. With this approach, the design of the MAC function is limited to the choice of the block cipher, the number of rounds per iteration r, the injection layout and ℓ_m. The goal is to choose these such that the resulting MAC function fulfills the security claims for iterated MAC functions for some value of ℓ_m and ℓ_c near the block length.

3.2 Motivation

Prior to the chaining phase, the state is initialized to zero and it is transformed by applying the block cipher, resulting in a state value unknown to the attacker. In the chaining phase every iteration injects ℓ_w message bits into the state with

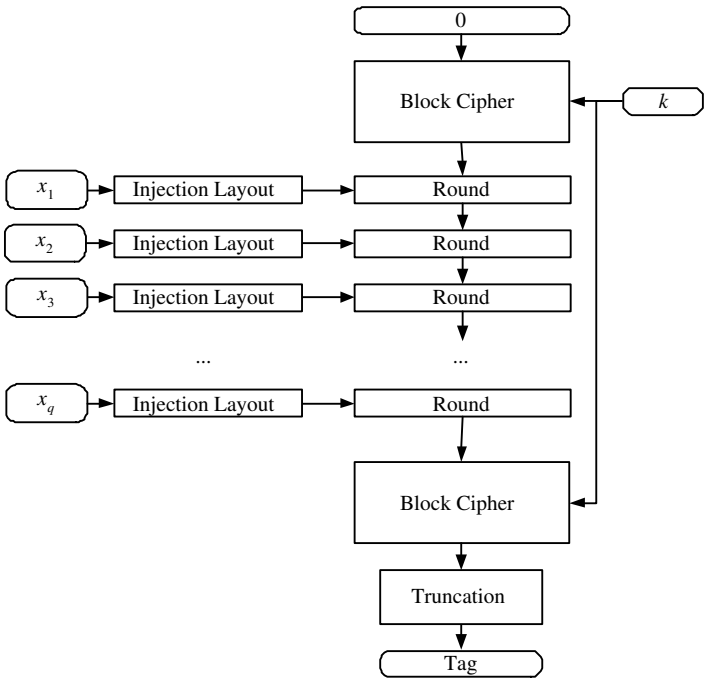

Fig. 1. Scheme of the ALRED construction with $r = 1$.

an unkeyed iteration function. Without the block cipher application in the initialization, generating an internal collision would be similar to finding a collision in an unkeyed hash function which can be conducted without known message tag pairs. The initial block cipher application makes the difference propagation through the chaining phase, with its nonlinear iteration function, depend on the key.

The iteration function consists of r block cipher rounds where message word bits are mapped onto the round key inputs. The computational efficiency of ALRED depends on the word length. Where in CBC based constructions for long messages there is one block cipher execution per block, ALRED takes merely r rounds per word. Clearly, the performance of ALRED becomes interesting if the word length divided by r is larger than the block length divided by the number of rounds of the block cipher.

Decreasing the message word length with respect to the round key length makes the MAC function less efficient, but also reduces the degrees of freedom available to an attacker to generate internal collisions (see Section 6.1 for an example). Another way to reduce these degrees of freedom is to have the message words first undergo a message schedule, and apply the result as round keys. This is similar to the key schedule in a block cipher, the permutation of message

words between the rounds in MD4 [27] or the message expansion in SHA-1 [2]. However, such a message schedule also introduces need for additional memory and additional workload per iteration. Therefore, and for reasons of simplicity, we decided to limit the message injection to a simple layout. Limiting the word length and carefully choosing the injection layout allows to demonstrate powerful upper bounds on the probability of sets of known or chosen messages with any chosen difference leading to an internal collision.

3.3 Provability

ALRED has some similarity to constructions based on block ciphers in CBC mode. The modes typically come with security proofs that make abstraction of the internal structure of the used cryptographic primitive. In this section we prove that an ALRED MAC function is as strong as the underlying block cipher with respect to key recovery and, in the absence of internal collisions, is resistant against forgery if the block cipher is resistant against ciphertext guessing.
Observation: The proofs we give are valid for any chaining phase that transforms y_0 into y_{final} parameterized by a message. In the proofs we denote this by $y_{\text{final}} = F_{\text{cf}}(y_0, m)$.

Theorem 1. *Every key recovery attack on* ALRED*, requiring t (adaptively) chosen messages, can be converted to a key recovery attack on the underlying block cipher, requiring $t + 1$ adaptively chosen plaintexts.*

Proof: Let $\mathcal{A}$ be an attack requiring the t tag values corresponding to the t (adaptively) chosen messages m_j, yielding the key. Then, the attack on the underlying block cipher works as follows.

1. Request $c_0 = \text{Enc}(k, 0)$, where '0' denotes the all-zero plaintext block.
2. For $j = 1$ to t, compute $p_j = F_{\text{cf}}(c_0, m_j)$.
3. For $j = 1$ to t, request $c_j = \text{Enc}(k, p_j)$.
4. Input the tag values $\text{Trunc}(c_j)$ to $\mathcal{A}$ and obtain the key.

□

Theorem 2. *Every forgery attack on* ALRED *not involving internal collisions, requiring t (adaptively) chosen messages, can be converted to a ciphertext guessing attack on the underlying block cipher, requiring $t+1$ adaptively chosen plaintexts.*

Proof: Let $\mathcal{B}$ be an attack, not involving internal collisions, requiring the t tag values corresponding to the t (adaptively) chosen messages m_j yielding a forged tag for the message m. Then, the ciphertext guessing attack on the underlying block cipher works as follows.

1. Request $c_0 = \text{Enc}(k, 0)$, where '0' denotes the all-zero plaintext block.
2. For $j = 1$ to t, compute $p_j = F_{\text{cf}}(c_0, m_j)$.
3. For $j = 1$ to t, request $c_j = \text{Enc}(k, p_j)$.
4. Input the tag values $\text{Trunc}(c_j)$ to $\mathcal{B}$ and obtain the tag for the message m.

5. Compute $p = F_{\mathrm{cf}}(c_0, m)$.
6. If there is a j for which $p = p_j$, then $\mathcal{B}$ has generated an internal collision, which conflicts with the requirement on $\mathcal{B}$. Otherwise, input the tag values $\mathrm{Trunc}(c_j)$ to $\mathcal{B}$ and obtain the tag, yielding the truncated ciphertext of p.

□

3.4 On the choice of the cipher

One may use any block cipher in the ALRED construction. The block length imposes an upper limit to the capacity ℓ_c relevant in the number of tags that may be generated with the same key before an internal collision occurs. When using ciphers with a block length of 64 bits as (Triple) DES and IDEA, the number of tags generated with the same keys should be well below 2^{32}.

The use of the round function for building the iteration function restricts the ALRED construction somewhat. Ciphers that are well suited in this respect are (Triple) DES, IDEA, Blowfish, Square, RC6, Twofish and AES. An ALRED construction based on Serpent, with its eight different rounds, would typically have $r = 8$, with the iteration function consisting of the eight Serpent rounds. MARS with its non-uniform round structure is less suited for ALRED. The choice of the injection layout is a design exercise specific for the round function of the chosen cipher. Note that whatever the choice of the underlying cipher, the strength of the ALRED construction with respect to key search is that of the underlying cipher.

4 On internal collisions in ALRED

In general, any pair of message sequences, possibly of different length, that leads to the same value of the internal state is an internal collision. We see two approaches to exploit knowledge of the iteration function to generate internal collisions. The first is to generate pairs of messages of equal length that have a difference (with respect to some group operation at the choice of the attacker) that may result in a zero difference in the state after the difference has been injected. We call this *extinguishing differentials*. The second is to insert in a message a sequence of words that do not impact the state. We call this *fixed points*.

4.1 Extinguishing differentials

Finding high-probability extinguishing differentials is similar to differential cryptanalysis of block ciphers. In differential cryptanalysis the trick is to find an input difference that leads to an output difference with high probability. For an iterative MAC function, the trick is to find an extinguishing differential with high probability. Resistance against differential cryptanalysis is often cited as one of the main criteria for the design of the round function of block ciphers. Typically the round function is designed in such a way that upper bounds can be

demonstrated on the probability of differentials over a given number of rounds. One may design MAC functions in a similar way: design the iteration function such that upper bounds can be demonstrated on the probability of extinguishing differentials. In ALRED the only part of the iteration function to be designed is the injection layout. So the criterion for the choice of the injection layout is the upper bound on the probability of extinguishing differentials.

4.2 Fixed points

Given a message word value x_i, one may compute the number of state values that are invariant under the iteration function $y_i = f(y_{i-1}, x_i)$, called *fixed points.* If the number of fixed points is w, the probability that inserting the message word x_i in a message will not impact its tag is $w \times 2^{-n}$ with n the block length.

We can try to find the message word value $x_{\max}$ with the highest number of fixed points. If this maximum is $w_{\max}$, inserting $x_{\max}$ in a message and assuming that the resulting message has the same tag, is a forgery attack with success probability $w_{\max} \times 2^{-n}$. Since Claim 3 requires that this probability be smaller than $1 - \exp(-2^2/2^{\ell_c+1}) = 1 - \exp(-(2^{1-\ell_c})) \approx 2^{1-\ell_c}$, this imposes a limit to the capacity: $\ell_c < n + 1 - \log_2 w_{\max}$.

If the iteration function can be modeled as a random permutation, the number of fixed points has a Poisson distribution with $\lambda = 1$. The expected value of $w_{\max}$ depends on the number of different iteration functions with a given message word, i.e. the word length ℓ_w. For example, the expected $w_{\max}$ values for 16, 32, 64 and 128 bits are 8, 12, 20 and 33 respectively. However, if $r = 1$, the iteration function is just a round function of a block cipher and it may not behave as a random function in this respect. If the round function allows it, one may determine the number of fixed points for a number of message word values to determine whether the Poisson distribution applies.

One may consider the number of fixed points under a sequence of g rounds. In the random model, the expected value of $w_{\max}$ over all possible sequences of g message words now is determined by the total number of messagebits injected in the g rounds. For most round functions determining the number of fixed points given the message word values is hard even for $g = 2$. However, for multiple iterations it is very likely that the random model will hold. The value of $w_{\max}$ grows with g but actually finding message word sequences with a number of fixed points of the order $w_{\max}$ becomes quickly infeasible as g grows. If we consider a sequence of iterations taking 500 message bits (for example 10 iterations taking each 50 message bits), the expected value of $w_{\max}$ is 128. In conclusion, if analysis of the iteration function confirms that the number of fixed points has a Poisson distribution, taking $\ell_c \leq n - 8$ provides a sufficient security margin with respect to forging using fixed points.

5 ALPHA-MAC

ALPHA-MAC is an ALRED construction with AES [3] as underlying block cipher. As AES, ALPHA-MAC supports keys of 16, 24 and 32 bytes. Its iteration function

consists of a single round, its word length is 4 bytes and the injection layout places these bytes in 4 byte positions of the AES state. We have chosen AES mainly because we expect AES to be widely available thanks to its status as a standard. Additionally, AES is efficient in hardware and software and it has withstood intense public scrutiny very well since its publication as Rijndael [14].

5.1 Specification

The number of rounds per iteration r is 1 and the word length ℓ_w is equal to 32 bits. The AES round function takes as argument a 16-byte round key, represented in a 4×4 array. The injection layout positions the 4 bytes of the message word $[q_1, q_2, q_3, q_4]$ in 4 positions of this array, resulting in the following injection input:

$$\begin{bmatrix} q_1 & 0 & q_2 & 0 \\ 0 & 0 & 0 & 0 \\ q_3 & 0 & q_4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

The length of the tag ℓ_m may have any value below or equal to 128. Alpha-MAC should satisfy the security claims for iterative MAC functions for the three key lengths of AES with $\ell_m \leq 128$ and $\ell_c = 120$. Appendix B gives and equivalent description of Alpha-MAC.

5.2 Message padding

Alpha-MAC is only defined for messages with a length that is a multiple of 32 bits. One may extend Alpha-MAC to message of any length by preprocessing the message with a reversible padding scheme. We propose to use the following padding scheme: append a single 1 followed by the minimum number of 0 bits such that the length of the result is a multiple of 32. This corresponds to padding method 2 in [5].

6 Internal collisions and injection layout of Alpha-MAC

With respect to fixed points, we implemented a program that determines the number of fixed points for the Alpha-MAC iteration function. It turns out that the number of fixed points behaves as a Poisson distribution with $\lambda = 1$. The choice of the Alpha-MAC injection was however mainly guided by the analysis of extinguishing differentials, as we explain in the following subsections.

6.1 A simple attack on a variant of Alpha-MAC

Let us consider a simple extinguishing differential for a variant of Alpha-MAC with an injection layout mapping a 16 byte message block to a 16-byte round key. Assume the difference in the first message word has a single active byte with value a in position i, j (row i, column j).

– AddRoundKey (AK) injects the difference in the state giving a single active byte with value a in the state in position i, j.
– SubBytes (SB) converts this to a single active byte in position i, j with value b. Given a, b has 127 possible values. 126 of these values have probability 2^{-7} and one has probability 2^{-6}.
– ShiftRows (SR) moves the active byte with value b to position i, ℓ with $\ell = j - i \bmod 4$.
– MixColumns (MC) converts the single active byte to an active column, column ℓ. The value of the single active byte completely determines the values of bytes of the active column.

Hence a message difference with a single active byte may lead to 127 state differences before the injection of the second message word. Of these, one has probability 2^{-6} and 126 have probability 2^{-7}. Assume now that the second message word has a difference given by the active column that has probability 2^{-6}. Clearly, the probability of the resulting extinguishing differential is 2^{-6} and the expected number of message pairs that must be queried to obtain an internal collision using it is 2^6.

We can reduce the number of required messages to query by applying a set of n messages that have pairwise differences of the type described above. About half of the $n(n-1)/2$ differences are extinguishing differentials with a probability of 2^{-7} and due to the birthday paradox a set of only 20 messages are likely to generate an internal collision.

When achieving an internal collision, the fact that the active S-box converts difference a to b gives 6 or 7 bits of information on the absolute value of the state. Applying this attack for all byte positions allows the reconstruction of the state at the beginning of the iteration phase for the given key. When this is known, generating internal collisions is easy.

In the described internal collision attack, the attacker is not hampered by the injection layout. He has full liberty in positioning the differences in the injection inputs. We see that the described difference leads to an internal collision if there is a match in a single S-box: the S-box must map the difference a to difference b. This is an extinguishing characteristic with one active S-box.

For the injection layout of Alpha-MAC this attack is not possible as it requires in the second injection input a difference pattern with four active bytes in the same column. However, attacks can be mounted with other message difference patterns. The goal of the injection layout is exactly to impose that there are no extinguishing differentials with high probability and hence that there are no extinguishing characteristics with a small number of active S-boxes. Together with the implementation complexity, this has been the main criterion for the selection of the injection layout. We will treat this in the following sections.

6.2 Choosing the injection layout

In order to select the injection layout we have written a program that determines the minimum number of active S-boxes over all extinguishing truncated

Table 1. Number of injection layout equivalence classes.

Word length (in bytes)	1	2	3	4	5
Total number of layout classes	3	21	77	252	567
with minimum extinguishing cost equal to 16	3	21	68	87	0

characteristics for a given injection layout. Truncated characteristics are *clusters* of ordinary characteristics [20]. All characteristics in a cluster have intermediate state differences with active bytes in the same positions. The probability of a truncated characteristic is the sum of the probabilities of all its characteristics. Similar to ordinary characteristics, the probability of a truncated characteristic can be expressed in terms of active S-boxes, but only a subset of the S-boxes with non-zero input difference are counted as active.

Our program iteratively builds a tree with the 2^{16} possible state difference patterns (only distinguishing between 'zero' and 'non-zero' values) as nodes. The root is the all-zero pattern and each edge has an S-box cost and message difference pattern. The program builds the tree such that the minimum extinguishing cost of a pattern is the sum of the S-box costs of the edges leading to the root. It also includes the all-zero pattern as a leaf in the tree, and hence the minimum extinguishing cost is that of this leaf. Note that for any injection layout an upper bound for the minimum extinguishing cost is 16 as it is always possible to guess all bits of the state at a given time.

In total there are 2^{16} different injection layouts. With respect to this propagation analysis, they are partitioned in 8548 equivalence classes:

- Thanks to the horizontal symmetry in the AES round function injection layouts that can be mapped one to the other by means of a horizontal shift $(i, j) \mapsto (i, j + a \bmod 4)$ are equivalent.
- Injection layouts that can be mapped one to the other by means of a mirroring around $(0, 0)$, i.e. $(i, j) \mapsto (-i \bmod 4, -j \bmod 4)$, are equivalent. This is thanks to the fact that SB and SR are invariant under this transformation and the branch number of MC, the only aspect of MC relevant in this propagation analysis, is not modified.

The results for word lengths 5 and below are summarized in Table 1.

As Alpha-MAC requires one round function computation per message word, the performance is proportional to message word length. Note that the minimum extinguishing cost of an injection layout is upper bounded by those of all injection layouts that can be formed by removing one or more of its bytes. We see that the maximum word length for which there are injection layouts with a minimum extinguishing cost of 16, is 4 bytes. In the choice of the injection layout from the 87 candidates we have taken into account the number of operations it takes to apply the message word to the state. As in 32-bit implementations of AES the columns of state and round keys are coded as 32-bit words, the number of active columns in the injection layout is best minimized. Among the 87 layouts, 40 have

four active columns, 42 have three and 5 have two. We chose the ALPHA-MAC injection layout from the latter.

7 On extinguishing differentials in ALPHA-MAC

We start with a result on the minimum length of an extinguishing differential.

Theorem 3. *An extinguishing differential in* ALPHA-MAC *spans at least 5 message words.*

Proof: The proof is divided into 3 steps.
Step 1: It can easily be verified that for the AES round transformation, there are no two different round keys that result in a common round input being mapped to a common round output. Hence, extinguishing differentials must span at least two message words.
Step 2: There are also no extinguishing differentials spanning only 2 message words. This can be shown as follows.

Let x_i be the first message word with non-zero difference. Hence y_{i-1} has no differences. The state y_i can have non-zero differences in the positions $(0,0)$, $(0,2)$, $(2,0)$ and $(2,2)$ and nowhere else. The application of SR and SB doesn't change this. Since MC has branch number 5, its application will result in a state with at least 3 non-zero bytes in the first or the third column, or both. The choice of the injection layout ensures that these differences can't be extinguished in the next AK step.
Step 3: We show that there are no extinguishing differentials that span 3 or 4 message words by means of an 'impossible differential.' The impossible differential is illustrated in Figure 2.

Let again x_i be the first message word with non-zero difference. The state y_{i+1} has zero differences in the second and the fourth column. At least one of the remaining columns contains a non-zero difference, because there are no extinguishing differentials of length 2.

Assume now that y_{i+3} has zero difference. This is only possible if before the application of the step AK in iteration $i+3$, the second and the fourth column contain no zeroes. Propagating this condition backwards, we obtain that y_{i+2} must have zero differences in the positions $(0,1)$, $(0,3)$, $(1,0)$, $(1,2)$, $(2,1)$, $(2,3)$, $(3,0)$ and $(3,1)$.

Since AK doesn't act on any of these bytes, the same condition must hold on the state before the application of AK in iteration $i+2$. But then the MC step in iteration $i+2$ has an input with at least 2 zeroes in each column, and an output with at least 2 zeroes in each column, and a least one column with at least one non-zero byte. This is impossible because the branch number of MC is 5. □

We have the following corollary.

Corollary 1. *Given y_{t-1}, the state value before iteration t, the map*

$$s : (x_t, x_{t+1}, x_{t+2}, x_{t+3}) \rightarrow y_{t+3}$$

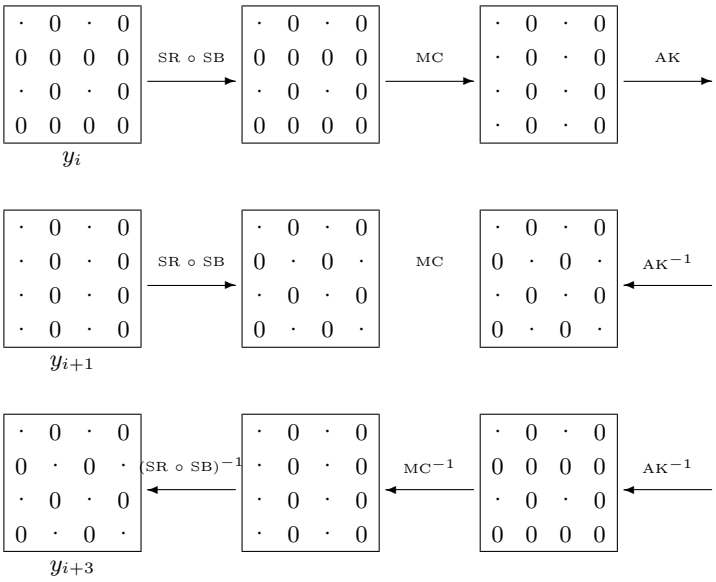

Fig. 2. The impossible differential used in the proof of Theorem 3.

from the sequence of four message words $(x_t, x_{t+1}, x_{t+2}, x_{t+3})$ *to the state value before iteration* $t + 4$ *is a bijection.*

Proof: This follows directly from the fact that the number of possible message word sequences of length four and the number of state values are equal and that starting from a given state, any pair of such message sequences will lead to different state values. □

The Alpha-MAC injection layout is one of the few 4-byte injection layouts with this property. Note that $s^{-1}(y_{t+3})$ can easily be computed, for any given y_{t-1}. It follows that if the value of the state leaks, it becomes trivial to construct forgeries forgeries based on internal collisions. However, we see no other methods for obtaining the value of the state than key recovery or the generation of internal collisions.

In the assumption that our program described in Section 6.2 is correct, its result is a proof for the fact that the minimum extinguishing cost is 16. An analytical proof for this minimum cost can be constructed, but is left out here because of the space restrictions. The minimum extinguishing cost imposes an upper bound to the probability of a truncated characteristic of $(2^{-6})^{16} = 2^{-96}$. A closer analysis reveals that in almost all cases an active S-box contributes a factor 2^{-8} rather than 2^{-6}. An active S-box contributes a factor 2^{-6} only if it was 'activated' by the previous application of AK, hence, if it was passive before AK.

We have written a variant of our program taking these aspects into account, resulting in an upper bound for the probability of truncated characteristics of

2^{-126}. For a single extinguishing differential there can be multiple truncated characteristics leading to an extinguishing probability that is the sum of those of the characteristics. In our security claims we have taken a margin by claiming $\ell_c = 120$.

8 Performance

We estimate the relative performance difference between AES and Alpha-MAC. We compare this estimate with some benchmark performance figures.

8.1 Compared to AES

In this section we express the performance of Alpha-MAC in terms of AES operations, more particularly, the AES key schedule and the AES encryption operation. This allows to use AES benchmarks for software implementations on any platform or even hardware implementations to get a pretty good idea on the performance of Alpha-MAC. We illustrate the comparison for the case of a Pentium processor, because it is easy to obtain figures for this platform. We note however that in most of the security-critical applications the cryptographic services are delivered by dedicated security modules: smart cards, HSMs, set-top boxes, ... These modules typically use 8-bit processors, 486 processors and the new ones may have AES accelerators. Clearly, in this respect Alpha-MAC takes advantage from the efficiency of AES on a wide range of platforms.

One iteration of Alpha-MAC corresponds roughly to 1 round of AES, hence roughly 1/10 of an AES encryption. The differences are due to the following facts. Firstly, the iteration of Alpha-MAC replaces the addition of a 16-byte round key by the injection layout and the addition of the 4 bytes. Some implementations of the AES recompute the round keys for every encryption. This overhead is not present in Alpha-MAC. Finally, the last round of AES is not equal to the first 9 rounds. Using this rough approximation, we can state that MACing a message requires:

setup: 1 AES key schedule and 1 AES encryption,
message processing: 0.1 AES encryptions per 4-byte message word,
finalization: 1 AES encryption to compute the tag.

Hence, the performance of the Alpha-MAC message processing can be estimated at $0.25 \times 0.1^{-1} = 2.5$ times the performance of AES encryption, with a fixed overhead of 1 encryption for the final tag computation. The setup overhead can be written off over many tag computations.

8.2 On the Pentium III

A 32-bit optimized implementation of the AES round transformation implements MC, SR and SB together by means of 16 masking operations, 12 shifts, 12 XOR operations, and 16 table lookups. The implementation of AK requires 4 XOR

Table 2. Performance on the Pentium III/Linux platform, as defined by NESSIE

Primitive Name	message processing (cycles/byte)	setup (cycles)	setup + finalization (cycles)
HMAC/MD5	7.3	804	2634
HMAC/SHA-1	15	1346	4697
CBC-MAC/AES	26	616	2056
Umac-32	2.9	54K	55K
Alpha-MAC (estimate)	10.6	1032	1448

operations. The iteration function of Alpha-MAC replaces the 4 XORs by 2 masks, 2 XORs and one shift for the combination of the injection layout and AK. If we estimate that all operations have the same cost, then the cost of the iteration function equals $61/60 \approx 1.02$ times the cost of the AES round transformation.

Table 2 is based on the performance figures given by the NESSIE consortium [6]. The performance of Alpha-MAC was estimated using the NESSIE figures for AES. We conclude that the performance of Alpha-MAC is quite good. It outperforms HMAC/SHA-1 and CBC-MAC/AES for all message lengths. Alpha-MAC is slower than Umac-32 but its setup time is a factor 50 shorter.

9 Conclusions

We have proposed a set of three security claims for iterated MAC functions, addressing the issue of internal collisions. We presented a new construction method for block cipher based MAC functions. We proved that, in the absence of internal collisions, the security of the construction can be reduced to the security of the underlying block cipher.

Secondly, we proposed Alpha-MAC, an efficient MAC function constructed from AES with the method presented in the first part. We explained our design decisions and provided the results of our preliminary security analysis. The performance of Alpha-MAC turns out to be quite competitive.

References

1. ANSI X9.19, *Financial institution retail message authentication,* American Bankers Association, 1986.
2. *Federal Information Processing Standard 180-2, Secure Hash Standard*, National Institute of Standards and Technology, U.S. Department of Commerce, August 2002.
3. *Federal Information Processing Standard 197, Advanced Encryption Standard (AES)*, National Institute of Standards and Technology, U.S. Department of Commerce, November 2001.

4. *Federal Information Processing Standard 198, The Keyed-Hash Message Authentication Code (HMAC)*, National Institute of Standards and Technology, U.S. Department of Commerce, March 2002.
5. *ISO/IEC 9797-1, Information technology - Security Techniques - Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher,* ISO 1999.
6. *Performance of optimized implementations of the NESSIE primitives, version 2.0*, The NESSIE Consortium, 2003, https://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/D21-v2.pdf.
7. Mihir Bellare, Ran Canetti, Hugo Krawczyk, "Keying hash functions for message authentication," *Advances in Cryptology - Crypto 96, LNCS 1109,* N. Koblitz, Ed., Springer-Verlag, 1996, pp. 1–15.
8. John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, Phillip Rogaway, "UMAC: Fast and Secure Message Authentication," *Advances in Cryptology – Crypto '99, LNCS 1666,* M.J. Wiener, Ed., Springer-Verlag, 1999, pp. 216–233.
9. John Black and Phillip Rogaway, "CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions," *Advances in Cryptology - CRYPTO '00, LNCS 1880,* M. Bellare, Ed., Springer-Verlag, 2000, pp. 197–215.
10. Bert den Boer, Bart Van Rompay, Bart Preneel and Joos Vandewalle., "New (Two-Track-)MAC based on the two trails of RIPEMD — Efficient, especially on short messages and for frequent key-changes," *Selected Areas in Cryptography — SAC 2001, LNCS 2259,* S. Vaudenay and Amr M. Youssef, Eds., Springer-Verlag, pp. 314–324.
11. K. Brincat and C. J. Mitchell, "New CBC-MAC forgery attacks," *Information Security and Privacy, ACISP 2001, LNCS 2119*, V. Varadharajan and Y. Mu, Eds., Springer-Verlag, 2001, pp. 3–14.
12. Don Coppersmith and Chris J. Mitchell, "Attacks on MacDES MAC Algorithm," *Electronics Letters, Vol. 35*, 1999, pp. 1626–1627
13. Don Coppersmith, Lars R. Knudsen and Chris J. Mitchell, "Key recovery and forgery attacks on the MacDES MAC algorithm," *Advances in Cryptology - CRYPTO'2000 LNCS 1880,* M. Bellare, Ed., Springer Verlag, 2000, pp. 184–196.
14. Joan Daemen, Vincent Rijmen, "AES Proposal: Rijndael," *AES Round 1 Technical Evaluation CD-1: Documentation*, National Institute of Standards and Technology, Aug 1998.
15. Donald W. Davies, "A message authenticator algorithm suitable for a mainframe computer," *Advances in Cryptology – Proceedings of Crypto '84, LNCS 196,* G. R. Blakley and D. Chaum, Eds., Springer-Verlag, 1985, pp. 393-400.
16. Tetsu Iwata and Kaoru Kurosawa, "OMAC: One-key CBC MAC," *Fast Software Encryption 2003, LNCS 2887,* T. Johansson, Ed., Springer-Verlag, 2003, pp. 129–153.
17. E. Jaulmes, A. Joux and F. Valette, "On the security of randomized CBC-MAC beyond the birthday paradox limit: A new construction," *Fast Software Encryption 2002, LNCS 2365,* J. Daemen and V. Rijmen, Eds., Springer-Verlag, 2002, pp. 237-251.
18. Antoine Joux, Guillaume Poupard and Jacques Stern, "New Attacks against Standardized MACs," *Fast Software Encryption 2003, LNCS 2887,* T. Johansson, Ed., Springer-Verlag, 2003, pp. 170–181.
19. Lars R. Knudsen and Bart Preneel, "MacDES: a new MAC algorithm based on DES," *Electronics Letters, Vol. 34, No. 9,* 1998, pp. 871–873.
20. Lars R. Knudsen, "Truncated and higher order differentials," Fast Software Encryption '94, LNCS 1008, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196–211.

21. Lars R. Knudsen and Chris J. Mitchell, "Partial key recovery attack against RMAC," *Journal of Cryptology, to appear*.
22. Kaoru Kurosawa and Tetsu Iwata, "TMAC: Two-Key CBC MAC," *Topics in Cryptology: CT-RSA 2003, LNCS 2612*, M. Joye, Ed., Springer-Verlag, 2003, pp. 265–273.
23. Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
24. Bart Preneel and Paul C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Advances in Cryptology, Proceedings Crypto'95, LNCS 963,* D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1–14.
25. Bart Preneel and Paul C. van Oorschot, "A key recovery attack on the ANSI X9.19 retail MAC," *Electronics Letters, Vol. 32*, 1996, pp. 1568-1569.
26. Bart Preneel and Paul C. van Oorschot, "On the security of iterated Message Authentication Codes," *IEEE Trans. on Information Theory, Vol. IT45, No. 1,* 1999, pp. 188-199.
27. Ron Rivest, "The MD4 message digest algorithm," *Network Working Group Request for Comments:1186*, 1990.

A Detecting internal collisions

If the final transformation is reversible, any pair of messages with the same tag form an internal collision. Otherwise, for two messages with the same tag, the collision could have taken place in the final transformation itself. If the ratio between the size of the state and the tag length is v, one can query the MAC function for message tuples $\{m_i \| a_1, m_i \| a_2, \ldots, m_i \| a_{\lceil v \rceil}\}$, with the a_j a set of $\lceil v \rceil$ different strings. If we have two messages m_i and m_j for which all components of the corresponding tuples have matching tags, m_i and m_j very probably form an internal collision. With respect to a tag with the same length as the size of the state, having a shorter tag only multiplies the required number of MAC function queries by v.

B Another way to write Alpha-MAC

The standard [3] explains how to construct an equivalent description for the inverse cipher of AES. We have a similar effect here. Firstly, in the definition of f, the order of the steps SR and SB plays no role. Therefore a sequence of applications of f can also be written as follows:

$$\cdots \circ f \circ f \circ \cdots = \cdots \circ \mathrm{AK}[LI(x_{i+1})] \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{SR} \circ \mathrm{AK}[LI(x_i)] \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{SR} \circ \cdots$$

Secondly, the order of the steps SR and AK can be changed, if the injection layout is adapted accordingly:

$$\mathrm{SR} \circ \mathrm{AK}[LI(x_i)] = \mathrm{AK}[\mathrm{SR}(LI(x_i))] \circ \mathrm{SR} = \mathrm{AK}[LI'(x_i)] \circ \mathrm{SR}.$$

We obtain the following:

$$\cdots \circ f \circ f \circ \cdots = \cdots \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{AK}[LI'(x_i)] \circ \mathrm{SR} \circ \mathrm{MC} \circ \mathrm{SB} \circ \mathrm{AK}[LI'(x_{i-1})] \circ \mathrm{SR} \circ \cdots$$

Concluding, when we ignore the boundary effects at the beginning and the end of the message, ALPHA-MAC can also be described using an alternative iteration function f' and an alternative injection layout function LI', given by:

$$f'(y_{i-1}, x_i) = (\mathrm{AK}[LI'(x_i)] \circ \mathrm{SR} \circ \mathrm{MC} \circ \mathrm{SB})\,(y_{i-1})$$

$$LI'(m) = \begin{bmatrix} q_0 & 0 & q_1 & 0 \\ 0 & 0 & 0 & 0 \\ q_3 & 0 & q_2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The alternative injection layout is equivalent to the original injection layout applied to message words with the rightmost two bytes swapped.