

Almost Universal Forgery Attacks on AES-Based MAC's

Orr Dunkelman · Nathan Keller · Adi Shamir

the date of receipt and acceptance should be inserted later

Abstract A Message Authentication Code (MAC) computes for each (arbitrarily long) message m and key k a short authentication tag which is hard to forge when k is unknown. One of the most popular ways to process m in such a scheme is to use some variant of AES in CBC mode, and to derive the tag from the final ciphertext block. In this paper we analyze the security of several proposals of this type, and show that they are vulnerable to a new type of attack which we call *almost universal forgery*, in which it is easy to generate the correct tag of any given message if the attacker is allowed to change a single block in it, after performing a one-time preprocessing phase which is faster than exhaustive key search.

Keywords 94A60 · 68P25 · Message Authentication Codes · Almost Universal Forgery · ALRED · Pelican

The first author was supported in part by the Israel Science Foundation through grant No. 827/12 and by the German-Israeli Foundation for Scientific Research and Development through grant No. 2282-2222.6/2011.

The second author was supported by the Alon Fellowship.

O. Dunkelman and Nathan Keller, and Adi Shamir
Faculty of Mathematics and Computer Science
Weizmann Institute of Science
P.O. Box 26, Rehovot 76100, Israel
E-mail: nathan.keller,adi.shamir@weizmann.ac.il
O. Dunkelman
Computer Science Department
University of Haifa
Haifa 31905, Israel
E-mail: orrd@cs.haifa.ac.il
N. Keller
Department of Mathematics
Bar-Ilan University
Ramat Gan 52900, Israel
E-mail: nathan.keller@weizmann.ac.il

1 Introduction

Message Authentication Codes (MAC's) are designed to compute for each message an authentication tag which is easy to verify (when the key is known) but hard to forge (when the key is unknown). They combine the properties of hash functions (by dealing with arbitrarily long messages), symmetric encryption algorithms (by using a shared secret key) and signature schemes (by dealing with forgery rather than with secrecy). The main security requirement is that even after choosing a large number of messages m_i and obtaining their corresponding tags t_i for some unknown key k of length n , the adversary should not be able to compute with a high probability of success the tag t for a new message m in time which is substantially smaller than the 2^n complexity of exhaustive key search (or $2^{|t|}$ verification queries). The weakest and strongest flavors of this problem are called *existential forgery* where the attacker has to find the tag of a single new message of her choice, and *universal forgery* where the attacker has to find the tag of any new message which is externally provided after all the queries to the authentication oracle are posed and answered.

In this paper we propose a new flavor of MAC security which we call an *almost universal forgery attack*. It is similar to a universal forgery attack in the sense that tags have to be computed for any provided message (when the attacker can no longer access the authentication oracle¹), but it allows the forger to make a minimal change from the given m to a modified m' by replacing one² of its blocks (whose position can be adversarially chosen), before producing its tag. Since many types of files are allowed to start with a metadata block which is ignored, or end with a padded block which can be arbitrarily chosen, such a small change may be invisible to the user when the file is displayed on a computer screen. This new notion of security had already been adopted and further analyzed in two follow-on papers by Peyrin et al. [19] and Sasaki [20].

Some of the best known constructions of MAC schemes in the literature use other types of cryptographic primitives as starting points. In particular, it is easy to turn any block cipher into a MAC by using it in CBC mode, discarding all the intermediate ciphertext blocks, and basing the authentication tag on the final ciphertext block (e.g., by truncating it to a desired bit length). However, block ciphers and MAC's are designed with completely different sets of requirements. For example, block ciphers must be invertible mappings, whereas MAC's can use noninvertible operations since there is no need to recover the input message from the final tag. More importantly, the standard security requirement for block ciphers (namely, that they should hide any information about the plaintext blocks given the ciphertext blocks) is irrelevant in a MAC construction in which all the plaintext blocks are actually given to the adversary, and all the ciphertext blocks (except the last one) are discarded. It is thus conceivable that one can speed up the MAC computation by using weaker versions of standard block ciphers. Such a simplified block cipher might not be sufficiently secure as a stand-alone cryptosystem, but it can still be an excellent MAC which makes it very difficult to compute new authentication tags from given tags. A concrete example of such an approach is

¹ In a practical scenario, giving the adversary continued access to the authentication oracle when a challenge message is given would enable her to trivially create the tag by using her oracle access. Any attempt to disallow only the challenge query is therefore artificial.

² One can easily extend this definition by allowing other types of small modifications.

the ALRED family of MAC's [9], which was designed in 2005 by Joan Daemen and Vincent Rijmen. It uses a keyless version of AES-128 in CBC mode to process the blocks of the message (using the secret authentication key only at the beginning and the end of the chain), and it reduces the effective number of AES rounds per block to 4. Compared to the standard 10-round AES-128, such schemes use only 40% of the number of rounds to process each chunk of 128 input bits, and eliminate one of the four steps (the AddRoundKey operation) in each round.³ After the publication of the original ALRED paper, other researchers had published several follow-up papers in which they proposed new members of this growing family of MAC schemes, such as PC-MAC-AES [18], MT-MAC-AES [18], Marvin [21, 22], and ALE [3].

In this paper we analyze the security implications of such simplified MAC designs in the context of our new notion of almost universal forgery attacks. In Section 2 we review the general ALRED construction, and its particular incarnation Pelican (upon which all the newer incarnations are based). We then show how to obtain for such schemes an almost universal forgery attack by using a preprocessing phase (which has to be run only once for each key) in order to recover the secret initial value of the CBC iteration. As we demonstrate in this paper, finding this value is often easier than finding the authentication key, and its knowledge suffices in order to forge the tag of any (slightly modified) message in linear time. In particular, we show in Section 3 how to exploit any (standard or impossible) differential weakness of the underlying block cipher in order to find a two-block internal collision, from which we can recover this secret value. Since the well-known 4-round (keyed or unkeyed) impossible differential of AES admits a straightforward attack on five rounds, we obtain an almost universal forgery attack on the 5-round extended version of Pelican in about 2^{80} preprocessing time, which is much faster than the 2^{128} time complexity of exhaustive key search. Our attack is faster and can handle more rounds than the best previously published attack by Yuan et al. [24], which uses 2^{85} time to attack the 4-round version of Pelican.⁴ We then show how to extend the attack to the 6-round version of Pelican with a preprocessing time complexity of 2^{110} . A different kind of attack, which exploits the self similarity properties of keyless AES, is described in Section 4. Surprisingly, this attack can be applied to any number of rounds of AES (4, 10, or 100), with a preprocessing time complexity of 2^{96} . This technique yields the first known attack on the MAC Marvin, which seems to be much stronger than Pelican since it was specifically designed to resist all the previously published attacks on such MAC schemes. Finally, in Section 5, we show that any MAC which uses a keyless block cipher (not necessarily AES) is vulnerable to new time/memory tradeoff attacks which are faster than generic tradeoff attacks on one-way functions. A summary of all these new attacks appears in Table 1.

³ It is important to notice that our new attacks do not invalidate ALRED's formal security claims, since its designers decided to limit the number of messages which can be queried for each key to 2^{64} , even though they did not describe in their paper any concrete attack which was successful beyond this bound.

⁴ In an earlier version of this paper [12], we have shown how to reduce the time complexity of the attack on the 4-round version of Pelican to 2^{65} time. A few months later, another attack with the same time complexity was also developed independently in [5]. Hence, we do not describe our improved attack on 4-round Pelican in this paper, and refer the reader to [5, 12].

Attack Type	Number of AES Rounds	Keyed or Keyless	Complexity		
			Data	Time	Memory
Imp. Diff. (Sect. 3.2)	5	Keyed	2^{80} CM	2^{80}	2^{78}
Imp. Diff. (Sect. 3.2)	6	Keyed	2^{110} CM	2^{110}	2^{64}
Generic (Sect. 4)	any	Keyless	2^{96} CM	2^{96}	2^{32}
Generic (Sect. 4)	any	Keyless	2^{96} ACM	2^{96}	1
Generic (Sect. 5)	any	Keyless	$2^{85.3}$ CM	$2^{85.3}$	$2^{85.3}$

CM — Chosen message, ACM — Adaptively chosen message.
Time complexity is measured in MAC evaluation units.

Table 1 Summary of Our Attacks on Variants of Pelican

2 The ALRED Construction

In this section, we describe the ALRED construction and its specific instantiation Pelican, which serve as a typical case study for the attack techniques we present in this paper. First, we describe the structure of ALRED and Pelican. Then, we show that recovering the initial value $y_0 = E_k(0)$ in an ALRED-type construction allows the attacker to mount almost universal forgery attacks on the MAC.

2.1 The Structure of ALRED and Pelican

As described in the introduction, ALRED is a MAC construction based on an iterated block cipher. Given a secret key k and a message m , the generation of the tag is composed of four steps:

1. **Message padding and splitting:** The message is padded with a single 1 and the minimum number of 0's so that the resulting length is a multiple of ℓ_w bits, where ℓ_w is a characteristic of the MAC. The padded message is then divided into blocks m_1, m_2, \dots, m_ℓ of length ℓ_w each.
2. **State initialization:** The state is initialized with the all-zero ℓ_b -bit block (where ℓ_b is the block size of E), and then the full block cipher is applied to it, to obtain $y_0 = E_k(0)$.
3. **Chaining:** The following iteration is applied to the blocks m_1, m_2, \dots, m_ℓ sequentially:
 - The message block m_i is mapped to an injection input Inj_i whose length is equal to the length of r round keys of the block cipher.
 - A reduced r -round variant of the block cipher is applied to the state y_{i-1} , with Inj_i replacing the round keys. The resulting state is denoted by y_i .
4. **Finalization:** The full block cipher is applied to the state y_ℓ to obtain $z' = E_k(y_\ell)$. Then z' is truncated to the required length to obtain the tag $z = \text{Truncate}(z')$.

The construction is illustrated in Figure 1, where f denotes a reduced r -round variant of the block cipher.

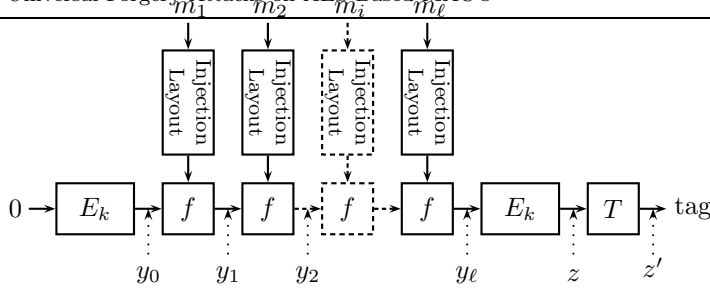


Fig. 1 The ALRED construction

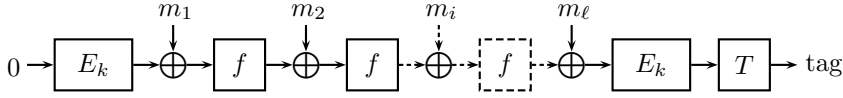


Fig. 2 The structure of Pelican

2.1.1 Pelican

The main instantiation of ALRED considered in this paper is Pelican, which also served as the basis for the later designed MACs PC-MAC-AES, Marvin, and ALE. In Pelican, the underlying block cipher is AES, the number of rounds is $r = 4$, the block length is $\ell_w = 128$ bits, and the injection input consists of using the message as the first round subkey, and zeros as the other three subkeys. An equivalent description, which is more convenient for our analysis, is that in the i 'th iteration of the chaining step, the 128-bit message block m_i is XORed to the state y_{i-1} , and then a *four-round keyless AES* (i.e., AES without the AddRoundKey operations) is applied to obtain the next state y_i . The equivalent description is shown in Figure 2, where f denotes four-round keyless AES.

2.2 Almost Universal Forgery Attacks on ALRED

An easy but important observation on the structure of ALRED is that recovering the initial state $y_0 = E_k(0)$ makes it possible to mount an almost universal forgery attack on the MAC. Notice that finding y_0 may be easier than finding k , which requires the successful cryptanalysis of full AES from a single plaintext/ciphertext pair.

Assume that in a Pelican-type construction, the adversary obtains a message $m = (m_1, m_2, \dots, m_\ell)$ and a number j , and wants to compute the tag of a message of the form $m' = (m_1, \dots, m_{j-1}, m'_j, m_{j+1}, \dots, m_\ell)$ (i.e., the adversary is allowed to alter only the j 'th word of m).

The adversary can act as follows, in order to obtain such a message m' whose tag is equal to $\text{Truncate}(y_0)$. First, she assumes that the value y_ℓ in the MAC computation of m' equals 0. She rolls the computation back, using the message words $m_\ell, m_{\ell-1}, \dots, m_{j+1}$ to obtain the value $f^{-1}(y_j)$. In the other direction, the adversary rolls the initial value $y_0 = E_k(0)$ forward using the message words

m_1, m_2, \dots, m_{j-1} , to obtain the value y_{j-1} . Now, let $m'_j = f^{-1}(y_j) \oplus y_{j-1}$, and let $m' = (m_1, \dots, m_{j-1}, m'_j, m_{j+1}, \dots, m_\ell)$. Then, the MAC computation of m' is identical with that of m until the value y_{j-1} , and the next chaining value $f(y_{j-1}) \oplus m'_j$ is equal to the value of y_j obtained in the backward computation. This assures that the chaining value y_ℓ equals to 0, and hence, the tag corresponding to m' is $\text{Truncate}(E_k(0)) = \text{Truncate}(y_0)$.

In a similar way, the adversary can produce a message m' whose tag is equal to the tag of any previously seen message.

In all the attacks on ALRED presented in the sequel, the goal of the adversary is to recover the initial value y_0 . By the above discussion, this is sufficient for mounting an almost universal forgery attack on any of the flavors of ALRED.

3 Differential-Type Attacks

The first class of almost universal forgery attacks we present is differential-type attacks based on two-block internal collisions. Our attacks apply when the reduced-round block cipher f used in the MAC is weak with respect to some differential-type attack, which is often the case for reduced-round block ciphers (like 4-round AES used in Pelican and in PC-MAC-AES). The attacks are based on a generic procedure, that allows to leverage a differential-type attack on f to an attack on the MAC which recovers the initial value $y_0 = E_k(0)$, thus allowing for almost universal forgery.

We apply the procedure to leverage impossible differential attacks on 5-round and 6-round AES to attacks on an enhanced Pelican construction, in which 4-round keyless AES is replaced by 5-round or even 6-round AES with independent round keys. The data and time complexities of the resulting 5-round attack are less than 2^{80} . We note that the best previously known impossible differential attack on Pelican, by Yuan et al. [24], has data and time complexities of $2^{85.5}$, and thus, our new attack breaks a stronger variant of Pelican with a lower complexity.

3.1 The Generic Leveraging Procedure

The main observation used in the attack is that two-block internal collisions in a Pelican-type scheme can be viewed as input/output pairs for the block cipher f , where the unknown initial state $E_k(0)$ is viewed as an initial whitening key of f . In order to understand the observation, we consider an equivalent representation of the first two steps of a Pelican-type construction, presented in Figure 3. In terms of the equivalent representation, given a pair of two-block messages (m_1, m_2) and (m_1^*, m_2^*) , we treat (m_1, m_1^*) as an input pair for the block cipher f . If (m_1, m_2) and (m_1^*, m_2^*) form an internal collision, then we know that the difference between the corresponding outputs of f must be $m_2 \oplus m_2^*$, since f is a permutation and thus the only way to create a zero difference after the second f is to have a zero difference before it.

Hence, even if the function f is keyed, any local collision provides the adversary with a pair of input/output values for f , in which the actual inputs (i.e., m_1 and m_1^*) and the output difference (i.e., $m_2 \oplus m_2^*$) are known to the adversary. This

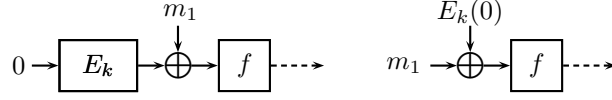


Fig. 3 The original structure of Pelican (on the left) and the equivalent structure

makes it possible to leverage several classes of differential-type attacks on f to attacks on the MAC construction.

Specifically, the usual structure of differential-type attacks (both ordinary differential attacks and impossible differential attacks) is as follows:

1. The analyzed cipher E is considered as a cascade $E = E_1 \circ E' \circ E_0$, where E' is the “core” of the cipher, and E_0 and E_1 are either empty or consist of a few rounds.
2. The adversary finds a *differential-type distinguisher* for E' , showing that if a pair of inputs of E' has some prescribed difference α , then the corresponding outputs of E' have difference β with an unexpected probability (either high in differential cryptanalysis, or zero in an impossible differential attack).
3. The adversary guesses (all or part of) the subkeys used in E_0 and E_1 , performs partial encryption/decryption, and checks whether the prediction of the distinguisher on E' holds. This makes it possible to retrieve the value of the guessed subkeys, provided there is a sufficiently large number of input/output pairs for E .

In our situation, assuming that f is keyed, the adversary cannot perform partial decryption on the outputs of the block cipher f , since she knows only the output difference, but not the actual values. However, if in the differential-type attack, the subcipher E_1 in the subdivision of f into $E_1 \circ E' \circ E_0$ is empty, then the attack can be applied directly to the input/output pairs obtained from internal collisions, allowing to retrieve the internal state $E_k(0)$ (which serves as the subkey used in E_0).

Therefore, any differential-type attack with no rounds after the distinguisher can be leveraged directly into an attack on the MAC construction. The data and time complexities of the attack *given the internal collisions* are the same as the complexities of the original attack on f .

We note that the total complexity of the attack on the MAC is expected to be much higher than the complexity of the attack on the block cipher due to the need to obtain internal collisions by the birthday paradox at the starting point of the attack. Generally, a differential-type attack using 2^k chosen plaintext pairs is transformed into an attack on the MAC requiring at least $2^{(k+l_b)/2}$ messages, where ℓ_b is the block size of f , since only this amount of messages is sufficient for obtaining 2^k internal collisions by the birthday paradox. Moreover, in actual attacks it may be desirable to tweak the differential-type attack before leveraging it to an attack on the MAC, in order to allow using larger data structures in the attack, and thus reduce the amount of data required for obtaining the internal collisions. Such a tweak is demonstrated in the attacks on enhanced Pelican presented below.

3.2 Impossible Differential Attacks on Enhanced Pelican

As an example of the general technique outlined above, we show how to use impossible differential attacks on reduced-round AES in order to break enhanced variants of Pelican, in which 4-round keyless AES is replaced by 5-round or 6-round AES with AddRoundKey operations (possibly with independent subkeys). For sake of simplicity, we call such variants 5-round Pelican and 6-round Pelican, respectively.

In the description of the attacks on AES, we use the following standard notations. Each state during the AES encryption is represented by a 4-by-4 byte matrix, and the entries of the matrix are numbered by $0, 1, \dots, 15$, such that the j -th entry in the i -th row (for $0 \leq i, j \leq 3$) is numbered by $i + 4j$. The four operations applied in each round – SubBytes, ShiftRows, MixColumns, and AddRoundKey – are denoted by SB, SR, MC, and ARK, respectively. The rounds are numbered $1, 2, \dots$. The subkey used in the r -th round is denoted by k_r , and the initial whitening key is denoted by k_0 .

3.2.1 Impossible Differential Attacks on 5-round and 6-round AES

Impossible differential attacks on reduced-round AES with 128-bit keys were extensively studied in the last decade, and several attacks on 5, 6, and 7 rounds were presented (see [17] for a summary of the attacks).

The simplest attack, by Biham and Keller [2] on 5-round AES, does not use any rounds after the distinguisher, and thus, can be applied directly to 5-round Pelican using the generic technique described above. The more advanced attacks on 7-round AES (e.g., by Bahrak and Aref [1]) cannot be leveraged directly since they analyze rounds on both sides of the distinguisher, but a reduced 6-round variant of the attacks, which drops the round after the distinguisher, can be leveraged directly.

In order to enable the adversary to use larger structures and thus reduce the data complexity of the attacks on the MAC, we present slightly different impossible differential attacks on 5-round and 6-round AES, which we will then apply to 5-round and 6-round Pelican.

Our attacks are based on a 3-round impossible differential of AES which is similar to (but not identical with) the impossible differentials used in all previous attacks. The differential, depicted in Figure 4, asserts that if the difference in the input to round i of (either keyed or unkeyed) AES is zero in bytes 0, 5, 10, 15 (regardless of the difference in the rest of the bytes), then the difference in the input to round $i + 3$ cannot be non-zero only in byte 0.

Indeed, consider a pair (P, P^*) of inputs to the i -th round of AES, such that the difference $P \oplus P^*$ is zero in bytes 0, 5, 10, 15. By the structure of AES, the corresponding intermediate difference in the input to round $i + 1$ is zero everywhere in the first column.

On the other hand, if the difference in the input to round $i + 3$ is non-zero in byte 0 and zero in all other bytes, then the input difference to round $i + 2$ is non-zero in bytes 0, 5, 10, 15 and zero in the other bytes. Consequently, the difference in the input to round $i + 1$ is non-zero in all bytes, contradicting the forward direction.

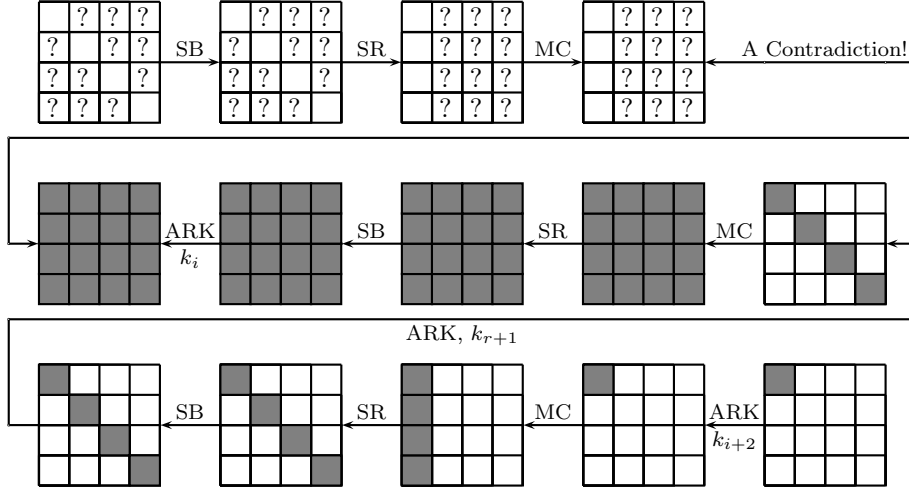


Fig. 4 An Impossible Differential of 3-round AES. White cells denote zero difference, gray cells denote non-zero difference, and “?” denotes arbitrary difference.

Similar impossible differentials hold if the zero input difference is placed at any of the four sets of bytes: $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$, and $\{3, 4, 9, 14\}$, and if the output is replaced by any single-byte difference.

A standard way to use this impossible differential to break 5-round AES, is to perform the following three-step attack procedure (see, e.g., [2] for a similar attack):

1. Ask for the encryption of a structure of plaintexts in which the values of several bytes are constant, and the other bytes assume different values (these bytes are called “active”).
2. Consider only those ciphertext pairs within the structure in which the difference in the input to round 5 is in a single byte (using a hash table, these pairs can be instantly found).
3. Guess the part of the first subkey which are used in the active bytes, and for each guess and for each pair, check whether the difference in the input to round 2 is zero in one of the four sets of bytes: $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$, and $\{3, 4, 9, 14\}$. If so, discard the key guess.

In order to extend the attack to 6-round AES, the adversary can add one round at the beginning, guess relevant subkey material in the first two subkeys, and check whether the input to round 3 is zero in one of the four sets: $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$, and $\{3, 4, 9, 14\}$. We omit the details of the attacks here, since they are essentially the same as the attacks on enhanced Pelican we present below.

3.2.2 Leveraging the attacks to 5-round and 6-round Pelican

Attack on 5-round Pelican. The algorithm of the attack on 5-round Pelican is as follows:

1. **Detecting internal collisions:**

- (a) Ask for the MAC evaluation of two structures S_1 and S_2 of 2^{78} two-block messages each, such that:
 - The first blocks in all messages in both structures have a fixed value in bytes 1, 6, 11, 12, 13, 14 (i.e., the same value for all messages in both structures), and assume 2^{79} different values in the remaining 10 bytes in the two structures.
 - The second blocks in all messages in S_1 are fixed to a constant value m_2 and the second blocks of all messages in S_2 are fixed to a constant value m_2^* , such that $MC^{-1}(m_2 \oplus m_2^*)$ is non-zero only in byte 0.
- (b) Insert the tags into a hash table and search for internal collisions $(m_1, m_2), (m_1^*, m_2^*)$, such that $(m_1, m_2) \in S_1$ and $(m_1^*, m_2^*) \in S_2$.⁵

2. **Attacking the internal reduced block cipher:** For each internal collision and for each guess of bytes 0, 2, 3, 4, 5, 7, 8, 9, 10, 15 of the internal state $E_k(0)$,⁶ partially encrypt the pair $(m_1 \oplus E_k(0), m_1^* \oplus E_k(0))$ through the first round of AES, and check whether the input difference to round 2 is zero in one of the four sets of bytes $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$, and $\{3, 4, 9, 14\}$. If this is the case, discard the guess of $E_k(0)$.

Analysis of the attack. The time complexity of the first phase of the attack (i.e., detecting internal collisions) is 2^{79} MAC evaluations, and its memory complexity is 2^{78} 128-bit blocks. The data is expected to contain $2^{78} \cdot 2^{78} \cdot 2^{-128} = 2^{28}$ internal collisions of the form $(m_1, m_2), (m_1^*, m_2^*)$, where $(m_1, m_2) \in S_1$ and $(m_1^*, m_2^*) \in S_2$.

In the second phase of the attack, for each internal collision, the adversary guesses 80 bits of $E_k(0)$ and checks whether the difference between the intermediate states in the input to round 2 of AES is zero in one of the four sets of bytes $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$, and $\{3, 4, 9, 14\}$. Since the difference in the entire Column 3 in the input to round 2 is zero (independently of the guessed subkey values), the probability that a subkey guess is discarded is $4 \cdot 2^{-24} = 2^{-22}$. Hence, the expected number of remaining subkey guesses is $2^{80} \cdot (1 - 2^{-22})^{2^{28}} \approx 2^{80} \cdot e^{-64} \approx 2^{-12}$, i.e., only the correct guess is expected to remain.

⁵ If the length of the tag is ℓ_w bits then a collision in the tag value results from an internal collision with high probability. If the tag is shorter, there can be many false alarms, but the adversary can verify that a collision is internal by appending to the two messages the same block m_3 and checking whether the new tags also collide. For the sake of simplicity, we assume in the sequel that the tag length is ℓ_w bits.

⁶ Note that if f contains an AddRoundKey operation at the beginning, with a whitening key k_0 , then the adversary should guess an equivalent key $E_k(0) \oplus k_0$ instead of $E_k(0)$. In this case, the attack makes it possible to retrieve only the value $E_k(0) \oplus k_0$, rather than $E_k(0)$. However, this value is still sufficient for mounting the almost universal forgery attacks on the MAC described in Section 2. For sake of simplicity, we assume in the sequel that f does not contain a whitening key.

The time complexity of a naive application of this step is $2^{80} \cdot 2^{28} = 2^{108}$ partial encryptions. However, this complexity can be significantly reduced by noting that for each internal collision, the adversary can perform the partial encryption in each of the three active columns independently. Specifically, the adversary can perform a two-step procedure:

1. For each of the 2^{28} internal collisions, the adversary computes all the values of bytes 3, 4, 9 of $E_k(0)$ that lead to a zero difference in byte 5 in the input to round 2, and stores them in a table.
2. For each guess of bytes 0, 2, 5, 7, 8, 10, 15 of $E_k(0)$:
 - (a) The adversary goes over all the internal collisions and checks whether the difference in bytes 0 and 10 in the input to round 2 is zero. Only $2^{28} \cdot 2^{-16} = 2^{12}$ internal collisions are expected to pass this filtering.
 - (b) The adversary considers a list of all possible values of bytes 3, 4, 9 of $E_k(0)$, and for each remaining internal collision, she discards the values of $E_k(0)$ that lead to a zero difference in byte 5 in the input to round 2 (using the table computed in the first step). If all the possible values of bytes 3, 4, 9 of $E_k(0)$ are discarded, the adversary discards the guess of bytes 0, 2, 5, 7, 8, 10, 15 of $E_k(0)$ made at the beginning of the step.

This procedure allows discarding all guesses of $E_k(0)$ which lead to zero difference in bytes $\{0, 5, 10, 15\}$ in the input to round 2, and by repeating it four times, the adversary can discard also key guesses which lead to a zero difference in one of the sets $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$, and $\{3, 4, 9, 14\}$. The time complexity of this procedure is $2^{28} \cdot 2^{58} = 2^{86}$ simple operations, which are dominated by the MAC evaluations performed in the first phase of the attack (since evaluating the full MAC is much slower than a single memory access). The overall memory complexity of this step is $2^{28} \cdot 2^{16} = 2^{44}$ 128-bit blocks, but *random accesses* are made only to much smaller lists of size at most 2^{28} 128-bit blocks.

After the 80 bits of $E_k(0)$ are found, the adversary repeats the procedure with another set of active bytes in the first round (and another set of chosen messages) to obtain the rest of $E_k(0)$. The subkeys used in the AddRoundKey operations of AES can be found in a similar manner by attacking a 4-round (or even smaller) variant of AES using known cryptanalytic techniques.

The overall data complexity of the attack on 5-round Pelican is 2^{80} chosen two-block messages, the time complexity is 2^{80} MAC evaluations, and the memory complexity is 2^{78} 128-bit blocks. Note that this is better than the best previously known impossible differential attack due to Yuan et al. [24], which attacks 4-round Pelican in $2^{85.5}$ data and time.

Attack on 6-round Pelican. A similar attack can be applied to 6-round Pelican.

In the first phase of the attack, the adversary considers pairs of structures S_1, S_2 of 2^{64} two-block messages each, such that in the first block, 8 bytes which form two shifted columns (e.g., bytes 0, 3, 4, 5, 9, 10, 14, 15) are fixed to the same value in both structures and the remaining 8 bytes assume all 2^{64} possible values. The second blocks are the same as in the 5-round attack, and for each pair of structures, we expect to find a single collision which is detected in the same way as in the 5-round attack.

In the second phase of the attack, the adversary guesses the eight bytes of $E_k(0)$ that correspond to the active bytes in the data (in our example above,

these are bytes 1, 2, 6, 7, 8, 11, 12, 13), and for each internal collision, she partially encrypts the pair $(m_1 \oplus E_k(0), m_1^* \oplus E_k(0))$ through the first round of AES. The analysis of the internal collision is continued only if the intermediate difference before the MixColumns operation in round 2 of AES is non-zero only in the first three columns. In such cases, the difference is non-zero only in two bytes in each of these columns (in our example these are bytes 2, 3, 5, 6, 8, 9). The adversary guesses the value of these six bytes in the subkey used in the AddRoundKey operation in round 1 of AES, partially encrypts the pair of messages through round 2, and checks whether the difference in the input to round 3 is zero in one of the four sets of bytes $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$, and $\{3, 4, 9, 14\}$. If yes, the 14-byte subkey guess is discarded. For a single examined internal collision and a fixed subkey guess, the probability that the guess is discarded is $4 \cdot 2^{-16} \cdot 2^{-24} = 2^{-38}$. Therefore, in order to discard most of the subkey suggestions, the adversary can examine 2^{44} internal collisions, so that the expected number of remaining subkey suggestions is $2^{112} \cdot e^{-64} \approx 2^{19.7}$. The attack is completed by examining another data set (with other active bytes) and comparing the subkey suggestions in the overlapping subkey bytes.

Since each pair of structures is expected to contain a single internal collision, the data complexity of the attack is $2 \cdot 2^{44} \cdot 2^{65} = 2^{110}$ chosen messages. The time complexity is dominated by the MAC evaluations of these chosen messages (since the second part of the attack can be performed column-wise, like in the 5-round attack), and the memory requirement is 2^{64} 128-bit blocks. (There is no need to store a list of the 2^{112} possible subkey guesses, since the guesses of the 8 active bytes in $E_k(0)$ can be checked sequentially, and for each of them, the adversary can keep a list of the values of the 6 active bytes in the second subkey that have to be discarded.)

It seems that it will be hard to extend this type of attack to 7-round Pelican, as such an attack would be roughly equivalent to an impossible differential attack on 8-round AES-128, which seems out of reach with current cryptanalytic techniques.

4 Self-Similarity Attacks on ALRED Based on Keyless AES

The second class of almost universal forgery attacks we present targets ALRED constructions based on keyless AES. We show that a little-known self-similarity property of keyless AES can be leveraged to attacks on the MAC, allowing to retrieve the initial value $y_0 = E_k(0)$. The attacks are independent of the number of rounds in the keyless variant of AES used as f , and have a fixed data complexity of 2^{97} messages and an extremely small memory complexity. While the data complexity of the attacks is higher than that of the attacks presented in Section 5 below, the extremely low memory requirement and their applicability to ALRED variants in which the adversary can control only part of the state (such as Alpha-MAC [9]), makes these attacks more attractive in various scenarios.

In Section 4.1, we present the basic variant of the attack, applicable to Pelican-type constructions. Then, we apply the technique to devise the first known forgery attack on the MAC Marvin in Section 4.2.

4.1 The Basic Attack on Pelican-type Constructions

The attack is based on a simple observation on the structure of keyless AES, first presented in [15]. We note that this observation was also applied in [4] to attack the AES-based hash function Lesamnta.

Observation 1 (Le et al.) *Consider a single round of keyless AES, i.e., a sequence of the operations SubBytes, ShiftRows, MixColumns. Denote the states before and after the round by (x, y, z, w) and $(x', y', z', w') = F(x, y, z, w)$, respectively, where each of the variables denotes a column 32-bit vector. Then we have the following:*

$$\text{If } F(x, y, z, w) = (x', y', z', w'), \quad \text{then} \quad F(y, z, w, x) = (y', z', w', x').$$

In particular, for the input (x, y, x, y) with two repeated columns, $F(x, y, x, y)$ also has repeated columns of the form (x', y', x', y') .

It is clear that while in ordinary AES, the special form (x, y, x, y) is destroyed by the AddRoundKey operation, in keyless AES this special form is preserved through an arbitrarily large number of rounds.

Using the observation, we can mount the following simple attack on generalized Pelican with any number of rounds of keyless AES as the internal block cipher.

1. For each of the 2^{64} possible 128-bit blocks of the form $m_1 = (x, y, 0, 0)$, do the following:
 - (a) Ask for the MAC evaluation of a pair of structures S_1 and S_2 of 2^{32} two-block messages each, such that:
 - i. In S_1 , the first block is fixed to m_1 , and in S_2 , the first block is fixed to $m_1 \oplus (1, 1, 1, 1)$.⁷
 - ii. In each of the structures, the second block assumes 2^{32} random values of the form (x', y', x', y') .
 - (b) Insert the tags into a hash table and search for a collision between the two structures. If a collision is found, deduce that the initial state is one of the 2^{64} possible values of the form $(z, w, z, w) \oplus m_1$, for some z, w . If not, discard the guess of m_1 .
2. For the single expected value of m_1 , that remains, find the values of z, w by exhaustive search.

The idea behind the algorithm is the following: If $E_k(0)$ is of the form $m_1 \oplus (z, w, z, w)$, then $E_k(0) \oplus m_1$ and $E_k(0) \oplus m_1 \oplus (1, 1, 1, 1)$ are both of the special form (z, w, z, w) (i.e., have two repeated columns). Since all the values of the second block m_2 are also of the special form, the corresponding y_2 values also have two repeated columns (independently of the number of rounds of keyless AES). The space of such special values is of size 2^{64} . On the other hand, the number of messages in each structure is 2^{32} , and thus the number of pairs $(m_1, m_2), (m_1^*, m_2^*)$, where $(m_1, m_2) \in S_1$ and $(m_1^*, m_2^*) \in S_2$, is 2^{64} . For all these 2^{64} pairs, the corresponding y_2 values reside in the space of special values which is of size 2^{64} , and hence it is expected that there exists an internal collision $(m_1, m_2), (m_1^*, m_2^*)$, where $(m_1, m_2) \in S_1$ and $(m_1^*, m_2^*) \in S_2$.

⁷ Note that the first blocks in the two structures must differ, since otherwise there will be no collision between the structures as keyless AES is a permutation.

On the other hand, if $E_k(0)$ is not of the form $m_1 \oplus (z, w, z, w)$, then $E_k(0) \oplus m_1$ and $E_k(0) \oplus m_1 \oplus (1, 1, 1, 1)$ are not of the special form, and thus, there is no restriction on the y_2 values. Therefore, for the 2^{64} pairs of the form $(m_1, m_2), (m_1^*, m_2^*)$, where $(m_1, m_2) \in S_1$ and $(m_1^*, m_2^*) \in S_2$, the corresponding y_2 values reside in a space of size 2^{128} , and hence the probability that these pairs contain an internal collision is extremely low.

The attack so far recovers 64 bits of $E_k(0)$, and the rest of the bits can be found by exhaustive key search. The data complexity of the attack is 2^{97} two-block messages, the time complexity is 2^{97} MAC evaluations, and the memory requirement is only 2^{32} 128-bit blocks.

4.1.1 A Memoryless Variant of the Attack

If the adversary is allowed to ask for the MAC evaluation of adaptively chosen messages, the memory requirement of the attack can be further reduced to a few cells of memory, using Floyd's cycle finding algorithm [14].

The attack algorithm is as follows:

1. For each of the 2^{64} possible 128-bit blocks of the form $m_1 = (x', y', 0, 0)$, do the following:
 - (a) Ask for the MAC evaluation of a sequence of adaptively chosen two-block messages, defined as follows:
 - $m^0 = (m_1, 0)$.
 - For $j > 0$, if $MAC(m^{j-1}) = (x, y, z, w)$, then $m^j = (m_1 \oplus (x, y, x, y), (z, w, z, w))$.
 - (b) Use Floyd's cycle finding algorithm to find a cycle in the sequence of tag values. If the algorithm does not terminate within 2^{32} steps, discard the guess of m_1 . If the algorithm terminates, deduce that $E_k(0)$ is of the form $(z', w', z', w') \oplus m_1$ for some z', w' .
2. Only one value of m_1 is expected to survive. For this value, find z', w' by exhaustive search on their 2^{64} possible values.

The idea behind this algorithm is the same as the idea behind the basic algorithm. If $E_k(0)$ is of the form $(z', w', z', w') \oplus m_1$, then all the tag values in the generated sequence lie in a smaller space of size 2^{64} , and thus Floyd's algorithm is expected to terminate in about 2^{32} steps. If $E_k(0)$ is not of the desired form, then the tag values of the sequence lie in the entire space of size 2^{128} , and thus no short cycle is expected.

The data and time complexities of the algorithm are about 2^{96} , and the memory requirement is only a few memory cells.

4.2 Application to the Marvin MAC

An example for a CBC MAC construction that can be attacked using the generic algorithm described above is the Marvin MAC designed by Simplicio et al. [21, 22]. Marvin is based on Pelican, and uses a block cipher E and its reduced-round variant f to process the data. Marvin itself was proposed with E being AES and f being 4 rounds of keyless AES.

Marvin initializes the first internal state y_0 (called R in [21, 22]) by computing $y_0 = E_k(0||c) \oplus c$ for some non-zero constant c . Then each block m_i of the message is encrypted independently to

$$F_i(m_i) = AES4(m_i \oplus y_0 \cdot c_i),$$

where c_i are a series of known constants, \cdot denotes multiplication over $GF(2^n)$, and $AES4$ denotes a 4-round keyless variant of AES. The values $F_i(m_i)$ are then XORed together to obtain

$$y_\ell = F_1(m_1) \oplus \dots \oplus F_\ell(m_\ell),$$

and the finalization is the encryption (under k) of $y_0 \oplus y_\ell \oplus |M|$ (where $|M|$ is the message length). See [21] for the exact description of Marvin algorithm.

It seems that in Marvin, the simplest form of an internal collision an adversary can obtain is achieved by altering the values of two message words m_i, m_j and hoping that the differences generated in $F_i(m_i)$ and $F_j(m_j)$ will cancel each other. Such an internal collision provides the adversary with two input/output pairs for 4-round AES, in which the input differences are known, and it is known that the output differences are equal (but it is not known what is this output difference). This information is insufficient either for any of our other attacks or for the attacks presented by Yuan et al. [24], and thus, it is unclear whether these attacks can be applied to Marvin.

On the other hand, it can be easily seen that the self-similarity attack presented above applies to Marvin (up to a small modification). Indeed, note that if the XOR difference between the two halves of the value y_0 is known, then the differences between the halves of $y_0 \cdot c_i$ are known as well. Hence, for each possible guess of the difference between the halves of y_0 , the adversary can construct a pool of 2^{32} two-block messages (m_1, m_2) with the appropriate differences between the halves of the two blocks, such that if the guess is correct, the two halves of $m_1 \oplus y_0 \cdot c_1$ and of $m_2 \oplus y_0 \cdot c_2$ are equal. Then, the attack can be completed as in the basic attack presented above. The memoryless variant of the attack can be modified similarly.

Thus, our attack on Marvin allows to recover the initial value y_0 and thus to perform an almost universal forgery of the MAC with data and time complexity of 2^{96} and only a few memory cells. The attack extends immediately without any change, even if the number of keyless AES rounds is increased.⁸

We note that an additional advantage of the self-similarity attack is that it performs similarly even if the adversary can control only part of the state bytes. This allows to apply the self-similarity attack to other variants of the ALRED construction, such as an Alpha-MAC [9] variant with an arbitrary number of keyless AES rounds as the internal block cipher. The easy adaptation of the attack to such scenarios is described in [12].

5 Generic Time/Memory Tradeoff Attacks

The third class of almost universal forgery attacks we present is generic time/memory tradeoff attacks, which apply whenever ALRED uses a keyless block cipher (possibly different than AES). Our attacks are similar in spirit to Daemen's chosen

⁸ We note that this attack does not violate the security claims of Marvin, as these ensure security only as long as the number of queries is below the birthday bound.

plaintext attack [7] on the Even-Mansour encryption scheme [13] and to the attack of Coppersmith et al. [6] on MacDES.

We first present a basic attack which requires data, memory and online time complexities of slightly more than $2^{\ell_w/2}$, and a preprocessing of 2^{ℓ_w} operations. Then, we show a tradeoff that allows to reduce the preprocessing time, at the expense of increasing the data and time complexities.

5.1 The Basic Attack on Pelican-type Constructions

In this attack, we assume that the structure of the MAC is as shown in Figure 2 above, where E is some block cipher (possibly provably secure), and f is some keyless permutation (which is possibly perfectly random). Note that for such constructions, both E and f operate on ℓ_w -bit blocks.

The algorithm of the basic attack is as follows.

1. **Preprocessing phase.** Choose an arbitrary non-zero ℓ_w -bit value Δ , and perform the following operations for each ℓ_w -bit value C :
 - (a) Compute the values $P = f^{-1}(C)$, $P^* = f^{-1}(C \oplus \Delta)$.
 - (b) Check whether the first $\ell_w/2$ bits of $P \oplus P^*$ are zeros. If this is the case, store the pair (P, P^*) in a hash table indexed by the last $\ell_w/2$ bits of the difference $P \oplus P^*$.
2. **On-line phase.**
 - (a) Ask for the MAC evaluation of two structures S_1 and S_2 of $2^{\ell_w/2}$ two-block messages each, such that:
 - In S_1 , in all messages, the first $\ell_w/2$ bits of the block m_1 are zeros, the remaining $\ell_w/2$ bits of m_1 assume all the $2^{\ell_w/2}$ possible values, and the block m_2 is fixed to zero.
 - In S_2 , the first $\ell_w/2$ bits of the block m_1 are zeros, the remaining $\ell_w/2$ bits of m_1 assume all the $2^{\ell_w/2}$ possible values, and the block m_2 is fixed to Δ .
 - (b) Insert the tags into a hash table, and search for an internal collision (i.e., collision before the final application of $E_k(\cdot)$).
 - (c) If the messages $(m_1, m_2) \in S_1$ and $(m_1^*, m_2^*) \in S_2$ form an internal collision, look at the table prepared in the preprocessing phase, in the cell corresponding to $m_1 \oplus m_1^*$. For each pair (P, P^*) in the cell, assume that $E_k(0) = P \oplus m_1$, and verify the guess using the MAC evaluation of one of the messages in the data set.

5.1.1 Analysis of the algorithm

The table constructed in the preprocessing phase contains all the pairs of input/output values of f in which the output difference is Δ , and the first $\ell_w/2$ bits of the input difference are zeros. It is expected that the number of pairs (P, P^*) in the table is close to $2^{\ell_w/2}$, and that for each possible value of $P \oplus P^*$, the table contains at most several pairs corresponding to that difference.

The idea behind the algorithm is that if $(m_1, m_2) \in S_1$ and $(m_1^*, m_2^*) \in S_2$ form an internal collision, then the inputs to the function f in the first step of the chaining processes of (m_1, m_2) and of (m_1^*, m_2^*) must form a pair in the pre-constructed table.

To prove this claim, denote the intermediate values in the chaining processes of (m_1, m_2) and of (m_1^*, m_2^*) by (y_0, y_1, y_2) , and (y_0^*, y_1^*, y_2^*) , respectively. Denote the inputs to the function f in the first step of the chaining processes by x and x^* . In order to show that the pair (x, x^*) appears in the table, we have to show that the difference between the corresponding outputs of f , which are y_1 and y_1^* , is Δ , and that the first $\ell_w/2$ bits of $x \oplus x^*$ are zeros.

This indeed follows from the structure of S_1 and S_2 . On the one hand, since (m_1, m_2) and (m_1^*, m_2^*) form an internal collision, we have $y_2 = y_2^*$. Since the function $f(\cdot)$ is invertible, this implies that $y_1 \oplus y_1^* = m_2 \oplus m_2^* = \Delta$, as required. On the other hand, by definition of the ALRED construction, $y_0 = y_0^* = E_k(0)$, and thus, $x \oplus x^* = m_1 \oplus m_1^*$. By the choice of the structures, the first $\ell_w/2$ bits of both m_1 and m_1^* are zeros, and thus the first $\ell_w/2$ bits of $x \oplus x^*$ are zeros as well.

Hence, the pair (x, x^*) indeed appears in the pre-computed table, and as mentioned before, there are likely to be only a few pairs in the table with the difference $m_1 \oplus m_1^*$. Since $x = E_k(0) \oplus m_1$, each such pair yields a single suggestion for the initial state $y_0 = E_k(0)$.

The data complexity of the attack is $2^{\ell_w/2+1}$ chosen messages, and the time complexity is dominated by evaluating the MAC value of the messages. The memory required for the attack is $\ell_w \cdot 2^{\ell_w/2+1}$ bits. The attack succeeds for sure once an internal collision is encountered, and hence, the success probability of the attack is $1 - 1/e$ (which is the probability that the given data set during the on-line phase of the attack contains an internal collision).

5.2 Preprocessing/Data Tradeoff

If the preprocessing time available to the adversary is smaller than 2^{ℓ_w} , she can still mount a variant of the attack, but with a higher data complexity.

Assume that the available preprocessing time is 2^{ℓ_w-t} . In such situation, during the preprocessing phase, the adversary is able to check only a 2^{-t} fraction of the pairs $(C, C \oplus \Delta)$, and thus, the table contains only a small portion of the pairs (P, P^*) with the prescribed input and output differences. As a result, the adversary cannot assure that the pair (x, x^*) obtained from the internal collision appears in the pre-computed table. However, if the adversary will examine 2^t random internal collisions, then with probability of $1 - 1/e$, one of them would appear in the table and allow to retrieve the initial state $E_k(0)$.

Formally, the attack algorithm is similar to the original attack, with the following changes:

1. In the preprocessing phase, the adversary checks 2^{ℓ_w-t} pairs of the form $(C, C \oplus \Delta)$, and stores the corresponding plaintext pair (P, P^*) in the table if the first $(\ell_w - t)/2$ bits of $P \oplus P^*$ are zeros. The table is indexed by the $(\ell_w + t)/2$ last bits of $P \oplus P^*$.
2. In the on-line phase, in both structures, only the $(\ell_w - t)/2$ first bits of m_1 are fixed to zeros, while the remaining $(\ell_w + t)/2$ bits assume all the $2^{(\ell_w+t)/2}$ possible values. Thus, the size of the structures is increased to $2^{(\ell_w+t)/2}$, and the adversary expects to obtain 2^t internal collisions. For each collision, the adversary checks the pre-computed table like in the basic attack, and it is expected that in one of the internal collisions, the pair (x, x^*) exists in the table and allows to retrieve $E_k(0)$.

The preprocessing phase of the attack requires $2^{\ell_w - t}$ operations, and the data and time complexities are $2^{(\ell_w + t)/2 + 1}$. The memory requirement is $\ell_w \cdot 2^{(\ell_w + t)/2 + 1}$ bits.

Since such attack is possible for any value of t , this yields the tradeoff curve:

$$PD^2 = 2^{2\ell_w},$$

where P is the preprocessing time, and D is the data complexity. In particular, the adversary can mount an attack with overall complexity of $2^{2\ell_w/3}$, without using any additional preprocessing time.

We note that like in the case of the self-similarity attack presented in Section 4, the generic time/memory tradeoff attack can be adapted to scenarios in which the adversary can control only part of the state bytes, such as Alpha-MAC-type constructions. The adaptation of the attack to such scenarios is described in [12].

References

1. Behnam Bahrak and Mohammad Reza Aref, *A Novel Impossible Differential Cryptanalysis of AES*, proceedings of the Western European Workshop on Research in Cryptology 2007, Bochum, Germany, 2007.
2. Eli Biham and Nathan Keller, *Cryptanalysis of Reduced Variants of Rijndael*, unpublished manuscript, 1999.
3. Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen and Elmar Tischhauser, *ALE: AES-Based Lightweight Authenticated Encryption*, presented at Fast Software Encryption 2013, to appear in Lecture Notes in Computer Science, Springer.
4. Charles Bouillaguet, Orr Dunkelman, Gaëtan Leurent, and Pierre-Alain Fouque, *Another Look at Complementarity Properties*, proceedings of Fast Software Encryption 2010, Lecture Notes in Computer Science 6147, pp. 347–364, Springer, 2010.
5. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque, *Automatic Search of Attacks on Round-Reduced AES and Applications*, Advances in Cryptography, proceedings of CRYPTO 2011, Lecture Notes in Computer Science 6841, pp. 169–187, Springer, 2011.
6. Don Coppersmith, Lars R. Knudsen, and Chris J. Mitchell, *Key Recovery and Forgery Attacks on the MacDES MAC Algorithm*, Advances in Cryptography, proceedings of CRYPTO 2000, Lecture Notes in Computer Science 1880, pp. 184–196, Springer, 2000.
7. Joan Daemen, *Limitations of the Even-Mansour Construction*, proceedings of Asiacrypt 1991, Lecture Notes in Computer Science 739, pp. 495–498, Springer, 1991.
8. Joan Daemen and Vincent Rijmen, *The design of Rijndael: AES — the Advanced Encryption Standard*, Springer, 2002.
9. Joan Daemen and Vincent Rijmen, *A New MAC Construction ALRED and a Specific Instance, ALPHA-MAC*, proceedings of Fast Software Encryption 2005, Lecture Notes in Computer Science 3557, pp. 1–17, Springer, 2005.
10. Joan Daemen and Vincent Rijmen, *The Pelican MAC Function*, IACR ePrint report 2005/088.
11. Joan Daemen and Vincent Rijmen, *Refinements of the ALRED construction and MAC security claims*, IET Information Security **4**(3), pp. 149–157, 2010.
12. Orr Dunkelman, Nathan Keller, and Adi Shamir, *ALRED Blues: New Attacks on AES-Based MAC's*, IACR ePrint report 2011/095.
13. Shimon Even and Yishai Mansour, *A Construction of a Pseudorandom Cipher from Single Pseudorandom Permutation*, Journal of Cryptology **10**(3), pp. 151–162, 1997.
14. Donald Knuth, *The Art of Computer Programming*, 2nd Edition, Vol. 2, pp. 7, Addison-Wesley, 1981.
15. Tri Van Le, Rüdiger Sparr, Ralph Wernsdorf, and Yvo Desmedt, *Complementation-like and Cyclic Properties of AES Round Functions*, proceedings of 4-th AES conference, Lecture Notes in Computer Science 3373, pp. 128–141, Springer, 2004.
16. Jiqiang Lu, Orr Dunkelman, Nathan Keller, Jongsung Kim, *New Impossible Differential Attacks on AES*, proceedings of INDOCRYPT 2008, Lecture Notes in Computer Science 5365, pp. 279–293, Springer, 2008.

17. Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi, *Improved Impossible Differential Cryptanalysis of 7-Round AES-128*, proceedings of Indocrypt 2010, Lecture Notes in Computer Science 6498, pp. 282–291, Springer, 2010.
18. Kazuhiko Minematsu and Yukiyasu Tsunoo, *Provably Secure MACs from Differentially-Uniform Permutations and AES-Based Implementations*, proceedings of Fast Software Encryption 2006, Lecture Notes in Computer Science 4047, pp. 226–241, Springer, 2006.
19. Thomas Peyrin, Yu Sasaki, and Lei Wang, *Generic Related-Key Attacks for HMAC*, Advances in Cryptology, proceedings of ASIACRYPT 2012, Lecture Notes in Computer Science 7658, pp. 580–597, Springer, 2012.
20. Yu Sasaki, *Cryptanalyses on a Merkle-Damgrd Based MAC - Almost Universal Forgery and Distinguishing-H Attacks*, Advances in Cryptography, proceedings of EUROCRYPT 2012, Lecture Notes in Computer Science 7237, pp. 411–427, Springer, 2012.
21. Marcos A. Simplicio Jr., Pedro d'Aquino F. F. S. Barbuda, Paulo S. L. M. Barreto, Tereza Cristina M. B. Carvalho, and Cintia B. Margi, *The MARVIN message authentication code and the LETTERSOUP authenticated encryption scheme*, Security and Communication Networks **2**(2), pp. 165–180, 2009.
22. Marcos A. Simplicio Jr., Paulo S. L. M. Barreto, and Tereza Cristina M. B. Carvalho, *Revisiting the Security of the Alred Design*, proceedings of Information Security Conference (ISC) 2010, Lecture Notes in Computer Science 6531, pp. 69–83, Springer, 2011.
23. Lei Wang, Kazuo Sakiyama, Iwamasa Nishikado, and Kazuo Ohta, *Security Evaluation of PC-MAC-AES*, CRYPTREC report, 2011. Available online at: http://www.cryptrec.go.jp/estimation/techrep_id2005.pdf.
24. Zheng Yuan, Wei Wang, Keting Jia, Guangwu Xu, and Xiaoyun Wang, *New Birthday Attacks on Some MACs Based on Block Ciphers*, Advances in Cryptology, proceedings of CRYPTO 2009, Lecture Notes in Computer Science 5677, pp. 209–230, Springer, 2009.