

Universal Forgery and Multiple Forgeries of MergeMAC and Generalized Constructions

Tetsu Iwata¹, Virginie Lallemand², Gregor Leander², and Yu Sasaki³

¹ Nagoya University, Nagoya, Japan, tetsu.iwata@nagoya-u.jp

² Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany,
firstname.lastname@rub.de

³ NTT Secure Platform Laboratories, Tokyo, Japan, sasaki.yu@lab.ntt.co.jp

Abstract. This article presents universal forgery and multiple forgeries against MergeMAC that has been recently proposed to fit scenarios where bandwidth is limited and where strict time constraints apply. MergeMAC divides an input message into two parts, $m\|\tilde{m}$, and its tag is computed by $\mathcal{F}(\mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{m}))$, where \mathcal{P}_1 and \mathcal{P}_2 are PRFs and \mathcal{F} is a public function. The tag size is 64 bits. The designers claim 64-bit security and imply a risk of accepting beyond-birthday-bound queries.

This paper first shows that it is inevitable to limit the number of queries up to the birthday bound, because a generic universal forgery against CBC-like MAC can be adopted to MergeMAC.

Afterwards another attack is presented that works with a very few number of queries, 3 queries and $2^{58.6}$ computations of \mathcal{F} , by applying a preimage attack against weak \mathcal{F} , which breaks the claimed security.

The analysis is then generalized to a MergeMAC variant where \mathcal{F} is replaced with a one-way function \mathcal{H} .

Finally, multiple forgeries are discussed in which the attacker's goal is to improve the ratio of the number of queries to the number of forged tags. It is shown that the attacker obtains tags of q^2 messages only by making $2q - 1$ queries in the sense of existential forgery, and this is tight when q^2 messages have a particular structure. For universal forgery, tags for $3q$ arbitrary chosen messages can be obtained by making $5q$ queries.

Keywords: MergeMAC, universal forgery, multiple forgeries, public finalization, preimage, splice-and-cut, Hellman's time-memory tradeoff

1 Introduction

Fully aware of the rapid expansion of pervasive computing and of what is usually referred to as the Internet of Things (IoT), symmetric cryptographers proposed solutions to ensure appropriate security for the new use cases. Lightweight cryptography became a hot research topic as it was understood that finding the correct compromise between security and efficiency was challenging. Many cryptographers – whether from academic community, from government agencies or from private companies – proposed new primitives, starting with a myriad

of lightweight block ciphers (like Present [9], Prince [10], SKINNY [5], CLEFIA [19] and Simon [4] just to name a few). While in comparison the design of other primitives seems less popular, some lightweight stream ciphers and hash functions were also proposed⁴. The design of Message Authentication Codes was also addressed, with the publication of SipHash [3], Chaskey [16], of the MAC mode LightMAC [15] and very recently of MergeMAC [1].

MergeMAC was proposed by Ankele, Böhl and Friedberger at ACNS18. As for all the Message Authentication Codes, it is meant at providing integrity and authenticity by producing a fixed-length tag from a message and a secret key. MergeMAC was designed to fit extremely constrained environments with strict time requirements and limited bandwidth, and in particular for the Controller Area Network (CAN) bus⁵. The necessity to bring authentication for this later scenario comes from the fact that some of the components at play are also connected to the Internet, creating remote attack opportunities. The MAC construction proposed by Ankele et al. is based on 3 components: 2 variable input-length Pseudo-Random Functions (parameterized by independent keys), and a so-called Merge function. Each PRF modifies one part of the input message, and the two outputs are recombined by the merge function.

Our Contributions. In this paper, we investigate the resistance of MergeMAC against forgery attacks in different scenarios.

First, we show that an attacker can take advantage of its special structure and forge messages by adapting the universal forgery attack proposed by Jia et al. at CANS09 [14], and this regardless of the choice of the PRF or of the MERGE function. This first technique has a data complexity slightly higher than the limit set by the designers, which shows its tightness.

Our second result is a universal forgery that breaks the security claim of MergeMAC by only requiring 3 queries to forge a tag. This attack exploits the details of the MERGE function (in particular its low diffusion and its feed-forward structure) to perform a perimage attack using the splice-and-cut Meet-in-the-Middle technique [2].

We also discuss the possibility of forgery attacks in the situation where the MERGE function is an ideal one-way function. We call this construction MergeMAC^{OW}. By using the fact that it is public and can be evaluated offline, we deduce possible tradeoffs that can be more practical than the generic attack, but still less efficient than the one using the specificities of the MERGE function.

Our last contribution is the analysis of MergeMAC to forge multiple tags: first in the case of existential forgeries, and next in the case of universal ones. The problem is known as the MAC reforgeability [8], where one takes advantage of the computational efforts for the first forgery to reduce the complexity for the subsequent forgeries. In the case of existential forgery, we show that it is

⁴ We refer to [7] for a thorough review of lightweight constructions.

⁵ The CAN bus is the standard message system used in most modern cars to connect together the different components (engine control unit, airbags, audio system, doors, etc).

possible to forge $(q - 1)^2$ tags by making $2q - 1$ queries, i.e., we can obtain more forgeries than the number of queries. We also discuss the tightness on the number of queries. In the case of universal forgery, we show that we can forge q tags by making $2q - 1$ queries, and we also show that this can be improved when q is divisible by 3. We remark that no security claim has been made by the designers regarding multiple forgeries, and hence our analyses give the first insight about the security of MergeMAC in this attack scenario.

Paper Outline. Section 2 introduces specification of MergeMAC. Section 3 presents universal forgery against MergeMAC. Section 4 generalizes the analysis to MergeMAC^{OW}. Section 5 discusses multiple forgeries.

2 Specification of MergeMAC

MergeMAC is a new MAC construction that has been recently proposed by Ankele, Böhl and Friedberger [1] to fit scenarios where bandwidth is limited and where strict time constraints apply. More precisely, the designers aim for an efficient solution for authenticating messages on the CAN⁶ bus, a communication system widely used in modern cars to manage the different electronic components. In addition to the bandwidth constraint inherent to the CAN technology, the fact that the components in questions are as critical as brakes or airbags makes it plain that the MAC must have a low latency.

To meet these requirements, the solution proposed by Ankele et al. uses different techniques⁷: for instance, it saves bandwidth by not transmitting some low-entropy bits of the message, and it can be build from lightweight ciphers such as PRINCE [10] to limit the latency. One of the design ideas that impacted the most their construction was the wish to speed up MAC verifications by storing frequently needed intermediate parts in the cache instead of computing them again. This point leads them to a construction that combines the output of two PRFs (each operating on a part of the input message) into a merging function (see Figure 1). The authors propose to precompute and cache the PRF outputs, and stress that this solution only requires simple computations, an advantage in comparison to other cache-able construction.

In what follows, we use the same notation as in the specification. As shown in Figure 1, the input of MergeMAC is first split in two parts, m and \tilde{m} , each entering one of the PRFs $\mathcal{P}_1, \mathcal{P}_2$. These PRFs are of variable input length and depend on two k -bit keys K_1, K_2 . The n -bit outputs of $\mathcal{P}_1, \mathcal{P}_2$ are denoted by ρ and $\tilde{\rho}$, respectively, and both enter the MERGE function which returns the n -bit *tag*.

The authors state that any MAC scheme that is a secure PRF (as for example AES-CMAC or Chaskey) can be used to instantiate \mathcal{P}_1 and \mathcal{P}_2 . To fit in with the constrained environment use-case, they propose to use Present [9] or Prince [10]

⁶ Controller Area Network.

⁷ We refer to the specification [1] for details.

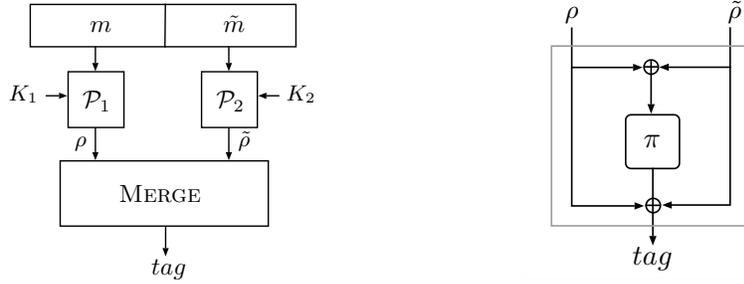


Fig. 1. MergeMAC construction (left) and MERGE function (right).

in CMAC mode. The MERGE function follows a Davies-Meyer construction with $\rho \oplus \tilde{\rho}$ as input: $tag = \pi(\rho \oplus \tilde{\rho}) \oplus (\rho \oplus \tilde{\rho})$, where π is a permutation on n bits. The authors define π as a 3-round variant of CHASKEY [16] operating on 64-bit blocks (note that the reduced block size is required to achieve compatibility with the block size of Present/Prince). The other changes made to the round function can be read in Figure 2.

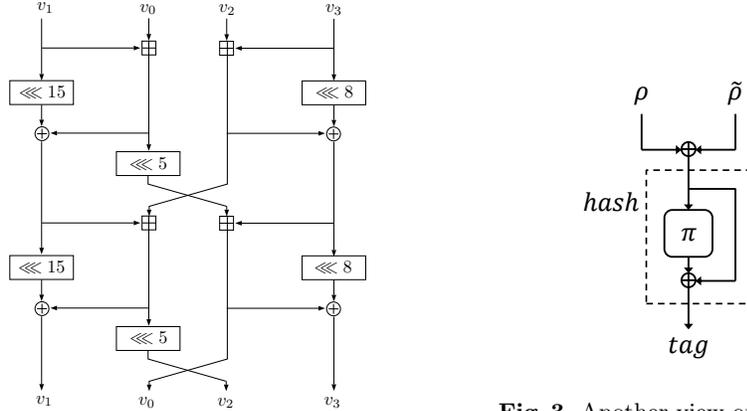


Fig. 3. Another view of MERGE.

Fig. 2. One round of the π function used in MERGE. Each wire represents 16 bits.

The instantiation of MERGE can be viewed as an XOR-then-hash construction, i.e. two inputs are XORed then the result is processed by a public hash function. We will use this view when it is convenient to understand our analysis.

Security Claim. The authors claim that their construction is a provably secure MAC, and in particular that it reaches n -bit security against forgery attacks. Their proof requires that \mathcal{P}_1 and \mathcal{P}_2 are secure PRFs and that the MERGE function satisfies Random Input Indistinguishability.

To prove this last point, they provide a security analysis of the MERGE function with respect to various types of attacks. An argument that they use repeatedly in the discussion is that, since the input of the MERGE function is unknown and comes from a PRF, an attacker cannot force a specific property on it, which removes the threat of many attacks such as the ones based on differentials.

The authors also claim that finding a preimage of MERGE is “*as hard as exhaustively guessing the internal state after the initial PRFs*” so that MergeMAC is resistant to attacks based on Meet-in-the-Middle techniques. They justify the resistance to more advanced MitM attacks by the fact that “*MergeMAC does not implement an inverse function for the merging function MERGE*”⁸.

The designers claim the security for each underlying primitive as in Table 1. The designers also notice the risk of using a small block size against birthday attacks demonstrated by the Sweet32 attack [6], and suggest that the amount of data blocks that are processed by the initial PRFs of MergeMAC must be limited appropriately. Although the designers do not specify the details of the appropriate level, Table 1 may be interpreted as security claims under the condition that key is renewed after the number of queries reaches the birthday bound.

Table 1. Security claims according to the underlying primitives [1, Table 1].

Underlying BC	Block size	Key size	Existential forgery resistance
PRESENT	64	80	2^{-64}
PRESENT	64	128	2^{-64}
PRINCE	64	128	2^{-64}

3 Universal Forgery against MergeMAC

In Sect. 3.1, we show that limiting the number of queries up to the birthday bound is almost tight because a generic universal forgery can be applied irrelevant to the choice of PRFs and the MERGE function. In Sect. 3.2, we present an attack only with 3 queries by exploiting the weak mixing effect of π .

⁸ As we will show later in the paper, this argument turns wrong.

3.1 Generic Attacks with High Data Complexity

Jia et al. proposed universal forgery with the birthday-bound complexity that generally works against CBC-like MACs and PMAC-like MACs [14]. The attack can be directly applied to MergeMAC. Let $m\|\tilde{m}$ be a challenged message. The goal of the attacker is producing the tag t for this message without querying $m\|\tilde{m}$. The attack works as follows.

1. For distinct $x_i, 1 \leq i \leq 2^{n/2}$, query $x_i\|\tilde{m}$ to obtain a tag t_i .
2. For distinct $\tilde{y}_j, 1 \leq j \leq 2^{n/2}$, query $m\|\tilde{y}_j$ to obtain a tag t_j .
3. Find a collision of t_i and t_j . Let \hat{i}, \hat{j} be the indices of the colliding pair.
4. Query $x_{\hat{i}}\|\tilde{y}_{\hat{j}}$ to obtain the corresponding tag t' .
5. Output t' as a valid tag for $m\|\tilde{m}$.

Analysis. We view MERGE as Fig. 3. We first evaluate the attack by replacing the hash function in MERGE with a permutation. Then, a collision of the tag implies a collision of the XOR of two PRF's outputs, namely

$$\mathcal{P}_1(x_i) \oplus \mathcal{P}_2(\tilde{m}) = \mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{y}_j). \quad (1)$$

Therefore,

$$\mathcal{P}_1(x_i) \oplus \mathcal{P}_2(\tilde{y}_j) = \mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{m}), \quad (2)$$

which shows that the tag for $m\|\tilde{m}$ is equal to the tag for $x_i\|\tilde{y}_j$.

The attack requires $2 \cdot 2^{n/2} + 1$ queries, which is roughly $O(2^{n/2})$ queries (and the computational cost of $O(2^{n/2})$ memory accesses to operate on the data).

Consideration of Error Probability. We now evaluate the case with MERGE following the actual construction. Then, a collision of t_i and t_j does not imply Eq. (1). Suppose that $\mathcal{P}_1(x_i) \oplus \mathcal{P}_2(\tilde{m}) = \alpha, \mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{y}_j) = \beta, \alpha \neq \beta$ and $\text{MERGE}(\alpha) = \text{MERGE}(\beta)$. Then Eq. (2) becomes

$$\mathcal{P}_1(x_i) \oplus \mathcal{P}_2(\tilde{y}_j) = \mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{m}) \oplus \alpha \oplus \beta,$$

with unknown α and β . Hence, a tag for $m\|\tilde{m}$ cannot be computed.

This issue can be solved by iterating the attack (finding a collision between t_i and t_j) several times until the attacker probabilistically hits the case with $\alpha = \beta$. For an n -bit to n -bit function, the number of multicollisions can be upper bounded by n . Hence, by iterating the entire attack procedure n times, the attacker can predict the correct tag with probability $1/n$. The attack can be improved slightly. When the attacker makes $2^{n/2}$ queries of $x_i\|\tilde{m}$ and $m\|\tilde{y}_j$, the attacker can make $2^{n/2} \cdot n/2$ queries. This generates n pairs of \hat{i}, \hat{j} , which is sufficient for the attack. In the end, the complexity of the application of the generic attack is upper bounded by $O(n2^{n/2})$.

The average complexity is smaller than the upper bound. The range size of an n -bit to n -bit function is e^{-1} times smaller than the domain size. Hence, an output value should have e distinct preimages on average. In the end, the average complexity of the generic attack is $O(e2^{n/2})$, which is $O(2^{n/2})$.

Complexity for MergeMAC. In MergeMAC, n is 64. Hence, the attack complexity is about $e \cdot 2^{32} \leq 2^{34}$. Given that the authors imply to limit the number of queries up to an appropriate level in the context of the Sweet32 attack, the generic universal forgery may not break the claimed security but show the tightness of their bounding data complexity.

3.2 Universal Forgery with Very Low Data Complexity

In this section we present a universal forgery with a very low data complexity but with a higher offline computational cost than that of the generic attack.

Attack Overview. The idea is to exploit the fact that the MERGE function mixes the data very lightly. To be more precise, we present a preimage attack against the 3-round variant of Chaskey with the feed-forward operation used in MergeMAC.

Recall that the core idea of the generic attack is to obtain some information on the input to the MERGE function by finding collisions of the tag. To reduce the data complexity, we avoid searching for a collision (with many queries), instead invert the MERGE function by spending offline computational cost. Note that this strategy can only be applied when the finalization function is public, hence the following attack shows another feature particular to MergeMAC.

Suppose that $m\|\tilde{m}$ is a target. If the attacker recovers $\mathcal{P}_1(m)\|\mathcal{P}_2(\tilde{m})$, the tag can be forged by processing MERGE function offline. We notice that $\mathcal{P}_1(m)\|\mathcal{P}_2(\tilde{m})$ can be recovered by 3 queries and 3 executions of the preimage attack. Let x and \tilde{y} be the former half and the latter half of an arbitrary chosen message. Then, the attacker queries three messages $x\|\tilde{m}$, $m\|\tilde{y}$ and $x\|\tilde{y}$ to obtain the corresponding tags t_1 , t_2 , and t_3 that are expressed as follows.

$$\begin{aligned} t_1 &\leftarrow \pi(\mathcal{P}_1(x) \oplus \mathcal{P}_2(\tilde{m})) \oplus \mathcal{P}_1(x) \oplus \mathcal{P}_2(\tilde{m}) \\ t_2 &\leftarrow \pi(\mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{y})) \oplus \mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{y}) \\ t_3 &\leftarrow \pi(\mathcal{P}_1(x) \oplus \mathcal{P}_2(\tilde{y})) \oplus \mathcal{P}_1(x) \oplus \mathcal{P}_2(\tilde{y}) \end{aligned}$$

Suppose that for a given o , the attacker can execute a preimage attack to find i such that $o \leftarrow \pi(i) \oplus i$. Then, by finding preimages of t_1 , t_2 , and t_3 , the attacker obtains $\mathcal{P}_1(x) \oplus \mathcal{P}_2(\tilde{m})$, $\mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{y})$ and $\mathcal{P}_1(x) \oplus \mathcal{P}_2(\tilde{y})$. The sum of those 3 values equals to $\mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{m})$, then the attacker can compute the tag offline.

Preimage attacks on cryptographic functions have been discussed deeply. We follow the framework of meet-in-the-middle preimage attacks [2,18]. Due to the construction, the attack framework is closer to the preimage attack against the block-cipher based compression functions first demonstrated against AES in the Davies-Meyer mode [17].

Meet-in-the-Middle Preimage Attacks. Meet-in-the-Middle (MitM) attack [11] was originally proposed to recover a key of a block cipher. When a ciphertext c is computed with two encryption algorithms E_1 and E_2 with independent keys

k_1 and k_2 , i.e. $c = E_{2,k_2} \circ E_{1,k_1}(p)$, k_1 and k_2 can be recovered with a complexity $\min\{|k_1|, |k_2|\}$ instead of $|k_1| + |k_2|$.

Sasaki [17] presented a framework to apply the MitM attack to $t = P(x) \oplus x$ for recovering unknown x for a given t , where P consists of an iteration of a round function \mathcal{R} with imperfect diffusion. Suppose that \mathcal{R} consists of r rounds, namely t is computed from x as

$$V_0 \leftarrow x, \quad V_i \leftarrow \mathcal{R}(V_{i-1}) \text{ for } i = 1, 2, \dots, r, \quad t \leftarrow V_r \oplus V_0.$$

The splice-and-cut technique [2] allows the attacker to regard the first and the last rounds as consecutive rounds. Indeed, t is computed by $V_r \oplus V_0$. For any fixed t , computing V_0 (resp. V_r) immediately fixes V_r (resp. V_0).

The overview of the attack framework is illustrated in Figure 4. The attacker first determines a starting round p and matching round q , such that the computation from V_{p-1} to V_q (forward computation) and the computation from V_{p-1} to V_0 , $V_r = t \oplus V_0$, and from V_r to V_q (backward computation) can be independently performed. The results of the two computations are matched on V_q .

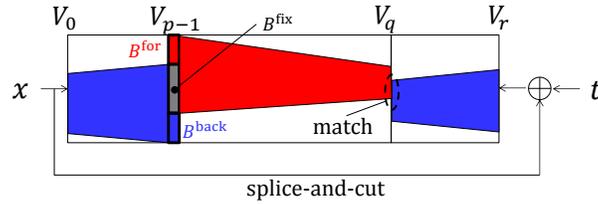


Fig. 4. Overview of meet-in-the-middle preimage attacks for $t = H(x) \oplus x$.

More precisely, each bit of the state V_{p-1} is classified into three groups:

- B^{for} : all possible values are examined during the forward computation.
- B^{back} : all possible values are examined during the backward computation.
- B^{fix} : the value is fixed during the independent computations.

Suppose that the value of B^{fix} is fixed. The attacker, for each possible value of B^{for} , proceeds the forward computation without using the value of B^{back} . Because B^{back} is unknown, the forward computation cannot compute all bits of the state. However, when the diffusion of \mathcal{R} is imperfect, the partial computation can be performed for a few rounds (until round q). Independently, the attacker computes the backward computation by examining all possible values of B^{back} without using the value of B^{for} (up to V_q). For a correct combination of B^{fix} , B^{for} , B^{back} , the partially computed values always match at V_q , and the correct value of V_{p-1} can be recovered efficiently. Finally, the MitM attack is iterated for the exhaustive guesses of B^{fix} . The algorithmic description is given in Alg. 3.

Algorithm 1 Meet-in-the-middle preimage attack for $t = P(x) \oplus x$

Require: $t, p, q, B^{\text{fix}}, B^{\text{for}}, B^{\text{back}}$

Ensure: x

```

1: for all candidates of  $B^{\text{fix}}$  do
2:   for all candidates of  $B^{\text{for}}$  do
3:     Partially compute  $V_i \leftarrow \mathcal{R}(V_{i-1})$  for  $i = p, p+1, \dots, q$ , and store the result
       in a list  $L$ .
4:   end for
5:   for all candidates of  $B^{\text{back}}$  do
6:     Partially compute  $V_{i-1} \leftarrow \mathcal{R}^{-1}(V_i)$  for  $i = p-1, p-2, \dots, 1$ .
7:     Partially compute  $V_r \leftarrow V_0 \oplus t$ .
8:     Partially compute  $V_{i-1} \leftarrow \mathcal{R}^{-1}(V_i)$  for  $i = r, r-1, \dots, q+1$ .
9:     if the computed value exists in  $L$  then
10:      Set  $v_{p-1} \leftarrow (B^{\text{fix}}, B^{\text{for}}, B^{\text{back}})$ , and compute corresponding  $V_0$  and  $V_r$ .
11:      if  $V_0 \oplus V_r = T$  then
12:        return  $V_0$ .
13:      end if
14:    end if
15:  end for
16: end for

```

Attacks on 3-Round Chaskey with Feed-Forward. As shown in Figure 2, one round of π consists of two iterations of the half-round transformation. Hence, 3-round transformation of π is regarded as 6-round half-transformation of π . Let $(v_1^i, v_0^i, v_2^i, v_3^i)$ denote a 64-bit internal state which is an input to the i th half-transformation (or an output from the $i-1$ th transformation, where $i = 0, 1, \dots, 6$). We divide this transformation into two independent computations. Readers may refer to Figure 5 for the illustration of the independent computations.

Choices of $B^{\text{for}}, B^{\text{back}}, B^{\text{fix}}$. In Figure 4, the starting round is defined as an input state to some round. However, we can choose 64 bits of the state in different rounds as the starting position, as long as they fix the entire transformation. In our attack, we choose $(v_1^3, v_2^3, v_2^2, v_3^2)$ as a starting position. It is easy to see that all the possible internal state values can be simulated by exhaustively examining 2^{64} values of $(v_1^3, v_2^3, v_2^2, v_3^2)$. We then choose $B^{\text{for}}, B^{\text{back}}$, and B^{fix} as follows.

B^{for} : bit positions 0 to 4 and 13 to 15 of v_3^2 (total 8 bits)

B^{back} : bit positions 8 to 15 of v_2^3 (total 8 bits)

B^{fix} : v_1^3, v_2^2 and bit positions 5 to 12 of v_3^2 and bit positions 0 to 7 of v_2^3

The forward computation partially computes $(v_1^6, v_2^6, v_2^6, v_3^6) \oplus t$ and the backward computation partially computes $(v_1^0, v_2^0, v_2^0, v_3^0)$. We match the results of two independent computations in 8 bits.

Forward computation. All bits of v_1^3, v_2^2 and v_3^2 are known, while bit positions 8 to 15 of v_2^3 are unknown. When we compute $v_0^4 \leftarrow v_2^3 \boxplus v_3^3$, we only can compute

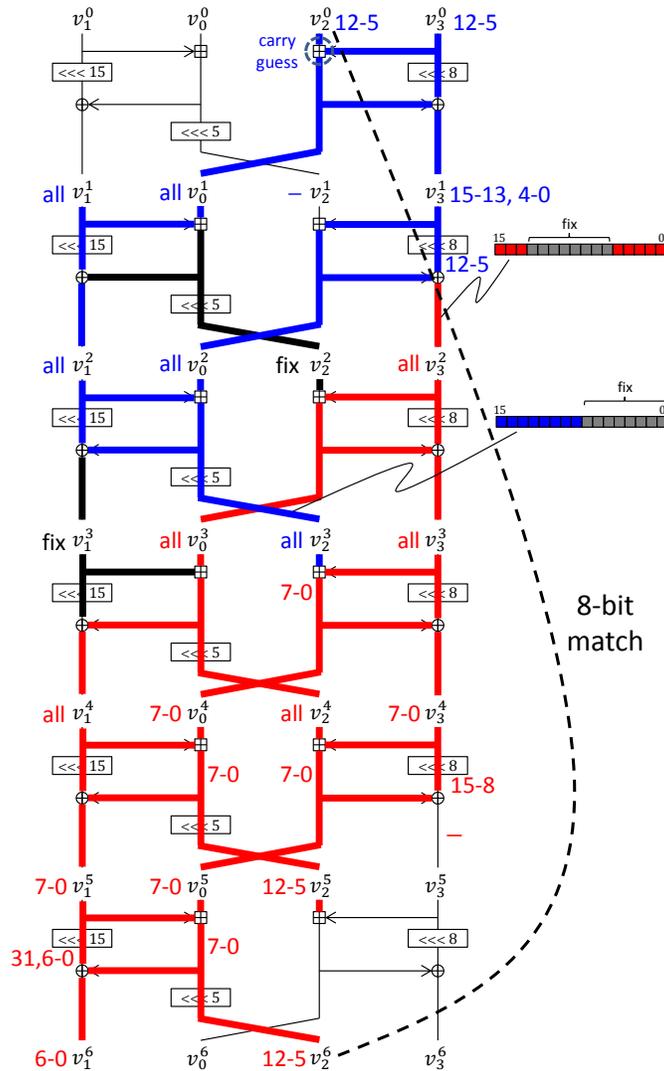


Fig. 5. Details of two independent computations for $\pi(x) \oplus x$.

the 8 LSBs of v_0^4 . Similarly, $v_3^4 \leftarrow v_0^4 \oplus (v_3^3 \lll 8)$ can be computed only in 8 LSBs. With the same analysis, as shown in Figure 5, the forward computation can compute 8 bits of v_2^5 in bit positions 5 to 12, and thus the corresponding 8 bits after xoring the tag value. Note that all partial computations of the modular addition in the forward computation are done from the LSBs, thus we do not need to consider the unknown carry effect.

Backward computation. All bits of v_1^3, v_2^2 and v_3^3 are known, while bit positions 0 to 4 and 13 to 15 of v_3^2 are unknown. All bits of v_0^1 can be computed while we can compute only 8 bits (bit positions 0 to 4 and 13 to 15) of $v_3^1 \leftarrow (v_3^2 \oplus v_0^2) \ggg 8$. This allows us to compute only 8 bits (bit positions 5 to 12) of $v_3^0 \leftarrow (v_0^1 \oplus v_3^1) \ggg 8$. Finally, we compute $v_2^0 \leftarrow v_0^1 \boxminus v_3^0$ in bit positions 5 to 12, where \boxminus is a modular subtraction. We do not know the carry from bit position 4 to 5. Hence, we guess the carry and compute v_2^0 in both cases. Thus, we have 2^9 results of the backward computation.

Complexity Evaluation. For each fixed choice of B^{fix} , we obtain 2^8 and 2^9 results from two computations. They can match in 8 bits. Correct $B^{\text{for}}, B^{\text{back}}, B^{\text{fix}}$ always match, thus we obtain $(2^8 \cdot 2^9)/2^8 = 2^9$ candidates, which needs to be tested further. The procedure is iterated for exhaustive guesses of B^{fix} . Hence, the complexity to find a preimage is $2^{48} \cdot 2^9 = 2^{57}$ computations of π . The attack requires 2^8 amount of memory to store the results of the forward computation.

Summary of Attacks and Error Probability. The attack requires 3 queries and 3 executions of the preimage attack. Hence, the data, time and memory complexities are 3, $2^{58.6} (\approx 3 \times 2^{57})$ and 2^8 , respectively.

Let us finally discuss the error probability. As discussed before, each target has e preimages on average. The MitM preimage attack exhaustively examines all internal state values (in an efficient way), hence it collects all of e preimages with 1 execution. When we sum up preimages of t_1, t_2 and t_3 , we have e^3 combinations on average. This can be regarded that the success probability of our attack is $e^{-3} \approx 0.0498$. We can also store those e^3 values as candidates, and iterate the attack for another choice of x and \tilde{y} to obtain another e^3 candidates. The correct internal state is included in both e^3 pools of candidates. In this case, the data and time complexities become 6 and $2^{59.6}$, respectively.

Discussion of the attack. MergeMAC is a provably secure MAC under the assumption that \mathcal{P}_1 and \mathcal{P}_2 are secure PRFs and the MERGE function satisfies Random Input Indistinguishability (RII). The preimage attack on 3-round Chaskey with feed-forward presented in this section shows that, using the terminology of [1], there exists a (t, q, ϵ) -RII-adversary, where $t = 3 \times 2^{57}$, $q = 2$, and ϵ is close to 1. As a consequence, as far as we see, our attack contradicts the overall security claim on MergeMAC by the designers, but it does *not* contradict the provable security claim. More precisely, the provable security claim excludes the possibility of forgery attacks whose success probability is larger than about ϵ , i.e., the attack with a high success probability itself is not excluded once ϵ turns out to be large.

4 Analysis on MergeMAC^{OW}

In this section, the attack against MergeMAC is extended to MergeMAC^{OW}, in which the MERGE function of MergeMAC (3-round-Chaskey with feed-forward) is replaced with an ideal one-way function \mathcal{H} .

The generic universal forgery discussed in Sect. 3.1 still works even if \mathcal{H} is invertible. Our approach here is to exploit the feature that \mathcal{H} is public, and thus can be evaluated offline. We preprocess \mathcal{H} for various inputs and make a look-up table so that the attacker can look-up the input x efficiently from the observed $\mathcal{H}(x)$. Namely, a precomputation phase is introduced to trade the data complexity in the online phase by the offline computational cost.

4.1 Definition of MergeMAC^{OW}

Let $\mathcal{P}_1, \mathcal{P}_2$ be two PRFs and \mathcal{H} be a public one-way function. For a given message $m \parallel \tilde{m}$, MergeMAC^{OW} computes a tag t as follows.

$$\rho \leftarrow \mathcal{P}_1(m), \quad \tilde{\rho} \leftarrow \mathcal{P}_2(\tilde{m}), \quad t \leftarrow \mathcal{H}(\rho \oplus \tilde{\rho}).$$

4.2 Tradeoff between Time and Data

As in Sect. 3, we first explain the attack by assuming that \mathcal{H} is injective.

Offline Phase

1. For 2^ℓ distinct z , compute $\mathcal{H}(z)$ and store $(z, \mathcal{H}(z))$ in a list L_p .

Online Phase

2. For distinct $x_i, 1 \leq i \leq 2^{3(n-\ell)/2}$, query $x_i \parallel \tilde{m}$ to obtain the tag t_x . If t_x is included in L_p , store x_i and the corresponding $\mathcal{P}_1(x_i) \oplus \mathcal{P}_2(\tilde{m})$ in a list L_x .
3. For distinct $\tilde{y}_j, 1 \leq j \leq 2^{3(n-\ell)/2}$, query $m \parallel \tilde{y}_j$ to obtain the tag t_y . If t_y is included in L_p , store \tilde{y}_j and the corresponding $\mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{y}_j)$ in a list L_y .
4. For all combinations of x_i and \tilde{y}_j in L_x and L_y , query $x_i \parallel \tilde{y}_j$ to obtain the corresponding tag t' until t' is included in L_p and thus the attacker obtains the value of $\mathcal{P}_1(x_i) \oplus \mathcal{P}_2(\tilde{y}_j)$.
5. Compute $\mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{m})$ by

$$(\mathcal{P}_1(x_i) \oplus \mathcal{P}_2(\tilde{m})) \oplus (\mathcal{P}_1(m_1) \oplus \mathcal{P}_2(\tilde{y}_j)) \oplus (\mathcal{P}_1(x_i) \oplus \mathcal{P}_2(\tilde{y}_j)).$$

Compute $t = \mathcal{H}(\mathcal{P}_1(m) \oplus \mathcal{P}_2(\tilde{m}))$, and output t as a tag for $m \parallel \tilde{m}$.

Evaluation and Tradeoff.

- In the offline phase (Step 1), time and memory complexities are 2^ℓ .
- In Step 2, $2^{3(n-\ell)/2}$ queries are made. Each tag is included in L_p with probability $2^{n-\ell}$, thus $2^{(n-\ell)/2}$ x_i are stored in L_x .
- In Step 3, $2^{3(n-\ell)/2}$ queries are made and $2^{(n-\ell)/2}$ \tilde{y}_j are stored in L_y .

- In Step 4, $2^{(n-\ell)}$ queries are made and there exists one pair of (x_i, \tilde{y}_j) such that t' is included in L_p .

Let T_{off} , D , and N be offline computational cost (2^ℓ), data complexity ($2^{3(n-\ell)/2}$), and a cardinality of the tag space (2^n), respectively. The tradeoff curve is represented as

$$T_{\text{off}}^{3/2} \cdot D = N^{3/2}. \quad (3)$$

Setting $T_{\text{off}} > 2^{2n/3}$ leads to $D < 2^{n/2}$, i.e. the number of online queries can be reduced compared to the generic attack in Sect. 3.1.

The attack requires to store 2^ℓ , $2^{n-\ell}$, $2^{n-\ell}$ values for L_p , L_x and L_y , respectively. When $T_{\text{off}} > 2^{2n/3}$, $M = T_{\text{off}}$. The online computational complexity, T_{on} , is only for processing queried data, thus equals to D . As long as $T_{\text{off}} > 2^{2n/3}$, T_{on} is negligible.

Example. MergeMAC supposes that $N = 2^{64}$. By spending $T_{\text{off}} = 2^{48}$ computational cost and memory amount, the number of online queries is reduced to $D = 2^{24}$ with online computational cost $T_{\text{on}} = 2^{24}$, which is more practical than the general attack with $D = 2^{32}$ queries.

Remarks. The dedicated low data complexity attack against MergeMAC in Sect. 3.2 succeeds with $T_{\text{off}} = 2^{58.6}$ and $D = 3$. This is more efficient than the generic attack against MergeMAC^{OW}. As long as it is available, using the preimage attack against \mathcal{H} is more efficient.

4.3 Reducing Memory Requirement

In this section, we reduce the memory requirement of the above attack by introducing Hellman’s time-memory tradeoff [13]. The overall idea is depicted in Figure 6. Namely, during the offline phase in order to generate the look-up table L_p , we process many inputs to make chains of values.

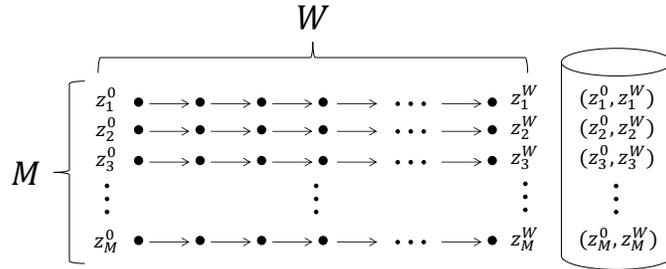


Fig. 6. Introducing Hellman’s tradeoff to reduce memory requirement.

Firstly, we choose an input value denoted by z_1^0 from the tag space $\{0,1\}^n$, and compute $z_1^i \leftarrow \mathcal{H}(z_1^{i-1})$ for $i = 1, 2, \dots, w$. We then only store the first and last values z_1^0 and z_1^w in the look-up table L_p . This procedure is iterated M times by starting from M distinct starting value $z_1^0, z_2^0, \dots, z_M^0$. During the online phase, whenever we look-up the preimage of the obtained tag t , we set $z_0 \leftarrow t$ and process $z_i \leftarrow \mathcal{H}(z_{i-1})$ for $i = 1, 2, 3, \dots$ until z_i matches one of the end points in the look-up table. This increases the online computational cost w times.

Figure 6 indicates that the tradeoff Eq. 3 is a special case of Figure 6 with $M = T$ (or $W = 1$). Remember that with the setting of Eq. 3, the online computational cost is negligible compared to the offline computational cost. The idea in Figure 6 can be interpreted that we increase the online computational cost (as high as the offline computational cost) in order to reduce the memory requirement.

We begin with the previous example, $N = 2^{64}$, $T_{\text{off}} = 2^{48}$, $T_{\text{on}} = 2^{24}$ and $D = 2^{24}$. When we process $T_{\text{off}} = 2^{48}$ values to make a look-up table, we generate 2^{24} chains in which the length of each chain is 2^{24} . Namely, we set $M = 2^{24}$ and $W = 2^{24}$ and stay with $T_{\text{off}} = 2^{48}$. Then for each of $D = 2^{24}$ queries in Steps 2 and 3 of the online phase, we need to compute the chain at most W times, which makes the entire online computational complexity $T_{\text{on}} = 2^{24} \times W = 2^{48}$. In the end, T_{off} and T_{on} are now balanced at 2^{48} , and the memory amount is reduced to 2^{24} , which is a square root of the straightforward construction of the look-up table.

We now start the general analysis. By applying the same attack procedure as in Sect. 4.2, the attack complexity becomes as follows.

$$T_{\text{off}} = MW, \tag{4}$$

$$D = (N/MW)^{3/2}, \tag{5}$$

$$T_{\text{on}} = W \cdot (N/MW)^{3/2}. \tag{6}$$

By setting $T_{\text{off}} = T_{\text{on}}$, we have

$$W = N/M^{5/3}. \tag{7}$$

Hence, the computational cost $T (= T_{\text{off}} = T_{\text{on}})$ becomes

$$T = N/M^{2/3}. \tag{8}$$

From Eq. (5) and Eq. (7), we have

$$D = M. \tag{9}$$

From Eq. (9) and Eq. (8), we have

$$T^{3/2} \cdot D = N^{3/2}. \tag{10}$$

This equation is only different from the tradeoff in the previous section Eq. (3) in terms of T_{off} and T , but the tradeoff now ensures $D = M$. Namely, by increasing T_{on} as high as T_{off} , the memory requirement is reduced to the same level as D .

5 Multiple Forgeries

In this section, we consider a problem of producing multiple forgeries by making as small number of queries as possible. This is known as the security notion called MAC reforgeability [8], where the adversary utilizes the computational complexity for the first forgery to reduce the complexity for the subsequent forgeries. This notion was also studied in the context of authenticated encryption [12].

5.1 Existential Forgery

We first consider the existential forgery, where we focus on producing as many forgeries as possible, but we do not care about the content of forged messages. From the results of Sect. 3.2, we see that given the tags of (m_1, \tilde{m}_1) , (m_1, \tilde{m}_2) , and (m_2, \tilde{m}_1) , we obtain the tag of (m_2, \tilde{m}_2) . In other words, we make 3 queries to output one forgery, and we need 3 preimage attacks. If we define the rate r as the number of queries needed to produce one forgery, i.e.,

$$r = \frac{\# \text{ queries}}{\# \text{ forgeries}},$$

then we have $r = 3$. We note that for an ideally secure MAC, if the rate to produce one forgery is r , then the rate remains the same for multiple forgeries.

Now we call it the basic attack, which can be represented by using the following matrix:

$$\begin{array}{c|cc} & \begin{matrix} j \\ 1 \ 2 \end{matrix} \\ \begin{matrix} i \\ 1 \\ 2 \end{matrix} & Q & Q \\ & Q & X \end{array}$$

The matrix shows that we make queries (m_i, \tilde{m}_j) for $(i, j) = (1, 1), (1, 2), (2, 1)$ that are shown with Q , and we obtain the forgery for $(i, j) = (2, 2)$ that is shown with X .

We show that, for $q \geq 2$, it is possible to output $(q - 1)^2$ forgeries by making $2q - 1$ queries. We first present a small example with $q = 3$. Consider the case where we make 5 queries represented by the following matrix:

$$\begin{array}{c|ccc} & \begin{matrix} j \\ 1 \ 2 \ 3 \end{matrix} \\ \begin{matrix} i \\ 1 \\ 2 \\ 3 \end{matrix} & Q & Q & Q \\ & Q & & \end{array}$$

Observe that we obtain the tag for $(i, j) = (2, 2)$ from the submatrix with $i \in \{1, 2\}$ and $j \in \{1, 2\}$, and once this is obtained, we obtain the tag for $(i, j) = (2, 3)$ from the submatrix with $i \in \{1, 2\}$ and $j \in \{2, 3\}$. At this point, we have

the following matrix:

$$\begin{array}{c|ccc}
 & & j & \\
 & & 1 & 2 & 3 \\
 \hline
 1 & Q & Q & Q \\
 i & 2 & Q & X & X \\
 3 & Q & & &
 \end{array}$$

It is easy to see that we also obtain the tags for $(i, j) = (3, 2)$ and $(3, 3)$ from the submatrix with $i \in \{2, 3\}$ and $j \in \{1, 2\}$, and then from that with $i \in \{2, 3\}$ and $j \in \{2, 3\}$. In this case, we need to make 5 queries and 5 executions of the preimage attack to produce 4 forgeries. This gives the rate $r = 5/4 = 1.25$, which is lower than the case of the basic attack.

We now generalize this to arbitrarily $q \geq 2$. We start with the following matrix:

$$\begin{array}{c|cccc}
 & & j & & \\
 & & 1 & 2 & \cdots & q \\
 \hline
 1 & Q & Q & \cdots & Q \\
 i & 2 & Q & & \\
 & \vdots & & & \\
 q & Q & & &
 \end{array}$$

For each $i = 2, 3, \dots, q$, we see that we can successively obtain the tag for (i, j) with $j = 2, 3, \dots, q$. We present the algorithmic description to show the details of this attack.

Algorithm 2 Producing $(q - 1)^2$ forgeries with $2q - 1$ queries

Require: q , the oracle \mathcal{O} that computes the tag

- 1: fix m_1, \dots, m_q and $\tilde{m}_1, \dots, \tilde{m}_q$, where m_i 's are distinct and \tilde{m}_j 's are distinct.
 - 2: for $i = 1, \dots, q$, obtain the tag of (m_i, \tilde{m}_1) by making queries to \mathcal{O} .
 - 3: for $j = 2, \dots, q$, obtain the tag of (m_1, \tilde{m}_j) by making queries to \mathcal{O} .
 - 4: **for** $i = 2, \dots, q$ **do**
 - 5: **for** $j = 2, \dots, q$ **do**
 - 6: compute the tag for (m_i, \tilde{m}_j) from the tags of $(m_{i-1}, \tilde{m}_{j-1})$, (m_{i-1}, \tilde{m}_j) , and (m_i, \tilde{m}_{j-1}) .
 - 7: **end for**
 - 8: **end for**
-

Observe that we make $2q - 1$ queries and execute $2q - 1$ preimage attacks to obtain $(q - 1)^2$ forgeries, and this gives the rate $r = (2q - 1)/(q - 1)^2$.

It is interesting to note that for $q \geq 4$, the rate becomes smaller than 1, and thus we obtain more forgeries than the number of queries. However, we remark that when q is large, the time complexity exceeds 2^{64} as the time complexity of one preimage attack is 2^{57} .

5.2 Tightness of Existential Forgery

In this section, we consider a problem of the tightness on the rate in the existential forgery. More precisely, we consider the following problem setting:

- Suppose we are given q half messages m_1, m_2, \dots, m_q and q half messages $\tilde{m}_1, \tilde{m}_2, \dots, \tilde{m}_q$.
- To obtain tags of all q^2 messages of the form (m_i, \tilde{m}_j) , where $i, j \in \{1, 2, \dots, q\}$, how many queries are necessary?

We show that $2q - 1$ queries are necessary, showing the tightness of the attack presented in the previous section.

For $i, j \in \{1, \dots, q\}$, let $t_{i,j}$ be the tag of (m_i, \tilde{m}_j) , and $s_{i,j}$ be the preimage of $t_{i,j}$, i.e., we let

$$\begin{cases} s_{i,j} \leftarrow \rho_i \oplus \tilde{\rho}_j, \\ t_{i,j} \leftarrow \pi(s_{i,j}) \oplus s_{i,j}, \end{cases}$$

where $\rho_i \leftarrow \mathcal{P}_1(m_i)$ and $\tilde{\rho}_j \leftarrow \mathcal{P}_2(\tilde{m}_j)$.

Now we observe that the relationship, $s_{i,j} \leftarrow \rho_i \oplus \tilde{\rho}_j$ for $i, j \in \{1, \dots, q\}$, can be represented by using a binary $q^2 \times 2q$ matrix \mathbf{M} as follows:

$$\mathbf{M} \cdot \boldsymbol{\rho} = \mathbf{s}, \quad (11)$$

where $\boldsymbol{\rho}$ is a column vector of length $2q$ and \mathbf{s} is a column vector of length q^2 , and they are defined as

$$\begin{cases} \boldsymbol{\rho} = [\rho_1, \dots, \rho_q, \tilde{\rho}_1, \dots, \tilde{\rho}_q]^T, \\ \mathbf{s} = [s_{1,1}, \dots, s_{1,q}, \dots, s_{q,1}, \dots, s_{q,q}]^T. \end{cases}$$

For instance when $q = 3$, Eq. (11) is

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \tilde{\rho}_1 \\ \tilde{\rho}_2 \\ \tilde{\rho}_3 \end{bmatrix} = \begin{bmatrix} s_{1,1} \\ s_{1,2} \\ s_{1,3} \\ s_{2,1} \\ s_{2,2} \\ s_{2,3} \\ s_{3,1} \\ s_{3,2} \\ s_{3,3} \end{bmatrix}.$$

Now the tightness problem is equivalent to prove the minimum number of $s_{i,j}$ that fully determines the linear system of Eq. (11) having q^2 equations and $2q$ variables. From its form, it is easy to see that the rank of the matrix \mathbf{M} is $2q - 1$. As a consequence, we need at least $2q - 1$ values of $s_{i,j}$ to determine the system, and hence we need to make at least $2q - 1$ queries.

We remark that the tightness is obtained with respect to the problem setting mentioned as above, and there are cases that are not covered. We leave the tightness of a general case as an open question.

		j					
		1	2	3	4	5	6
i	1	X					
	2		X				
	3			X			
	4				X		
	5					X	
	6						X

		j					
		1	2	3	4	5	6
i	1	X	Q				
	2	Q	X	Q			
	3		Q	X	Q		
	4			Q	X	Q	
	5				Q	X	Q
	6					Q	X

		j					
		1	2	3	4	5	6
i	1	X_2	Q	Q			
	2	Q	X_1	Q			
	3		Q	X_3	Q		
	4			Q	X_4	Q	
	5				Q	X_5	Q
	6					Q	X_6

Fig. 7. Left: Messages to be forged given as a challenge. Middle: $2q - 1$ queries we make for the attack. Right: One more query is sufficient to compute q tags for the challenge.

5.3 Universal Forgery

We next consider the universal forgery, where a list of messages to be forged is given as a challenge. Suppose that $(m_1, \tilde{m}_1), \dots, (m_q, \tilde{m}_q)$ are the challenge messages. For simplicity, we assume that m_i 's are all distinct, and \tilde{m}_i 's are all distinct.

We illustrate the case $q = 6$. Our goal is to output the tags shown with X in the left matrix given in Figure 7. For this, we make queries represented by the middle matrix given in Figure 7. At this point, we cannot obtain any of the tags of the targets. However, observe that one more appropriate query allows us to obtain the entire q tags of the targets. For instance if we make a query (m_1, \tilde{m}_3) , then we obtain the right matrix in Figure 7, and we see that it is possible to compute 6 tags with the order of X_1, \dots, X_6 .

This can be generalized to arbitrary q in an obvious way, and for completeness, we present the algorithmic description of the attack.

Algorithm 3 Producing q universal forgeries with $2q - 1$ queries

Require: $(m_1, \tilde{m}_1), \dots, (m_q, \tilde{m}_q)$, the oracle \mathcal{O} that computes the tag

- 1: for $i = 1, \dots, q - 1$, obtain the tags of (m_i, \tilde{m}_{i+1}) and (m_{i+1}, \tilde{m}_i) by making queries to \mathcal{O} .
 - 2: obtain the tag of (m_1, \tilde{m}_3) by making a query to \mathcal{O} .
 - 3: compute the tag for (m_2, \tilde{m}_2) from the tags of (m_1, \tilde{m}_2) , (m_1, \tilde{m}_3) , and (m_2, \tilde{m}_3) .
 - 4: compute the tag for (m_1, \tilde{m}_1) from the tags of (m_1, \tilde{m}_2) , (m_2, \tilde{m}_1) , and (m_2, \tilde{m}_2) .
 - 5: **for** $i = 3, \dots, q$ **do**
 - 6: compute the tag for (m_i, \tilde{m}_i) from the tags of $(m_{i-1}, \tilde{m}_{i-1})$, (m_{i-1}, \tilde{m}_i) , and (m_i, \tilde{m}_{i-1}) .
 - 7: **end for**
-

With this attack, we make $2q - 1$ queries and it uses executions of $2q - 1$ preimage attack to obtain q forgeries, which gives the rate $r = (2q - 1)/q \approx 2$.

5.4 Universal Forgery with Better Rate

We show below that it is possible to arrange the queries differently in order to improve the previous rate and obtain one that is close to $5/3$. First, we remark that 3 tags can be forged by making 5 queries (and 5 preimage attacks), as can be seen from the following matrix:

		j		
		1	2	3
1	X	Q	Q	
i 2	Q	X	Q	
3		Q	X	

Now, assume that the number of tags we want to forge is a multiple of 3, so that we are given a list of challenge messages $(m_1, \tilde{m}_1), \dots, (m_q, \tilde{m}_q)$, where $q = 3\ell$. We start by dividing the list into ℓ lists, each consisting of 3 messages as

$$\{(m_i, \tilde{m}_i), (m_{i+1}, \tilde{m}_{i+1}), (m_{i+2}, \tilde{m}_{i+2})\}_{i=1,4,7,\dots,q-2}$$

and we then treat the lists $\{(m_i, \tilde{m}_i), (m_{i+1}, \tilde{m}_{i+1}), (m_{i+2}, \tilde{m}_{i+2})\}$ individually. Each requires 5 queries and 5 preimage attacks, so to produce 3ℓ tags, we make 5ℓ queries and execute 5ℓ times the preimage attack, which gives a rate of $r = 5\ell/3\ell \approx 1.67$.

In case the number of challenges is not a multiple of 3 and is equal to $q = 3\ell + r$ with $0 < r < 3$, we proceed as before and forge each of the ℓ lists of 3 challenges with 5 queries and 5 preimage attacks. The remaining r tags can be forged by making 2 additional queries for each of them, as depicted on the following matrix:

				j			
		...	$3\ell - 2$	$3\ell - 1$	3ℓ	$3\ell + 1$	$3\ell + 2$
...	...						
$3\ell - 2$			X	Q	Q		
$3\ell - 1$			Q	X	Q		
i 3ℓ				Q	X	Q	
$3\ell + 1$					Q	X	Q
$3\ell + 2$						Q	X

We formalize this as follows. Assume we forged the first 3ℓ challenges with the previous technique. If $r = 1$, we query the tags corresponding to $(m_{3\ell+1}, \tilde{m}_{3\ell})$ together with $(m_{3\ell}, \tilde{m}_{3\ell+1})$. We combine them with the previously-forged $(m_{3\ell}, \tilde{m}_{3\ell})$ and we are able to forge $(m_{3\ell+1}, \tilde{m}_{3\ell+1})$. If $r = 2$ we also query the tags corresponding to $(m_{3\ell+2}, \tilde{m}_{3\ell+1})$ and to $(m_{3\ell+1}, \tilde{m}_{3\ell+2})$, and combine them with $(m_{3\ell+1}, \tilde{m}_{3\ell+1})$ to forge $(m_{3\ell+2}, \tilde{m}_{3\ell+2})$.

To sum up, $q = 3\ell + r$ tags (with $0 \leq r < 3$) can be forged by making $5\ell + 2r$ queries and the same number of preimage attacks, leading to a rate equal to $r = (5\ell + 2r)/(3\ell + r)$.

Note that we do not know the tightness of the rate, which is left as an open question.

6 Concluding Remarks

In this paper we presented several attacks and observations on MergeMAC. They are build around pre-image attacks on the merge functions that are possible as the merge function is public (generically) and not-one way (in the specific instance given).

We also studied the reforgeability of MergeMAC, with the result that the number of forgeries we can produce increases quadratically with the number of queries. For example, it is possible to produce roughly 2^{64} forgeries using 2^{33} forgeries and 2^{64} computation, so the cost per forgery becomes as small as legitimately computing one tag.

Finally, we like to mention interesting topics for future work. First, as stated above, we are not able to prove the tightness of the rate in the case of universal forgeries. We preformed a limited computer search for the optimal solution and were able to confirm that no solution with a better rate exist for up to 6 challenges. In our opinion, proving the optimality, or finding better strategies, is an interesting (but challenging) open question. As a second topic, generalizations of MergeMAC could be investigated, where instead of splitting the initial message into two parts, the message is split into t parts that are processed by t PRFs. The input to the merge function than becomes the xor of the t outputs of the PRFs. It would be interesting to see how our analysis could be adopted to this case.

References

1. Ankele, R., Böhl, F., Friedberger, S.: MergeMAC: A MAC for Authentication with Strict Time Constraints and Limited Bandwidth. In: Preneel, B., Vercauteren, F. (eds.) Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10892, pp. 381–399. Springer (2018)
2. Aoki, K., Sasaki, Y.: Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5381, pp. 103–119. Springer (2008)
3. Aumasson, J., Bernstein, D.J.: SipHash: A Fast Short-Input PRF. In: Galbraith, S.D., Nandi, M. (eds.) Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7668, pp. 489–508. Springer (2012), https://doi.org/10.1007/978-3-642-34931-7_28
4. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. IACR Cryptology ePrint Archive 2013, 404 (2013), <http://eprint.iacr.org/2013/404>
5. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology -

- CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016), https://doi.org/10.1007/978-3-662-53008-5_5
6. Bhargavan, K., Leurent, G.: On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 456–467 (2016)
 7. Biryukov, A., Perrin, L.: State of the Art in Lightweight Symmetric Cryptography. IACR Cryptology ePrint Archive 2017, 511 (2017), <http://eprint.iacr.org/2017/511>
 8. Black, J., Cochran, M.: MAC Reforgeability. In: Dunkelman, O. (ed.) Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5665, pp. 345–362. Springer (2009), https://doi.org/10.1007/978-3-642-03317-9_21
 9. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4727, pp. 450–466. Springer (2007), https://doi.org/10.1007/978-3-540-74735-2_31
 10. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In: Wang, X., Sako, K. (eds.) Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7658, pp. 208–225. Springer (2012), https://doi.org/10.1007/978-3-642-34961-4_14
 11. Diffie, W., Hellman, M.E.: Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *IEEE Computer* 10(6), 74–84 (1977)
 12. Forler, C., List, E., Lucks, S., Wenzel, J.: Reforgeability of Authenticated Encryption Schemes. In: Pieprzyk, J., Suriadi, S. (eds.) Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10343, pp. 19–37. Springer (2017), https://doi.org/10.1007/978-3-319-59870-3_2
 13. Hellman, M.E.: A cryptanalytic time-memory trade-off. *IEEE Trans. Information Theory* 26(4), 401–406 (1980), <https://doi.org/10.1109/TIT.1980.1056220>
 14. Jia, K., Wang, X., Yuan, Z., Xu, G.: Distinguishing and Second-Preimage Attacks on CBC-Like MACs. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) Cryptology and Network Security, 8th International Conference, CANS 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5888, pp. 349–361. Springer (2009)
 15. Luykx, A., Preneel, B., Tischhauser, E., Yasuda, K.: A MAC Mode for Lightweight Block Ciphers. In: Peyrin, T. (ed.) Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9783, pp. 43–59. Springer (2016), https://doi.org/10.1007/978-3-662-52993-5_3

16. Mouha, N., Mennink, B., Herrewewege, A.V., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In: Joux, A., Youssef, A.M. (eds.) Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8781, pp. 306–323. Springer (2014), https://doi.org/10.1007/978-3-319-13051-4_19
17. Sasaki, Y.: Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In: Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers. pp. 378–396 (2011)
18. Sasaki, Y., Aoki, K.: Finding Preimages in Full MD5 Faster Than Exhaustive Search. In: Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. pp. 134–152 (2009)
19. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Biryukov, A. (ed.) Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4593, pp. 181–195. Springer (2007), https://doi.org/10.1007/978-3-540-74619-5_12