Generic Internal State Recovery on Strengthened HMAC: n-bit Secure HMAC Requires Key in All Blocks

Sasaki, Yu; Wang, Lei

2016

Sasaki, Y., & Wang, L. (2016). Generic Internal State Recovery on Strengthened HMAC: n-bit Secure HMAC Requires Key in All Blocks. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E99.A (1), 22-30.

https://hdl.handle.net/10356/82093

https://doi.org/10.1587/transfun.E99.A.22

IEICE TRANS. FUNDAMENTALS, VOL.E99–A, NO.1 JANUARY 2016

PAPER Special Section on Cryptography and Information Security

Generic Internal State Recovery on Strengthened HMAC: *n*-bit Secure HMAC Requires Key in All Blocks*

Yu SASAKI[†a)], ***Member and*** **Lei WANG**[††b)], ***Nonmember***

SUMMARY HMAC is the most widely used hash based MAC scheme. Recently, several generic attacks have been presented against HMAC with a complexity between $2^{n/2}$ and 2^n, where n is the output size of an underlying hash function. In this paper, we investigate the security of strengthened HMAC instantiated with a Merkle-Damgård hash function in which the key is used to process underlying compression functions. With such a modification, the attacker is unable to precompute the property of the compression function offline, and thus previous generic attacks are prevented. In this paper, we show that keying the compression function in all blocks is necessary to prevent a generic internal state recovery attack with a complexity less than 2^n. In other words, only with a single keyless compression function, the internal state is recovered faster than 2^n. To validate the claim, we present a generic attack against the strengthened HMAC instantiated with a Merkle-Damgård hash function in which only one block is keyless, thus pre-computable offline. Our attack uses the previous generic attack by Naito et al. as a base. We improve it so that the attack can be applied only with a single keyless compression function while the attack complexity remains unchanged from the previous work.

key words: *HMAC, generic attack, internal state recovery, multi-collision*

1. Introduction

A message authentication code (MAC) ensures the integrity of messages transferred between two players sharing the secret key K in advance. When a sender sends a message M, he generates a tag T computed by MAC(K, M), and sends a pair of (M, T). A receiver computes the tag of the received message with his own key K and checks the match with received T. If they match, it turns out that no one modified the original message M during the communication.

A MAC can be built upon several symmetric-key primitives. Due to the usage of the key, a natural approach is constructing a MAC based on a block-cipher, i.e. constructing a mode-of-operation of the block-cipher. It is hard to show all the existing block-cipher based MACs. Two famous examples are CBC-MAC and CMAC. Another way of constructing a MAC is using a hash function. A hash function can take as input an almost arbitrary length message, thus producing a tag by hashing a combination of K and M is another natural approach. Recently, NIST announced the plan to standardize two permutation based MACs called KMAC and XMAC [2], which may be widely used in future. Each approach has its own advantages and disadvantages. All of them are important and deserve careful analysis. In this paper, we focus our attention on hash function based MACs.

A hash function based MAC often processes the key K first which takes a role of initialization, and then processes the message M, and finally processes the key K again which takes a role of finalization. This framework is called a hybrid MAC [3]. A MAC only with the initialization, prefix MAC, and a MAC only with the finalization, suffix MAC, are known to have several weaknesses. Thus, the hybrid MAC is a reasonable approach to construct a MAC.

HMAC, which was originally designed by Bellare et al. [4], is the most widely used hash function based MAC. HMAC is standardized by several standardization authorities such as NIST [5] and ISO [6]. Let $\mathcal{H}$ be an underlying hash function and `ipad` and `opad` be two constant values defined by the specification in advance. For a message M and a key K, HMAC computes a tag T by

$$T \leftarrow \mathcal{H}(K \oplus \texttt{opad} \| \mathcal{H}(K \oplus \texttt{ipad} \| M)).$$

HMAC has a nice feature that the implementation does not need to modify the underlying hash function $\mathcal{H}$. A tag can be computed only with two calls of any hash function $\mathcal{H}$. Let n be the output size of $\mathcal{H}$. The designers originally provided the security proof for the pseudo-randomness of HMAC up to $O(2^{n/2})$ queries [4]. However, it later turned out that the proof contained some mistake. Then, Bellare fixed the proof [7] and HMAC is now provably secure up to $O(2^{n/2})$ queries when a narrow-pipe Merkle-Damgård hash function, e.g. NIST's hash function standard SHA-family [8], is instantiated.

As mentioned before, HMAC can be instantiated with any hash function. However, considering the practical usage, assuming the hash function with some iterated structure such as Merkle-Damgård structure is natural. In particular, narrow-pipe Merkle-Damgård hash functions, in which the intermediate state size is the same as the last output size, are known to be efficient and many existing hash functions, e.g. SHA-2, adopt this structure. Hence, we focus our attention on the HMAC with a narrow-pipe Merkle-Damgård hash function. In the rest of this paper, we do not mention the instantiated hash function structure explicitly. Whenever the underlying hash function structure is considered, the narrow-pipe Merkle-Damgård structure is assumed. Note that, in history, security of HMAC with narrow-pipe Merkle-

Manuscript received March 23, 2015.
Manuscript revised June 5, 2015.

[†]The author is with NTT Secure Platform Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.
[††]The author is with the Nanyang Technological University, Singapore.

*A preliminary version was presented in SCN 2014 [1]. This is the full version.

a) E-mail: sasaki.yu@lab.ntt.co.jp
b) E-mail: wang.lei@ntu.edu.sg

DOI: 10.1587/transfun.E99.A.22

Damgård hash functions has been analyzed as a state-of-art of HMAC security. Namely, generic results for HMAC with narrow-pipe Merkle-Damgård hash functions have represented the generic security of HMAC. An example is the lower-bound of the security proof of HMAC (with narrow-pipe Merkle-Damgård hash function) explained in the next paragraph.

On one hand, the lower-bound of the security proof of HMAC is tight. Preneel and van Oorschot showed a generic existential forgery attack with $O(2^{n/2})$ queries [9] by exploiting the collision of the internal state generated by the birthday paradox. On the other hand, the internal state collisions cannot be utilized to break stronger security notions such as internal state recovery attack, universal forgery attack, key recovery attack, and so on. Security of HMAC against those notions was an open problem for a while.

Recently, several researches reported generic attacks on HMAC. Dodis et al. studied the indifferentiability security of HMAC in which key values are chosen by the attacker [10]. Independently, Peyrin et al. showed the similar weak key relation in the related key attack model [11], which can recover the internal state value with a complexity less than 2^n. Then in 2013, two types of generic internal state recovery attacks were proposed, one was by Leurent et al. [12] and the other was by Naito et al. [13]. Moreover in 2014, Peyrin and Wang showed the first universal forgery attack against HMAC [14]. Guo et al. further improved the complexity of the universal forgery attack and they also showed a more efficient selective forgery attack on HMAC and an application of Hellman's time-memory tradeoff in order to reduce the online complexity of the key recovery attack [15]. Dinur and Leurent showed a new approach of the universal forgery attack on HMAC that can work even if the underlying hash function adopts a HAIFA framework [16].

Considering the recent progress of the cryptanalysis against HMAC with a plain Merkle-Damgård hash function, it seems reasonable to start the research on the following two directions.

1. Analyzing HMAC instantiated with more secure mechanism than the plain Merkle-Damgård hash function.
2. Proposing an efficient countermeasures to prevent the generic attacks.

Actually, the recent work by Dinur and Leurent [16] on HMAC with HAIFA is an example of the research in the first direction. In this paper, we also step into the first direction, i.e. we investigate the security of a strengthened HMAC.

Because our work is based on the previous internal state recovery attacks on HMAC [12], [13], we explain a few more details of them. The first approach was proposed by Leurent et al. [12]. In short, it utilizes several properties of a functional graph generated by the underlying compression function with a fixed message input. In the offline phase, the attacker precomputes the behavior of the underlying function when a long message, $O(2^{n/2})$ blocks, is queried. Then, in the online phase, the attacker queries such long messages and checks if the precomputed properties are satisfied. The attack requires long queries, while the attack complexity is relatively low, $O(2^{n/2})$.

The second approach was proposed by Naito et al. [13]. It utilizes a multi-collision for the underlying compression function with a fixed message input. In the offline phase, the attacker precomputes the multi-collision of the compression function. In the online phase, the bias observed by the multi-collision is verified with higher probability than 2^{-n}. The attack requires queries of only 3 message blocks, while the attack complexity is high, $O(2^n/n)$.

Several researches showed that by using the generic internal state recovery attack as a tool, more powerful attacks can be mounted [17]–[20]. It is meaningful to strengthen the computation of HMAC so as to prevent the generic internal state recovery attack, and provide n-bit security for it.

Because the previous related-key attack [11] suggested an efficient countermeasure, in this paper we focus our attention on the generic internal state recovery attack in the single-key setting [12], [13]. The essence of those attacks is that the attacker can precompute the property of an underlying hash function or compression function, which holds irrespective of the key value. Thus, to prevent those attacks, we consider tweaking the compression function with key. For example, we fix n bits of each message block to the key. Then, the attacker becomes unable to perform the precomputation. Note that this paper is not the first one that discusses the HMAC like construction with key in all compression function invocations. An and Bellare in 1999 analyzed the NI function (Nested Iterated function) in [21], which is close to NMAC with key in all compression function invocations. Also note that a multilane variant of HMAC achieving beyond the birthday bound security was already proposed by Yasuda [22]. In this paper, we discuss how to strengthen HMAC while keeping it as a single lane, which is different from the approach in [22].

1.1 Our Contributions

We show that keying the compression function in all blocks is necessary to prevent a generic internal state recovery attack with a complexity less than 2^n. In other words, only with a single keyless compression function, the internal state is recovered faster than 2^n. We validate the claim by presenting a generic attack against the strengthened HMAC in which only one block is keyless, thus pre-computable offline. The observation can be applied for not only HMAC but also other similar type of hybrid MAC schemes such as NMAC [4] and Sandwich MAC [23]. Due to the wide usage of HMAC, we focus our attention to HMAC in this paper.

Our attack uses the previous generic attack by Naito et al. [13] as a base. We improve it so that the attack can be applied only with a single keyless compression function. The intuition of the attack in [13] is shown in Fig. 1. One message block m_{fix} is fixed and the attacker searches for input chaining values to the compression function which form a multi-collision on its output. Let $v_1, v_2, \ldots, v_t$ be input chain-

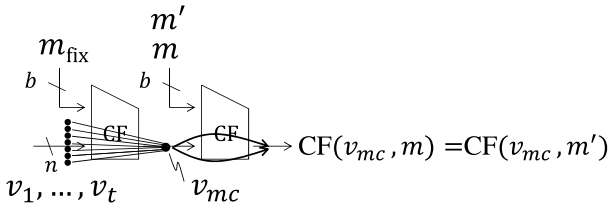


Fig. 1 Necessity of simulating two compression functions in [13].

ing values forming a t-collision[†] where $2 \le t \le n$. Also let v_{mc} be the colliding output. This indicates that as long as the message block is fixed to m_{fix}, the colliding value v_{mc} appears more frequently than others, which gives some advantage to the attacker. To detect the occurrence of v_{mc}, Naito et al. in the subsequent message block appended another colliding message pair m and m' which maps v_{mc} to an identical value. Then, if tags for two queries $x\|m_{\text{fix}}\|m$ and $x\|m_{\text{fix}}\|m'$ for any message block x collide, the internal state value is recovered to be v_{mc} with a high probability. All in all, the attacker needs to precompute at least two blocks offline.

Because our attack model assumes that only a single block is keyless, the same approach as the previous work cannot be used. In our attack, we perform the precomputation of the multi-collision for the keyless block, and change the detecting method of the occurrence of the multi-collision.

Naito et al. evaluated that their attack complexity is $O(2^n/n)$, which comes from the fact that up to n-collisions can be generated for an n-bit function with a complexity up to $O(2^n)$. In the same evaluation, our attack complexity is $O(2^n/n)$, which is the same as the previous work. The exact complexity depends on the output size n, and must be carefully evaluated. Considering the applications to SHA-256 and SHA-512, we evaluate the complexity for $n = 512$ and $n = 256$. For $n = 512$, the internal state is recovered with a complexity of $2^{508.66}$. For $n = 256$, the internal state is recovered with a complexity of $2^{253.41}$. Those lead to our claim that only with a single keyless block, the internal state is recovered with a complexity less than 2^n.

1.2 Importance of Our Work

In practice, one may not expect n-bit security by using a narrow-pipe Merkle-Damgård hash function as $\mathcal{H}$ from the following reasons.

1. The attack complexity of $O(2^n/n)$, in particular the memory usage of this size, is close to the brute force attack. Thus, the observation is not important.
2. The generic existential forgery attack exists with $O(2^{n/2})$ complexity. Thus the attack with a complexity beyond $O(2^{n/2})$ is less important.
3. The simplest way to achieve the n-bit secure HMAC is using a hash function with $2n$-bit internal state as $\mathcal{H}$ (called double-pipe hash function). Adding more modifications on the HMAC computation is less interesting than using a double-pipe hash function.

For practical usage of HMAC, those arguments are basically true. We do not claim that our observations give immediately meaningful impact to the practical usage of HMAC. However, we believe that achieving n-bit security with a narrow-pipe hash function is a long-term goal of the MAC construction, and our work contributes to make a progress in the theory of designing MACs.

We would like to note that several attacks with a complexity of $O(2^n/n)$ has been discussed in literature. Examples are the collision attack on MDC-2 [24] and the key recovery attack on the multiple-round Even-Mansour scheme [25].

1.3 Paper Outline

The organization of this paper is as follows. In Sect. 2, HMAC with a narrow-pipe Merkle-Damgård hash function is specified. Previous generic attacks on HMAC are explained in Sect. 3. Our attack which requires only a single unkeyed compression function is explained in Sect. 4. Finally, the paper is concluded in Sect. 5.

2. Specification

2.1 Merkle-Damgård Hash Functions

Merkle-Damgård structure is a common approach to process the input message of arbitrary length into the fixed hash digest. Suppose that both of the internal state size and the hash digest size are n bits[††], and also suppose that the message block size is b bits. It firstly pads the input message so that its bit size becomes a multiple of the block size. In this paper, we suppose that the padding scheme only depends on the input message size, and does not depend on the message value. Note that the MD-strengthening padding, which is widely used in the current standard hash functions such as the SHA-family [8], satisfies this property. The padded message is then split into message blocks $M_0, M_1, \ldots, M_{L-1}$, where the size of each M_i is b bits. Then the compression function $\text{CF} : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ is iteratively applied as follows:

$$H_0 \leftarrow \text{IV},$$
$$H_i \leftarrow \text{CF}(H_{i-1}, M_{i-1}) \qquad \text{for } i = 1, 2, \ldots, L,$$

where IV stands for initial value, which is an n-bit constant defined in the specification. H_L is the hash digest of the input message.

[†] t-collision means t different input values which return an identical output value.

[††] If the internal state size is bigger than the hash digest size, the construction is called wide-pipe [26]. In this paper, we focus our attention on only narrow-pipe, where the internal state size and the hash digest size are identical.

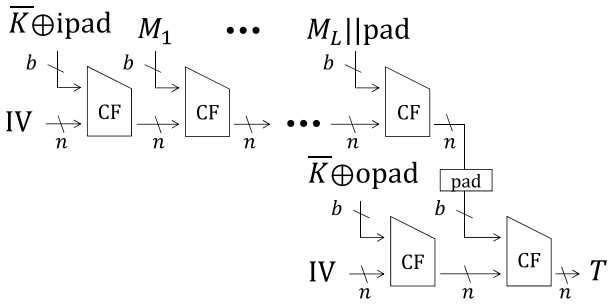


Fig. 2 Block-wise computation structure of HMAC.

2.2 HMAC

HMAC with a secret key K produces a MAC tag only with two calls of hash function $\mathcal{H}$. At first, the padding is applied to K in order to make its size b bits.

- If $|K| < b$, append a necessary number of zeros, i.e. $\overline{K} \leftarrow K||00\cdots0$.
- If $|K| = b$, nothing is processed, i.e. $\overline{K} \leftarrow K$.
- If $|K| > b$, first compress K with an underlying hash function $\mathcal{H}$ i.e. $\overline{K} \leftarrow \mathcal{H}(K)$.

Then, the MAC tag is computed as follows.

$$\mathrm{HMAC}_K(M) \leftarrow \mathcal{H}(\overline{K} \oplus \mathtt{opad}||\mathcal{H}(\overline{K} \oplus \mathtt{ipad}||M)),$$

where `ipad` and `opad` are b-bit constant values specified in HMAC.

$$\mathtt{ipad} \triangleq \mathtt{0x363636}\cdots\mathtt{36},$$
$$\mathtt{opad} \triangleq \mathtt{0x5c5c5c}\cdots\mathtt{5c},$$

in which '`0x`' represents hexadecimal numbers. The block-wise representation of the HMAC construction is depicted in Fig. 2. For clarity, the pseudo-code of the HMAC algorithm is given in Algorithm 1.

Algorithm 1 HMAC Algorithm

Input: a hash function $\mathcal{H}$, key K, message M
Output: tag T
1: $\overline{K} \leftarrow$ Padding(K)
2: $\mathtt{tmp1} \leftarrow \mathcal{H}((\overline{K} \oplus \mathtt{ipad})||M)$
3: $\mathtt{tmp2} \leftarrow \mathcal{H}((\overline{K} \oplus \mathtt{opad})||\mathtt{tmp1})$
4: $T \leftarrow \mathtt{tmp2}$
5: **return** T

HMAC was firstly proven to be a secure MAC up to $O(2^{n/2})$ queries under the assumption that $\mathcal{H}$ is collision resistant [4]. It was then proven to be a pseudo-random function (PRF) up to $O(2^{n/2})$ queries under the assumption that $\mathcal{H}$ is weak collision resistant [7]. Finally, the proof of the exact PRF of HMAC is given by [27]. For more queries than the birthday bound, the proof of HMAC cannot ensure its security. Indeed, several generic attacks have already been proposed, which will be explained in Sect. 3.

3. Previous Work

Our work is based on the previous internal state recovery attacks on HMAC [12], [13]. This section explains more details of these attacks.

3.1 Long Message Attack Exploiting Functional Graph

Leurent et al. proposed an efficient internal state recovery attack when an attacker can query long messages, $O(2^{n/2})$ blocks [12]. The attack utilizes several properties of a functional graph generated by the underlying hash function $\mathcal{H}$. In the narrow-pipe Merkle-Damgård hashes, $\mathcal{H}$ iterates the compression function CF. The attacker first fixes the b-bit message in the compression function CF in all blocks, which is denoted by M. Then, the function is represented as $\mathrm{CF}_M : \{0,1\}^n \to \{0,1\}^n$. Because CF_M is a public function, the attacker can simulate the properties of CF_M offline.

In a functional graph, the chain of the function is computed. Namely, a start point v_0 is determined randomly, then $v_i \leftarrow \mathrm{CF}_M(v_{i-1})$ is computed for $i = 1, 2, 3, \cdots$ until v_i reaches the previously computed value. A functional graph satisfies several generic properties, *e.g.*, each point enters a cycle after $O(2^{n/2})$ iterations, and the size of the largest cycle of the functional graph is $O(2^{n/2})$.

The attack by Leurent et al. [12] works as follows.

Offline The attacker computes the size of the largest cycle of CF_M, which is denoted by L.
Online The attacker queries the following two messages of the size $O(2^{n/2})$ blocks:

$$M^{[2^{n/2}]}||M_{\mathrm{rand}}||M^{[2^{n/2}+L]},$$
$$M^{[2^{n/2}+L]}||M_{\mathrm{rand}}||M^{[2^{n/2}]},$$

where $M^{[x]}$ represents the x iterations of the message block M and M_{rand} represents a randomly chosen message block such that $M_{\mathrm{rand}} \neq M$. For HMAC with $\mathcal{H}$, the above two queries return the same tag with a high probability, and this gives significant information to the attacker. Leurent et al. pointed out that the internal state value can be recovered [12] and Peyrin and Wang [19] showed a generic universal forgery attack by combining more properties of the functional graph.

3.2 Short Message Attack Exploiting Multi-Collisions

Naito et al. proposed another internal state recovery attack [13]. The advantage of this attack is that it only requires queries of 3 message blocks, while the complexity of the attack is high, which is some value between 2^n and $2^n/n$. In other words, the attack efficiency is lower-bounded by $2^n/n$.

The attack fixes a b-bit message input M to the compression function to form a random function $\mathrm{CF}_M : \{0,1\}^n \to \{0,1\}^n$. The attack uses the multi-collision of CF_M which is generated by the complexity between 2^n and $2^n/n$.

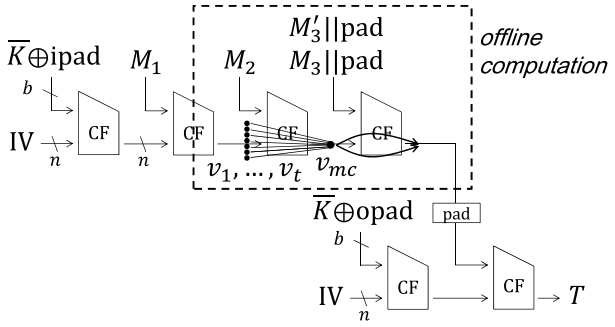


Fig. 3 Previous internal state recovery attack on HMAC with short message input.

A t-collision is expected to be found with about $2^{(t-1)n/t}$ computations of CF_M. Thus, with a complexity up to 2^n, up to n-collision is expected. Note that this is a rough evaluation, and the exact complexity that optimizes the entire attack must be carefully calculated as was done in [13]. Let v_{mc} be the output value of the t-collision.

Naito *et al.* pointed out that v_{mc} appears more frequently than other values due to the bias confirmed by the t-collision, and this fact can be used to recover the internal state value. The entire attack procedure is described in Algorithm 2, which is also depicted in Fig. 3. In the online phase, one of j choices of M_1 will reach one of $v_1, \ldots, v_t$, and thus the collision is generated after $M_3||\mathsf{pad}$ and $M'_3||\mathsf{pad}$. Then, the internal state value v_{mc} is recovered. The attack complexity depends on the choice of r and the corresponding j, which are between 2^n and $2^n/n$.

Algorithm 2 Internal State Recovery on HMAC with Short Messages

Input: Attack parameter r, where $2^n/n \le r \le 2^n$
Output: Internal state value v_{mc} after a particular two message blocks
Offline Phase
1: Fix the value of M_2.
2: **for** $j \leftarrow 1, 2, \ldots, r$ **do**
3: Choose distinct values of the chaining variable input v_r, and then compute $\mathrm{CF}_{M_2}(v_r)$. The computed results are stored.
4: **end for**
5: Find a t-collision of $\mathrm{CF}_{M_2}(v_r)$ where $2 \le t \le n$. For simplicity, let $v_1, v_2, \ldots, v_t$ be the chaining variable inputs forming a t-collision for CF_{M_2}.
6: For the later detection of the occurrence of v_{mc}, find a colliding pair $M_3||\mathsf{pad}$ and $M'_3||\mathsf{pad}$ such that $\mathrm{CF}(v_{mc}, M_3||\mathsf{pad}) = \mathrm{CF}(v_{mc}, M'_3||\mathsf{pad})$, where pad is the padding string for the online phase.
Online Phase
7: **for** $j \leftarrow 1, 2, \ldots, 2^n/t$ **do**
8: Choose distinct values of the message input M_1^j
9: Query $M_1^j||M_2||M_3$.
10: Query $M_1^j||M_2||M'_3$.
11: **if** Tags for those two queries collide **then**
12: **return** $M_1^j||M_2$ and the corresponding internal state v_{mc}.
13: **end if**
14: **end for**

3.3 Preventing Internal State Recovery Attacks

Let us consider possible countermeasures against the previous internal state recovery attacks by modifying the computation of HMAC.

The essence of the previous attack is that the behavior of the compression function with a fixed message CF_M can be simulated by the attacker offline. Thus, by tweaking the compression function with a secret value derived from K the attacks are invalidated. On the contrary, the number of such a tweak should be minimized so that the bad influence to the computation efficiency is minimized.

The long message attack [12] requires the precomputation of a big functional graph. Hence, giving a key tweak in every a few blocks can prevent the attack. The short message attack [12] requires the precomputation of two consecutive compression functions. Hence, the attack can be prevented by giving a key tweak in every two blocks.

4. New Attack against Single Keyless Block

In this section, we show that if there exists one compression function that can be simulated by the attacker offline, the internal state recovery attack can be performed with a complexity less than 2^n. An important implication of this fact is that to ensure n-bit security against the internal state recovery attack, the compression function must depend on the key in all blocks so that the attacker cannot perform the precomputation.

4.1 Target Structure

We consider a strengthened HMAC as our attack target. That is to say, all compression function computations but for one block (for processing the i-th message block) are tweaked by key value and the tweak depends on the block position. The tweak, for example, can be given by setting n bits of a b-bit message block to a value derived by the key. The modification is conservative but for the single keyless block, which will clearly indicate how the single keyless block gives a bad influence to the entire structure. The computation structure of our attack target is depicted in Fig. 4.

4.2 Attack Overview

Due to the key in most of blocks, we follow the approach of the short message attack [13] that exploits the multi-collision of the compression function.

The offline phase with the dominant complexity part is the same as the one in [13]. Namely, we fix the message input M_i and choose r distinct chaining variable inputs $v_1, v_2, \ldots, v_r$ to find a t-collision of the fixed function CF_{M_i}, where r is a some value between $2^n/n$ and 2^n. Let $v_1, v_2, \ldots, v_t$ be the obtained chaining variable inputs forming a t-collision.

Then in the online phase, we generate two messages

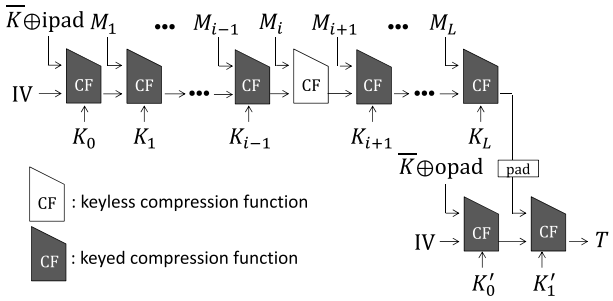


Fig. 4 Attack target only with a single keyless block.

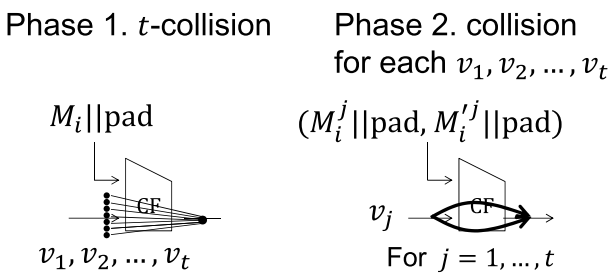


Fig. 5 Offline phase.

$M_1||M_2||\cdots||M_{i-1}$ which will reach one of $v_1, v_2, \ldots, v_t$. If we can detect such messages, the internal state value can take only t choices, and it is possible to recover the exact internal state value by exhaustively testing all the t choices. To generate such two messages, we fix the value of $M_1||M_2||\cdots||M_{i-2}$, choose $2 \times 2^n/t$ distinct messages of M_{i-1} and query $M_1||M_2||\cdots||M_{i-2}||M_{i-1}||M_i$ for each choice of M_{i-1}. Two messages reaching any of $v_1, v_2, \ldots, v_t$ collide after the i-th block, and thus produce an identical tag. Hence, we pick all message pairs with colliding tags. Of course this will generate a lot of noise, i.e. collisions of tags without reaching $v_1, v_2, \ldots, v_t$. We filter out those noise by using additional queries.

4.3 Offline Phase

The offline phase consists of two sub-phases which are depicted in Fig. 5.

Phase 1 is for constructing a t-collision of the keyless compression function CF. We fix the message input $M_i||\mathsf{pad}$ where pad is the padding string for the online phase. As explained later, the length of queries in the online phase is i blocks plus the size of M_i. M_i can be set to any value as long as $M_i||\mathsf{pad}$ fits in one block. Then, the compression function can be described as an n-bit to n-bit function $\mathrm{CF}_{M_i}(\cdot)$. We then choose r distinct chaining variable inputs $v_1, v_2, \ldots, v_r$, and compute $\mathrm{CF}_{M_i}(v_1), \mathrm{CF}_{M_i}(v_2), \ldots, \mathrm{CF}_{M_i}(v_r)$ to find a t-collision of the function's output, where r is a some value between $2^n/n$ and 2^n determined later. Let $v_1, v_2, \ldots, v_t$ be the obtained chaining variable inputs forming the t-collision.

Phase 2 is for generating a message pair $(M_i^j, M_i'^j)$ such that $\mathrm{CF}(v_j, M_i^j) = \mathrm{CF}(v_j, M_i'^j)$ for $1 \le j \le t$ for the later detection of the occurrence of $v_1, v_2, \ldots, v_t$. The idea behind is that if we later find a message $M_1||\cdots||M_{i-1}$ which may reach one of $v_1, v_2, \ldots, v_r$, we can confirm it by checking the tag collision of the message pair $M_1||\cdots||M_{i-1}||M_i^j$ and $M_1||\cdots||M_{i-1}||M_i'^j$.

The complexity of the offline phase is as follows. In phase 1, the computational cost is r compression function computations and the memory requirement is to store the r corresponding tags, where $2^n/n \le r \le 2^n$. The computational cost of phase 2 is $t \times 2^{n/2}$ compression function computations, which is negligible compared to phase 1. The collision search of phase 2 can be memoryless. In the end, the complexity of the offline phase is r compression function computations and a memory to store r n-bit tags. Because this is offline, the number of query is 0.

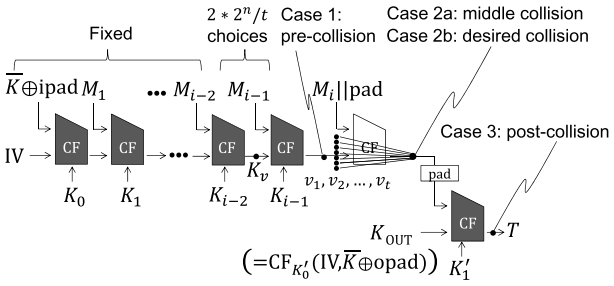


Fig. 6 Online phase.

4.4 Online Phase

The online phase consists of three sub-phases called 1) *collision collection*, 2) *noise elimination*, and 3) *recovering value*. The online phase is depicted in Fig. 6. Note that the first message block of the outer function, $\overline{K} \oplus \mathsf{opad}$, is not related to our attack. We omit its description in Fig. 6 by representing the first block's output as K_{OUT}.

4.4.1 Collision Collection

As explained in Sect. 4.2, we generate two messages which will reach one of $v_1, v_2, \ldots, v_t$ and collect all message pairs producing the tag collision. Because a t-collision is generated offline, we query $2 \times 2^n/t$ messages in the collision collection phase. We first fix the value of $M_1||M_2||\cdots||M_{i-2}$. Then for any query, the internal state value after processing M_{i-2} becomes a fixed value, denoted by K_v in Fig. 6, and it behaves as an equivalent key. We then choose $2 \times 2^n/t$ distinct messages $M_{i-1}^1, M_{i-1}^2, \ldots, M_{i-1}^{2^n/t}$ so that we can obtain two messages reaching one of $v_1, v_2, \ldots, v_t$. We query $M_1||M_2||\cdots||M_{i-2}||M_{i-1}^j||M_i$ for $j = 1, 2, \ldots, 2^n/t$. For those queries, HMAC pads the padding string pad and the last message block becomes exactly the same as the one fixed in the offline phase. Two messages reaching any of $v_1, v_2, \ldots, v_t$ collide after the last message block, and thus produce the tag collision. Here, we also obtain many collisions that do not reach $v_1, v_2, \ldots, v_t$. Those collisions are called noise. There are 3 cases that noise occurs: after processing M_{i-1} which are called pre-collisions, after pro-

cessing $M_i \| \mathtt{pad}$ but without reaching $v_1, v_2, \ldots, v_t$ which are called middle-collisions, and after processing the outer function which are called post-collisions. Let us count the number of noise. With $2 \times 2^n/t$ queries, about $2^{2n+1}/t^2$ pairs are generated and each pair produces an identical tag value with probability 2^{-n}. In the end, we obtain the following collisions.

Case 1. $2^{n+1}/t^2$ pre-collisions
Case 2a. $2^{n+1}/t^2$ middle-collisions
Case 2b. 1 desired collision via some of $v_1, \ldots, v_t$
Case 3. $2^{n+1}/t^2$ post-collisions

In total, we obtain $3 * 2^{n+1}/t^2 + 1$ collisions.

4.4.2 Noise Elimination

In noise elimination, we discard all the pre-collisions and post-collisions with two more additional queries for each noise.

To eliminate pre-collisions, we query each colliding message pair by replacing M_i with another one M_i^*. Because the internal state value is already collided after processing M_{i-1}, the pair also generates a tag collision with new M_i^*. We discard all the pairs that generate another collision with new M_i^*. The probability of wrongly discarding the desired collision is 2^{-n}, which is negligible. Note that wrongly discarding other noise does not cause any problem.

To eliminate post-collisions, we query each colliding message pair by appending the padding string pad and then appending a common message block M_{i+1}. Because input value to the outer function changes, new pairs do not produce a collision with an overwhelming probability. We discard all the pairs that do not produce another collision. The probability of wrongly discarding the desired collision is 0 because the collision after processing $M_i \| \mathtt{pad}$ is preserved even after the additional common message block. Note that a post-collision wrongly remain with probability of $2^{-n} \times 2^{n+1}/t^2 = 2/t^2$. Such a small number of post-collisions only give negligible impact to the next phase. Moreover, by iterating the noise eliminating phase twice, even such a negligible event can be avoided.

The remaining collisions are $2^{n+1}/t^2$ middle-collisions and 1 desired collision. Unfortunately, eliminating middle-collisions is hard. However, we can still recover the internal state by analyzing $2^{n+1}/t^2$ middle-collisions and 1 desired collision together.

4.4.3 Recovering Value

For each of $2^{n+1}/t^2 + 1$ collision pairs, we confirm that one message of the pair reaches some of $v_1, v_2, \ldots, v_t$. Thus, we replace M_i with M_i^j, and then with $M_i'^j$ for $j = 1, 2, \ldots, t$ and check if the corresponding tags collide. The desired collision always generates collision with M_i and M_i^j. Moreover, we can iterate the same test for the other message of the pair. Hence, the probability of the false positive is negligible. In the end, the correct internal state value is recovered.

4.4.4 Complexity Evaluation

The complexity of the collision collecting phase is $2 \times 2^n/t$ queries, $2 \times 2^n/t$ memory accesses to deal with the obtained tags and a memory to store $2 \times 2^n/t$ tags. The complexity of the noise elimination phase is $2 * 3 * 2^{n+1}/t^2 + 1$ queries for eliminating pre-collisions and $2 * 2 * 2^{n+1}/t^2 + 1$ queries for eliminating post-collisions, which is about $20 * 2^n/t^2$ queries in total. The computational cost and memory requirement are negligible. The complexity of the recovering value phase is $2t * (2^{n+1}/t^2 + 1) \approx 4 * 2^n/t$ queries. The computational cost and memory requirement are negligible.

All in all, the total attack complexity of the online phase is as follows.

- The number of queries is $2 * 2^n/t + 20 * 2^n/t^2 + 4 * 2^n/t = 6 * 2^n/t + 20 * 2^n/t^2$.
- The computational cost is $2 * 2^n/t$.
- The memory requirement is $2 * 2^n/t$.

4.5 Choosing Optimal Parameters

The bottleneck of the offline phase is finding a t-collision with r computations of CF. Suzuki *et al.* showed the complexity to find a t-collision with probability 1/2 [28], [29], which tells the exact value of r in our attack.

$$r = (t!)^{1/t} \times (2^{n \cdot \frac{t-1}{t}}) + t - 1. \quad (1)$$

The attack complexity of the online phase is $6 * 2^n/t + 20 * 2^n/t^2$ queries and $2 * 2^n/t$ computational cost and memory requirement, in which the number of queries is dominant. The data complexity is evaluated by the number of queried message blocks. Hence, the data complexity is

$$i \times (6 * 2^n/t + 20 * 2^n/t^2) \quad (2)$$

The attack does not specify the exact block position of the keyless compression function i. For security of the construction should be evaluated in the worst case scenario, in which the attacker has a control of the choice of i. Then, we assume that $i = 2$, and choose the value of t to make a balance between Eq. (1) and Eq. (2). The best choice of t depends on the tag size n. Considering the application to SHA-256 and SHA-512, the evaluation of t for $n = 512$ and $n = 256$ are summarized in Table 1.

4.6 Attack Procedure

We finally describe the attack procedure in Algorithm 3.

Table 1 Attack complexity with optimal parameter choices.

n	t	Offline Eq.(1)	Online Eq.(2)
512	64	$2^{508.625}$	$2^{508.658}$
256	39	$2^{253.380}$	$2^{253.418}$

The exact choices of t can be found in Table 1, and the corresponding r can be calculated by Eq. (1). As shown in Table 1, the internal state value can be recovered with a complexity less than 2^n.

Algorithm 3 Internal State Recovery on HMAC with a Single Keyless Block

Output: Two pairs of $i-1$ message blocks $M_1\|\cdots\|M_{i-1}$ and the internal state after the last block

Offline Phase

Fix M_i such that $M_i\|\mathsf{pad}$ fits in one block.
for $j \leftarrow 1, 2, \ldots, r$ **do**
  Choose the value of v_j, then compute $\mathrm{CF}_{M_i}(v_j)$ and store the result.
end for
Find a t-collision for $\mathrm{CF}_{M_i}(\cdot)$. Let t values forming a t-collision be $v_1, v_2, \ldots, v_t$.
for $j \leftarrow 1, 2, \ldots, t$ **do**
  Find $(M_i^j, M_i'^j)$ such that $\mathrm{CF}(v_j, M_i^j) = \mathrm{CF}(v_j, M_i'^j)$.
end for

Online Phase

Fix $M_1, M_2, \ldots, M_{i-2}$.
for $j \leftarrow 1, 2, \ldots, 2^n/t$ **do**
  Choose the value of M_{i-1}^j, then query $M_1\|\cdots\|M_{i-2}\|M_{i-1}^j\|M_i$ and store the tag.
end for
Pick up message pairs producing the tag collision.
Let $(\overline{M_z}, \overline{M_z'})$ for $z = 1, 2, \ldots, 3 * 2^n/t^2 + 1$ be pairs of $M_1\|\cdots\|M_{i-2}\|M_{i-1}$ producing the tag collision.
for $z = 1, 2, \ldots, 3 * 2^n/t^2 + 1$ **do**
  Choose a new value of M_i^*, then query $\overline{M_z}\|M_i^*$ and $\overline{M_z'}\|M_i^*$.
  if tags collide **then**
    Discard the pair.
  end if
end for
for $z = 1, 2, \ldots, 2 * 2^n/t^2 + 1$ **do**
  Choose a new value of M_{i+1}, then query $\overline{M_z}\|M_i\|\mathsf{pad}\|M_{i+1}$ and $\overline{M_z'}\|M_i\|\mathsf{pad}\|M_{i+1}$.
  if tags do not collide **then**
    Discard the pair.
  end if
end for
for $z = 1, 2, \ldots, 2^n/t^2 + 1$ **do**
  for $j = 1, 2, \ldots, t$ **do**
    Query $\overline{M_z}\|M_i^j$ and $\overline{M_z}\|M_i'^j$.
    if tags collide **then**
      for $k = 1, 2, \ldots, t$ **do**
        Query $\overline{M_z'}\|M_i^k$ and $\overline{M_z'}\|M_i'^k$.
        if tags collide **then**
          return $(\overline{M_z}, v_j)$ and $(\overline{M_z'}, v_k)$.
        end if
      end for
    end if
  end for
end for

5. Concluding Remarks

In this paper, we investigated the security of strengthened HMAC in which the underlying compression function is keyed to avoid the precomputation by the attacker. We improved the previous generic internal state recovery attack by Naito et al. so that it can be applied only with a single keyless compression function without increasing the attack complexity. This fact indicates that in order to avoid the generic internal state recovery attack faster than 2^n complexity, the compression function must be keyed in all blocks.

In this paper, we aimed to identify the necessary condition to achieve n-bit security in HMAC with a narrow-pipe Merkle-Damgård hash function. A possible future research direction is identifying sufficient conditions to achieve n-bit security by modifying HMAC.

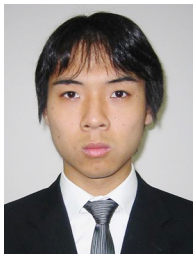
Acknowledgments

Lei Wang is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

[1] Y. Sasaki and L. Wang, "Generic attacks on strengthened HMAC: n-bit secure HMAC requires key in all blocks," Security and Cryptography for Networks, Lecture Notes in Computer Science, vol.8642, pp.324–339, Springer, 2014.
[2] R. Perlner, "Special publication on KMAC." Presented in NIST SHA3 workshop, 2014. http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/
[3] G. Tsudik, "Message authentication with one-way hash functions," Proc. IEEE INFOCOM'92: The Conference on Computer Communications, vol.3, pp.2055–2059, 1992.
[4] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," Advances in Cryptology — CRYPTO'96, Lecture Notes in Computer Science, vol.1109, pp.1–15, Springer, 1996.
[5] U.S. Department of Commerce, National Institute of Standards and Technology, The Keyed-Hash Message Authentication Code (HMAC) (Federal Information Processing Standards Publication 198), July 2008. http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
[6] ISO/IEC 9797-2:2011, Information technology — Security techniques — Message Authentication Codes (MACs) — Part 2, 2011.
[7] M. Bellare, "New proofs for NMAC and HMAC: Security without collision-resistance," Advances in Cryptology — CRYPTO 2006, Lecture Notes in Computer Science, vol.4117, pp.602–619, Springer, 2006.
[8] U.S. Department of Commerce, National Institute of Standards and Technology, Secure Hash Standard (SHS) (Federal Information Processing Standards Publication 180-3), 2008. http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf
[9] B. Preneel and P.C. van Oorschot, "On the security of two MAC algorithms," Advances in Cryptology — EUROCRYPT'96, Lecture Notes in Computer Science, vol.1070, pp.19–32, Springer, 1996.
[10] Y. Dodis, T. Ristenpart, J. Steinberger, and S. Tessaro, "To hash or not to hash again? (In)Differentiability results for H^2 and HMAC," Advances in Cryptology — CRYPTO 2012, Lecture Notes in Computer Science, vol.7417, pp.348–366, Springer, 2012.
[11] T. Peyrin, Y. Sasaki, and L. Wang, "Generic related-key attacks for HMAC," Advances in Cryptology — ASIACRYPT 2012, Lecture Notes in Computer Science, vol.7658, pp.580–597, Springer, 2012.
[12] G. Leurent, T. Peyrin, and L. Wang, "New generic attacks against hash-based MACs," Advances in Cryptology — ASIACRYPT 2013, Lecture Notes in Computer Science, vol.8270, pp.1–20, Springer, 2013.
[13] Y. Naito, Y. Sasaki, L. Wang, and K. Yasuda, "Generic state-recovery and forgery attacks on ChopMD-MAC and on NMAC/HMAC," Advances in Information and Computer Security, Lecture Notes in Computer Science, vol.8231, pp.83–98, Springer, 2013.

[14] T. Peyrin and L. Wang, "Generic universal forgery attack on iterative hash-based MACs," Advances in Cryptology — EUROCRYPT 2014, Lecture Notes in Computer Science, vol.8441, pp.147–164, Springer, 2014.
[15] J. Guo, T. Peyrin, Y. Sasaki, and L. Wang, "Updates on generic attacks against HMAC and NMAC," Advances in Cryptology — CRYPTO 2014, Lecture Notes in Computer Science, vol.8616, pp.131–148, Springer, 2014.
[16] I. Dinur and G. Leurent, "Improved generic attacks against hash-based MACs and HAIFA," Advances in Cryptology — CRYPTO 2014, Lecture Notes in Computer Science, vol.8616, pp.149–168, Springer, 2014.
[17] J. Guo, Y. Sasaki, L. Wang, M. Wang, and L. Wen, "Equivalent key recovery attacks against HMAC and NMAC with whirlpool reduced to 7 rounds," Fast Software Encryption, Lecture Notes in Computer Science, vol.8540, pp.571–590, Springer, 2015.
[18] J. Guo, Y. Sasaki, L. Wang, and S. Wu, "Cryptanalysis of HMAC/NMAC-Whirlpool," Advances in Cryptology — ASIACRYPT 2013, Lecture Notes in Computer Science, vol.8270, pp.21–40, Springer, 2013.
[19] T. Peyrin and L. Wang, "Generic universal forgery attack on iterative hash-based MACs," Advances in Cryptology — EUROCRYPT 2014, Lecture Notes in Computer Science, vol.8441, pp.147–164, Springer, 2014.
[20] Y. Sasaki and L. Wang, "Improved single-key distinguisher on HMAC-MD5 and key recovery attacks on sandwich-MAC-MD5," Selected Areas in Cryptography — SAC 2013, Lecture Notes in Computer Science, vol.8282, pp.493–512, Springer, 2014.
[21] J.H. An and M. Bellare, "Constructing VIL-MACs from FIL-MACs: Message authentication under weakened assumptions," Advances in Cryptology — CRYPTO'99, Lecture Notes in Computer Science, vol.1666, pp.252–269, Springer, 1999.
[22] K. Yasuda, "Multilane HMAC — Security beyond the birthday limit," INDOCRYPT, ed. K. Srinathan, C.P. Rangan, and M. Yung, Lecture Notes in Computer Science, vol.4859, pp.18–32, Springer, 2007.
[23] K. Yasuda, "'Sandwich" is indeed secure: How to authenticate a message with just one hashing," in ACISP, ed. J. Pieprzyk, H. Ghodosi, and E. Dawson, Lecture Notes in Computer Science, vol.4586, pp.355–369, Springer, 2007.
[24] L.R. Knudsen, F. Mendel, C. Rechberger, and S.S. Thomsen, "Cryptanalysis of MDC-2," Advances in Cryptology — EUROCRYPT 2009, Lecture Notes in Computer Science, vol.5479, pp.106–120, Springer, 2009.
[25] I. Dinur, O. Dunkelman, N. Keller, and A. Shamir, "Key recovery attacks on 3-round even-mansour, 8-step LED-128, and full AES2," Advances in Cryptology — ASIACRYPT 2013, Lecture Notes in Computer Science, vol.8269, pp.337–356, Springer, 2013.
[26] S. Lucks, "A failure-friendly design principle for hash functions," Advances in Cryptology — ASIACRYPT 2005, Lecture Notes in Computer Science, vol.3788, pp.474–494, Springer, 2005.
[27] P. Gaži, K. Pietrzak, and M. Rybár, "The exact PRF-security of NMAC and HMAC," Advances in Cryptology — CRYPTO 2014, Lecture Notes in Computer Science, vol.8616, pp.113–130, Springer, 2014.
[28] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota, "Birthday paradox for multi-collisions," Information Security and Cryptology — ICISC 2006, Lecture Notes in Computer Science, vol.4296, pp.29–40, Springer, 2006.
[29] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota, "Birthday paradox for multi-collisions," IEICE Trans. Fundamentals, vol.E91-A, no.1, pp.39–45, Jan. 2008.



Yu Sasaki received the B.E., M.E. and Ph.D. from The University of Electro-Communications in 2005, 2007, and 2010. Since 2007, he has been a researcher at NTT Secure Platform Laboratories. His current research interests are in cryptography. He was awarded a paper prize from SCIS 2007 and IEICE Trans. in 2009. He also received a best paper award from IWSEC 2009, SECRYPT 2012, and IWSEC 2012.

Lei Wang received the M.E. and Ph.D. from The University of Electro-Communications in 2009, and 2011, respectively. His current research interests are in cryptography. He was awarded a paper prize from SCIS 2008 and IEICE Trans. in 2009. He also received a best paper award from IWSEC 2009 and IWSEC 2012.