Improved Single-Key Distinguisher on HMAC-MD5 and Key Recovery Attacks on Sandwich-MAC-MD5

Yu Sasaki[1(✉)] and Lei Wang[2]

[1] NTT Secure Platform Laboratories, Tokyo, Japan
sasaki.yu@lab.ntt.co.jp
[2] Nanyang Technological University, Singapore, Singapore
Wang.Lei@ntu.edu.sg

Abstract. This paper presents key recovery attacks on Sandwich-MAC instantiating MD5, where Sandwich-MAC is an improved variant of HMAC and achieves the same provable security level and better performance especially for short messages. The increased interest in lightweight cryptography motivates us to analyze such a MAC scheme. We first improve a distinguishing-H attack on HMAC-MD5 proposed by Wang *et al.* We then propose key recovery attacks on Sandwich-MAC-MD5 by combining various techniques such as distinguishing-H for HMAC-MD5, IV Bridge for APOP, dBB-near-collisions for related-key NMAC-MD5, meet-in-the-middle attack *etc.* In particular, we generalize a previous key-recovery technique as a new tool exploiting a conditional key-dependent distribution. Our attack also improves the partial-key (K_1) recovery on MD5-MAC, and extends it to recover both K_1 and K_2.

Keywords: HMAC · Sandwich-MAC · MD5-MAC · MD5 · Key recovery

1 Introduction

A Message Authentication Code (MAC) is a cryptographic primitive which produces authenticity and data integrity. It takes a message M and a secret key K as input and computes a tag τ. A secure MAC must resist forgery attacks.

A MAC is often constructed from a hash function such as MD5 [1] and SHA-2 [2] for its performance and availability in software libraries. There are three hash-based MAC constructions [3]. Let $\mathcal{H}$ be a hash function. A *secret-prefix* method computes a tag of a message M by $\mathcal{H}(K\|M)$. A *secret-suffix* method computes a tag by $\mathcal{H}(M\|K)$. A *hybrid* method computes a tag by $\mathcal{H}(K\|M\|K)$.

When $\mathcal{H}$ processes M by iteratively applying a compression function h, a generic existential forgery attack with a complexity of $2^{n/2}$ exists for any of those three methods, where n is the size of the tag, τ, and the internal chaining variable [4]. Besides, each of the three types has its own features. The secret-prefix method

T. Lange, K. Lauter, and P. Lisoněk (Eds.): SAC 2013, LNCS 8282, pp. 493–512, 2014.
DOI: 10.1007/978-3-662-43414-7_25, © Springer-Verlag Berlin Heidelberg 2014

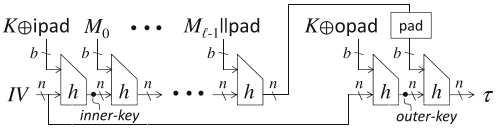

Fig. 1. Description of HMAC

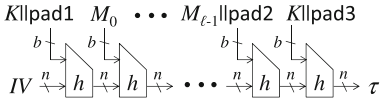

Fig. 2. Description of Sandwich-MAC

is vulnerable when a finalization process is not performed. This is called *length-extension attack* [5,6]. The secret-suffix method suffers from the collision attack on h. Two distinct messages (M, M') such that $h(M) = h(M')$ cause forgery attacks. The hybrid method seems to hide the weakness of two methods at a short glance. Strictly speaking, the hybrid method in [3] computes a tag by $\mathcal{H}(K\|\mathtt{pad}\|M\|K')$ where K and K' are two independent keys and `pad` denotes the padding string making the length of $K\|\mathtt{pad}$ equal to the block length. The security of this construction can be proven by [7] up to $O(2^{n/2})$ queries. The single-key version, where $K = K'$, is well-known as *envelope MAC*, and was standardized for IPsec version 1 [8,9]. However, Preneel and van Oorschot showed that collisions of h can reveal the second key K or K' of the hybrid method [10] when the padding is not performed between M and the second key.

Currently, the most widely used hash-based MAC is HMAC [7,11] whose structure is a hybrid method with an appropriate padding before the second key. It computes a tag by $\mathcal{H}((K \oplus \mathtt{opad})\|\mathcal{H}((K \oplus \mathtt{ipad})\|M))$ as shown in Fig. 1. HMAC was proven to be a secure PRF up to $O(2^{n/2})$ queries [12]. Several researchers proposed improvement of HMAC from various viewpoints, e.g., security bound [13], performance [14–16], and side-channel resistance [17].

Comparison of HMAC and Sandwich-MAC. Sandwich-MAC [15] is another hybrid-type MAC with an appropriate padding before the second key. It computes a tag by $\mathcal{H}(K\|\mathtt{pad1}\|M\|\mathtt{pad2}\|K)$ as shown in Fig. 2. As with HMAC, it can call current hash functions without modifying the Merkle-Damgård (MD) implementations. It was proven to have the same security as HMAC, i.e., it is a PRF up to $O(2^{n/2})$ queries as long as the underlying compression function h is a PRF. Then, Sandwich-MAC has several advantages compared to HMAC.

Sandwich-MAC can be computed only with a single key K, while HMAC creates an inner-key $h(\mathrm{IV}, K \oplus \mathtt{ipad})$ and an outer-key $h(\mathrm{IV}, K \oplus \mathtt{opad})$. This reduces the number of additional blocks, where the "additional" is defined to be the number of h invocations in the scheme minus that in the usual Merkle-Damgård. HMAC requires 3 additional blocks, while Sandwich-MAC requires 1 or 2. It also avoids a related-key attack on HMAC [18] which exploits two keys with difference $\mathtt{ipad} \oplus \mathtt{opad}$. Another advantage is the number of hash function calls. HMAC requires 2 invocations of $\mathcal{H}$, while Sandwich-MAC requires only 1.

Table 1. Summary and comparison of results. ISR stands for internal state recovery.

Target	Model	Attack goal	Data	Time	Memory	Ref.	Remarks
HMAC-MD5	Adaptive	Dist-H/ISR	2^{97}	2^{97}	2^{89}	[32]	
	Adaptive	Dist-H/ISR	$2^{89.09}$	2^{89}	2^{89}	Ours	
	Non-adaptive	Dist-H/ISR	2^{113}	2^{113}	2^{66}	[32]	
	Non-adaptive	Dist-H/ISR	2^{113-x}	2^{113-x}	2^{66+x}	Ours	$0 \leq x \leq 6$
MD5-MAC		K_1-recovery	2^{97}	2^{97}	2^{89}	[32]	
		K_1-recovery	$2^{89.09}$	2^{89}	2^{89}	Ours	
		(K_1, K_2)-recovery	$2^{89.04}$	2^{89}	2^{89}	Ours	
Sandwich-	Basic	Key recovery	$2^{89.04}$	2^{89}	2^{89}	Ours	
MAC-MD5	Variant B	Key recovery	$2^{89.04}$	2^{89}	2^{89}	Ours	
	Extended B	Key recovery	$2^{89.04}$	2^{89}	2^{89}	Ours	

As shown in [19], these drawbacks of HMAC are critical especially for short messages. Taking these features into account, though it is not widely used at present, Sandwich-MAC is potentially a good candidate for a future MAC use.

Cryptanalysis Against Hybrid MAC. If the padding is not applied before the second key, the key is recovered with $O(2^{n/2})$ [10]. The attack was optimized when the underlying hash function is MD5 [20–23] through attacks against APOP protocol [24]. In this paper, the *IV Bridge* technique [21] will be exploited. However, these analyses basically cannot be used if an appropriate padding is applied before the second key as HMAC and Sandwich-MAC.

For HMAC/NMAC, most of attacks have been proposed with specific underlying hash functions. Kim *et al.* [25] proposed the notion of distinguishing-H attack. Contini and Yin presented how to exploit a differential characteristic of an underlying compression function to recover an inner-key of HMAC/NMAC [26]. Since then, many attacks have developed for HMAC/NMAC instantiating the MD4-family [27–31]. Regarding MD5, inner-key and outer-key recovery attacks were proposed only for NMAC only in the related-key model. Wang *et al.* presented a distinguishing-H attack on full HMAC-MD5 in the single-key model [32]. This is the only known result in the single-key model against hybrid MAC constructions with an appropriate padding instantiating full MD5.

Our Contributions. In this paper, we present key-recovery attacks against several hybrid MAC schemes with an appropriate padding when MD5 is instantiated as an underlying hash function. The summary of results is given in Table 1. The main contribution is an original-key recovery attack against Sandwich-MAC-MD5. This is the first result that recovers the original-key in the hybrid method. Even if the key-length is longer than the tag size n, the key is recovered faster than 2^n computations. Moreover, an attacker does not need to know the key length in advance. Given the specification of MD5, up to a 447-bit key is recovered with $2^{89.04}$ queries, 2^{89} table look-ups, and 2^{89} memory.

For the first step, we improve the distinguishing-H attack against HMAC-MD5 in the single-key model presented by Wang *et al.* [32], which can be utilized to reveal an internal state value. This reduces the number of queries from 2^{98} to $2^{89.09}$. This can be achieved by combining the attack in [32] with the message modification technique presented by Contini and Yin [26].

We then explain our original-key recovery attack against Sandwich-MAC-MD5 and its variant with combining various techniques on MD5. Specifically, we generalize the idea in [31] as a tool exploiting conditional key-dependent distributions. Note that a similar idea can be seen in [33] against Phelix. In this paper our goal is generalizing and simplifying the technique so that it can be applied to other cases. In the below, let α, κ and β be x-bit variables, and α_i, κ_i and β_i be the i-th bit of α, κ and β, respectively, where $0 \leq i \leq x-1$.

> *Let us consider a modular addition $\alpha + \kappa = \beta$; α is a partially known variable where 1 bit (MSB) of α_{x-1} is known but α_i is unknown for the other i. κ is an unknown constant. β is a public variable computed by $\alpha + \kappa$, and its value is known. Intuitively, α, κ, and β correspond to the internal state, the key, and the tag, respectively. Then, the attacker can recover all bits of κ by iteratively collecting many pairs (β, α_{x-1}).*

Experimental verification of this observation is shown in Appendix A.

Our attack on Sandwich-MAC-MD5 recovers the key with a complexity below 2^n, hence it also leads to a universal forgery attack on Sandwich-MAC-MD5.

MD5-MAC [4] generates three keys K_0, K_1, and K_2. The previous attack [32] only recovers K_1 with a cost of 2^{97}. Our improvement of HMAC-MD5 also reduces this complexity to $2^{89.09}$. Moreover, by applying our techniques on Sandwich-MAC-MD5, we achieve the first attack that recovers both K_1 and K_2.

2 Preliminaries

2.1 HMAC

HMAC is a hash-based MAC proposed by Bellare *et al.* [7]. Denote a hash function by $\mathcal{H}$. On an input message M, HMAC based on $\mathcal{H}$ is computed using a single secret key K as HMAC-$\mathcal{H}_K(M) = \mathcal{H}(\overline{K} \oplus \texttt{opad} \| \mathcal{H}(\overline{K} \oplus \texttt{ipad} \| M))$, where $\overline{K}$ is K padded to a full block by adding '0's, `opad` and `ipad` are two public constants, and '$\|$' denotes the concatenation.

2.2 Sandwich-MAC

Sandwich-MAC [15] is another hash-based MAC proposed by Yasuda. Besides the main scheme called *Basic*, there exist three variants called *variant A*, *B*, and *C*. Inside variant B, one extension is proposed, which we call *extended B*. In this paper, we analyze Basic, variant B, and extended B. We assume that the length of the key after the padding, $|K \| \texttt{pad}|$, is shorter than the block length, b.

Sandwich-MAC Basic. Sandwich-MAC Basic computes tag values as follows.

$$\text{Sandwich-MAC-}\mathcal{H}_K(M) = \mathcal{H}(K\|\texttt{pad1}\|M\|\texttt{pad2}\|K), \tag{1}$$

where, pad1 appends $b - |K|$ bits of '0's so that $|K\|\texttt{pad1}|$ becomes equal to b and pad2 appends a single bit '1' and $b - ((|M| + 1) \bmod b)$ bits of '0's so that $|M\|\texttt{pad2}|$ becomes a multiple of b. Note that the input message for the last block always becomes $K\|\texttt{pad3}$, where pad3 is the padding scheme defined in a hash function $\mathcal{H}$. As long as MD5 is analyzed, pad3 is MD-strengthening.

Variant B and Extended B. Variant B is an optimized version when $|M|$ is already a multiple of the block length. The computation is described in Eq. (2).

Extended B is another optimization when the input message M ends in the middle of the block. Intuitively, the meaningless bits of '0' in pad3 in the last message block can be replaced with the message to be processed. For example, MD5 uses the MD-strengthening as pad3 and 65 bits are enough for it. Therefore, up to $b - |K| - 66$ bits in the last message block can be used to process the message. Let M consist of ℓ blocks $(M_0\| \cdots \|M_{\ell-1})$, and $|M_{\ell-1}| < b - |K| - 66$. The computation of extended B is described in Eq. (3).

$$\text{Sandwich-MAC}_{\text{B}}\text{-}\mathcal{H}_K(M) = \mathcal{H}(K\|\texttt{pad1}\|M\|K\|1). \tag{2}$$

$$\text{Sandwich-MAC}_{\text{ExtB}}\text{-}\mathcal{H}_K(M) = \mathcal{H}(K\|\texttt{pad1}\|M_0\| \cdots \|M_{\ell-2}\|K\|1\|M_{\ell-1}). \tag{3}$$

2.3 MD5 Specification and Free-Start Collision Attack on MD5

MD5 [1] is a Merkle-Damgård based hash function. Its block length is 512 bits and the output size is 128 bits. At first, an input message M is padded by the MD strengthening. The padded message is divided into 512-bit blocks, M_i $(i = 0, 1, \ldots, N-1)$. First H_0 is set to IV, which is the initial value defined in the specification. Then, $H_{i+1} \leftarrow h(H_i, M_i)$ is computed for $i = 0, 1, \ldots, N-1$, where h is a compression function and H_N is the hash value of M.

h takes a 128-bit value H_i and a 512-bit value M_i as input. M_i is divided into sixteen 32-bit values $m_0\|m_1\| \cdots \|m_{15}$, and H_i is divided into four 32-bit values $Q_{-3}\|Q_0\|Q_{-1}\|Q_{-2}$. Then, $Q_{j+1} \leftarrow R_j(Q_{j-3}\|Q_j\|Q_{j-1}\|Q_{j-2}, m_{\pi(j)})$ is computed for $j = 0, \ldots, 63$ and $(Q_{61}{+}Q_{-3})\|(Q_{64}{+}Q_0)\|(Q_{63}{+}Q_{-1})\|(Q_{62}{+}Q_{-2})$ is the output of h. R_j is the step function which computes Q_{j+1} as below.

$$Q_{j+1} \leftarrow Q_j + (Q_{j-3} + \Phi_j(Q_j, Q_{j-1}, Q_{j-2}) + m_{\pi(j)} + c_j) \lll s_j,$$

where Φ_j, c_j, and $\lll s_j$ denote Boolean function, constant, and left rotation by s_j-bits, respectively. $\pi(j)$ denotes a message expansion. Refer to [1] for details. Hereafter, we denote the B-th bit of variable X and Q_j by X_B and $Q_{j,B}$.

den Boer and Bosselaers [34] generated paired values (H_i, M_i) and (H'_i, M_i) such that $h(H_i, M_i) = h(H'_i, M_i)$, where H_i and H'_i have the difference: $H_i \oplus H'_i = (\texttt{80000000}, \texttt{80000000}, \texttt{80000000}, \texttt{80000000})$. Moreover, the MSB of the second, third, and fourth variables of H_i must be equal. Hereafter, we denote this difference (including two conditions of H_i) by Δ^{MSB}. To satisfy the characteristic, 46

conditions shown below must be satisfied: $Q_{j-1,31} = Q_{j-2,31}$ $(2 \leq j \leq 15)$, $Q_{j,31} = Q_{j-1,31}$ $(16 \leq j \leq 31)$, $Q_{j,31} = Q_{j-2,31}$ $(48 \leq j \leq 63)$.

3 Improved Single-Key Attacks on HMAC-MD5

3.1 Previous Distinguishing-H Attack on HMAC-MD5

Wang *et al.* presented the distinguishing-H attack on HMAC-MD5 [32], which can also recover the internal-state value. The attack aims to detect a 2-block message where Δ^{MSB} is generated by the birthday paradox in the first block and the second block forms the dBB-collision [34]. The procedure is as follows.

1. Prepare 2^{89} distinct M_0 and a single message block M_1. Then, make queries of 2^{89} two-block messages $M_0 \| M_1$, and collect collisions of tags.
2. For each collision $(M_0 \| M_1, M_0' \| M_1)$, replace M_1 with different M_1', and query $(M_0 \| M_1', M_0' \| M_1')$. If a collision of the tag is obtained, the pair is not a dBB-collision and is erased.
3. For the remaining collisions, choose up to 2^{47} distinct values of M_1', and query $(M_0 \| M_1', M_0' \| M_1')$. If a collision is obtained, the pair is a dBB-collision.

$2^{2*89-1} = 2^{177}$ pairs are generated at step 1. We expect a pair $(M_0 \| M_1, M_0' \| M_1)$ such that the internal state after the first block denoted by H_1 and H_1' satisfy Δ^{MSB} (with probability 2^{-130}; 2^{-128} for the difference and 2^{-2} for the MSB values) and the second block follows the dBB-differential characteristic (with probability 2^{-46}). The other collisions are either collisions after the first block, i.e., $H_1 = H_1'$ (2^{49} pairs), or random collisions after the second block, i.e., $\Delta H_1 \notin \{0, \Delta^{\mathrm{MSB}}\}$ (2^{50} pairs). At step 2, collisions of $H_1 = H_1'$ are erased and at step 3, a dBB-collision can be identified. Step 1 requires 2^{89} queries, table lookups, and memory. Step 2 requires $(1 + 2^{49} + 2^{50}) \cdot 2 \approx 2^{51.58}$ queries. Step 3 requires $(1 + 2^{50}) \cdot 2^{47} \approx 2^{97}$ queries. Thus, step 3 dominates the entire cost.

Wang *et al.* also tweaked their attack to a chosen message attack. Firstly choose 2^{66} distinct M_0. Secondly build a structure of 2^{66} two-block messages $M_0 \| M_1$ by choosing a random message M_1. Then build 2^{47} such structures by choosing 2^{47} distinct M_1. Thirdly, query each structure and collect collisions of the tag. Finally, for each collision $(M_0 \| M_1, M_0' \| M_1)$, check the situation for the other $2^{47} - 1$ M_1. If there exists at least one M_1' such that $(M_0 \| M_1', M_0' \| M_1')$ do not collide, which implies $H_1 \neq H_1'$, and exists another M_1'' such that $(M_0 \| M_1'', M_0' \| M_1'')$ collides, then $(M_0 \| M_1, M_0' \| M_1)$ is a dBB-collision. The attack requires $2^{66+47} = 2^{113}$ queries, while the memory is reduced to 2^{66}.

Distinguishing-H Attack. Let MD5^r be a hash function where the compression function of MD5 is replaced with a random function with the same domain and range. This implies that the domain extension and the padding algorithm for MD5^r are the same as the ones of MD5. The distinguishing-H attack aims to decide whether a given oracle is HMAC-MD5 or HMAC-MD5^r. Wang *et al.* applied their attack to the given oracle. If a dBB-collision is found, they decide that the given oracle is HMAC-MD5. Otherwise, the oracle is HMAC-MD5^r.

Internal-State Recovery Attack. After a dBB-collision $(M_0\|M_1, M_0'\|M_1)$ is obtained, Wang *et al.* apply the technique proposed by Contini and Yin [26] to recover the chaining variables $Q_7\|Q_8\|Q_9\|Q_{10}$ of $h(H_1, M_1)$. Then H_1 will be recovered by an inverse computation. For a completed description we refer to [26]. The complexity of recovering H_1 is only 2^{44} queries and 2^{60} computations. The procedure of recovering H_1 is an adaptive chosen message attack. Thus the whole attack is an adaptive chosen message attack with a complexity of 2^{97} queries.

3.2 Improved Attacks on HMAC-MD5

We observe that the complexity of the core part i.e., finding a dBB-collision can be improved by applying the technique in [26]. In order to verify whether a collision $(M_0\|M_1, M_0'\|M_1)$ is a dBB-collision at step 3, Wang *et al.* chooses 2^{47} *completely* different values as M_1' to generate a second pair following the dBB-characteristic. Our idea is generating many M_1' by modifying M_1 only *partially* so that the differential characteristic for the first several steps remains satisfied.

We focus on the computations of $h(H_1, M_1)$ and $h(H_1', M_1)$. Recall the MD5 specification. M_1 is divided into $m_0\|m_1\|\cdots\|m_{15}$ and m_i is used at step i in the first 16 steps. Our strategy is only modifying message words that appear later. Note that one bit of m_{13} and the entire bits of m_{14} and m_{15} are fixed to the padding string and thus cannot be modified. So we modify m_{12} and 31 bits of m_{13} to generate distinct $m_{12}'\|m_{13}'$. Therefore, if $(M_0\|M_1, M_0'\|M_1)$ is a dBB-collision, the modified pair can always satisfy the conditions for the first 12 steps. Thus we only need to generate $2^{35(=47-12)}$ pairs at step 3. The complexity of step 3 is now reduced to $(1 + 2^{50}) \cdot 2^{35} \approx 2^{85}$ queries. Finally, the query complexity is improved from the previous 2^{97} to the sum of 2^{89} for step 1 and 2^{85} for step 3, which is $2^{89.09}$. Time and memory complexities remain unchanged (2^{89}). The success probability is around 0.87, following the similar evaluation in [32].

Our idea can also improve the previous non-adaptive chosen message attack. We prepare 2^{66+x} $(0 \le x \le 6)$ distinct values for M_0. We can make 2^{131+2x} pairs of $M_0\|M_1$ for a fixed M_1. ΔH_1 satisfies Δ^{MSB} with probability 2^{-130}, and we need 2^{131} pairs to observe this event with a good probability. Therefore, with 2^{131+2x} pairs, one pair should satisfy Δ^{MSB} at H_1 and conditions for the first $2x$ steps in the second block. Then, M_1 is partially modified. We choose 2^{47-2x} distinct M_1 differing in the words m_{2x} and m_{2x+1}, and build 2^{47-2x} structures. Then, the above conditions are satisfied in any structure. Finally we find about two collisions $(M_0\|M_1, M_0'\|M_1)$ and $(M_0\|M_1', M_0'\|M_1')$, where $H_1 \neq H_1'$ holds, i.e., there exists at least one M_1'' such that $(M_0\|M_1'', M_0'\|M_1'')$ do not collide. The complexity is 2^{113-x} queries and the memory is 2^{66+x}, where $0 \le x \le 6$.

4 Key Recovery Attacks on Sandwich-MAC-MD5

4.1 Attacks on Sandwich-MAC-MD5 Basic

We show the attack for a key K with $|K| < 447$, which indicates that $K\|\mathtt{pad3}$ fits in one block. The attack can recover all bits of $K\|\mathtt{pad3}$ and the value of $\mathtt{pad3}$

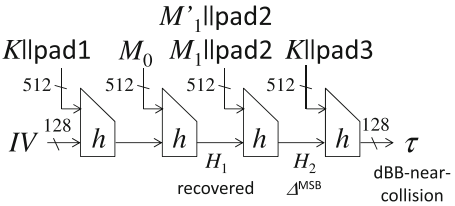

Fig. 3. Attack structure for Sandwich-MAC-MD5

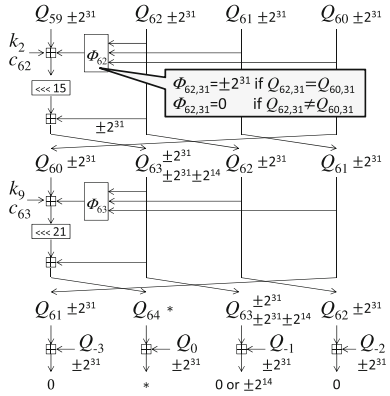

Fig. 4. dBB-near-collisions

depends on $|K|$. Hence the attacker does not have to know $|K|$ in advance. Also note that the value of pad3 is determined as the MD-strengthening defined in MD5, whereas the Sandwich-MAC can principally accept any padding scheme but the same padding as pad1. Our attack can be extended for any padding scheme as long as $K\|$pad3 fits in one block. Hereafter, we denote a 512-bit value $K\|$pad3 by sixteen 32-bit values $k_0\|k_1\|\cdots\|k_{15}$, and aim to recover these values.

Overview. The attack is divided into 5 phases. The structure is shown in Fig. 3

1. Apply the internal state recovery attack in Sect. 3.2 to Sandwich-MAC to obtain the first message block M_0 and the corresponding internal state H_1.
2. For the second message block, search for 2^{77} message pairs (M_1, M_1') such that $\Delta H_2 = h(H_1, M_1\|\texttt{pad2}) \oplus h(H_1, M_1'\|\texttt{pad2}) = \Delta^{\mathrm{MSB}}$. Because H_1 is already recovered, the computation can be done offline.
3. Query 2^{77} 2-block message pairs $(M_0\|M_1, M_0\|M_1')$, and pick the ones which produce dBB-near-collisions at the tag τ. A pair forms a dBB-near-collision with a probability 2^{-45}. Hence, we will obtain $2^{77-45} = 2^{32}$ pairs.
4. From 2^{32} pairs, recover the 32-bit subkey for the last step by exploiting a conditional key-dependent distribution.
5. As with phase 4, recover 512-bit key during the last 16 steps.

Phase 1: Internal State Recovery. The same procedure as the internal state recovery for HMAC-MD5 can be applied. Strictly speaking, the procedure can be optimized for Sandwich-MAC. Recall that our method in Sect. 3.2 could not modify m_{14} and m_{15} because they are fixed for the padding. In Sandwich-MAC, pad2 forces only 1 bit to be fixed, and thus we can modify m_{14} and 31 bits of m_{15}. This reduces the number of queries from $2^{89} + 2^{85}$ to $2^{89} + 2^{84} \approx 2^{89.04}$.

Phase 2: Generating (M_1, M_1') Producing Δ^{MSB}. This phase is offline without queries. For any underlying hash function, 2^{77} message pairs (M_1, M_1') can be found by the birthday attack with 2^{104} computations and memory. For MD5, the attack can be optimized. With the help of the collision attack techniques [35,36], Sasaki *et al.* proposed a tool called IV Bridge [21], which is a message difference producing the output difference $\Delta H_{i+1} = \Delta^{\mathrm{MSB}}$ from the input difference $\Delta H_i = 0$ with a complexity of 2^{42}. The complexity was later improved by Xie and Feng to 2^{10} [37]. With the IV Bridge, message pairs can be found much faster than the birthday attack. Note that both characteristics in [21,37] assume that H_i is MD5's IV. Therefore, if IV is replaced with another H_1, the differential characteristic search must be performed again. Because the known automated differential characteristic search [37–39] can deal with any IV, a new characteristic will be found in the same manner. Also note that if the padding string `pad2` forces many bits to be fixed, the IV Bridge search becomes harder or impossible due to the hardness of applying the message modification [36]. Because `pad2` forces only 1 bit to be fixed, this is not a problem. The complexity for this phase is one execution of the differential characteristic search and $2^{10} \cdot 2^{77} = 2^{87}$ computations. The memory can be saved by running phase 3 as soon as we obtain each pair.

Phase 3: Detecting dBB-Near-Collisions. For the last message block, the probability that a pair produces the dBB-collision is 2^{-46}. We observe that producing collisions is not necessary because the attacker can observe the output values as a tag τ. Hence, the dBB-collision can be relaxed to the dBB-near-collision, and this increases the probability of the differential characteristic.

Considering the details for phase 4, the pair must follow the dBB-collision characteristic up to step 62. The differential propagation for the last 2 steps is depicted in Fig. 4. One condition in step 63 is erased, and the probability of the characteristic becomes 2^{-45}. After examining 2^{77} pairs, we obtain $2^{77-45} = 2^{32}$ pairs. This phase requires 2^{77} queries, and the memory to store 2^{32} pairs.

Note that false positives are unlikely. Our dBB-near-collisions do not produce any difference in the left most and right most words. Besides, the difference for the second right most word is limited to 2 patterns. The probability for randomly satisfying the dBB-near-collision is 2^{-95}, which is unlikely with 2^{77} trials.

Phase 4: Recovering the Last Subkey. Because both tags and H_2 are known, the attacker can compute $Q_{61}\|Q_{64}\|Q_{63}\|Q_{62}$ for each dBB-near-collision. We then analyze the last step. The equation to compute Q_{64} is $Q_{64} = Q_{63} + (Q_{60} + \Phi_{63}(Q_{63}, Q_{62}, Q_{61}) + k_9 + c_{63}) \lll 21$. The value of $(Q_{64} \ggg 21) - Q_{63} - \Phi_{63}(Q_{63}, Q_{62}, Q_{61}) - c_{63}$ can be computed with known values of $Q_{61}\|Q_{64}\|Q_{63}\|Q_{62}$. We denote this value by Z_{63}. Then, the equation becomes $Z_{63} = Q_{60} + k_9$.

We then observe that the attacker can know the MSB of Q_{60} from the difference of Q_{63}. The difference $\Delta Q_{63} = \pm 2^{31}$ indicates that $\Delta\Phi_{62} = \pm 2^{31}$. This only occurs when $Q_{62,31} = Q_{60,31}$. The difference $\Delta Q_{63} = \pm 2^{31} \pm 2^{14}$ indicates that $\Delta\Phi_{62} = 0$. This only occurs when $Q_{62,31} \neq Q_{60,31}$. Because the value of

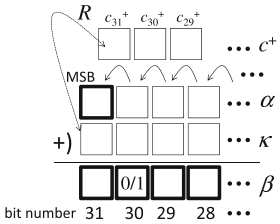

Fig. 5. Recovering κ_{31} and κ_{30}. Known bits are in bold squares.

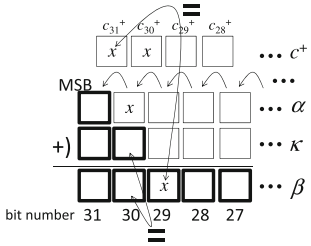

Fig. 6. Recovering κ_{29} to κ_0. Known bits are in bold squares.

Q_{62} is known, the value of $Q_{60,31}$ can be computed. In the following, we show how to recover k_9 with exploiting a conditional key-dependent distribution.

> **Conditional Key-dependent Distribution Technique:** *Let us consider a modular addition $\alpha + \kappa = \beta$; α is a variable where 1 bit (MSB) is known but the other bits are unknown. κ is an unknown constant. β is a public variable computed by $\alpha + \kappa$, and its value is known. Then, the attacker can recover all bits of κ by collecting many pairs (β, α_{x-1}).*[1]
>
> *The attacker separates the collected data into two groups depending on a condition on several bits of β. For each separated group, behavior of the other unconditioned bits is analyzed, i.e., conditional distribution is analyzed. If the conditional distribution differs depending on some bits of κ, those bits can be recovered by observing the conditional distribution.*

The details of the modular addition computation is shown in Fig. 5. We denote the carry value from bit position B to $B+1$ by c_{B+1}^+, e.g. the carry value to the MSB is c_{31}^+. Because α_{31} and β_{31} are known, the 1-bit relation of $c_{31}^+ \oplus \kappa_{31}$ denoted by R can be computed by $R = \alpha_{31} \oplus \beta_{31}$.

At first, we recover κ_{31} and κ_{30}. We separate the data into two groups by the condition $\beta_{30} = 0$ or 1, i.e., a group satisfying $\beta_{30} = 0$ and a group satisfying $\beta_{30} = 1$. For the group with $\beta_{30} = 0$, the distribution of other bits differs depending on the value of κ_{30}.

- If $\kappa_{30} = 0$, c_{31}^+ is 0 with probability 1/2 and is 1 with probability 1/2. This is because $\beta_{30} = \kappa_{30} = 0$ occurs only if $\alpha_{30} = c_{30}^+ = 0$ (with $c_{31}^+ = 0$) or $\alpha_{30} = c_{30}^+ = 1$ (with $c_{31}^+ = 1$).
- If $\kappa_{30} = 1$, c_{31}^+ is 1 with probability 1.

To utilize this difference, for each data in the group with $\beta_{30} = 0$, we simulate the value of κ_{31} by assuming that c_{31}^+ is 1. If $\kappa_{30} = 0$, the simulation returns the right value and wrong value of κ_{31} with a probability of 1/2. Therefore, we will obtain 2 possibilities of κ_{31}. If $\kappa_{30} = 1$, the simulation always returns the right

[1] As a tool, the technique can be generalized more. If the B-th bit of α is known instead of the MSB, from the LSB to the B-th bit of κ can be recovered.

value of κ_{31}. Therefore, we can obtain the unique (right) value of κ_{31}. Due to the difference, we can recover κ_{30}, and at the same time, recover κ_{31}.

We can do the same for the group with $\beta_{30} = 1$.

- If $\kappa_{30} = 0$, c_{31}^{+} is 0 with probability 1.
- If $\kappa_{30} = 1$, c_{31}^{+} is 0 with probability 1/2 and is 1 with probability 1/2.

For each data in the group with $\beta_{30} = 1$, we simulate the value of κ_{31} by assuming that c_{31}^{+} is 0, and check the number of returned values of the simulation.

We then recover κ_{29} to κ_0 in this order. In this time, we filter the data rather than separate it. In order to recover κ_B, where $29 \geq B \geq 0$, we set $(31 - B)$-bit conditions, and only pick the data satisfying all conditions. The conditions are $(\kappa_{30} = \beta_{30}), \ldots, (\kappa_{B+1} = \beta_{B+1})$, and $(c_{31}^{+} = \beta_B)$. Note that $\kappa_{31,30,\ldots,B+1}$ are already recovered and c_{31}^{+} can be easily computed by $\alpha_{31} \oplus \kappa_{31} \oplus \beta_{31}$. Let x be the value of c_{31}^{+}, where $x \in \{0, 1\}$. Then, we can deduce that the value of κ_B is x. The proof is shown below, and is described in Fig. 6.

Proof. The value of β_B is x by the condition $c_{31}^{+} = \beta_B$. From the condition $\kappa_{30} = \beta_{30}$, the values of α_{30} and c_{30}^{+} are also known to be x. By iterating the same analysis from bit position 30 to $B + 1$, the values of α_{B+1} and c_{B+1}^{+} are known to be x. The event $c_{B+1}^{+} = \beta_B = 0$ only occurs when $\kappa_B = 0$. Similarly, the event $c_{B+1}^{+} = \beta_B = 1$ only occurs when $\kappa_B = 1$. □

The number of necessary pairs to recover all bits of κ is dominated by the recovery for κ_0, which is 2^{31} pairs. To increase the success probability, we generate 2^{32} pairs. Note that these pairs can also be used to analyze the other bits.

By replacing (α, κ, β) with (Q_{60}, k_9, Z_{63}), k_9 is recovered with 2^{32} dBB-near-collisions. If a high success probability is required, more pairs than 2^{32} should be collected. See Appendix A for more discussion.

Note that recovering κ with exhaustive search instead of the conditional key-dependent distribution is possible but inefficient. The attempt is as follows. *Guess κ, and then compute α by $\beta - \kappa$. The known 1-bit α_{31} takes a role of the filtering function.* During the computation of $\beta - \kappa$, the probability that flipping κ_0 changes the value of α_{31} (through the carry effect) is 2^{-31}. If we collect 2^{32} pairs of (β, α_{x-1}) and guess 32 bits of κ, all wrong guesses can be filtered out. However, this requires 2^{64} additions, which is worse than our attack.

Phase 5: Recovering 512-Bit Key in the Last 16 Steps. This phase is basically the iteration of phase 4. After k_9 is recovered, the tag value can be computed until step 63 in backward, and the same analysis as k_9 can be applied to the second last step to recover k_2. By iterating this for the last 16 steps, the original key K and the padding string `pad3` are recovered. The number of dBB-near-collisions that we can use will increase as we recover more subkeys. This is because the probabilistic part of the differential characteristic will be shorter.

Attack Evaluation. Phase 1 requires $2^{89.04}$ queries, 2^{89} table look-ups, and a memory for 2^{89} states. Phase 2 requires $2^{10} \cdot 2^{77} = 2^{87}$ compression function computations. Phase 3 queries 2^{77} 2-block paired messages. It also requires to store 2^{32} pairs of H_2 and H_2', which requires a memory for 2^{33} states. Phase 4 requires $2^{32} \cdot 1/64 = 2^{26}$ computations. Phase 5 requires $15 \cdot 2^{32} \cdot 16/64$ which is less than 2^{34} computations. Hence, the dominant part is the internal state recovery attack for Phase 1. Our experiment in Appendix A suggests that generating more pairs at Phase 2 is better to obtain a high success probability. Then, the complexity for Phase 2 becomes 2^{88} or 2^{89} compression functions. The attack works without knowing $|K|$ as long as $|K| < 447$. The length of the queried message can always be a multiple of the block size. Hence, the attack can be extended to Sandwich-MAC variant B.

4.2 Attacks on Sandwich-MAC-MD5 Extended B

For this variant, the last message block can contain several bits chosen by the attacker. This reduces the complexity of the key recovery phase. Although the bottleneck of the attack is the internal state recovery phase, we show the attacks from two viewpoints. (1) We show the security gap between extended B and Basic. Although they have the the same provable security, the attack is easier in extended B. (2) In practice, K may be stored in a tamper-resistant device to prevent the side-channel analysis. However, the internal state value may not be protected, and the bottleneck of the attack may become the key-recovery part.

The range of $|K|$ in extended B is $|K| < 446$ because `pad3` for MD5 is 65 bits minimum and one extra bit '1' is appended right after K. Although the attack strategy and the complexity depend on $|K|$, the initial part of the attack is the same. Due to the message block structure $K\|1\|M_1\|\texttt{pad3}$ and the MD5 message expansion $\pi(\cdot)$, the first steps of the compression function are updated by K. We call these steps *keyed steps*. The following steps are updated by the controlled message or the padding string until step 16. For example, if $|K|$ is 128, the first 4 steps are the keyed steps. The initial part of the attack is as follows.

1. Recover the internal state value H_1 by applying the internal state recovery attack in Sect. 3.2 or some side-channel analysis.
2. Searching for $\#X \cdot 2^{45}$ message pairs (M_1, M_1') such that $\Delta H_2 = \Delta^{\mathrm{MSB}}$, where $\#X$ depends on $|K|$. Query them to obtain $\#X$ dBB-near-collisions.
3. Recover the internal state value right after the keyed steps by using the freedom degrees of M_2 with the approach by Contini and Yin [26].

Phase 2 requires about $\#X \cdot 2^{45} \cdot 2^{10}$ computations and $\#X \cdot 2^{45}$ queries. Phase 3 requires about $\#X \cdot 2^{47}$ queries. We then recover K with the recovered internal state value right after the keyed steps. The attack strategy depends on $|K|$.

Case Study for $|K| = 128$. Because the tag size is 128 bits, $|K| = 128$ is a natural choice. We choose $\#X = 1$ for this case. In the last block, the value of $H_2 = Q_{-3}\|Q_0\|Q_{-1}\|Q_{-2}$ is known. After phase 3, the value of $Q_1\|Q_4\|Q_3\|Q_2$

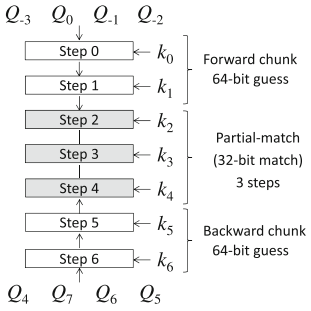

Fig. 7. MitM procedure for $|K| = 224$.

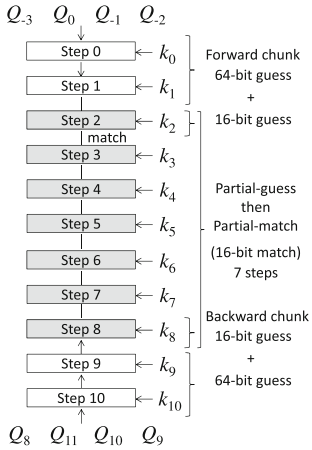

Fig. 8. MitM procedure for $|K| = 352$.

becomes known. Then, all of k_0, k_1, k_2, and k_3 are easily recovered by solving the equation of the step function, e.g. k_0 is recovered by $k_0 = ((Q_1 - Q_0) \ggg 7) - Q_{-3} - \Phi_0(Q_0, Q_{-1}, Q_{-2}) - c_0$. Other keys are also recovered with 1 computation.

Case Study for $|K| = 224$. K is divided into 7 words $k_0, \ldots, k_6$. In the last block, the values for $Q_{-3}\|Q_0\|Q_{-1}\|Q_{-2}$ and $Q_4\|Q_7\|Q_6\|Q_5$ are known after phase 3. To recover $k_0, \ldots, k_6$, we use the meet-in-the-middle (MitM) attack [40,41]. Particularly, *all subkey recovery attacks* [42] can be applied directly. The attack structure is depicted in Fig. 7. For each of the forward and backward chunks, the attacker guesses 64 key bits. The results from two chunks can match without computing 3 middle steps with the *partial-matching* [43]. To reduce the key space into a sufficiently small size, 4 pairs of $Q_{-3}\|Q_0\|Q_{-1}\|Q_{-2}$ and $Q_4\|Q_7\|Q_6\|Q_5$ are required. Hence, we set $\#X = 4$. The attack complexity is about $4 \cdot 2^{64} = 2^{66}$.

Case Study for $|K| = 352$. K is divided into 11 words $k_0, \ldots, k_{10}$. The attack structure is depicted in Fig. 8. For each chunk, 16 key bits are additionally guessed (all bits of k_0, k_1, k_9, k_{10} and 16 bits of k_2, k_8). This increases the number of skipped steps from 3 to 7 with the *partial-fixing* [44] or the *indirect partial-matching* [45]. To reduce the key space, we use 10 pairs of $Q_{-3}\|Q_0\|Q_{-1}\|Q_{-2}$ and $Q_8\|Q_{11}\|Q_{10}\|Q_9$, thus $\#X = 10$. The complexity for the attack is about $10 \cdot 2^{80} < 2^{84}$. After k_0, k_1, k_9, k_{10} and 16 bits of k_2, k_8 are recovered, the remaining 192 bits can be recovered by iterating the MitM attack. Note that if $|K| > 352$, the attack becomes worse than the one in Sect 4.1.

5 Discussion About HMAC and Sandwich-MAC

The compression function takes two information as input; previous chaining variable and message. For block-cipher based compression functions including the MD4-family, these correspond to the key input and plaintext input. Matyas-Meyer-Oseas (MMO) mode [46, Algorithm 9.41] takes the previous chaining variable as the key input and Davies-Meyer (DM) mode [46, Algorithm 9.42] takes it as the message input. The main difference between HMAC and Sandwich-MAC is the structure of the finalization (computation after M is processed by the MD structure). HMAC adopts the MMO mode while Sandwich-MAC adopts the Davies-Meyer DM mode. Our attack shows that the (outer-)key can be recovered if both modes in the MD structure and the finalization are the DM-mode and a differential characteristic ($\Delta H_i \neq 0, \Delta M = 0, \Delta H_{i+1} = 0$) exists in h. The attack can also work if both modes are the MMO-mode. In summary, to minimize the risk, using different modes for the MD structure and the finalization is preferable. On the other hand, Okeya showed that, among 12 secure PGV modes [47], using the MMO-mode in the finalization is the only choice to protect the outer-key from the side-channel analysis [48,49]. Taking into account our results, Okeya's results, and the fact that most of hash functions in practice adopt the DM-mode, we can learn that the HMAC construction is best.

The padding rule can impact the attack complexity. If the MD-strengthening is adopted as `pad2` of Sandwich-MAC, the number of attacker's controlling bits decreases. This prevents the IV Bridge and makes the attack less efficient.

There are some gaps between the hardness of the attack and the provable security. From the provable security viewpoint, the choice of the padding scheme and the choice of HMAC, Sandwich-MAC Basic, variant B, and extended B are not very different. However, once the assumption for the proof (PRF of h) is broken, these choices make a significant difference. Hence, this is a trade-off between security and performance depending on how the assumption is trusted. These differences should be taken into account when a system is designed. We never conclude that Sandwich-MAC extended B is a bad idea. Reducing the amount of meaningless padding bits is very nice especially for tree hashing, where the hash value is computed with several hash function calls and thus the amount of the padding bits is bigger than the sequential hashing. Our point is that the damage of the scheme when the assumption is broken is usually not discussed, but it deserves the careful attention because industry continues using broken hash functions such as MD5 for long time.

In general, the impact of a differential attack on h for applications is unclear. Wang *et al.* showed the characteristic with $\Pr[h(H_i, M) = h(H'_i, M)] > 2^{n/2}$ can mount the distinguishing-H attack against HMAC [32]. We extend it to the key-recovery on Sandwich-MAC. Finding such a conversion is an open problem.

6 Applications to MD5-MAC

MD5-MAC is an instantiation of the message authentication code MDx-MAC proposed by Preneel and van Oorschot [4] based on the hash function MD5.

MD5-MAC takes a single 128-bit key K as input, which is expanded to three 128-bit subkeys K_0, K_1 and K_2 as $K_i = \overline{\text{MD5}}(K\|U_i\|K), 0 \leq i \leq 2$; where $\overline{\text{MD5}}$ is the MD5 algorithm without the padding, and U_i with $0 \leq i \leq 2$ is a public constant. K_0 is used to replace the public initial value (IV) of MD5, and transforms the public function $\text{MD5}(\text{IV}, M)$ into a keyed hash function $\text{MD5}(K_0, M)$. K_1 is used inside the MD5 compression function. More precisely, K_1 is split into four 32-bit substrings $K_1[i]$ $(0 \leq i \leq 3)$, and $K_1[i]$ is added to the constants used in round i of the MD5 compression function in modulo 2^{32}. We denote MD5 with K_1 inside the compression function and without the padding by $\overline{\text{MD5}}_{K_1}$. Then K_2 is expanded to a full block $\overline{K_2}$, namely 512-bit long, as $\overline{K_2} = K_2\|(K_2 \oplus T_0)\|(K_2 \oplus T_1)\|(K_2 \oplus T_2)$ where T_i with $0 \leq i \leq 2$ is a public constant. Let M be an input message and $\texttt{pad}$ be the padding algorithm of MD5. Then, MD5-MAC is computed as below:

$$\text{MD5-MAC}_K(M) = \overline{\text{MD5}}_{K_1}(K_0, \texttt{pad}(M)\|\overline{K_2}).$$

Previous Attacks. Wang *et al.* proposed a partial key-recovery attack on MD5-MAC [32], which recovers a 128-bit key K_1 with about 2^{97} MAC queries and $2^{61.58}$ offline computations. Their attack [32] is divided into three phases.

1. Generate 3 dBB-collisions of the form $(M_0\|M_1)$ and $(M_0'\|M_1)$.
2. Recover 95 bits of Q_1, Q_2, Q_3, Q_4, Q_5 and 90 bits of $Q_6, Q_7, Q_8, Q_9, Q_{10}$ with the method proposed by Contini and Yin [26].
3. Recover $K_1[0]$. Then recover $K_1[1], K_1[2]$, and $K_1[3]$.

The first phase requires 2^{97} queries. The second phase requires $(95 + 90) \cdot 2^{47} \approx 2^{54.53}$ queries. To recover $K_1[0]$ in the third phase, the step function equation is solved by guessing unknown 65 bits of $(Q_1, Q_2, Q_3, Q_4, Q_5)$. For each guess, the following 5 steps are computed and check the match with already recovered 90 bits of $(Q_6, Q_7, Q_8, Q_9, Q_{10})$. Hence, this requires $2^{65} \cdot 6/64 \approx 2^{61.58}$ MD5 computations. $K_1[1], K_1[2]$, and $K_1[3]$ are recovered with the divide-and-conquer approach. Hence the cost to recover each key is several iterations of 2^{32} guesses. Overall, the dominant part of the attack is finding dBB-collisions. Note that the attack cannot recover any information about K_0 and K_2.

Improved Key Recovery for K_1. Because the dominant part of the attack is finding 3 dBB-collisions, the attack can be improved with our improved procedure on HMAC-MD5 in Sect. 3. The application is straight-forward and thus we omit the details. The attack cost becomes $2^{89.09}$ queries and 2^{89} table lookups.

Extended Key Recovery for K_2. Once K_1 is recovered, the MAC computation structure becomes essentially the same as the one for Sandwich-MAC Basic with MD5. Because our attack on Sandwich-MAC-MD5 can recover 512-bit secret information of the last message block faster than 2^{128} queries and computations, a 512-bit key $\overline{K_2}$ can be recovered with exactly the same procedure as the one for Sandwich-MAC-MD5. The bottleneck of the attack is still

finding dBB-collisions, which requires $2^{89.04}$ queries and 2^{89} table lookups. We emphasize that this is the first result which can recover K_2 of MD5-MAC.

7 Concluding Remarks

In this paper, we first improved the distinguishing-H attacks on HMAC-MD5. We then proposed the key-recovery attack on Sandwich-MAC-MD5 by combining various techniques. In particular, we generalized the key-recovery technique exploiting the conditional key-dependent distributions. As a result, we achieved the first results that can recover the original-key against a hybrid MAC with an appropriate padding. Our results also improved the previous key-recovery attack on MD5-MAC, and extended the recovered key to both of K_1 and K_2. We believe our results lead to a better understanding of the MAC construction.

A Testing Conditional Key-Dependent Distributions

We implemented the key recovery procedure with the conditional key-dependent distributions. The first experiment verifies the key recovery procedure for κ_{31} and κ_{30}. The second experiment verifies the key recovery procedure for κ_{29} to κ_0.

To recover κ_{31} and κ_{30}, we first observe whether the simulated value of κ_{31} is always the same or not for the group with $\beta_{30} = 0$. We then observe the same thing for the group with $\beta_{30} = 1$. If $\kappa_{30} = 0$ (resp. $\kappa_{30} = 1$), two values are returned from the group with $\beta_{30} = 0$ (resp. $\beta_{30} = 1$) and only one value is returned from the group with $\beta_{30} = 1$ ($\beta_{30} = 0$). κ_{30} is recovered after two values are returned from one of two groups. If the number of data is small, the simulation may return only one value. This occurs probabilistically.

Let D be the number of available data. In our experiment, we first fix the value of κ. We then choose α D times from a uniformly distributed space, and compute $\beta = \kappa + \alpha$ for each α. Then, we run the key recovery algorithm and check κ_{30} is recovered or not i.e., one group returns two values. Finally, we iterate this procedure 100,000 times and count how many times κ_{30} is recovered. The results are shown in Table 2. From Table 2, collecting 2^{10} data seems enough to recover κ_{31} and κ_{30} with a high probability. Because our attack generates 2^{32} dBB-near-collisions, recovering κ_{30} and κ_{31} succeeds with probability almost 1.

To recover κ_{29} to κ_0, we search for a data satisfying all conditions. Because the recovery procedure is almost the same for different bit positions, we do the experiment only for recovering the 8 bits, κ_{29} to κ_{22}. The experiment successfully recovers the key as long as sufficient data is available. We performed the key recovery procedure 1,000 times by changing the number of data D. The number of successes is listed in Table 3. Underlined values show the data for the theoretical evaluation. We can see that the theoretical evaluation has a low success probability. In our attack, we generate 2^{32} data for recovering κ_0, which is a double of the theoretical evaluation. From Table 3, the probability that κ_0 is successfully recovered is expected to be about 55 %. Moreover, the success probability of recovering κ_1 is about 75 %, κ_2 is about 87 %, κ_3 is about 94 %,

Table 2. Experiment for the recovery procedure of κ_{31} and κ_{30}.

D	#Success	Success prob. (%)
2	8368	8.4
4	28673	28.7
8	56067	56.1
16	76510	76.5
32	87970	88.0
64	93743	93.7
128	96955	97.0
256	98465	98.5
512	99216	99.2

Table 3. Experiment for the recovery procedure of κ_{29} to κ_{22}.

Target bit	D									
	2	4	8	16	32	64	128	256	512	1024
κ_{29}	238	378	598	768	891	928	975	978	989	995
κ_{28}	116	202	374	559	751	886	953	973	989	987
κ_{27}	58	122	214	360	539	721	878	935	969	980
κ_{26}	15	58	101	195	361	587	731	871	952	969
κ_{25}	10	28	50	118	212	381	557	774	862	944
κ_{24}	4	20	36	70	122	208	370	566	760	875
κ_{23}	2	7	10	28	64	119	199	380	552	752
κ_{22}	4	2	8	12	31	60	104	211	373	573

and so on. If the expected value is calculated, 2.25 candidates, which is about $2^{1.18}$, candidates of κ will remain after the analysis of 2^{32} pairs. We can use the exhaustive search to reduce these space. At phase 5 of the procedure, the analysis with 2^{32} data is iterated 16 times, and thus the remaining space will be $2^{1.18*16} = 2^{18.88}$. Of course, by generating more pairs than 2^{32} at phases 2 and 3, the success probability of recovering κ can be close to 1.

References

1. Rivest, R.L.: Request for Comments 1321: The MD5 Message Digest Algorithm. The Internet Engineering Task Force (1992). http://www.ietf.org/rfc/rfc1321.txt
2. 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
3. Tsudik, G.: Message authentication with one-way hash functions. ACM SIGCOMM Comput. Commun. Rev. **22**(5), 29–38 (1992)
4. Preneel, B., van Oorschot, P.C.: MDx-MAC and building fast MACs from hash functions. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 1–14. Springer, Heidelberg (1995)

5. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: how to construct a hash function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005)
6. U.S. Department of Commerce, National Institute of Standards and Technology: Federal Register, vol. 72, no. 212, November 2, 2007/Notices (2007). http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf
7. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
8. Kaliski Jr., B.S., Robshaw, M.J.B.: Message authentication with MD5. Technical report, CryptoBytes (1995)
9. Metzger, P., Simpson, W.A.: Request for Comments 1852: IP Authentication using Keyed SHA. The Internet Engineering Task Force (1995). http://www.ietf.org/rfc/rfc1852.txt
10. Preneel, B., van Oorschot, P.C.: On the security of two MAC algorithms. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 19–32. Springer, Heidelberg (1996)
11. 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
12. Bellare, M.: New proofs for NMAC and HMAC: security without collision-resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
13. Yasuda, K.: Multilane HMAC— security beyond the birthday limit. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 18–32. Springer, Heidelberg (2007)
14. Yasuda, K.: Boosting Merkle-Damgård hashing for message authentication. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 216–231. Springer, Heidelberg (2007)
15. Yasuda, K.: "Sandwich" is indeed secure: how to authenticate a message with just one hashing. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 355–369. Springer, Heidelberg (2007)
16. Yasuda, K.: HMAC without the "second" key. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 443–458. Springer, Heidelberg (2009)
17. Gauravaram, P., Okeya, K.: An update on the side channel cryptanalysis of MACs based on cryptographic hash functions. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 393–403. Springer, Heidelberg (2007)
18. Peyrin, T., Sasaki, Y., Wang, L.: Generic related-key attacks for HMAC. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 580–597. Springer, Heidelberg (2012)
19. Patel, S.: An efficient MAC for short messages. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 353–368. Springer, Heidelberg (2003)
20. Leurent, G.: Message freedom in MD4 and MD5 collisions: application to APOP. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 309–328. Springer, Heidelberg (2007)
21. Sasaki, Y., Wang, L., Ohta, K., Kunihiro, N.: Security of MD5 challenge and response: extension of APOP password recovery attack. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 1–18. Springer, Heidelberg (2008)

22. Sasaki, Y., Yamamoto, G., Aoki, K.: Practical password recovery on an MD5 challenge and response. Cryptology ePrint Archive, Report 2007/101 (2007). http://eprint.iacr.org/2007/101
23. Wang, L., Sasaki, Y., Sakiyama, K., Ohta, K.: Bit-free collision: application to APOP attack. In: Takagi, T., Mambo, M. (eds.) IWSEC 2009. LNCS, vol. 5824, pp. 3–21. Springer, Heidelberg (2009)
24. Myers, J., Rose, M.: Post office protocol - version 3. RFC 1939 (Standard), May 1996. Updated by RFCs 1957, 2449. http://www.ietf.org/rfc/rfc1939.txt
25. Kim, J.-S., Biryukov, A., Preneel, B., Hong, S.H.: On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (extended abstract). In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 242–256. Springer, Heidelberg (2006)
26. Contini, S., Yin, Y.L.: Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 37–53. Springer, Heidelberg (2006)
27. Fouque, P.-A., Leurent, G., Nguyen, P.Q.: Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 13–30. Springer, Heidelberg (2007)
28. Lee, E., Chang, D., Kim, J.-S., Sung, J., Hong, S.H.: Second preimage attack on 3-Pass HAVAL and partial key-recovery attacks on HMAC/NMAC-3-pass HAVAL. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 189–206. Springer, Heidelberg (2008)
29. Rechberger, C., Rijmen, V.: On authentication with HMAC and Non-random properties. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 119–133. Springer, Heidelberg (2007)
30. Rechberger, C., Rijmen, V.: New results on NMAC/HMAC when instantiated with popular hash functions. J. Univ. Comput. Sci. **14**(3), 347–376 (2008)
31. Wang, L., Ohta, K., Kunihiro, N.: New key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 237–253. Springer, Heidelberg (2008)
32. Wang, X., Yu, H., Wang, W., Zhang, H., Zhan, T.: Cryptanalysis on HMAC/NMAC-MD5 and MD5-MAC. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 121–133. Springer, Heidelberg (2009)
33. Wu, H., Preneel, B.: Differential-linear attacks against the stream cipher phelix. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 87–100. Springer, Heidelberg (2007)
34. den Boer, B., Bosselaers, A.: Collisions for the compression function of MD-5. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 293–304. Springer, Heidelberg (1994)
35. Klima, V.: Tunnels in hash functions: MD5 collisions within a minute. IACR Cryptology ePrint Archive: Report 2006/105 (2006). http://eprint.iacr.org/2006/105.pdf
36. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
37. Xie, T., Feng, D.: How to find weak input differences for MD5 collision attacks. Cryptology ePrint Archive, Report 2009/223 (2009) Version 20090530:102049. http://eprint.iacr.org/2009/223
38. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: general results and applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)

39. Mendel, F., Rechberger, C., Schläffer, M.: MD5 Is weaker than weak: attacks on concatenated combiners. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 144–161. Springer, Heidelberg (2009)
40. Diffie, W., Hellman, M.E.: Exhaustive cryptanalysis of the NBS data encryption standard. Computer **10**(6), 74–84 (1977)
41. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 229–240. Springer, Heidelberg (2011)
42. Isobe, T., Shibutani, K.: All subkeys recovery attack on block ciphers: extending meet-in-the-middle approach. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 202–221. Springer, Heidelberg (2013)
43. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009)
44. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg (2009)
45. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., Wang, L.: Preimages for step-reduced SHA-2. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 578–597. Springer, Heidelberg (2009)
46. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
47. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: a synthetic approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994)
48. Okeya, K.: Side channel attacks against HMACs based on block-cipher based hash functions. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 432–443. Springer, Heidelberg (2006)
49. Okeya, K.: Side channel attacks against hash-based MACs with PGV compression functions. IEICE Transactions **91**–**A**(1), 168–175 (2008)