Equivalent Key Recovery Attacks against HMAC and NMAC with Whirlpool Reduced to 7 Rounds

Jian Guo[1], Yu Sasaki[2], Lei Wang[1], Meiqin Wang[3] ⋆, and Long Wen[3]

[1] Nanyang Technological University, Singapore
ntu.guo@gmail.com, wang.lei@ntu.edu.sg
[2] NTT Secure Platform Laboratories, Japan
sasaki.yu@lab.ntt.co.jp
[3] Key Laboratory of Cryptologic Technology and Information Security,
Ministry of Education, Shandong University, Jinan 250100, China
mqwang@sdu.edu.cn, longwen@mail.sdu.edu.cn

Abstract. A main contribution of this paper is an improved analysis against HMAC instantiating with reduced Whirlpool. It recovers equivalent keys, which are often denoted as K_{in} and K_{out}, of HMAC with 7-round Whirlpool, while the previous best attack can work only for 6 rounds. Our approach is applying the meet-in-the-middle (MITM) attack on AES to recover MAC keys of Whirlpool. Several techniques are proposed to bypass different attack scenarios between a block cipher and a MAC, *e.g.,* the chosen plaintext model of the MITM attacks on AES cannot be used for HMAC-Whirlpool. Besides, a larger state size and different key schedule designs of Whirlpool leave us a lot of room to study. As a result, equivalent keys of HMAC with 7-round Whirlpool are recovered with a complexity of (Data, Time, Memory) $=$ $(2^{481.7}, 2^{482.3}, 2^{481})$.

Keywords: HMAC, NMAC, Whirlpool, universal forgery, key recovery

1 Introduction

A cryptographic hash function is a public algorithm that compresses arbitrary long messages into short and random digests. An important application is a Message Authentication Code (MAC). A MAC is a keyed algorithm that takes a secret key and an arbitrary long message as input, and produces a short random string as the tag. The tag provides the authenticity and the integrity for the original messages. In this paper, we mainly study the security of one dedicated hash-based MAC, HMAC based on the hash function Whirlpool.

Whirlpool was proposed by Barreto and Rijmen in 2000 [1]. Its security was evaluated and approved by NESSIE [2]. Moreover, Whirlpool has been internationally standardized by ISO/IEC, and practically implemented in various cryptographic software libraries such as Crypto++ [3]. Many cryptanalysis results have been published on Whirlpool [4–9]. Particularly, collision attack and preimage attacks on Whirlpool hash function reach 5 and 6 rounds out of 10 rounds respectively [5, 9]. Moreover, a distinguisher on full-round Whirlpool compression function was found in [5]. Although it is an interesting and beautiful attack, the security impact of such a distinguisher seems limited. Thus Whirlpool still stands secure after receiving more than 10 years consecutive analysis from worldwide cryptanalysts.

The HMAC scheme was designed by Bellare *et al.* in 1996 [10], and becomes the most well-known hash-based MAC scheme. HMAC has been standardized by many international standardization organizations including ANSI, IETF, ISO and NIST. Also it has been widely deployed in various practical protocols including SSL, TLS and IPSec. Cryptanalysts have been continuously evaluating the security of both HMAC and HMAC based on dedicated hash functions. Generic attacks on HMAC include [11–14]. The attacks on HMAC with popular dedicated hash functions can be found in [15–21].

⋆ This work has been partially supported by 973 Program (No. 2013CB834205), NSFC Project (No. 61133013), Program for New Century Excellent Talents in University of China (NCET-13-0350), as well as Interdisciplinary Research Foundation of Shandong University (No. 2012JC018).

To appear in C. Cid and C. Rechberger (Eds.): FSE 2014

Table 1. Summarization of key-recovery results on HMAC-Whirlpool

Key type	#Rounds	Complexity			Reference
		Time	Memory	Data	
Original Key	5	2^{402}	2^{384}	2^{384}	[23]
	6	2^{496}	2^{448}	2^{384}	[23]
Equivalent Keys	5	2^{448}	2^{377}	2^{321}	[23]
	6	2^{451}	2^{448}	2^{384}	[23]
	7	$\mathbf{2^{482.3}}$	$\mathbf{2^{481}}$	$\mathbf{2^{481.7}}$	**Ours**

Due to the important roles of Whirlpool and HMAC in current cryptography as briefly described above, the security evaluation of HMAC-Whirlpool is important and interesting. Very recently in October 2013, ENISA (The European Union Agency for Network and Information Security) published a report for recommending cryptographic algorithms [22]. In particular, the report recommends (for future applications in industry) three dedicated hash functions with Whirlpool included, and two hash-based MAC with HMAC included. So it can be expected that HMAC-Whirlpool is going to have more applications in industry in the coming years, Thus HMAC-Whirlpool should receive a careful security evaluation from the cryptographic community in advance.

The first cryptanalysis of HMAC-Whirlpool was published by Guo *et al.* [23], which is also the only algorithmic security evaluation on HMAC-Whirlpool so far to our best knowledge. They proposed key recovery attacks on HMAC with Whirlpool reduced to 5 and 6 rounds out of 10 rounds.

Our contributions. This paper presents improved analysis on HMAC-Whirlpool. HMAC, from the original key K, derives two keys K_{in} and K_{out} which are usually referred to as *equivalent* keys. If both K_{in} and K_{out} are recovered, an adversary can perform the universal forgery attack. In this paper, we present an equivalent key recovery attack on HMAC with 7-round Whirlpool, which extends the number of attacked rounds by one round compared with previous work [23].

The design of Whirlpool is based on the AES block cipher. Our idea is applying the recent meet-in-the-middle (MITM) attack on 7-round AES [24, 25] to recover MAC keys of 7-round Whirlpool. The analysis seems quite simple at a short glance, however, such an extension is not trivial at all due to differences of attack scenarios between a block cipher and a MAC. For example, MITM attacks on AES work under the chosen plaintext model, while the input message for the outer function of HMAC is a hash digest of the inner function, which cannot be chosen by the attacker. The output of AES block cipher, *i*.e. ciphertext, can be observed by the attacker, while the output of the intermediate compression function in HMAC cannot be observed. Besides, a larger state size and different key schedule designs of Whirlpool leave us a lot of room to study.

A summary of our results and previous *key recovery* attacks is given in Table 1. Our attack can also be applied to NMAC-Whirlpool. It is interesting to recall that the current best collision and preimage attacks on Whirlpool hash function reach only 6 rounds. Such a phenomenon is not common particularly for key recovery attacks. For example, key recovery attack on HMAC-SHA-1 reaches only 34 rounds out of 80 rounds [18], while collision attack on SHA-1 hash function reaches full rounds [26] and preimage attack reaches 57 rounds [27].

Throughout this paper, we target HMAC-Whirlpool that uses a 512-bit key and produces full size, *i.e.,* 512-bit, tags. Targeting this case has theoretical interests since HMAC is defined to use a key of any bit size. Moreover, HMAC instantiating with a key size of one block of an underlying hash function (512 bits for Whirlpool) and with full size tags is utilized in cryptographic protocols. One example is HMAC-based Extract-and-Expand Key Derivation Function [28].

Besides HMAC, we briefly discuss other MACs. For Prefix-MAC with 7-round Whirlpool, we can also recover the equivalent key. On the other hand, for LPMAC with 7-round Whirlpool, we cannot recover the equivalent key. Nevertheless, we modify the attack procedure and manage to launch universal forgery attack.

To appear in C. Cid and C. Rechberger (Eds.): FSE 2014

Organization of the rest paper. Section 2 describes previous related works. Section 3 presents an overview of our attack on `HMAC` with 7-round `Whirlpool`. Section 4 describes the details of our attacks and shows the application to other `MAC`. Finally we conclude the paper in Section 5.

2 Related Work

2.1 `Whirlpool` Hash Function

`Whirlpool` [1] takes any message with less than 2^{256} bits as input, and outputs a 512-bit hash value. It adopts the Merkle-Damgård structure. The input message M is padded into a multiple of 512 bits. The 256-bit binary expression of the bit length ℓ is padded according to the MD-strengthening, *i.e.* $M\|1\|0^*\|\ell$. The padded message is divided into 512-bit blocks $M_0\|M_1\|\cdots\|M_{N-1}$. Let H_n be a 512-bit chaining variable. First, an initial value IV is assigned to H_0. Then, $H_{n+1} \leftarrow \mathcal{CF}(H_n, M_n)$ is computed for $n = 0, 1, \dots, N-1$, where $\mathcal{CF}$ is a compression function. H_N is produced as the hash value of M.

The compression function $\mathcal{CF}$ consists of an AES-based block-cipher E_k with the Miyaguchi-Preneel mode, which takes a 512-bit chaining variable H_i as a key and a 512-bit message block M_i as a plaintext. The output of $\mathcal{CF}$ is computed by $H_{n+1} \leftarrow E_{H_i}(M_i) \oplus M_i \oplus H_i$. Inside the block cipher E_k, an internal state is represented by an $8*8$ byte array. At first, H_i is assigned to the key value k_{-1}, and M_i is assigned to the plaintext s_{-1}. Then, the whitening operation with the master key k_{-1} is performed and the result is stored into a variable s_0, *i.e.* $s_0 \leftarrow k_{-1} \oplus s_{-1}$. The cipher generates ten 512-bit subkeys $k_0, k_1, \dots, k_9$ from k_{-1} by the key schedule function, and updates s_0 through ten rounds with generated subkeys. The computation of the block cipher output s_{10} is as follows:

$$\begin{aligned}\text{Key Schedule: } k_n &\leftarrow \text{AC} \circ \text{MR} \circ \text{SC} \circ \text{SB}(k_{n-1}), \text{ for } n = 0, 1, \dots, 9,\\ \text{Data Processing: } s_n &\leftarrow \text{AK} \circ \text{MR} \circ \text{SC} \circ \text{SB}(s_{n-1}), \text{ for } n = 0, 1, \dots, 9,\end{aligned}$$

where the details of each operation are as follows.

- SubBytes (SB): apply the AES S-Box to each byte.
- ShiftColumns (SC): cyclically shift the j-th column downwards by j bytes.
- MixRows (MR): multiply each row of the state matrix by an MDS matrix.
- AddRoundConstant (AC): XOR a 512-bit pre-specified constant.
- AddRoundKey (AK): XOR a 512-bit subkey k_n.

We sometimes swap the order of MR and AC for the key schedule and MR and AK for the data processing. In this case, the AK operation XORs $\text{MR}^{-1}(k_n)$. Hereafter, we denote $\text{MR}^{-1}(k_n)$ by u_n.

Notations. The byte position in the i-th row and the j-th column of state S is denoted by two-dimensional integers $S[i][j]$, where $0 \le i, j \le 7$. We denote the initial state for round n by x_n. Internal states immediately after SB, SC and MR in round n are denoted by y_n, z_n and w_n, respectively. We often denote several byte positions by using comma, *e.g.,* 8 bytes in the top row of state S are denoted by $S[0][0, 1, \dots, 7]$. We also use the following notations:

- $S[row(i)]$: 8 byte-positions in the i-th row of state S,
- $S[\text{SC}(row(i))]$: 8 byte-positions which SC is applied to $S[row(i)]$,
- $S[\text{SC}^{-1}(row(i))]$: 8 byte-positions which SC^{-1} is applied to $S[row(i)]$.

We use $\mathcal{H}$ to denote a hash function, and $\mathcal{CF}(ch, M)$ to denote a compression function. For the ease of notation, M may be of multiple blocks, then $\mathcal{CF}$ acts the same as hash function $\mathcal{H}$ except no padding.

2.2 Hash Based MACs

HMAC and NMAC. HMAC and NMAC [29] are hash-based MACs proposed by Bellare *et al.* [30, 10]. NMAC requires two keys K_{in} and K_{out} while HMAC requires only a single key K, and generates two equivalent keys by processing $K \oplus \texttt{ipad}$ and $K \oplus \texttt{opad}$, where opad and ipad are two public constants. Let $\mathcal{H}$ be a hash function. Also, let $\mathcal{H}(\mathrm{IV}, \cdot)$ represent that the initial value of $\mathcal{H}$ is IV. On an input message M, NMAC and HMAC computes the tag value as

$$\texttt{NMAC-}\mathcal{H}_{K_{in},K_{out}}(M) = \mathcal{H}\big(K_{out}, \mathcal{H}(K_{in}, M)\big),$$
$$\texttt{HMAC-}\mathcal{H}_{K}(M) = \mathcal{H}(K \oplus \texttt{opad} \| \mathcal{H}(K \oplus \texttt{ipad} \| M)).$$

Prefix-MAC and LPMAC. Prefix-MAC is a classical MAC construction [31], which computes a tag of message M by $\mathcal{H}(K\|M)$. It is known to be vulnerable against the length extension attack, *i.e.,* from a given pair of message and tag (M, t), the attacker can forge the tag for $M\|x$ for any x. However, no generic attack is known in terms of the key recovery. We suppose that the prefix K is processed independently of M, *i.e.,* the tag is computed by $\mathcal{H}(K\|\texttt{pad}\|M)$ in which pad is a padding string and the size of $K\|\texttt{pad}$ is a multiple of the block size. LPMAC is a strengthened version of Prefix-MAC [31] so that the length-extension attack is prevented. Let ℓ be the input message size. A tag is computed by $\mathcal{H}(K\|\ell\|\texttt{pad}\|M)$, where the size of $K\|\ell\|\texttt{pad}$ is a multiple of the block size.

2.3 Generic Internal State Recovery Attack on HMAC

Leurent *et al.* [14] provide a generic inner state recovery attack against HMAC/NMAC with time complexity $2^{3n/4}$. The result is that, $h_{in} = \mathcal{CF}(IV, K\|M_p^{2^{n/4}})$ can be recovered, with $M_p^{2^{n/4}}$ satisfying the padding rule, and of about $2^{n/4}$ blocks. We can then recover inner state $\mathcal{CF}(IV, K\|M_p^1)$, with M_p^1 of one block. This can be done through detecting colliding tags, *i.e.,* we randomly choose $2^{n/2}$ messages x, x' independently so that the two messages $M_p^{2^{n/4}}\|x$ and x' follow the padding rules, and query their tags, denoted as t and t', respectively. When t and t' collide, the chance that they collide at inner hash, *i.e.,* $h'_{in} = \mathcal{CF}(IV, K\|x') = \mathcal{CF}(IV, K\|M_p^{2^{n/4}}\|x) = \mathcal{CF}(h_{in}, x)$ is roughly 1/3.

2.4 6-Round Key Recovery Attack on HMAC-Whirlpool

The first cryptanalysis of HMAC-Whirlpool was published by Guo *et al.* [23], which showed a key recovery attack on HMAC reduced to 6 rounds. They first apply the generic internal state recovery in Sect. 2.3, and then find a message pair satisfying a particular differential characteristic. The fact that the pair satisfies the characteristic reduces a possible differential patterns of internal states. This allows an attacker to exhaustively guess internal state values and differences, and the correct guess is identified by the MITM attack. On one hand, the MITM attack in [23] is a classic type which divides the computation into two independent subfunctions, *e.g.,* [32–35]. On the other hand, the MITM attacks on AES later explained in Sect. 2.5 are based on a different framework,

2.5 Meet-In-The-Middle Attack on AES

The unified view of a series of MITM attacks on AES [36, 24, 25, 37] is well-summarized in [24]. The concept of *δ-set* takes an important role of the attack.

Definition 1 (δ-set [38]). *Let a δ-set be a set of 256 states that are all different in one state bytes (the active byte) and all equal in the other state bytes (the inactive bytes).*

The number of active bytes in the δ-set is often increased, *e.g.,* [39]. It is easy to extend the concept of the δ-set to deal with multi-active bytes.

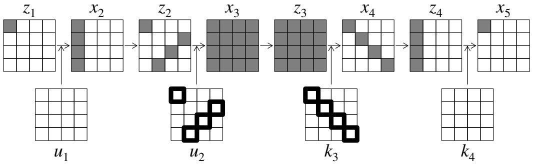


Fig. 1. 4-round differential characteristic for MITM attacks on AES [25]. Grey bytes are active. Subkey bytes represented by bold square are the ones used as parameters.

Definition 2 **(n-δ-set).** *Let an n-δ-set be a set of $(256)^n$ states that are all different in n state bytes and all equal in the other state bytes.*

The MITM attack divides the cipher into three parts:

$$s_0 \longrightarrow (s_{n_1} \longrightarrow s_{n_2}) \longrightarrow s_{\text{last}},$$

so that the middle part can satisfy a certain property, which is later verified with the partial encryption for the first part and the partial decryption for the last part. The general attack consists of the following five successive steps:

Precomputation phase

1. A lookup table T is built which contains all the possible sequences constructed from a δ-set such that one message verifies the property for the middle part.

Online phase

2. Through the oracle query, candidates of the plaintext-ciphertext pair that satisfies the target property for the middle part are searched.
3. For each candidate pair, subkeys for the first part that achieves the property for the middle part are guessed, and then the internal state value at the beginning of the middle part, s_{n_1}, is modified so that a δ-set containing a state value verifying the desired property is constructed.
4. With the guessed subkeys for the first part, the δ-set at s_{n_1} is decrypted to obtain the corresponding plaintexts. Those plaintexts are queried to the encryption oracle, and the corresponding ciphertexts are obtained.
5. Finally, subkeys for the last part are guessed, and the obtained ciphertexts associated by the δ-set at s_{n_1} are partially decrypted through the last part and tested whether it belongs to T.

If the analyzed pair is the right one and the guessed subkeys are right ones, the result of Step 5 belongs to T with probability 1, and the key is recovered. Because of Step 4, the attack is a chosen plaintext attack.

Previous work consider a function $f : \{0,1\}^8 \rightarrow \{0,1\}^8$ that maps the active byte value of a δ-set to another byte of the state after four rounds, s_{n_2}. Gilbert and Minier [37] found that an ordered sequence $(f(0), \ldots, f(255))$ for AES four rounds are parameterized only by 25 bytes, which takes significantly smaller space, *i.e.,* $2^{25\cdot 8} = 2^{200}$, than the theoretically possible space, $2^{8\cdot 2^8} = 2^{2048}$. Considering the difference $(f(0) - f(0), f(1) - f(0), \ldots, f(255) - f(0))$, the attack is improved so that the function is parameterized only by 24 bytes. Also note that the effect of the filtering with T is strong enough even with a fraction of $(f(0), \ldots, f(255))$. On average, storing $(f(0), \ldots, f(31))$ is enough to make the key space sufficiently small. Such optimization was discussed in [40].

Dunkelman *et al.* introduced the four-round truncated differential characteristic in the middle part [25], which is shown in Fig. 1. The characteristic is parameterized only by 16 bytes, *i.e.,* the states x_3 and z_3 can only take 2^{32} differences each so that the number of solutions for these two states is 2^{64}. Then, at most 4 bytes in u_2 and 4 bytes in k_3 can affect the characteristic. Hence, the number of paired state values (z_1, z_1') satisfying the characteristic is at most 2^{128}. For each of such (z_1, z_1'), the attacker constructs the δ-set at x_1 and can compute the 1-byte difference at x_5. They also introduced the concept of the multiset rather than the ordered sequence. This enables the attacker to avoid guessing 1 subkey byte at the online phase, *i.e.,* the

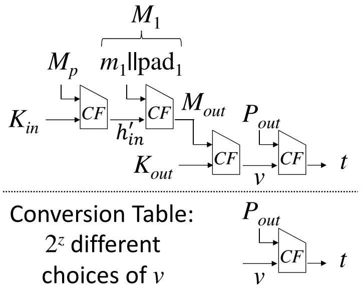
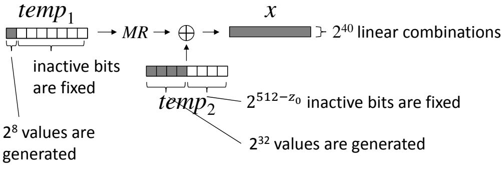


Fig. 2. Overall strategy


$temp_1$
MR
x
2^{40} linear combinations
inactive bits are fixed
$temp_2$
2^{512-z_0} inactive bits are fixed
2^8 values are generated
2^{32} values are generated

- When 2^{z_0} different x are chosen ($32 \leq z_0 \leq 56$), 2^{64-z_0} inactive bits are fixed to a constant.
- For any pair of *(in,out)* and *(in',out')* s.t. *in,in'* ∈ *temp2* and *out,out'* ∈ *x*, MR^{-1}(*in* ⊕ *out* ⊕ *in'* ⊕ *out'*) has 1 active byte with *pr* = 2^{-32}, which is bigger than 2^{-56}.

Fig. 3. Optimized choice for table inputs

partial decryption at the online phase becomes from x_1 to plaintext instead of from z_1 to plaintext, which avoids guessing 1 subkey byte to bypass the SB operation between x_1 and z_1. Because the theoretically possible numbers of multisets with 256 elements is $2^{467.6}$, the 2^{128} possible patterns for AES is significantly small, which is enough to filter out all the noise.

The latest attack by Derbez *et al.* [24] is an improvement of the attack in [25]. They found that the four-round characteristic in Fig. 1 is parameterized only by 10 bytes, *i.e.,* the number of solutions for the characteristic is at most 2^{80}. They also consider the multi-active bytes at z_1 and multi-differential characteristics for the middle four-round characteristic so that the active byte positions of z_1 and x_5 can take any of $\binom{4}{2}$ patterns and $\binom{4}{1}$ patterns respectively.

3 Overview of Our Attacks

This section gives a high level overview of our attacks on `HMAC` with 7-round `Whirlpool`. The `HMAC` computation structure in our attack is shown in the upper half of Fig. 2. Our goal is to recover the two equivalent keys K_{in} and K_{out}.

In the mode-of-operation level, we follow the approach of the previous work [23]. Namely, with the generic attack [14], we first find a single-block message M_p whose compression function output h'_{in} is recovered. The knowledge of h'_{in}, for any message of the form $M_p \| x$, allows the attacker to compute the input value for the outer hash function, M_{out}. We then recover the output value of the first compression function in the outer hash function, v, from the observed tag value, t. By iterating this procedure, the attacker collects many pairs of (M_{out}, v). We then recover $E_{K_{out}}$ by using our compression function analysis approach which will be explained in the next paragraph. Once $E_{K_{out}}$ is recovered, K_{in} can be recovered by the same analysis as K_{out}.

In the compression function level, to recover K_{out} or K_{in}, we plan to extend the recent 7-round MITM attack on `AES` [24] to attack `HMAC-Whirlpool`. More precisely, the first message block in the inner and the outer hash functions are computed as $E_{K_{in}}(M) \oplus M \oplus K_{in}$ and $E_{K_{out}}(M) \oplus M \oplus K_{out}$ respectively, where $E_{K_{in}}$ and $E_{K_{out}}$ are two `AES`-like block ciphers. The target values K_{in} and K_{out} are used as the key for the `AES`-like block ciphers. Therefore, by regarding the input value of the inner/outer hash functions as the plaintext and by regarding the output value as the ciphertext, K_{in} and K_{out} should be recovered by applying the MITM attack on `AES`. However, immediately we find that such an extension is not trivial at all because of the following differences in the attack scenarios.

- The control ability of the attacker on choosing plaintexts is different. Particularly for the outer hash function in `HMAC-Whirlpool`, the input message is the inner hash digest, and thus cannot be chosen by the attacker.
- The knowledge of the ciphertext is different. In `HMAC Whirlpool`, the output is the intermediate hash values, which are confidential to the attacker.

- The state size is different. Whirlpool has a larger state size then AES. This yields both advantages and disadvantages for the attack.
- The key schedule function is different. Several research, in the context of hash function, show that the similar diffusions between the key schedule and the data processing of Whirlpool is easier to analyze than AES [23, 5, 9].

3.1 On Recovering Ciphertexts: Generating Conversion Table

Recall Fig. 2. To recover K_{out}, the corresponding value of v is necessary. However, due to the additional padding block P_{out}, the attacker cannot observe v.

We solve this problem by generating a conversion table denoted by T_c. Note that P_{out} is a fixed value because the input message length to the outer hash function is always the same (512 bits). We then precompute $t \leftarrow \mathcal{CF}(v, P_{out})$ for many choices of v, and store pairs of (v, t) as a look-up table. Later at online phase, v can be recovered by matching the observed t and the elements in T_c.

Suppose 2^z pairs of (v, t) are generated to construct T_c. Then, for any M_{out}, we can recover the v from t with probability about 2^{z-512}.

3.2 On Choosing Plaintexts

M_{out} can be computed but cannot be chosen by the attacker. Therefore, the chosen plaintext attack cannot be used to recover K_{out} by analyzing $\mathcal{CF}(K_{out}, M_{out})$. This is crucial to apply the previous MITM attack on AES [24].

We solve this problem by converting the attack into the known plaintext attack. In the previous procedure in Sect. 2.5, queries are made in Step 2 and Step 4. Regarding Step 2, the previous work used the structure, while we generate more plaintexts at random so that the difference is satisfied probabilistically.

Converting Step 4 to the known plaintext attack is much harder. The previous attack on AES uses 256 plaintexts for the δ-set, and the corresponding multiset is queried. Without the chosen plaintext model, we cannot guarantee that all 256 plaintexts for the δ-set are known. To solve this problem, we use an n-δ-set with a relatively big n instead of a δ-set. Because n is big, we can obtain sufficient information even only with a fraction of the 2^{8n} plaintexts.

3.3 On Large State Size

A large state size is easier to analyze than a small state size. For example, guessing one row of a subkey requires only 1/8 of the entire key space for Whirlpool while it is 1/4 for AES. Then, we have more choices of attack parameters *e.g.,* the number of active bytes in the n-δ-set, the number of active rows in the input and output, *etc.* As a side-effect, identifying the best parameters becomes harder. We optimize the attack with exhaustively trying all parameters by programming.

The theoretical number of multisets for 256 elements is $2^{467.6}$, which is unlikely to occur on AES-128 where the attack complexity is below 2^{128}. However, the key space of HMAC-Whirlpool is 2^{512}, hence we sometimes cannot filter out all the noise. This problem can be solved by the weak key schedule of Whirlpool.

3.4 On Key Schedule

Recall the attack on AES in Fig. 1. For AES, 4-byte values in u_2 and k_3 do not reveal any other subkey byte in u_1 and k_4, respectively. However, for Whirlpool, 8 bytes for each inverse diagonal in $u_2[\mathrm{SR}(row(i))]$ and 8 bytes for each diagonal in $k_3[\mathrm{SR}^{-1}(row(i))]$ reveal 8 bytes of $u_1[row(i)]$ and 8 bytes of $k_4[row(i)]$, respectively. This means, we do not need previous smart ideas of [36, 25], *i.e.,* when we generate a look-up table for the middle 4-round characteristic for each parameter, we can compute rows of x_5 and rows of x_1. Hence, each element in the look-up table T_δ can be an ordered sequence of values instead of a multiset of differences. This also gives us another advantage that the theoretically possible numbers of ordered sequences is much larger than the multisets. As explained before, $2^{467.6}$ possibilities of multisets are not enough to analyze Whirlpool with a 512-bit key. By using the ordered sequence, this problem can be avoided.

3.5 Overview of the Attack

We detect that using the 4-round differential characteristic 12 → 24 → 64 → 8 → 1 for the middle 4 rounds optimizes the attack on Whirlpool. The entire differential characteristic is constructed by extending this middle 4-round characteristic by 1 round in backwards and 2 rounds in forwards. The number of solutions to satisfy the characteristic is 2^{370}. We then construct a 12-δ-set for each of all possible 2^{370} pairs, and store them as a look-up table T_δ.

Precomputation phase

0. A conversion table T_c, containing 2^z pairs of (v, t) is generated. Hereafter, for any M_{out}, we can recover the v from t with probability 2^{z-512}.
1. A lookup table T_δ is built which contains all possible 2^{370} pairs satisfying the middle 4-round characteristic. For each of 2^{370} pairs, a 12-δ-set is constructed at s_{n_1}, and an ordered sequence of 2^{96} values $f(0), \ldots, f(2^{96}-1)$ that map 12 bytes values at s_{n-1} to 1 byte value at s_{n-2} is stored.

Online phase

2. 2^Q random messages of the form $M_p \| M_1$ are queried, and only the ones whose t belongs to T_c are picked. The number of expected (M_{out}, v) is $2^{Q+(z-512)}$. To find candidate pairs satisfying the middle 4-round differential characteristic, we make about $2^{2(Q+(z-512))-1}$ pairs of (M_{out}, v), and only pick the ones satisfying the input and output differential forms.
3. For each candidate pair, subkeys for the first part and also for the last part that satisfy the middle 4-round characteristic are exhaustively guessed.
4. The internal state value at the beginning of the middle part, s_{n_1}, is modified so that a 12-δ-set is constructed. With the guessed subkeys for the first part, the 12-δ-set at s_{n_1} is decrypted to the plaintext M_{out}. If M_{out} belongs to the ones generated at Step 2, the corresponding v is recovered and (M_{out}, v) is stored as data associated with a fraction of the 12-δ-set.
5. Finally, with the guessed subkeys for the last part, each of the obtained v is partially decrypted and tested whether it belongs to T_δ. Note that the obtained (p, v) at online is the data associated with only a fraction of the 12-δ-set. Which of $f(0), \ldots, f(2^{96}-1)$ is used for the match cannot be fixed, and thus we cannot properly sort the elements in T_δ so that the match can be done only with 1 computation. We later solve this problem in Sect 4.

4 Recovering K_{in} and K_{out} of HMAC/NMAC-Whirlpool

We first apply the generic internal state recovery attack by Leurent *et al.* [14] to find a single-block message M_p and its compression function output h'_{in}.

As described in Section 3.1, in order to invert the last compression function call of the outer layer and recover the output of the attacked compression function, we build a conversion table of size 2^{512-z}, for some $z \geq 0$ to be decided later. It is important to note that, the attacker is not able to choose the M_{out}, the output of the inner layer, and M_{out} plays the role similar to "plaintext" for block ciphers in our attack. Hence, our attack setting for the key recovery is similar to the "known-plaintext" attack for block ciphers. Also, due to collisions of the compression function, the chance that a lookup gives the right v is of probability $1 - 1/e$. At the moment, the input values v of the conversion table are randomly chosen, later we show how the choices of v can be used to optimize the overall attack.

4.1 The 4-Round Differential Characteristic

Our attack follows the previous MITM attacks on AES, which relies on a 4-round differential characteristic, from state y_1 of round 2 to x_5 of round 6 of the Whirlpool compression function, as depicted in Fig. 4. The number of active bytes, in gray color, follows 12 → 24 → 64 → 8 → 1. Let us denote the set of bytes at positions $[0][0,1,2], [1][0,1,7], [2][0,6,7], [3][5,6,7]$ as B_{in}, and byte at position $[0][0]$ as B_{out}, then the input/output differences of the middle 4-round characteristic can be simply denoted as $y_1[B_{in}]$ and $x_5[B_{out}]$. With this characteristic, one round and two rounds are added before and after it, to form the 7-round Whirlpool as the attack target.

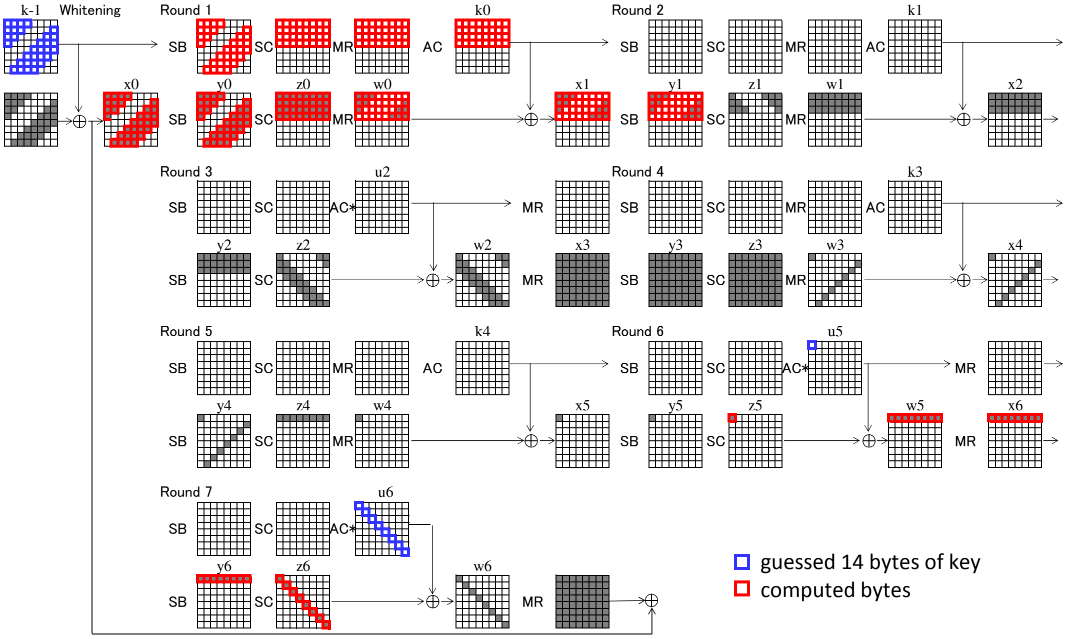


Fig. 4. Differential characteristic used in our 7-round attack

4.2 Computing the δ-set Table

Based on the 4-round differential characteristic above, we build the δ-set table, denoted as T_δ. Given a difference $\Delta y_1[B_{in}]$, and a state value of first three rows of x_2, denoted as $x_2[row(0,1,2)]$, both value and difference in active bytes of z_2 can be derived, followed by difference in active bytes of w_2 and x_3. Similarly, given a difference $\Delta x_5[B_{out}]$ and state value of $z_4[row(0)]$, $\Delta x_4[\mathrm{SC}^{-1}(row(0))]$ can be obtained, followed by Δy_3. For each pair of $(\Delta x_3, \Delta y_3)$, one solution of state value x_3 and y_3 is expected on average. With actual values in the active bytes of z_2 and w_2, $u_2[\mathrm{SC}(row(0,1,2))]$ can be derived. Due to the diffusion property of the key schedule, we can further derive other key bytes, *i.e.,* $k_1[row(0,1,2)]$. Similarly $k_3[\mathrm{SC}^{-1}(row(0))]$ is derived and we can further derive $k_4[row(0)]$. We denote all the subkey bytes obtained as k_{ob}. Due to the newly recovered k_{ob}, we can further compute more bytes in the data processing part. Finally, both value and difference in all active bytes of the 4-round characteristic can be determined. All the recovered state and key bytes can be found in Fig. 5.

These state information at every round of the differential characteristic allows to derive the value of $x_5[B_{out}]$ from any $\Delta' y_1[B_{in}]$, even if $\Delta' y_1[B_{in}]$ does not follow the characteristic. We briefly illustrate it here. Given $\Delta' y_1[B_{in}]$ and $x_2[row(0,1,2)]$, both difference and actual values of $x_2[row(0,1,2)]$ can be obtained, followed by difference in active bytes of y_2, z_2, w_2 and x_3. Together with actual value of x_3, we derive difference in y_3, z_3, w_3 and x_4. Note, for an arbitrary given $\Delta' y_1[B_{in}]$, the difference in w_3 and x_4 may not follow the characteristic any more. However, since we only care about the difference in those active bytes in gray, *i.e.,* $\Delta x_4[\mathrm{SC}^{-1}(row(0))]$, those unwanted bytes values can be discarded. Together with actual value of $x_4[\mathrm{SC}^{-1}(row(0))]$, derive the differences in the first row of z_4, w_4 and x_5. Together with actual value of $x_5[B_{out}]$, one can get the value of $x'_5[B_{out}]$. The details of the δ-set table computation are shown in Algorithm 1.

4.3 The Online Phase

For Step 2 of the procedure in Section 3.5, we search for paired message candidates satisfying the middle 4-round characteristic. To ensure at least one pair follows the entire 7-round characteristic as in Fig. 4, we

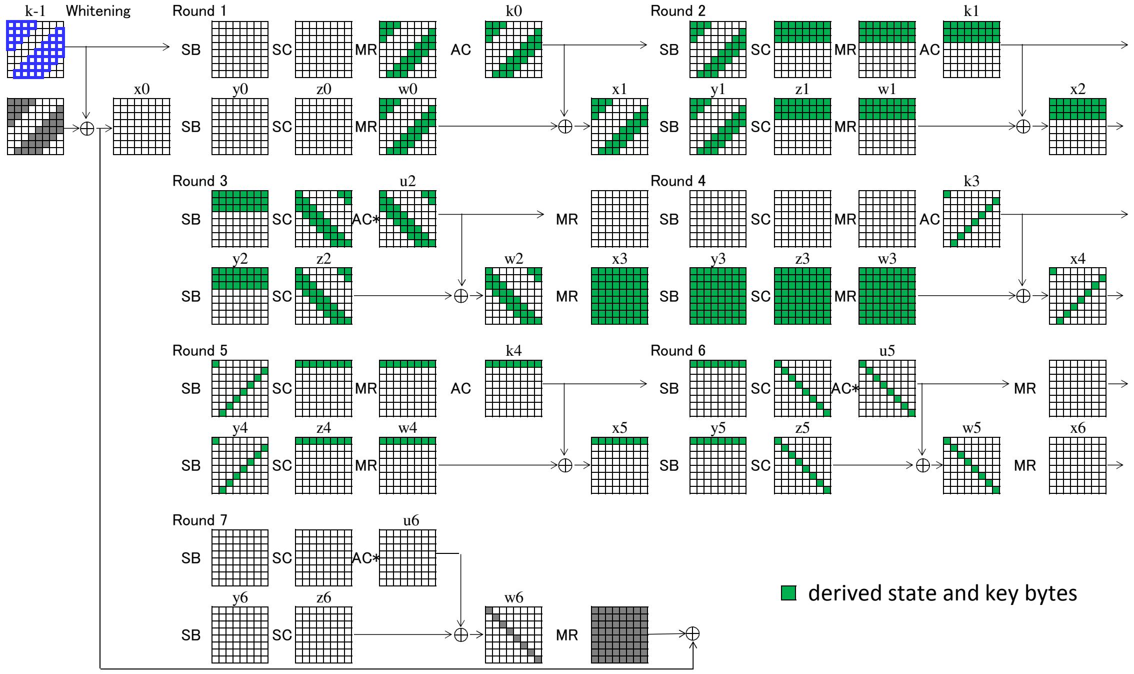


Fig. 5. Derived state and key bytes from the differential characteristic

count the number of conditions of the 7-round characteristic, there are 32 bytes conditions in the input state, 20 bytes for $z_0 \to w_0$, 56 bytes for $z_3 \to w_3$, 7 bytes for $z_4 \to w_4$, and the rest is almost for free. In total 115 bytes = 920 bits, hence we need $2^{(920+1)/2} \simeq 2^{461}$ random inputs to generate enough pairs. The overall data complexity will be $2^{461} \times 2^{512-z}$.

With the provided data (p, v) with p as input and v as output of the attacked compression function, we filter the pairs according to the input/output differences, *i.e.*, $\Delta p[\mathrm{SC}^{-1}(row(4,5,6,7))] = 0$, and output difference follows the pattern in w_6 as in Fig. 4, *i.e.*, $\Delta \mathrm{MR}^{-1}(p+v)[\mathrm{SC}(row(1,2,\ldots,7))] = 0$. We sort all the data according to the value of these non-active bytes, and find the right pairs, as done in the first for loop in Algorithm 2. A randomly generated pair satisfies both the input and output differences with a probability of $2^{-256-448} = 2^{-704}$. Hence, $2^{920-704} = 2^{216}$ candidate pairs will remain.

For Step 3 of the online phase as in Section 3.5, with any pair (p, v) and (p', v'), we partially encrypt the p, p' by 1 round, and decrypt v, v' by 2 rounds. To do that, we guess the whitening key bytes $k_{-1}[\mathrm{SC}^{-1}(row(0,1,2,3))]$ in a linear space of size $2^{12*8} = 2^{96}$ so that w_0 fulfills the desired difference pattern. Note that the guess can be done diagonal wise independently. For example, one can guess 3 bytes ([0][0], [7][1], [6][2]) of the first diagonal in k_{-1} first, compute both difference and value of these three bytes in x_0, followed by the difference in $w_0[row(0)]$, backwards compute the other 5 byte differences in the first diagonal of x_0, from which the other 5 byte value in the first diagonal of k_{-1} can be derived. Repeat the same procedure for the other diagonals, hence each key guess can be done in computation 1. Due to the similarity of the round function and key schedule, $k_0[row(0,1,2,3)]$ can be derived from $k_{-1}[\mathrm{SC}^{-1}(row(0,1,2,3))]$, then both state value and difference of $y_1[row(0,1,2,3)]$ can be obtained. Similarly, one can guess $u_6 \oplus u_{-1}[\mathrm{SC}(row(0))]$ and $u_5[0][0]$, and compute the $\Delta x_5[0][0]$. For each given (p, v), (p', v') pair, there will be 2^8 guessed keys $u_6 \oplus u_{-1}[\mathrm{SC}(row(0))]$ on average making it follow the characteristic. Overall, with each data pair, we guessed 14-byte key values. The total number of candidates so far is $2^{216+112} = 2^{328}$ pairs.

For Step 4 as in Section 3.5, with each candidate pair, the 12-δ-set is constructed with respect to $y_1[B_{in}]$. With the guessed subkeys $k_{-1}[\mathrm{SC}^{-1}(row(0,1,2,3))]$ and $k_0[row(0,1,2,3)]$, 2^{96} state values in the set are decrypted to the plaintext. Each of 2^{96} plaintexts are tested if the corresponding v is stored in Step 2. Because 2^{461} (p, v) relations are generated, the probability of having the corresponding v is $2^{461-512} = 2^{-51}$.

Algorithm 1 Construction of the δ-set table

```
Empty a lookup table $T_\delta$.
for all 12 bytes differences in $\Delta y_1[B_{in}]$ and 24 bytes values of $x_2[row(0,1,2)]$ do
    Deduce differences in $x_3$.
    for all 1 byte differences in $\Delta x_5[B_{out}]$ and 8 bytes values of $z_4[row(0)]$ do
        Deduce differences in $y_3$.
        Use the differential property of SBox to deduce the values in $x_3, x'_3, y_3, y'_3$.
        Deduce $k_{ob}$ and all state values of the active bytes of the characteristic.
        Empty an ordered sequence $M$.
        for all 12 bytes values in $y_1[B_{in}]$ do        // construct 12-$\delta$-set at $y_1$
            Compute the corresponding $x_5[B_{out}]$ and add it to $M$.
        end for
        Add $M$ to the lookup table $T_\delta$, indexed by ($\Delta y_1[B_{in}]$, $\Delta x_5[B_{out}]$, $x_2[row(0,1,2)]$, $z_4[row(0)]$), $M$ and $k_{ob}$
    as entry values.
    end for
end for
Output: $T_\delta$ of $2^{456}$ entries.        //$2^{360}$ indices each containing $2^{96}$ relations of $M$
```

Therefore, $2^{96-51} = 2^{45}$ plaintexts can have the corresponding v and stored as D_f that is associated with a fraction of the 12-δ-set. Note that we do not need to use all these 2^{45} elements in the 12-δ-set. Instead, we will select 2^8 elements from them for the attack steps later.

For Step 5 as in Section 3.5, together with the guessed subkeys $u_6 \oplus u_{-1}[\mathrm{SC}(row(0))]$ and $u_5[0][0]$, $x_5[B_{out}]$ is computed from the output values of 2^8 selected texts in D_f. Denote them as $M = \{(x_1^1, x_5^1), \ldots, (x_1^{2^8}, x_5^{2^8})\}$. Finally, we can check if 2^8 elements of $y_1[B_{in}]$ and $x_5[B_{out}]$ match with one of elements in the precomputed look-up table T_δ. However, because T_δ is not sorted with respect to the 2^8 elements, the match cannot be done with 1 computation.

The precomputation table T_δ is ordered by the index $(\Delta y_1, x_2[row(0,1,2)], z_4[row(0)], \Delta x_5)$. For each pair analyzed in the online phase, we have the knowledge of Δy_1 and Δx_5. Moreover, we have the knowledge of $y_1[row(0,1,2,3)]$ and $k_0[row(0,1,2,3)]$. If we switch the computation order of MR and AC in round 2 and let $w'_1 = z_1 \oplus u_1$, then we have $x_2 = \mathrm{MR}(w'_1)$. From $y_1[row(0,1,2,3)]$ and $k_0[row(0,1,2,3)]$, we can compute 12 bytes of $w'_1[row(0,1,2)]$. Then we exhaustively guess all the remaining 12 bytes of $w'_1[row(0,1,2)]$, which gives us the value of $x_2[row(0,1,2)]$. We also exhaustively guess all the values of $z_4[row(0)]$. Actually we need to guess the values of $z_4[0][1,2,\cdots,7]$ only, since we guessed $u_5[0][0]$ and computed $z_5[0][0]$, $z_4[0][0]$ can be derived from these two values. This gives us in total 2^{152} index of T_δ. Then we look up T_δ to get the corresponding 12-δ-set for each index. After that, we look up x_5^i for x_1^i in the 12-δ-set, and match it to the value of x_1^i in previously computed M for all $1 \leq i \leq 2^8$. We adopt early aborting technique when we look up the 12-δ-set. Namely if the x_1^i in the 12-δ-set is not equal to x_1^i in M, we immediately abort and will not match for the remaining elements. By using aborting technique, the number of table looking up for matching M to each 12-δ-set can be counted as one on average. Overall, it needs 2^{152} table lookups to match one M to the table T_δ.

The probability of the false positive, *i.e.,* the probability of two random 2^8 ordered byte relations happen to match is $2^{-8\cdot 2^8}$, which is negligible. Hence, only the right pair and the right subkey guesses can be detected. The details of the online phase attack is shown in Algorithm 2.

4.4 Complexities and Optimization

Optimizing the conversion table. We note that the conversion table is of size 2^z, and the inputs were randomly chosen from the overall 2^{512} choices. We found that, by carefully choosing the inputs, *i.e.,* the output v of the attacked compression function, we are able to reduce the conditions of the 7-round differential characteristic, by forcing v, then the w_6 as in Fig. 4, into a subspace. We know $w_6 = \mathrm{MR}^{-1}(v \oplus p \oplus k_{-1})$. Since our trick is row wise, we will take the first row as an example. We need $\Delta w_6[0][1,2,3,4,5,6,7] = 0$, and we

Algorithm 2 Online phase: attack on 7-round `Whirlpool` compression function

```
Input 1: $T_\delta$ obtained from Algorithm 1.
Input 2: Conversion table $T_c$ of size $2^z$.
Input 3: $2^d$ $(p, t)$ pairs, with $p$ input message block, and $t$ the corresponding tag.
Look up the table $T_c$ for all $t$, and obtain $2^{z+d-512}$ $(p, v)$ pairs.
Empty a temporary table $T$.
for obtained $(p, v)$ do
    $index \leftarrow \mathrm{MR}^{-1}(p \oplus v)[\mathrm{SC}(row(1,2,3,4,5,6,7))]$ and $p[\mathrm{SC}^{-1}(row(4,5,6,7))]$.
    Add $(p, t)$ to $T[index]$.
end for
for all collisions in T, i.e., pair $(p, v)$ and $(p', v')$ do
    for all $k_{-1}[\mathrm{SC}^{-1}(row(0,1,2,3))]$ s.t. $\Delta w_0[\mathrm{SC}^{-1}(row(4,5,6,7))] = 0$ do
        Construct a 12-$\delta$-set $D$ at $y_1$ by modifying 12 active bytes in $y_1[B_{in}]$.
        Decrypt $2^{96}$ elements in $D$ to the plaintext $p$, and check if $p$ is stored in $T_c$.
        If stored, add $(p, v)$ to $D_f$, which is associated to a fraction of 12-$\delta$-set.
        for all $(u_6 \oplus u_{-1})[\mathrm{SC}(row(0))]$ s.t. $\Delta z_5[0][1, \dots, 7] = 0$ do
            Decrypt each element in $D_f$ to obtain $w_5[row(0)]$.
            for all values of $u_5[0][0]$ do
                Decrypt each element in $D_f$ to obtain $w_5[0][0]$.
                Construct the ordered sequence $M$ for $D_f$.
                Compute 12 bytes of $w'_1 = z_1 \oplus u_1$: $[0][0,5,6,7], [1][0,1,6,7], [2][0,1,2,7]$.
                for all values of the 12 bytes of $w'_1$: $[0][1,2,3,4], [1][2,3,4,5], [2][3,4,5,6]$,
                 and $z_4[row(0)]$
                    compute $x_2[row(0,1,2)]$ and construct the index of $T_\delta$:
                        $(\Delta y_1[B_{in}],\ x_2[row(0,1,2)],\ \Delta_5[0][0],\ z_4[row(0)])$.
                    Get 12-$\delta$-set with the index in $T_\delta$.
                    Match $M$ to 12-$\delta$-set.
                    if $M \in T_\delta$, then
                        Exhaustively search the right key, Output: $K$ if found.
                    end if
                end for
            end for
        end for
    end for
end for
```

know there is no difference in k_{-1} and $\Delta p[row(0)]$ only takes 2^{32} values when it follows the characteristic. We force $\Delta v[row(0)]$ to be $\mathrm{MR}(\Delta w_6[0][0]) \oplus \Delta p[0][0,1,2,3]$, so that when we do $\mathrm{MR}^{-1}(\Delta p \oplus v)$, the $\Delta w_6[row(0)]$ is forced into a space of 2^{40} v.s. 2^{64}, this increases the chance $\Delta w_6[row(0)]$ to be the right pattern by a factor of 2^{24}. Note this forces the choices of $v[0]$ into a space of 2^{40} out of 2^{64}, we can apply the same trick to other rows, in the meanwhile, we need to ensure at least 2^z candidates of v are left for the conversion table.

By choosing balanced parameters $z = 481$, and $d = 481$, the overall complexity for data, and memory are 2^{481}, and time 2^{481}. Due to the false-positive of the conversion table, the data complexity increases by a factor of $(1 - 1/e)^{-1} = 2^{0.7}$ to $2^{481.7}$, and time complexity by a factor of $(1 - 1/e)^{-2} = 2^{1.3}$ to $2^{482.3}$.

Computer search. There are several parameters, which affect the overall attack complexities, including:

1. r_1, c_1, for number of active rows and columns in state z_1, respectively.
2. r_2, c_2, for number of active rows and columns in state x_5, respectively.
3. size of conversion table, denoted as $c_t = \log_2(T_c)$.

We can derive the attack complexities from these parameters in different phases:

1. The conversion table costs Memory = Time = 2^{c_t}

2. The number of conditions for the entire 7-round characteristic are: $(8 - c_1) \times 64$ for the input messages, $c_1 \times (8 - r_1) \times 8$ for $z_0 \to w_0$ transition, $(8 - r_2) \times 64$ for $z_3 \to w_3$ transition, $(8 - c_2) \times 8$ for $z_4 \to w_4$ transition. The trick with the conversion table saves $c_{con} = 512 - c_t - \lceil (512 - c_t)/(64 - 8 \times c_1) \rceil \times c_2 \times 8$ bits conditions. Denote the overall number of conditions in bits as c_{diff}, we then need $2^{c_{diff}}$ pairs to have at least one pair following the differential characteristic, due to the loss in the conversion table, overall the data complexity is Time = Data = $2^{(c_{diff}+1)/2+512-c_t}$.
3. The size of δ-set table T_δ is computed as follows. There are $r_1 \times c_1 \times 8 + r_2 \times c_2 \times 8$ bits in input/output differences, for each of them, we need to guess $r_1 \times 64 + r_2 \times 64$ state bits, and $r_1 \times c_1 \times 8$ bits state value in y_1 for the full δ-set. The overall number of bits is $\log_2(T_\delta) = r_1 \times c_1 \times 16 + r_2 \times c_2 \times 8 + r_1 \times 64 + r_2 \times 64$.
4. At online phase, we filter $(8 - c_1) \times 64$ conditions in message, and $(8 - c_2) \times 64 - c_{con}$ in ciphertext, and number of pairs left for online phase is $c_{pair} = c_d - (8 - c_1) \times 64 - (8 - c_2) \times 64 + c_{con}$. For each pair, we guessed $c_{key} = r_1 \times c_1 \times 8 + r_2 \times c_2 \times 8$ number of key bits. For each of the considered combination, we guessed $c_{mat} = r_1 \times 64 - r_1 \times c_1 \times 8 \times 2 + r_2 \times 64$ bits in the matching stage, the overall time complexity is $2^{c_{pair}+c_{key}+c_{mat}}$.

By a bruteforce search over all possible choices of the r_1, c_1, r_2, c_2 and c_t, we find the best parameters, $r_1 = 3, c_1 = 4, r_2 = 1, c_2 = 1, c_t = 481$ as presented already, give the best complexity $2^{481.7}$ and 2^{481} for Data, Memory and $2^{482.3}$ for Time.

4.5 Recovering K_{in} and Application to Prefix MAC

The attack setting for recovering K_{in} is different from K_{out}. The NMAC computation $\mathcal{H}\big(K_{out}, \mathcal{H}(K_{in}, M)\big)$ can be derived to $\mathcal{CF}(IV, K_{out} \| \mathcal{CF}(K_{in}, M \| P_{in}) \| P_{out})$, with M of full blocks, P_{in} and P_{out} for padding blocks for inner and outer layer, respectively. When M be of a fixed length, the P_{in} and P_{out} are fixed. With knowledge of K_{out}, the computation after processing M contains no secret, let us denote it as g, *i.e.,* the expression is simplified to $g \circ \mathcal{CF}(K_{in}, M)$. To recover K_{in}, the setting is a little bit different from that for recovering K_{out}. Here, we are able to *choose* the message M, the setting is similar to "chosen plaintext" attack to block ciphers. As done in previous attacks against AES, we choose the M in structures, where in each structure, the non-active bytes are fixed to a constant, and all possibilities in active bytes are taken. Refer to Fig. 4, one fixes bytes in $M[\mathrm{SR}^{-1}(row(4,5,6,7))]$ to a constant and the rest of the bytes take all 2^{256} possibilities. In this way, the pairs from the same structure will have difference pattern in message automatically fulfilled. With help of this, the overall complexities of the attack can be reduced to 2^{465} for time, data and memory with 2^7 differential characteristics.

It is interesting to note that the equivalent key for Prefix MAC can be recovered in exactly the same way. For LPMAC, $K'_{eq} = \mathcal{CF}(IV, K \| \ell \| \mathtt{pad})$ can also be recovered in the same way too. However, recovering $K_{eq} = \mathcal{CF}(IV, K)$ requires inverting the compression function $\mathcal{CF}$ since $K'_{eq} = \mathcal{CF}(K_{eq}, \ell \| \mathtt{pad})$, which is currently unknown for Whirlpool reduced to 7 rounds. Hence, in case of LPMAC, we are able to launch universal forgery attack only.

5 Conclusion

Based on very recent advances in generic state recovery attacks on HMAC/NMAC, and meet-in-the-middle attacks against AES, we present equivalent key recovery attacks against HMAC/NMAC with Whirlpool reduced to 7 rounds. We also showed the application to Prefix-MAC, including LPMAC. This improves one round over the existing work on HMAC/NMAC-Whirlpool, and interestingly, the number attacked already exceeds that for collision and preimage attack against the Whirlpool hash function itself. One reason is that, collision attacks with complexity at or beyond the birthday bound $2^{n/2}$ did not attract much attention in history, and these collisions were found to reveal information about the internal state, and also the key material in MAC applications. Due to the full block key size, the security level is expected to be higher than that of collision resistance, these collisions become essentially helpful to carry out our attacks.

Future work. It is interesting to note that length of the message plays an important role in our attacks. The current generic attacks rely on queries of long message, from $2^{n/4}$ to $2^{n/2}$ blocks, or the complexities approach 2^n when the message length tends to 1 block. Some of the hash function such as current NIST standard SHA-256, supports messages with length up to 2^{64} bits, shorter than $2^{n/4}$ blocks. This simple restriction may stop the attack or bring the attack complexity close to 2^n for many other hash functions as well. The second restriction is the length of the given message to be forged, in settings of forgery attacks. In many protocols using hash function based MAC, the message used can be very short, *e.g.,* one or two blocks. This won't stop our attacks since we focus on attacks against the compression function, which is a key advantage of our attack over the recent generic attacks. Progress in state recovery with low complexity and short messages will remove the dependency of our attack from the need of long message.

Our attacks do not apply to Sandwich MAC [41], *i.e.,* $\mathcal{H}(K\|p\|M\|p'\|K')$ with p, p' paddings, and Envelope MAC [42], *i.e.,* $\mathcal{H}(K\|p\|M\|p'\|K)$. One can still carry out the internal state recovery attack for long message between the two keys, however it is then not possible to convert it to a state recovery attack for short message, since the padding p' will be different. In our attacks, the recovered K_{in} and K_{out} play a role similar to key, while in Sandwich MAC, K' (or the second K in Envelope MAC) appears in message block, and could not be recovered since it plays a role as "plaintext", which is currently not explored in attacks against `AES`-like ciphers. For the same reason, we did not recover the master key of `HMAC`. It will be interesting to see any progress in this direction.

Acknowledgement

We would like to thank the organizers, Meiqin Wang and Hongbo Yu, of ASK 2013 workshop `http://www.infosec.sdu.edu.cn/ask2013/` in China, without which the collaboration in this work could not be possible. Jian Guo and Lei Wang were supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Rijmen, V., Barreto, P.S.L.M.: The WHIRLPOOL Hashing Function. Submitted to NISSIE (September 2000)
2. NESSIE: New European Schemes for Signatures, Integrity, and Encryption. IST-1999-12324 Available online at `http://cryptonessie.org/`.
3. Dai, W.: Crypto++ Library. `http://www.cryptopp.com/`
4. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Dunkelman, O., ed.: FSE. Volume 5665 of LNCS., Springer (2009) 260–276
5. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of LNCS., Springer (2009) 126–143
6. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. Journal of Cryptology Published online 12 November (2013) full version of Asiacrypt 2009 paper.
7. Sasaki, Y.: Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In Joux, A., ed.: FSE. Volume 6733 of LNCS., Springer (2011) 378–396
8. Wu, S., Feng, D., Wu, W., Guo, J., Dong, L., Zou, J.: (Pseudo) Preimage Attack on Round-Reduced Grøstl Hash Function and Others. In Canteaut, A., ed.: FSE. Volume 7549 of LNCS., Springer (2012) 127–145
9. Sasaki, Y., Wang, L., Wu, S., Wu, W.: Investigating Fundamental Security Requirements on Whirlpool: Improved Preimage and Collision Attacks. In Wang, X., Sako, K., eds.: ASIACRYPT. Volume 7658 of LNCS., Springer (2012) 562–579
10. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In Koblitz, N., ed.: CRYPTO. Volume 1109 of LNCS., Springer (1996) 1–15
11. Preneel, B., van Oorschot, P.C.: On the Security of Two MAC Algorithms. In Maurer, U.M., ed.: EUROCRYPT. Volume 1070 of LNCS., Springer (1996) 19–32
12. Peyrin, T., Sasaki, Y., Wang, L.: Generic Related-key Attacks for HMAC. In Wang, X., Sako, K., eds.: ASIACRYPT. Volume 7658 of LNCS., Springer (2012) 580–597

13. Naito, Y., Sasaki, Y., Wang, L., Yasuda, K.: Generic State-Recovery and Forgery Attacks on ChopMD-MAC and on NMAC/HMAC. In Sakiyama, K., Terada, M., eds.: IWSEC. LNCS (2013) 83–98
14. Leurent, G., Peyrin, T., Wang, L.: New Generic Attacks Against Hash-based MACs. In Sako, K., Sarkar, P., eds.: ASIACRYPT (2). Volume 8270 of LNCS. Springer (2013) 1–20
15. 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. Volume 4284 of LNCS., Springer (2006) 37–53
16. Fouque, P.A., Leurent, G., Nguyen, P.Q.: Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5. In Menezes, A., ed.: CRYPTO. Volume 4622 of LNCS., Springer (2007)
17. Rechberger, C., Rijmen, V.: On Authentication with HMAC and Non-random Properties. In Dietrich, S., Dhamija, R., eds.: Financial Cryptography. Volume 4886 of LNCS., Springer (2007) 119–133
18. Rechberger, C., Rijmen, V.: New Results on NMAC/HMAC when Instantiated with Popular Hash Functions. J. UCS **14** (2008) 347–376
19. Lee, E., Chang, D., Kim, J., Sung, J., Hong, S.: Second Preimage Attack on 3-Pass HAVAL and Partial Key-Recovery Attacks on HMAC/NMAC-3-Pass HAVAL. In Nyberg, K., ed.: FSE. Volume 5086 of LNCS., Springer (2008) 189–206
20. Yu, H., Wang, X.: Full Key-Recovery Attack on the HMAC/NMAC Based on 3 and 4-Pass HAVAL. In Bao, F., Li, H., Wang, G., eds.: ISPEC. Volume 5451 of LNCS., Springer (2009) 285–297
21. Sasaki, Y., Wang, L.: Improved Single-Key Distinguisher on HMAC-MD5 and Key Recovery Attacks on Sandwich-MAC-MD5. In Lange, T., Lauter, K., Lisonek, P., eds.: Selected Area in Cryptography. LNCS (2013)
22. European Union Agency for Network and Information Security: Algorithms, Key Sizes and Parameters Report. `http://www.enisa.europa.eu/` (Oct 2013)
23. Guo, J., Sasaki, Y., Wang, L., Wu, S.: Cryptanalysis of HMAC/NMAC-Whirlpool. In Sako, K., Sarkar, P., eds.: ASIACRYPT (2). Volume 8270 of LNCS. Springer (2013) 21–40
24. Derbez, P., Fouque, P.A., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Johansson, T., Nguyen, P.Q., eds.: EUROCRYPT. Volume 7881 of LNCS., Springer (2013) 371–387
25. Dunkelman, O., Keller, N., Shamir, A.: Improved Single-Key Attacks on 8-Round AES-192 and AES-256. [43] 158–176
26. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of LNCS., Springer (2005)
27. Knellwolf, S., Khovratovich, D.: New Preimage Attacks against Reduced SHA-1. In Safavi-Naini, R., Canetti, R., eds.: CRYPTO. Volume 7417 of LNCS., Springer (2012) 367–383
28. Krawczyk, H.: RFC: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). `https://tools.ietf.org/html/rfc5869.txt` (May 2010)
29. 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`.
30. Bellare, M.: New Proofs for NMAC and HMAC: Security Without Collision-Resistance. In Dwork, C., ed.: CRYPTO. Volume 4117 of LNCS., Springer (2006) 602–619
31. Tsudik, G.: Message Authentication with One-Way Hash Functions. SIGCOMM Comput. Commun. Rev. **22**(5) (October 1992) 29–38
32. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., Wang, L.: Preimages for Step-Reduced SHA-2. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of LNCS., Springer (2009) 578–597
33. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. [43] 56–75
34. Wei, L., Rechberger, C., Guo, J., Wu, H., Wang, H., Ling, S.: Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster). In Parampalli, U., Hawkes, P., eds.: ACISP. Volume 6812 of LNCS., Springer (2011) 433–438 Full version available: `http://eprint.iacr.org/2011/201`.
35. Chaum, D., Evertse, J.H.: Crytanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In Williams, H.C., ed.: CRYPTO. Volume 218 of LNCS., Springer (1985) 192–211
36. Demirci, H., Selçuk, A.A.: A Meet-in-the-Middle Attack on 8-Round AES. In Nyberg, K., ed.: FSE. Volume 5086 of LNCS., Springer (2008) 116–126
37. Gilbert, H., Minier, M.: A Collision Attack on 7 Rounds Rijndael. In: Third AES Candidate Conference (AES3), New York, USA (2000) 230–241
38. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. (1998)
39. Knudsen, L.R., Rijmen, V.: Known-Key Distinguishers for Some Block Ciphers. In Kurosawa, K., ed.: ASIACRYPT. Volume 4833 of LNCS. Springer (2007) 315–324

40. Wei, Y., Lu, J., Hu, Y.: Meet-in-the-Middle Attack on 8 Rounds of the AES Block Cipher under 192 Key Bits. In Bao, F., Weng, J., eds.: ISPEC. Volume 6672 of LNCS., Springer (2011) 222–232
41. Yasuda, K.: "Sandwich" Is Indeed Secure: How to Authenticate a Message with Just One Hashing. In Pieprzyk, J., Ghodosi, H., Dawson, E., eds.: ACISP. Volume 4586 of LNCS., Springer (2007) 355–369
42. Kaliski Jr., B.S., Robshaw, M.J.B.: Message Authentication with MD5. Technical report, CryptoBytes (1995)
43. Abe, M., ed.: Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. In Abe, M., ed.: ASIACRYPT. Volume 6477 of LNCS., Springer (2010)

To appear in C. Cid and C. Rechberger (Eds.): FSE 2014