Cryptanalyses of Narrow-Pipe Mode of Operation in AURORA-512 Hash Function

Yu Sasaki

NTT Information Sharing Platform Laboratories, NTT Corporation,
3-9-11 Midoricho, Musashino-shi, Tokyo, 180-8585 Japan
sasaki.yu@lab.ntt.co.jp
The University of Electro-Communications,
1-5-1 Choufugaoka, Choufu-shi, Tokyo, 182-8585 Japan

Abstract. We present cryptanalyses of the AURORA-512 hash function, which is a SHA-3 candidate. We first describe a collision attack on AURORA-512. We then show a second-preimage attack on AURORA-512/-384 and explain that the randomized hashing can also be attacked. We finally show a full key-recovery attack on HMAC-AURORA-512 and universal forgery on HMAC-AURORA-384. Our attack exploits weaknesses in a narrow-pipe mode of operation of AURORA-512 named "Double-Mix Merkle-Damgård (DMMD)," which produces 512-bit output by updating two 256-bit chaining variables in parallel. We do not look inside of the compression function. Hence, our attack can work even if the compression function is regarded as a random oracle. The time complexity of our collision attack is approximately 2^{236} AURORA-512 operations, and $2^{236} \times 512$ bits of memory is required. Our second-preimage attack works on any given message. The time complexity is approximately 2^{290} AURORA-512 operations, and $2^{288} \times 512$ bits of memory is required. Our key-recovery attack on HMAC-AURORA-512, which uses 512-bit secret keys, requires 2^{257} queries, 2^{259} off-line AURORA-512 operations, and a negligible amount of memory. The universal forgery on HMAC-AURORA-384 is also possible by combining the second-preimage and key-recovery attacks.

Keywords: AURORA, DMMD, collision, second preimage, HMAC.

1 Introduction

Hash functions are important cryptographic primitives used for various purposes. Currently, the National Institute of Standards and Technology (NIST) is conducting a SHA-3 competition for determining a new hash standard algorithm [1]. In the SHA-3 competition, 51 algorithms were accepted as candidates. One of the most important design aspects is the size of the internal state. To make hash algorithms efficient and compact, the internal state size should be as small as possible. If the internal state size is the same as the hash size, the structure is called a narrow-pipe mode [2]. On the other hand, to make hash algorithms secure, the internal state size should be larger than the hash size. Such a structure is called a wide-pipe mode [2]. Therefore, there is a trade-off of efficiency and security in the choice between the narrow-pipe or wide-pipe modes.

AURORA [3] is one of the hash algorithms submitted for SHA-3, which was designed by Iwata et al. AURORA mainly has four algorithms: AURORA-224, AURORA-256, AURORA-384, and AURORA-512. The output of AURORA-224 is obtained by truncating the last output value of AURORA-256. Similarly, the output of AURORA-384 is obtained by truncating the last output value of AURORA-512. Hence, two algorithms AURORA-256 and AURORA-512 are important to evaluate the security of AURORA. AURORA operates in narrow-pipe mode. In AURORA-256, a hash value is computed by iteratively applying a compression function that takes a 256-bit chaining variable and a 512-bit message block as input and a 256-bit chaining variable as output. AURORA-512 adopts a different mode of operation named Double-Mix Merkle-Damgård (DMMD), which produces 512-bit output by updating two 256-bit chaining variables in parallel. Update of 256-bit chaining variables is done by using almost the same compression functions as AURORA-256. A unique characteristic of the DMMD structure is computing 512-bit chaining variables by combining the component of AURORA-256. This gives a large advantage with respect to the size of the component to be implemented because components for AURORA-512 and AURORA-256 can be shared. Due to this structure, AURORA is efficient, especially in hardware. Hence, evaluating the security of AURORA-512 is useful for the cryptographic community to understand the tradeoff between efficiency and security in hash function design.

1.1 SHA-3 Requirements and Claimed Security of AURORA

NIST requires SHA-3 candidates to satisfy several security properties [1], e.g.,

- Preimage resistance of n bits,
- Second-preimage resistance of $n-k$ bits for any message shorter than 2^k blocks,
- Collision resistance of $n/2$ bits,
- Resistance on randomized hashing [4] of $n-k$ bits (See Section 2.2 for details.),
- $2^{n/2}$ queries and 2^n off-line computations against distinguishing attacks on HMAC [5].

According to Iwata et al. [3], DMMD has provable security for collision resistance and preimage resistance. It was proven that any adversary needs at least 2^{201} computations to find a collision of AURORA-512, and needs at least 2^{512} computations to find a preimage of AURORA-512. On the other hand, it was claimed that security of AURORA-512 is 256 bits for collision resistance, 512 bits for preimage resistance, and $(512-k)$ bits for second preimage resistance of 2^k-block messages. It is also mentioned that AURORA can be securely used as randomized hashing and as a HMAC.

1.2 Our Contribution

We investigate the weaknesses of the DMMD mode of operation adopted in AURORA-512. We first show a collision attack on AURORA-512, where the time complexity is approximately 2^{236} AURORA-512 operations and requires $2^{236} \times 512$ bits of memory. We then show a second-preimage attack on AURORA-512 and -384. Our attack generates a second preimage of any given message. Generated messages are 8 blocks long. The time complexity is approximately 2^{290} AURORA-512 operations and requires $2^{288} \times 512$ bits of memory. We then explain that the randomized hashing can also be attacked. These attacks use the multi-collision attack on a Merkle-Damgård structure proposed by Joux [6]. However, direct application of [6] to AURORA-512 does not work regarding a collision attack and is not efficient regarding a second-preimage attack. This is due to the mixing function of AURORA-512, which is designed to prevent attacks using multi-collisions such as [6]. We show that AURORA-512 is vulnerable against multi-collision attacks even if the mixing function is adopted. Note that a similar approach was taken by Knudsen et al. [7] to attack MDC2 [8]. We finally show a full key-recovery attack on HMAC-AURORA-512 with 512-bit secret keys, which require 2^{257} queries, 2^{259} off-line AURORA-512 operations, and negligible amount of memory. The universal forgery on HMAC-AURORA-384 is also possible by combining the second-preimage and key-recovery attacks. Results of our attacks are summarized in Table. 1.

Outline. In Section 2, we describe the specifications of AURORA-512, randomized hashing, and HMAC. We then introduce Joux's multi-collision attack. In Section 3, we discuss a collision attack on AURORA-512. In Section 4, we discuss a second-preimage attack on AURORA-512 and -384. We then explain that randomized hashing can also be attacked. In Section 5, we present a key recovery attack on HMAC-AURORA-512 and a universal forgery attack on

Table 1. Summary of attacks on AURORA

Attack type	Hash size	Reference	Time	Memory	
Collision	512	[9]†	$2^{234.4}$	$2^{229.6}$	
Collision	512	[9]†	2^{249}	-	
Collision	512	Ours	2^{236}	2^{236}	
2nd-preimage	512/384	[9]†	2^{291}	$2^{31.5}$	
2nd-preimage	512/384	Ours	2^{291}	2^{288}	
Randomized hash	512/384	Ours	2^{291}	2^{288}	
Attack type	Hash size	Reference	Time	Memory	Query
HMAC key recovery	512	Ours	2^{259}	–	2^{257}
HMAC universal forgery	384	Ours	2^{291}	2^{288}	2^{256}

† Ferguson and Lucks [9] also explained attacks on AURORA. Our work is independent of [9]. We describe the relationship between these two works in the Appendix.
‡ After our submission, Joux and Lucks showed improved analyses on AURORA [10].

HMAC-AURORA-384. In Section 6, we summarize what we can learn from these attacks and conclude this paper.

2 Related Works

2.1 Description of AURORA-512 and AURORA-384

We briefly describe the specifications of AURORA-512 and AURORA-384. Please refer to Ref. [3] for details.

An input message M is padded to be a multiple of 512 bits by the standard MD message padding, namely, a single bit '1', necessary numbers of '0's, and a 64-bit string representing the block length of M are appended to the end of M. Then, the padded message is divided into 512-bit message blocks $(M_0, M_1, \ldots, M_{N-1})$.

The computation for AURORA-384 is the same as AURORA-512 but for the initial value and truncating the last 512-bit value to 384 bits. Hence, we explain data processing in AURORA-512. AURORA-512 adopts a narrow-pipe mode of operation named DMMD, where two half-size (256-bit) chaining variables are updated independently by using the same message in each block. However, if all blocks are updated independently, the construction becomes vulnerable to Joux's multi-collision attack [6]. To prevent this attack, DMMD periodically computes the mixing function, which takes concatenation of two half-size chaining variables as input, to introduce the dependency of two chaining variables.

More strictly, in AURORA-512, compression functions $F_0, F_1, \ldots, F_7, G_0, G_1, \ldots, G_7 : \{0,1\}^{256} \times \{0,1\}^{512} \to \{0,1\}^{256}$, two functions $MF, MFF : \{0,1\}^{512} \to \{0,1\}^{512}$, and two 256-bit initial values (IVs) H_0^U and H_0^D are defined. The algorithm to compute a hash value is as follows. This is also illustrated in Fig. 1. In the procedure below, we use k' to denote k mod 8.

```
1. for $k$=0 to $N-1$ {
2.         $H_{k+1}^U \leftarrow F_{k'}(H_k^U, M_k)$
3.         $H_{k+1}^D \leftarrow G_{k'}(H_k^D, M_k)$
4.         if$(0 < k < N-1) \wedge (k \bmod 8 = 7)$ {
5.                 temp $\leftarrow H_{k+1}^U \| H_{k+1}^D$
6.                 $H_{k+1}^U \| H_{k+1}^D \leftarrow MF($temp$)$
7.         }
8. }
9. Output $MFF(H_N^U \| H_N^D)$
```

2.2 Description of Randomized Hashing

Randomized hashing [4] improves the security of the digital signature schemes from the collision attacks on the hash functions. It makes it difficult for the attacker to obtain a signature for one of the colliding messages from the signer and produce it as a signature for the other colliding message because the signer always uses a different random value in every signature generation process. One may note the discussion on its security by Gauravaram and Knudsen [11].

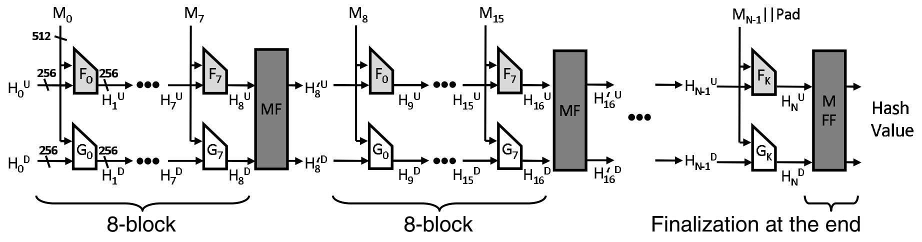

Fig. 1. Hash computation in AURORA-512

The algorithm of randomized hashing takes a message M and a key K as input and outputs a randomized message M^R. The procedure is as follows.

1. Process M to the padding procedure to make sure that the processed message M' is longer than K. Let M'_L and K_L be the length of M' and K, respectively.
2. Set *counter* $\longleftarrow \lfloor M'_L/K_L \rfloor$ and *remainder* $\longleftarrow M'_L \bmod K_L$.
3. Let R be concatenation of *counter* copies of the K and the *remainder* left-most bits of the K.
4. Output $M^R \longleftarrow K \| M' \oplus R \| K_{L(2)}$, where $K_{L(2)}$ is a 16-bit binary representation of K_L.

According to NIST [1], SHA-3 candidates with n-bit output must provide $(n-k)$-bit security in the following attack: 1) The attacker chooses a message M_1, which is shorter than 2^k blocks. 2) Then, randomization value K_1 is chosen without control of the attacker. 3) The attacker finds M_2 and K_2 s.t. $(M_1, K_1) \neq (M_2, K_2)$, which yield the same randomized hash value.

2.3 Description of HMAC

HMAC [5] is an algorithm to compute a MAC when a key and a message are input. According to Krawczyk et al. [12], the minimal recommended length for the secret key is L, where L is the size of the hash function output. Therefore, it is reasonable to use 512-bit keys for HMAC with 512-bit output hash functions, and use 384-bit keys for 384-bit output hash functions. The HMAC algorithm to compute an output with a hash function H and an initial value H_0 when a key K and a message M are input is as follows.

$$K_0 \leftarrow Pad(K), \tag{1}$$

$$\text{temp} = H(H_0, (K_0 \oplus \texttt{ipad}) \| M), \tag{2}$$

$$\text{HMAC}-H(M) = H(H_0, (K_0 \oplus \texttt{opad}) \| \text{temp}), \tag{3}$$

where, `ipad` and `opad` are constant values defined in the specification of HMAC, and $Pad(\cdot)$ is a padding process of K. In $Pad(\cdot)$, if the size of K is shorter than the block length, zeros are appended to the end of K to make its length the same as the block length signified as K_0. If the size of K and the block length are identical, K is signified as K_0.

SHA-3 candidates with n-bit output are required to be secure against distinguishing attacks that require much fewer than $2^{n/2}$ queries and significantly less computation than a preimage attack.

2.4 Description of Joux's Multi-collision Attack

Let t-collision be t different messages that result in the same hash value. Joux showed that 2^k-collision of any n-bit iterated hash function can be found with a complexity of $k \cdot 2^{\frac{n}{2}}$ [6]. For chaining variables H_j and a compression function $CF(H_j, M_j) = H_{j+1}$, Joux's attack generates (M_j, M'_j) such that $CF(H_j, M_j) = CF(H_j, M'_j) = H_{j+1}$ for $j = 0, 1, \ldots, k-1$. Any choice of (M_j, M'_j) for $j = 0, 1, \ldots, k-1$ will result in the same H_k, hence a 2^k-collision is generated.

Joux applied this technique to a cascaded construction. Let $A(\cdot)$ be an n-bit iterated hash function and $B(\cdot)$ be an n-bit hash function. For an input message M, the cascaded construction outputs a $2n$-bit value $A(M) \| B(M)$. Intuitively, the cascaded construction has $2n$-bit security. However, with Joux's attack, collisions and preimages can be found with a complexity of $\frac{n}{2} \cdot 2^{\frac{n}{2}}$ and $n \cdot 2^{\frac{n}{2}} \cdot 2^n$, respectively. To find a collision, the attacker generates a $2^{\frac{n}{2}}$-collision of $A(\cdot)$. From these $2^{\frac{n}{2}}$ messages, two paired messages will also collide with each other by computing $B(\cdot)$. To find a preimage, the attacker generates a 2^n-collision of $A(\cdot)$. Then exhaustively searches for a message that connects the collision value to the n-bit of the given hash value for $A(\cdot)$. Since there are 2^n messages that match the n-bit of given hash value, one of the messages will also satisfy the n-bit of the given hash value for $B(\cdot)$.

Restriction of Joux's Multi-collision Attack. Joux's multi-collision attack is useful if a compression function includes two independent parts through several blocks like AURORA-512. In fact, if two independent parts in AURORA-512 continues for 256 blocks or 512 blocks, the Joux's attack can be applied to find collisions or second preimages. However, the mixing function inserted at every 8 blocks guarantees that the independent part continues for at most 8 blocks, and this prevents efficient application of Joux's attack.

3 Collision Attack on AURORA-512

Our attack finds collisions of 8-block messages with a complexity of 2^{236}.

Attack Procedure. The attack procedure is as follows. The attack is also illustrated in Fig. 2

1. Randomly choose $2^{224} (= 2^{256 \cdot \frac{7}{8}})$ M_0, and compute $H_1^U \leftarrow F_0(H_0^U, M_0)$ for each M_0. This yields an 8-collision (=2^3-collision) of H_1^U.
2. By applying Joux's attack [6] to M_1 through M_6, we obtain a 2^{21}-collision of H_7^U. Let these 7-block messages yielding the 2^{21}-collision be $M_{[06]}^{(i)}, 0 \leq i \leq 2^{21} - 1$.
3. Compute $H_{k+1}^D \leftarrow G_k(H_k^D, M_k^{(i)}), 0 \leq k \leq 6$ for all i. Let $H_7^{D(i)}$ be the corresponding 2^{21} H_7^Ds.

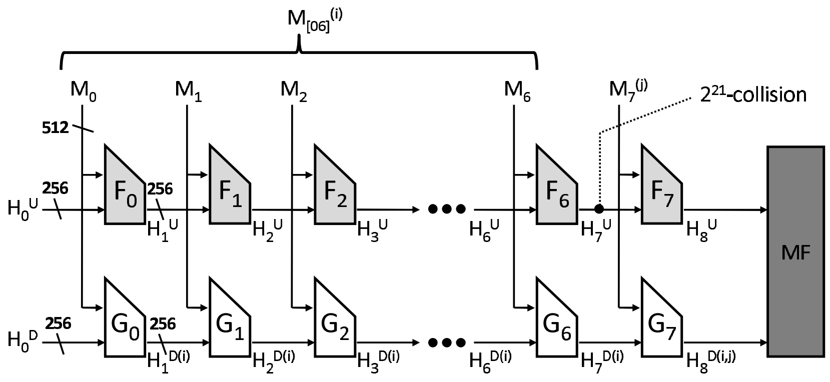

Fig. 2. Collision construction on AURORA-512

4. Set M_7 to be a randomly chosen value, and compute $H_8^{D(i)} = G_7(H_7^{D(i)}, M_7)$ for all i. Check whether or not a collision exists among 2^{21} $H_8^{D(i)}$.
5. If not, go back to Step 4 and try a different M_7. If a collision is found, let the corresponding 'i's be $i1$ and $i2$, and corresponding M_7 be $M_7^{(j)}$. Then, $M_{[06]}^{(i1)} \| M_7^{(j)}$ and $M_{[06]}^{(i2)} \| M_7^{(j)}$ are the colliding pair.

At Step 4, since there are 2^{21} $H_8^{D(i)}$, we can make roughly $2^{41}(= (2^{21})^2/2)$ pairs of $H_8^{D(i)}$. Therefore, the probability that a collision will be found is $2^{-215}(= 2^{-256} \cdot 2^{41})$. As a result, after 2^{215} iterations of Step 4, we expect to obtain a colliding pair.

Complexity Evaluation. Steps 1 and 2 cost $7 \cdot 2^{224}$ F_k-operations. Step 3 costs $7 \cdot 2^{21}$ G_k-operations. At Steps 4 and 5, the complexity of Step 4 for a chosen M_7 is 2^{21} G_k-operations. Therefore, 2^{215} iterations cost $2^{236}(= 2^{21} \cdot 2^{215})$ G_k-operations. Hence, the time complexity of this collision attack is $7 \cdot 2^{224} + 7 \cdot 2^{21} + 2^{236} \approx 2^{236}$ F_k or G_k operations. At Steps 1 and 2, we need to prepare $2^{236} \times 512$ bits of memory to find a 2^3-collision.

Remark on Success Probability of Generating Multi-collision. At Steps 1 and 2 of the attack procedure, the success probability of generating multi-collisions is much lower than 1/2. Suzuki et al. [13] gives us the complexity for finding s-collisions of n-bit value with a probability of approximately 1/2:

$$(s!)^{1/s} \times (2^{n \cdot \frac{s-1}{s}}) + s - 1. \tag{4}$$

The value of this equation is $2^{225.91} \approx 2^{226}$ when $n = 256$ and $s = 2^3$. However, considering that our attack generates 2^3-collisions 7 times at Steps 1 and 2, we need to dramatically increase the success probability. For this purpose, our attack computes 2^{230} different messages to find a 2^3-collision for each block. Since $2^{230-226} = 16$, the success probability for Steps 1 and 2 becomes $(1-(1/2)^{16})^7 \approx 1$.

Under this strategy, the attack complexity is $7 \cdot 2^{230} + 7 \cdot 2^{21} + 2^{236} = 2^{236.150}$ F_k or G_k operations, which is approximately 2^{236} AURORA-512 operations.

4 Second-Preimage Attack on AURORA-512 and -384

Our attack can generate second-preimages of any given message. Generated second-preimages are 8 blocks long. The time complexity of our attack is approximately 2^{290} AURORA-512 operations. Since the complexity is much lower than 2^{384}, the attack can also be applied to AURORA-384. Strictly speaking, the attack complexity depends on the output distribution of the compression function. We first assume that the output distribution is perfectly balanced, then discuss other cases later.

Attack Procedure. The attack procedure for some given message is as follows. The attack is also illustrated in Fig. 3.

1. Compute a hash value of the given message. Let T^U and T^D be the upper 256 bits and the lower 256 bits of the input values for the MFF function, respectively.
2. Choose an M_0 and compute $H_1^U \leftarrow F_0(H_0^U, M_0)$. Repeat this computation with changing M_0 until a 2^{32}-collision of H_1^U is obtained.
3. Following the first block, we apply Joux's attack [6] to M_1 through M_6. In total, we obtain a $2^{32\times 7} = 2^{224}$-collision of H_7^U.
4. Compute $H_8^U \leftarrow F_7(H_7^U, M_7\|\texttt{Pad})$ for $2^{288}(= 2^{256} \cdot 2^{32})$ different M_7s, where `Pad` is the padding string for 8-block messages and the length of $M_7\|\texttt{Pad}$ must be 1 block. If the output distribution of F_7 is perfectly balanced with respect to $M_7\|\texttt{Pad}$, namely, the output distribution of $F_7(H_7^U, \cdot)$ is balanced,

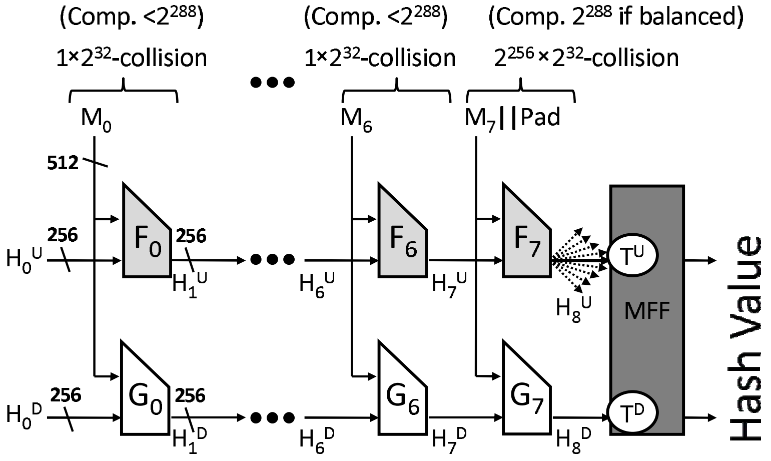

Fig. 3. Second-preimage construction for AURORA-512

we obtain 2^{32}-collisions for all possible values of H_8^U. Therefore, we obtain a 2^{32}-collision of $M_7 \| \mathtt{Pad}$ that maps H_7^U to T^U. Consequently, we obtain $2^{256} (= 2^{224} \cdot 2^{32})$ messages $M_0 \| M_1 \| \cdots \| M_7 \| \mathtt{Pad}$ that produce T^U.

5. Compute $H_{k+1}^D \leftarrow G_k(H_k^D, M_k), 0 \le k \le 7$ for all $M_0 \| M_1 \| \cdots \| M_7 \| \mathtt{Pad}$ obtained at Step 4. Since we have 2^{256} different choices, we expect that one will match T^D. The matched message $M_0 \| M_1 \| \cdots \| M_7$ is a second preimage of the given message.

Complexity Evaluation. At Steps 2 and 3, if we try $2^{288} (= 2^{256} \cdot 2^{32})$ different M_k for each block, we obtain a 2^{32}-collision due to the pigeonhole principle. The time complexity is at most $7 \cdot 2^{288}$ F_k operations, and the success probability is 1. Step 4 costs exactly 2^{288} F_7-operations if the output distribution of $F_7(H_7^U, \cdot)$ is perfectly balanced. Step 5 costs $8 \cdot 2^{256}$ G_k-operations. Therefore, the total time complexity of this attack is $7 \cdot 2^{288} + 2^{288} + 8 \cdot 2^{256} \approx 2^{291}$ F_k or G_k-operations, which is approximately 2^{290} AURORA-512 operations. At Steps 2 and 3, we need to prepare $2^{288} \times 512$ bits of memory.

Remark on Output Distribution. At Steps 2 and 3, we need only one 2^{32}-collision. Therefore, the attack complexity lessens if the distribution is not balanced. At Step 4, we need one 2^{32}-collision that produces T^U. If the distribution is not balanced and T^U is produced more frequently than other values, the complexity lessens. However, if T_U is not produced as much as other values, 2^{288} trials may not be enough to produce a desired 2^{32}-collision. In such a case, one solution is simply trying more messages until we obtain a 2^{32}-collision. Another solution is keeping other multi-collisions of H_7^U at Step 3, and start to compute F_7 by replacing the value of H_7^U.

Attack on Randomized Hashing. Second-preimage attacks that work for any IV can also attack randomized hashing if a hash function has an iterative structure, e.g., Merkle Damgård. Since our second-preimage attack can work for any IV, AURORA-512 and -384 are not secure in randomized hashing. The attack procedure is as follows. Note that this attack finds a 16-block message.

1. The attacker chooses any M and receives K that is chosen without the attacker's control. Then, compute a hash value of the randomized message and obtain T^U and T^D that are the input for the MFF function.
2. Randomly generate a 1-block value K' and a 7-block value $M'_1 \| M'_2 \| \cdots \| M'_7$.
3. Process the randomized 8-block message $K' \| K' \oplus M'_1 \| K' \oplus M'_2 \| \cdots \| K' \oplus M'_7$, and obtain $H_8'^U$ and $H_8'^D$ that are the output from the MF function.
4. Find an 8-block message $M'_8 \| M'_9 \| \cdots \| M'_{15}$ that maps $(H_8'^U \| H_8'^D)$ to $(T^U \| T^D)$, where M'_{15} is a concatenation of 431-bit free value m'_{15}, 16-bit value $K'_{L(2)}$, and 65-bit padding string for a 15-block and 447-bit message. This can be done with our second-preimage attack by considering $(H_8'^U \| H_8'^D)$ as the initial value.
5. Output the key K' and the message $M'_1 \| M'_2 \| \cdots \| M'_7 \| K' \oplus M'_8 \| K' \oplus M'_9 \| \cdots \| \lceil K' \rceil^{431} \oplus m'_{15}$, where $\lceil K' \rceil^{431}$ represents the 431 left-most bits of K'.

The attack complexity is the same as that for the second-preimage attack. Note that at Step 1 of the procedure, the message M can be randomly given. Hence, this attack is stronger than breaking randomized hashing.

Remark on Iterated Compression Function Scenario. During the 8-step computation between two MF computations, AURORA uses 16 different functions $F_0, \ldots, F_7, G_0, \ldots, G_7$. It is interesting to observe the scenario where $F_0, \ldots, F_7$ are replaced with the same function F and $G_0, \ldots, G_7$ are replaced with G. In this scenario, the attack complexity can be reduced by generating multi-fixed-points.

In this attack, for a given H_0^U, the attacker generates 2^{32} messages denoted by $M^{(S)}$ that make $F(H_0^U, M^{(S)}) = H_0^U$. This requires the time complexity of approximately 2^{288} F computations. Then, self-concatenation of any choice of $M^{(S)}$ for 7 blocks guarantees that H_7^U is equal to H_0^U because any $M^{(S)}$ maps H_0^U to H_0^U during 7 blocks. This enables us to save the complexity of generating a multi-collision 6 times. The rest of the attack is exactly the same as the one for standard AURORA-512. Finally, the time complexity becomes $2^{289} (= 2 \times 2^{288})$, which is better than the attack on standard AURORA-512.

5 Key Recovery Attack on HMAC-AURORA

In this section, we present a full key recovery attack on HMAC-AURORA-512 when 512-bit secret keys are used and the MAC length is 512-bit long. Our attack requires 2^{257} queries and the off-line complexity is 2^{259} AURORA-512 operations. The attack can be carried out with a negligible amount of memory. This attack does not make any impact on security of AURORA as a SHA-3 candidate, however, the complexity is significantly less than that of the exhaustive search for a 512-bit key. Our attack can also recover the inner-key of HMAC-AURORA-384 with almost the same complexity as in HMAC-AURORA-512. This attack does not recover the outer-key of HMAC-AURORA-384, but universal forgery is possible by combining the inner-key recovery and second-preimage attacks. Different from collision and second-preimage attacks, this attack does not use multi-collisions. Hence, this attack reveals another security weakness of AURORA-512 and -384.

5.1 Full Key Recovery Attack on HMAC-AURORA-512

In this attack, we mainly ask 1-block messages (including padding bits) as queries. The structure to process a 1-block message in HMAC-AURORA-512 is illustrated in Fig. 4.

Attack Procedure

1. Prepare 2^{257} different messages that are the same length but shorter than 448 bits so that the length of padded messages does not exceed 1-block. Let M^i be prepared messages. Ask all M^i to the oracle, and obtain corresponding $\mathrm{HMAC}_K(M^i)$.

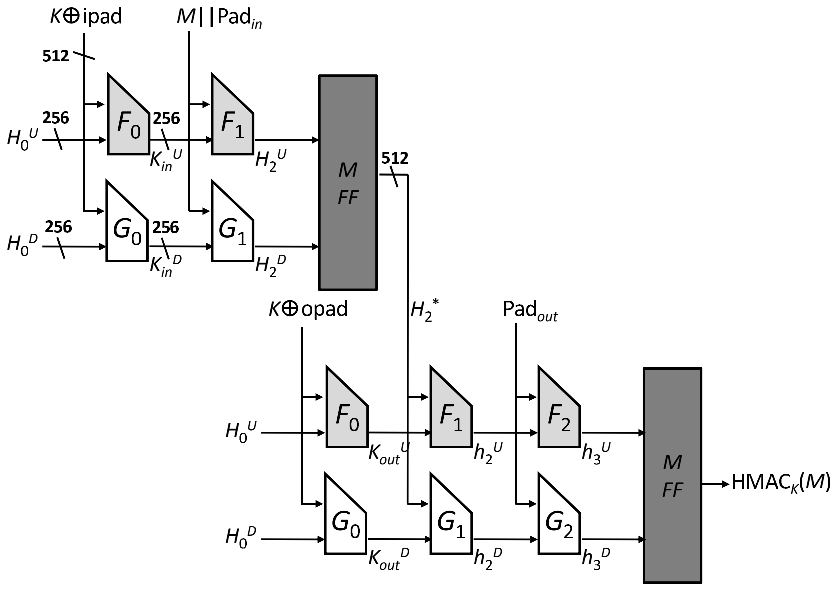

Fig. 4. Structure for processing a 1-block message in HMAC-AURORA-512

2. Find message pairs $(M^j, M^{j\prime})$ in which $\mathrm{HMAC}_K(M^j)$ and $\mathrm{HMAC}_K(M^{j\prime})$ are a collision. Due to the computation structure, a pair of messages has the following five possibilities to be a collision.
   Case 1: H_2^Us collide and H_2^Ds collide.
   Case 2: Case 1 does not occur and H_2^*s collide.
   Case 3: Case 1 and 2 do not occur and h_2^Us collide and h_2^Ds collide.
   Case 4: Case 1, 2, and 3 do not occur and h_3^Us collide and h_3^Ds collide.
   Case 5: Case 1, 2, 3, and 4 do not occur and HMAC values collide.
   Therefore, we expect to obtain several collisions in this Step.
3. To detect a Case-1 collision in Step 2, ask $M^j \| \mathtt{Pad}_{in} \| x$ and $M^{j\prime} \| \mathtt{Pad}_{in} \| x$ for any x to the oracle, and check whether $\mathrm{HMAC}_K(M^j \| \mathtt{Pad}_{in} \| x)$ and $\mathrm{HMAC}_K(M^{j\prime} \| \mathtt{Pad}_{in} \| x)$ are a collision or not. If they are a collision, $(M^j, M^{j\prime})$ is a desired pair with a negligible error probability.
4. Let $(M^{j1}, M^{j1\prime})$ be a colliding pair of Case 1 in Step 2. First, we exhaustively search for K_{in}^U by computing $F_1(K_{in}^U, M^{j1})$ and $F_1(K_{in}^U, M^{j1\prime})$ for all 2^{256} K_{in}^U and check whether the computed values are a collision or not. If they are a collision, the corresponding K_{in}^U is the correct value. Similarly, we detect K_{in}^D by computing $G_1(K_{in}^D, M^{j1})$ and $G_1(K_{in}^D, M^{j1\prime})$ for all 2^{256} K_{in}^D and check whether the computed values are a collision or not.
5. For all HMAC collision pairs $(M^j, M^{j\prime})$ obtained in Step 2, we compute the values of H_2^* and $H_2^{*\prime}$ with recovered K_{in}^U and K_{in}^D. If H_2^* and $H_2^{*\prime}$ are a collision, we discard that pair. Note, each of the remaining collision pairs are of Cases 3, 4 or 5 in Step 2.

6. Take a collision pair $(M^{j2}, M^{j2\prime})$ from all remaining collision pairs, and assume this pair is a collision of Case 3. We then recover K^U_{out} and K^D_{out} by the same method as Step 4. Namely, we exhaustively search for K^U_{out} such that $F_1(K^U_{out}, H_2^{*j2}) = F_1(K^U_{out}, H_2^{*j2\prime})$ and K^D_{out} such that $G_1(K^D_{out}, H_2^{*j2}) = G_1(K^D_{out}, H_2^{*j2\prime})$.
7. With recovered K_{in} and K_{out}, compute $\mathrm{HMAC}_K(M)$ for any M that are already asked to the oracle, and check whether its HMAC value matches with the one obtained from the oracle. If matched, that K_{out} is the correct value. Otherwise, discard the pair $(M^{j2}, M^{j2\prime})$ and go back to Step 6. Repeat the attack by choosing a different collision pair until K_{out} is recovered.

Complexity and Success Probability. At Step 1, we ask 2^{257} queries to the oracle. At Step 2, the probability that the collision of each case is obtained can be considered as independent. According to [14, Theorem 3.2], the probability of obtaining a collision for a $\log_2 N$-bit output hash function, with trying $\theta \cdot N^{1/2}$ different messages is as follows.

$$1 - e^{-\frac{\theta^2}{2}} \tag{5}$$

Eq. 5 becomes approximately 0.86 when $\theta = 2$. Therefore, we expect to obtain a collision of each case with a probability of 0.86. To successfully recover K_{in} and K_{out}, we need to obtain a Case-1 and a Case-3 collision. By 2^{257} queries, the probability of obtaining these two collisions is $(0.86)^2 \approx 0.75$. This is higher than the probability of obtaining a single collision with 2^{256} queries, which is approximately 0.39. For simplicity, we assume that five collisions in total, a single collision in each case, are obtained. At Step 3, we need two queries for each collision. Hence, if we obtained five collisions, we need eight queries in the worst case, which is negligible compared to Step 1. At Step 4, we compute F_1 $2 \cdot 2^{256}$ times to recover K^U_{in}. For each guess of K^U_{in}, the probability that $F_1(K^U_{in}, M^{j1}) = F_1(K^U_{in}, M^{j1\prime})$ is expected to be 2^{-256}. Hence, we can expect that only one K^U_{in} is chosen as the correct guess. Similarly we compute G_1 $2 \cdot 2^{256}$ times to recover K^D_{in}. As a result, the time complexity for this Step is $2 \cdot 2^{256}$ F_1-operations + $2 \cdot 2^{256}$ G_1-operations $\approx 2^{257}$ AURORA-512 operations. Step 5 costs negligible time. In our assumption, three collisions, one for Cases 3, 4, and 5, will remain. Step 6 costs the same complexity as Step 4, which is 2^{257} AURORA-512 operations, and this is repeated three times in the worst case due to Step 7. Therefore, the time complexity for Steps 6 and 7 is $3 \cdot 2^{257}$ AURORA-512 operations. Finally, the total time complexity is 2^{257} AURORA-512 operations for Step 4 and $3 \cdot 2^{257}$ AURORA-512 operations for Step 6, which is 2^{259} AURORA-512 operations.

This attack can be easily carried out if we have a large amount of memory. Moreover, if we apply the memoryless collision search [15] for Step 2, all Steps can be carried out with a negligible amount of memory. To apply the memoryless collision search, we use the HMAC values obtained from the oracle as the next query. Therefore, Step 1 becomes adaptive. The memoryless collision search of our attack requires a message space of 512 bits[1]. Hence, we use 2-block messages as queries. Due to the increment of the message block, at Step 2, a message pair has six possibilities to be a collision. However, since this collision is filtered out at Step 3 with two additional queries, this does not impact the total attack complexity.

5.2 Universal Forgery on HMAC-AURORA-384

Inner Key Recovery Attack. AURORA-384 supports HMAC for a 384-bit MAC length. The structure for processing a 1-block message in HMAC-AURORA-384 is illustrated in Fig. 5.

The inner-key recovery procedure for HMAC-AURORA-384 is almost the same as that of HMAC-AURORA-512. For HMAC-AURORA-384, at Step 2 of the attack procedure, a pair of messages has the following six possibilities to be a collision.

Case 1: H_2^Us collide and H_2^Ds collide.
Case 2: Case 1 does not occur and H_2^*s collide.
Case 3: Case 1 and 2 do not occur and H_2^Ts collide.

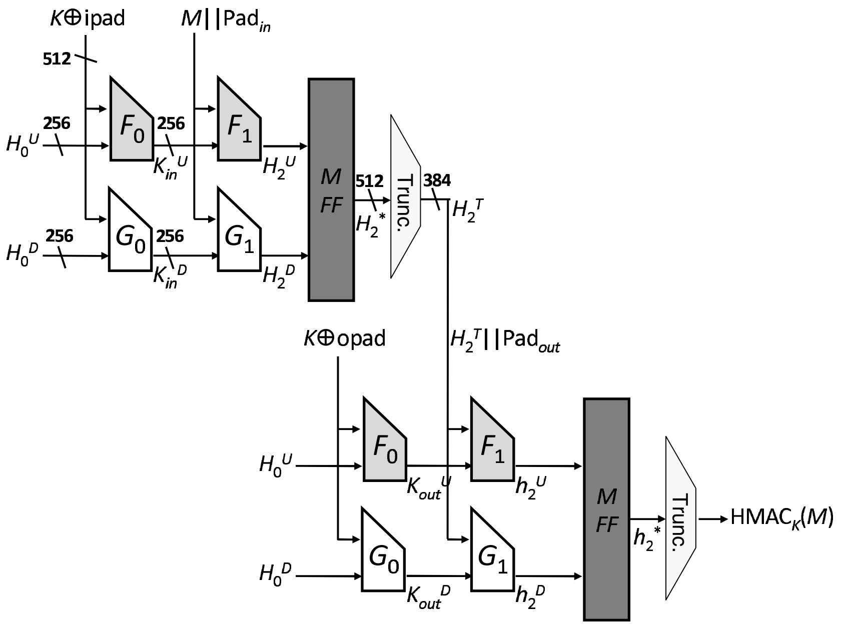

Fig. 5. Structure for processing a 1-block message in HMAC-AURORA-384

[1] If the message space is much smaller than 512 bits, for example 447 bits, the randomness for the memoryless collision search will collide after $2^{223.5}$ trials and we cannot make 2^{257} different queries.

Case 4: Case 1, 2, and 3 do not occur and h_2^Us collide and h_2^Ds collide.
Case 5: Case 1, 2, 3, and 4 do not occur and h_2^*s collide.
Case 6: Case 1, 2, 3, 4, and 5 do not occur and HMAC values collide.

Remember that H_2^T and HMAC values are 384 bits. By asking 2^{256} queries, we will obtain a single collision pair of Cases 1, 2, 4, and 5, and $2^{127}(= 2^{256 \cdot 2-1-384})$ collision pairs of Cases 3 and 6; therefore, we expect to obtain $2^{128}+4$ collisions in total. To recover the inner-key, we need to detect the collision pair of Case 1. At Step 3 of the attack procedure, this can be achieved by asking two additional queries $M^j \| \mathtt{pad}_{in} \| x$ and $M^{j'} \| \mathtt{pad}_{in} \| x$ for each collision pair $(M^j, M^{j'})$. The inner-key recovery procedure at Step 4 is exactly the same, in which we need a time complexity of 2^{257} AURORA-384 operations.

Finally the inner-key is recovered with $2^{257}+2\cdot(2^{128}+4) \approx 2^{257}$ queries and a time complexity of 2^{257} AURORA-384 operations. This attack can be performed with a negligible amount of memory.

Universal Forgery by Combining the Inner-Key Recovery and Second-Preimage Attacks. Although our attack cannot recover the outer-key, we can perform a universal forgery on HMAC-AURORA-384 by using the recovered inner-key and applying the second-preimage attack, which is explained in Section 4 or by Ferguson and Lucks [9].

In a universal forgery attack, the attacker has access to the oracle which returns $\mathrm{HMAC}_k(\cdot)$. For any given message M, our attack can find the value of $\mathrm{HMAC}_k(M)$ without asking M to the oracle. After revealing the inner-key, our attack requires one query and the same off-line complexity and memory as that of the second-preimage attack on AURORA-512, which are 2^{290} AURORA-512 operations and $2^{288} \times 512$ bits of memory in Section 4 of this paper and 2^{291} AURORA-512 operations and $2^{31.5}$ message blocks of memory in [9]. The attack procedure is as follows.

Target:
0. Receive M.

Preparation:
1. Recover the inner-key K_{in} with the attack explained in Section 5.2.

Universal forgery:
2. For the given M, find a second-preimage M' s.t. $\mathrm{AURORA}{-}384(K_{in}, M) = \mathrm{AURORA}{-}384(K_{in}, M')$ by using the second-preimage attack.
3. Ask M' to the oracle, and receive $\mathrm{HMAC}_k(M')$.
4. $\mathrm{HMAC}_k(M')$ is the HMAC value of M.

6 Discussion and Conclusions

The designers of AURORA showed proof for preimage resistance but did not show proof for second-preimage resistance. In fact, we showed that AURORA does not satisfy second-preimage resistance. Therefore, it would be useful to consider the differences of these two properties. We give some intuition by summarizing observations obtained from our attacks.

Assume that a hash function H is a sequence of several independent functions $H_1, H_2, \ldots, H_j$. The preimage resistance can be guaranteed if at least one of the functions is preimage resistant. However, this is not true for second-preimage resistance. To guarantee second-preimage resistance, all functions should be secure. The security bound of the second-preimage resistance is dependent on the weakest part of the hash function. AURORA can be regarded as consisting of two parts; the first 8-block H_1 and the MFF function H_2. Because the MFF function is secure, AURORA is secure on preimage resistance. However, because the first 8 blocks is not secure, AURORA does not satisfy second-preimage resistance.

From this observation, designing hash functions which are provably secure for second-preimage resistance seems harder than designing hash functions which are provably secure for preimage resistance.

7 Conclusion

We pointed out the weakness of the DMMD mode of operation. We first presented a collision attack on AURORA-512. We then presented a second-preimage attacks on AURORA-512 and -384, then explained that randomized hashing could also be attacked. Finally, we showed a full key-recovery attack on HMAC-AURORA-512 and a universal forgery attack on HMAC-AURORA-384.

Acknowledgements. I would like to thank Praveen Gauravaram (Department of Mathematics, DTU) for his comments on randomized hashing and thank Orr Dunkelman (Ecole normale superieure) for sharing his observations on the second-preimage attack in iterated compression function scenario. I also thank anonymous referees for helpful comments.

References

1. U.S. Department of Commerce, National Institute of Standards and Technology: Federal Register 72(212), Friday (November 2, 2007) Notices, `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`
2. Lucks, S.: A failure-friendly design principle for hash functions. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 474–494. Springer, Heidelberg (2005)
3. Iwata, T., Shibutani, K., Shirai, T., Moriai, S., Akishita, T.: AURORA: A Cryptographic Hash Algorithm Family. Initial submission version (October 31, 2008), AURORA home page, `http://www.sony.net/Products/cryptography/aurora/index.html`, NIST home page: `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`
4. U.S. Department of Commerce, National Institute of Standards and Technology: Randomized Hashing for Digital Signatures (NIST Special Publication 800-106) (February 2009), `http://csrc.nist.gov/publications/nistpubs/800-106/NIST-SP-800-106.pdf`

5. U.S. Department of Commerce, National Institute of Standards and Technology: The Keyed-Hash Message Authentication Code (HMAC) (Federal Information Processing Standards Publication 198) (July 2008), http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf
6. Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
7. Knudsen, L.R., Mendel, F., Rechberger, C., Thomsen, S.S.: Cryptanalysis on MDC-2. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 106–120. Springer, Heidelberg (2009)
8. International Organization for Standardization: ISO/IEC 10118-2:1994, Information technology – Security techniques – Hash-functions – Part 2: Hash-functions using an n-bit block cipher algorithm (1994) (Revised in 2000)
9. Ferguson, N., Lucks, S.: Attacks on AURORA-512 and the Double-Mix Merkle-Damgaard transform. Cryptology ePrint Archive, Report 2009/113, Ver. 20090311:092718 (2009), http://eprint.iacr.org/2009/113
10. Joux, A., Lucks, S.: Improved generic algorithms for 3-collisions. Cryptology ePrint Archive, Report 2009/305 (2009), http://eprint.iacr.org/2009/305
11. Gauravaram, P., Knudsen, L.R.: On randomizing hash functions to strengthen the security of digital signatures. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 88–105. Springer, Heidelberg (2009)
12. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. The Internet Engineering Task Force (1997), http://www.ietf.org/rfc/rfc2104.txt
13. Suzuki, K., Tonien, D., Kurosawa, K., Toyota, K.: Birthday paradox for multi-collisions. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A(1), 39–45 (2008)
14. Vaudenay, S.: A Classical Introduction to Cryptography: Applications for Communications Security. Springer, Heidelberg (2006)
15. Quisquater, J.J., Delescaille, J.P.: How easy is collision search. New results and applications to DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 408–413. Springer, Heidelberg (1990)

A Relationship between Our Work and [9]

This paper mainly presents three attacks on AURORA-512; a collision attack, a second-preimage attack and its application to randomized hashing, and a key-recovery attack on HMAC. Ferguson and Lucks independently found similar results on collision and second-preimage attacks [9].

Our work on collision and second-preimage attacks was motivated by the discussion by Ferguson, Lucks, and Iwata during a presentation on AURORA by Iwata at the first SHA-3 conference on 27th February 2009. We found our collision attack immediately after Iwata's presentation and informed it to the AURORA team that same day. On the other hand, Ferguson and Lucks mentioned that "at that point of time, (the concerns) had not been thought through" [9][Sec. 6]. Hence, we believe that we first found collision attack.

Regarding second-preimage resistance, we found the attack in a few days after the conference. Hence, the work is independent of Ferguson and Lucks. However, we heard they independently found the second-preimage attack during the SHA-3 conference before we found it.

From a technical viewpoint, the attack found by Ferguson and Lucks [9] and ours are in the same framework. However, we use 8-block multi-collisions, whereas [9] uses 9-block multi-collisions. Hence, the attack complexity of [9] is superior to ours in both collision and second-preimage attacks. The evaluation for the amount of memory is significantly different, in which our attack requires 2^{288}, whereas their attack [9] requires only $2^{31.5}$. This difference is based on the assumption on compression functions rather than attack techniques. Ferguson and Lucks assumes that compression functions are "balanced", whereas our attack also considers the case where the output distribution is very biased.