Attacks on AURORA-512 and the Double-MIX Merkle-Damgård Transform

Niels Ferguson[1] and Stefan Lucks[2]

[1] Microsoft Corp., niels@microsoft.com
[2] Bauhaus-Universität Weimar, Germany ⟨http://medsec.medien.uni-weimar.de/⟩

Abstract. We analyse the Double-Mix Merkle-Damgård construction (DMMD) used in the AURORA family of hash functions. We show that DMMD falls short of providing the expected level of security. Specifically, we are able to find 2nd pre-images for AURORA-512 in time 2^{291}, and collisions in time $2^{234.4}$. A limited-memory variant finds collisions in time 2^{249}.

1 Introduction and Overview

AURORA is a family of cryptographic hash functions submitted to the NIST SHA-3 hash function competition [2]. For $m \in \{224, 256, 384, 512\}$, AURORA-$m$ represents a hash function with m bits of output size, for which the authors make the following security claims:

- pre-image resistance of approximately m bits,
- 2nd pre-image resistance of approximately $m - k$ bits, for any message of length $\leq 2^k$, and
- collision resistance of approximately $m/2$ bits.

Internally, all the members of the AURORA family employ compression functions which map a 256-bit chaining value and a 512-bit message block to generate a new 256-bit chaining value. AURORA-224 and AURORA-256 use the classic Merkle-Damgård construction and are not affected by our attacks.

AURORA-512 maintains an internal state of 512 bit. To use the 256-bit compression function it employs a novel construction called *Double-MIX Merkle-Damgård Transform* (DMMD) to build a 512-bit compression function out of a 256-bit compression function. In general, DMMD turns an n-bit compression function in to a $2n$-bit compression function.

Unfortunately, DMMD fails to provide the level of security one would expect from a $2n$-bit hash function. In this paper we describe

- a collision attack in time $\ll 2^n$, and
- a 2nd pre-image attack for short pre-images in time $\ll 2^{2n}$.

In Section 3, we exploit these weaknesses to attack AURORA-512:

- We create collisions in time $2^{234.4} \ll 2^{256}$ (or time $2^{249} \ll 2^{256}$ with limited memory).
- We create 2nd pre-images in time 2^{291}.

In Section 4, we discuss the applicability of our attacks to two other members of the AURORA-family, which have the same DMMD structure. The 2nd pre-image attack applies to AURORA-384, and our collision attack can be used to find multi-collisions for AURORA-256M, which, unlike other members of the AURORA-family, has been claimed to resist multi-collision attacks.

2 Attacking the Double-MIX Merkle-Damgård (DMMD) Structure

The goal of DMMD is to define a $2n$-bit hash function using compression functions which use an n bit chaining value. We first describe the DMMD construction.

Let n the number of bits in a single chaining value, m the number of bits in a message block, and s be the number of message blocks processed between the extra mixing steps. AURORA-512 uses $n = 256$, $m = 512$, and $s = 8$.

DMMD uses $2s$ different compression functions that each map an n-bit chaining input and an m-bit message input to an n-bit chaining output.

$$f_0, \ldots, f_{s-1}, g_0, \ldots, g_{s-1} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n,$$

It also uses two additional mix functions that mix two n-bit chaining values to produce two new chaining values.

$$\mathrm{MF}, \mathrm{MFF} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n.$$

Our attacks are generic; we model all of these functions as independent random functions.

Given initial values $K_0, L_0 \in \{0,1\}^n$ and a padded message $M = (M_0, \ldots, M_{t-1}) \in \{0,1\}^m$, DMMD works as follows:

for i in $\{0, \ldots, t-1\}$ do
  $j := i \bmod s$
  $K'_{i+1} := f_j(K_i, M_i)$
  $L'_{i+1} := g_j(L_i, M_i)$
  if $j < s-1 \vee i = t-1$ then
    $(K_{i+1}, L_{i+1}) := (K'_{i+1}, L'_{i+1})$
  else
    $(K_{i+1}, L_{i+1}) := \mathrm{MF}(K'_{i+1}, L'_{i+1})$;
return $\mathrm{MFF}(K_t, L_t)$.

Essentially, DMMD runs two Merkle-Damgård chains in parallel and applies a mixing function MF to mix the chaining state of the two chains every s message blocks.

2.1 A 2nd pre-image attack

Assume a padded original message of at least $2s + 1$ blocks. After hashing $2s$ of these blocks, and before applying the MF operation, we have an internal state of the form $(K^*, L^*) = (K'_{2s}, L'_{2s})$. For a 2nd pre-image attack we need to find an arbitrary $(2s)$-block message fragment $(M_0, \ldots, M_{2s-1})$ which hashes to the same internal target (K^*, L^*). The only restriction is that these $2s$ blocks must be different from those in the original message.

For the actual attack, we fix some parameter r. Then we proceed as follows:

- Fix $M_0, \ldots, M_{s-2} \in \{0,1\}^m$ and compute K_{s-1} and L_{s-1} accordingly.
- Choose $K_s \in \{0,1\}^n$ and find r message blocks M^1_{s-1}, …, M^r_{s-1} which lead to the chosen value K_s. [Expected time: $r * 2^n$ calls to each f_{s-1}, g_{s-1}, and MF.]
- Choose $K_{s+1} \in \{0,1\}^n$ and find r message blocks M^1_s, …, M^r_s with $f_0(K_s, M^j_s) = K_{s+1}$. [Expected time: $r * 2^n$ calls to f_0.]
- …
- Choose $K_{2s-1} \in \{0,1\}^n$ and find r message blocks $M^1_{2s-2}, \ldots, M^r_{2s-2}$ with $f_{s-2}(K_{2s-2}, M^j_{2s-2}) = K_{2s-1}$. [Expected time: $r * 2^n$ calls to f_{s-2}.]
- Find r message blocks M^1_{2s-1}, …, M^r_{2s-1} with $f_{s-1}(K_{2s-1}, M^j_{2s-1}) = K^*$. [Expected time: $r * 2^n$ calls to f_{s-1}.]

– We now have r^{s+1} different messages, which all hash to the required internal half-state $K'_{2s} = K^*$. If

$$r^{s+1} \geq 2^n, \tag{1}$$

then we expect at least one of these messages to also hash to to the second half-state $L^* = L'_{2s}$.

Constructing the r^{s+1} messages that give the right K^* takes

$$(s+3)r2^n$$

calls to f_i, g_i, or MF. The last stage of the attack requires $(s+1)2^n$ calls and is insignificant. Memory requirements are that the attacker must store $r \cdot s$ message blocks.

We are now free to choose r as long as we satisfy condition 1. If $s = 1$ we can choose $r = 2^{n/2}$ leading to an attack in time $4 \cdot 2^{3n/2}$ which is much less than the generic 2nd pre-image attack time of $2 \cdot 2^{2n}$. (The factor 2 is because each message block is involved in two calls tot the compression functions.) For larger s we can shrink r and make the attack even faster.

We conclude that DMMD fails to provide the expected security against 2nd pre-image attacks, irrespective of the choice of s.

2.2 A collision attack

Our collision attack is based on finding local multi-collisions. In general this requires a lot of memory, but a limited-memory variation is described in Section 2.3.

As analysed in [4], the probability for an r-collision exceeds 0.5, if one chooses at least

$$\gamma(r,n) := \left(r! * 2^{(r-1)n}\right)^{1/r} \quad \text{random samples} \tag{2}$$

uniformly distributed from a set of size 2^n.

Like before, we fix some parameter r and proceed as follows

– Fix $M_0, \ldots, M_{s-2} \in \{0,1\}^m$ and compute K_{s-1} and L_{s-1} accordingly.
– Find an r-collision of messages $M^1_{s-1}, \ldots, M^r_{s-1}$ that all collide on the same K_s. [Expected time: $\gamma(r,n)$ calls to each f_{s-1}, g_{s-1}, and MF.]
– Find an r-collision $M^1_s, \ldots, M^r_s$ with $f_0(K_s, M^j_s) = K_{s+1}$ for some K_{s+1}. [Expected time: $\gamma(r,n)$ calls to f_0.]
– ...
– Find an r-collision $M^1_{2s-2}, \ldots, M^r_{2s-2}$ with $f_{s-2}(K_{2s-2}, M^j_{2s-2}) = K_{2s-1}$ for some K_{2s-1}. [Expected time: $\gamma(r,n)$ calls to f_{s-2}.]

At this point of the attack, we actually have a r^s-collision, i,e, we have r^s different message fragments

$$\left(M_0^{i_0}, M_1^{i_1}, \ldots, M_{s-2}^{i_{s-2}}\right),$$

which all happen to hash to the same internal half-state K_{2s-1}. But all of them hash to different half-states L_{2s-1} (or we would already have a collision).

We now try values for message block M_{2s-1}. We get the following two properties:

1. All messages $(M_0^{i_0}, M_1^{i_1}, \ldots, M_{s-2}^{i_{s-2}}, M_{2s-1})$ collide on K'_{2s}.
2. By the birthday paradox, the chance that two of these messages *also* collides on L'_{2s} is about

$$\frac{r^{2s-1}}{2^n}.$$

Thus, we need to try some $2^n/r^{2s-1}$ values for message block M_{2s-1} to find a collision on the entire internal state. Each try requires r^s calls to g_{s-1}. Thus, the second part of the attack has to make

$$\frac{2^n * r^s}{r^{2s-1}} = \frac{2^n}{r^{s-1}} \qquad \text{function calls.}$$

The complete time for the collision attack is thus the time for

$$(s+2) * \gamma(r,n) + \frac{2^n}{r^{s-1}} = (s+2) * \left(r! * 2^{(r-1)n}\right)^{1/r} + \frac{2^n}{r^{s-1}} \quad \text{function calls} \tag{3}$$

to f_i, g_i, or MF. Note that the memory for finding multi-collisions is large: The attacker has to store $\gamma(r,n)$ message blocks.

This attack works for any s. For $s = 1$ and $n \geq 80$ we can choose $r = 16$ and need fewer than the 2^{n+1} compression function calls that the generic attack would require for a collision. If s or n (or both) are larger, the advantage of our attack over the standard birthday attack improves.

2.3 A Memoryless Variant of the Collision Attack

Standard techniques, such as Floyd or Brent cycle finding, allow us to find ordinary 2-collisions with limited memory, while only marginally increasing the time (i.e., the number of function calls). Thus, if we set $r = 2$, we get memory-efficient 2^s collisions for K_{2s-1}. To exploit this for a collision on (K'_{2s}, L'_{2s}), we still need $2^n/2^{2-1} = 2^{n-s+1}$ function calls. Equation 3 with $r = 2$ thus shows that the time for the memoryless collision attack is equivalent to

$$(s+2) * 2^{1/2} * 2^{n/2} + 2^{n-s+1} \quad \text{function calls.}$$

For moderate s this is dominated by the last term, and still beats the generic attack of 2^{n+1} function calls even for $s = 1$.

3 Attacking AURORA-512

In this section, we show how to apply the attacks from the previous section to AURORA-512. This hash function has been designed according to the DMMD construction with $n = 256$ and $s = 8$. For the 2nd pre-image attack, we suggest to use $r = 365284285 \approx 2^{28.44}$, the smallest r satisfying $r^{s+1} = r^9 > 2^{256}$. Finding a 2nd pre-image for AURORA-512 then requires

$$(s+3) * r * 2^n \approx 12 * 2^{28.44} * 2^{256} \approx 2^{288} \qquad \text{function calls.}$$

The memory required for the 2nd pre-image attack is very small: the attacker only needs to store

$$s * r < 2^{31.5} \qquad \text{message blocks.}$$

Similarly, we can find collisions for Aurora-512. Recall Equation 3, describing the time for the collision attack by

$$(s+2) * \gamma(r,n) + \frac{2^n}{r^{s-1}} = 10 * \gamma(r,256) + \frac{2^{256}}{r^7} = 10 * \left(r! * 2^{(r-1)256}\right)^{1/r} + \frac{2^{256}}{r^7} \quad \text{function calls.}$$

To minimise this, we set $r := 9$. This allows us to find collisions for AURORA-512 in time

$$2^{232.9} + 2^{233.8} < 2^{234.5},$$

assuming sufficient memory for storing $2^{229.6}$ message blocks. Alternatively, we can set $r := 2$ to find collisions for AURORA-512 with limited memory in time

$$2^{256}/2^{8-1} = 2^{256-7} = 2^{249}.$$

4 Attacking AURORA-384 and AURORA-256M

Aurora-384 and AURORA-256M follow exactly the same internal structure as AURORA-512, except for truncating the output to 384 And 256 bit, respectively. Thus, our attacks apply to both of these hash functions as well.

For AURORA-384, the naive way to find collisions only requires time 2^{192}, so our collision attack is irrelevant. But finding short 2nd pre-images for AURORA-384 *should* take time close to 2^{384}, while our 2nd pre-image attack only needs time 2^{291}.

AURORA-256M is yet another variant of the AURORA family, with exactly the same structure as AURORA-512 and AURORA-384, except that AURORA-256M truncates the output down to 256 bit. [2] specifically claims resistance against multi-collisions for AURORA-256M. [2] does not quantify this claim, but ideally, finding a k-collision for an n-bit hash function should take time $(k! * 2^{(k-1)*256})^{1/k}$ [4]. As it turns out, we can apply our collision attack and combine it with with Joux' approach to turn a small number of collisions into huge multi-collisions.

Consider $k = 16 = 2^4$. To find a k-collision, we just need four collisions in a row. A single collision needs the time of less than $2^{234.5}$ function calls. Four such collisions need time $< 4 * 2^{234.5} = 2^{236.5}$. With this amount of work, we can generate a 16-collision. But for an ideal hash function, finding even a 12-collision should require time $(12! * 2^{11*256})^{1/12} \approx 2^{237} > 2^{236.5}$. So for any $k \geq 12$, we can find k-collisions faster than the generic attack.

The memoryless variant of our attack can also be used. Finding a 31-collision for an ideal hash function would need time $(31! * 2^{30*256})^{1/31} \approx 2^{251.38}$. And, as far as we know, that attack seems to require a huge amount of memory. But for AURORA-256M, our attack requires finding five collisions in a row allows us to generate a 32-collision in time $5 * 2^{249} \approx 2^{251.32} < 2^{251.38}$.

5 DMMD compared to other approaches

The DMMD structure was designed to achieve the security of a $2n$-bit hash function while only using compression functions which take n-bits of chaining values. [2, Page 57] explicitly compares DMMD to other double-block length constructions, namely to Lucks' double-pipe hash [3] and Hirose's construction [1]. DMMD and Hirose's and Lucks' approach share the idea of using one or more compression functions or internal block ciphers that generate an n-bit output, i.e., one half of the new $2n$-bit chaining value with. To generate the full chaining value, one needs to call the compression function(s) twice.

Both Hirose's and Lucks' construction, however, take the full $2n$-bit chaining value as the input for each compression function call. DMMD is different: each compression function call only takes one half (n bit) of the old chaining value as the input. To avoid turning into a cascade of two independent Merkle-Damgård hash functions, DMMD employs the MIX function MF frequently. As it seems, a benefit of DMMD over Hirose's and Lucks' approach is an improvement in efficiency. For example, if we use any of AURORA's f_i or g_i functions for Lucks' double-pipe hash, each compression function call would need to take the full $2n$-bit chaining value, and thus could only process message blocks of size $n = 256$ bit, instead of $2n = 512$ bit. Neglecting the work for the MF functions, DMMD would appear twice as fast as the double-pipe hash.

As we have seen, however, the DMMD approach fails to provide the security one would expect. Even for $s = 1$, performing one MF operation for each message block, we could find collisions and 2nd pre-images faster than the generic attacks. And, if we set $s = 2$, i.e., if we perform a MF operation for every 2nd message block, we can find collisions twice as fast as, and using far less memory than, the generic attack. Furthermore, observe that for such small s, one could hardly neglect the time to evaluate the MF function. All in all, our results casts severe doubts on the soundness of the DMMD construction for hash functions.

6 Acknowledgements

We would like to thank the Aurora team for their comments and encouragement.

7 Conclusion

The DMMD construction used by AURORA-512 and AURORA-384 has inherent weaknesses. This leads to 2nd-preimage attacks on AURORA-512 and AURORA-384, as well as a collision attack on AURORA-512. We conclude that these two hash functions do not satisfy the security requirements that NIST specified for SHA-3.

References

1. S. Hirose. Some plausible constructions of double-block-length hash functions. FSE 2006, Springer LNCS 4047, pp. 210–225.
2. T. Iwata, K. Shibutani, T. Shirai, S. Moriai, T. Akishita. AURORA: A Cryptographic Hash Algorithm Family. Submission to NIST, October 31, 2008.
3. S. Lucks. A failure-friendly design principle for hash functions. Asiacrypt 2005 Springer LNCS 3788, pp. 474–494.
4. K. Suzuki, D. Tonien, K. Kurosawa, K. Toyota. Birthday Paradox for Multi-Collisions. ICISC 2006, Springer LNCS 4296, pp. 29–40.