# Cryptanalysis of Vortex

Jean-Philippe Aumasson[1], Orr Dunkelman[2], Florian Mendel[3], Christian Rechberger[3], and Søren S. Thomsen[4]

[1] FHNW, Windisch, Switzerland
[2] École Normale Supérieure, Paris, France
[3] IAIK, Graz University of Technology
[4] DTU Mathematics, Technical University of Denmark

**Abstract.** Vortex is a hash function presented at ISC'2008 and submitted to the NIST SHA-3 competition, after some modifications that aim to strengthen it. This paper describes several attacks (collision, second preimage, preimage, distinguishers) on both versions of Vortex. Our attacks exploit both properties of the operations "inside the box" (low number of rounds, non-surjectivity, etc.) and of the high-level structures.

## 1 Introduction

Vortex is the name of an AES-based hash function proposed by Gueron and Kounavis [4] at ISC'2008, and is also the name of the modified version [7] submitted to the NIST Hash Competition[5]. We call these two functions Vortex-0 and Vortex-1, respectively, and present attacks on both versions, making the latter unoptimal as the SHA-3. Table 1 summarizes our attacks.

The paper is structured as follows: §2 briefly introduces the hash functions Vortex-0 and Vortex-1; §3 shows that many digests *cannot* be produced by Vortex (both versions); §4 and §5 present collision attacks for Vortex-0 and Vortex-1, respectively; second preimage attacks on Vortex-1 are given in §6, and preimage attacks in §7. Eventually, §8 concludes.

## 2 Vortex-0 and Vortex-1

### 2.1 Vortex-0

Vortex-0 is a Merkle-Damgård iterated hash function with 256-bit chaining value and 256-bit digest. Given a 2×128-bit chaining value $A\|B$ and a 4×128-bit message block $W_0\|W_1\|W_2\|W_3$, the compression function of Vortex-0 sequentially computes

$$A\|B \leftarrow (A\|B) \oplus \mathsf{subblock}(A, B, W_0, W_1)$$
$$A\|B \leftarrow (A\|B) \oplus \mathsf{subblock}(A, B, W_2, W_3)$$

[5] See `http://www.nist.gov/hash-competition`.

**Table 1.** Summary of our results on Vortex-0 and Vortex-1 (256-bit digest).

| Target | Type | Time | Memory |
|---|---|---|---|
| Vortex | impossible images | n.a. | n.a. |
| Vortex | distinguisher | $2^{112}$ | negl. |
| Vortex-0 | collision | $2^{60}$ | negl. |
| Vortex-1 | pseudo-collision | $2^{64}$ | negl. |
| Vortex-1 | free-start collision | $2^{64}$ | negl. |
| Vortex-1 | collision | $2^{124.5}$ | $2^{124.5}$ |
| Vortex-1 | second-preimage$^\star$ | $2^{129}$ | negl. |
| Vortex-1 | second-preimage | $2^{192}$ | $2^{64}$ |
| Vortex-1 | preimage | $2^{195}$ | $2^{195}$ |

$^\star$: for a class of weak messages

and returns the new $A\|B$ as the new chaining value (or as the digest, if the message block is the last one). The function $\mathsf{subblock}(A, B, W_i, W_j)$ returns the 256-bit value

$$V\left(\mathsf{C}_{W_i}(A), \mathsf{C}_{W_j}(B)\right) \ .$$

The block cipher $\mathsf{C}$ is a reduced version of AES with three rounds, where a round (unlike in AES) is the sequence AddRoundKey, SubBytes, ShiftRows, and MixColumns, and with a simplified key schedule.

The merging function $V : \{0,1\}^{256} \mapsto \{0,1\}^{256}$ takes two 128-bit inputs $A$ and $B$, which are parsed as four 64-bit words as $A_1\|A_0 \leftarrow A$ and $B_1\|B_0 \leftarrow B$. The function $V$ updates these words as follows ("$\otimes$" denotes carryless multiplication, and addition is modulo $2^{64}$):

- $L_1\|L_0 \leftarrow A_1 \otimes B_0$
- $O_1\|O_0 \leftarrow A_0 \otimes B_1$
- $A_0 \leftarrow A_0 \oplus L_0$
- $A_1 \leftarrow A_1 \oplus O_1$
- $B_0 \leftarrow B_0 + O_0$
- $B_1 \leftarrow B_1 + L_1$

### 2.2 Vortex-1

Vortex-1 is similar to Vortex-0 but with a different compression function, which computes

$$A\|B \leftarrow \mathsf{subblock}(A, B, W_0, W_1)$$
$$A\|B \leftarrow \mathsf{subblock}(A, B, W_2, W_3)$$

Note that, compared to Vortex-0, this new version omits the feedforward of the chaining value $A\|B$. Furthermore, $\mathsf{subblock}(A, B, W_i, W_j)$ now computes

$$A\|B \leftarrow V(\mathsf{C}_A(W_i) \oplus W_i, \mathsf{C}_B(W_i) \oplus W_i)$$
$$A\|B \leftarrow V(\mathsf{C}_A(W_j) \oplus W_j, \mathsf{C}_B(W_j) \oplus W_j) \ ,$$

where $A$, $B$, $W_i$, and $W_j$ are 128-bit words (see also Fig. 1). The AES-like cipher C still makes three rounds, and the $V$ function is the same as in Vortex-0. Note that the compression function of Vortex-1 is similar to MDC-2 [9], except that the final transform $V$ is not a permutation (see §3).

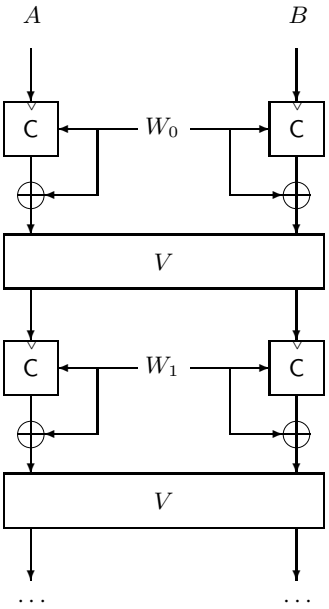


**Fig. 1.** Schematical view of Vortex-1's computation of a message digest (a hatch marks the key input).

Vortex-1's iteration mode differs from the classical Merkle-Damgård: the last message block is 256-bit, instead of 512-bit for the previous blocks, and is processed differently—as in the "Enveloped MD" mode [3] (EMD). A detailed description of this mode is not necessary to the understanding of (most of) our attacks.

A 512-bit version of Vortex-1 is described in [7]; instead of 128-bit Rijndael rounds, 512-bit Vortex-1 uses 256-bit Rijndael rounds. The merging function $V$ is similar but with words which are twice as large, and the message blocks are twice as large as well. The attacks in this paper are mainly described on the 256-bit version, but apply to the 512-bit version as well.

## 3 On impossible images

We show that both versions of Vortex have *impossible images*. That is, the range of Vortex-0 and Vortex-1 does not span the whole space of $\{0,1\}^{256}$. This obser-

vation allows slightly faster preimage and collision search, and can be used to mount distinguishers on PRF's based on Vortex (e.g. HMAC [2]).

Both Vortex-0 and Vortex-1 use the $V$ function after each evaluation of C. In particular, their final output is an output of $V$. But $V$ is a non-surjective function, hence there exist impossible images by the Vortex hash functions. Experiments on reduced versions suggest that $V$ behaves more like a random function than like a random permutation: for example, with 12-bit $A$ and $B$, about 66% of the outputs are reachable (against $1 - 1/e \approx 63\%$ for a random function[6]). It appears that the longer $A$ and $B$, the closer $V$ is to a random permutation.

In the remainder we denote $\mathcal{R}_V$ the range of $V$, and thus have $\mathcal{R}_V \subsetneq \{0,1\}^{256}$.

### 3.1 Multicollisions for $V$

Recall that a *multicollision* for a hash function is a set of distinct messages that map to the same digest; when there are $r$ messages, we talk of an $r$-collision (a collision is thus a 2-collision).

We present a simple method to find multicollisions over $V$: set $A_1 = B_1 = 1$, and choose a $A_0$ and a $B_0$ that have not bit "1" at a same position (for example, $A_0 = \texttt{FF00}\ldots\texttt{00}$ and $B_0 = \texttt{00FF}\ldots$). The $V$ function then sets:

- $L_1 \| L_0 \leftarrow 0 \| B_0$
- $O_1 \| O_0 \leftarrow 0 \| A_0$
- $A_0 \leftarrow A_0 \oplus L_0 = A_0 \oplus B_0$
- $A_1 \leftarrow A_1 \oplus O_1 = 1$
- $B_0 \leftarrow B_0 + O_0 = B_0 + A_0 = B_0 \oplus A_0$
- $B_1 \leftarrow B_1 + L_1 = 1$

Now one can modify the initial $A_0$ and $B_0$ such that $A_0 \oplus B_0$ remains unchanged (and still have no bit "1" at a same position), which does not affect the output. Note that this even allows multicollisions, since for a given pair $(A_0, B_0)$ there exists many colliding modified pairs.

One can easily derive an upper bound on $|\mathcal{R}_V|$ from the above technique: consider the case of a weight-32 $A_0$; there are $\binom{64}{32} \approx 2^{60.5}$ possible choices; then, given a $A_0$, there are at least $2^{32}$ suitable choices of $B_0$. Each of the $2^{92.5}$ pairs $(A_0, B_0)$ gives a $2^{32}$-collision, since all 1 bits of $A_0$ can be flipped without changing the output. This gives about $2^{60.5} \times 2^{32} \times 2^{32}$ collisions, which means that more than about $2^{124.5}$ elements of $\{0,1\}^{256}$ are impossible images, that is, $|\mathcal{R}_V| < 2^{256} - 2^{124.5}$. It follows that search for preimages and collisions is slightly faster than expected.

[6] Note that a random function $\{0,1\}^n \mapsto \{0,1\}^n$ has in average about 63% of its outputs reachable, but a random function $\{0,1\}^m \mapsto \{0,1\}^n$, $m \gg n$, has the space $\{0,1\}^n$ as range with high probability.

### 3.2 Inverting $V$

Given a random element $y$ of $\{0,1\}^{256}$, the best generic algorithm to decide whether $y$ lies in $\mathcal{R}_V$ is to try all the $2^{256}$ inputs. Below we describe an algorithm that solves this problem much faster, and finds a preimage of $y$ when $y \in \mathcal{R}_V$. We use the notations of §2, and writing "LOH" for "low-order half" and "HOH" for "high-order half":

- let $y = C\|D = C_1\|C_0\|D_1\|D_0$ be the given 256-bit value
- for each choice of $O_0$ and of the LOH of $A_1$:
    - from $O_0$ and $D_0$ compute $B_0$.
    - from $B_0$ and the LOH of $A_1$, compute the LOH of $L_0$
    - from $C_0$ and the LOH of $L_0$, compute the LOH of $A_0$
    - from $O_0$ and the LOH of $A_0$, compute the LOH of $B_1$ (by solving a linear system of 32 equations)
    - from $D_1$ and the LOH of $B_1$, compute the LOH of $L_1$
    - from $B_0$, the LOH of $A_1$, and the LOH of $L_1$, compute the HOH of $A_1$ (solve a linear system of 32 equations)
    - from $A_1$ and $C_1$, compute $O_1$
    - from $A_1$ and $B_0$, compute the HOH of $L_0$
    - from $C_0$ and the HOH of $L_0$, compute the HOH of $A_0$
    - from $O_0$ and the HOH of $A_0$, compute the HOH of $B_1$ (by solving a linear system of 32 equations)
- if no solution for $A$ and $B$ is found, return "$y \notin \mathcal{R}_V$", else return "$y \in \mathcal{R}_V$" and the preimage found

The running time of the algorithm is dominated by the solving of two sets of 32 GF(2) equations for each guess, i.e., in total finding $2^{97}$ solutions (we guess 96 bits) to a set of 32 equations in 32 unknowns over GF(2). A rough estimate yields complexity $32^3 \times 2^{97} = 2^{112}$. This algorithm can enjoy parallelism ( given $n$ CPUs the running time is divided by $n$).

We can now distinguish the output of Vortex from a random string by solving $2^{97}$ equations (because if the algorithm fails to find a preimage of the output, then the string was not produced by Vortex). Hence, given about ten outputs of Vortex, we can almost surely identify that the string was indeed produced by Vortex.

Note that similar claims may be made about a Merkle-Damgård hash function, when the last block is a full padding block. In such a case, the output space is indeed only 63% of the possible values. However, unlike the case of Vortex, this space changes when the padding changes (i.e., a different number of bits is hashed). Moreover, in the case of a Merkle-Damgård construction with a random function as the compression function, the adversary has to try all possible input chaining values before deducing that the output is indeed not in the range of the specific case of the function, which is clearly not the case for Vortex.

## 4 Collision attacks on Vortex-0

We present a collision attack on Vortex-0 that exploits structural properties of the subblock function, assuming that the main component (the block cipher) is perfect, i.e., we work in the *ideal cipher* model. We then prove that the actual C cipher in Vortex-0 is close enough (sic) to an ideal cipher to be vulnerable to our attack.

The attack goes as follows: given the IV $A\|B$, choose arbitrary $W_1, W_2, W_3$, and compute $\mathsf{C}_{W_0}(A)$ for $2^{64}$ distinct values of $W_0$; in the ideal cipher model, one thus gets $2^{64}$ random values, each uniformly distributed over $\{0,1\}^{128}$, hence a collision

$$\mathsf{C}_{W_0}(A) = \mathsf{C}_{W_0'}(A)$$

occurs with probability $1-1/e^2 \approx 0.39$ (by the birthday paradox), which directly gives a collision for the compression function. The cost of this attack is $2^{64}$ evaluations of C (which is equivalent to $2^{62}$ of the compression function of Vortex-0), whereas $2^{128}$ compressions was conjectured in [4] to be a minimum.

The attack would not work if the map key-to-ciphertext induced by C were significantly more injective than a random function. In Appendix A, we prove that, under reasonable assumptions, we have, for any $x \in \{0,1\}^{128}$

$$\Pr_{K,K'}[\mathsf{C}_K(x) = \mathsf{C}_{K'}(x)] \approx \frac{1}{2^{128}} \ .$$

More precisely, we show that for C with two rounds (denoted $\mathsf{C}^2$), instead of three, we have

$$\Pr_{K,K'}[\mathsf{C}^2_K(x) = \mathsf{C}^2_{K'}(x)] = \frac{2^{128}-2}{(2^{128}-1)^2} \approx \frac{1}{2^{128}} \ ,$$

which means that our collision attack works with the actual C.

## 5 Collision attacks on Vortex-1

### 5.1 Pseudo-collisions

We show how to find a pair of colliding messages for Vortex-1 with two distinct IV's, of the form $A\|B$ and $A'\|B$, respectively, for any fixed $B$ and random $A$, $A'$.

Observe that for an IV $A\|B$, the 128-bit $A$ is used only once in the compression function, to compute $\mathsf{C}_A(W_0)$. One can thus find a collision on the compression function by finding a collision for $\mathsf{C}_A(W_0) \oplus W_0$: fix $W_0$ and cycle through $2^{64}$ distinct $A$'s to find a collision with high probability. One can thus find collisions for two IV's $A\|B$ and $A'\|B$ in $2^{64}$ evaluations of C (instead of $2^{128}$ compressions ideally).

### 5.2 Free-start collisions

We show how to find a pair of colliding messages for Vortex-1 with any IV of the form $A\|B = A\|A$; which we call a *symmetric* IV.

To find a collision for Vortex-1 with a symmetric IV, it suffices to find $W_0, W_0'$ such that

$$\mathsf{C}_A(W_0) \oplus W_0 = \mathsf{C}_A(W_0') \oplus W_0'$$

to get two colliding messages with a same random IV. When the IV is fixed (and not symmetric) one can precompute a message block that leads to an IV $A\|A$ within $2^{128}$ trials, and then find collisions in $2^{64}$; for example, finding 10 000 000 collisions costs about $2^{128}$ trials with our attack, against $2^{151}$ with Joux's attack [5].

### 5.3 Full collisions

As mentioned, Vortex-1's construction is very similar to MDC-2 [9]; a recently discovered collision attack on MDC-2 [6] applies to Vortex-1 as well. In the following, $n$ denotes the size of C's block (128 and 256 for the 256- and 512-bit versions of Vortex-1, respectively).

We introduce the notation $g(h\|\tilde{h}, W_i)$ to denote, in subblock, the result of the operations

1. $a \leftarrow \mathsf{C}_h(W_i) \oplus W_i$
2. $\tilde{a} \leftarrow \mathsf{C}_{\tilde{h}}(W_i) \oplus W_i$
3. $h\|\tilde{h} \leftarrow V(a\|\tilde{a})$.

We write $\gamma(x, y) = \mathsf{C}_x(y) \oplus y$. The function $g$ can then be written:

$$g(h\|\tilde{h}, m) = V\left(\gamma(h, m)\|\gamma(\tilde{h}, m)\right).$$

Let $A\|\tilde{A} = g(h\|\tilde{h}, m)$, where $|A| = |\tilde{A}| = n$. The idea of the collision attack is to first find (by brute force) an $r$-collision in the variable $A$, resulting in $r$ different values $\tilde{A}_1, \ldots, \tilde{A}_r$ of $\tilde{A}$. The $n$-bit message sub-blocks producing the $r$-collision are denoted by $W_0^i$, $i = 1, \ldots, r$. Then, one finds a message sub-block $W_1$ such that $\gamma(\tilde{A}_i, W_1) = \gamma(\tilde{A}_j, W_1)$ for some $i \neq j$. Since $A_i = A_j$, we know that $\gamma(A_i, W_1) = \gamma(A_j, W_1)$, and hence we have found a two-block collision.

More precisely, given an arbitrary chaining value $h\|\tilde{h}$, do as follows (see also Fig. 2).

1. Find $r$ different message blocks $W_0^i$, $i = 1, \ldots, r$, such that
$$A_i\|\tilde{A}_i = V(\gamma(h, W_0^i)\|\gamma(\tilde{h}, W_0^i)),$$
for $1 \leq i \leq r$, and $A_i = A_j$ for all $i, j$.
2. Choose an arbitrary message block $W_1$, and compute $\tilde{B}_i = \gamma(\tilde{A}_i, W_1)$ for all $i$, $1 \leq i \leq r$. If $\tilde{B}_i = \tilde{B}_j$ for some $i, j$, $i \neq j$, then the two messages $W_0^i\|W_1$ and $W_0^j\|W_1$, collide, since $\gamma(A_i, W_1) = \gamma(A_j, W_1)$. If no such pair exists, repeat this step with a different choice of $W_1$.

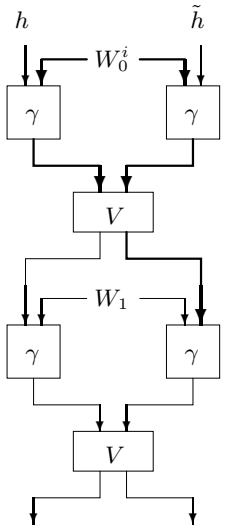


**Fig. 2.** The collision attack on Vortex-1. Thick lines mean that there are $r$ different values of this variable. Thin lines mean that there is only one.

Finding the $r$-collision in step 1 requires the equivalent of about

$$(r! \times 2^{(r-1)n})^{1/r}$$

evaluations of $g$ [11]. In the second step, $\gamma$ is evaluated $r$ times. Assuming that $V$ takes negligible time, the $r$ evaluations of $\gamma$ correspond to about $r/2$ applications of $g$. If evaluating $V$ is less efficient than assumed here, then the complexity of step 2 in terms of evaluations of $g$ is lower. For each choice of $W_1$ in step 2, the probability that a collision is found is about $\binom{r}{2} \times 2^{-n} = r(r-1)/2 \times 2^{-n}$. Hence, the expected time spent in step 2 is about $2^n/(r-1)$.

For the optimal $r$ (14), our attack on 256-bit Vortex-1 runs in $2^{124.5}$, and requires as much memory. For the 512-bit version, the optimal $r$ is 24, and the attack runs in $2^{251.7}$.

## 6 Second-preimage attacks on Vortex-1

### 6.1 Second preimages for weak messages

We now show how to find second preimages of messages that produce a symmetric chaining value (that is, of the form $A\|A$) during the digest computation. A key observation is that if $A = B$ and is of the form $(x\|0)$ or $(0\|y)$, then $V(A, B)$ maintains the equality of $A$ and $B$.

The attack works as follows: Given a message that produces the chain value $\tilde{A}\|\tilde{A}$, find a message that leads to a symmetric chaining value $A\|B = A\|A$. Then find message blocks that preserve the symmetry and that eventually give

$A = \tilde{A}$ (after as many blocks as in the original message). One then fills the new message with the blocks of the original message to get a second preimage of it.

Reaching the first symmetric chaining value costs about $2^{128}$, preserving the property for each step costs $2^{64}$, and the connection costs $2^{128}$. The total complexity is thus about $2^{129}$. Note that the computation of a message that leads to a symmetric chaining value is message-independent, hence can be precomputed.

This attack, however, applies with low probability to a random message of reasonable size: for a random $m$-bit message, there are about $\lfloor m/128 \rfloor - 1$ "chaining values" (note that we can connect inside the compression function as well), thus the probability that a random message is weak is approximately $2^{-128} \times \lfloor m/128 - 1 \rfloor$.

**Time-memory tradeoff variant.** We show that a variant of the above attack with precomputation $2^{135}$ and as much memory runs in only $2^{33}$ trials, using the tree-construction technique in [1].

Consider a set of *special chaining values*

$$\mathcal{S} = \left\{(x||0)\|(x||0), x \in \{0,1\}^{64}\right\} \cup \left\{(0||y)\|(0||y), y \in \{0,1\}^{64}\right\}.$$

As noted earlier, these chaining values are maintained under the $V(\cdot)$ transformation. The preprocessing phase is composed of three phases:

1. finding a message block $W$ such that $A\|A \leftarrow V(\mathsf{C}_{IV_0}(W) \oplus W, \mathsf{C}_{IV_1}(W) \oplus W)$ where the IV is treated as $IV = IV_0\|IV_1$
2. for each $B$, finding a special chaining value $s$ and a message word $W'$ such that
$$\mathsf{C}_s(W') \oplus W' = B \ ,$$
and store it in a table for each possible $B$
3. for each $s \in \mathcal{S}$, find two message blocks $W_1$ and $W_2$ such that
$$\mathsf{C}_s(W_1) \oplus W_1, \mathsf{C}_s(W_2) \oplus W_2 \in \mathcal{S}$$

It is easy to see that the first precomputation phase takes $2^{128}$ calls to $V(\mathsf{C}(\cdot)||\mathsf{C}(\cdot))$, and outputs one message word of 128 bits to memorize. The second phase can be done by picking $s \in \mathcal{S}$ and message words $W$ at random, until all outputs $B$ are covered. Assuming that the process is random (i.e., by picking $s$ and $W$ randomly and independently from previous calls) we can model the problem as the coupon collector (see e.g. [10, p.57]), which means that about $\ln(2^{128}) \cdot 2^{128} < 2^{135}$ computations are performed, and about $2^{128}$ memory cells are needed. Finally, the third phase can be done in means of exhaustive search for each special chaining value $s$, and we expect about $2^{64}$ computations for each of the $2^{65} - 1$ special values. The memory needed for the output of the last precomputation is about $2^{66}$ memory cells. With respect to the precomputation we note that it is entirely parallelizable, and can enjoy a speed up of a factor $x$ given $x$ processors.

The online phase of the attack is as follows. Given the weak message that has a chaining value $\tilde{A}\|\tilde{A}$, we find in the first table the special chaining value

$s \in \mathcal{S}$ and the message block $W$ that lead to $\tilde{A}\|\tilde{A}$. We then start from the $IV$, and using the precomputed message blocks, reach a state $s' \in \mathcal{S}$. Now we have to find a path of message blocks from $s'$ to $s$. This is done by randomly picking message blocks from $s'$ which maintain the chaining value in the special set, until the distance between the reached state $s''$ and $s$ is 65 message blocks.

To connect $s''$ and $s$ we use the tree-construction technique described in [1]: from $s''$ one constructs a tree with all the $2^{33}$ possible special chaining values reachable after 33 blocks; similarly, one constructs a tree with the (expected) $2^{32}$ possible chaining values that may arrive to $s$ after 32 blocks. As the size of the space is about $2^{65}$, we expect a collision, and a path from $s''$ to $s$.

The preprocessing of this phase costs $2^{128}$ trials, storage is $2^{64}$, and the online complexity is composed of performing a birthday on space of about $2^{65}$ values—which we expect to take about $2^{33}$ operations. So given about $2^{128}$ precomputation, $2^{128}$ storage that needs to be accessed once (store it on DVDs and put them in the closet), $2^{64}$ storage that is going to be accessed randomly, the online complexity of the attack is only $2^{33}$.

### 6.2 Second preimage attack

This attack is based on a partial meet-in-the-middle attack, and finds a second preimage for any message. The attack applies to messages of three partial blocks or more (i.e., 384 bits or more), and replaces the first three blocks. We denote the consecutive chaining values of these partial blocks by $IV = A_0\|B_0$, $A_1\|B_1$, $A_2\|B_2$, $A_3\|B_3$, etc., and write

$$W \oplus \mathsf{C}_W(A_2)||W \oplus \mathsf{C}_W(B_2) = X_2\|Y_2,$$

so that $A_3\|B_3 = V(X_2, Y_2)$.

The attack goes as follows:

1. For every $A_2 = 0||x$ (where $x$ is 64-bit), the attacker tries all $W$'s, until he finds $W_x$ such that $W_x \oplus \mathsf{C}_{W_x}(0||x) = X_2$. On average, there is one such $W_x$ for each $x$. The attacker stores pairs $(x, W_x)$ in a table.
2. The attacker takes $2^{192}$ two partial block messages, and computes for them $A_2||B_2$. If $A_2$ is not of the form $0||x$, the attacker discards the message; otherwise (i.e., $A_2 = 0\|y$), the attacker retrieves $W_y$ from the table, and checks whether $Y_2$ equals $\mathsf{C}_{W_y}(B_2) \oplus W_y$. If yes, the two partial blocks along with $W_y$, can replace the first three message blocks of the original message.

As we start with $2^{192}$ messages, we expect about $2^{128}$ messages which generate the required pattern for $A_2$. For each of these messages, the probability that indeed $Y_2 = \mathsf{C}_{W_y}(B_2) \oplus W_y$, is $2^{-128}$, and thus we expect one second preimage to be found.

We note that if multiple computing devices are available, they can be used efficiently. By picking the special structure of $A_2$, it is possible to "discard" many wrong trials, and access the memory very rarely. It is also possible to route the queries in the second phase between the various devices if each device

is allocated a different segment of the special $A_2$'s. Once one of the devices finds a message block which leads to a special $A_2$, it can send the message block to the computing device.

## 7 Preimage attacks on Vortex-1

A preimage attack on MDC-2 having complexity below the brute force complexity of $2^{2n}$ (where, again, $n$ is the size of the underlying block cipher) has been known for many years [8]. The attack has time complexity about $2^{3n/2}$. The attack applies to Vortex-1 as well, but is slightly more complicated due to the EMD extension, and due to $V$ not being efficiently invertible.

Consider first a second preimage attack, where the chaining value after processing the first $t-1$ message blocks of the first preimage is known. That is, we can ignore the EMD extension for now. Let this chaining value be $h_{\mathrm{T}}\|\tilde{h}_{\mathrm{T}}$. We may find a second preimage by the following meet-in-the-middle method, similar to the attack described in [8] ($\gamma$ is defined as in the collision attack described above).

1. Compute $Z\|\tilde{Z} = V^{-1}(h_{\mathrm{T}}\|\tilde{h}_{\mathrm{T}})$ by inverting $V$ as described above.
2. Choose $W_3$ arbitrarily and compute $\gamma(a, W_3)$ for many different values of $a$, until $\gamma(a, W_3) = Z$.
3. Likewise, compute $\gamma(\tilde{a}, W_3)$ for many different values of $\tilde{a}$, until $\gamma(\tilde{a}, W_3) = \tilde{Z}$.
4. Repeat $2^{n/2}$ times the above two steps with different choices of $W_3$. This yields $2^{n/2}$ values of $a\|\tilde{a}$ and $W_3$ such that $g(a\|\tilde{a}, W_3) = h_{\mathrm{T}}\|\tilde{h}_{\mathrm{T}}$.
5. Compute $g(g(g(h_0\|\tilde{h}_0, W_0), W_1), W_2)$ for about $2^{3n/2}$ different choices of $W_0, W_1, W_2$, until a triple is found such that $g(g(g(h_0\|\tilde{h}_0, W_0), W_1), W_2) = a\|\tilde{a}$ for some $a\|\tilde{a}$ computed in the previous step.

Here we produce a preimage $W_0\|W_1\|W_2\|W_3$ of $h_{\mathrm{T}}\|\tilde{h}_{\mathrm{T}}$, ignoring padding and the EMD extension. Step 1 takes expected time $2^{n-1}$, and Steps 2 and 3 can be done combined in time about $2^n$ each, which means that when repeated $2^{n/2}$ times, the time complexity is about $2^{3n/2}$. Step 5 takes expected time about $2^{3n/2}$ in terms of evaluations of $g$. Hence, the total time complexity is roughly $2^{3n/2+1}$. Taking length padding into account is not a problem in Step 5. One may simply partially hash a message of the appropriate length, and carry out Step 5 starting from the resulting intermediate hash value.

In a preimage attack we do not know the chaining value before the EMD extension. However, we can invert the hash function through the EMD extension as follows. Let the target image be $h_{\mathrm{T}}\|\tilde{h}_{\mathrm{T}}$. First, we note that the EMD extension can be seen as a short Merkle-Damgård iteration by itself, using a single 512-bit message block (in the 256-bit case), or equivalently, four 128-bit message blocks $W_0, W_1, W_2, W_3$. The initial value of this Merkle-Damgård iteration is $T\|\tilde{T}$; the first two sub-blocks, $W_0$ and $W_1$, form the chaining value from the processing of the first $t-1$ message blocks, and the last two sub-blocks, $W_2$ and $W_3$, contain at least 65 bits of padding. $W_2$ and $W_3$ are treated specially, since the subblock function is applied to them five times.

1. Choose a final message length, and construct $2^{n/2}$ different versions of $W_2 \| W_3$, e.g., varying bits in $W_2$ only (at least 65 bits in $W_3$ are fixed by the choice of message length and one bit of padding).
2. For each version of $W_2 \| W_3$, invert five times the function subblock using the same technique as in steps 1–3 above. Now we have $2^{n/2}$ values of $a \| \tilde{a}$ and $W_2 \| W_3$ such that $\mathsf{subblock}^5(a \| \tilde{a}, W_2, W_3) = h_{\mathrm{T}} \| \tilde{h}_{\mathrm{T}}$.
3. Compute $g(g(T \| \tilde{T}, W_0), W_1)$ for about $2^{3n/2}$ different choices of $W_0, W_1$, or until a pair is found such that $g(g(T \| \tilde{T}, W_0), W_1) = a \| \tilde{a}$ for some $a \| \tilde{a}$ computed in the previous step.

The attack yields a chaining value $W_0 \| W_1$ that may be used in place of $h_{\mathrm{T}} \| \tilde{h}_{\mathrm{T}}$ in the second preimage attack described above. Hence, one may now carry out this attack, keeping in mind that the message length has been fixed.

The time required to invert through the EMD extension is about $21 \times 2^{3n/2}$ (two inversions of each of $\gamma$ and $V$ are needed per application of the subblock function). The different phases of the attack can be scaled differently to reduce the time complexity by a factor about four. Of course, the attack also fixes the padded version of the $t$-th message block $W_2 \| W_3$.

Our attack runs in $2^{195}$ on the 256-bit version, and in $2^{387}$ on the 512-bit version, with as much memory required. By a different scaling of the number of times one needs to invert $g$ and the subblock function, and the number of times to compute values in the forward direction, less memory may be used at the cost of a longer running time.

## 8 Conclusion

We presented several attacks against the hash function Vortex as submitted to NIST, and against its original version. The new version of Vortex appears to be stronger than the original, but fails to provide ideal security against collision attacks and (second) preimage attacks, and suffers from impossible images, which slightly reduces the entropy of a digest. These results seem to make Vortex unsuitable as a new hash standard. None of our attacks, however, is practical.

## References


1. Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second preimage attacks on dithered hash functions. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *LNCS*, pages 270–288. Springer, 2008.
2. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *LNCS*, pages 1–15. Springer, 1996.
3. Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the EMD transform. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 299–314. Springer, 2006.

4. Shay Gueron and Michael E. Kounavis. Vortex: A new family of one-way hash functions based on AES rounds and carry-less multiplication. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC*, volume 5222 of *LNCS*, pages 331–340. Springer, 2008.
5. Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 306–316. Springer, 2004.
6. Lars R. Knudsen, Florian Mendel, Christian Rechberger, and Søren S. Thomsen. Cryptanalysis of MDC-2. In *EUROCRYPT*, LNCS. Springer, 2009. To appear.
7. Michael Kounavis and Shay Gueron. Vortex: A new family of one way hash functions based on Rijndael rounds and carry-less multiplication. Submission to NIST, 2008. `http://eprint.iacr.org/2008/464.pdf`.
8. Xuejia Lai and James L. Massey. Hash function based on block ciphers. In Rainer A. Rueppel, editor, *EUROCRYPT*, volume 658 of *LNCS*, pages 55–70. Springer, 1992.
9. Carl H. Meyer and Michael Schilling. Secure program load with manipulation detection code. In *SECURICOM 88*, pages 111–130, 1988.
10. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
11. Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyota. Birthday paradox for multi-collisions. In Min Surp Rhee and Byoungcheon Lee, editors, *ICISC*, volume 4296 of *LNCS*, pages 29–40. Springer, 2006.

## A Proof that the collision attack on Vortex-0 works

In Vortex-0 a *round* consists of the AddRoundKey operation (which xors the 128-bit round key with the 128-bit state), followed by a permutation defined by the sequence SubBytes, ShiftRows, and MixColumns. The key schedule of C is much simpler than that of Rijndael; given a 128-bit key $K$, it computes the 128-bit rounds keys

$$\begin{aligned} RK_1 &\leftarrow \pi_1(K) \\ RK_2 &\leftarrow \pi_2(RK_1) \\ RK_3 &\leftarrow \pi_3(RK_2) \end{aligned}$$

where the $\pi_i$'s are permutations defined by S-boxes, bit permutations and addition with constants. We denote $RK_i^K$ a round key derived from $K$.

Denote $\Pi_i^K$ the permutation corresponding to the $i$-th round of C; $\Pi_i$ depends of the $RK_i$ derived from $K$. Observe that for any state $x$ and any distinct keys $K, K'$, we have $K \oplus x \neq K' \oplus x$, therefore for any $x$

$$\Pi_i^K(x) \neq \Pi_i^{K'}(x).$$

In other words, a 1-round C mapping a key to a ciphertext, for any fixed plaintext, is a *permutation*. In the following we show that for 2 rounds it is *not* a permutation.

From the above observation, we have, for any $K \neq K'$, and for any $x_1, x_2$,

$$\Pi_1^K(x_1) \neq \Pi_1^{K'}(x_1)$$
$$\Pi_2^K(x_2) \neq \Pi_2^{K'}(x_2).$$

We show that, however, the probability over $K, K', x$ that

$$\Pi_2^K \circ \Pi_1^K(x) = \Pi_2^{K'} \circ \Pi_1^{K'}(x)$$

is nonzero, and is even close to what one would expect if $\Pi_2 \circ \Pi_1$ were a random permutation; in the latter, for clarity, we write $\Pi_i = \Pi_i^K$, $\Pi_i' = \Pi_i^{K'}$, and $\Pi = \{\Pi_1, \Pi_2\}$, $\Pi' = \{\Pi_1', \Pi_2'\}$. Recall that $\Pi_i$, $\Pi_i'$ are permutations such that: $\nexists x, \Pi_i(x) = \Pi_i'(x)$, for $i = 1, 2$.

We now compute the probability of a collision after two rounds. First, observe that

$$\Pr_{\Pi,\Pi',x}[\Pi_1 \circ \Pi_2(x) = \Pi_1' \circ \Pi_2'(x)] = \Pr_{y \neq y', \Pi_1, \Pi_1'}[\Pi_1(y) = \Pi_1'(y')] \ .$$

The probability holds over random *distinct* 128-bit $y$ and $y'$. We have (with $N = 2^{128}$):

$$\begin{aligned}
\Pr_{y \neq y', \Pi_1, \Pi_1'}[\Pi_1(y) = \Pi_1'(y')] &= \frac{1}{N} \sum_{y=0}^{N-1} \Pr_{y' \neq y, \Pi_1, \Pi_1'}[\Pi_1(y) = \Pi_1'(y')] \\
&= \frac{1}{N} \sum_{y=0}^{N-1} \frac{1}{N-1} \sum_{\Pi_1(y)=0, \Pi_1(y) \neq y}^{N-1} \Pr[\Pi_1(y) = \Pi_1'(y')] \\
&= \frac{1}{N} \sum_{y=0}^{N-1} \frac{1}{N-1} \sum_{\Pi_1(y)=0, \Pi_1(y) \neq y}^{N-1} (0 + (N-2) \times \frac{1}{N-1}) \\
&= \frac{1}{N} \sum_{y=0}^{N-1} \frac{N-2}{(N-1)^2} \\
&= \frac{N-2}{(N-1)^2} = \frac{2^{128}-2}{(2^{128}-1)^2} \approx \frac{1}{2^{128}}.
\end{aligned}$$

The above result shows that the 2-round $\mathsf{C}$ seen as a key-to-ciphertext mapping, for any fixed plaintext, has a distribution close to that of a random function. With three rounds, the distribution is much closer to that of a random function. Therefore, the birthday paradox is applicable, and so our attack works on the real Vortex-0 algorithm.