Security Enhancement of the Vortex Family of Hash Functions

Shay Gueron[1,2] and Michael Kounavis[3]
[1]Mobility Group, Intel Corporation
[2]University of Haifa, Israel
[3]Intel Labs, Circuits and Systems Research

February 2009, revised May 2009

Abstract

Vortex is a new family of one-way hash functions which has been submitted to the NIST SHA-3 competition. Its design is based on using the Rijndael block cipher round as a building block, and using a multiplication-based merging function to support fast mixing in a small number of steps. Vortex is designed to be a fast hash function, when running on a processor that has AES acceleration and has a proven collision resistance [2]. Several attacks on Vortex have been recently published [3, 4, 5, 6] exploiting some structural properties of its design, as presented in the version submitted to the SHA-3 competition. These are mainly first and second preimage attacks with time complexity below the ideal, as well as attempts to distinguish the Vortex output from random. In this paper we study the root-cause of the attacks and propose few amendments to the Vortex structure, which eliminate the attacks without affecting its collision resistance and performance.

1 Introduction

Vortex [1, 2] is a family of one way hash functions that can produce message digests of 224, 256, 384 and 512 bits. It uses the Rijndael round [10] and Galois Field or integer multiplication as building blocks. By using well known cryptographic primitives, Vortex achieves faster diffusion with a small number of steps, than algorithms that use primitives such as shift, rotate ADD or XOR operations. The Vortex family is expected to perform very efficiently on processors that have hardware acceleration for AES and carry-less multiplication. Vortex is collision resistant, as shown in [2]: the number of queries required for finding a collision with probability greater or equal to 0.5, under an ideal block cipher approximation of Vortex 256, is at least $1.18 \cdot 2^{122.5}$, if the attacker uses randomly selected message words.

Several attacks on Vortex have been published [3, 4, 5, 6]. References [3, 4] describe a multi-collision attack that works with complexity $2^{124.5}$ on Vortex 256 and $2^{251.7}$ on Vortex 512. This attack reduces the collision resistance of Vortex by a few exponent bits and does not invalidate the main security result of reference [2]. References [3, 4] also describe a first preimage attack with time complexity $O(2^{0.75N})$ (the ideal is $O(2^N)$), where N is the digest length in bits. The storage complexity of this attack is $O(2^{0.25N})$, so the product of the storage and time complexity remains $O(2^N)$. In fact, this attack is not specific to Vortex. It is associated with the MDC-2 mode of operation [7] where two parallel block ciphers are feeding their outputs into a merging function. Vortex inherits this structure from MDC-2. Reference [4] describes a generic second preimage attack with time complexity $O(2^{0.75N})$ and storage complexity $O(2^{0.25N})$. For this attack the product of the storage and time complexity also

remains ideal. Reference [4] describes a specialized second preimage attack for Vortex 256 that works for a class of weak messages. There are 2^{64} images associated with weak messages out of a total of 2^{256} images. This attack has a preprocessing phase with 2^{128} time complexity and storage requirement of 2^{128} state values plus an on-line phase with 2^{33} time complexity. Reference [5] shows that the Vortex output has impossible images and proves the existence of 2^{101} impossible images out of 2^{256} images. Reference [6] shows correlations in the output of Vortex, as some output bits are equal to each other with probability higher than 0.5.

In this paper we present a root-cause analysis of the published attacks and show that most of them result from a small number of structural properties of Vortex:

i Openness to inversion: The Vortex compression function can be inverted up to a point. This facilitates multi-collision attacks, second preimage attacks, as well as second preimage attacks for weak messages;

ii Use of a narrow pipe across the whole domain extension transform. This facilitates first preimage attacks;

iii Symmetry in the merging function. This leads to the existence of impossible images and output bit correlation, and to preimage attacks for weak messages;

In response to these published attacks, we propose few simple amendments to the Vortex structure, and show that these amendments address all of the attacks without affecting the collision resistance and performance.

The first amendment is adding a feed-forward XOR inside and around the Vortex sub-block transformation. A feed-forward XOR around the Vortex sub-block forces an adversary to guess the message word processed by the sub-block. Such guessing can be hard, potentially increasing the complexity of meet-in-the-middle attacks or making them impossible. This mitigates all collision and second preimage attacks. We call this amended version of Vortex "Vortex-M".

The second amendment is increasing the hash state (i.e., widen the pipe) only in the last block. Widening the pipe in this manner substantially increases the complexity of first preimage attacks based on the meet-in-the-middle principle. This is because an adversary needs to maintain much larger state and perform substantially more operations, even if a single wide pipe block is present in the hash chain. This addresses the first preimage attacks. We call this version of Vortex "Vortex-W".

The third amendment is skipping the final merging function transformation in the last Vortex sub-block. This amendment eliminates the correlations between output bits, and makes impossible images computationally harder to detect. We call this version of Vortex "Vortex-S". One can also consider variants of Vortex with more than one amendment. We call Vortex with all three amendments "Vortex+" (this addresses all published attacks).

The paper is structured as follows: In Section 2 we provide a brief description of the Vortex hash family. In Section 3 we discuss the published attacks. In Section 4 we present a root-cause analysis of these attacks. In Section 5 we discuss the amendments. Finally, in Section 6 we provide some concluding remarks.

2 Description of Vortex

The Vortex [1, 2] family of hashing algorithms is based on using Rijndael round primitives as building blocks and a multiplication-based merging function for performing binary mixing. Vortex also uses the Enveloped Merkle-Damgård [8] construction as its domain extension

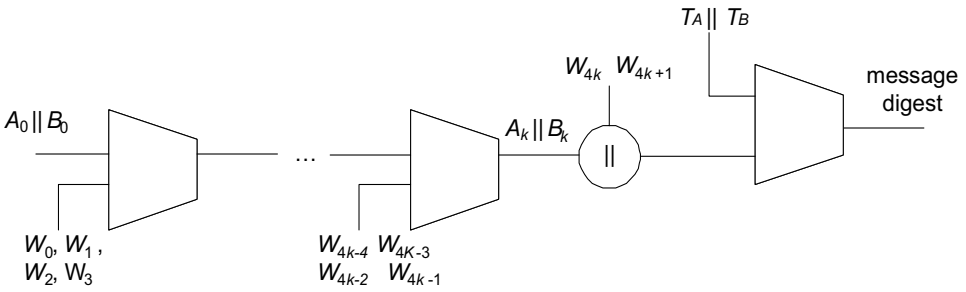


Figure 1: Enveloped Merkle-Damgård Construction

transform. Here, we discus the domain extension transform of Vortex, its compression function called 'Vortex block', and the tunable parameters the algorithm accepts as input. The complete description of Vortex is found in [2].

2.1 Domain Extension Transform

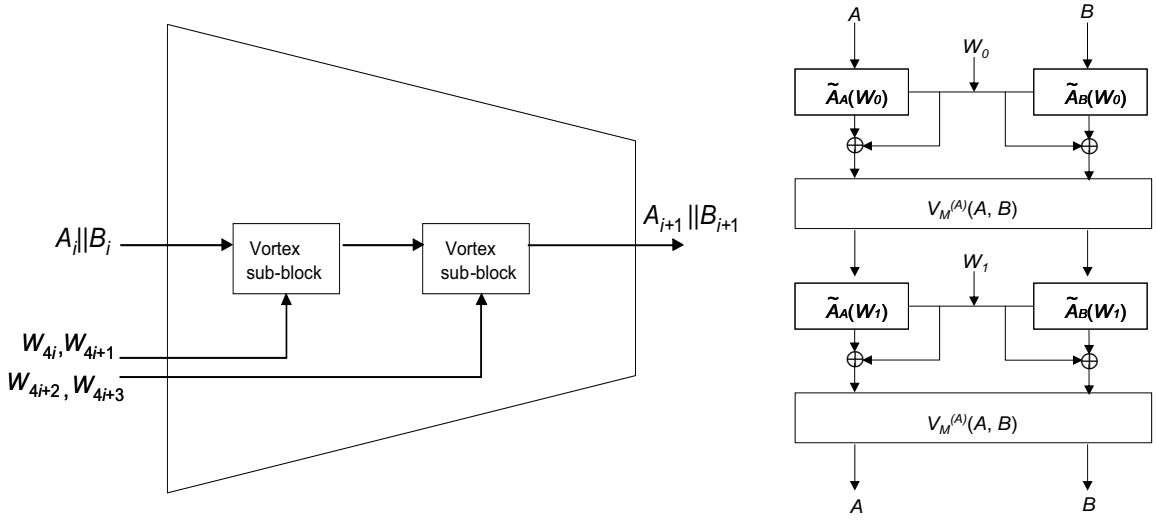
Vortex operates on a chaining variable resulting from the concatenation of two $N/2$-bit variables A and B initialized to $A_0||B_0$, where N is the digest size. Vortex also uses a tweak value consisting of the concatenation of variables T_A and T_B. T_A and T_B are $N/2$ bits long. To support collision resistance as well as pseudorandom function and pseudorandom oracle preservation, Vortex uses the Enveloped Merkle-Damgård construction [8] as its domain extension transform (see Figure 1).

The Enveloped Merkle-Damgård construction is very similar to the strengthened Merkle-Damgård [9] with one exception: the processing done on the last block differs from the processing done on other blocks. For the last block, the compression function uses the tweak value $T_A||T_B$ as a chaining variable and the concatenation of $[A_k : B_k]$, which is the current value of the hash state, and $[W_{4\cdot k} : W_{4\cdot k+1}]$ which are the last message words, as input block. For Vortex 256 and Vortex 512, the message digest resulting from the input stream is equal to the final value of the chaining variable. For Vortex 224 and Vortex 384, the message digest resulting from the input stream is equal to the 224 and 284 least significant bits of the chaining variable respectively.

2.2 Vortex Block

Vortex design is built upon the Vortex block algorithm, which is the compression function used as part of an Enveloped Merkle-Damgård construction. The Vortex block algorithm incorporates two repetitions of an algorithm called 'Vortex sub-block', as shown in Figure 2(a). The first repetition of Vortex sub-block accepts as input the chaining variable $A_i||B_i$ and two least significant input block words $W_{4\cdot i}, W_{4\cdot i+1}$. It returns an intermediate value for the chaining variable $A||B$. The second repetition of Vortex sub-block accepts as input the intermediate value of the chaining variable $A||B$ and two most significant input block words $W_{4\cdot i+2}, W_{4\cdot i+3}$. It returns an update on the chaining variable $A_{i+1}||B_{i+1}$. Pseudo-code for the Vortex block algorithm is given below:

Vortex Block $(A_i, B_i, W_{4\cdot i}, W_{4\cdot i+1}, W_{4\cdot i+2}, W_{4\cdot i+3})$
begin
$A||B \leftarrow$ **Vortex Sub-block**$(A_i, B_i, W_{4\cdot i}, W_{4\cdot i+1})$ // uses $W_{4\cdot i}, W_{4\cdot i+1}$
$A_{i+1}||B_{i+1} \leftarrow$ **Vortex Sub-block**$(A, B, W_{4\cdot i+2}, W_{4\cdot i+3})$ // uses $W_{4\cdot i+2}, W_{4\cdot i+3}$



(a) Vortex Block (b) Vortex Sub-block

Figure 2: The Compression Function of Vortex

return $A_{i+1}||B_{i+1}$
end

With the exception of the last sub-block (see below), the algorithm for processing a Vortex sub-block is the following:

Vortex Sub-block (A, B, W_0, W_1)
begin
;W_0 is the first word of the current sub-block to be processed
$A \leftarrow \tilde{A}_A(W_0) \oplus W_0$
$B \leftarrow \tilde{A}_B(W_0) \oplus W_0$ // Matyas-Meyer-Oseas transformation
$A||B \leftarrow V_M^{(A)}(A, B)$

;W_1 is the second word of the current sub-block to be processed
$A \leftarrow \tilde{A}_A(W_1) \oplus W_1$
$B \leftarrow \tilde{A}_B(W_1) \oplus W_1$
$A||B \leftarrow V_M^{(A)}(A, B)$
return $A||B$
end

The Vortex sub-block is built upon two functions: The transformation $\tilde{A}_K(x)$ called 'A-Rijndael', a block cipher based on Rijndael rounds, and the merging function $V_M^{(A)}$. There are four instances of the transformation $\tilde{A}_K(x)$ in the Vortex sub-block as shown in Figure 2(b). Each instance is wrapped using a Matyas-Meyer-Oseas structure to make the transformation non-reversible. The first two instances process input word W_0. The other two instances process the input word W_1. W_0 is the least significant word of the current sub-block to be processed. Instances of $\tilde{A}_K(x)$ that accept the same input word processes a different variable from among A, B. Each instance treats its input variable A or B as a key and its input word, which is one from W_0 or W_1 as plaintext, as it is the norm in the a Matyas-Meyer-Oseas structure.

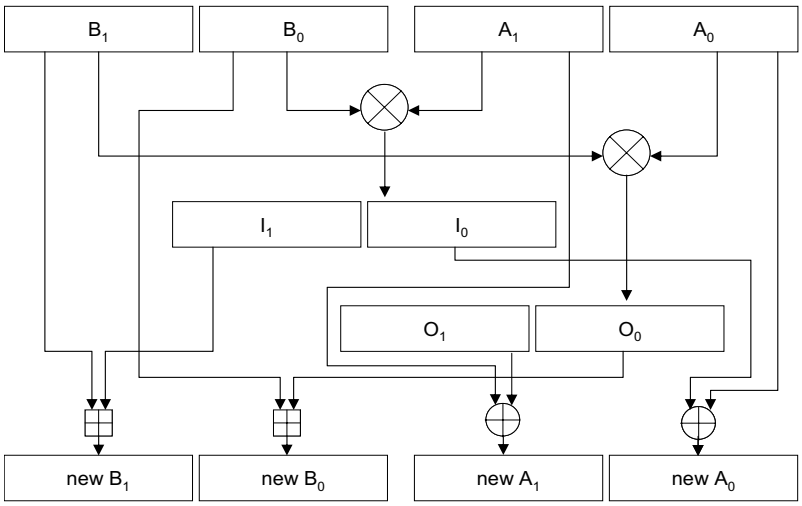


Figure 3: The Merging Function of Vortex

2.3 Merging Function

The merging function $V_M^{(A)}$ combines the outputs of the two instances of $\tilde{A}_K(x)$ into a new value of $A||B$. The merging function of Vortex is shown in Figure 3. It uses multiplication as a building block. Multiplication can be carry-less or integer. The multiplication type is determined by a tunable parameter M_T. If the multiplication type is "carry-less", $V_M^{(A)}$ operates as follows:

$V_M^{(A)}$ (A, B)
begin
    let $A = [A_1 : A_0]$ // A_1, A_0 are N/8 bit words
    let $B = [B_1 : B_0]$ // B_1, B_0 are N/8 bit words
    $O \leftarrow A_0 \otimes B_1$
    $I \leftarrow A_1 \otimes B_0$
    let $I = [I_1 : I_0]$ // I_1, I_0 are N/8 bit words
    let $O = [O_1 : O_0]$ // O_1, O_0 are N/8 bit words
    return $[B_1 \boxplus I_1 : B_0 \boxplus O_0 : A_1 \oplus O_1 : A_0 \oplus I_0]$
end

where by $\boxplus$ we addition modulo 2^{64}, and $\otimes$ we mean carry-less multiplication.

2.4 Last Vortex Sub-block

The algorithm for the last Vortex sub-block repeats the stages of Matyas-Meyer-Oseas transformations and merging several times (D_F) to increase the security of the hash:

Last Vortex Sub-block (A, B, W_0, W_1, D_F)
begin
    ;W_0 is the first word of the last sub-block to be processed
    $A \leftarrow \tilde{A}_A(W_0) \oplus W_0$
    $B \leftarrow \tilde{A}_B(W_0) \oplus W_0$
    $A||B \leftarrow V_M^{(A)}(A, B)$
    for $i \leftarrow 1$ to D_F **do**
        ;D_F is the degree of diffusion

;W_1 is the second word of the last sub-block to be processed
$A \leftarrow \tilde{A}_A(W_1) \oplus W_1$
$B \leftarrow \tilde{A}_B(W_1) \oplus W_1$
$A||B \leftarrow V_M^{(A)}(A, B)$
return $A||B$
end

2.5 The A-Rijndael Transformation

The A-Rijndael transformation $\tilde{A}_K(x)$ is a block cipher based on Rijndael rounds that encrypts x, which is $N/2$ bits long, using the key K which is also $N/2$ bits long. $\tilde{A}_K(x)$ uses a tunable number of Rijndael rounds which we symbolize as N_R. For $N/2 = 128$ rounds are as specified in AES, FIPS-197 [10]. A version of A-Rijndael is also defined, where the size of the cipher state is 256. Each Rijndael round $R()$ consists of an $SBox()$ substitution phase, a 'Shift Rows' transformation, a 'Mix Columns' transformation and a round key addition in $GF(2)$. The key schedule algorithm used by $\tilde{A}_K(x)$ is different from that of Rijndael. $\tilde{A}_K(x)$ uses a variable number N_R of $N/2$-bit wide *Rcon* values RC_1, RC_2,..., RC_{N_R} to derive N_R round keys RK_1, RK_2, ..., RK_{N_R} as follows:

$RK_1 \leftarrow \mathsf{Perm}(SBox(K \boxplus RC_1))$,
$RK_2 \leftarrow \mathsf{Perm}(SBox(RK_1 \boxplus RC_2))$,
...
$RK_{N_R} \leftarrow \mathsf{Perm}(SBox(RK_{N_R-1} \boxplus RC_{N_R}))$

where $\mathsf{Perm}()$ is a byte permutation and by $\boxplus$ we mean addition modulo 2^{64}. The $SBox()$ transformation in the key schedule is applied on $N/16$ bytes, i.e., $N/2$ bits (i.e., 128 bits or 16 bytes for Vortex 256 and 256 bits or 32 bytes for Vortex 512). The *Rcon* values are set to constant values.

3 Description of the Published Attacks

3.1 Collision Attacks

References [3, 4] describe a 'free start' collision attack of complexity $2^{N/4}$ with N being the digest size. The attack works as follows:

Attack A_1:
A_1, step$_1$: The adversary computes $2^{N/4}$ values $x = \tilde{A}_A(W_0) \oplus W_0$ for fixed W_0 and $2^{N/4}$ random choices for A. These values are stored in a list L_A.
A_1, step$_2$: The adversary computes another set of $2^{N/4}$ values $y = \tilde{A}_B(W_0) \oplus W_0$ for the same fixed W_0 and $2^{N/4}$ random choices for B. These values are stored in a second list L_B.
A_1, step$_3$: In the third step, the adversary computes pairs (x, y) such that $(x||y) = (\tilde{A}_A(W_1) \oplus W_1 ||\tilde{A}_B(W_1) \oplus W_1)$ for arbitrary values of A and B and a different input message block W_1.

Due to the birthday paradox, the number of efforts needed to find a match in the list L_A for x is $2^{N/4}$. Similarly, the number of efforts for finding a match in the list L_B is $2^{N/4}$. Hence the overall complexity of this attack is $2^{N/4}$.

This free-start collision attack works for appropriately selected chaining variable words A and B. However, in a realistic situation, selecting A and B cannot be done in a straightfor-

ward manner because, typically, the adversary is in control of the message words W_0, W_1 but not the chaining variable words A and B. Therefore, this attack scenario cannot be easily applied to Vortex. To address this, references [3, 4] describe another attack based on creating multi-collisions, which works as follows. Let $L_{N/2}()$ and $M_{N/2}()$ denote the $N/2$ least and most significant bits of an N-bit quantity, respectively. We denote also the combination of Matyas-Meyer-Oseas transformations and merging by:

$$Q(A, B, W) = V_M^{(A)}(\tilde{A}_A(W) \oplus W, \tilde{A}_B(W) \oplus W) \tag{3.1}$$

(or $Q(A||B, W)$ in other instances).

Attack A_2:
A_2, step$_1$: The adversary computes R message blocks $W_0^{(0)}, W_0^{(1)}, \ldots, W_0^{(R-1)}$ such that $M_{N/2}(Q(A, B, W_0^{(i)})) = M_{N/2}(Q(A, B, W_0^{(j)})), \forall i, j \in [0, R-1]$ for some A, B.
A_2, step$_2$: The adversary chooses an arbitrary message block W_1 and computes all quantities $\tilde{M}_i = \tilde{A}_{M^{(i)}}(W_1) \oplus W_1$ where $M^{(i)} = L_{N/2}(Q(A, B, W_0^{(i)}))$ and $0 \leqslant i \leqslant R-1$. These computations are done for the A, B variables used in step$_1$ of A_2.
A_2, step$_3$: The adversary checks if $\tilde{M}_i = \tilde{M}_j$ for some $i, j \in [0, R-1], i \neq j$. If this is the case, then the two messages $W_0^{(i)}||W_1$ and $W_0^{(j)}||W_1$ collide. Otherwise, step$_2$ of A_2 is repeated.

This attack works with complexity $2^{124.5}$ for Vortex 256 and $2^{251.7}$ for Vortex 512. In other words, it reduces the collision resistance of Vortex by only a few exponent bits, and does not invalidate the main security result proven in [2].

3.2 First Preimage Attacks

References [3, 4] describe a 'meet-in-the-middle' first preimage attack that is associated with time complexity $O(2^{0.75N})$ as opposed to the ideal $O(2^N)$, where N is the digest length in bits. This attack is associated with the generic structure of two parallel block ciphers stages feeding their outputs into a merging function. The attack works by creating a "special" set of inputs to the basic structure of Vortex (i.e., Matyas-Meyer-Oseas transforms followed by merging) that result in a given image. For Vortex 256, the time complexity of creating a set of size 2^{64} is 2^{192}. Once the set is created, a brute-force preimage attack can hit an element in the pre-computed set in 2^{192} efforts.

For simplicity, the attack is described here as if the structure used was simple Merkle-Damgård and the last Vortex sub-block was the same as the regular Vortex sub-block transformation (i.e., no padding and EMD applied). Let $A_F||B_F$ be a value coming out of the hash function which is known. References [3, 4] suggest that the merging function $V_M^{(A)}$ can be inverted with complexity $2^{N/2}$. This can be done by guessing $N/2$ bits of the internal state of the merging function and by solving the resulting system of equations.

Attack A_3:
A_3, step$_1$: The adversary computes a pair $A_I||B_I = (V_M^{(A)})^{-1}(A_F, B_F)$ by inverting the function $V_M^{(A)}$.
A_3, step$_2$: The adversary chooses a message word W_3 arbitrarily and computes $x = \tilde{A}_A(W_3) \oplus W_3$ for many different values of the chaining variable word A until $x = A_I$. The adversary also

computes $y = \tilde{A}_B(W_3) \oplus W_3$ for many different values of the chaining variable word B until $y = B_I$. This brute-force attack is accomplished with complexity $2^{N/2}$.
$\underline{\text{A}_3, \text{step}_3}$: In step three, the previous step is repeated another $2^{N/4} - 1$ times. In the end of this step the adversary has a list of $2^{N/4}$ triplets (A, B, W_3) for which: $V_M^{(A)}(\tilde{A}_A(W_3) \oplus W_3, \tilde{A}_B(W_3) \oplus W_3) = A_F || B_F$. The total complexity for creating this list of triplets is $2^{3N/4}$.
$\underline{\text{A}_3, \text{step}_4}$: In the last step of this attack the adversary computes the output of three successive sequences of Matyas-Meyer-Oseas transformations and merging for about $2^{3N/4}$ choices of message words W_0, W_1 and W_2. The computation continues until a triplet is found that results in a hash value $A||B$ included in the list computed in the previous step.

The overall time complexity of the attack is $O(2^{0.75N})$, which is smaller than the ideal $O(2^N)$. The attack is also associated with storage complexity of $2^{0.25N}$. The attack can be extended to cover the final padding and EMD extensions [3, 4].

3.3 Second Preimage Attacks

Vortex is vulnerable to two second preimage attacks described in [4]. The first attack (A_4) is a meet-in-the-middle generic attack and applies to messages of three $N/2$ bit-blocks or more. We denote (as in [4]) the chaining variable words coming out of successive invocations of the function $Q(A, B, W)$ as $A_0||B_0$, $A_1||B_1$, $A_2||B_2$, $A_3||B_3$, where $Q()$ is defined by equation 3.1. We also denote:

$$X_2||Y_2 = \tilde{A}_{A_2}(W) \oplus W || \tilde{A}_{B_2}(W) \oplus W \tag{3.2}$$

for some W, and $A_3||B_3 = V_M^{(A)}(X_2, Y_2)$. The attack works as follows:

$\underline{\text{Attack A}_4}$:
$\underline{\text{A}_4, \text{step}_1}$: The adversary tries all $N/2$ message words W until he finds a word W_x such that $X_2 = W_x \oplus \tilde{A}_{(0||x)}(W_x)$ for some 64-bit value $x \in [0, 2^{64} - 1]$. The complexity of this step is 2^{128}. On average there is one value W_x for every $x \in [0, 2^{64} - 1]$. The adversary stores the pair (x, W_x) in a table.
$\underline{\text{A}_4, \text{step}_2}$: The adversary repeats step_1 of A_4 $2^{64} - 1$ times for all remaining values of x and fills the table with (x, W_x) pairs. In the end the table should have 2^{64} entries. The complexity of this step is 2^{192}.
$\underline{\text{A}_4, \text{step}_3}$: The adversary tries 2^{192} random messages and computes the resulting chaining variable words $A_2||B_2$ for each of them. If is not of the form $0||y$ for some $y \in [0, 2^{64} - 1]$, then the adversary discards the message. On the other hand, if A_2 is of the form $0||y$ then the adversary retrieves the message word W_y from the table computed in step_2 of A_4 and checks whether $Y_2 = \tilde{A}_{B_2}(W_y) \oplus W_y$. If this is true, then the message resulting in $0||y||B_2$ together with W_y can replace three message words of the original message.

Reference [4] shows that at least one preimage is found following steps A_4, step_1 through A_4, step_3. The overall time complexity of this attack is 2^{192} and its storage requirement is 2^{64} state values.

In addition to this generic attack, reference [4] describes a specialized attack that works for a class of weak messages. These are messages which result in chaining variables of the form $A||A$ where $A = 0||x$ or $A = y||0$, $x, y \in [0, 2^{64} - 1]$. There are 2^{64} images associated with such weak messages out of 2^{256} images. For this class of chaining variables $V_M^{(A)}(A, A) = A||A$. We

denote:

$$S = \{x||0||x||0, x \in [0, 2^{64} - 1]\} \cup \{0||y||0||y, y \in [0, 2^{64} - 1]\} \tag{3.3}$$

The following attack refers to a simplified version of Vortex that uses the Merkle-Damgård construction with strengthening as opposed to EMD. It works as follows:

Attack A_5, preprocessing:
A_5, $prep_1$: The adversary finds two message blocks W_0 and W_1 such that $Q(A_0, B_0, W_0) = Y||Y$ and $Q(Y, Y, W_1) = s' \in S$ for some intermediate value $Y \in [0, 2^{128} - 1]$. The words A_0, B_0, are the initial chaining variable words used by the Merkle-Damgård construction. The function $Q()$ is defined by equation 3.1. It is easy to show that this step can be completed with time complexity 2^{128}.
A_5, $prep_2$: For each value B, the adversary finds a special chaining value s of the form $0||x$ or $y||0$, $x, y \in [0, 2^{64} - 1]$, and a message word W' such that $B = \tilde{A}_s(W') \oplus W'$. The adversary stores the result in a table for all possible values of B. This step is associated with 2^{128} time and space complexity.
A_5, $prep_3$: For each s of the form $0||x$ or $y||0$, $x, y \in [0, 2^{64} - 1]$, the adversary finds two message words W_1 and W_2 such that the quantities $\tilde{A}_s(W_1) \oplus W_1$ and $\tilde{A}_s(W_2) \oplus W_2$ are also of the form $0||x$ or $y||0$.The time complexity of this step is 2^{65}.

Attack A_5, on-line phase:
A_5, $step_1$: The adversary finds a special chaining value s that leads to a given image from the table computed in the $prep_2$ step of A_5.
A_5, $step_2$: The adversary begins constructing a second preimage starting with the message blocks W_0 and W_1 computed in the step $prep_1$ of A_5. These message blocks result in a special chaining value $s' \in S$.
A_5, $step_3$: The adversary connects the chaining value s' with s using two trees of depth 33 blocks. These trees are constructed from the values computed in step $prep_3$ of A_5.

Reference [4] shows that this attack requires a preprocessing phase with time complexity 2^{128} and storage complexity of 2^{128}, plus an on-line phase having 2^{33} time complexity. There are 2^{64} images associated with weak messages out of a total of 2^{256} images.

3.4 Impossible Images

Reference [5] points out that Vortex has impossible images, and therefore its output can be distinguished from random. It suggests that the Vortex merging function has at least 2^{101} colliding pairs. Since the output of the Vortex hash is obtained from this merging function, this means that more than 2^{101} elements of the set $\{0, 1\}^{256}$ are impossible images.

Let $A = [A_1 : A_0]$ and $B = [B_1 : B_0]$ where A_1, A_0, B_1 and B_0 are $N/8$ bit words. Let also $A_1 = B_1 = 1$ and $A_0 \oplus B_0 = A_0 \boxplus B_0$. Then $V_M^{(A)}(A, B) = V_M^{(A)}(B, A) = [1 : A_0 \oplus B_0 : 1 : A_0 \oplus B_0]$. Hence the pairs A, B and B, A are colliding. Since the size of the domain of $V_M^{(A)}(A, B)$, which is 2^{256}, is equal to the size of its range (also 2^{256}) each different colliding pair is associated with a different impossible image. References [4, 5] claim that there are at least 2^{101} colliding pairs which result in 2^{101} impossible images. They also describe an algorithm of complexity 2^{96} that determines whether a given image is impossible or not. A distinguisher attack is constructed as follows:

Attack A_6:
A_6, $step_1$: The adversary selects an image from an input set. It tests whether the image is in the set of impossible images of Vortex. Impossible images are independent of the input message and, hence, a property of the hash structure.
A_6, $step_2$: If the answer to the $step_1$ of A_6 is yes then the distinguisher returns a reply that the input is definitely not coming from Vortex. Otherwise it repeats $step_1$ of A_6 for all remaining input images.

3.5 Output Bit Correlation

Reference [6] shows that some bits of the Vortex output are equal with much higher probability as compared to a random signal, and therefore the output can be distinguished from random. It therefore suggests that Vortex is not applicable to random number generation and key derivation. The least significant bits of the variables A_0 and B_0 depend on 3 bits only, two of which are in common. A_0 and B_0 derive from the following logical equations $A_0 = (X \text{ AND } Y) \oplus Z$ and $B_0 = (Z \text{ AND } W) \oplus Y$ for some input bit values W, X, Y, Z. If $X = W = 1$ then $A_0 = B_0$. Hence these bits are equal with probability 5/8 (higher than the ideal 0.5). The distinguisher attack works as follows:

Attack A_7:
A_7, $step_1$: The adversary counts the frequency of equality of output bits A_0 and B_0 in a set of given images. If this is 5/8 then the distinguisher returns that these images are generated by Vortex .

4 Structural Analysis of the Vortex Hash

We present here a root-cause analysis of the various attacks (whether practical or not): (i) openness to inversion, (ii) use of a narrow pipe across the entire transform, and (iii) symmetry in merging;

The design characteristics (ii) and (iii) result from conscious decisions made in order to balance the security of Vortex with high enough performance. As we show later, simple amendments can address the attacks.

4.1 Openness to Inversion

The Vortex sub-block transformation is open to inversion: the merging function, which is the last processing step of this transformation, can be inverted at computational cost ranging from 2^{96} [5] to 2^{128} [3] operations, depending on the algorithm used. This makes Vortex vulnerable to a class of meet-in-the-middle attacks, where an adversary inverts the transformations that result in an image up to a middle point. The adversary also creates some state for this middle point and connects the state with the initial or final value of the hash by choosing appropriate message blocks. This is the rationale behind all first and second preimage attacks A_3, A_4 and A_5.

Solutions to this problem could be to change the merging function or to introduce additional processing steps in the Vortex sub-block to prevent adversaries from inverting it. Changing the merging function is not a trivial task. We fear it would result in radical modifications of the Vortex specification with potentially unpredictable security implications. The merging function design was motivated by performance [2] and from the need to perform non-linear

mixing of bits across 128-bit blocks. Given these design requirements, we believe that the merging function fulfils its role as part of the Vortex structure.

Another solution to this problem is to introduce more processing steps in the Vortex sub-block transformation to make it less open to inversion. The processing step we can think of with the least performance impact is a feed-forward XOR around Vortex sub-block. Such feed-forward XOR was present in an earlier version of Vortex that was presented at ISC 2008 [1]. In this section we argue that Vortex needs feed-forward XOR operations at both levels: inside and around the Vortex sub-block transformation. A feed-forward XOR around the Vortex sub-block forces an adversary to guess the message word processed by the sub-block. Such guessing can be hard, potentially increasing the complexity of meet-in-the-middle attacks or making them impossible. In Section 5 we show that such feed-forward XOR mitigates attacks A_2, A_4 and A_5.

4.2 Narrow Pipe Across the Domain Extension Transform

Another characteristic of the Vortex design is that it maintains a narrow pipe across its entire domain extension transform. By "narrow pipe" we mean that the size of the hash state inside its compression function is equal to the size of the output digest. A wide pipe, in contrast, maintains hash state which is much larger in size than the output digest. The narrow pipe design of Vortex is also motivated by performance.

The narrow pipe of Vortex uses block cipher stages based on the Rijndael round transformation and multiplication. These are very strong mixing primitives and result in good collision resistance [2]. While having a good narrow pipe design is not necessarily a design flaw it may make a design vulnerable to a class of attacks. For example in the case of Vortex, the first preimage attack A_3 results from using a narrow pipe across its entire domain extension transform. Because of the presence of a narrow pipe, an adversary can reduce the first preimage resistance of the hash from the ideal of 2^{256} to 2^{192} efforts by just maintaining intermediate state of 2^{64} triplets (A, B, W). Such attack cannot be mitigated by a feed-forward XOR mechanism like the one discussed earlier in Section 4.1. This is because, in this attack, the adversary has the freedom to use arbitrary message words to connect a middle point with the final hash value.

A solution to this problem is to increase the hash state inside every compression function call Vortex makes. Such solution, however needs a radical revision of the Vortex compression function, which we want to avoid due to its performance and security implications. We propose an alternative solution where, instead of increasing the hash state size inside every compression function call, we only increase the state size in the last Vortex block transformation. Widening the pipe in this manner substantially increases the complexity of first preimage attacks based on the meet-in-the-middle principle. This is because an adversary needs to maintain much larger state and perform substantially more operations, even if a single wide pipe block is present in the hash chain. In the next section we suggest a modification to the last Vortex block transformation that mitigates the first preimage attack A_3. Our modification is based on increasing the internal hash state from 256 bits to 384 bits for Vortex 256 and from 512 bits to 764 bits for Vortex 512. We show that the complexity of this attack is pushed back to the ideal values of 2^{256} and 2^{512} efforts for Vortex 256 and Vortex 512.

4.3 Symmetry in the Merging Function

The merging function of Vortex has some symmetry which can be exploited. Symmetry results from using multiplication as a building block. Multiplication is a highly non-linear operation,

and hence a good cryptographic component due to its ability to destroy bit differentials. However it has several weaknesses. First it is a commutative operation. Second, the degree of mixing is not uniform across all bits of the output. For example, in the case of 64-bit carry-less multiplication bits 0 and 126 of the output result from a single logical product. Bit 63 of the output, however, results from XOR-ing 64 logical products. The Vortex merging function uses multiplication in a rather simple way. It first splits the input words into two halves and then computes the products of the inner and outer halves. The resulting products are added with the input words producing the merging function output. The only measure Vortex takes inside the merging function to avoid collisions is to make merging non-commutative. This is done by performing additions between computed products and input words as XORs for the least significant half of the output, and as integer additions modulo 2^{64} for the most significant half of the output.

References [5, 6] show that such structure has impossible images and correlates few bits of the output. One implication is that an adversary can distinguish the Vortex output from random. Another implication is that Vortex cannot be used as a replacement for a random oracle in any cryptosystem, because the Vortex compression function does not approximate a fixed input length random oracle, even though Vortex uses the EMD transform, which is pseudo-random oracle preserving.

To improve upon the current design we propose to skip the merging step in the last block of Vortex. This amendment solves the output bit correlation problem [6]. It does not eliminate impossible images, however, but makes them harder to detect. Another amendment would be to extend the Vortex block transformation with an additional block cipher stage eliminating all impossible images and output bit correlation but the security of the resulting hash would be depending on the design of this additional stage. Such amendment is not trivial and may have unexpected security implications.

5 Amendments to the Vortex Specification

This section details minor amendments to the Vortex structure, that address all of the above attacks.

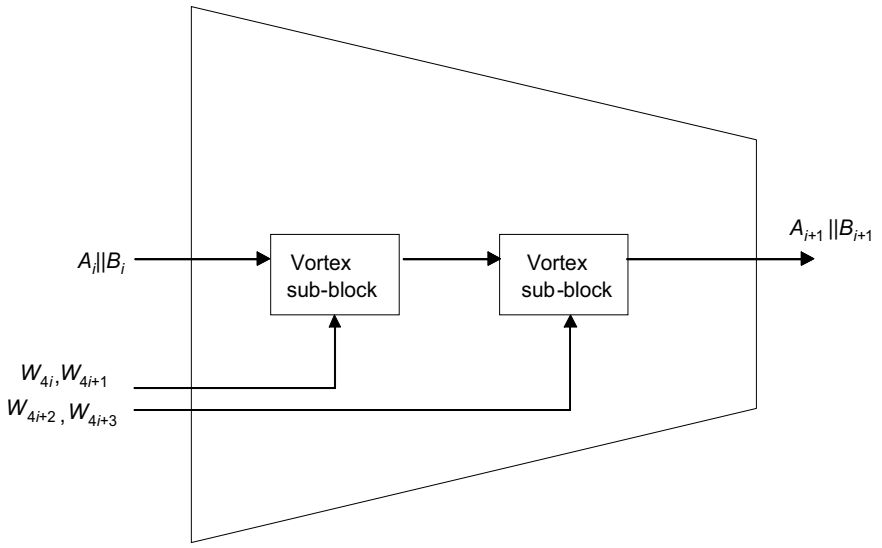
5.1 Vortex-M: Sub-block with additional Feed-forward XOR

The first amendment is to include a feed-forward XOR operation around the Vortex sub-block transformation, as illustrated in Figure 4. We call this amended version of Vortex "Vortex-M". Pseudo-code for the Vortex-M sub-block and last Vortex-M sub-block is given below. The processing steps in the Vortex-M sub-block are the same as in the original Vortex sub-block, apart form the inclusion of a feed-forward XOR operation in the end of the transformation.

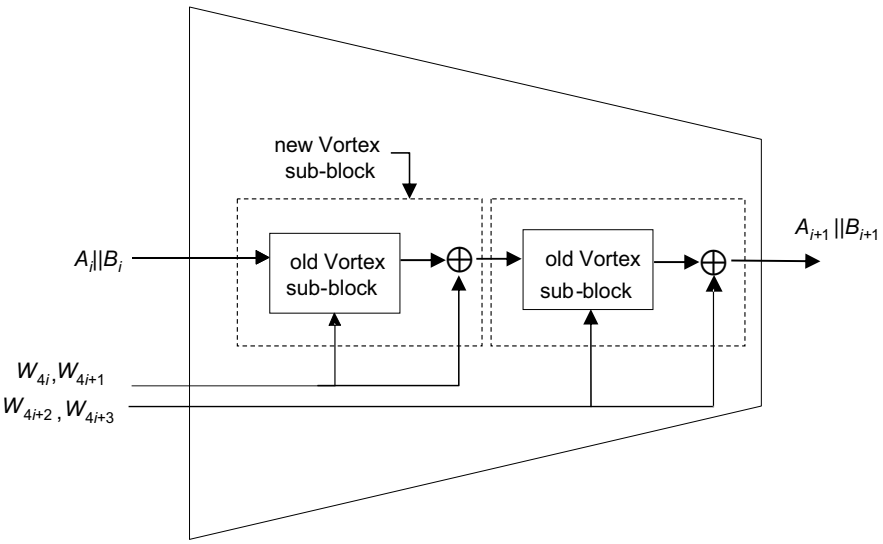
Vortex-M Sub-block (A, B, W_0, W_1)
begin

;W_0 is the first word of the current sub-block to be processed
$A \leftarrow \tilde{A}_A(W_0) \oplus W_0$
$B \leftarrow \tilde{A}_B(W_0) \oplus W_0$ // Matyas-Meyer-Oseas transformation
$A||B \leftarrow V_M^{(A)}(A, B)$

;W_1 is the second word of the current sub-block to be processed
$A \leftarrow \tilde{A}_A(W_1) \oplus W_1$
$B \leftarrow \tilde{A}_B(W_1) \oplus W_1$
$A||B \leftarrow V_M^{(A)}(A, B)$



(a) Original Vortex Block



(b) Vortex-M Block

Figure 4: Feed-forward XOR Amendment

return $A \oplus W_0 \oplus W_1 || B \oplus W_0 \oplus W_1$
end

Last Vortex-M Sub-block (A, B, W_0, W_1)
begin
;W_0 is the first word of the current sub-block to be processed
$A \leftarrow \tilde{A}_A(W_0) \oplus W_0$
$B \leftarrow \tilde{A}_B(W_0) \oplus W_0$
$A||B \leftarrow V_M^{(A)}(A, B)$
for $i \leftarrow 1$ to D_F **do**
;D_F is the degree of diffusion
;W_1 is the second word of the current sub-block to be processed
$A \leftarrow \tilde{A}_A(W_1) \oplus W_1$
$B \leftarrow \tilde{A}_B(W_1) \oplus W_1$
$A||B \leftarrow V_M^{(A)}(A, B)$
return $A \oplus W_0 \oplus W_1 || B \oplus W_0 \oplus W_1$
end

In what follows we analyze the security of Vortex-M:

Theorem 5.1 *The feed-forward XOR amendment of Vortex-M mitigates collision attack* A_2.

Proof: Step$_3$ of A_2 returns two message pairs $W_0^{(i)}||W_1$ and $W_0^{(j)}||W_1$ which collide. By "collide" we mean that $Q(Q(A, B, W_0^{(i)}), W_1) = Q(Q(A, B, W_0^{(j)}), W_1)$ for some index pair $i, j \in [0, R-1], i \neq j$. We distinguish between two cases. Case 1: $W_0^{(i)}$ and $W_0^{(j)}$ are the first message words fed into a Vortex-M sub-block transformation, and W_1 is the second word fed into the same transformation. Case 2: $W_0^{(i)}$ and $W_0^{(j)}$ are the second message words fed into a Vortex-M sub-block transformation, and W_1 is the first word fed into a subsequent sub-block transformation. Let's assume that case 1 is true. The output from the Vortex-M sub-block is $Q(Q(A, B, W_0^{(i)}), W_1) \oplus (W_0^{(i)} \oplus W_1 || W_0^{(i)} \oplus W_1)$ if the input is $W_0^{(i)}||W_1$. Similarly, the output from the Vortex-M sub-block is $Q(Q(A, B, W_0^{(j)}), W_1) \oplus (W_0^{(j)} \oplus W_1 || W_0^{(j)} \oplus W_1)$ if the input is $W_0^{(j)}||W_1$. Since $W_0^{(i)} \neq W_0^{(j)}$, the two outputs differ, so no collision occurs under this attack. If case 2 is true, then the output from the first Vortex-M sub-block is $Q(Q(A, B, W_{-1}), W_0^{(i)}) \oplus (W_{-1} \oplus W_0^{(i)} || W_{-1} \oplus W_0^{(i)})$ if the input is $W_{-1}||W_0^{(i)}$. Similarly, the output from the first Vortex-M sub-block is $Q(Q(A, B, W_{-1}), W_0^{(j)}) \oplus (W_{-1} \oplus W_0^{(j)} || W_{-1} \oplus W_0^{(j)})$ if the input is $W_{-1}||W_0^{(j)}$. Since $W_0^{(i)} \neq W_0^{(j)}$ the feed-forward XOR in the first Vortex-M sub-block modifies the chaining variable words fed into the second Vortex-M sub-block so no left collisions occur, which attack A_2 is based upon.

Theorem 5.2 *The feed-forward XOR amendment of Vortex-M mitigates the second preimage attack* A_4.

Proof: Step$_3$ of A_4 returns two message blocks W_0 and W_1 resulting in hash state $A_2||B_2 = 0||y||B_2$. These blocks together with a third message block W_y replace three message words $W_0^{(m)}, W_1^{(m)}, W_2^{(m)}$ of the original message, in the original attack. This is because the message sequences W_0, W_1, W_y and $W_0^{(m)}, W_1^{(m)}, W_2^{(m)}$ result in the same hash state $A_3||B_3 = V_M^{(A)}(X_2, Y_2)$. Once merging is complete, the feed-forward XOR of Vortex-M, further modifies the hash state coming out of the Vortex-M sub-block processing W_y (or $W_2^{(m)}$). Since $W_y \neq W_2^{(m)}$, the final hash-state associated with sequences W_0, W_1, W_y and $W_0^{(m)}, W_1^{(m)}, W_2^{(m)}$ is different.

Theorem 5.3 *The feed-forward XOR amendment of Vortex-M mitigates the second preimage attack* A_5.

Proof: In step_3 of A_5, the adversary connects the chaining value s' with s using two trees of depth 33 blocks. These message blocks span multiple Vortex-M sub-block transformations each ending in a feed-forward XOR operation. Since message blocks do not have, in general, their 64 most or least significant bits equal to zero these feed forward XOR operations push the intermediate hash state out of the set S which is the basic assumption under which the attack works. Moreover let's assume that $W^{(s)}$ and $W^{(m)}$ are message words of a second preimage, returned by A_5, and an original message for which the two sequences differ. Let's also assume that all subsequent message words are the same. Once merging is complete the sub-block that processes $W^{(s)}$ (or $W^{(m)}$) applies a feed-forward XOR operation on the resulting hash state. Such XOR operation further modifies the hash state coming out of this Vortex-M sub-block. Since $W^{(s)} \neq W^{(m)}$, the final hash-state associated with the returned second preimage is not the same as the one of the original message.

5.2 Vortex-W: Widen the Pipe in the Last Block

The second amendment is to widen the pipe of the last block transformation of Vortex, as illustrated in Figure 5. The idea behind this amendment is to increase the internal state of the hash so that the complexity of any pre-image attack is pushed back to 2^N.

We extend the last block transformation to include a tweak value consisting of 3 $N/2$-bit variables $T_A||T_B||T_C$ instead of two. These are passed into three Matyas-Meyer-Oseas stages instead of two. Let A, B, C be their outputs. The merging function employed on these outputs is:

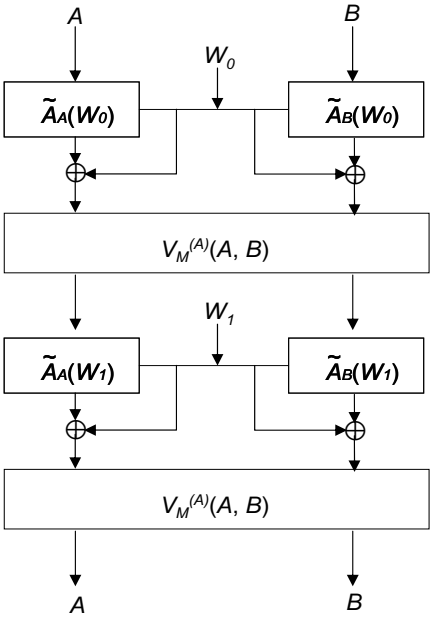
$$V_M^{(A+)}(A,B,C) = V_M^{(A)}(A{\oplus}C, B{\oplus}C)||M_{N/2}(V_M^{(A)}(A{\oplus}B, C)){\oplus}L_{N/2}(V_M^{(A)}(A{\oplus}B, C)) \quad (5.1)$$

instead of just $V_M^{(A)}(A,B)$. Here, $M_{N/2}()$ and $L_{N/2}()$ we denote the N/2 most and least significant bits of $V_M^{(A)}(A,B)$, respectively. The new merging function $V_M^{(A+)}(A,B,C)$, shown in Figure 6, uses the function $V_M^{(A)}(A,B)$ as building block in the following manner: It first XORs the input word $C$with A and B and passes the results as input to $V_M^{(A)}()$. The output of this function call (N bits) constitutes the N most significant bits of the output of $V_M^{(A+)}(A,B,C)$. It also XORs A with B and passes the words $A{\oplus}B$, C as input to $V_M^{(A)}()$. The $N/2$ most and least significant bits of this output are XOR-ed with each other. This makes up for the remaining $N/2$ least significant bits of the output of $V_M^{(A+)}(A,B,C)$. The amended last block transformation is listed below:

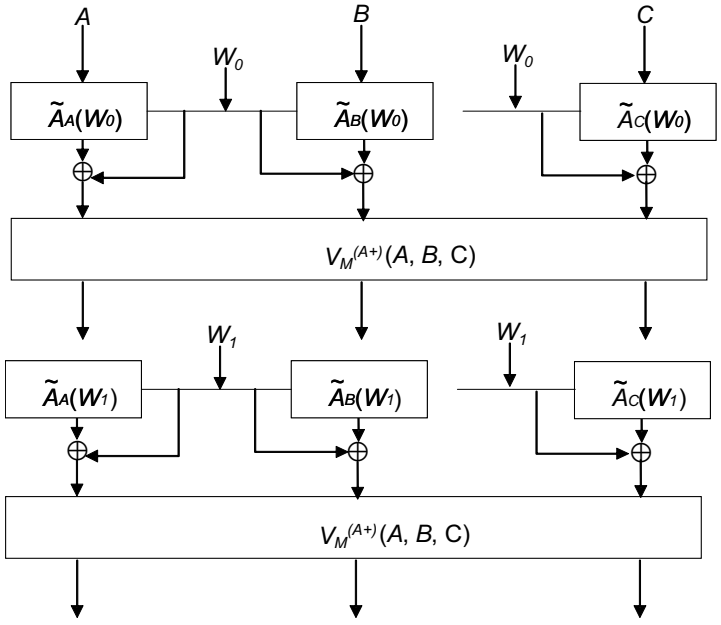
Last Vortex-W Block $(A_k, B_k, W_{4\cdot k}, W_{4\cdot k+1})$
begin
    $A||B||C \leftarrow$ **Vortex-W Sub-block**$(T_A, T_B, T_C, A_k, B_k)$
    $A||B||C \leftarrow$ **Last Vortex-W Sub-block**$(A, B, C, W_{4\cdot k}, W_{4\cdot k+1})$
    return $A||B$
end

The algorithms Vortex-W sub-block() and Last Vortex-W sub-block() are:

Vortex-W Sub-block (A, B, C, W_0, W_1)



(a) Narrow Pipe Vortex Sub-block



(b) Wide Pipe Vortex Sub-block

Figure 5: Last Block Wide Pipe Amendment

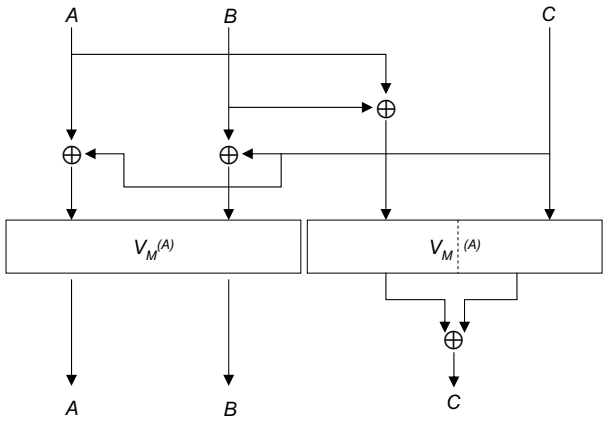


Figure 6: The Merging Function Used in the Last Block of Vortex-W

begin

;W_0 is the first word of the current sub-block to be processed
$A \leftarrow \tilde{A}_A(W_0) \oplus W_0$
$B \leftarrow \tilde{A}_B(W_0) \oplus W_0$ // Matyas-Meyer-Oseas transformation
$C \leftarrow \tilde{A}_C(W_0) \oplus W_0$
$A||B||C \leftarrow V_M^{(A+)}(A, B, C)$

;W_1 is the second word of the current sub-block to be processed
$A \leftarrow \tilde{A}_A(W_1) \oplus W_1$
$B \leftarrow \tilde{A}_B(W_1) \oplus W_1$
$C \leftarrow \tilde{A}_C(W_1) \oplus W_1$
$A||B||C \leftarrow V_M^{(A+)}(A, B, C)$
return $A||B||C$

end

Last Vortex-W Sub-block (A, B, C, W_0, W_1, D_F)
begin
;W_0 is the first word of the last sub-block to be processed
$A \leftarrow \tilde{A}_A(W_0) \oplus W_0$
$B \leftarrow \tilde{A}_B(W_0) \oplus W_0$
$C \leftarrow \tilde{A}_C(W_0) \oplus W_0$
$A||B||C \leftarrow V_M^{(A+)}(A, B, C)$
for $i \leftarrow 1$ to D_F **do**
;D_F is the degree of diffusion
;W_1 is the second word of the last sub-block to be processed
$A \leftarrow \tilde{A}_A(W_1) \oplus W_1$
$B \leftarrow \tilde{A}_B(W_1) \oplus W_1$
$C \leftarrow \tilde{A}_C(W_1) \oplus W_1$
$A||B||C \leftarrow V_M^{(A+)}(A, B, C)$
return $A||B||C$
end

By extending the internal state of the hash from N to $3N/2$ bits we increase the complexity of the described preimage attack to 2^N. This is because the optimal size of the list pre-computed by the adversary is increased to $2^{N/2}$ from $2^{N/4}$. In this case the adversary needs 2^N efforts to find an element in the list.

Theorem 5.4 *The wide pipe amendment of Vortex-W mitigates the first preimage attack* A_3.

Proof: We follow the steps of this attack on Vortex-W instead of the original Vortex and determine its complexity. For simplicity, we consider the strengthened Merkle-Damgård construction instead of EMD.

Attack A_3 on Vortex-W:
A_3, step$_1$: The adversary computes a triplet $A_I||B_I||C_I = (V_M^{(A+)})^{-1}(A_F, B_F, C_F)$ by inverting the function $V_M^{(A)}()$ and using an arbitrary value for C_F.
A_3, step$_2$: The adversary chooses a message word W_3 arbitrarily and computes $x = \tilde{A}_A(W_3) \oplus W_3$ for many different values of the chaining variable word A until $x = A_I$. The adversary also computes $y = \tilde{A}_B(W_3) \oplus W_3$ for many different values of the chaining variable word B until $y = B_I$. Finally, the adversary computes $z = \tilde{A}_C(W_3) \oplus W_3$ for many different values of the chaining variable word C until $z = C_I$. This brute-force attack is accomplished with complexity $2^{N/2}$.
A_3, step$_3$: In step three, the previous step is repeated another $2^T - 1$ times, where T is a tunable parameter of the attack. In the end of this step the adversary has a list of 2^T quadruplets (A, B, C, W_3) for which: $V_M^{(A+)}(\tilde{A}_A(W_3) \oplus W_3, \tilde{A}_B(W_3) \oplus W_3, \tilde{A}_C(W_3) \oplus W_3) = A_F||B_F||C_F$. The total complexity for creating this list is $2^{T+N/2}$.
A_3, step$_4$: In the last step of this attack the adversary computes the output of successive sequences of Matyas-Meyer-Oseas transformations and merging. The computation continues until a quadruplet is found, giving a hash state value $A||B||C$ included in the list computed in the previous step. The number of possible values for the hash state $A||B||C$ is $2^{3N/2}$. The size of the list computed in step$_3$ of A_3 is 2^T. Hence the number of iterations needed to find one element in the list is $2^{3N/2-T}$. The complexity of this attack is $\max(2^{T+N/2}, 2^{3N/2-T})$.

If one uses the value $T = N/4$ as in the case of the original attack the complexity of this attack becomes $2^{1.25N}$ which is higher than the complexity of the brute-force attack. The complexity of this attack is minimized for $T = N/2$ becoming 2^N, which is ideal. Hence attack A_3 is mitigated.

We note that this second amendment modifies the definition of the EMD construction as described in [8]. The changes we make is that the size of the tweak value $T_A||T_B||T_C$ is larger than the size of the chaining variable $A||B$. Also, the compression function used in the last block (i.e., the envelope) is different form the one used in all other blocks. It is not difficult to show that the properties of EMD hold for this variant as well.

5.3 Vortex-S: Skipping the Merging Function in the Last Sub-block

The third amendment is to skip the last merging stage in the last Vortex sub-block transformation. The last Vortex sub-block repeats the stages of Matyas-Meyer-Oseas transformations and merging $D_F - 1$ times instead of D_F and then does Matyas-Meyer-Oseas one more time without merging. In this way the output comes directly from block cipher stages, which approximate ideal block ciphers. As a result, no output bit correlation characterizes this version of Vortex, and attack A_7 is mitigated. Further, impossible images become much harder to detect. To find impossible images in this case, one has to invert a Matyas-Meyer-Oseas transformation. Ideally that should be impossible, but in practice we say it is hard. We call this version of Vortex, Vortex-S. We note that attack A_7 is in our opinion the only practical attack on Vortex, and this amendment the simplest to implement. So Vortex can become secure from a practical stand-point with only this amendment. The Vortex-S last sub-block pseudo-code

	attack A_2	attack A_3	attack A_4	attack A_5	attack A_6	attack A_7
description	collision attack at time complexities $2^{124.5}$, $2^{251.7}$	pre-image attacks with product of time and space complexity equal to ideal			distinguisher at time complexity at least 2^{96}	practical distinguisher
	protected against?					
Vortex	NO	NO	NO	NO	NO	NO
Vortex-M	YES	NO	YES	YES	NO	NO
Vortex-W	NO	YES	NO	NO	NO	NO
Vortex-S	NO	NO	NO	NO	YES	YES
Vortex+	YES	YES	YES	YES	YES	YES

Table 1: The Impact of Amendments on the Security of Vortex

is given below.

Last Vortex-S Sub-block (A, B, W_0, W_1, D_F)
begin
;W_0 is the first word of the last sub-block to be processed
$A \leftarrow \tilde{A}_A(W_0) \oplus W_0$
$B \leftarrow \tilde{A}_B(W_0) \oplus W_0$
$A||B \leftarrow V_M^{(A)}(A, B)$
for $i \leftarrow 1$ to $D_F - 1$ **do**
;D_F is the degree of diffusion
;W_1 is the second word of the last sub-block to be processed
$A \leftarrow \tilde{A}_A(W_1) \oplus W_1$
$B \leftarrow \tilde{A}_B(W_1) \oplus W_1$
$A||B \leftarrow V_M^{(A)}(A, B)$
$A \leftarrow \tilde{A}_A(W_1) \oplus W_1$
$B \leftarrow \tilde{A}_B(W_1) \oplus W_1$
return $A||B$
end

5.4 The Impact of the Amendments

The impact of each amendment on the security of Vortex is summarized in Table 1. Attack A_1 is not included as it assumes control of variables A and B which Vortex does not offer to the attacker. As shown in the table Vortex as submitted to the SHA-3 competition does not offer protection against attacks A_2-A_7. We remind that among these attacks, A_2 drops only few exponent bits of collision resistance, A_3 A_4 and A_5 have time and space complexity product equal to ideal, and A_6 is associated with time complexity which is at least 2^{96}. A_7 is the only practical attack. Vortex-M adds protection against A_2, A_4 and A_5 but not A_3, A_6 and A_7. Vortex-W adds protection against A_3 only. Finally Vortex-S adds protection against A_6 and A_7 but not A_2-A_5, making Vortex secure from a practical point of view. One can also consider variants of Vortex with more than one amendment. For example by combining our first, second and third amendments we create a version of Vortex ("Vortex+") that addresses all published attacks. It is quite obvious that the performance impact of the amendments is marginal. This is because the wide pipe (Vortex-W) and skipping (Vortex-S) amendments apply only to the last sub-block, and hence do not impact performance, whereas the feed-forward XOR (Vortex-M) amendment adds only 3 XOR operations per sub-block (256 or 512 bits) which can be

completed in a few clocks by most processors.

6 Concluding Remarks

We presented here a root-cause analysis of a number of recently published attacks on the Vortex hash function family, and showed that they stem from: (i) Openness to inversion (resulting in multi-collision attacks, second pre-image attacks, and second pre-image attacks for weak messages) (ii) Use of a narrow pipe across the whole domain extension transform (resulting in first pre-image attacks); (iii) Symmetry in the merging function (resulting in impossible images and output bit correlation). We derived three amendments to the Vortex structure and showed that they address all of the attacks. These amendments can be used separately or in combination, depending on those attacks that need to be addressed. From the practical standpoint, the only practical attack is Attack A_7 and it is mitigated by the third amendment (Vortex-S).

References

[1] Shay Gueron and Michael Kounavis "Vortex: A New Family of One Way Hash Functions based on AES Rounds and Carry-less Multiplication" *ISC* 2008.

[2] Shay Gueron and Michael Kounavis "Vortex: A New Family of One Way Hash Functions based on Rijndael Rounds and Carry-less Multiplication" *Submission to NIST*, 2008, available online: `http://ehash.iaik.tugraz.at/wiki/Vortex_(SHA-3_submission)`

[3] Lars R. Knudsen, Florian Mendel, Christian Rechberger and Søren S. Thomsen "Collision and Preimage Attacks on Vortex as submitted to the SHA-3 competition" *available online*: `http://ehash.iaik.tugraz.at/wiki/Vortex_(SHA-3_submission)`

[4] Jean-Philippe Aumasson, Orr Dunkelman, Florian Mendel, Christian Rechberger and Søren S. Thomsen "Cryptanalysis of Vortex" *AfricaCrypt* 2009.

[5] Jean-Philippe Aumasson and Orr Dunkelman "A note on Vortex's security" *available online*: `http://ehash.iaik.tugraz.at/wiki/Vortex_(SHA-3_submission)`

[6] Niels Ferguson "Simple correlation on some of the output bits of Vortex" *available online*: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html`

[7] B. Brachtl, D. Coppersmith, M. Hyden, S. Matyas, C. Meyer, J. Oseas, S. Pilpel, and M. Schilling "Data authentication, using modification detection ocdes based on a public one-way encryption function" *United States Patent* No. 4,908,861. Filed August 28, 1987.

[8] M. Bellare and T. Ristenpart "Multi-Property-Preserving Hash Domain Extension and the EMD Transform" *Advances in Cryptology - ASIACRYPT 2006*, LNCS 4284, pp. 299-314, 2006.

[9] I. Damgård "A Design Principle for Hash Functions" *Advances in Cryptology - CRYPTO 1989*, LNCS 435, pp. 416-427, 1989.

[10] "Advanced Encryption Standard" *Federal Information Processing Standards Publication 197*, available at: `http://csrc.nist.gov/publication/fips`