A Note on Vortex' Security

Jean-Philippe Aumasson[1] and Orr Dunkelman[2]

[1] FHNW, Windisch, Switzerland
[2] École Normale Supérieure, Paris, France

Abstract. Vortex is a hash function based on the AES that was presented at ISC'2008, and submitted to the NIST SHA-3 competition after some modifications that aim to strengthen it. This note first shows that the original Vortex is not collision-resistant, by describing an attack running in about 2^{58} compressions, instead of 2^{128} ideally. In the new version submitted to NIST, we present several properties that seem to render it unsuitable for the new hash standard. In particular, both versions of Vortex have the undesirable property of impossible images, which gives distinguishers for a HMAC based on Vortex and slightly speeds up preimage search.

1 Introduction

Vortex is the name of a AES-based hash function proposed by Gueron and Kounavis [2] at ISC'2008, and is also the name of the modified version [3] submitted to the NIST Hash Competition[3] 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.

The attack on Vortex-0 exploits the independence between processed message words, which allows collision search on a smaller space, reducing the complexity from 2^{128} to 2^{64}. We then present several attacks on Vortex-1: first, a pseudo-collision attack in time 2^{64}; then, a free-start collision attack in 2^{64} as well, which gives a standard collision attack with 2^{128} precomputation and 2^{64} online computations. We identify a weak class of messages, for which second preimages can be found in time 2^{128}, or with an additional preprocessing and storage using about 2^{33} time. Finally, we show that both versions of Vortex have many impossibles images: we describe techniques to find collisions through the output function V, and to detect impossible images (which gives a distinguisher for PRF's based on Vortex, like HMAC); we conclude that the use of Vortex as a PRF is to be avoided.

2 Vortex-0

Vortex-0 is a Merkle-Damgård iterated hash with 256-bit chain values and 256-bit digest. Given a 2×128-bit chain 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

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

and returns $A\|B$ as the new chain 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

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

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

The block cipher C is a reduced version of AES with three rounds, where a round (unlike in AES) is the sequence AddRoundKey, SubBytes, ShiftRows, and MixColumns.

The 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 computation of $V(A, B)$ goes as follows ("$\otimes$" denotes carryless multiplication, and addition is modulo 2^{64})

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

2.1 Black-box collisions

We present a collision attack that exploits structural properties of the subblock function, assuming that the main component (the block cipher) is perfect, i.e. we work under 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 $1 - 1/e^2 \approx 0.39$, 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 [2] 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 the following 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 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. A proof of the above formula goes as follows:

In Vortex-0 a *round* consists in 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

$$RK_1 \leftarrow \pi_1(K)$$
$$RK_2 \leftarrow \pi_2(RK_1)$$
$$RK_3 \leftarrow \pi_3(RK_2)$$

where the π_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.

Observe that

$$K \neq K' \Rightarrow RK_i^K \neq RK_i^{K'}, i = 1, 2, 3.$$

Assume that the permutations π_i's satisfy the properties expected for a random permutation, i.e. they are "random-looking" enough. Denote Π_i^K the permutation corresponding to the i-th round of C; Π_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. That is, composition breaks the "disjointness" property.

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 Π_i, Π_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 C seen as a key-to-ciphertext mapping, for any fixed plaintext, has a distribution close to that of a random function. With 3 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.

2.2 Faster collisions

We improve the previous collision attack by exploiting properties of the algorithm inside the block cipher C. More precisely, the attack exploits the low number of AES-like rounds (3, instead of at least 10 in AES), and the special form of a round in C.

A key observation is that AddRoundKey is not performed after the last (third) round of C. It thus suffices to find a collision right after the third AddRoundKey to have a collision on $\mathsf{C}_W(A)$.

Let's denote the rounds keys (RK_1, RK_2, RK_3) and (RK'_1, RK'_2, RK'_3) of our two instances. Denoting T the permutation made of SubBytes, ShiftRows, and MixColumns, and starting from a same plaintext x, we want a collision

$$y = T\left(T(x \oplus RK_1) \oplus RK_2\right) \oplus RK_3 = T\left(T(x \oplus RK'_1) \oplus RK'_2\right) \oplus RK'_3 \ ,$$

that is,

$$z = T(x \oplus RK_1) \oplus RK_2 \oplus T^{-1}(RK_3) = T(x \oplus RK'_1) \oplus RK'_2 \oplus T^{-1}(RK'_3) \ .$$

Now consider a specific byte of z seen as the 4×4 inner state of C (for example the byte at coordinates $(0, 0)$); clearly, this byte is affected by only byte of RK_2, and because of the ShiftRows and MixColumns operations in T or (T^{-1}), this byte is also affected by 4 bytes from RK_1 and 4 bytes from RK_3.

One can exploit the fact that not all key bytes affect a specific z byte: for example, given a RK_1 and a RK'_1, one can immediately modify a byte of RK'_1 to get a colliding byte in z. One thus gets a collision over 8 bits "for free", and so seeks a collision over 120 bits only. The cost of the attacks thus drops from 2^{64} to 2^{60} evaluations of C, compared to the attack in §2.1.

3 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 chain 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. The AES-like cipher C still makes 3 rounds, and the V function is the same as in Vortex-0. Note that Vortex-1 is not vulnerable to the attacks in §2.1 and §2.2.

3.1 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 input 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).

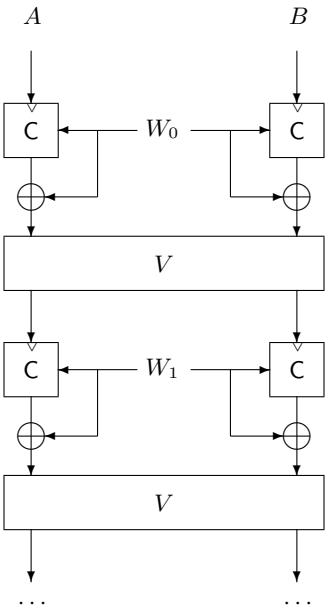

Fig. 1. Schematical view of Vortex-1's computation of a message digest (a hatch marks the key input).

3.2 Free-start collisions

We show how to find a pair colliding messages for Vortex-1 with any IV of the form $A\|B = A\|A$—call this 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. Note that 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} ideally.

3.3 Second preimages for weak messages

We show how to find second preimages of messages that produce a symmetric chain value (that is, of the form $A\|A$) during the digest computation. A key observation is that if $A = B$ and A 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 produce the chain value $\tilde{A}\|\tilde{A}$, find a message that leads to a symmetric chain 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 preimage of it.

Reaching the first symmetric chain 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 $2^{-127} \times (m/128 - 1)$.

Time-memory tradeoff variant. We show that a variant of the above attack with precomputation 2^{128} 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 $A\|A$ finding a special chaining value $s = s\|s$ and a message word W' such that

$$\mathsf{C}_s(W') \oplus W' = A\ ,$$

   and store it in a table for each possible A
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 A 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. [4, 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}.

4 Impossible images

We show that both versions of Vortex have *impossible images*. That is, the range of Vortex-0 and Vortex-1 doesn't span the whole $\{0,1\}^{256}$, but is a proper subset of it. This observation allows slightly faster preimage and collision search, and can be used to mount distinguishers on PRF's based on Vortex (e.g. HMAC).

Both Vortex-0 and Vortex-1 use the V function (see §2 for its definition) after each C evaluation. In particular, the final output of both Vortex-0 and Vortex-1 is an output of V. We investigated V, which maps an element of $\{0,1\}^{256}$ to an element of $\{0,1\}^{256}$, and found that it is not surjective, i.e. there exists impossible images. Experiments on reduced version suggest that V behaves more like a random function than like a permutation: for example, with the version of V with 12-bit A and B, about 66% of the outputs are reachable (against $1 - 1/e \approx 63\%$ for a random function[4]). In the remainder we denote $\mathcal{R}_V$ the range of V, and thus have $\mathcal{R}_V \subsetneq \{0,1\}^{256}$.

4.1 Multicollisions for V

We present a simple method to find two distinct inputs that map to the same value through 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:

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

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 won't modify the output. Note that this even allows multicollisions, since for a given pair (A_0, B_0) there exists many colliding modified pairs. Interestingly, it is easy to find a colliding pair $(A\|B, A'\|B')$ such that second preimages of $C\| \leftarrow V(A\|B)$ are easy to find; this is because the property $A_1 = B_1 = 1$ is preserved.

One can easily derive an upper bound on $|\mathcal{R}_V|$ from the above technique: observe that there are $\binom{64}{32} \approx 2^{60.5}$ possible weight-32 choices for A_0, and as much choices for B_0 given any fixed A_0; this gives 2^{120} colliding pairs, which means that more than 2^{120} elements of $\{0,1\}^{256}$ are impossible images, that is, $|\mathcal{R}_V| < 2^{256} - 2^{120}$. It follows that search for preimages and collisions is slightly faster than expected.

[4]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 actual range $\{0,1\}^n$ with high probability.

4.2 Detecting impossible images

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 for V much faster than the generic algorithm (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 128-bit value
- for each choice of O_0:
    - 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 "$y \in \mathcal{R}_V$"

The running time of the algorithm is mostly dominated by solving two sets of 32 equations over GF(2) 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). This algorithm can enjoy parallelism (i.e., given n CPUs the running time is divided by n).

Now, this allows us to distinguish the output of Vortex from a random string by solving 2^{97} operations (because if the algorithm fails to find a preimage of the output, then the string was not produced by a Vortex). Hence, given about 10 outputs of Vortex, we can almost surely identify that the string was indeed produced by Vortex calls.

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.

5 Conclusion

Our analysis suggests that the new version of Vortex is indeed stronger than the original one. Although our findings on Vortex-1 do not directly invalidate the security claims made by its designers, it seems that Vortex-1 (especially in MAC constructions such as HMAC or in key derivation scheme) may offer less than optimal security.

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. 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.
3. 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`.
4. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.