Vortex

A New Family of One-Way Hash Functions

Based on AES Rounds and Carry-less Multiplication

Shay Gueron

Intel Corporation, IL

and University of Haifa, IL

Michael E. Kounavis

Intel Corporation, US

Information Security Conference (ISC) 2008, Taipei, September 15-18, 2008

Agenda

- Goals
- Requirements from a cryptographic hash function
- Design philosophy
- Vortex description
- Initial results

Goals and Motivation

- Goals:
  - A family of one way hash functions that produce a 256 bits digest
  - Optimize for performance (use new the instructions)
- Motivation
  - Code Integrity:
    - Protect against viruses, rootkits, botnets, stealthware, malware
    - Hashed code execution
    - Fast to produce & verify code signatures
  - Data Integrity:
    - Efficient message authenticity
    - Surpass the performance of AES-GCM?
    - Fast digital signatures
  - Random Number Generation

Some Requirements from a Hash Function

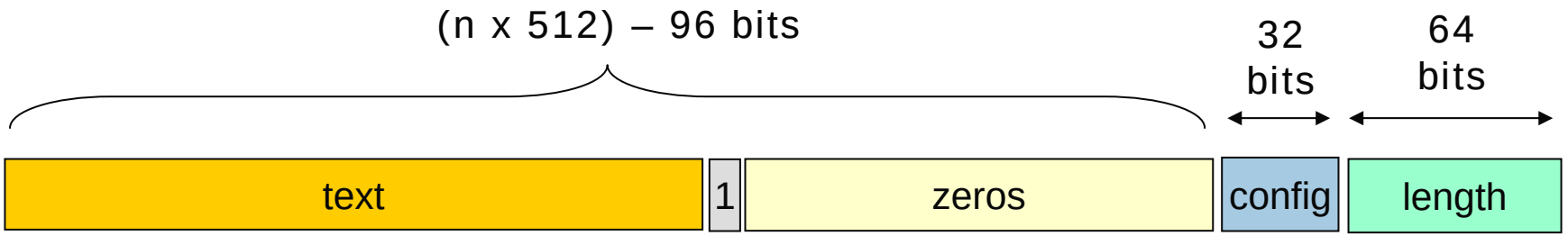
- Collision Resistance Preservation
  - Measures how difficult it is to find two texts that hash to the same value
- Pre-image Resistance Preservation
  - Measures how difficult it is to find a text that hashes to a particular value
- Pseudo-Random Function Preservation
  - Measures how close an appropriately keyed hash function is, to a random function
- Pseudo-Random Oracle Preservation
  - Measures how close the output of the one way hash function is (when we do not know the input), to the output of a random oracle
- Entropy Preservation
  - Measures how close the entropy of the output of the hash function is, compared to the entropy of the input

Vortex Design Philosophy

- Use the AES round as a building block
  - AES round does good mixing within the 128-bit State
    - Gain high performance from new AES instructions
- New methods for merging 128-bit blocks into a longer digest
  - Non-commutative merging
  - New “mode” of operation
    - Gain high performance from PCLMULQDQ instruction
- Why not just a full-AES for the hash function?
  - Straightforward: use the AES cipher a compression function
  - Variants:
    - Stronger key schedule algorithm
    - Smaller number of rounds
    - Balanced cryptographic strength between small number of AES rounds, stronger key schedule and the merging function
- Padding
  - Embeds the file size (standard)
  - Embeds also the hash configuration

Vortex Description

Vortex Padding



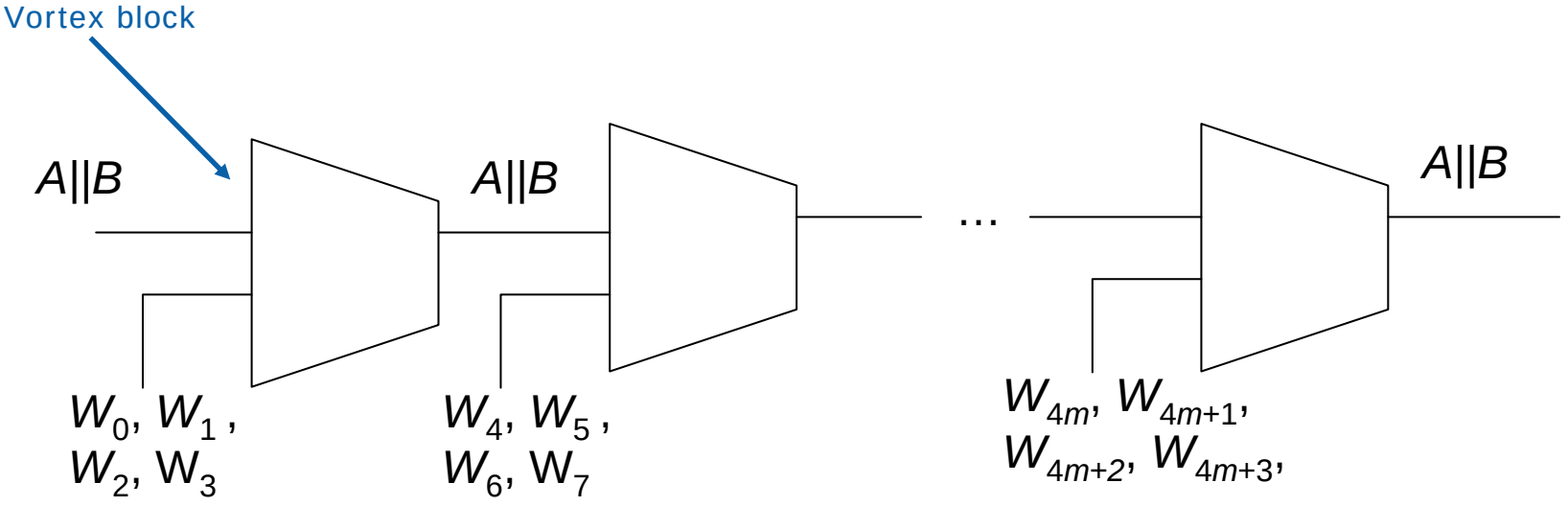
Vortex Flow Overview

Essentially a Merkle-Damgård construction

In: chaining variable $A||B$, and 4 input words $W0$, $W1$, $W2$, $W3$

("words" are 128b; $A||B$ are initialized using some constants)

Out: an updated value of the chaining variable

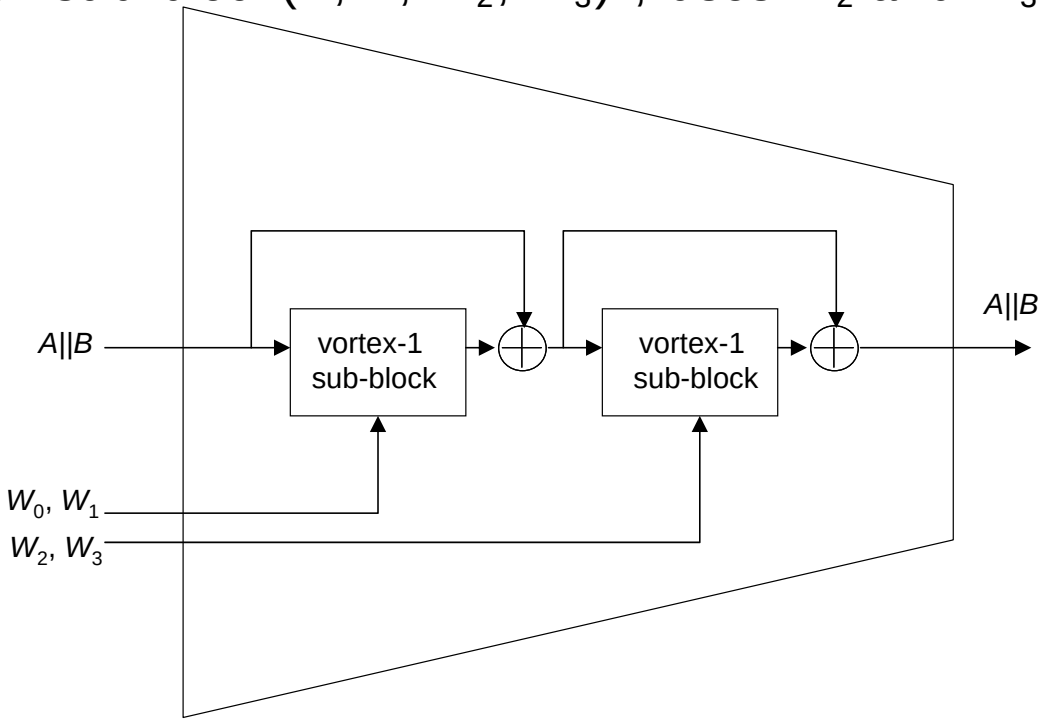


Vortex Block Operation

- Use chaining variables (2x128b): *A||B*
- Consume next 512 bits (4x128b) of the file: *W0, W1, W2, W3*

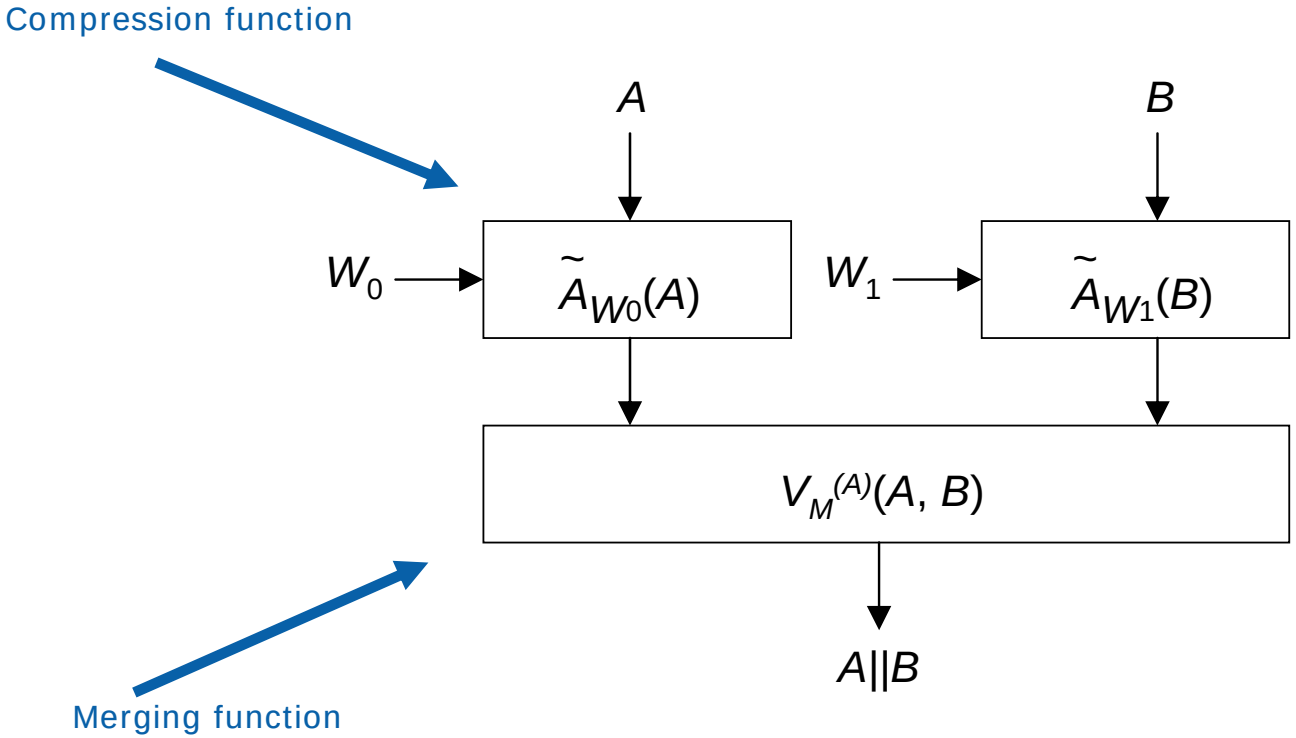
Vortex block(A, B, W_0, W_1, W_2, W_3)
{
$(A,B) \leftarrow (A,B) \oplus$ Vortex sub-block(A, B, W_0, W_1) ; uses W_0 and W_1
$(A,B) \leftarrow (A,B) \oplus$ Vortex sub-block(A, B, W_2, W_3) ; uses W_2 and W_3
}

Vortex block incorporates a Davies-Meyer structure around the Vortex sub-block in order to make the transformation non-reversible. Such structure is repeated twice



Vortex Sub-block

- Consume next 256 bits of the hashed file *W0, W1*
- Set Sub-block input to: *A, B* (current value of the hash) and *W0, W1*
- Use *W0* (key) and *A* (block) to compute *compression function* (*A, W0*)
- Use *W1* (key) and *B* (block) to compute *compression function* (*B, W1*)
- Merge *A* and *B* into a 256-bit digest using a merging function



The $\tilde{A}_K$ (X) Compression Transformation

- $\tilde{A}_K$ (X) is a lightweight block cipher based on an AES round
- Uses m (m=3) AES rounds (as specified in FIPS-197)
  - AES round operates consists of: ShiftRows, SubByte, MixColumns, AddRoundKey (operating on a State and a round key 128 bits each)

tmp = AESRound (X, RK1); tmp = AESRound (tmp, RK2); tmp = AESRound (tmp, RK3);

- The key schedule algorithm
  - Three 128b *Rcon* (fixed) values *RC*1, *RC*2 and *RC*3
  - Derive three round keys *RK*1, *RK*2 and *RK*3 as follows:
    - *RK*1 ← Perm(SBox(*K* ⊞ *RC*1))
    - *RK*2 ← Perm(SBox(*RK*1 ⊞ *RC*2))
    - *RK*3 ← Perm(SBox(*RK*2 ⊞ *RC*3))

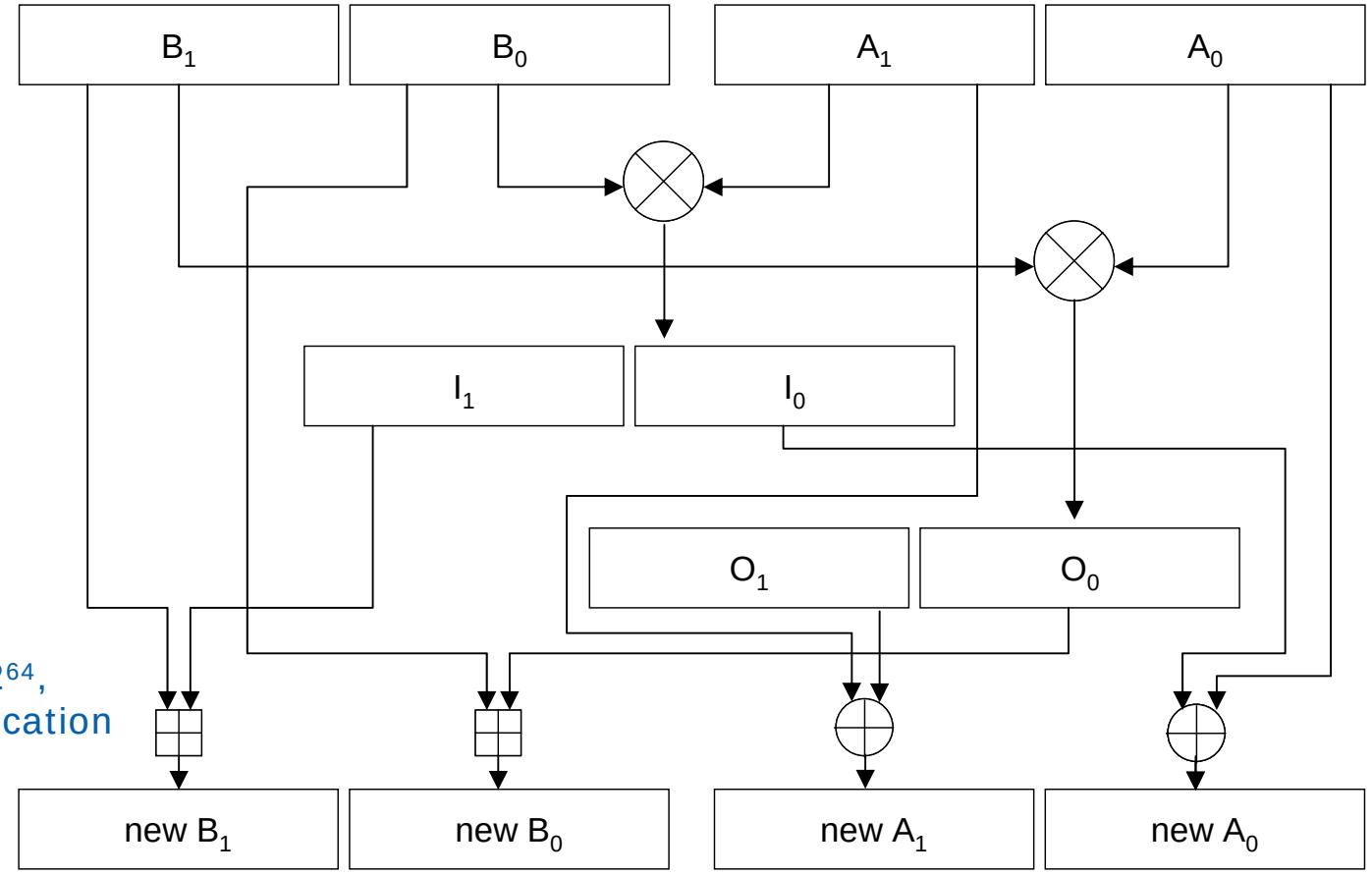
'⊞' is addition modulo 2^{128}

Perm() is a bit permutation

SBox applied on all 16 bytes

The Merging Function

A = [$A1$, $A0$]; B = [$B1$, $B0$] (chunks of 64-b)
O ← A0 ⊗ B1; I ← A1 ⊗ B0
I = [$I1$, $I0$]; let O = [$O1$, $O0$]
return [$B1 \boxplus I1$, $B0 \boxplus O0$, $A1$ *xor* $O1$, $A0$ *xor* $I0$]



'⊞' is addition modulo 2^{64},
'⊗' is carry-less multiplication

Results

Preliminary Results

- 2^{20} experiments (m=3 rounds; no byte shuffles in merging)
  - Arbitrary input (Vortex spec) with random 2^{20} bit perturbations
- Computed differentials
  - Differential = HASH (DOC perturbed) XOR HASH (DOC)
- Computed the number of collisions
  - Should be 0 for a good hash
- Results:

```
number of bytes in vortex.doc read: 247808
vortex hashes computed successfully
no collisions occurred for any vortex hash in 1048576 experiments
standard deviation of vortex3 differentials 0.999289
standard deviation of sha256  differentials 0.999676
```

- Results provide indications regarding
  - Collision resistance of Vortex
  - The randomness of the output bit differentials
    - (comparable with SHA256)

Performance

Performance – Use Intel's New Instructions

AESENC xmm1, xmm2/m128

Tmp:= xmm1;

Round Key:= xmm2/m128;

Tmp:= Shift Rows (Tmp);

Tmp:= Substitute Bytes (Tmp);

Tmp:= Mix Columns (Tmp);

xmm1:= Tmp xor Round Key

AESENCLAST xmm1, xmm2/m128

Tmp:= xmm1;

Round Key:= xmm2/m128;

Tmp:= Shift Rows (Tmp);

Tmp:= Substitute Bytes (Tmp);

xmm1:= Tmp xor Round Key

PCLMULQDQ xmm1, xmm2/m128, imm8

xmm1 – [xH : xL], xmm2 = [yH : yL]

imm8 = 00h: dst = CLMUL(XL, YL)

imm8 = 01h: dst = CLMUL(XH, YL)

imm8 = 10h: dst = CLMUL(XL,YH)

imm8 = 11h: dst = CLMUL(XH,YH)

xmm1 = dst

Combinations to Isolate AES Transforms

- All AES transformations can be isolated
  - Shift Rows, Substitute Bytes, Mix Columns, Inverse Shift Rows, Inverse Substitute Bytes, Inverse Mix Columns
- Isolating SubBytes (SBox)
  - X = PSHUFB (State, `0306090C0F0205080B0E0104070A0D00`)
  - Y = AESENCLAST (X, 0)

    ➔ Y = SubBytes (State)
- ➔ Fast $\tilde{A}_K$ (X)

tmp = AESRound (X, RK1);
tmp = AESRound (tmp, RK2);
tmp = AESRound (tmp, RK3);
*RK*1 ← Perm(SBox(*K* ⊞ *RC*1))
*RK*2 ← Perm(SBox(*RK*1 ⊞ *RC*2))
*RK*3 ← Perm(SBox(*RK*2 ⊞ *RC*3))

Vortex Expected Performance

Expected Performance: ~1.5 cycles/byte

SHA-256 performance: ~20 cycles/byte

Some Variants

- 512 bits digest (using Rijndale 256 bits blocks)
- Improve the quality of merging
  - Strengthen the diffusion properties of the key schedule
  - Repeat sub-block as part of the mode of operation
  - Strengthen the diffusion in the merging function (full AES for merging)
  - Choose larger m
  - Enhance the key schedule

Thank You