

BLAKE and 256-bit advanced vector extensions

Samuel Neves¹ Jean-Philippe Aumasson²

¹University of Coimbra, Portugal

²NAGRA, Switzerland

The Third SHA-3 Candidate Conference

BLAKE

- ▶ Main bottleneck is the keyed permutation
- ▶ State: 4×4 matrix of 32- or 64-bit words

v_0, \dots, v_{15}

$$\begin{array}{ll} G_0(v_0, v_4, v_8, v_{12}) & G_1(v_1, v_5, v_9, v_{13}) \\ G_2(v_2, v_6, v_{10}, v_{14}) & G_3(v_3, v_7, v_{11}, v_{15}) \\ G_4(v_0, v_5, v_{10}, v_{15}) & G_5(v_1, v_6, v_{11}, v_{12}) \\ G_6(v_2, v_7, v_8, v_{13}) & G_7(v_3, v_4, v_9, v_{14}) \end{array}$$

- ▶ 14 (or 16) of these per compression
- ▶ 4-way parallelism

BLAKE

BLAKE-256's G_i looks like this:

$$a \leftarrow a + b + (m_{\sigma_r[2i]} \oplus u_{\sigma_r[2i+1]})$$

$$d \leftarrow (d \oplus a) \ggg 16$$

$$c \leftarrow c + d$$

$$b \leftarrow (b \oplus c) \ggg 12$$

$$a \leftarrow a + b + (m_{\sigma_r[2i+1]} \oplus u_{\sigma_r[2i]})$$

$$d \leftarrow (d \oplus a) \ggg 8$$

$$c \leftarrow c + d$$

$$b \leftarrow (b \oplus c) \ggg 7$$

BLAKE

BLAKE-512's G_i looks like this:

$$a \leftarrow a + b + (m_{\sigma_r[2i]} \oplus u_{\sigma_r[2i+1]})$$

$$d \leftarrow (d \oplus a) \ggg 32$$

$$c \leftarrow c + d$$

$$b \leftarrow (b \oplus c) \ggg 25$$

$$a \leftarrow a + b + (m_{\sigma_r[2i+1]} \oplus u_{\sigma_r[2i]})$$

$$d \leftarrow (d \oplus a) \ggg 16$$

$$c \leftarrow c + d$$

$$b \leftarrow (b \oplus c) \ggg 11$$

SIMD BLAKE

Put v_0, v_4, v_8, v_{12} in SIMD register r_0

Put v_1, v_5, v_9, v_{13} in SIMD register r_1

...

Perform single G call over r_0, r_1, r_2, r_3

Put v_4, v_5, v_{10}, v_{15} in SIMD register r_0

Put v_1, v_6, v_{11}, v_{12} in SIMD register r_1

...

Perform single G call over r_0, r_1, r_2, r_3

Anatomy of SIMD BLAKE

function ROUND(*r*)

```
m0..3 = LoadMsg(r)           ; m $\sigma_r[2i]$   $\oplus$  u $\sigma_r[2i+1]$   
state = G(state, m0..1)     ; G0, G1, G2, G3  
state = Diag(state)           ; Diagonalize  
state = G(state, m2..3)     ; G4, G5, G6, G7  
state = Undiag(state)        ; Undiagonalize
```

end function

Anatomy of SIMD BLAKE

function ROUND(r)

$m_{0..3} = \text{LoadMsg}(r)$

$state = \mathbf{G}(state, m_{0..1})$

$state = \text{Diag}(state)$

$state = \mathbf{G}(state, m_{2..3})$

$state = \text{Undiag}(state)$

end function

; $m_{\sigma_r[2i]} \oplus u_{\sigma_r[2i+1]}$

; $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$

; Diagonalize

; $\mathbf{G}_4, \mathbf{G}_5, \mathbf{G}_6, \mathbf{G}_7$

; Undiagonalize

Anatomy of SIMD BLAKE

function ROUND(r)

$m_{0..3} = \text{LoadMsg}(r)$

$state = \mathbf{G}(state, m_{0..1})$

$state = \text{Diag}(state)$

$state = \mathbf{G}(state, m_{2..3})$

$state = \text{Undiag}(state)$

end function

; $m_{\sigma_r[2i]} \oplus u_{\sigma_r[2i+1]}$

; $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$

; Diagonalize

; $\mathbf{G}_4, \mathbf{G}_5, \mathbf{G}_6, \mathbf{G}_7$

; Undiagonalize

Anatomy of SIMD BLAKE

function ROUND(r)

$m_{0..3} = \text{LoadMsg}(r)$

$state = \mathbf{G}(state, m_{0..1})$

$state = \text{Diag}(state)$

$state = \mathbf{G}(state, m_{2..3})$

$state = \text{Undiag}(state)$

end function

; $m_{\sigma_r[2i]} \oplus u_{\sigma_r[2i+1]}$

; $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$

; **Diagonalize**

; $\mathbf{G}_4, \mathbf{G}_5, \mathbf{G}_6, \mathbf{G}_7$

; **Undiagonalize**

Anatomy of SIMD BLAKE

function ROUND(r)

$m_{0..3} = \text{LoadMsg}(r)$

$state = \mathbf{G}(state, m_{0..1})$

$state = \text{Diag}(state)$

$state = \mathbf{G}(state, m_{2..3})$

$state = \text{Undiag}(state)$

end function

; $m_{\sigma_r[2i]} \oplus u_{\sigma_r[2i+1]}$

; $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$

; Diagonalize

; $\mathbf{G}_4, \mathbf{G}_5, \mathbf{G}_6, \mathbf{G}_7$

; Undiagonalize

Anatomy of SIMD BLAKE

function ROUND(*r*)

*m*_{0..3} = LoadMsg(*r*)

state = G(*state*, *m*_{0..1})

state = Diag(*state*)

state = G(*state*, *m*_{2..3})

state = Undiag(*state*)

end function

; *m* _{$\sigma_r[2i]$} \oplus *u* _{$\sigma_r[2i+1]$}

; G₀, G₁, G₂, G₃

; Diagonalize

; G₄, G₅, G₆, G₇

; Undiagonalize

Anatomy of SIMD BLAKE

function ROUND(r)

$m_{0..3} = \text{LoadMsg}(r)$; $m_{\sigma_r[2i]} \oplus u_{\sigma_r[2i+1]}$

$state = \mathbf{G}(state, m_{0..1})$; $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$

$state = \mathbf{Diag}(state)$; **Diagonalize**

$state = \mathbf{G}(state, m_{2..3})$; $\mathbf{G}_4, \mathbf{G}_5, \mathbf{G}_6, \mathbf{G}_7$

$state = \mathbf{Undiag}(state)$; **Undiagonalize**

end function

Timeline of SIMD BLAKE

BLAKE-256

2008 sse2
2009 ssse3
2010 sse41
2011 vect128
2012 avx, xop

BLAKE-512

2008 sse2
2009 ssse3
2011 vect128
2012 sse41, avx,
xop

AVX and XOP

AVX

- ▶ Sandy Bridge, 2011
- ▶ Extends 128-bit XMM to 256-bit YMM registers
- ▶ Non-destructive operations
- ▶ Floating-point only

XOP

- ▶ Bulldozer, 2011
- ▶ Integer rotations
- ▶ Advanced byte shuffles
- ▶ Integer MAD

AVX2

AVX2

Haswell, 2013

Extends AVX to the integers

Gather/Scatter instructions

More cross-lane instructions

Useful *new* instructions:

VPERMD

VPERMQ

VPGATHERDD

VPGATHERDQ

BLAKE-512 and AVX2

AVX2 enables same SIMD strategy as
BLAKE-256

VPSHUFD → VPERMQ

Multi-op permutations → VPGATHERDQ

BLAKE-512 and AVX2: message load

Option 1: Copy BLAKE-256's approach

Up to 12 instructions per message load

Some useful SSE4.1 instructions still not in AVX2

Option 2: Use VPGATHERDQ

```
vpcmpeqq   ymm14, ymm14, ymm14           ; set mask to 1111..11
vmovdqa   xmm8, [perm + 00]              ; permutation indices
vpgatherdq ymm4, [rsp + 8*xmm8], ymm14    ; permute from stack
...
vpxor     ymm4, ymm4, [const_z + 00]     ; xor with constant
```

BLAKE-512 and AVX2: G_i

```
vpaddq ymm0, ymm0, ymm4      ;  $a + (m_{\sigma_r[2i]} \oplus u_{\sigma_r[2i+1]})$   
vpaddq ymm0, ymm0, ymm1      ;  $a + b$   
vpxor  ymm3, ymm3, ymm0      ;  $d \oplus a$   
vpslufd ymm3, ymm3, 10110001b ;  $d \ggg 32$ 
```

```
vpaddq ymm2, ymm2, ymm3      ;  $c + d$   
vpxor  ymm1, ymm1, ymm2      ;  $b \oplus c$   
vpsllq ymm8, ymm1, 64-25     ;  
vpsrlq ymm1, ymm1, 25        ;  
vpxor  ymm1, ymm1, ymm8      ;  $b \ggg 25$ 
```

```
vpaddq ymm0, ymm0, ymm5      ;  $a + (m_{\sigma_r[2i+1]} \oplus u_{\sigma_r[2i]})$   
vpaddq ymm0, ymm0, ymm1      ;  $a + b$   
vpxor  ymm3, ymm3, ymm0      ;  $d \oplus a$   
vpslufb ymm3, ymm3, ymm15     ;  $d \ggg 16$ 
```

```
vpaddq ymm2, ymm2, ymm3      ;  $c + d$   
vpxor  ymm1, ymm1, ymm2      ;  $b \oplus c$   
vpsllq ymm8, ymm1, 64-11     ;  
vpsrlq ymm1, ymm1, 11        ;  
vpxor  ymm1, ymm1, ymm8      ;  $b \ggg 11$ 
```

BLAKE-512 and AVX2: performance

Pessimistic assumptions

- Message loading consumes 5 cycles

- Single cycle operations, 3-cycle odd rotations

- 6.63 cycles per byte

Optimistic

- Message loading can be run fully in parallel with G

- Single-cycle instructions, 3-cycle odd rotations

- 4.00 cycles per byte

BLAKE-256 and AVX, XOP

Non-destructive syntax greatly reduces μop count

AVX allows storing data in upper 128 bits of YMM registers

XOP adds native rotations(!)

VPPERM useful in message loads

BLAKE-256 and AVX2

Enables faster tree modes

Also useful for message loads in non-tree mode

```
vpcmpeqd   ymm12, ymm12, ymm12
vmovdqa    ymm8, [perm + 00]
vpgatherdd ymm4, [ymm8*4+rsp], ymm12

vpcmpeqd   ymm13, ymm13, ymm13
vmovdqa    ymm9, [perm + 32]
vpgatherdd ymm6, [ymm9*4+rsp], ymm13

vpxor      ymm4, ymm4, [const_z + 00]
vpxor      ymm6, ymm6, [const_z + 32]
```

BLAKE-256 and AVX2

Enables faster tree modes

Also useful for message loads in non-tree mode

```
vmovdqa ymm8, [perm0 + 00]
vmovdqa ymm9, [perm0 + 32]
vpermd ymm4, ymm8, ymm10
vpermd ymm5, ymm9, ymm11
vpblendd ymm4, ymm4, ymm5, 01111101b
```

```
vmovdqa ymm8, [perm0 + 64]
vmovdqa ymm9, [perm0 + 96]
vpermd ymm6, ymm8, ymm10
vpermd ymm7, ymm9, ymm11
vpblendd ymm6, ymm6, ymm7, 00010100b
```

```
vpxor ymm4, ymm4, [const_z + 00]
vpxor ymm6, ymm6, [const_z + 32]
```

BLAKE-256 and AVX2: performance

Unlike BLAKE-512, marginal improvement
Two compression functions at nearly no
extra cost
AVX and XOP have more direct effect

Message caching

10 distinct permutations

14 (resp 16) rounds

Reuse permutations from $r - 10$

Does not improve nor degrade performance

Results

AVX

7.62 cpb (Sandy Bridge)

AVX2

We'll only know in 2013

Estimated to range from 4 to 6.63 cpb

Updated numbers (20120310)

blake256

mangetsu: 7.49 cpb

hydra6: 11.83 cpb

blake512

mangetsu: 5.64 cpb

hydra6: 6.88 cpb

Compare to elroy (20110106)

blake256: 8.25 cpb

blake512: 7.93 cpb