SIMD Is a Message Digest

Gaëtan Leurent, Pierre-Alain Fouque, Charles Bouillaguet

École Normale Supérieure
Paris, France

http://www.di.ens.fr/~leurent/simd.html

First SHA-3 Conference
Main Features of SIMD

- **Security**
  - Strong message expansion
  - Proof of security against differential cryptanalysis

- **Parallelism**
  - Small scale parallelism (inside the compression function):
    - good for hardware / software with SIMD instructions
  - Can use two cores: message expansion / compression

- **Performance**
  - Very good on high-end desktops: 11 cycles/byte on Core2
  - Good if SIMD instructions are available:
    - SSE on x86, Altivec on PowerPC, IwMMXt on ARM, VIS on SPARC...
  - Drawback: no portable efficient implementation.
General Design

- Merkle-Damgård-like iteration
- Davies-Meyer-like compression function
- Feistel-based block cipher

Two versions:

<table>
<thead>
<tr>
<th></th>
<th>Message block size $m$</th>
<th>Internal state size $p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD-256</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>SIMD-512</td>
<td>1024</td>
<td>1024</td>
</tr>
</tbody>
</table>

can be truncated (e.g. SIMD-224, SIMD-384)
Outline

Introduction

Description
  Mode of operation
  Compression Function
  Message Expansion

Security
  Resistance to Differential Cryptanalysis

Implementation
  Performance
The iteration mode is based on ChopMD (a.k.a. wide pipe).

- Pad with zeros
- Use the message length as input of the last block: quite constrained, kind of blank round
- Tweaked final compression function (i.e. prefix-free encoding)
- Security proof: indifferentiable up to $2^n$
How to build a compression function?
Two inputs: $H_{i-1}$ hard to control / $M$ easy to control

Davies-Meyer:

$$H_i = E_M(H_{i-1}) \oplus H_{i-1}$$

- differential attack on $C$
- $\rightsquigarrow$ related key attack on $E$

- Message expansion can reduce control over $M$

Matyas-Meyer-Oseas:

$$H_i = E_{H_{i-1}}(M) \oplus M$$

- differential attack on $C$
- $\rightsquigarrow$ differential attacks $E$
How to build a compression function?

Two inputs: $H_{i-1}$ hard to control / $M$ easy to control

Davies-Meyer:

$$H_i = E_M(H_{i-1}) \oplus H_{i-1}$$

- differential attack on $C$
  $\implies$ related key attack on $E$
- Message expansion
  can reduce control over $M$

Matyas-Meyer-Oseas:

$$H_i = E_{H_{i-1}}(M) \oplus M$$

- differential attack on $C$
  $\implies$ differential attacks $E$
The Compression Function

- Modified Davies-Meyer mode.
  - XOR $M$ in the beginning: *no message modifications*
  - Use some more Feistel rounds as the feed-forward: *avoids some fixed points and multiblock attacks*
  - Same security proofs as DM: *good if $E$ if good*

- Feistel-based cipher

- Strong message expansion
The Feistel Round

- 4 parallel Feistel ladders (8 for SIMD-512) with 32 bit words
- 4 (expanded) message words enter each round
- Interaction between the Feistel ladders via the permutations $p^{(i)}$
- Constants hidden in the message expansion

\[
A_j^{(i)} = \left( D_j^{(i-1)} \oplus W_j^{(i)} \oplus \phi^{(i)}(A_j^{(i-1)}, B_j^{(i-1)}, C_j^{(i-1)}) \right) \ll s^{(i)} \oplus \left( A_j^{(i-1)} \right) \ll r^{(i)}
\]

\[
B_j^{(i)} = A_j^{(i-1)} \ll r^{(i)}
\]

\[
C_j^{(i)} = B_j^{(i-1)}
\]

\[
D_j^{(i)} = C_j^{(i-1)}
\]
Round Parameters

- Rotations and Boolean functions:

<table>
<thead>
<tr>
<th>$\phi^{(i)}$</th>
<th>$r^{(i)}$</th>
<th>$s^{(i)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>$\pi_0$</td>
<td>$\pi_1$</td>
</tr>
<tr>
<td>IF</td>
<td>$\pi_1$</td>
<td>$\pi_2$</td>
</tr>
<tr>
<td>IF</td>
<td>$\pi_2$</td>
<td>$\pi_3$</td>
</tr>
<tr>
<td>IF</td>
<td>$\pi_3$</td>
<td>$\pi_0$</td>
</tr>
<tr>
<td>MAJ</td>
<td>$\pi_0$</td>
<td>$\pi_1$</td>
</tr>
<tr>
<td>MAJ</td>
<td>$\pi_1$</td>
<td>$\pi_2$</td>
</tr>
<tr>
<td>MAJ</td>
<td>$\pi_2$</td>
<td>$\pi_3$</td>
</tr>
<tr>
<td>MAJ</td>
<td>$\pi_3$</td>
<td>$\pi_0$</td>
</tr>
</tbody>
</table>

- Permutations: chosen for maximal diffusion

\[
p(j) = j + 1
\]

\[
p(j) = j + 2
\]
Round Parameters

- Rotations and Boolean functions:

<table>
<thead>
<tr>
<th>( \phi(i) )</th>
<th>( r(i) )</th>
<th>( s(i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>( \pi_0 )</td>
<td>( \pi_1 )</td>
</tr>
<tr>
<td>IF</td>
<td>( \pi_1 )</td>
<td>( \pi_2 )</td>
</tr>
<tr>
<td>IF</td>
<td>( \pi_2 )</td>
<td>( \pi_3 )</td>
</tr>
<tr>
<td>IF</td>
<td>( \pi_3 )</td>
<td>( \pi_0 )</td>
</tr>
<tr>
<td>MAJ</td>
<td>( \pi_0 )</td>
<td>( \pi_1 )</td>
</tr>
<tr>
<td>MAJ</td>
<td>( \pi_1 )</td>
<td>( \pi_2 )</td>
</tr>
<tr>
<td>MAJ</td>
<td>( \pi_2 )</td>
<td>( \pi_3 )</td>
</tr>
<tr>
<td>MAJ</td>
<td>( \pi_3 )</td>
<td>( \pi_0 )</td>
</tr>
</tbody>
</table>

- Permutations: chosen for maximal diffusion

\[
p(j) = j + 1
\]

\[
p(j) = j + 2
\]
The Message Expansion

<table>
<thead>
<tr>
<th>Message block</th>
<th>Expanded message</th>
<th>Minimal distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD-256</td>
<td>512 bits</td>
<td>4096 bits</td>
</tr>
<tr>
<td>SIMD-512</td>
<td>1024 bits</td>
<td>8192 bits</td>
</tr>
</tbody>
</table>

- Provides resistance to differential attack
- Based on (error correcting) codes with a good minimal distance
- Concatenated code:
  - outer code gives a high word distance
  - inner code gives a high bit distance
Reed-Solomon code

- Interpret the input \((k\text{ words})\) as a polynomial of degree \(k - 1\) over some finite field
- Evaluate on \(n\) points \((n > k)\)
- **MDS code**: minimal distance \(n - k + 1\)

<table>
<thead>
<tr>
<th>SIMD</th>
<th>(k)</th>
<th>(n)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMD-256</td>
<td>64</td>
<td>128</td>
<td>65</td>
</tr>
<tr>
<td>SIMD-512</td>
<td>128</td>
<td>256</td>
<td>129</td>
</tr>
</tbody>
</table>

- **Efficiency:**
  - Compute with an FFT algorithm
  - Use the field \(\mathbb{F}_{257}\)
- Add a constant part: affine code
**Inner code**

We encode the output words of the NTT twice, through two different inner codes.

Very efficient codes, with a single 16-bit multiplication.

\[ I_{185} : \mathbb{F}_{257} \mapsto \mathbb{Z}_{2^{16}} \]
\[ x \rightarrow 185 \boxtimes \tilde{x} \quad \text{where} \quad -128 \leq \tilde{x} \leq 128 \quad \text{and} \quad \tilde{x} = x \pmod{257} \]

\[ I_{233} : \mathbb{F}_{257} \mapsto \mathbb{Z}_{2^{16}} \]
\[ x \rightarrow 233 \boxtimes \tilde{x} \quad \text{where} \quad -128 \leq \tilde{x} \leq 128 \quad \text{and} \quad \tilde{x} = x \pmod{257} \]

The magic constants 185 and 233 give a minimal distance of 4 bits. (also for signed difference)
Security of SIMD

- The mode of operation is indifferentiable.
- No generic multicollision attack, second-preimage on long messages, or herding attack.
- Any attack has to use some property of the block cipher.
- The most obvious property is to find differential trails.
Security Proof: Attacker goal

We model a differential attacker:

**Attacker game**

- Choose a message difference $\Delta$
- Build a differential path $u \leadsto v$
- Find a message $M$ s.t. $(M, M + \Delta)$ follows the path

At each step there is a probability $p$ that the path is followed \textit{i.e.} there are $c$ conditions, $c = -\log_2(p)$.

We want to show that $c \geq 128$. 
Two possible differentials:

- **XOR difference**: specifies which bits are modified
  - Easy to use
  - No condition for carry on bit 31
    (limited number due to the inner code)

- **Signed difference**: specifies which bits go up or down
  - More powerful:
    Used by Wang et al. to break MD4, MD5, SHA-1, HAVAL, ...
  - No condition when differences cancel out
  - Less conditions on the Boolean functions
  - Need a condition for the sign of bit 31
State Differences

- We consider a single isolated difference bit in the state.
- One condition to control the carry when the difference is introduced
- Three conditions for the Boolean functions
Security Proof: Attacker game

We will ask the adversary to play an easier game:

Simplified adversary

- You have 520 differences in the expanded message ($\delta W$)
- You want to get rid of them by placing differences in the state ($\delta A$):
  - Each $\delta A$ can consume some $\delta W$
  - But it costs you some conditions

The adversary is looking for a set of $\delta A$’s with a good exchange rate. He wins if the rate is less than $1/4$. 
Adversary I: No control over the message differences

Adversary I

1. Choose a message difference of minimal weight
2. Find a path connecting the $\delta W$’s

If the message difference has no other property,
Most of the $\delta W$ will introduce a $\delta A$, i.e. 4 conditions.

Realistic if optimal message pairs (minimal weight difference) are hard to find.

Exchange rate: 4/1. FAIL. ($p \approx 2^{-2048}$)

Lesson: the adversary need some control over the extended message.
Adversary II: Local Collisions

**Adversary II**

1. Choose a set $\delta A$
2. Use the neighbours of this $\delta A$ as $\delta W$

If the state difference are isolated, $c \approx 4\delta A$.

Realistic if optimal message pairs are not so easy to find.

$\delta W \leq 6\delta A$

Exchange rate: 4/6. FAIL. ($p \approx 2^{-340}$)

**Lesson:** the adversary needs to combine local collisions.
Adversary III: Combining Local Collisions

With a signed difference, many conditions can be avoided when two differences enters the same $\phi$.

Exchange rate as low as $1/4.5$. WIN? ($p \approx 2^{-113}$)

We expect that it is impossible to choose a possible $\delta W$ and a matching $\delta A$ that achieve this exchange rate.

Can we prove it?

We modelled this game as a linear integer program.

The solver proved that there is no solution with less than 130 conditions (and counting).
Adversary III: Combining Local Collisions

With a signed difference, many conditions can be avoided when two differences enters the same $\phi$.

Exchange rate as low as $1/4.5$. WIN? ($p \approx 2^{-113}$)

We expect that it is impossible to choose a possible $\delta W$ and a matching $\delta A$ that achieve this exchange rate.

Can we prove it?

We modelled this game as a linear integer program.

The solver proved that there is no solution with less than 130 conditions (and counting).
Adversary III: Combining Local Collisions

With a signed difference, many conditions can be avoided when two differences enters the same $\phi$.

Exchange rate as low as $1/4.5$. WIN? ($p \approx 2^{-113}$)

We expect that it is impossible to choose a possible $\delta W$ and a matching $\delta A$ that achieve this exchange rate.

Can we prove it?

We modelled this game as a linear integer program.

The solver proved that there is no solution with less than 130 conditions (and counting).
Proof summary

The adversary:

- Chooses the message difference and the \textit{expanded} message difference independently
- Can place the differences arbitrarily in the inner code
- Uses a signed difference

His optimal strategy:

- Use only local collisions (no error propagation)
- Locate the state differences next to each other to avoid most conditions.

Then, any differential path has \textbf{at least 130 conditions}. (that includes pseudo-near-collision paths)
The NTT and the Feistel ladder can be parallelized using SIMD instructions.

- Single Instruction, Multiple Data

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>A + B</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

- Available on most architectures:
  - x86 MMX (64-bit registers), SSE (128-bit registers)
  - PPC Altivec (128-bit registers)
  - ARM IwMMXt (64-bit registers)
  - Sparc VIS (64-bit registers)
Performance Overview

- Message expansion vs. Feistel: 50/50
- No need for 64-bit arithmetic
- Efficient on some embedded architectures: ARM Xscale, x86 Atom
- About 80% of the throughput of SHA-1 with a good SIMD unit (Core2, Atom, G4)
- SIMD units are improved in each generation of processors
# Performance in cycle/byte

<table>
<thead>
<tr>
<th>Architecture</th>
<th>SHA-1/256/512</th>
<th>Scalar</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core2 32 bits</td>
<td>11 21 63</td>
<td>90 118</td>
<td>12 13</td>
</tr>
<tr>
<td>Core2 64 bits</td>
<td>9 16 13</td>
<td>63 85</td>
<td>11 12</td>
</tr>
<tr>
<td>K10 32 bits</td>
<td>12 18 64</td>
<td>80 125</td>
<td>17</td>
</tr>
<tr>
<td>K10 64 bits</td>
<td>9 17 13</td>
<td>65 85</td>
<td>16</td>
</tr>
<tr>
<td>P4 32 bits</td>
<td>19 89 147</td>
<td>170 210</td>
<td>32 43</td>
</tr>
<tr>
<td>P4 64 bits</td>
<td>9 17 14</td>
<td>66 88</td>
<td>26</td>
</tr>
<tr>
<td>Atom 32 bits</td>
<td>24 46 133</td>
<td>220 280</td>
<td>25</td>
</tr>
<tr>
<td>Atom 64 bits</td>
<td>22 38 138</td>
<td>200 260</td>
<td>46</td>
</tr>
<tr>
<td>G4 32 bits</td>
<td>12 23 78</td>
<td>125 166</td>
<td>16 23</td>
</tr>
<tr>
<td>ARM</td>
<td>22 38 138</td>
<td>200 260</td>
<td></td>
</tr>
</tbody>
</table>

See eBASH for more accurate figures...

G. Leurent (ENS)
Conclusion

SIMD is

- Built on the MD/SHA legacy
- Secure (mode of operation and compression function)
- Fast on the reference platform: 11-13 cycles/byte