SIMD Instruction Set Extensions for KECCAK with Applications to SHA-3, Keyak and Ketje

Hemendra K. Rawat and Patrick Schaumont
Virginia tech, Blacksburg, USA
{hrawat, schaum}@vt.edu
Motivation

- Secure systems and protocols rely on a suite of cryptographic applications
  - Hashing, MAC, Encryption, PRNG, AEAD etc...

- Traditionally, different algorithms: SHA-1, SHA-2, AES, GMAC, HMAC

- Different Instruction set extensions: AES-NI, SHA and Carry-less multiplication
  - Vector extensions in Intel, ARM, SPARC and PowerPC

  KECCAK is the SHA-3
  - Successor of SHA-2

- Subset of the cryptographic primitive family KECCAK SPONGE
- Not just a Hash
- A Flexible Sponge for all symmetric cryptography

“A flexible SIMD Instruction set, for flexible KECCAK Sponge”
KECCAK Background
- Novelty Claims
- Design Principles
- Proposed Instruction Set Extensions

Results
- Performance
- Hardware Overhead

Conclusion
**KECCAK HASH (SHA-3)**

Hashing using **KECCAK Sponge Construction**

- Variable length input
- Output Digest
- Padding
- Rate ($r$ bits)
- Capacity ($c$ bits)
- State
- Cryptographic Permutation
- Absorbing
- Squeezing

Diagram showing the sponge construction process with variable length input, rate, capacity, state, cryptographic permutation, absorbing, and squeezing stages.
KECCAK-\(f\) in Nutshell

**KECCAK-\(f[b]\) Permutation**

```python
state [0 : b]
def keccakf[b]:
    for numRounds in range(0, maxRounds):
        theta(state)  # Add column parities
        rho(state)    # Rotate lanes
        pi(state)     # Transpose lanes
        chi(state)    # Add non-linearity
        iota(state)   # \(L_{00} \oplus K_{\text{round}}\)
```

\(b = 1600, 800, 400, 200, 100, 50, 25\) bits

\(b = (r +c)\) bits

```
0 0
```

```
x=0 x=1 x=2 x=3 x=4
```

```
0 1 2 3 4
```

\(x=0\) \(x=1\) \(x=2\) \(x=3\) \(x=4\)

\(y=0\) \(y=1\) \(y=2\) \(y=3\) \(y=4\)

\(L_{00}\) \(L_{01}\) \(L_{02}\) \(L_{03}\) \(L_{04}\)

\(L_{10}\) \(L_{11}\) \(L_{12}\) \(L_{13}\) \(L_{14}\)

\(L_{20}\) \(L_{21}\) \(L_{22}\) \(L_{23}\) \(L_{24}\)

\(L_{30}\) \(L_{31}\) \(L_{32}\) \(L_{33}\) \(L_{34}\)

\(L_{40}\) \(L_{41}\) \(L_{42}\) \(L_{43}\) \(L_{44}\)

\(z\)

\(x\)

\(y\)

\(plane\)

\(slice\)

\(State\)
Constructions with sponge: (a) Hash, (b) Message Authentication Code, (c) Keystream Generation. Construction with duplex: (d) Authenticated Encryption.
Novelty Claims

Previous work

- Custom-instruction designs for a 16 bit micro-controller for SHA-3 finalists, Constantin et al.
- Integration of 64 bit KECCAK(SHA-3) data path into a 32 bit LEON3, Wang et al.

This work

- Six new custom instructions for 128 bit SIMD unit.
- Multipurpose KECCAK: 1600, 800, 400 and 200 bits.
- Five KECCAK algorithms: SHA3-512 hash, LakeKEYAK, RiverKEYAK, KetjeSR and KetjeJR authenticated ciphers.
- Compatible with ARM NEON Instruction set
### Design Principles

<table>
<thead>
<tr>
<th>Portability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy integration into different processor architectures</td>
<td>RISC-like instruction format (2 input, 1 output operand)</td>
</tr>
<tr>
<td>No non-standard architectural features</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flexibility</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for multiple symmetric cryptographic applications</td>
<td>Hashing</td>
</tr>
<tr>
<td></td>
<td>MACing</td>
</tr>
<tr>
<td></td>
<td>Stream Ciphers</td>
</tr>
<tr>
<td></td>
<td>AEAD</td>
</tr>
<tr>
<td></td>
<td>PRNG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simplicity</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Small set of instructions</td>
<td>Low hardware Overhead</td>
</tr>
<tr>
<td></td>
<td>Simple operations like XOR, AND, NOT and rotations</td>
</tr>
<tr>
<td></td>
<td>Short critical path</td>
</tr>
</tbody>
</table>
### Design Principles

<table>
<thead>
<tr>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ <strong>Optimize Computation intensive part – KECCAK-{f,p} primitives</strong></td>
</tr>
<tr>
<td>▪ <strong>Partition dataflow graph into SIMD-like instruction patterns</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>▪ <strong>Minimize the schedule length of the graph</strong></td>
</tr>
<tr>
<td>▪ <strong>Optimize instruction shapes and functionality for</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Mapping KECCAK to Instructions

Data dependencies of \( \theta \)-effect

NEON Register Naming Convention
Qi = Quad word (128 bit)
Di = Double word (64 bit)
Si = Single word (32 bit)

NEON Instruction specifier
VADD.I32 q1, q2, q3 specifies q1, q2 and q3 have 4x32-bit integer
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Step</th>
<th>Description</th>
<th>Target Primitive</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| rl1x        | theta | Rotate Left 1 and XOR        | 1600, 800, 400, 200 | rl1x.u64 d2, d0, d1  
|             |      |                              |                  | rl1x.u32 s2, s0, s1  
|             |      |                              |                  | rl1x.u16 s2, s0, s1  
|             |      |                              |                  | rl1x.u8 s2, s0, s1    |
| kxorr64     | theta, rho & pi | XOR Rotate & Assign     | 1600             | kxorr64 d2, d0, d1, #i          |
| xorr        | theta, rho & pi | XOR Rotate & Assign      | 800, 400, 200    | xorrr.u32 s2, s0, s1, #i       
|             |      |                              |                  | xorrr.u16 s2, s0, s1, #i     
|             |      |                              |                  | xorrr.u8 s2, s0, s1, #i     |
| chi1        | chi  | chi step                    | 1600, 800, 400, 200 | chi1.u32 q2, q0, q1  
|             |      |                              |                  | chi1.u64 q2, q0, q1    |
| chi2        | chi  | chi step (last lane)        | 1600             | chi2.u64 d4, q0, q1            |
| chi3        | chi  | chi step (last lane)        | 800, 400, 200    | chi3.u32 s4, d0, d1            |
Use of KeccakCodePackge: http://keccak.noekeon.org/files.html

Hand Optimized Custom Instruction implementations:
- KECCAK-f,p {1600,800,400, 200}

Cross compiled GCC with KECCAK Instruction support

Timing simple CPU model in GEM5

Instructions in GEM5’s ISA description language

Simulation:
- Single core ARM CPU @ 1 GHz
- 32KB L1 I and D cache
- L2 cache of 2MB
Results: Performance

Performance gain between 1.4 - 2.6x over hand optimized assembly on ARMv7

Performance in instructions/byte for various KECCAK modes
## Results: Hardware Cost

### Gate equivalent estimates with UMC 90nm (4658 GE)

<table>
<thead>
<tr>
<th>CATEGORY NAME</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex-A9 Core</td>
<td>100%</td>
</tr>
</tbody>
</table>

- Cortex-A9 Core: 1869 GE
- Keccak Instructions: 1221 GE
- Kxorr64: 894 GE
- Kxor: 242 GE
- Chi1: 122 GE

**Percentage Breakdown:**
- 0% to 10%: 1 GE
- 10% to 20%: 1869 GE
- 20% to 30%: 1221 GE
- 30% to 40%: 894 GE
- 40% to 50%: 242 GE
- 50% to 60%: 122 GE

**Legend:**
- rl1x
- kxorr64
- xor
- chi1
- chi2
- chi3
Conclusion

- Analysis of the instruction set design space for KECCAK primitives
- Six custom instructions based on NEON instruction set in ARMv7
- Five different KECCAK applications: SHA3, LakeKEYAK, RiverKEYAK, KetjeSR and KetjeJR
- Performance gain between 1.4 - 2.6x over hand optimized assembly on ARMv7 at a hardware overhead of just 4658 GEs.
- Portability Aspects, Intel AVX, Generic 64/32 bit architectures, in the paper....
Questions
**Keccak (SHA-3)**

- **SHA3-224(M)**: $\text{Keccak}[448](M||01, 224)$
- **SHA3-256(M)**: $\text{Keccak}[512](M||01, 256)$
- **SHA3-384(M)**: $\text{Keccak}[768](M||01, 384)$
- **SHA3-512(M)**: $\text{Keccak}[1024](M||01, 512)$
- **SHAKE128(M, d)**: $\text{Keccak}[256](M||1111, d)$
- **SHAKE256(M, d)**: $\text{Keccak}[512](M||1111, d)$
Keccak

Source: http://keccak.noekeon.org/
KECCAK-\(f\) Permutation (2)

\[ R = \theta \circ \rho \circ \pi \circ \chi \circ \iota \]

Source: http://keccak.noekeon.org/
Proposed Instruction Set Extensions

Data dependencies of $\theta, \rho, \pi, \chi, \iota$ (one plane)
Proposed Instruction Set Extensions

Data dependencies X step (one plane)

http://keccak.noekeon.org/
Portability Aspects

- Based on instruction format, register width and the instruction encoding width.

Table 2: Feasibility of the proposed instructions on different platforms

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Intel AVX</th>
<th>64 bit Arch</th>
<th>32 bit Arch</th>
</tr>
</thead>
<tbody>
<tr>
<td>rl1x</td>
<td>✓</td>
<td>✓*</td>
<td>✓*</td>
</tr>
<tr>
<td>kxor64</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>xor</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>chi1</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>chi2</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>chi3</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

* Not all variants supported
Results: Hardware Cost

Table 1: Area, GE and Transistor count estimates for the proposed custom instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>$\mu^2$</th>
<th>Gate equivalent</th>
<th>Transistor Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>rl1x</td>
<td>1238</td>
<td>310</td>
<td>1238</td>
</tr>
<tr>
<td>kxorr64</td>
<td>7474</td>
<td>1869</td>
<td>7474</td>
</tr>
<tr>
<td>xorrr</td>
<td>4884</td>
<td>1221</td>
<td>4884</td>
</tr>
<tr>
<td>chi1</td>
<td>3576</td>
<td>894</td>
<td>3576</td>
</tr>
<tr>
<td>chi2</td>
<td>966</td>
<td>242</td>
<td>966</td>
</tr>
<tr>
<td>chi3</td>
<td>486</td>
<td>122</td>
<td>486</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>18624</strong></td>
<td><strong>4658</strong></td>
<td><strong>18624</strong></td>
</tr>
</tbody>
</table>

Typical Cortex-A9 CPU 3.8 Million Gates KECCAK Instructions have 0.1 % overhead
Cryptographic Instruction-Set
- Intel AES-NI, SHA-1,2, Carry-less multiplication extensions
- ARMv8 Crypto Instructions

Processor Architecture
- ARM (Soft IP)
- RISC-V (Open Architecture from UC, Berkeley)

Symmetric Cryptography Applications
- Hash, MAC, Encryption/Decryption, AEAD, PRNG

Universal Sponge Construction
- Hash: SHA-3 (Winner of NIST SHA-3 Competition)
- AEAD: LakeKeyak, Ketje AEAD
- PRNG, Stream Ciphers etc...