1001 Ways To Implement KECCAK

Guido Bertoni\textsuperscript{1} \quad Joan Daemen\textsuperscript{1} \\
Michaël Peeters\textsuperscript{2} \quad Gilles Van Assche\textsuperscript{1} \quad Ronny Van Keer\textsuperscript{1}

\textsuperscript{1}STMicroelectronics

\textsuperscript{2}NXP Semiconductors

Third SHA-3 candidate conference, Washington DC \\
March 22-23, 2012
Outline

1. KECCAK’s structure
2. How to cut a state
   - Cutting in lanes
   - Cutting in slices
   - Bit interleaving
3. High-end platforms
4. Protection against side-channel attacks
5. Closing words
Outline

1. KECCAK’s structure
2. How to cut a state
   - Cutting in lanes
   - Cutting in slices
   - Bit interleaving
3. High-end platforms
4. Protection against side-channel attacks
5. Closing words
KECCAK: the sponge construction

- One permutation for the SHA-3 competition:

  KECCAK-f[1600]

- Benefits of using a single permutation
  - Saving ROM code size / FPGA slices / ASIC area
  - No 32-bit/64-bit mismatch (see bit interleaving)
*But how to easily report speed vs security?*

- We report figures for Keccak\([r = 1024, c = 576]\)
- In general, throughput proportional to rate \(r\)

<table>
<thead>
<tr>
<th>Rate</th>
<th>Capacity</th>
<th>[NIST SP 800-57] Security strength</th>
<th>Relative performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1376</td>
<td>224</td>
<td>112</td>
<td>(\times 1.343)</td>
</tr>
<tr>
<td>1344</td>
<td>256</td>
<td>128</td>
<td>(\times 1.312)</td>
</tr>
<tr>
<td>1216</td>
<td>384</td>
<td>192</td>
<td>(\times 1.188)</td>
</tr>
<tr>
<td>1088</td>
<td>512</td>
<td>256</td>
<td>(\times 1.063)</td>
</tr>
<tr>
<td>1024</td>
<td>576</td>
<td>n/a</td>
<td>1.000</td>
</tr>
<tr>
<td>576</td>
<td>1024</td>
<td>n/a</td>
<td>(\div 1.778)</td>
</tr>
</tbody>
</table>
The state in KECCAK

- KECCAK-*f* operates on 3D state
- Efficient implementations based on state organization and transformations
The state in KECCAK

- KECCAK-$f$ operates on 3D state
- Efficient implementations based on state organization and transformations
The state in KECCAK

- KECCAK-f operates on 3D state
- Efficient implementations based on state organization and transformations
The state in KECCAK

- KECCAK-f operates on 3D state
- Efficient implementations based on state organization and transformations
The state in KECCAK

- KECCAK-\( f \) operates on 3D state
- Efficient implementations based on state organization and transformations
The step mappings of KECCAK-\(f\)
Outline

1. KECCAK’s structure
2. How to cut a state
   - Cutting in lanes
   - Cutting in slices
   - Bit interleaving
3. High-end platforms
4. Protection against side-channel attacks
5. Closing words
Not cutting it: straightforward hardware architecture

- Logic for one round + register for the state
  - very short critical path ⇒ high throughput
- Multiple rounds can be computed in a single clock cycle
  - 2, 3, 4 or 6 rounds in one shot
Lanes: straightforward software implementation

- Lanes fit in 64-bit registers
- Very basic operations required:
  - $\theta$ XOR and 1-bit rotations
  - $\rho$ rotations
  - $\pi$ just reading the correct words
  - $\chi$ XOR, AND, NOT
  - $\iota$ just a XOR
Lane-wise hardware architecture

- Basic processing unit + RAM
- Improvements over our co-processor:
  - 5 registers and barrel rotator
    [Kerckhof et al. CARDIS 2011]
  - 4-stage pipeline, \( \rho \) in 2 cycles, instruction-based parallel execution
    [San and At, ISJ 2012]
- Permutation latency in clock cycles:
  - From 5160, to 2137, down to 1062
Slice-wise hardware architecture

- Re-schedule the execution
  - $\chi$ and $\theta$ on blocks of slices
    [Jungk et al, ReConFig 2011]
- Suitable for compact FPGA or ASIC
- Performance-area trade-offs
  - Possible to select number of processed slices from 1 up to 32
    [VHDL on http://keccak.noekeon.org/]
Slice-wise hardware architecture

- Re-schedule the execution
  - \( \chi \) and \( \theta \) on blocks of slices
    [Jungk et al, ReConFig 2011]

- Suitable for compact FPGA or ASIC

- Performance-area trade-offs
  - Possible to select number of processed slices from 1 up to 32
    [VHDL on http://keccak.noekeon.org/]
Slice-wise hardware architecture

- Re-schedule the execution
  - $\chi$ and $\theta$ on blocks of slices
    [Jungk et al, ReConFig 2011]

- Suitable for compact FPGA or ASIC
- Performance-area trade-offs
  - Possible to select number of processed slices from 1 up to 32
    [VHDL on http://keccak.noekeon.org/]
Slice-wise hardware architecture

- Re-schedule the execution
  - $\chi$ and $\theta$ on blocks of slices
    [Jungk et al, ReConFig 2011]
- Suitable for compact FPGA or ASIC
- Performance-area trade-offs
  - Possible to select number of processed slices from 1 up to 32
    [VHDL on http://keccak.noekeon.org/]
Cutting the state in lanes or in slices?

- Both solutions are efficient, results for Virtex 5

<table>
<thead>
<tr>
<th>Architecture</th>
<th>T.put Mbit/s</th>
<th>Freq. MHz</th>
<th>Slices (+RAM)</th>
<th>Latency clocks</th>
<th>Efficiency Mbit/s/slice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lane-wise [1]</td>
<td>52</td>
<td>265</td>
<td>448</td>
<td>5160</td>
<td>0.12</td>
</tr>
<tr>
<td>Lane-wise [2]</td>
<td>501</td>
<td>520</td>
<td>151 (+3)</td>
<td>1062</td>
<td>3.32</td>
</tr>
<tr>
<td>Slice-wise [3]</td>
<td>813</td>
<td>159</td>
<td>372</td>
<td>200</td>
<td>2.18</td>
</tr>
<tr>
<td>High-Speed [4]</td>
<td>12789</td>
<td>305</td>
<td>1384</td>
<td>24</td>
<td>9.2</td>
</tr>
</tbody>
</table>

[1] Keccak Team, KECCAK implementation overview
[3] Jungk, Apfelbeck, ReConFig 2011 (scaled to $r = 1024$)
[4] GMU ATHENa (scaled to $r = 1024$)
Bit interleaving

- Ex.: map 64-bit lane to 32-bit words
  - $\rho$ seems the critical step
  - Even bits in one word
  - Odd bits in a second word
  - $\text{ROT}_{64} \leftrightarrow 2 \times \text{ROT}_{32}$

- Can be generalized
  - to 16- and 8-bit words

- Can be combined
  - with lane/slice-wise architectures
  - with most other techniques

[KECCAK impl. overview, Section 2.1]
Outline

1. KECCAK’s structure
2. How to cut a state
   - Cutting in lanes
   - Cutting in slices
   - Bit interleaving
3. High-end platforms
4. Protection against side-channel attacks
5. Closing words
1001 Ways To Implement Keccak

High-end platforms

SIMD and tree hashing

- Tree hashing is ...
  - attractive for exploiting multicore availability
  - already interesting on a single core

- Efficient evaluation of $2 \times \text{KECCAK-f}$ on latest CPUs
  - In eBASH: \texttt{keccakc512treed2} using SSE or AVX

$\approx 7 \text{ cycle} \cdot \text{core/byte}$ on Sandy Bridge [eBASH]
Instruction-level parallelism

- Improving CPUs via parallel execution units
- Degree of parallelism is intrinsic to the algorithm
- Parallelism for KECCAK transformations:
  - Up to 25 for $\chi$, $\rho$ and part of $\theta$
  - Minimum is 5 when computing $\theta$-effect
- For instance Itanium 2 versus Intel Core i7:
  - 6.02 cpb vs 11.48 cpb [eBASH]
Dedicated instructions

- Intel, AMD and ARM are adopting dedicated instructions for speeding-up cryptographic algorithms
- KECCAK can benefit of simple dedicated instructions:
  - Storing the state in 128/256-bit registers
  - XOR-AND-NOT for $\chi$
  - Rotate 64-bit words and assign

- *Can also benefit to other primitives!*

```
Reg A

<<< i

Reg B

<<< k

Reg C
```
Outline

1 KECCAK’s structure
2 How to cut a state
   - Cutting in lanes
   - Cutting in slices
   - Bit interleaving
3 High-end platforms
4 Protection against side-channel attacks
5 Closing words
Secure implementations

**Keyed modes** may require protected implementations

- KECCAK offers protection against:
  - timing or cache-miss attacks
  - no table look-ups
  - side channels (DPA)
  - efficient secret sharing thanks to degree-2 round function

[KECCAK impl. overview, Chapter 5]
Outline

1. KECCAK’s structure
2. How to cut a state
   - Cutting in lanes
   - Cutting in slices
   - Bit interleaving
3. High-end platforms
4. Protection against side-channel attacks
5. Closing words
Conclusions

- The state can be cut in many ways
  - Lane-wise or slice-wise (e.g., compact hardware)
  - Bit interleaving for low-end CPUs
- Good potential for improvements on high-end CPUs
  - Simple dedicated instructions
  - Instruction-level parallelism
  - SIMD instructions with 256-bit registers
- Very simple and efficient side channel protection
Some references

- **Keccak implementation overview** (version 3.1 or later)
- *Note on side-channel attacks and their countermeasures*, NIST hash forum 2009
- *Building power analysis resistant implementations of Keccak*, SHA-3 2010
- *Note on KECCAK parameters and usage*, NIST hash forum 2010

Software implementations

- Bernstein and Lange, *eBASH*
- Wenzel-Benner and Gräf, *XBX*

Hardware implementations on FPGA

- Kerckhof et al., CARDIS 2011
- Jungk and Apfelbeck, ReConFig 2011
- San and At, ISJ 2012
- ATHENa project

Hardware implementations on ASIC

- Henzen et al., CHES 2010
- Tillich et al., SHA-3 2010
- Guo et al., DATE 2012

http://keccak.noekeon.org/
Thank you!