KECCAK and the SHA-3 Standardization

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

\textsuperscript{1}STMicroelectronics
\textsuperscript{2}NXP Semiconductors

NIST, Gaithersburg, MD  
February 6, 2013
Outline

1. The beginning
2. The sponge construction
3. Inside KECCAK
4. Analysis underlying KECCAK
5. Applications of KECCAK, or sponge
6. Some ideas for the SHA-3 standard
Outline

1. The beginning
2. The sponge construction
3. Inside KECCAK
4. Analysis underlying KECCAK
5. Applications of KECCAK, or sponge
6. Some ideas for the SHA-3 standard
Cryptographic hash functions

Let $h : \{0, 1\}^* \rightarrow \{0, 1\}^n$

Input message $\rightarrow$ Digest

- **MD5**: $n = 128$ (Ron Rivest, 1992)
- **SHA-1**: $n = 160$ (NSA, NIST, 1995)
- **SHA-2**: $n \in \{224, 256, 384, 512\}$ (NSA, NIST, 2001)
Our beginning: RADIOGATÚN

- Initiative to design hash/stream function (late 2005)
  - rumours about NIST call for hash functions
  - forming of KECCAK Team
  - starting point: fixing PANAMA [Daemen, Clapp, FSE 1998]

- RADIOGATÚN [Keccak team, NIST 2nd hash workshop 2006]
  - more conservative than PANAMA
  - variable-length output
  - expressing security claim: non-trivial exercise

- Sponge functions [Keccak team, Ecrypt hash, 2007]
  - closest thing to a random oracle with a finite state
  - Sponge construction calling random permutation
From RADIOGATÚN to KECCAK

- **RADIOGATÚN confidence crisis (2007-2008)**
  - own experiments did not inspire confidence in RADIOGATÚN
  - neither did third-party cryptanalysis
    - [Bouillaguet, Fouque, SAC 2008] [Fuhr, Peyrin, FSE 2009]
  - follow-up design GNOBLIO went nowhere
  - NIST SHA-3 deadline approaching ...
  - U-turn: design a sponge with strong permutation $f$

- **KECCAK** [Keccak team, SHA-3, 2008]
Outline

1. The beginning
2. The sponge construction
3. Inside KECCAK
4. Analysis underlying KECCAK
5. Applications of KECCAK, or sponge
6. Some ideas for the SHA-3 standard
The sponge construction

More general than a hash function: arbitrary-length output

Calls a $b$-bit permutation $f$, with $b = r + c$
  - $r$ bits of rate
  - $c$ bits of capacity (security parameter)
Generic security of the sponge construction

- **RO-differentiating advantage** \( \leq N^2 / 2^{c+1} \)
  - \( N \) is number of calls to \( f \)
  - Proven in [Keccak team, Eurocrypt 2008]
  - As strong as a random oracle against attacks with \( N < 2^{c/2} \)

- **Bound assumes** \( f \) is **random** permutation
  - It covers generic attacks
  - ...but not attacks that exploit specific properties of \( f \)
Design approach

Hermetic sponge strategy
- Instantiate a sponge function
- Claim a security level of $2^{c/2}$

Mission
Design permutation $f$ without exploitable properties
How to build a strong permutation

- Build it as is an iterated permutation
- Like a block cipher
  - Sequence of identical rounds
  - Round consists of sequence of simple step mappings
- ...but not quite
  - No key schedule
  - Round constants instead of round keys
  - Inverse permutation need not be efficient
Criteria for a strong permutation

- Classical LC/DC criteria
  - Absence of large differential propagation probabilities
  - Absence of large input-output correlations

- Infeasibility of the CICO problem
  - Constrained Input Constrained Output
  - *Given partial input and partial output, find missing parts*

- Immunity to
  - Integral cryptanalysis
  - Algebraic attacks
  - Slide and symmetry-exploiting attacks
  - ...
Outline

1. The beginning
2. The sponge construction
3. Inside KECCAK
4. Analysis underlying KECCAK
5. Applications of KECCAK, or sponge
6. Some ideas for the SHA-3 standard
Keccak

- Instantiation of a *sponge function*
- the permutation *Keccak-f*
  - 7 permutations: \( b \in \{ 25, 50, 100, 200, 400, 800, 1600 \} \)
- Security-speed trade-offs using the same permutation, e.g.,
  - SHA-3 instance: \( r = 1088 \) and \( c = 512 \)
    - permutation width: 1600
    - security strength 256: post-quantum sufficient
  - Lightweight instance: \( r = 40 \) and \( c = 160 \)
    - permutation width: 200
    - security strength 80: same as SHA-1
The state: an array of $5 \times 5 \times 2^\ell$ bits

- $5 \times 5$ lanes, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- $(5 \times 5)$-bit slices, $2^\ell$ of them
The state: an array of $5 \times 5 \times 2^\ell$ bits

- $5 \times 5$ lanes, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- $(5 \times 5)$-bit slices, $2^\ell$ of them
The state: an array of $5 \times 5 \times 2^\ell$ bits

- 5 x 5 lanes, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- (5 x 5)-bit slices, $2^\ell$ of them
The state: an array of $5 \times 5 \times 2^\ell$ bits

- $5 \times 5$ lanes, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- $(5 \times 5)$-bit slices, $2^\ell$ of them
The state: an array of $5 \times 5 \times 2^\ell$ bits

- **5 × 5 lanes**, each containing $2^\ell$ bits (1, 2, 4, 8, 16, 32 or 64)
- **(5 × 5)-bit slices**, $2^\ell$ of them
\( \chi \), the nonlinear mapping in KECCAK-\( f \)

- “Flip bit if neighbors exhibit 01 pattern”
- Operates independently and in parallel on 5-bit rows
- Algebraic degree 2, inverse has degree 3
- LC/DC propagation properties easy to describe and analyze
$\theta'$, a first attempt at mixing bits

- Compute parity $c_{x,z}$ of each column
- Add to each cell parity of neighboring columns:

$$b_{x,y,z} = a_{x,y,z} \oplus c_{x-1,z} \oplus c_{x+1,z}$$
Diffusion of $\theta'$
Diffusion of $\theta'$ (kernel)
Diffusion of the inverse of $\theta'$
ρ for inter-slice dispersion

- We need diffusion between the slices ...
- ρ: cyclic shifts of lanes with offsets
  \[ i(i + 1)/2 \mod 2^\ell \]
- Offsets cycle through all values below \(2^\ell\)
Inside Keccak

\( \ell \) to break symmetry

- XOR of round-dependent constant to lane in origin
- Without \( \ell \), the round mapping would be symmetric
  - invariant to translation in the z-direction
- Without \( \ell \), all rounds would be the same
  - susceptibility to slide attacks
  - defective cycle structure
- Without \( \ell \), we get simple fixed points (000 and 111)
A first attempt at Keccak-f

- Round function: \( R = \iota \circ \rho \circ \theta' \circ \chi \)

- Problem: low-weight periodic trails by chaining:

\[
\begin{align*}
\chi &: \text{may propagate unchanged} \\
\theta' &: \text{propagates unchanged, because all column parities are 0} \\
\rho &: \text{in general moves active bits to different slices ...} \\
&\quad \text{...but not always}
\end{align*}
\]
Patterns in $Q'$ are $z$-periodic versions of patterns in $Q$. 

The Matryoshka property
$\pi$ for disturbing horizontal/vertical alignment

$$a_{x,y} \leftarrow a_{x',y'} \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$
A second attempt at **KECCAK-f**

- **Round function**: \( R = \iota \circ \pi \circ \rho \circ \theta' \circ \chi \)
- **Solves problem encountered before**:

\[ \begin{align*}
\text{\pi moves bits in same column to different columns!}
\end{align*} \]
Inside Keccak

Tweaking $\theta'$ to $\theta$
Inverse of $\theta$

- Diffusion from single-bit output to input very high
- Increases resistance against LC/DC and algebraic attacks
KECCAK-$f$ summary

- **Round function:**
  \[ R = \iota \circ \chi \circ \pi \circ \rho \circ \theta \]

- **Number of rounds:** $12 + 2\ell$
  - KECCAK-$f[25]$ has 12 rounds
  - KECCAK-$f[1600]$ has 24 rounds

- **Efficiency**
  - high level of parallelism
  - flexibility: bit-interleaving
  - software: competitive on wide range of CPU
  - dedicated hardware: very competitive
  - suited for protection against side-channel attack
Performance in software

- Faster than SHA-2 on all modern PC
- KeccakTree faster than MD5 on some platforms

<table>
<thead>
<tr>
<th>C/b</th>
<th>Algo</th>
<th>Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.79</td>
<td>keccakc256treed2</td>
<td>128</td>
</tr>
<tr>
<td>4.98</td>
<td>md5</td>
<td>&lt; 64</td>
</tr>
<tr>
<td>5.89</td>
<td>keccakc512treed2</td>
<td>256</td>
</tr>
<tr>
<td>6.09</td>
<td>sha1</td>
<td>&lt; 80</td>
</tr>
<tr>
<td>8.25</td>
<td>keccakc256</td>
<td>128</td>
</tr>
<tr>
<td>10.02</td>
<td>keccakc512</td>
<td>256</td>
</tr>
<tr>
<td>13.73</td>
<td>sha512</td>
<td>256</td>
</tr>
<tr>
<td>21.66</td>
<td>sha256</td>
<td>128</td>
</tr>
</tbody>
</table>

[eBASH, hydra6, http://bench.cr.yp.to/]
Efficient and flexible in hardware

From Kris Gaj’s presentation at SHA-3, Washington 2012:
Outline

1. The beginning
2. The sponge construction
3. Inside KECCAK
4. Analysis underlying KECCAK
5. Applications of KECCAK, or sponge
6. Some ideas for the SHA-3 standard
Our analysis underlying the design of KECCAK-f

- Presence of large input-output correlations
- Ability to control propagation of differences
  - Differential/linear trail analysis
  - Lower bounds for trail weights
  - Alignment and trail clustering
  - This shaped $\theta$, $\pi$ and $\rho$

- Algebraic properties
  - Distribution of $\#$ terms of certain degrees
  - Ability of solving certain problems (CICO) algebraically
  - Zero-sum distinguishers (third party)
  - This determined the number of rounds

- Analysis of symmetry properties: this shaped $\iota$

- See [KECCAK reference], [Ecrypt II Hash 2011], [FSE 2012]
## Third-party cryptanalysis of KECCAK

### Distinguishers on KECCAK-$f[1600]$  
<table>
<thead>
<tr>
<th>Rounds</th>
<th>Work</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>low CICO problem [Aumasson, Khovratovich, 2009]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>low cube testers [Aumasson, Khovratovich, 2009]</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$2^{491}$ unaligned rebound [Duc, Guo, Peyrin, Wei, FSE 2012]</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>$2^{1574}$ zero-sum [Duan, Lai, ePrint 2011]</td>
<td>[Boura, Canteaut, De Cannière, FSE 2011]</td>
</tr>
</tbody>
</table>

### Academic-complexity attacks on KECCAK
- **6-8 rounds:** second preimage [Bernstein, 2010]  
  - *slightly faster* than exhaustive search, but huge memory
- **attacks taking advantage of symmetry**
  - 4-round pre-images [Morawiecki, Pieprzyk, Srebrny, FSE 2013]
  - 5-rounds collisions [Dinur, Dunkelman, Shamir, FSE 2013]
## Third-party cryptanalysis of KECCAK

### Practical-complexity attacks on KECCAK

<table>
<thead>
<tr>
<th>Rounds</th>
<th>Attack Type</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>preimages and collisions</td>
<td>[Morawiecki, CC]</td>
</tr>
<tr>
<td>2</td>
<td>collisions</td>
<td>[Duc, Guo, Peyrin, Wei, FSE 2012 and CC]</td>
</tr>
<tr>
<td>3</td>
<td>40-bit preimage</td>
<td>[Morawiecki, Srebrny, 2010]</td>
</tr>
<tr>
<td>3</td>
<td>near collisions</td>
<td>[Naya-Plasencia, Röck, Meier, Indocrypt 2011]</td>
</tr>
<tr>
<td>4</td>
<td>key recovery</td>
<td>[Lathrop, 2009]</td>
</tr>
<tr>
<td>4</td>
<td>distinguishers</td>
<td>[Naya-Plasencia, Röck, Meier, Indocrypt 2011]</td>
</tr>
<tr>
<td>4</td>
<td>collisions</td>
<td>[Dinur, Dunkelman, Shamir, FSE 2012 and CC]</td>
</tr>
<tr>
<td>5</td>
<td>near-collisions</td>
<td>[Dinur, Dunkelman, Shamir, FSE 2012]</td>
</tr>
</tbody>
</table>

CC = Crunchy Crypto Collision and Preimage Contest
Observations from third-party cryptanalysis

- Extending distinguishers of KECCAK-f to KECCAK is not easy
- Effect of alignment on differential/linear propagation
  - **Strong**: low uncertainty in prop. along block boundaries
  - **Weak**: high uncertainty in prop. along block boundaries
  - Weak alignment in KECCAK-f limits feasibility of rebound attacks
- Effect of the inverse of the mixing layer $\theta$
  - $\theta^{-1}$ has very high average diffusion
  - Limits the construction of low-weight trails over more than a few rounds
Outline

1. The beginning
2. The sponge construction
3. Inside KECCAK
4. Analysis underlying KECCAK
5. Applications of KECCAK, or sponge
6. Some ideas for the SHA-3 standard
Applications of Keccak, or sponge

Regular hashing

- Electronic signatures
- Data integrity (shaXsum ...)
- Data identifier (Git, online anti-virus, peer-2-peer ...)
Salted hashing

- Randomized hashing (RSASSA-PSS)
- Password storage and verification (*Kerberos*, `/etc/shadow`)
Salted hashing

- Randomized hashing (RSASSA-PSS)
- Password storage and verification (*Kerberos*, /etc/shadow)
  - ...Can be as slow as you like it!
Mask generation function

- Key derivation function in SSL, TLS
- Full-domain hashing in public key cryptography
  - electronic signatures RSASSA-PSS [PKCS#1]
  - encryption RSAES-OAEP [PKCS#1]
  - key encapsulation methods (KEM)
Message authentication codes

- As a message authentication code
- Simpler than HMAC [FIPS 198]
  - Required for SHA-1, SHA-2 due to length extension property
  - No longer needed for sponge
Stream encryption

- As a stream cipher
  - Long output stream per IV: similar to OFB mode
  - Short output stream per IV: similar to counter mode
Single pass authenticated encryption

- Authentication and encryption in a single pass!
- Secure messaging (*SSL/TLS, SSH, IPSEC ...*)
Applications of Keccak, or sponge

The duplex construction

- **Generic security equivalent to Sponge** [Keccak Team, SAC 2011]
- **Applications include:**
  - Authenticated encryption: spongeWrap
  - Reseedable pseudorandom sequence generator
Reeseedable pseudorandom sequence generator

- **Defined in** [Keccak Team, CHES 2010] and [Keccak Team, SAC 2011]
- **Support for forward secrecy by** *forgetting* **in duplex:**

![Diagram of reseedable pseudorandom sequence generator](image)
Reseedable pseudorandom sequence generator

- Defined in [Keccak Team, CHES 2010] and [Keccak Team, SAC 2011]
- Support for forward secrecy by *forgetting* in duplex:

![Diagram](attachment:image.png)
Outline

1. The beginning
2. The sponge construction
3. Inside KECCAK
4. Analysis underlying KECCAK
5. Applications of KECCAK, or sponge
6. Some ideas for the SHA-3 standard
Some ideas for the SHA-3 standard
Capacity and security strength levels

Output length oriented approach

<table>
<thead>
<tr>
<th>Output length</th>
<th>Collision resistance</th>
<th>Pre-image resistance</th>
<th>Required capacity</th>
<th>Relative perf.</th>
<th>SHA-3 instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 160$</td>
<td>$s \leq 80$</td>
<td>$s \leq 160$</td>
<td>$c = 320$</td>
<td>$\times 1.250$</td>
<td>SHA3n160</td>
</tr>
<tr>
<td>$n = 224$</td>
<td>$s \leq 112$</td>
<td>$s \leq 224$</td>
<td>$c = 448$</td>
<td>$\times 1.125$</td>
<td>SHA3n224</td>
</tr>
<tr>
<td>$n = 256$</td>
<td>$s \leq 128$</td>
<td>$s \leq 256$</td>
<td>$c = 512$</td>
<td>$\times 1.063$</td>
<td>SHA3n256</td>
</tr>
<tr>
<td>$n = 384$</td>
<td>$s \leq 192$</td>
<td>$s \leq 384$</td>
<td>$c = 768$</td>
<td>$\div 1.231$</td>
<td>SHA3n384</td>
</tr>
<tr>
<td>$n = 512$</td>
<td>$s \leq 256$</td>
<td>$s \leq 512$</td>
<td>$c = 1024$</td>
<td>$\div 1.778$</td>
<td>SHA3n512</td>
</tr>
<tr>
<td>$n$</td>
<td>$s \leq n/2$</td>
<td>$s \leq n$</td>
<td>$c = 2n$</td>
<td>$\times \frac{1600-c}{1024}$</td>
<td></td>
</tr>
</tbody>
</table>

$s$: security strength level [NIST SP 800-57]

- These SHA-3 instances address
  - multiple security strengths each
  - levels outside of [NIST SP 800-57] range

- Performance penalty!
Security strength oriented approach

<table>
<thead>
<tr>
<th>Security strength</th>
<th>Collision resistance</th>
<th>Pre-image resistance</th>
<th>Required capacity</th>
<th>Relative perf.</th>
<th>SHA-3 instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>s = 80</td>
<td>n ≥ 160</td>
<td>n ≥ 80</td>
<td>c = 160</td>
<td>×1.406</td>
<td>SHA3c160</td>
</tr>
<tr>
<td>s = 112</td>
<td>n ≥ 224</td>
<td>n ≥ 112</td>
<td>c = 224</td>
<td>×1.343</td>
<td>SHA3c224</td>
</tr>
<tr>
<td>s = 128</td>
<td>n ≥ 256</td>
<td>n ≥ 128</td>
<td>c = 256</td>
<td>×1.312</td>
<td>SHA3c256</td>
</tr>
<tr>
<td>s = 192</td>
<td>n ≥ 384</td>
<td>n ≥ 192</td>
<td>c = 384</td>
<td>×1.188</td>
<td>SHA3c384</td>
</tr>
<tr>
<td>s = 256</td>
<td>n ≥ 512</td>
<td>n ≥ 256</td>
<td>c = 512</td>
<td>×1.063</td>
<td>SHA3c512</td>
</tr>
<tr>
<td>s</td>
<td>n ≥ 2s</td>
<td>n ≥ s</td>
<td>c = 2s</td>
<td>×\frac{1600-c}{1024}</td>
<td>SHA3[c=2s]</td>
</tr>
</tbody>
</table>

s: security strength level [NIST SP 800-57]

- These SHA-3 instances
  - are consistent with philosophy of [NIST SP 800-57]
  - provide a one-to-one mapping to security strength levels

- Higher efficiency
Choosing the capacity

### Ideas for discussion

1. **Let SHA-3 be a sponge**
   - Allow freedom in choosing $c$
   - Allow variable output length

2. **Decouple security and output length**
   - Set minimum capacity $c \geq 2s$ for [SP 800-57]’s level $s$

3. **Base naming scheme on security level**
   - For instance $SHA3c180$ for KECCAK [$c = 180$]

4. **For SHA-2-$n$ drop-in replacements, avoid slow instances**
   - Example option 1: $c = n$
   - Example option 2: $c = \min\{2n, 576\}$
   - Example option 3: $c = 576$
Choosing the capacity

Ideas for discussion

1. Let SHA-3 be a sponge
   - Allow freedom in choosing $c$
   - Allow variable output length

2. Decouple security and output length
   - Set minimum capacity $c \geq 2s$ for [SP 800-57]'s level $s$

3. Base naming scheme on security level
   - For instance $\text{SHA3c180}$ for KECCAK[$c = 180$]

4. For SHA-2-$n$ drop-in replacements, avoid slow instances
   - Example option 1: $c = n$
   - Example option 2: $c = \min\{2n, 576\}$
   - Example option 3: $c = 576$
Some ideas for the SHA-3 standard

Choosing the capacity

Ideas for discussion

1. Let SHA-3 be a sponge
   - Allow freedom in choosing $c$
   - Allow variable output length

2. Decouple security and output length
   - Set minimum capacity $c \geq 2s$ for [SP 800-57]'s level $s$

3. Base naming scheme on security level
   - For instance $\text{SHA3c180}$ for $\text{KECCAK}[c = 180]$

4. For SHA-2-$n$ drop-in replacements, avoid slow instances
   - Example option 1: $c = n$
   - Example option 2: $c = \min\{2n, 576\}$
   - Example option 3: $c = 576$
Choosing the capacity

Ideas for discussion

1. Let SHA-3 be a sponge
   - Allow freedom in choosing $c$
   - Allow variable output length

2. Decouple security and output length
   - Set minimum capacity $c \geq 2s$ for [SP 800-57]'s level $s$

3. Base naming scheme on security level
   - For instance $\text{SHA3c180}$ for $\text{KECCAK}[c = 180]$

4. For SHA-2-$n$ drop-in replacements, avoid slow instances
   - Example option 1: $c = n$
   - Example option 2: $c = \min\{2n, 576\}$
   - Example option 3: $c = 576$
Some ideas for the SHA-3 standard

Structuring the standard

Ideas for discussion

1. Standardize KECCAK-$f$, constructions and modes separately
   - Constructions and modes defined independently of KECCAK-$f$
   - Like block ciphers and their modes
     (It seems you have this in mind too.)

2. Propose a guideline for interfaces between these
Some ideas for the SHA-3 standard

Input formatting

Multiple instances of KECCAK

- $c_1 \neq c_2 \Rightarrow \text{KECCAK}[c = c_1]$ and $\text{KECCAK}[c = c_2]$ independent
- Joint security level determined by $\min\{c_1, c_2\}$

[KECCAK Team, SAC 2011]
Domain separation

Valid sponge input, rate- and mode-separated

Idea for discussion

1. Foresee domain separation from the start
   - To prevent potential clashes between different modes
   - If possible, anyone can define his/her domain
Example: domain separation with namespaces

- Basic idea: prefix input with namespace identifier (URI)
  - Payload syntax determined by namespace
  - Inspired from XML [http://www.w3.org/TR/REC-xml-names/]

- Presence of namespace indicated by suffix
  - plain input||0||10*1
  - UTF8(URI)||0^8||specifically-formatted input||1||10*1
Parallel hashing

- **Pros**
  - Can exploit parallelism in SIMD instructions
  - Can exploit parallelism in multi-core or distributed systems
  - Induce no throughput penalty when less parallelism available (for long messages)

- **Cons**
  - Needs more memory
  - Induce a performance penalty for short messages
A universal way to encode a tree

- Two related, yet distinct, aspects to specify:
  1. the exact (parameterized) tree layout and processing;
  2. the input formatting of leaves and nodes.

- Goals
  - Address the input formatting only
  - Be universal
    - agnostic of future tree structure specifications
  - Be sound [Keccak Team, ePrint 2009/210]

- Extra features
  - Flexible ways to spread message bits on nodes, e.g.,
    - interleaved 64-bit pieces for SIMD
    - 1MB chunks for independent processes
  - Possible re-use of hash function context (“connected hops”)
A universal way to encode a tree

- Two related, yet distinct, aspects to specify:
  1. the exact (parameterized) tree layout and processing;
  2. the input formatting of leaves and nodes.

- Goals
  - Address the input formatting only
  - Be universal
    -⇒ agnostic of future tree structure specifications
  - Be sound [KECCAK Team, ePrint 2009/210]

- Extra features
  - Flexible ways to spread message bits on nodes, e.g.,
    - interleaved 64-bit pieces for SIMD
    - 1MB chunks for independent processes
  - Possible re-use of hash function context (“connected hops”)
A universal way to encode a tree

- Two related, yet distinct, aspects to specify:
  1. the exact (parameterized) tree layout and processing;
  2. the input formatting of leaves and nodes.

- Goals
  - Address the input formatting only
  - Be universal
    - ⇒ agnostic of future tree structure specifications
  - Be sound [KECCAK Team, ePrint 2009/210]

- Extra features
  - Flexible ways to spread message bits on nodes, e.g.,
    - interleaved 64-bit pieces for SIMD
    - 1MB chunks for independent processes
  - Possible re-use of hash function context (“connected hops”)
Example 1/3

- $CV_i = h(M_i || \{\text{leaf}\} || \text{nonfinal})$
- $h(M_0 || \{\text{leaf}\} || CV_1 || CV_2 || CV_3 || \{#C = 4, CH, l = 64\} || \text{final})$
Some ideas for the SHA-3 standard

Parallel hashing

Example 2/3

- $CV_{i1} = h(M_{i1} || \{\text{leaf}\} || \text{nonfinal})$
- $CV_i = h(M_{i0} || \{\text{leaf}\} || CV_{i1} || \{\#C = 2, CH\} || \text{nonfinal})$
- $h(CV_0 || CV_1 || \{\#C = 2\} || \text{final})$
Example 3/3

$h(M || \text{leaf} || \text{final})$
Parallel hashing in SHA-3

\[ h(M \parallel \{\text{leaf}\} \parallel \text{final}) \]

Idea for discussion

1. Even if no parallel hashing mode is standardized at first
   - Foresee it in the input formatting
   - Make default sequential hashing a particular case of parallel hashing (i.e., a single root node)

[KECCAK Team, ePrint 2009/210]
Questions?

http://sponge.noekeon.org/
http://keccak.noekeon.org/