KECCAK and the SHA-3 Standardization

Guido BERTONI[1] Joan DAEMEN[1]
Michaël PEETERS[2] Gilles VAN ASSCHE[1]

[1]STMicroelectronics

[2]NXP Semiconductors

NIST, Gaithersburg, MD
February 6, 2013

Outline

Outline

Cryptographic hash functions

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

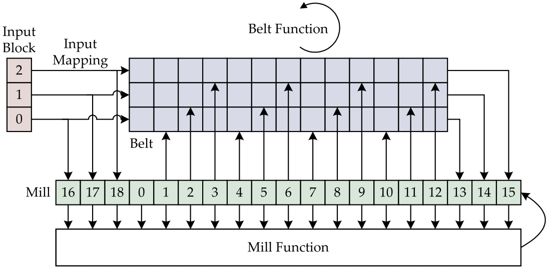
Input message →  → Digest

- 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

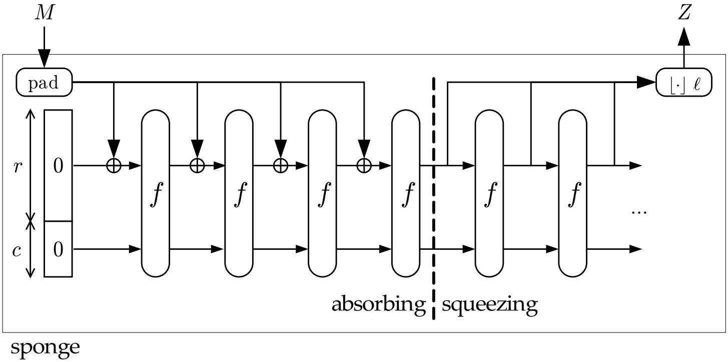
From RadioGatún to Keccak



- 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]

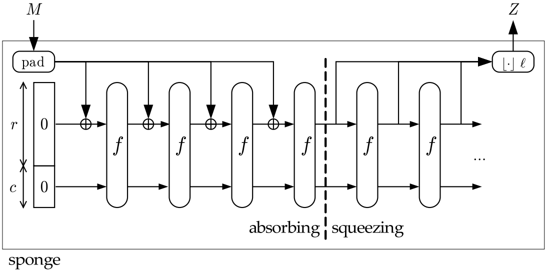
Outline

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

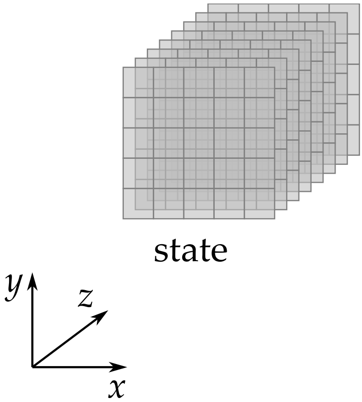
- Classical LC/DC criteria
  - Absence of large differential propagation probabilities
  - Absence of large input-output correlations
- Infeasibility of the CICO problem
  - Constrained Input Constrained Output
  - *Given partial input and partial output, find missing parts*
- Immunity to
  - Integral cryptanalysis
  - Algebraic attacks
  - Slide and symmetry-exploiting attacks
  - ...

Outline

KECCAK

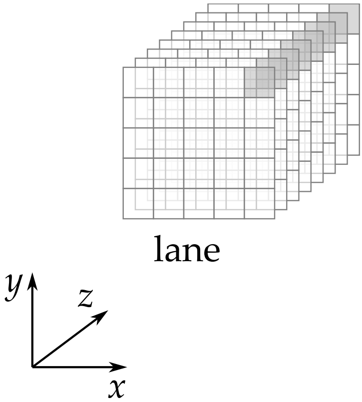
- Instantiation of a *sponge function*
- the permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
- Security-speed trade-offs using the same permutation, e.g.,
  - SHA-3 instance: $r = 1088$ and $c = 512$
    - permutation width: 1600
    - security strength 256: post-quantum 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



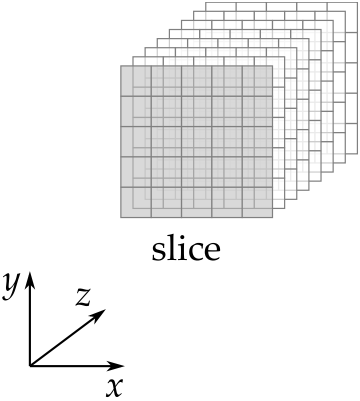
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

The state: an array of $5 \times 5 \times 2^{\ell}$ bits



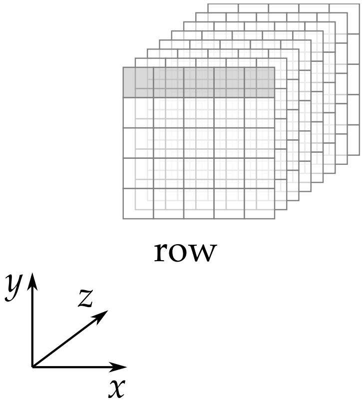
- 5×5 lanes, each containing 2^{ℓ} bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^{ℓ} of them

The state: an array of $5 \times 5 \times 2^\ell$ bits



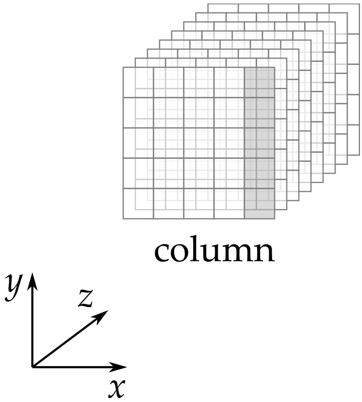
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

The state: an array of $5 \times 5 \times 2^\ell$ bits



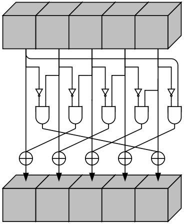
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

The state: an array of $5 \times 5 \times 2^\ell$ bits



- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

χ, the nonlinear mapping in KECCAK-f

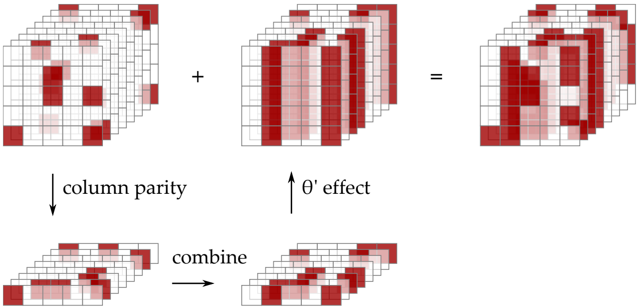


- "Flip bit if neighbors exhibit 01 pattern"
- Operates independently and in parallel on 5-bit rows
- Algebraic degree 2, inverse has degree 3
- LC/DC propagation properties easy to describe and analyze

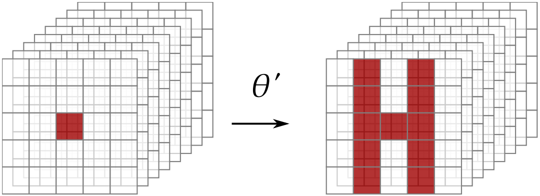
θ', a first attempt at mixing bits

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

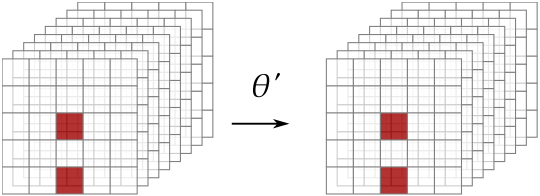
$$b_{x,y,z} = a_{x,y,z} \oplus c_{x-1,z} \oplus c_{x+1,z}$$



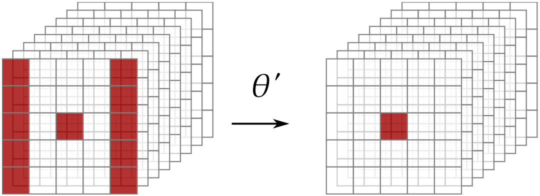
Diffusion of θ'



Diffusion of θ' (kernel)



Diffusion of the inverse of θ'

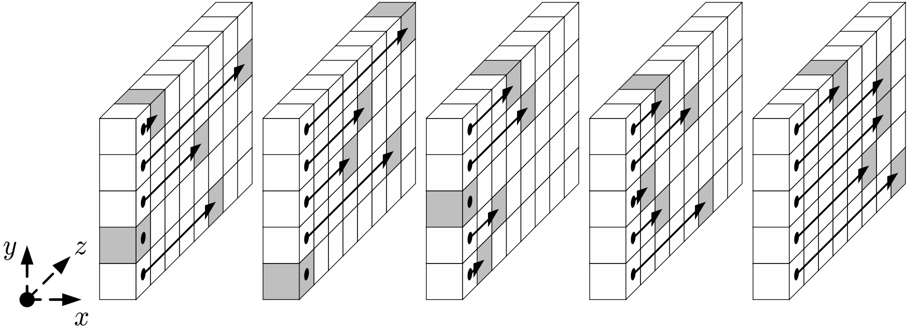


ρ for inter-slice dispersion

- We need diffusion between the slices ...
- ρ: cyclic shifts of lanes with offsets

$$i(i+1)/2 \bmod 2^{\ell}$$

- Offsets cycle through all values below 2^{ℓ}

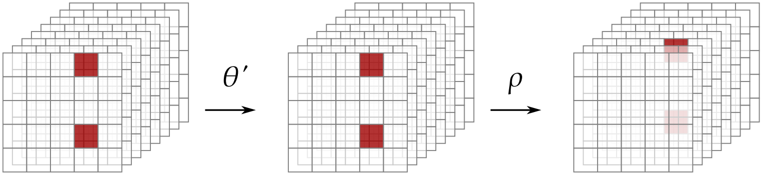


ι to break symmetry

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

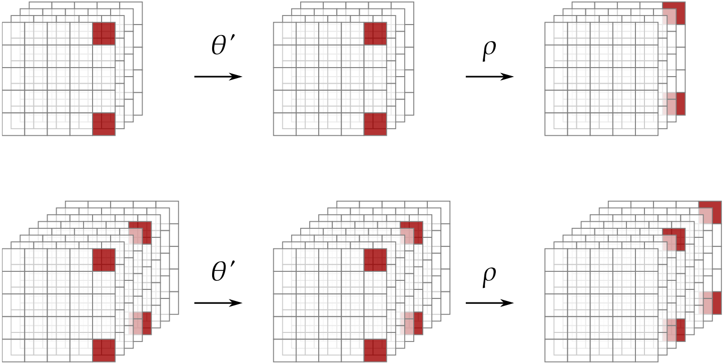
A first attempt at KECCAK-f

- Round function: $\mathrm{R} = \iota \circ \rho \circ \theta' \circ \chi$
- Problem: low-weight periodic trails by chaining:



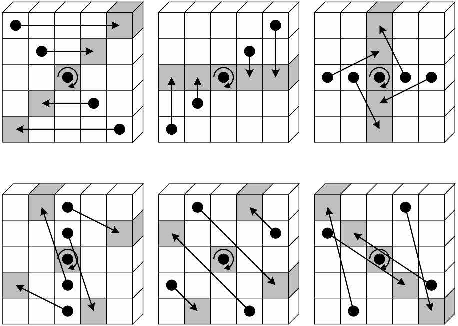
- χ: may propagate unchanged
- θ': propagates unchanged, because all column parities are 0
- ρ: in general moves active bits to different slices ...
- ...but not always

The Matryoshka property



- Patterns in Q' are z-periodic versions of patterns in Q

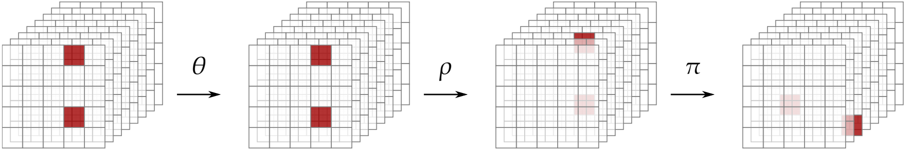
π for disturbing horizontal/vertical alignment



$$a_{x,y} \leftarrow a_{x',y'} \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

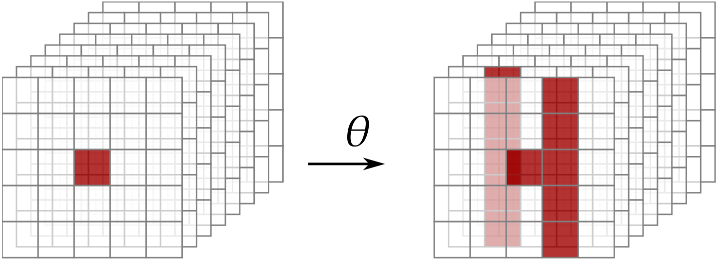
A second attempt at Keccak-f

- Round function: $\mathrm{R} = \iota \circ \pi \circ \rho \circ \theta' \circ \chi$
- Solves problem encountered before:

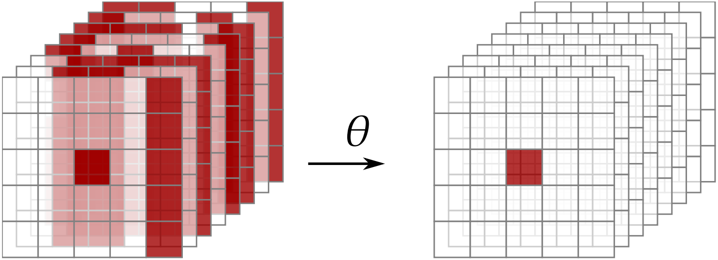


- π moves bits in same column to different columns!

Tweaking θ' to θ



Inverse of θ



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

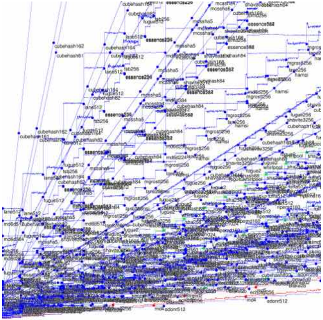
KECCAK-f summary

- Round function:

$$\mathsf{R} = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

- Number of rounds: $12 + 2\ell$
  - KECCAK-$f[25]$ has 12 rounds
  - KECCAK-$f[1600]$ has 24 rounds
- Efficiency
  - high level of parallellism
  - flexibility: bit-interleaving
  - software: competitive on wide range of CPU
  - dedicated hardware: very competitive
  - suited for protection against side-channel attack

Performance in software



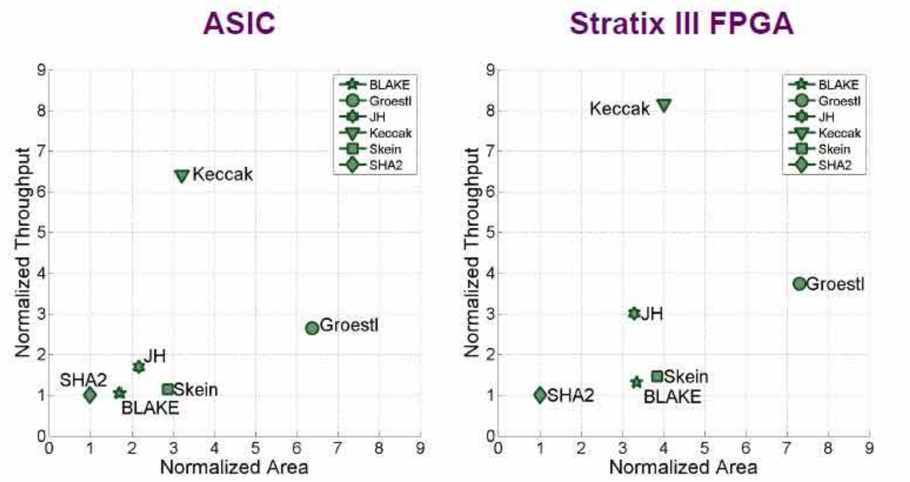
- Faster than SHA-2 on all modern PC
- KECCAKTREE faster than MD5 on some platforms

C/b	Algo	Strength
4.79	keccakc256treed2	128
4.98	md5	< 64
5.89	keccakc512treed2	256
6.09	sha1	< 80
8.25	keccakc256	128
10.02	keccakc512	256
13.73	sha512	256
21.66	sha256	128

[eBASH, hydra6, http://bench.cr.yp.to/]

Efficient and flexible in hardware

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



Outline

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 θ, π and ρ
- 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 ι
- See [KECCAK reference], [Ecrypt II Hash 2011], [FSE 2012]

Third-party cryptanalysis of KECCAK

Distinguishers on KECCAK-f[1600]

Rounds	Work	
3	low	CICO problem [Aumasson, Khovratovich, 2009]
4	low	cube testers [Aumasson, Khovratovich, 2009]
8	2^{491}	unaligned rebound [Duc, Guo, Peyrin, Wei, FSE 2012]
24	2^{1574}	zero-sum [Duan, Lai, ePrint 2011] [Boura, Canteaut, De Cannière, FSE 2011]

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

Rounds	
2	preimages and collisions [Morawiecki, CC]
2	collisions [Duc, Guo, Peyrin, Wei, FSE 2012 and CC]
3	40-bit preimage [Morawiecki, Srebrny, 2010]
3	near collisions [Naya-Plasencia, Röck, Meier, Indocrypt 2011]
4	key recovery [Lathrop, 2009]
4	distinguishers [Naya-Plasencia, Röck, Meier, Indocrypt 2011]
4	collisions [Dinur, Dunkelman, Shamir, FSE 2012 and CC]
5	near-collisions [Dinur, Dunkelman, Shamir, FSE 2012]

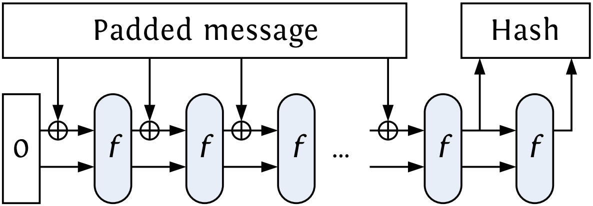
CC = Crunchy Crypto Collision and Preimage Contest

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 θ
  - θ^{-1} has very high average diffusion
  - Limits the construction of low-weight trails over more than a few rounds

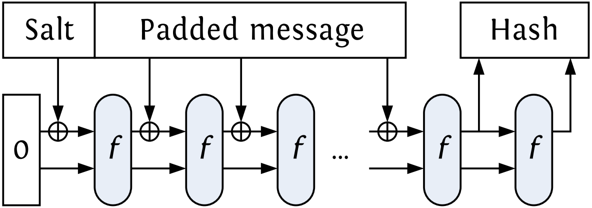
Outline

Regular hashing



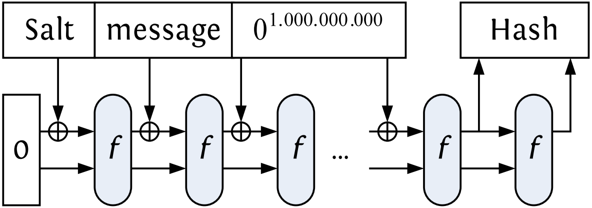
- Electronic signatures
- Data integrity (*shaXsum* ...)
- Data identifier (*Git, online anti-virus, peer-2-peer* ...)

Salted hashing



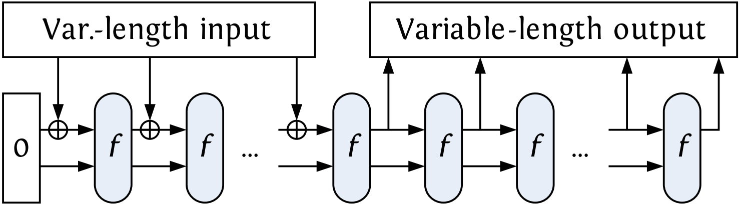
- Randomized hashing (RSASSA-PSS)
- Password storage and verification (*Kerberos*, /etc/shadow)

Salted hashing



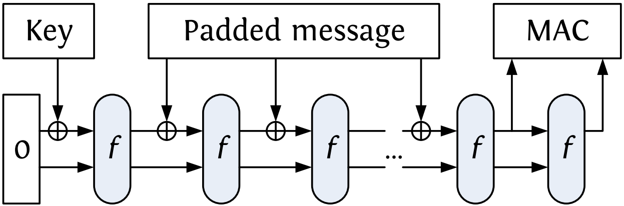
- Randomized hashing (RSASSA-PSS)
- Password storage and verification (*Kerberos*, /etc/shadow)
  - ...Can be as slow as you like it!

Mask generation function



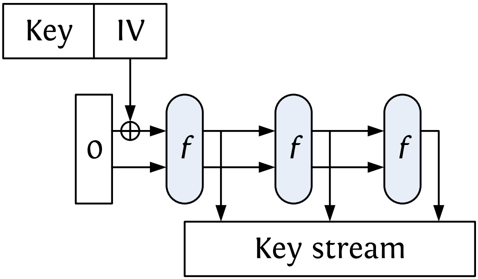
- Key derivation function in SSL, TLS
- Full-domain hashing in public key cryptography
  - electronic signatures RSASSA-PSS [PKCS#1]
  - encryption RSAES-OAEP [PKCS#1]
  - key encapsulation methods (KEM)

Message authentication codes



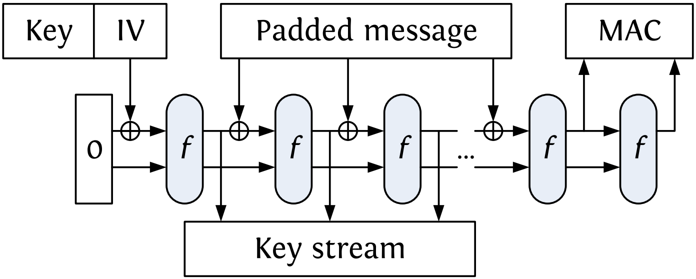
- As a message authentication code
- Simpler than HMAC [FIPS 198]
  - Required for SHA-1, SHA-2 due to length extension property
  - No longer needed for sponge

Stream encryption



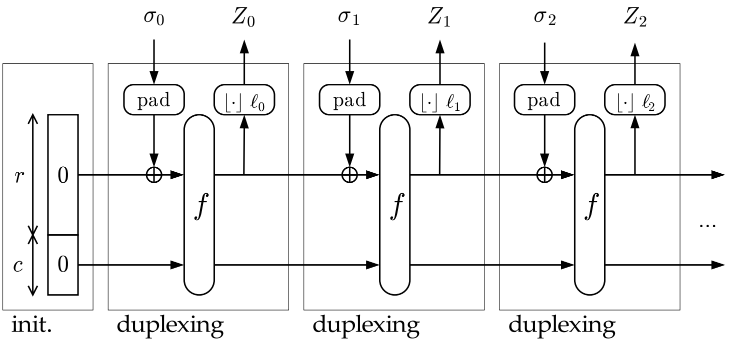
- As a stream cipher
    - Long output stream per IV: similar to OFB mode
    - Short output stream per IV: similar to counter mode

Single pass authenticated encryption



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

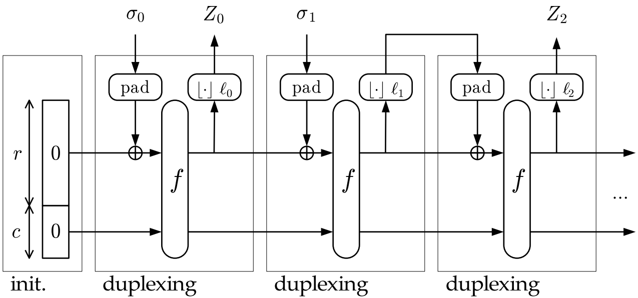
The duplex construction



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

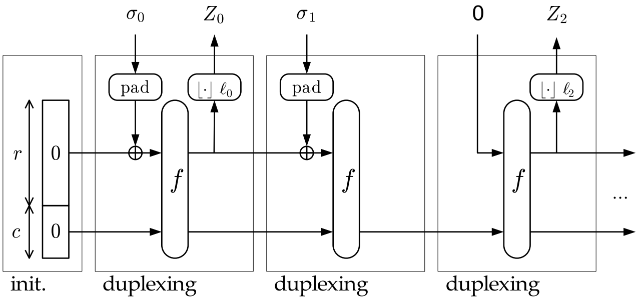
Reseedable pseudorandom sequence generator

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



Reseedable pseudorandom sequence generator

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



Outline

Output length oriented approach

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

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

Security strength	Collision resistance	Pre-image resistance	Required capacity	Relative perf.	SHA-3 instance
$s = 80$	$n \geq 160$	$n \geq 80$	$c = 160$	$\times 1.406$	**SHA3c160**
$s = 112$	$n \geq 224$	$n \geq 112$	$c = 224$	$\times 1.343$	**SHA3c224**
$s = 128$	$n \geq 256$	$n \geq 128$	$c = 256$	$\times 1.312$	**SHA3c256**
$s = 192$	$n \geq 384$	$n \geq 192$	$c = 384$	$\times 1.188$	**SHA3c384**
$s = 256$	$n \geq 512$	$n \geq 256$	$c = 512$	$\times 1.063$	**SHA3c512**
s	$n \geq 2s$	$n \geq s$	$c = 2s$	$\times \frac{1600-c}{1024}$	**SHA3[c=2s]**

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 **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

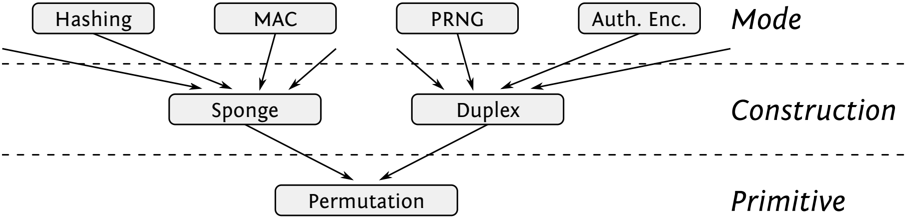
1. Let SHA-3 be a sponge
   - Allow freedom in choosing c
   - Allow variable output length
2. Decouple security and output length
   - Set minimum capacity $c \geq 2s$ for [SP 800-57]'s level s
3. Base naming scheme on security level
   - For instance **SHA3c180** for Keccak$[c = 180]$
4. For SHA-2-n 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 **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: $c = 576$

Structuring the standard

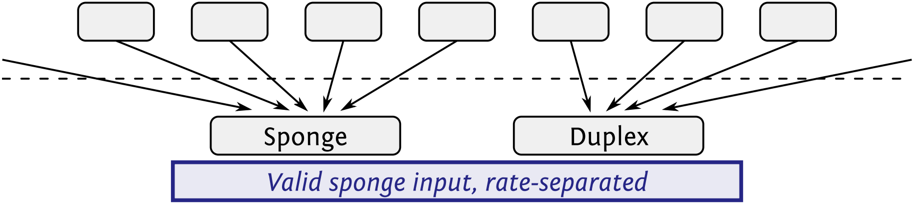


Ideas for discussion

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

Multiple instances of Keccak

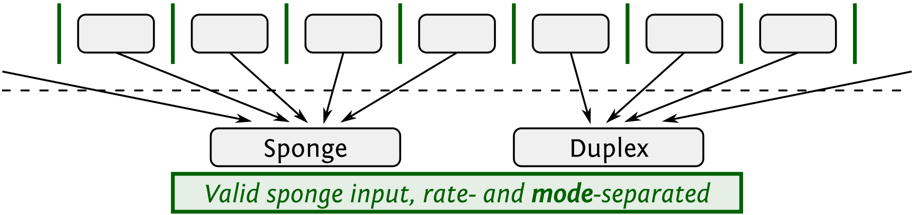


Multi-rate padding

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

  [Keccak Team, SAC 2011]

Domain separation



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
  - $\text{plain input}||0||10^*1$
  - $\text{UTF8}(\text{URI})||0^8||\text{specifically-formatted input}||1||10^*1$

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”)

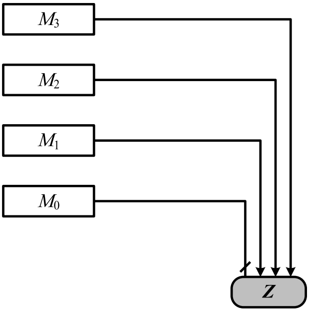
A universal way to encode a tree

- 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
    $\Rightarrow$ 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

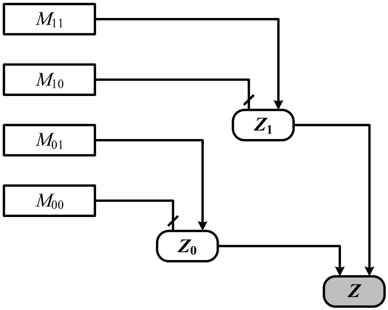
- Two related, yet distinct, aspects to specify:
  1. the exact (parameterized) tree layout and processing;
  2. the input formatting of leaves and nodes.
- Goals
  - Address the input formatting only
  - Be universal
    $\Rightarrow$ agnostic of future tree structure specifications
  - Be **sound** [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



- $\mathrm{CV}_i = h(M_i || \{\text{leaf}\} || \text{nonfinal})$
- $h(M_0 || \{\text{leaf}\} || \mathrm{CV}_1 || \mathrm{CV}_2 || \mathrm{CV}_3 || \{\#C = 4, \mathrm{CH}, I = 64\} || \textbf{final})$

Example 2/3



- $\mathrm{CV}_{i1} = h(M_{i1}||\{\mathrm{leaf}\}||\mathrm{nonfinal})$
- $\mathrm{CV}_{i} = h(M_{i0}||\{\mathrm{leaf}\}||\mathrm{CV}_{i1}||\{\#C = 2, \mathrm{CH}\}||\mathrm{nonfinal})$
- $h(\mathrm{CV}_0||\mathrm{CV}_1||\{\#C = 2\}||\mathbf{final})$

Example 3/3

$$\boxed{M}$$

- $h(M||\{\text{leaf}\}||\textbf{final})$

Parallel hashing in SHA-3

$$\boxed{M}$$

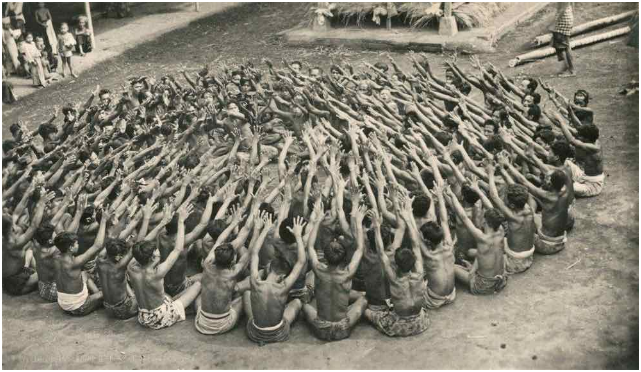
$$h(M||\{\text{leaf}\}||\textbf{final})$$

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/