Introduction to SHA-3 and KECCAK

Joan DAEMEN

STMicroelectronics and Radboud University

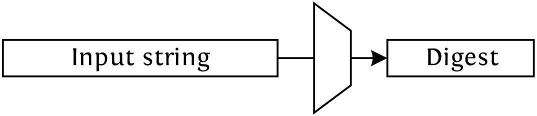
Crypto summer school 2015
Šibenik, Croatia, May 31 - June 5, 2015

Outline

Outline

Cryptographic hash functions

- Function h from $\mathbf{Z}_2^*$ to $\mathbf{Z}_2^n$
- Typical values for n: 128, 160, 256, 512



- Pre-image resistant: it shall take 2^n effort to
  - given y, find x such that $h(x) = y$
- 2nd pre-image resistance: it shall take 2^n effort to
  - given M and $h(M)$, find another M' with $h(M') = h(M)$
- Collision resistance: it shall take $2^{n/2}$ effort to
  - find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$
- More general: *should behave like a random oracle*

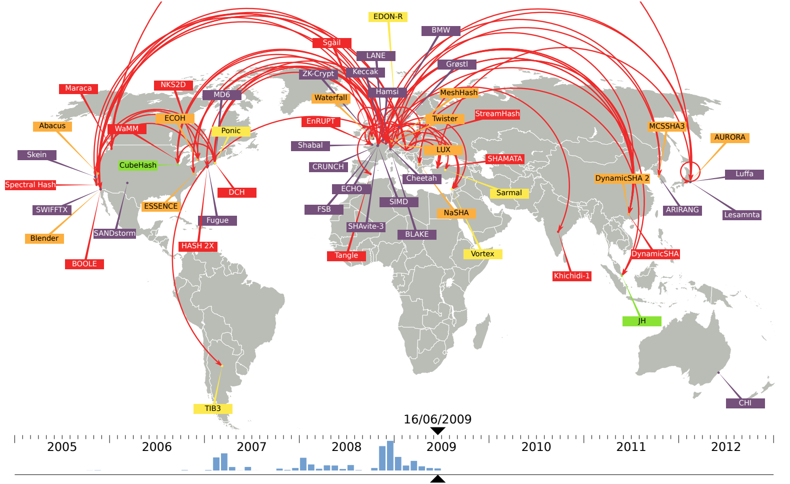
The origins of the SHA-3 competition

- 2005-2006: NIST thinks about having a SHA-3 contest
  - MD5 and standard SHA-1 were damaged by attacks
  - SHA-2 based on the same principles than MD5 and SHA-1
  - open call for SHA-2 successor
  - ...and for analysis, comparisons, etc.
- October 2008: Deadline for proposals
  - *more efficient than SHA-2*
  - output lengths: 224, 256, 384, 512 bits
  - security: collision and (2nd) pre-image resistant
  - specs, reference and optimized code, test vectors
  - design rationale and preliminary analysis
  - patent waiver

The SHA-3 competition

- First round: October 2008 to summer 2009
  - 64 submissions, 51 accepted
  - 37 presented at 1st SHA-3 candidate conf. in Leuven, February 2009
  - many broken by cryptanalysis
  - NIST narrowed down to 14 semi-finalists
- Second round: summer 2009 to autumn 2010
  - analysis presented at 2nd SHA-3 conf. in Santa Barbara, August 2010
  - NIST narrowed down to 5 finalists
- Third round: autumn 2010 to October 2012
  - analysis presented at 3rd SHA-3 conf. in Washington, March 2012
- October 2, 2012: NIST announces KECCAK as SHA-3 winner

NIST SHA-3: the battlefield



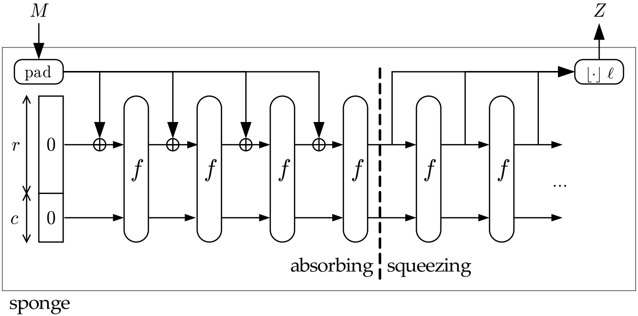
[courtesy of Christophe De Cannière]

Outline

Sponge origin: RadioGatún

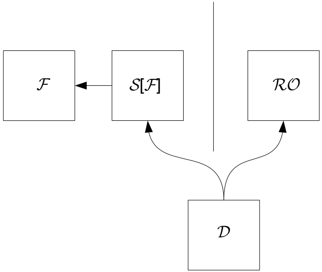
- Initiative to design hash/stream function (late 2005)
  - rumours about NIST call for hash functions
  - starting point: fixing Panama [Daemen, Clapp, FSE 1998]
  - with long-time colleagues Gilles Van Assche and Michaël Peeters
  - and ST Italy colleague Guido Bertoni joining in
- RadioGatún [Keccak team, NIST 2nd hash workshop 2006]
  - more conservative than Panama
  - arbitrary output length
  - **expressing security claim** for arbitrary output length function
- Sponge functions [Keccak team, Ecrypt hash, 2007]
  - random sponge instead of random oracle as security goal
  - sponge construction calling random permutation
  - ... *closest thing to a random oracle with a finite state* ...

The sponge construction



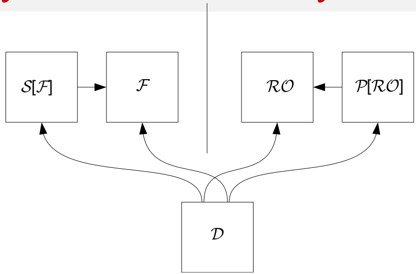
- Generalizes hash function: *extendable output function* (XOF)
- Calls a b-bit permutation f, with $b = r + c$
  - r bits of *rate*
  - c bits of *capacity* (security parameter)

Generic security: indistinguishability



- Success probability of distinguishing between:
  - ideal function: a monolithic random oracle $\mathcal{RO}$
  - construction $\mathcal{S}[\mathcal{F}]$ calling an random permutation $\mathcal{F}$
- Adversary $\mathcal{D}$ sends queries (M, ℓ) according to algorithm
- Express $\Pr(\text{success}|\mathcal{D})$ as a function of total cost of queries N
- Problem: in real world, $\mathcal{F}$ is available to adversary

Generic security: indifferentiability [Maurer et al. (2004)]



- Applied to hash functions in [Coron et al. (2005)]
  - distinguishing mode-of-use from ideal function ($\mathcal{RO}$)
  - covers adversary with access to permutation $\mathcal{F}$ at left
  - additional interface, covered by a *simulator* at right
- Methodology:
  - build $\mathcal{P}$ that makes left/right distinguishing difficult
  - prove bound for advantage given this simulator $\mathcal{P}$
  - $\mathcal{P}$ may query $\mathcal{RO}$ for acting $\mathcal{S}$-consistently: $\mathcal{P}[\mathcal{RO}]$

Generic security of the sponge construction

Concept of *advantage*:

$$\Pr(\text{success}|\mathcal{D}) = \frac{1}{2} + \frac{1}{2}\text{Adv}(\mathcal{D})$$

Theorem (Bound on the $\mathcal{RO}$-differentiating advantage of sponge)

$$A \leq \frac{N^2}{2^{c+1}}$$

A: differentiating advantage of random sponge from random oracle
N: total data complexity
c: capacity

[Keccak team, Eurocrypt 2008]

Implications of the bound

- Let $\mathcal{D}$: n-bit output pre-image attack. Success probability:
  - for random oracle: $P_{\text{pre}}(\mathcal{D}|\mathcal{RO}) = q2^{-n}$
  - for random sponge: $P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) = ?$
- A distinguisher $\mathcal{D}$ with $A = P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) - P_{\text{pre}}(\mathcal{D}|\mathcal{RO})$
  - do pre-image attack
  - if success, conclude random sponge and $\mathcal{RO}$ otherwise
- But we have a proven bound $A \leq \frac{N^2}{2^{c+1}}$, so

$$P_{\text{pre}}(\mathcal{D}|\mathcal{S}[\mathcal{F}]) \leq P_{\text{pre}}(\mathcal{D}|\mathcal{RO}) + \frac{N^2}{2^{c+1}}$$

- Can be generalized to any attack
- Note that A is independent of output length n

Implications of the bound (cont'd)

- Informally, random sponge is like random oracle for $N < 2^{c/2}$
- Security strength for output length n:
  - collision-resistance: $\min(c/2, n/2)$
  - first pre-image resistance: $\min(c/2, n)$
  - second pre-image resistance: $\min(c/2, n)$
- Proof assumes f is a random permutation
  - provably secure against generic attacks
  - ...but not against attacks that exploit specific properties of f
- No security against multi-stage adversaries

Design approach

Hermetic sponge strategy

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

Our mission

Design permutation f without exploitable properties

How to build a strong 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

Outline

KECCAK[r, c]

- Sponge function using the permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
    … from toy over lightweight to high-speed …
- 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 (initially expected from) SHA-1

See [The KECCAK reference] for more details

KECCAK[r, c]

- Sponge function using the permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
    … from toy over lightweight to high-speed …
- 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 (initially expected from) SHA-1

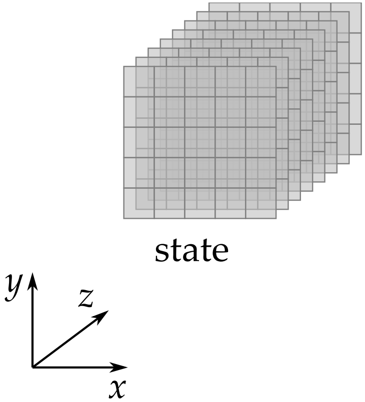
See [The KECCAK reference] for more details

KECCAK[r, c]

- Sponge function using the permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
    … from toy over lightweight to high-speed …
- 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 (initially expected from) SHA-1

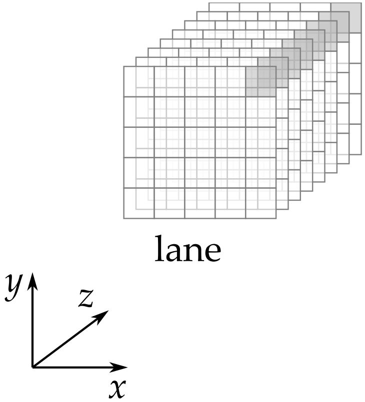
See [The KECCAK reference] for more details

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



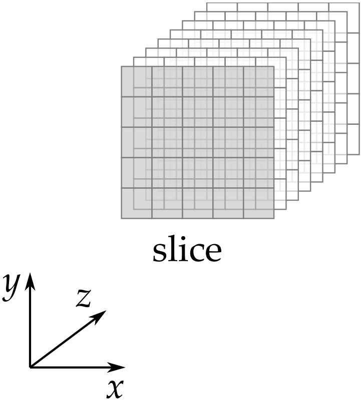
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

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



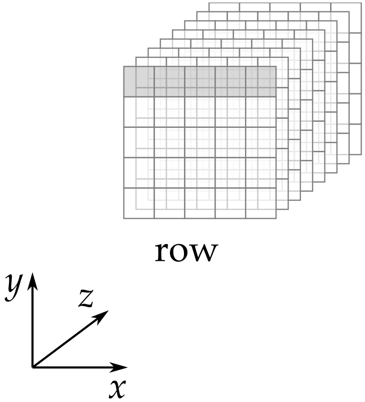
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

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



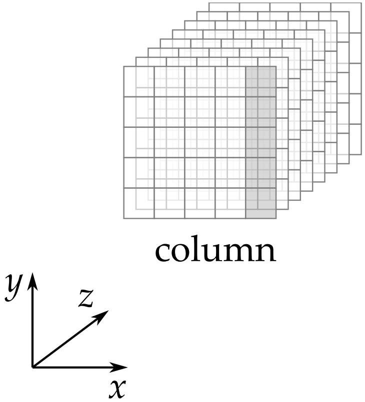
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

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



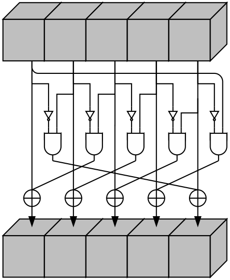
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

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



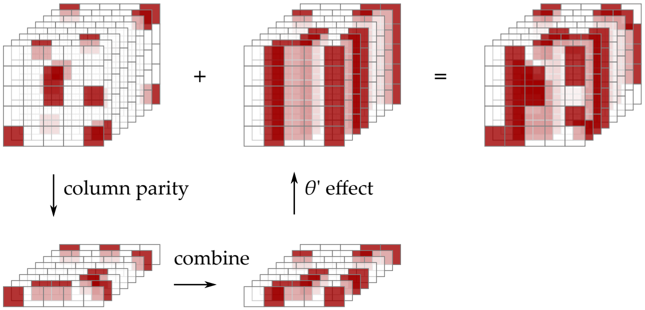
- "Flip bit if neighbors exhibit 01 pattern"
- Operates independently and in parallel on 5-bit rows
- Cheap: small number of operations per bit

θ', a first attempt at mixing bits

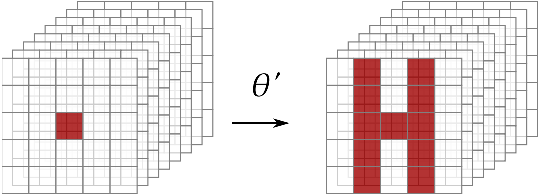
- Compute parity $c_{x,z}$ of each column
- Add to each cell parity of neighboring columns:

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

- Cheap: two XORs per bit

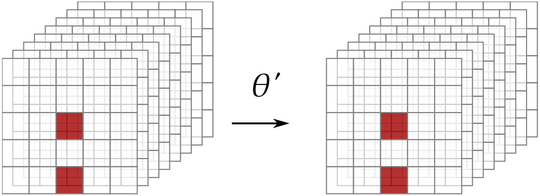


Diffusion of θ'



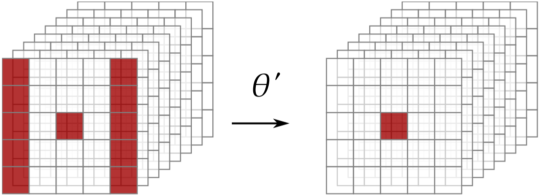
$$1 + \left(1 + y + y^2 + y^3 + y^4\right)\left(x + x^4\right)$$
$$\left(\bmod \left\langle 1 + x^5, 1 + y^5, 1 + z^w \right\rangle\right)$$

Diffusion of θ' (kernel)



$$1 + \left(1 + y + y^2 + y^3 + y^4\right)\left(x + x^4\right)$$
$$\left(\bmod \left\langle 1 + x^5, 1 + y^5, 1 + z^w \right\rangle\right)$$

Diffusion of the inverse of θ'



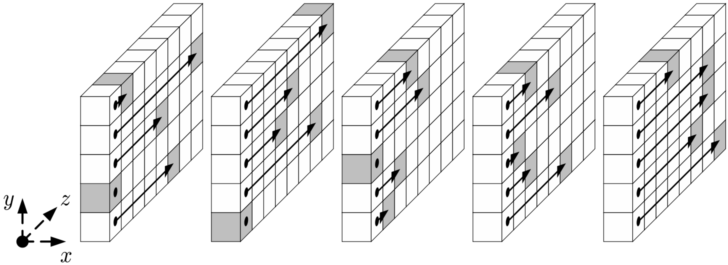
$$1 + \left(1 + y + y^2 + y^3 + y^4\right)\left(x^2 + x^3\right)$$
$$\left(\bmod \left\langle 1 + x^5, 1 + y^5, 1 + z^w \right\rangle\right)$$

ρ for inter-slice dispersion

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

$$i(i+1)/2 \bmod 2^{\ell}, \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^{i-1} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

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

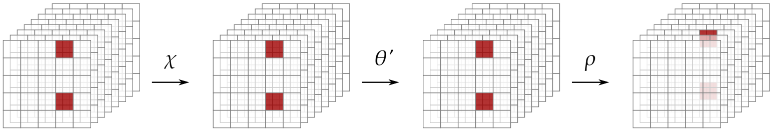


ι to break symmetry

- XOR of round-dependent constant to lane in origin
- Without ι, the round mapping would be symmetric
  - invariant to translation in the z-direction
  - susceptible to *rotational* cryptanalysis
- 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)

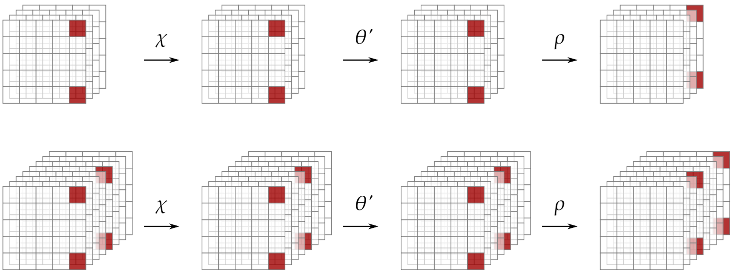
A first attempt at KECCAK-f

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



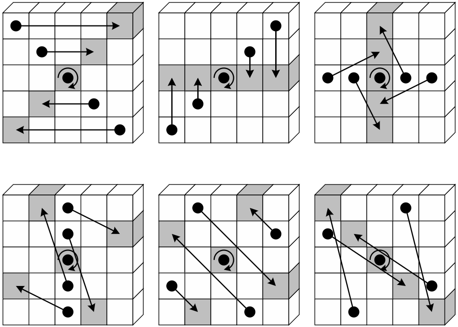
  - χ: propagates unchanged with weight 4
  - θ': propagates unchanged, because all column parities are 0
  - ρ: in general moves active bits to different slices ...
    ...but not always

The Matryoshka property



- Patterns in Q' are z-periodic versions of patterns in Q
- Weight of trail Q' is twice that of trail Q (or 2^n times in general)

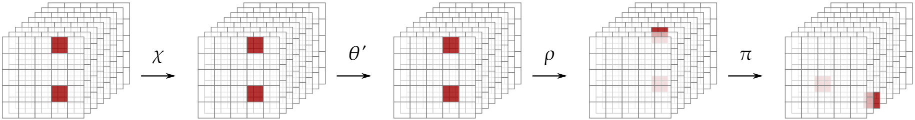
π for disturbing horizontal/vertical alignment



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

A second attempt at KECCAK-f

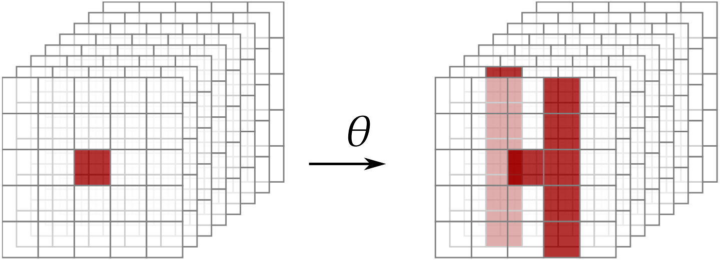
- Round function: $R = \iota \circ \pi \circ \rho \circ \theta' \circ \chi$
- Solves problem encountered before:



- π moves bits in same column to different columns!

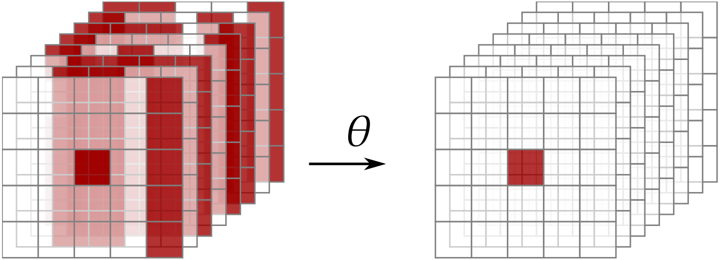
Almost there, still a final tweak ...

Tweaking θ' to θ



$$1 + \left(1 + y + y^2 + y^3 + y^4\right)\left(x + x^4 z\right)$$
$$\left(\bmod \left\langle 1 + x^5, 1 + y^5, 1 + z^w \right\rangle\right)$$

Inverse of θ



$$1 + \left(1 + y + y^2 + y^3 + y^4\right) \mathbf{Q},$$
$$\text{with } \mathbf{Q} = 1 + \left(1 + x + x^4 z\right)^{-1} \bmod \left\langle 1 + x^5, 1 + z^w \right\rangle$$

- **Q** is dense, so:
  - Diffusion from single-bit output to input very high
  - Increases resistance against LC/DC and algebraic attacks

KECCAK-f summary

- Round function:

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

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

Outline

The long road to the SHA-3 FIPS

- February 2013: NIST-Keccak-team meeting
  - SHA-2 replacement by now less urgent
  - ...but Keccak is more than just hashing!
- NIST disseminates joint SHA-3 proposal
- Summer 2013: Snowden revelations
  - alleged NSA backdoor in DUAL EC DRBG
  - SHA-3 proposal framed as “NIST weakening Keccak”
- Early 2014: standard takes shape addressing public concerns
- Friday, April 4, 2014: draft FIPS 202 for public comments
- August 2014: NIST announces plans at SHA-3 conference
- Mid 2015 (expected): FIPS 202 official publication

FIPS 202: what is inside?

- Content
  - KECCAK instances for
    - 4 hash functions
    - 2 XOFs
  - KECCAK-*f* all 7 block widths
    - even reduced-round versions
    - unlike AES FIPS that has only 1 of the 5 Rijndael widths
  - sponge construction
- Concept: toolbox for building other functions
  - tree hashing, MAC, encryption, ...
  - dedicated *special publications* (NIST SP 800-XX) under development

http://csrc.nist.gov/groups/ST/hash/sha-3/Aug2014/index.html

XOF: eXtendable Output Function

"XOF: a function in which the output can be extended to any length."

- Good for full domain hash, stream ciphers and key derivation
  [Ray Perlner, SHA 3 workshop 2014]
- Quite natural for sponge
  - keeps state and delivers more output upon request
  - bits of output do not depend on the number of bits requested
- Allows simplification:
  - instead of separate hash functions per output length
  - a single XOF can cover all use cases:

$$\text{H-256}(M) = \lfloor \text{XOF}(M) \rfloor_{256}$$

Domain separation

- Some protocols and applications need
  - multiple hash functions or XOFs
  - that should be *independent*
- With a single XOF?
- Yes: using domain separation
  - output of $\text{XOF}(M||0)$ and $\text{XOF}(M||1)$ are independent
  - ...unless XOF has a cryptographic weakness
- Generalization to 2^n functions with D an n-bit *diversifier*

$$\text{XOF}_D(M) = \text{XOF}(M||D)$$

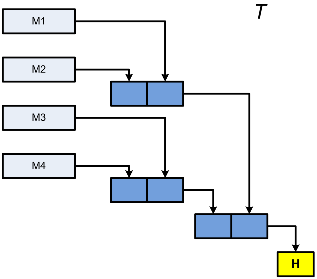
- Variable-length diversifiers: suffix-free set of strings

The XOFs and hash functions in FIPS 202

- Four drop-in replacements identical to those in KECCAK submission
- Two *extendable output functions* (XOF)
- Tree-hashing ready: SAKURA coding [Keccak team, ePrint 2013/231]

XOF	SHA-2 drop-in replacements
$\text{KECCAK}[c = 256](M\|\|11\|\|11)$	
	$\lfloor \text{KECCAK}[c = 448](M\|\|01) \rfloor_{224}$
$\text{KECCAK}[c = 512](M\|\|11\|\|11)$	$\lfloor \text{KECCAK}[c = 512](M\|\|01) \rfloor_{256}$
	$\lfloor \text{KECCAK}[c = 768](M\|\|01) \rfloor_{384}$
	$\lfloor \text{KECCAK}[c = 1024](M\|\|01) \rfloor_{512}$
SHAKE128 and **SHAKE256**	**SHA3-224** to **SHA3-512**

Tree hashing



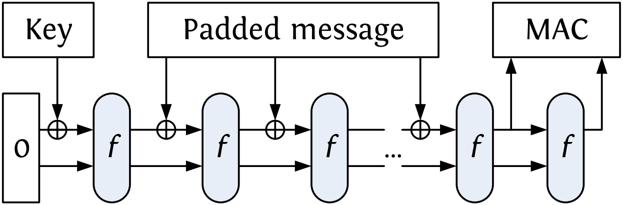
Features:

- hash recomputation when modifying small part of file
- parallelizable
- performance:

function	instruction	cycles/byte
KECCAK$[c = 256] \times 1$	x86_64	7.70
KECCAK$[c = 256] \times 2$	AVX2 (128-bit only)	5.30
KECCAK$[c = 256] \times 4$	AVX2	2.87

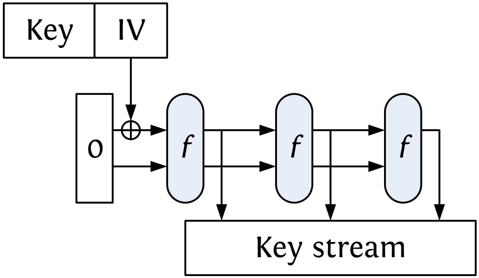
CPU: Haswell with AVX2 256-bit SIMD

MAC (and key derivation)



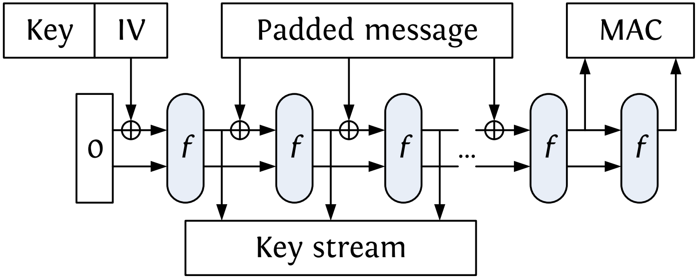
- $\mathsf{KMAC}[K](M) = \mathsf{H}(K||M)$
- $\mathsf{XMAC}[K](M, \lambda) = \mathsf{XOF}(K||M||\lambda)$
    - λ length of the output
- XKDF: key derivation function based on XOF (XMAC)
- HMAC [FIPS 198] no longer needed!

Stream encryption



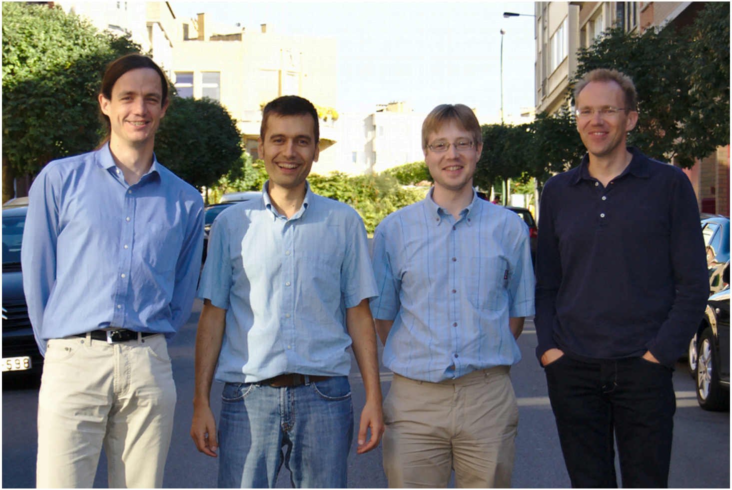
- Encryption: add key stream to plaintext bit per bit

Efficient authenticated encryption



- Basis: using Keccak-*f* in **Duplex** construction
  - generic security equivalent to sponge [Keccak team, SAC 2011]
  - also for random generation with reseeding (/dev/urandom ...)
- Domain separation layer on top: duplexWrap

Questions?



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