Cryptographic hash functions, sponge functions and KECCAK

Guido BERTONI[1]

Joint work with
J. DAEMEN[1], M. PEETERS[2], and G. VAN ASSCHE[1],

[1]STMicroelectronics [2]NXP Semiconductors

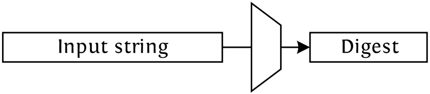
May 7, 2013

Outline

Outline

Cryptographic hash functions

- Function h
  - from any binary string $\{0,1\}^*$
  - to a fixed-size digest $\{0,1\}^n$
  - **One-way**: given $h(x)$ hard to find x...



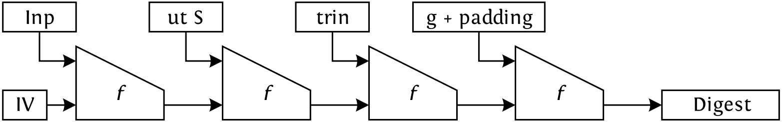
- Applications in cryptography
  - *Signatures*: $\text{sign}_{\text{RSA}}(h(M))$ instead of $\text{sign}_{\text{RSA}}(M)$
  - *Key derivation*: master key K to derived keys $(K_i = h(K\|i))$
  - *Bit commitment, predictions*: $h(\text{what I know})$
  - *Message authentication*: $h(K\|M)$
  - ...

Examples of popular hash functions

- MD5: $n = 128$
  - Published by Ron Rivest in 1992
  - Successor of MD4 (1990)
- SHA-1: $n = 160$
  - Designed by NSA, standardized by NIST in 1995
  - Successor of SHA-0 (1993)
- SHA-2: family supporting multiple lengths
  - Designed by NSA, standardized by NIST in 2001
  - 4 members named SHA-n
  - SHA-224, SHA-256, SHA-384 and SHA-512

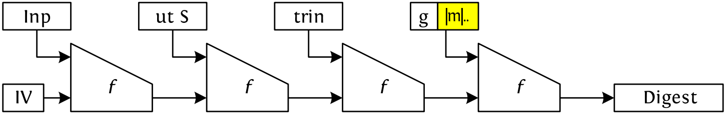
The chaining structure: Merkle-Damgård

- Simple iterative construction:
  - iterative application of compression function (CF)
- Proven collision-resistance preserving



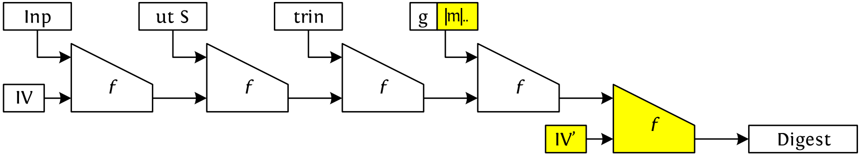
Merkle-Damgård strengthening

- Input length added to the input string



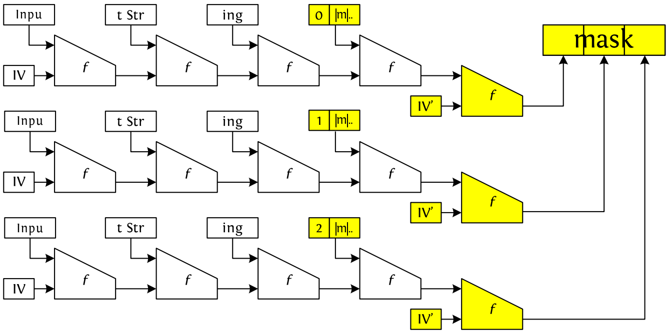
Enveloped Merkle-Damgård

- Special processing for last call

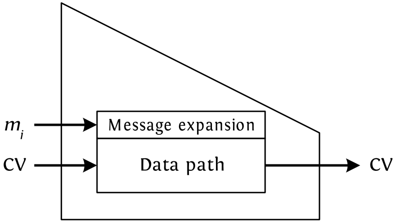


Variable-output-length Merkle-Damgård

- Mask generating function (MGF)



The compression function: Davies-Meyer (nearly)

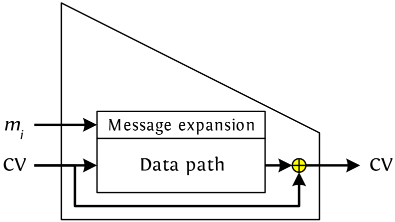


Uses a block cipher:

- Separated data path and message expansion

But not one-way!

The compression function: Davies-Meyer



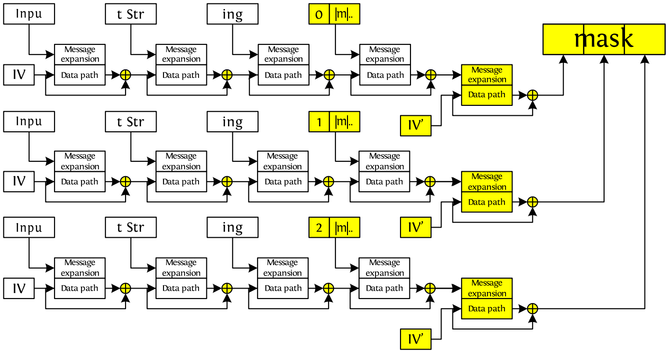
Uses a block cipher:

- Separated data path and message expansion

Some feedforward due to Merkle-Damgård

Combining them all

- This is not so simple anymore...



The use of basic operations

- All popular hash functions were based on ARX
  - addition modulo 2^n with $n = 32$ (and $n = 64$)
  - bitwise addition: XOR
  - bitwise shift operations, cyclic shift
  - security: "algebraically incompatible operations"
- ARX would be elegant
  - ...but silently assumes a specific integer coding
- ARX would be efficient
  - ...but only in software on CPUs with n-bit words
- ARX would have good cryptographic properties
  - but is very hard to analyze
  - ...attacks have appeared after years

Trouble in paradise

- 1991-1993: Den Boer and Bosselaers attack MD4 and MD5
- 1996: Dobbertin improves attacks on MD4 and MD5
- 1998: Chabaud and Joux attack SHA-0
- 2004: Joux et al. break SHA-0
- 2004: Wang et al. break MD5
- 2004: Joux show multicollisions on Merkle-Damgård
- 2005: Lenstra et al., and Klima, make MD5 attack practical
- 2005: Wang et al. theoretically break SHA-1
- 2005: Kelsey and Schneier: 2nd pre-image attacks on MD
- 2006: De Cannière and Rechberger further break SHA-1
- 2006: Kohno and Kelsey: herding attacks on MD

Outline

A way out of the hash function crisis

- 2005-2006: trust in established hash functions was crumbling, due to
  - use of ARX
  - adoption of Merkle-Damgård
  - and SHA-2 were based on the same principles
- 2007: NIST calls for SHA-3
  - similar to AES contest
  - a case for the international cryptographic community!

SHA-3 contest

- Open competition organized by NIST
  - NIST provides forum
  - scientific community contributes: designs, attacks, implementations, comparisons
  - NIST draws conclusions and decides
- Goal: replacement for the SHA-2 family
  - 224, 256, 384 and 512-bit output sizes
  - other output sizes are optional
- Requirements
  - security levels specified for traditional attacks
  - each submission must have
    - complete documentation, including design rationale
    - reference and optimized implementations in C

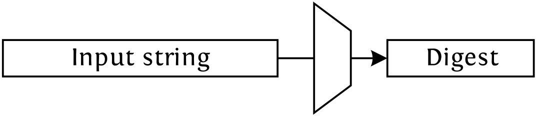
SHA-3 time schedule

- January 2007: initial call
- October 2008: submission deadline
- February 2009: first SHA-3 conference in Leuven
  - Presentation of 1st round candidates
- July 2009: NIST announces 2nd round candidates
- August 2010: second SHA-3 conference in Santa Barbara
  - cryptanalytic results
  - hardware and software implementation surveys
  - new applications
- December 2010: announcement of finalists
- 2012: final SHA-3 conference and selection of winner

Outline

Traditional security requirements of hash functions

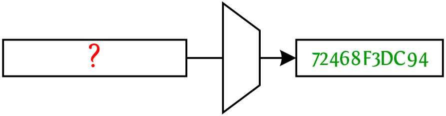
- Function h from $\mathbf{Z}_2^*$ to $\mathbf{Z}_2^n$



- Security requirements
  - pre-image resistance
  - 2nd pre-image resistance
  - collision resistance

Pre-image resistance

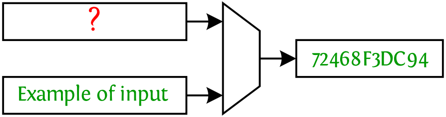
- Given $y \in \mathbf{Z}_2^n$, find $x \in \mathbf{Z}_2^*$ such that $h(x) = y$
- **Example**: given derived key $K_1 = h(K\|1)$, find master key K



- There exists a generic attack requiring about 2^n calls to h
- Requirement: there is no attack more efficient

2nd pre-image resistance

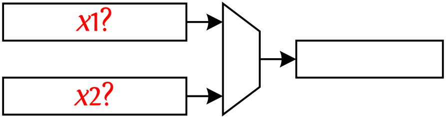
- Given $x \in \mathbf{Z}_2^*$, find $x' \neq x$ such that $h(x') = h(x)$
- **Example**: signature forging
  - given M and $\mathrm{sign}(h(M))$, find another M' with equal signature



- There exists a generic attack requiring about 2^n calls to h

Collision resistance

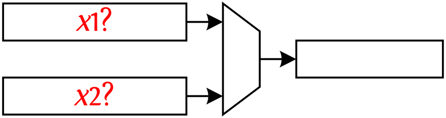
- Find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$



- There exists a generic attack requiring about $2^{n/2}$ calls to h

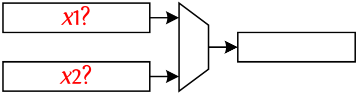
Collision resistance

- Find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$



- There exists a generic attack requiring about $2^{n/2}$ calls to h
  - Birthday paradox: among 23 people, two have the same birthday (with 50% probability)

Collision resistance (continued)



- **Example**: "secretary" signature forging
  - Set of good messages $\{M_i^{\text{good}}\}$
  - Set of bad messages $\{M_i^{\text{bad}}\}$
  - Find $h(M_i^{\text{good}}) = h(M_j^{\text{bad}})$
  - Boss signs M_i^{good}, but valid also for M_j^{bad}

Other requirements

- What if we use a hash function in other applications?
- To build a MAC function, e.g., HMAC (FIPS 198)
- To destroy algebraic structure, e.g.,
  - encryption with RSA: OAEP (PKCS #1)
  - signing with RSA: PSS (PKCS #1)
- Problem:
  - additional requirements on top of traditional ones
  - how to know what a hash function is designed for?

Contract

- Security of a concrete hash function h cannot be proven
  - sometimes reductions are possible...
  - rely on public scrutiny!
- Security claim: contract between designer and user
  - security claims $\geq$ security requirements
  - attack that invalidates claim, breaks h!
- Claims often implicit
  - e.g., the traditional security requirements are implied

List of claimed properties

- Security claims by listing desired properties
  - collision resistant
  - (2nd) pre-image resistant
  - correlation-free
  - resistant against length-extension attacks
  - chosen-target forced-prefix pre-image resistance
  - ...
- But ever-growing list of desired properties
- Moving target as new applications appear over time

But hey, the ideal hash function exists!

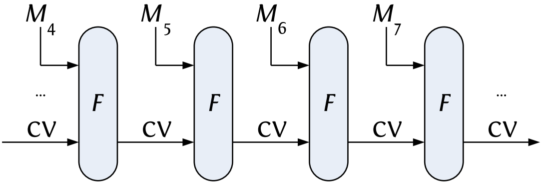
Random oracle $\mathcal{RO}$

- A random oracle [Bellare-Rogaway 1993] maps:
  - message of variable length
  - to an infinite output string
- Supports queries of following type: (M, ℓ)
  - M: message
  - ℓ: requested number of output bits
- Response Z
  - String of ℓ bits
  - Independently and identically distributed bits
  - Self-consistent: equal M give matching outputs

Compact security claim

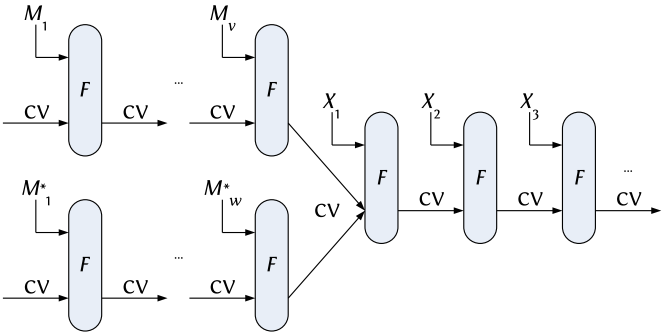
- Truncated to n bits, $\mathcal{RO}$ has all desired properties, e.g.,
  - Generating a collision: $2^{n/2}$
  - Finding a (2nd) pre-image: 2^n
  - And [my chosen requirement]: $f(n)$
- Proposal for a compact security claim:
  - "My function h behaves as a **random oracle**"
- Does not work, unfortunately

Iterated hash functions



- All practical hash functions are iterated
  - Message M cut into blocks $M_1, \ldots, M_l$
  - q-bit chaining value
- Output is function of final chaining value

Internal collisions!



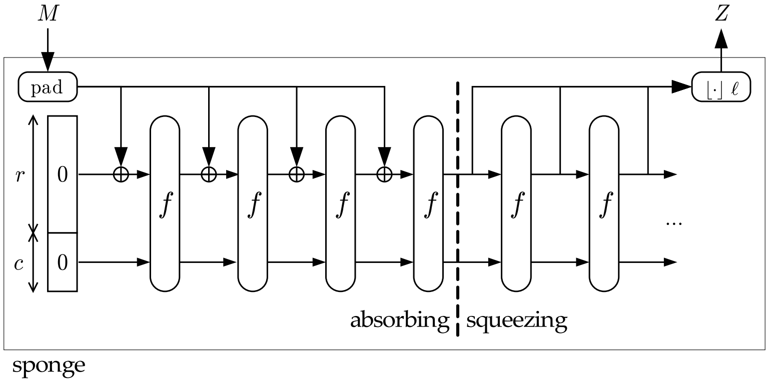
- Difference inputs M and M' giving the same chaining value
- Messages $M||X$ and $M'||X$ always collide for any string X

How to deal with internal collisions?

- $\mathcal{RO}$ has no internal collisions
  - If truncated to n bits, it does have collisions, say M and M'
  - But $M||X$ and $M'||X$ collide only with probability 2^{-n}
  - Random oracle has “infinite memory”
- Abandon *iterated modes* to meet the $\mathcal{RO}$ ideal?
  - In-memory hashing, non-streamable hash functions?
  - Model for finite memory, internal collisions!

Outline

The sponge construction



- r bits of *rate*
- c bits of *capacity*

Flat sponge claim

Simplifying the claim to a single parameter

Flat sponge claim with claimed capacity c

For any attack, the success probability is not above the sum of that for a $\mathcal{RO}$ and $N^2/2^{c+1}$, with N the number of calls to f

What does a flat sponge claim state?

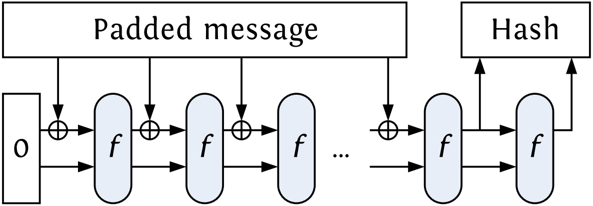
- Example: $c = 256$
- $N^2 / 2^{257}$ becomes significant when $N \approx 2^{128}$
- Collision-resistance:
  - Similar to that of random oracle up to $n = 256$
  - Maximum achievable security level: 2^{128}
- (2nd) pre-image resistance:
  - Similar to that of random oracle up to $n = 128$
  - Maximum achievable security level: 2^{128}
- Flat sponge claim forms a ceiling to the security claim

The NIST SHA-3 security requirements

Output length	224	256	384	512
Collision resistance	2^{112}	2^{128}	2^{192}	2^{256}
Pre-image resistance	2^{224}	2^{256}	2^{384}	2^{512}
2nd pre-image resistance	$2^{224}/\ell$	$2^{256}/\ell$	$2^{384}/\ell$	$2^{512}/\ell$

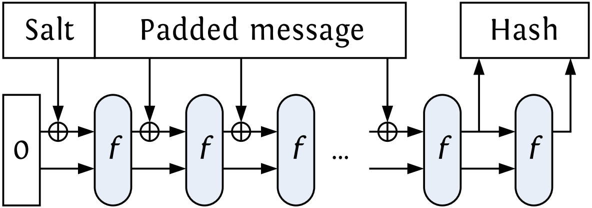
ℓ = message length

How to use a sponge function?



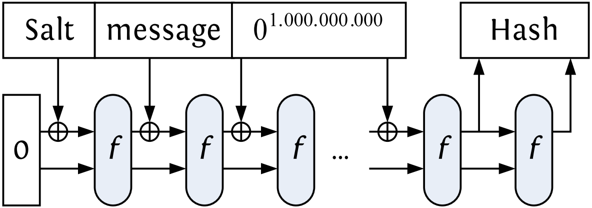
- For regular hashing

How to use a sponge function?



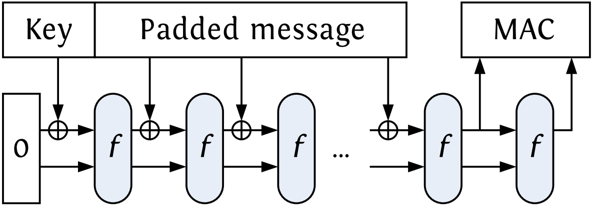
- For salted hashing

How to use a sponge function?



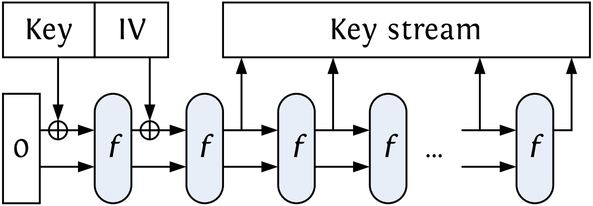
- For salted hashing, as slow as you like it

How to use a sponge function?



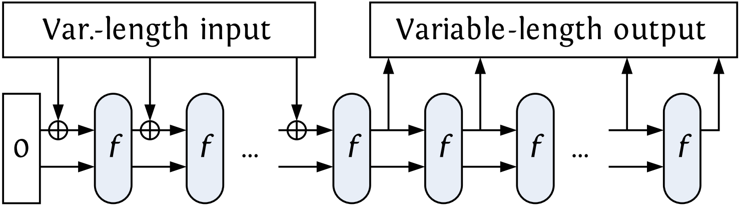
- As a message authentication code

How to use a sponge function?



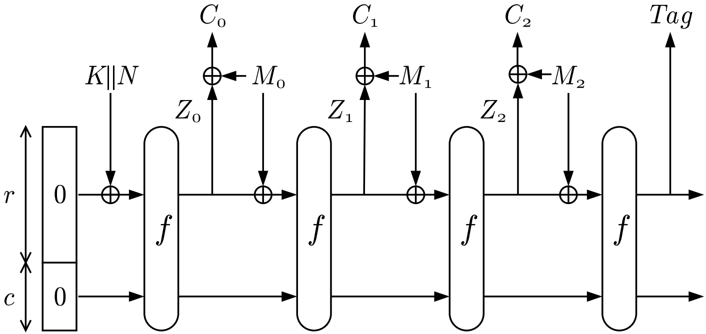
- As a stream cipher

How to use a sponge function?



- As a mask generating function [PKCS#1, IEEE Std 1363a]

Both encryption and MAC?



Absorbing and squeezing

Sponge functions: are they real?

			Width b
KECCAK	Bertoni, Daemen, Peeters, Van Assche	SHA-3 2008	25, 50, 100, 200 400, 800, 1600
Quark	Aumasson, Henzen, Meier, Naya-Plasencia	CHES 2010	136, 176 256
Photon	Guo, Peyrin, Poschmann	Crypto 2011	100, 144, 196, 256, 288
Spongent	Bogdanov, Knezevic, Leander, Toz, Varici, Verbauwhede	CHES 2011	88, 136, 176 248, 320

Outline

The beginning

- SUBTERRANEAN: Daemen (1991)
  - variable-length input and output
  - hashing and stream cipher
  - round function interleaved with input/output
- STEPRIGHTUP: Daemen (1994)
- PANAMA: Daemen and Clapp (1998)
- RADIOGATÚN: Bertoni, Daemen, Peeters and VA (2006)
  - experiments did not inspire confidence in RADIOGATÚN
  - NIST SHA-3 deadline approaching ...
  - U-turn: design a sponge with strong permutation f
- KECCAK (2008)

Designing the permutation KECCAK-f

Our mission

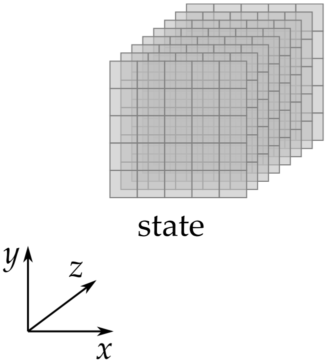
To design a permutation called KECCAK-f that cannot be distinguished from a random permutation.

- Classical LC/DC criteria
  - absence of large differential propagation probabilities
  - absence of large input-output correlations
- Immunity to
  - integral cryptanalysis
  - algebraic attacks
  - slide and symmetry-exploiting attacks
  - ...

KECCAK

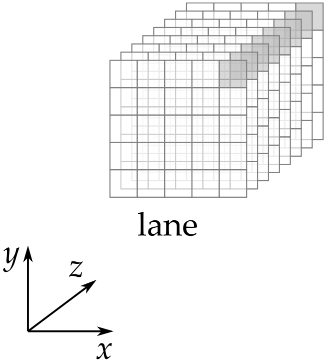
- Instantiation of a *sponge function*
- KECCAK uses a permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
- Security-speed trade-offs using the same permutation
- Examples
  - SHA-3: $r = 1024$ and $c = 576$ for $2^{c/2} = 2^{288}$ security
  - lightweight: $r = 40$ and $c = 160$ for $2^{c/2} = 2^{80}$ security

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



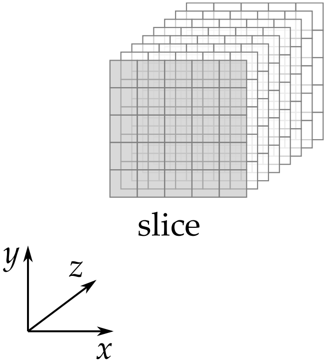
- 5×5 lanes, each containing 2^{ℓ} bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^{ℓ} of them

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



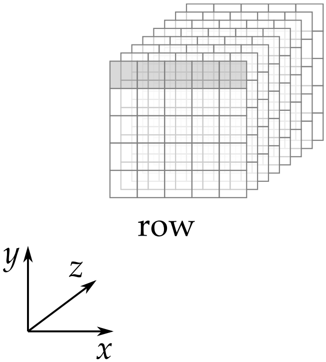
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

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



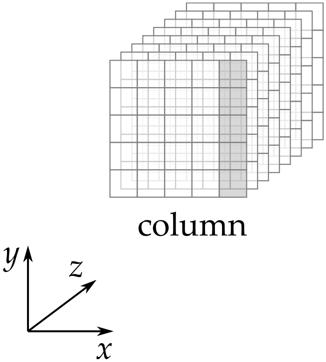
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

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



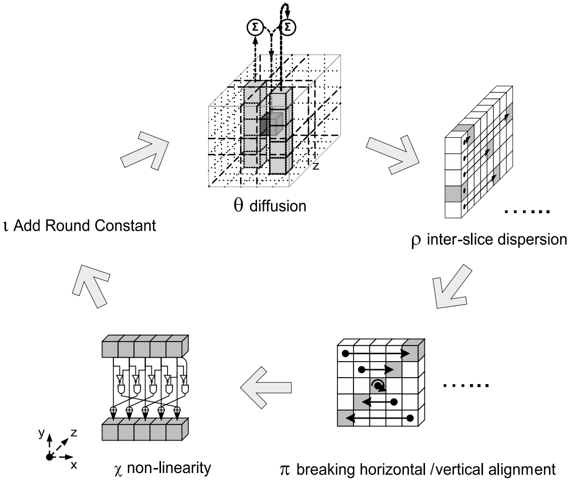
- 5×5 lanes, each containing 2^{ℓ} bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^{ℓ} of them

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

The step mappings of KECCAK-f



Outline

Status

- Number of capacities for the drop-in, c=1024 ruled out
- Variable length output
- Tree hashing
- PRNG and authenticated encryption

Questions?

Thanks for your attention!

Q?

More information on
http://keccak.noekeon.org/
http://sponge.noekeon.org/