Symmetric Cryptography 2.0

Guido Bertoni[1]

based on joint work with
Joan Daemen[2], Seth Hoffert, Michaël Peeters[3], Gilles Van Assche[3]
and Ronny Van Keer[3]

[1]Security Pattern [2]Radboud University [3]STMicroelectronics

Politecnico di Torino 2019, May 10th, 2019

Symmetric crypto: what textbooks say

Symmetric cryptographic primitives:

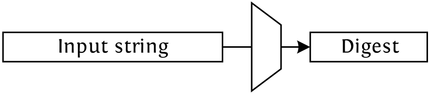
- Block ciphers
- Stream ciphers
- Hash functions

And their modes-of-use

Picture by GlasgowAmateur

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...



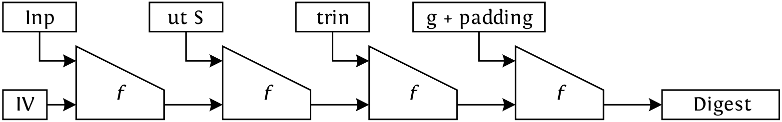
- Applications in cryptography
  - *Signatures*: $\text{sign}_{\text{RSA}}(h(M))$ instead of $\text{sign}_{\text{RSA}}(M)$
  - *Key derivation*: master key K to derived keys $(K_i = h(K\|i))$
  - *Bit commitment, predictions*: $h(\text{what I know})$
  - *Message authentication*: $h(K\|M)$
  - ...

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

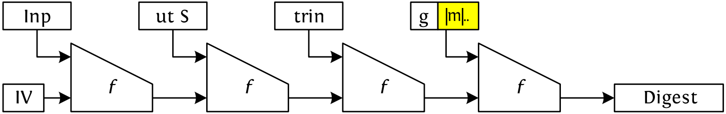
The chaining structure: Merkle-Damgård

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



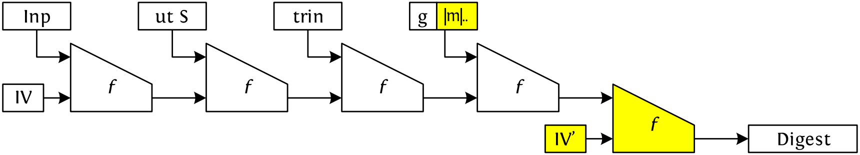
Merkle-Damgård strengthening

- Input length added to the input string



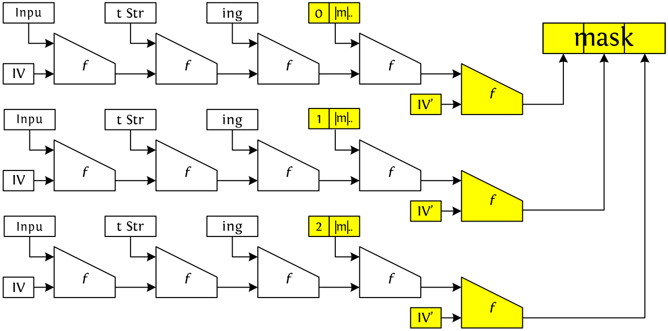
Enveloped Merkle-Damgård

- Special processing for last call

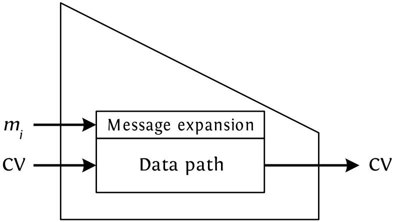


Variable-output-length Merkle-Damgård

- Mask generating function (MGF)



The compression function: Davies-Meyer (nearly)

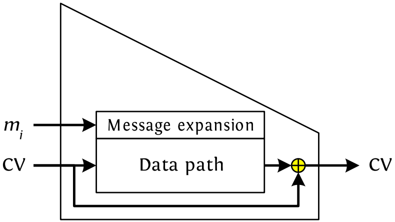


Uses a block cipher:

- Separated data path and message expansion

But not one-way!

The compression function: Davies-Meyer



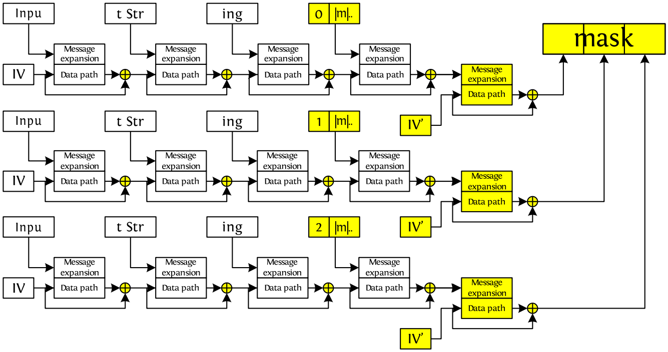
Uses a block cipher:

- 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
- ARX pretty complex to protect against side channel attacks

Cryptanalysis escalation

- 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

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

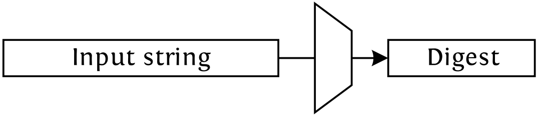
- Open competition organized by NIST
  - NIST provides forum
  - scientific community contributes: designs, attacks, implementations, comparisons
  - NIST draws conclusions and decides
- Goal: replacement for the SHA-2 family
  - 224, 256, 384 and 512-bit output sizes
  - other output sizes are optional
- 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
- 2015: FIPS 202 standard, SHA-3 and SHAKEs

Traditional security requirements of hash functions

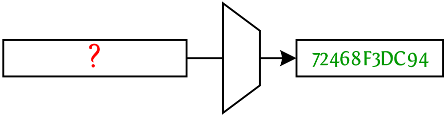
- Function h from $\mathbf{Z}_2^*$ to $\mathbf{Z}_2^n$



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

Pre-image resistance

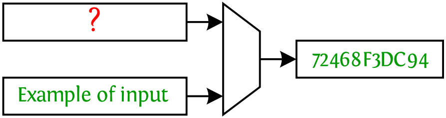
- Given $y \in \mathbf{Z}_2^n$, find $x \in \mathbf{Z}_2^*$ such that $h(x) = y$
- **Example**: given derived key $K_1 = h(K\|1)$, find master key K



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

2nd pre-image resistance

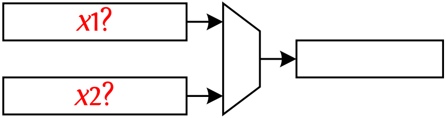
- Given $x \in \mathbf{Z}_2^*$, find $x' \neq x$ such that $h(x') = h(x)$
- **Example**: signature forging
  - given M and sign$(h(M))$, find another M' with equal signature



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

Collision resistance

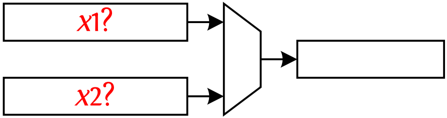
- Find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$



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

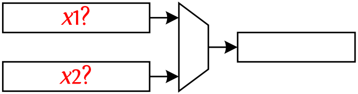
Collision resistance

- 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!

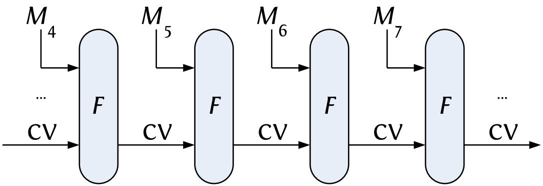
Random oracle RO

- 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

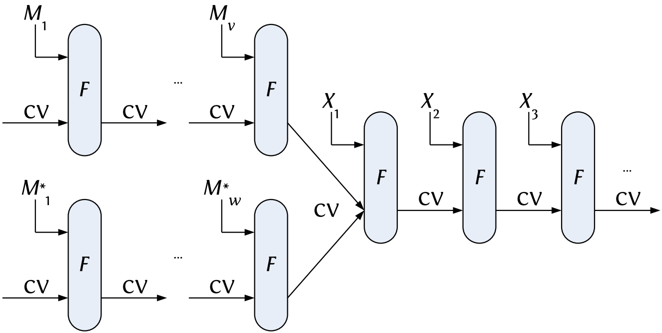
- Truncated to n bits, *RO* has all desired properties, e.g.,
  - Generating a collision: $2^{n/2}$
  - Finding a (2nd) pre-image: 2^n
  - And [my chosen requirement]: $f(n)$
- Proposal for a compact security claim:
  - "My function h behaves as a **random oracle**"
- Does not work, unfortunately

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!

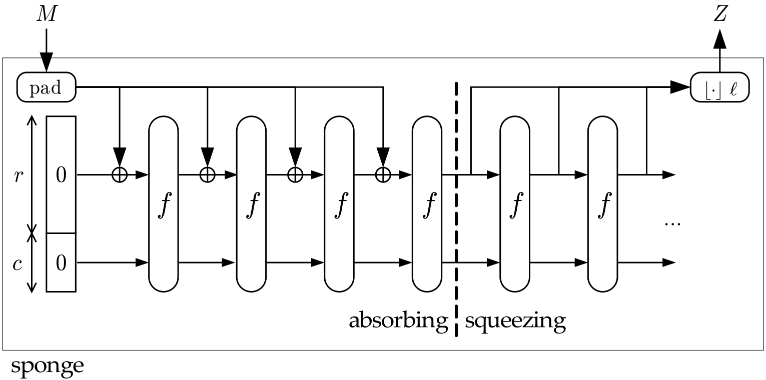


- Difference inputs M and M' giving the same chaining value
- Messages $M||X$ and $M'||X$ always collide for any string X

How to deal with internal collisions?

- *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 *RO* ideal?
  - In-memory hashing, non-streamable hash functions?
  - Model for finite memory, internal collisions!

The sponge construction

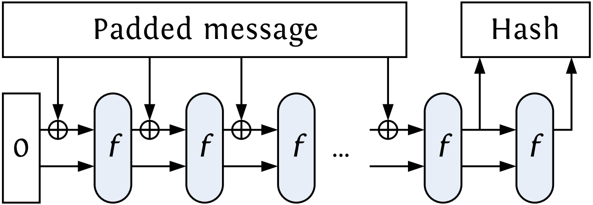


- r bits of *rate*
- c bits of *capacity*
- Flat sponge claim: security is $2^{c/2}$

What does a flat sponge claim state?

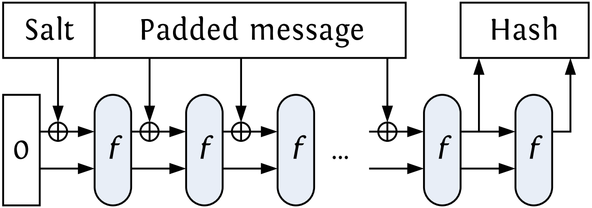
- Example: $c = 256$
- Collision-resistance:
  - Similar to that of random oracle up to $n = 256$
  - Maximum achievable security level: 2^{128}
- (2nd) pre-image resistance:
  - Similar to that of random oracle up to $n = 128$
  - Maximum achievable security level: 2^{128}
- Flat sponge claim forms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

How to use a sponge function?



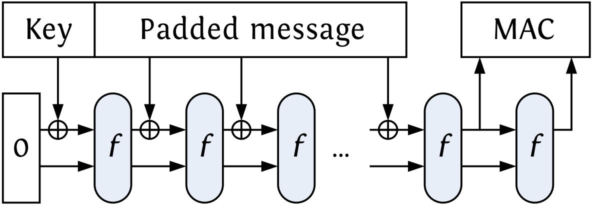
- For regular hashing

How to use a sponge function?



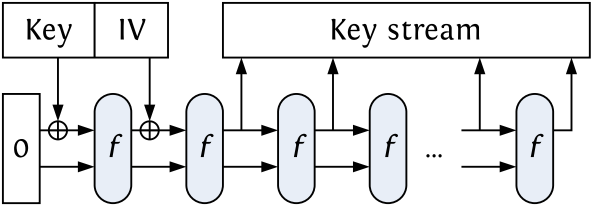
- For salted hashing

How to use a sponge function?



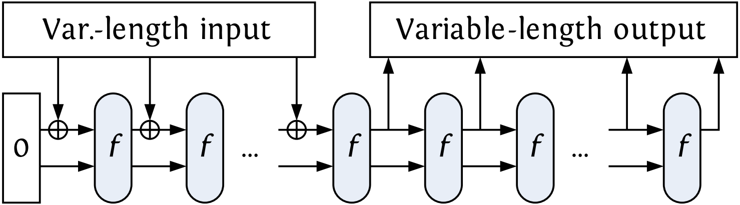
- As a message authentication code

How to use a sponge function?



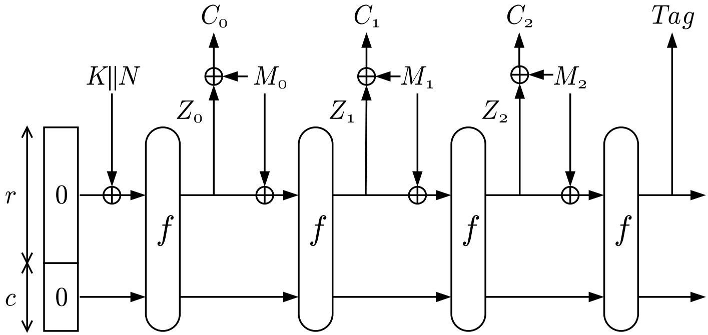
- As a stream cipher

How to use a sponge function?



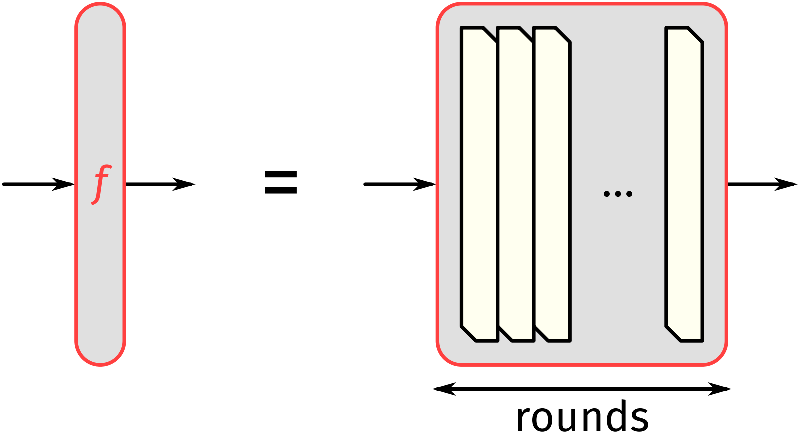
- As a mask generating function [PKCS#1, IEEE Std 1363a]

Both encryption and MAC?



Absorbing and squeezing

Inside the permutation



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

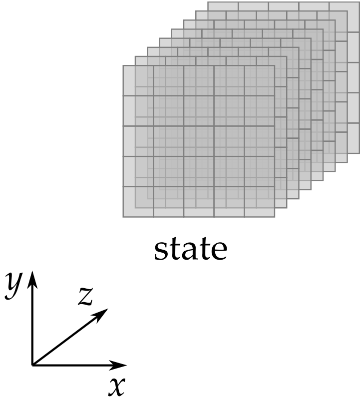
To design a permutation called KECCAK-f that cannot be distinguished from a random permutation.

- 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

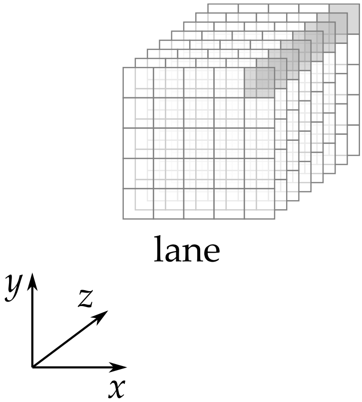
- Instantiation of a *sponge function*
- KECCAK uses a permutation KECCAK-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
- Security-speed trade-offs using the same permutation
- Examples
  - SHA-3-256: $r = 1088$ and $c = 512$ for $2^{c/2} = 2^{256}$ security
  - lightweight: $r = 40$ and $c = 160$ for $2^{c/2} = 2^{80}$ security

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



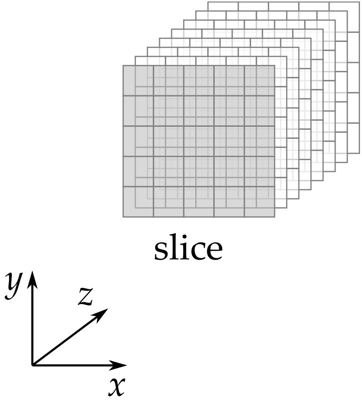
- 5×5 lanes, each containing 2^{ℓ} bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^{ℓ} of them

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



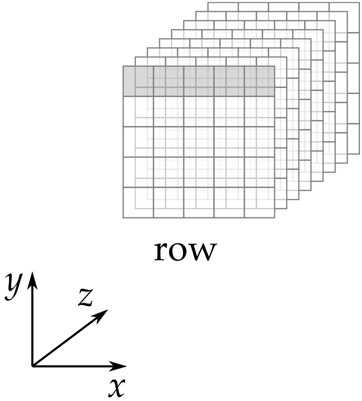
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

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



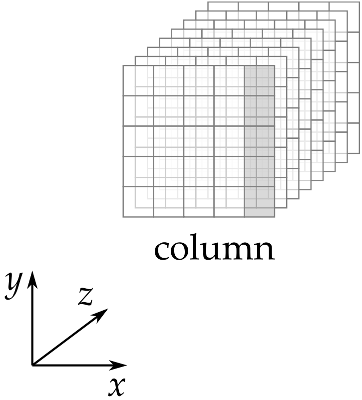
- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

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

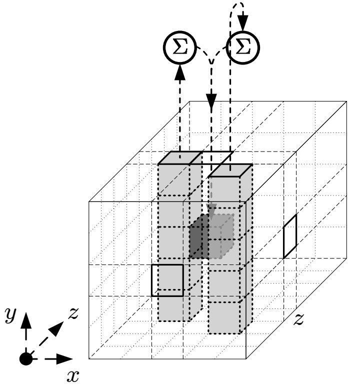


- 5×5 lanes, each containing 2^ℓ bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^ℓ of them

The Rounds of KECCAK-f

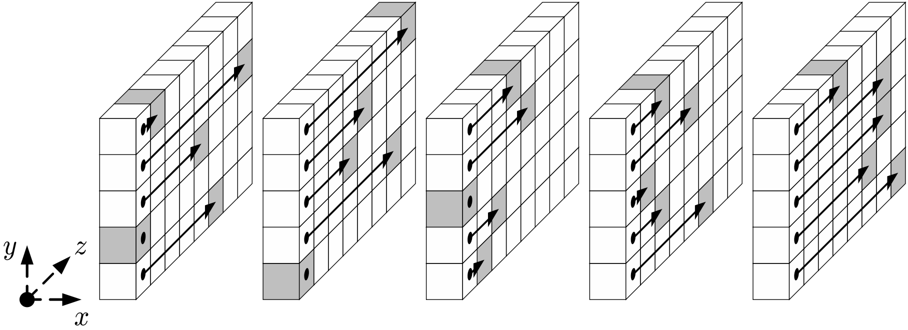
- A round consists of 5 invertible step mappings
  - θ for diffusion
  - ρ for inter-slice dispersion
  - π for disturbing horizontal/vertical alignment
  - χ for non-linearity
  - ι to break symmetry
- Number of rounds: $12 + 2\ell$
  - KECCAK-$f[25]$ has 12 rounds
  - KECCAK-$f[1600]$ has 24 rounds

θ for Diffusion



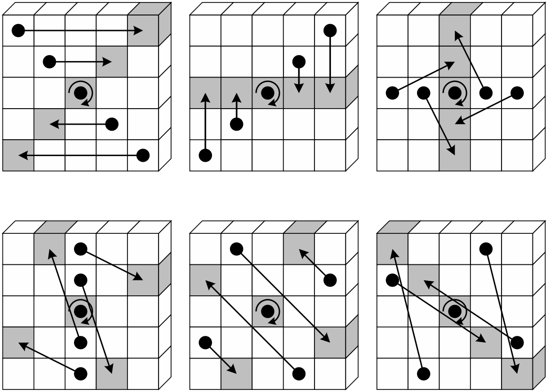
- To each bit, the parities of two columns are added
- Each input bit affects 11 output bits

ρ for Inter-Slice Dispersion



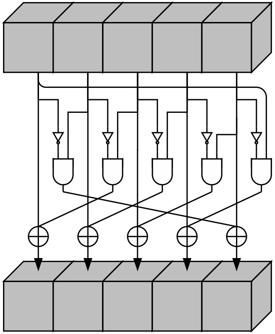
- Each lane is translated (cyclically) by a different amount
- Moves bits of a slice to 25 different slices

π for Disturbing Horizontal/Vertical Alignment



- Transposition of lanes
- Cycle with period 24 around a fixed origin

χ for Non-Linearity

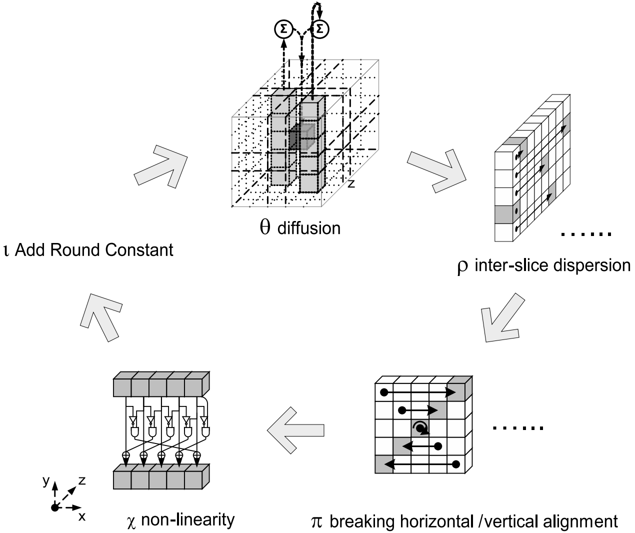


- Simple nonlinear mapping with well-understood properties
- Algebraic degree 2

ι to Break Symmetry

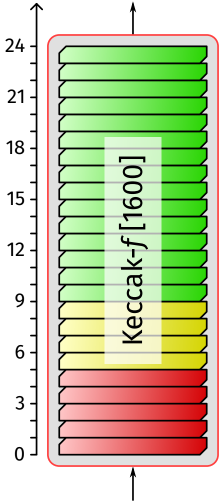
- XOR of round-dependent constant to lane in origin $(x = 0, y = 0)$
- Without ι, the round mapping would be symmetric
  - Invariant to translation in the z-direction
  - Advantage in analysis: *Matryoshka* structure
- Without ι, all rounds would be the same
  - Susceptibility to *slide* attacks
  - Defective cycle structure

The step mappings of Keccak-f



Number of rounds $12 + 2\ell$: from 12 to 24

Status of KECCAK



- Practical (collision) attacks up to 5 rounds
- Theoretical collision attacks up to 6 rounds
  [Qiao, Song, Liu, Guo 2016]
- Theoretical attack up to 9 rounds (2^{256} time...)
  [Dinur, Morawiecki, Pieprzyk, Srebrny, Straus 2014]

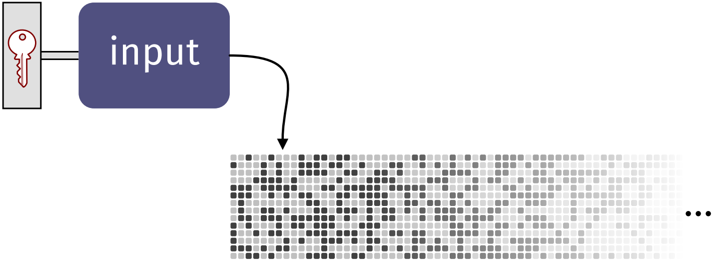
Round function unchanged since 2008

`https://keccak.team/third_party.html`

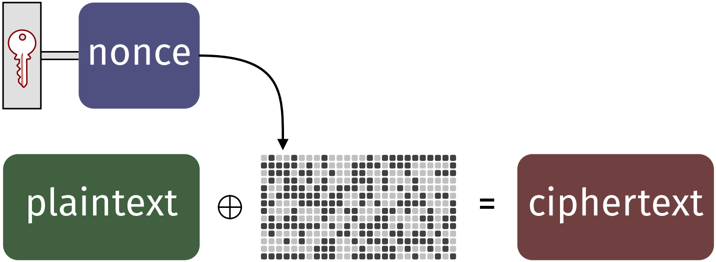
Another point of view

- All the primitives presented so far are the result of an incremental research,
  - starting from the design of hash function
- It is possible to reconsider the structure of symmetric primitives

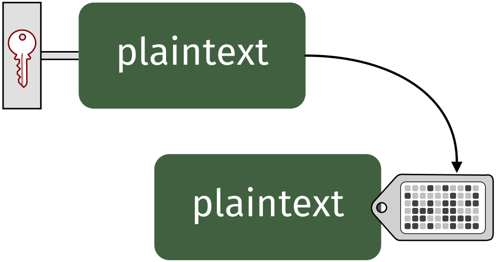
Pseudo-random function (PRF)



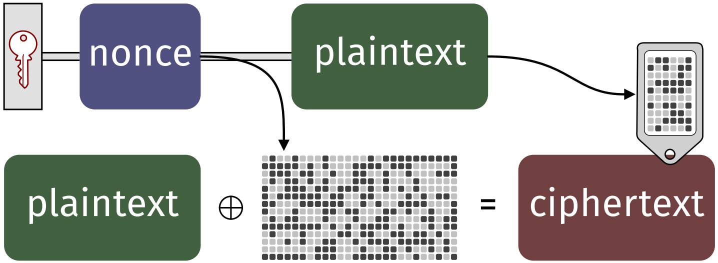
Stream encryption



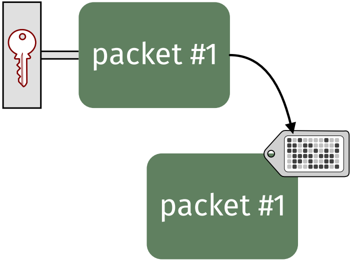
Message authentication (MAC)



Authenticated encryption

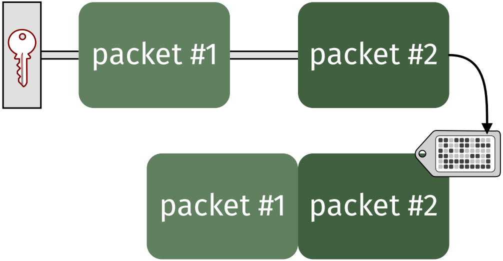


String sequence input and incrementality



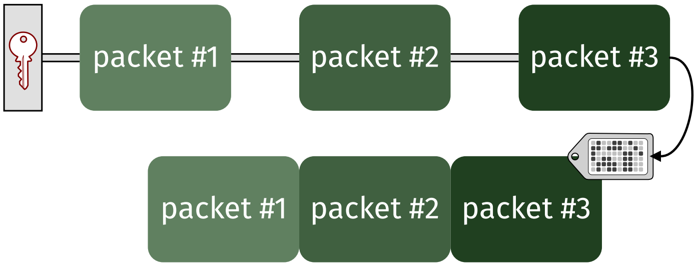
$$F_K\left(P^{(1)}\right)$$

String sequence input and incrementality



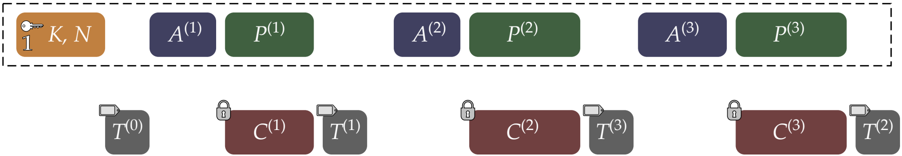
$$F_K\left(P^{(2)} \circ P^{(1)}\right)$$

String sequence input and incrementality

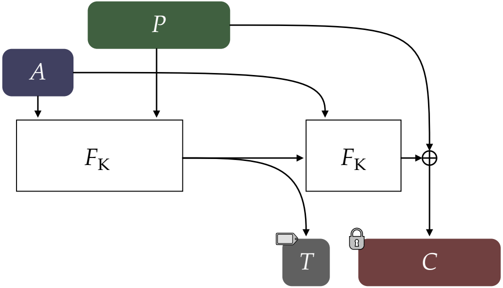


$$F_K\left(P^{(3)} \circ P^{(2)} \circ P^{(1)}\right)$$

Session authenticated encryption (SAE) [KT, SAC 2011]



Synthetic initialization value (SIV) of [KT, eprint 2016/1188]



Unwrap taking metadata A, ciphertext C and tag T
$P \leftarrow C + F_K(T \circ A)$
$\tau \leftarrow 0^t + F_K(P \circ A)$
if $\tau \neq T$ **then return** error!
else return plaintext P of length $|C|$

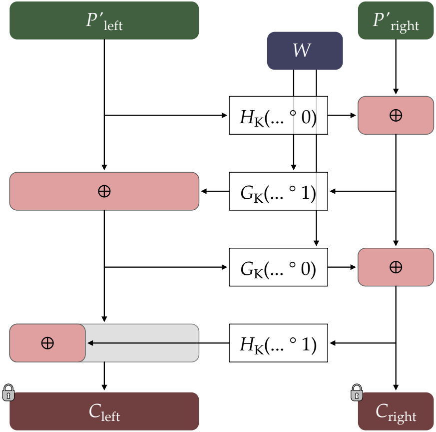
Variant of SIV of [Rogaway & Shrimpton, EC 2006]

Wide block cipher (WBC), as in [KT, eprint 2016/1188]

Encipher P with K and tweak W

$$
\begin{array}{llcl}
(L, R) & \leftarrow & & \mathrm{split}(P) \\
R_0 & \leftarrow R_0 & + & H_K(L \circ 0) \\
L & \leftarrow L & + & G_K\,(R \circ W \circ 1) \\
R & \leftarrow R & + & G_K\,(L \circ W \circ 0) \\
L_0 & \leftarrow L_0 & + & H_K(R \circ 1) \\
C & \leftarrow L & \| & R
\end{array}
$$

return ciphertext C of length $|P|$



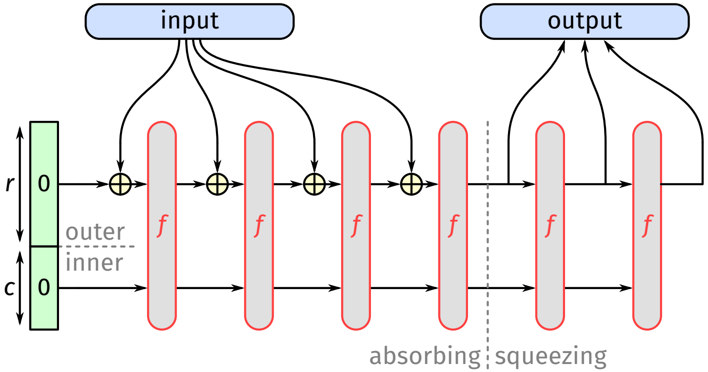
Inspired by HHFHFH of [Bernstein, Nandi & Sarkar, Dagstuhl 2016]

How to build a PRF?

How to build a PRF?

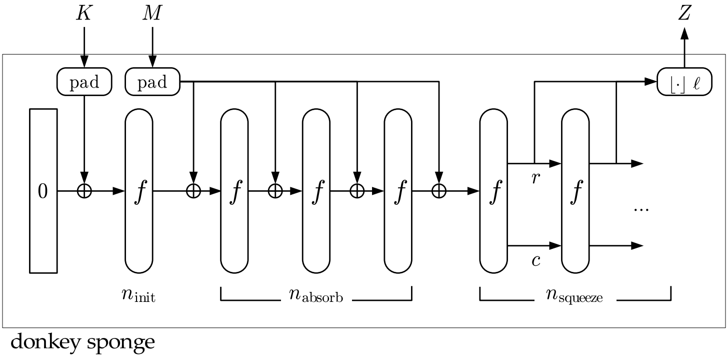
By icelight (flickr.com)

Sponge [Keccak Team, Ecrypt 2007]

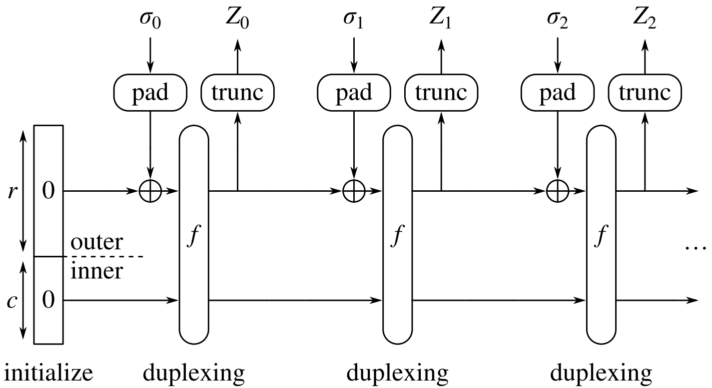


- Taking *K* as first part of input gives a PRF

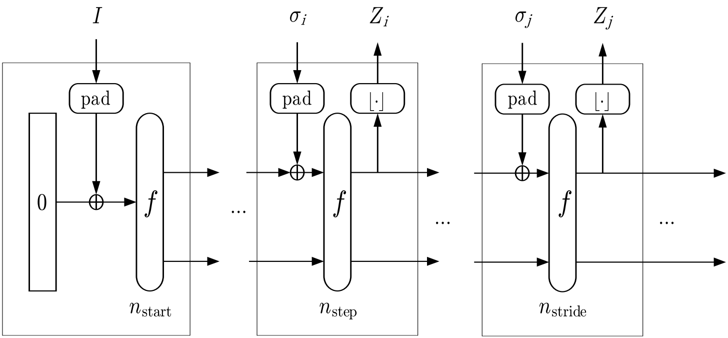
More efficient: donkeySponge [Keccak Team, DIAC 2012]



Incrementality: duplex [Keccak Team, SAC 2011]



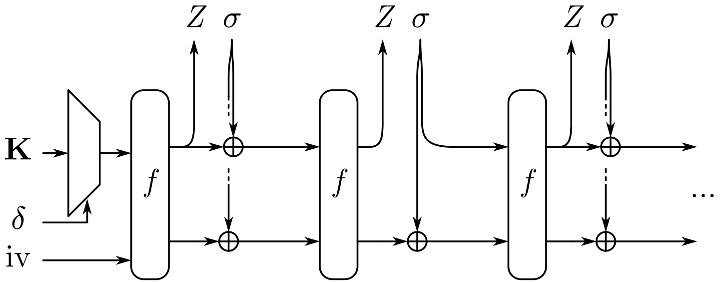
More efficient: MonkeyDuplex [Keccak Team, DIAC 2012]



Instances:

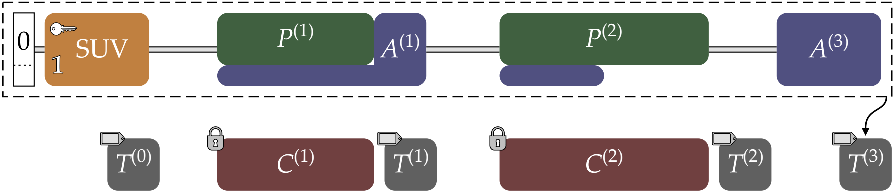
- KETJE [Keccak Team, now extended with Ronny Van Keer, CAESAR 2014]
- + half a dozen other CAESAR submissions

Consolidation: Full-state keyed duplex

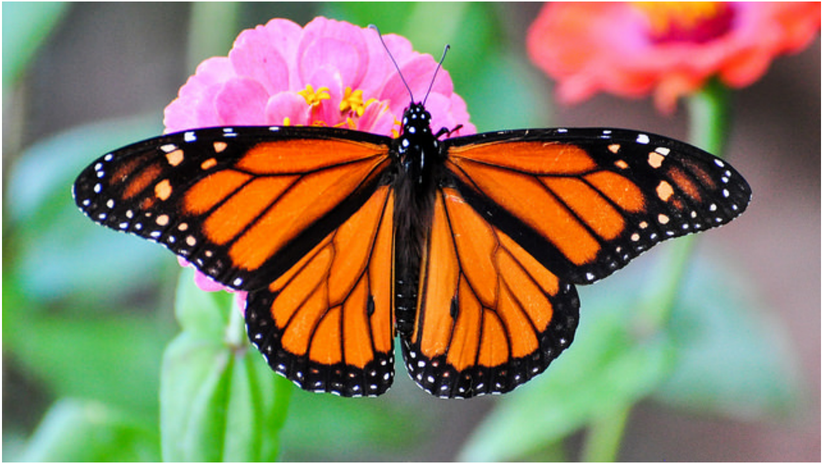


[Mennink, Reyhanitabar, & Vizar, Asiacrypt 2015]
[Daemen, Mennink & Van Assche, Asiacrypt 2017]

SAE with full-state keyed duplex: Motorist [KT, Keyak 2015]

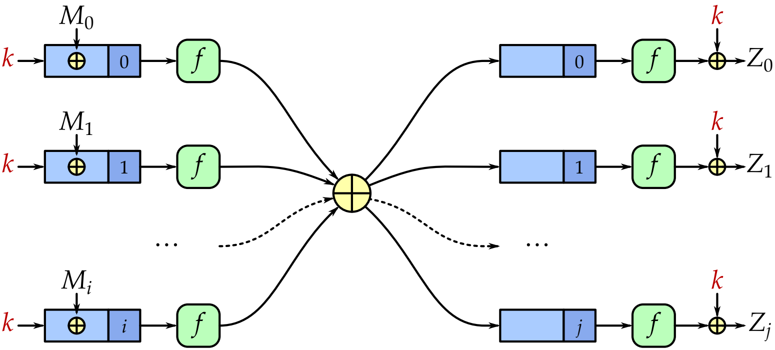


How to build a parallelizable PRF?



by Peter Miller (flick.com)

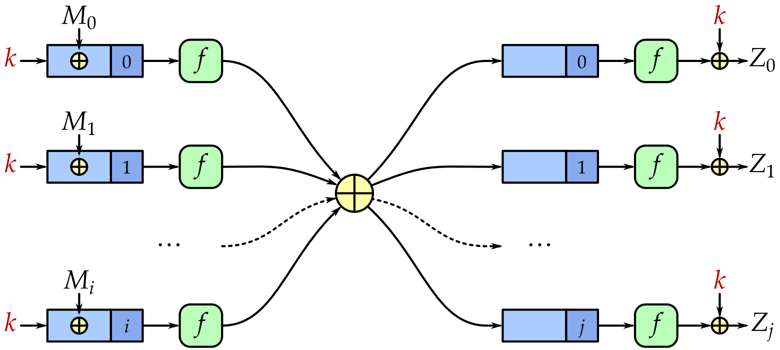
Farfalle: early attempt [KT 2014-2016]



Similar to Protected Counter Sums [Bernstein, "stretch", JOC 1999]

Problem: collisions with higher-order differentials if f has low degree

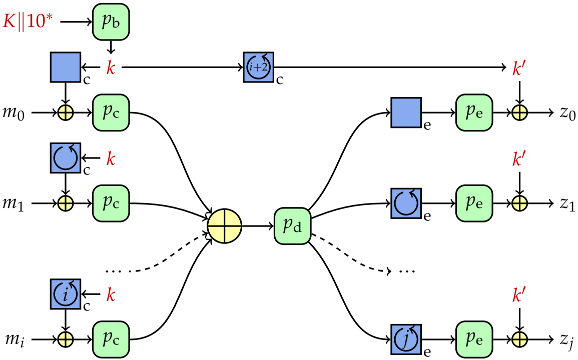
Farfalle: early attempt [KT 2014-2016]



Similar to Protected Counter Sums [Bernstein, ”stretch”, JOC 1999]

Problem: collisions with higher-order differentials if f has low degree

Farfalle now [Keccak Team + Seth Hoffert, ToSC 2017]

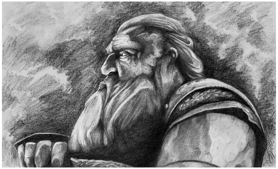


- Input mask rolling and p_c against accumulator collisions
- State rolling, p_e and output mask against state retrieval at output
- Middle p_d against higher-order DC
- Input-output attacks have to deal with $p_e \circ p_d \circ p_c$

The permutation that fits

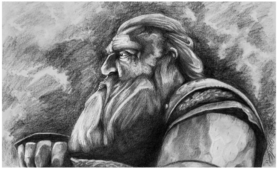
- Keccak-*p* designed to fulfill SHA-3 requirements
- Some proposals for lightweight crypto
  - Spongent, Quark, Photon..
- Now new directions for designing permutations:
  - Efficient on different platform
  - Right level of security

Gimli [Bernstein, Kölbl, Lucks, Massolino, Mendel, Nawaz, Schneider, Schwabe, Standaert, Todo, Viguier, CHES 2017]



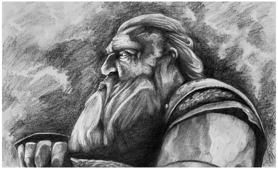
- has ideal size and shape: 48 bytes in 12 words of 32 bits
  - fits in registers of ARM Cortex M3/M4 and suitable for SIMD
- limits diffusion, see e.g. [Mike Hamburg, 2017]
  - no problem for nominal number of rounds: 24
  - not clear how many rounds needed in Farfalle

Gimli [Bernstein, Kölbl, Lucks, Massolino, Mendel, Nawaz, Schneider, Schwabe, Standaert, Todo, Viguier, CHES 2017]



- has ideal size and shape: 48 bytes in 12 words of 32 bits
  - fits in registers of ARM Cortex M3/M4 and suitable for SIMD
- limits diffusion, see e.g. [Mike Hamburg, 2017]
  - no problem for nominal number of rounds: 24
  - not clear how many rounds needed in Farfalle

Gimli [Bernstein, Kölbl, Lucks, Massolino, Mendel, Nawaz, Schneider, Schwabe, Standaert, Todo, Viguier, CHES 2017]



- has ideal size and shape: 48 bytes in 12 words of 32 bits
  - fits in registers of ARM Cortex M3/M4 and suitable for SIMD
- limits diffusion, see e.g. [Mike Hamburg, 2017]
  - no problem for nominal number of rounds: 24
  - not clear how many rounds needed in Farfalle

Xoodoo · [noun, mythical] · /zu: du:/ · Alpine mammal that lives in compact herds, can survive avalanches and is appreciated for the wide trails it creates in the landscape. Despite its fluffy appearance it is very robust and does not get distracted by side channels.

Xoodoo [Keccak team with Seth Hoffert and Johan De Meulder]

https://github.com/XoodooTeam/Xoodoo

- 384-bit permutation
- Main purpose: usage in Farfalle: XooPRF
  - Efficient on wide range of platforms
- But also for
  - small-state authenticated encryption, Ketje style
  - sponge-based hashing, ...

Keccak-p philosophy ported to Gimli dimensions $3 \times 4 \times 32$*!*

XOODOO [Keccak team with Seth Hoffert and Johan De Meulder]

https://github.com/XoodooTeam/Xoodoo

- 384-bit permutation
- Main purpose: usage in Farfalle: XOOPRF
  - Efficient on wide range of platforms
- But also for
  - small-state authenticated encryption, KETJE style
  - sponge-based hashing, ...

KECCAK-p philosophy ported to Gimli dimensions $3 \times 4 \times 32$!

Xoodoo [Keccak team with Seth Hoffert and Johan De Meulder]

https://github.com/XoodooTeam/Xoodoo

- 384-bit permutation
- Main purpose: usage in Farfalle: XooPRF
  - Efficient on wide range of platforms
- But also for
  - small-state authenticated encryption, Ketje style
  - sponge-based hashing, …

Keccak-p philosophy ported to Gimli dimensions $3 \times 4 \times 32$!

Xoodoo [Keccak team with Seth Hoffert and Johan De Meulder]

https://github.com/XoodooTeam/Xoodoo

- 384-bit permutation
- Main purpose: usage in Farfalle: XooPRF
  - Efficient on wide range of platforms
- But also for
  - small-state authenticated encryption, Ketje style
  - sponge-based hashing, ...

Keccak-p philosophy ported to Gimli dimensions $3 \times 4 \times 32$*!*

Xoodoo [Keccak team with Seth Hoffert and Johan De Meulder]

`https://github.com/XoodooTeam/Xoodoo`

- 384-bit permutation
- Main purpose: usage in Farfalle: Xoofff
  - Efficient on wide range of platforms
- But also for
  - small-state authenticated encryption, Ketje style
  - sponge-based hashing, ...

Keccak-p philosophy ported to Gimli dimensions $3 \times 4 \times 32$!

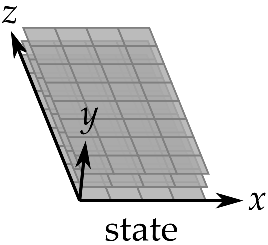
Xoodoo [Keccak team with Seth Hoffert and Johan De Meulder]

https://github.com/XoodooTeam/Xoodoo

- 384-bit permutation
- Main purpose: usage in Farfalle: XooPRF
  - Efficient on wide range of platforms
- But also for
  - small-state authenticated encryption, Ketje style
  - sponge-based hashing, ...

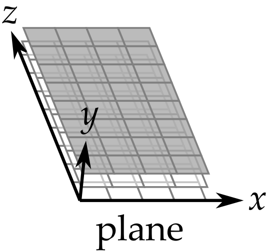
Keccak-p philosophy ported to Gimli dimensions $3 \times 4 \times 32$*!*

Xoodoo state



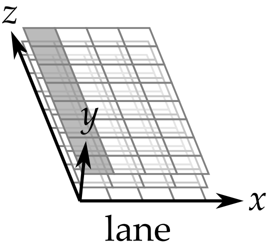
- State: 3 horizontal planes each consisting of 4 lanes

Xoodoo state



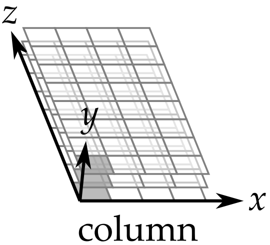
- State: 3 horizontal planes each consisting of 4 lanes

Xoodoo state



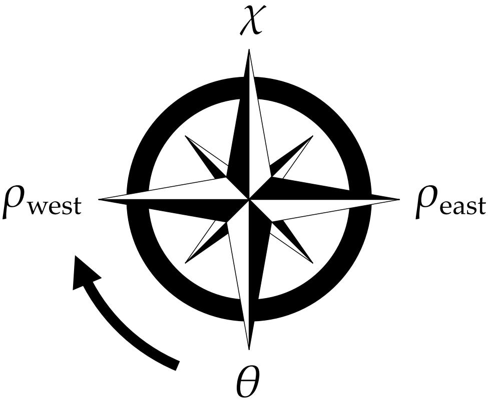
- State: 3 horizontal planes each consisting of 4 lanes

Xoodoo state



- State: 3 horizontal planes each consisting of 4 lanes

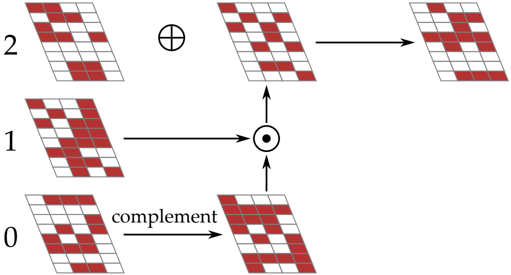
Xoodoo round function



Iterated: n_r rounds that differ only by round constant

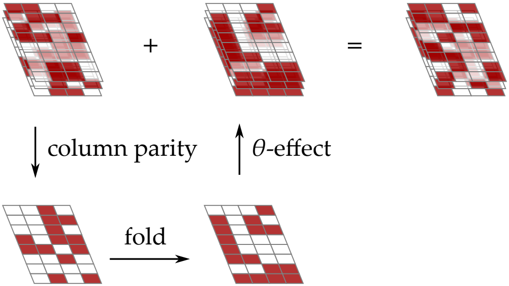
Nonlinear mapping χ

Effect on one plane:



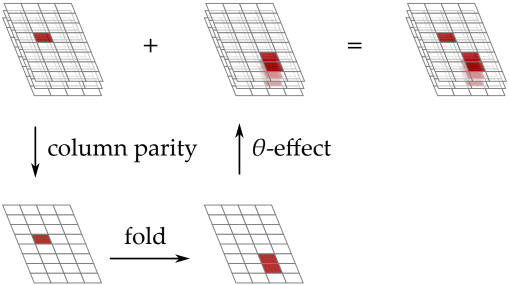
- χ as in Keccak-p, operating on 3-bit columns
- Involution and same propagation differentially and linearly

Mixing layer θ



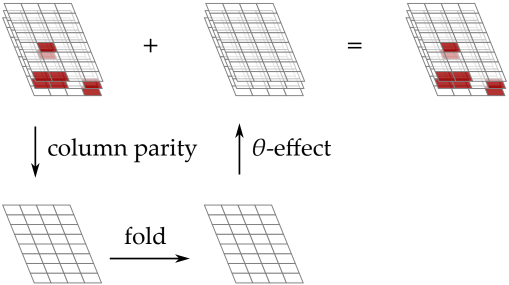
- Column parity mixer: compute parity, fold and add to state
- good average diffusion, identity for states in *kernel*

Mixing layer θ



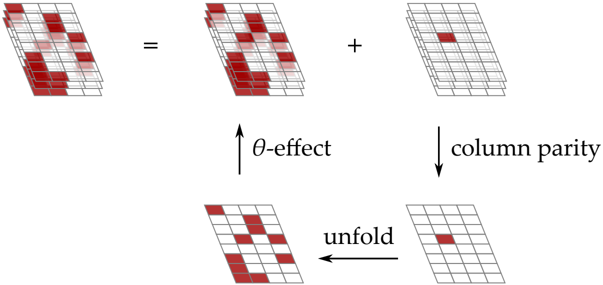
- Column parity mixer: compute parity, fold and add to state
- good average diffusion, identity for states in *kernel*

Mixing layer θ



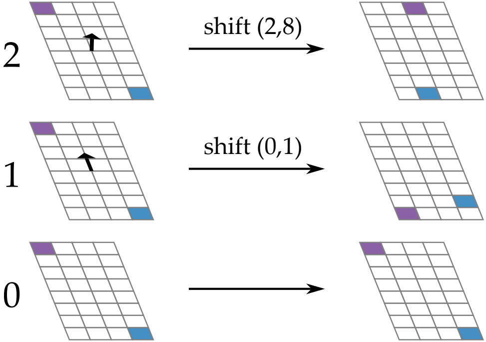
- Column parity mixer: compute parity, fold and add to state
- good average diffusion, identity for states in *kernel*

Mixing layer θ



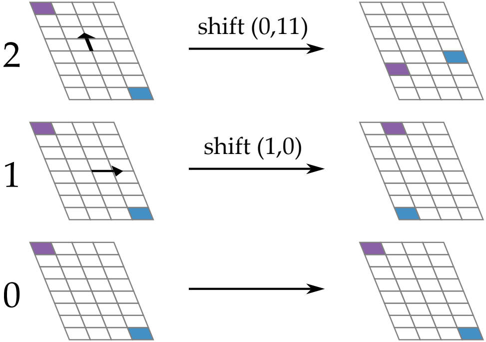
- Column parity mixer: compute parity, fold and add to state
- good average diffusion, identity for states in *kernel*

Plane shift ρ_{east}



- After χ and before θ
- Shifts planes $y = 1$ and $y = 2$ over different directions

Plane shift ρ_{west}



- After θ and before χ
- Shifts planes $y = 1$ and $y = 2$ over different directions

Xoodoo pseudocode

n_r rounds from $i = 1 - n_r$ to 0, with a 5-step round function:

$$
\begin{aligned}
\theta :\; & \\
& P \leftarrow A_0 + A_1 + A_2 \\
& E \leftarrow P \lll (1,5) + P \lll (1,14) \\
& A_y \leftarrow A_y + E \text{ for } y \in \{0,1,2\} \\
\rho_{\text{west}} :\; & \\
& A_1 \leftarrow A_1 \lll (1,0) \\
& A_2 \leftarrow A_2 \lll (0,11) \\
\iota :\; & \\
& A_{0,0} \leftarrow A_{0,0} + \text{rc}_i \\
\chi :\; & \\
& B_0 \leftarrow \overline{A_1} \cdot A_2 \\
& B_1 \leftarrow \overline{A_2} \cdot A_0 \\
& B_2 \leftarrow \overline{A_0} \cdot A_1 \\
& A_y \leftarrow A_y + B_y \text{ for } y \in \{0,1,2\} \\
\rho_{\text{east}} :\; & \\
& A_1 \leftarrow A_1 \lll (0,1) \\
& A_2 \leftarrow A_2 \lll (2,8)
\end{aligned}
$$

Xoodoo software performance

	width	cycles/byte per round	
	bytes	ARM Cortex M3	Intel Skylake
Keccak-$p[1600, n_r]$	200	2.44	0.080
ChaCha	64	0.69	0.059
Gimli	48	0.91	0.074*
Xoodoo	48	1.20	0.083

∗ on Intel Haswell

Cryptographic framework with a single permutation

- Network and security protocols require a complete set of symmetric primitives (hashing, AE, KDF, PRNG...)
- Keccak was already supplying most of the primitives [KT 2008]
- Duplex for AE and PRNG: the missing pieces [KT 2010]
- A first informal proposal of framework for IoT [KT CIoT 2012]
- Blinker [Saarinen, CT-RSA 2014]
- STROBE [Hamburg, ePrint 2017]
- DISCOCRYPTO [Wong, Black Hat Europe 2017]

What textbooks and intro's should say from now on :-)

Symmetric cryptographic primitives:

- **Permutations**
- Block ciphers
- Stream ciphers
- Hash functions

And their modes-of-use

Picture by Sébastien Wiertz

Questions?