Innovations in permutation-based crypto

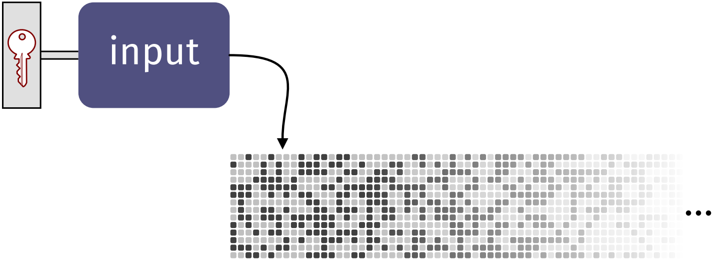
Joan Daemen[1,2]

based on joint work with
Guido Bertoni[3], Seth Hoffert, Michaël Peeters[1], Gilles Van Assche[1]
and Ronny Van Keer[1]

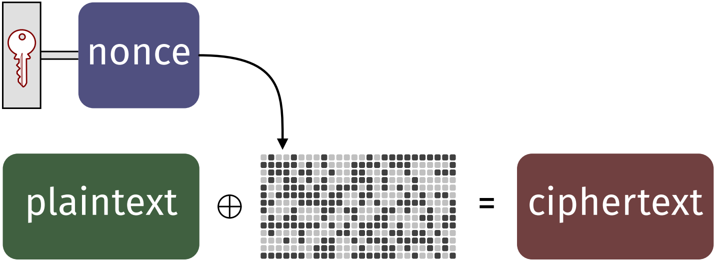
[1]STMicroelectronics [2]Radboud University [3]Security Pattern

ECC, Nijmegen, November 14, 2017

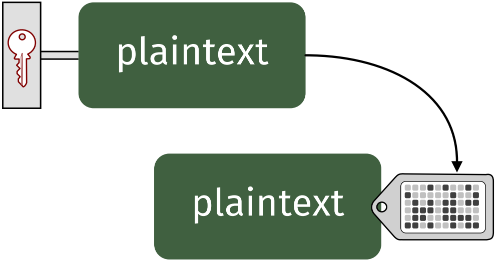
Pseudo-random function (PRF)



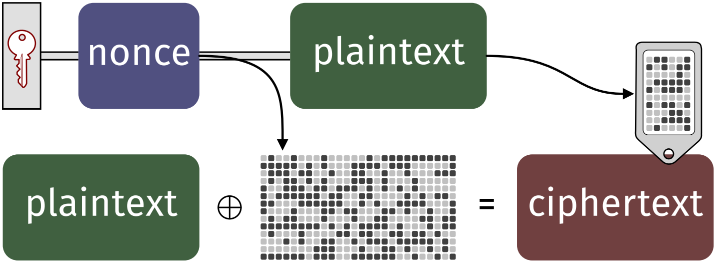
Stream encryption



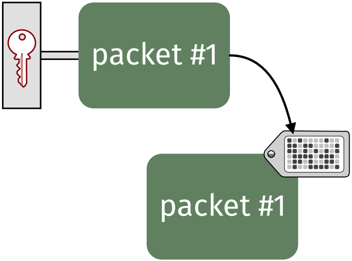
Message authentication (MAC)



Authenticated encryption

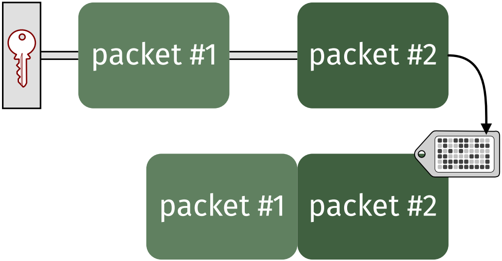


String sequence input and incrementality



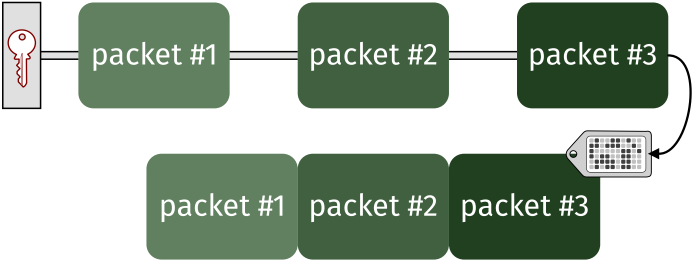
$$F_K\left(P^{(1)}\right)$$

String sequence input and incrementality



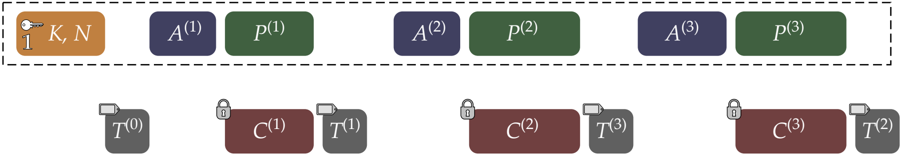
$$F_K\left(P^{(2)} \circ P^{(1)}\right)$$

String sequence input and incrementality



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

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



Initialization taking nonce N

$T \leftarrow 0^t + F_K(N)$

history $\leftarrow N$

return tag T of length t

Wrap taking metadata A and plaintext P

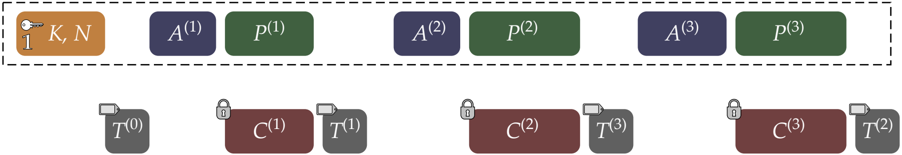
$C \leftarrow P + F_K(A \circ \text{history})$

$T \leftarrow 0^t + F_K(C \circ A \circ \text{history})$

history $\leftarrow C \circ A \circ$ history

return ciphertext C of length $|P|$ and tag T of length t

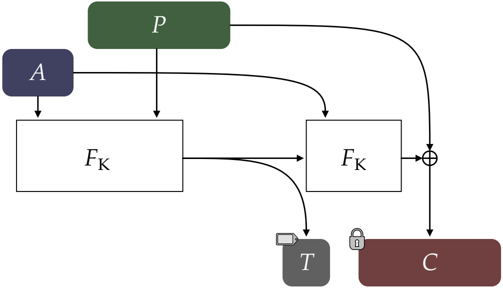
Session authenticated encryption (SAE) [KT, SAC 2011]



Initialization taking nonce N
$T \leftarrow 0^t + F_K(N)$
history $\leftarrow N$
return tag T of length t

Wrap taking metadata A and plaintext P
$C \leftarrow P + F_K(A \circ \text{history})$
$T \leftarrow 0^t + F_K(C \circ A \circ \text{history})$
history $\leftarrow C \circ A \circ$ history
return ciphertext C of length $|P|$ and tag T of length t

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 $|C|$

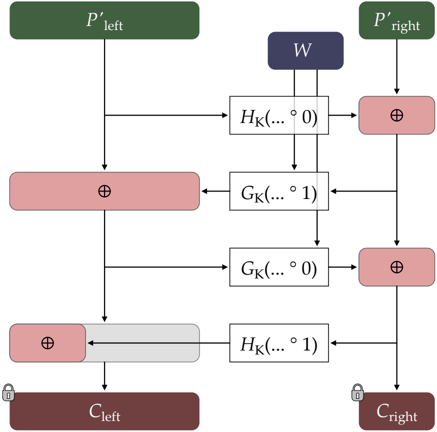
Variant of SIV of [Rogaway & Shrimpton, EC 2006]

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

Encipher P with K and tweak W

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

return ciphertext C of length $|P|$



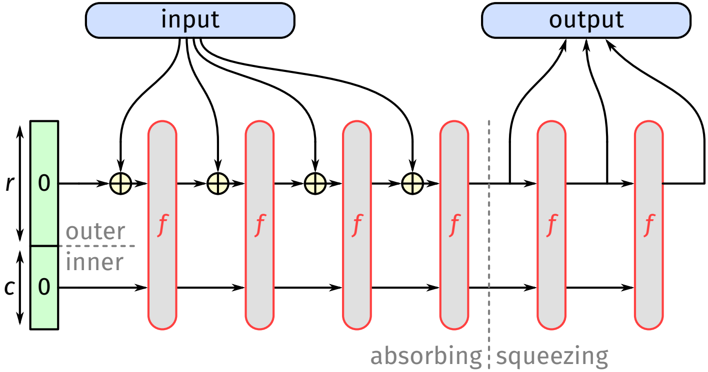
Inspired by HHFHFH of [Bernstein, Nandi & Sarkar, Dagstuhl 2016]

How to build a PRF?

How to build a PRF?

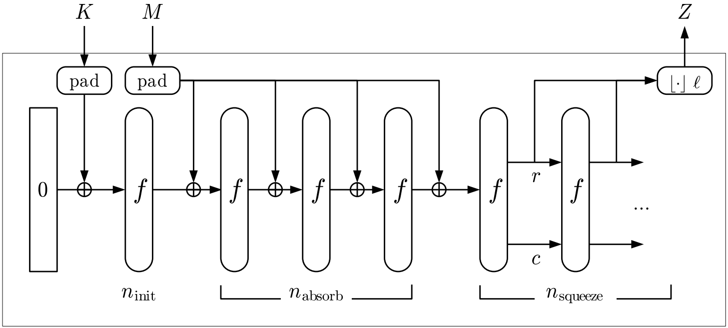
By icelight (flickr.com)

Sponge [Keccak Team, Ecrypt 2008]



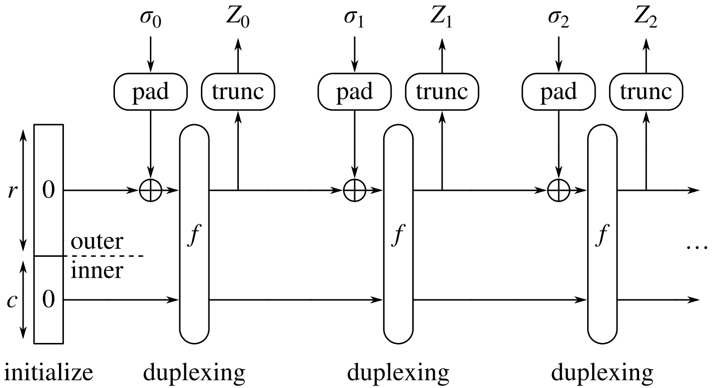
- Taking *K* as first part of input gives a PRF

More efficient: donkeySponge [Keccak Team, DIAC 2012]

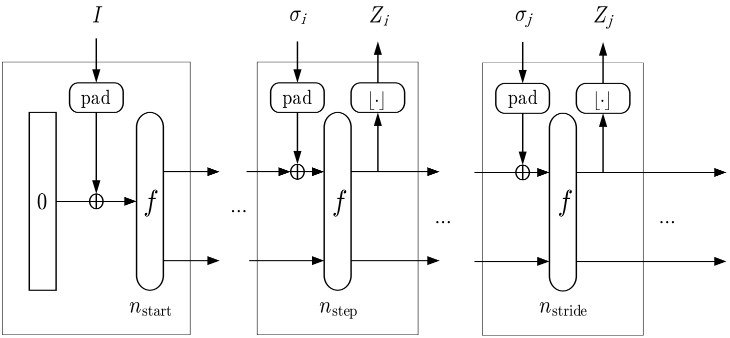


donkey sponge

Incrementality: duplex [Keccak Team, SAC 2011]



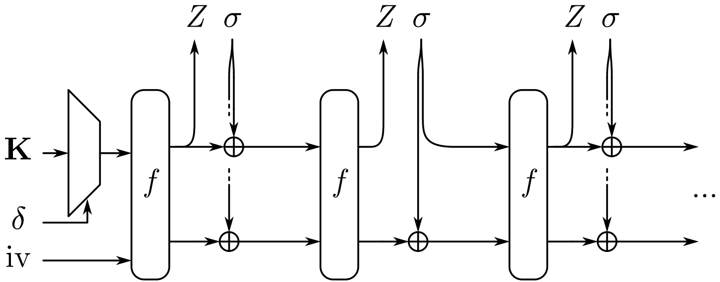
More efficient: MonkeyDuplex [Keccak Team, DIAC 2012]



Instances:

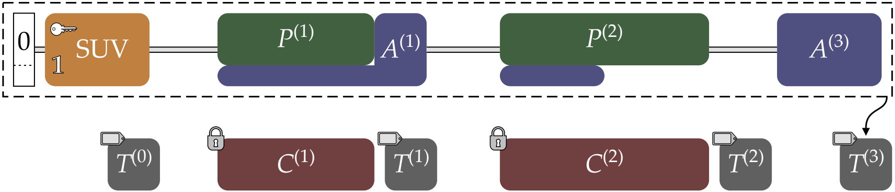
- KETJE [Keccak Team, now extended with Ronny Van Keer, CAESAR 2014]
- $+$ half a dozen other CAESAR submissions

Consolidation: Full-state keyed duplex

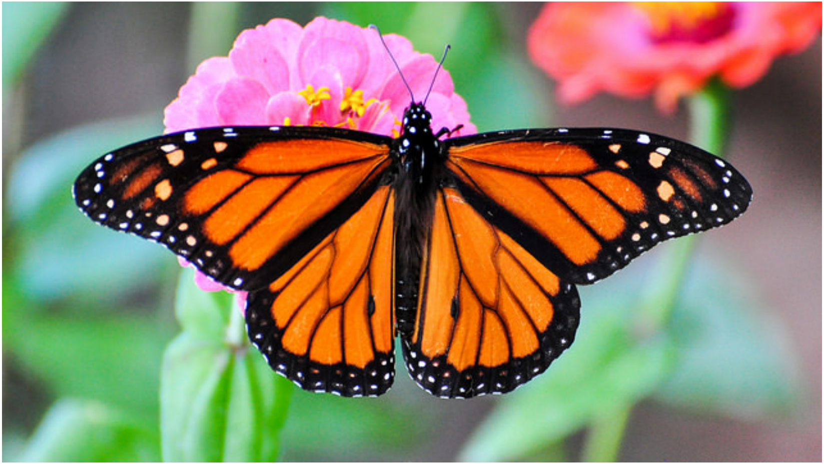


[Mennink, Reyhanitabar, & Vizar, Asiacrypt 2015]
[Daemen, Mennink & Van Assche, Asiacrypt 2017]

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

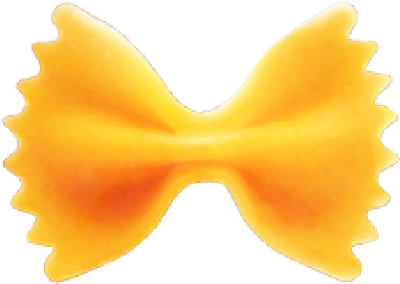


How to build a parallelizable PRF?



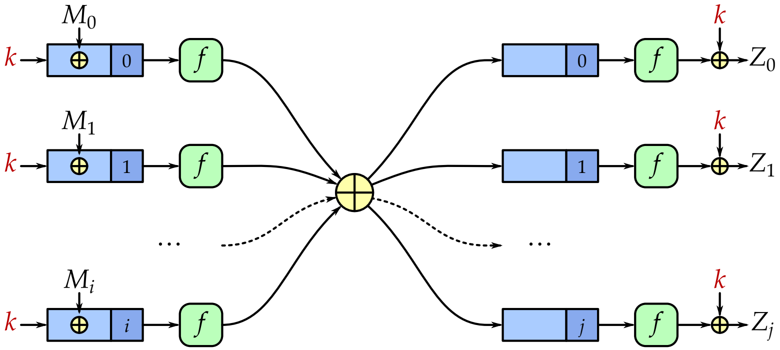
by Peter Miller (flick.com)

How to build a parallelizable PRF?



by Barilla Food Service

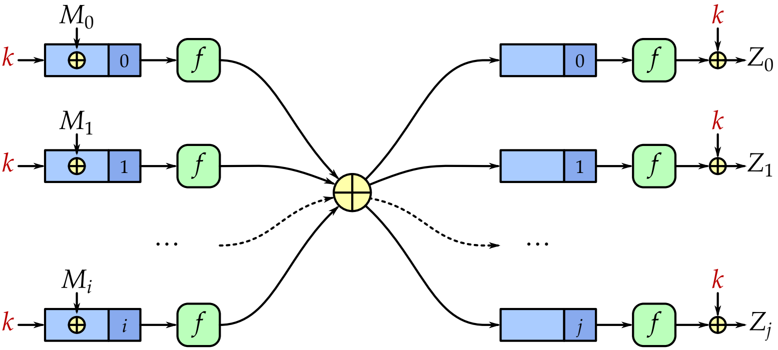
Farfalle: early attempt [KT 2014-2016]



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

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

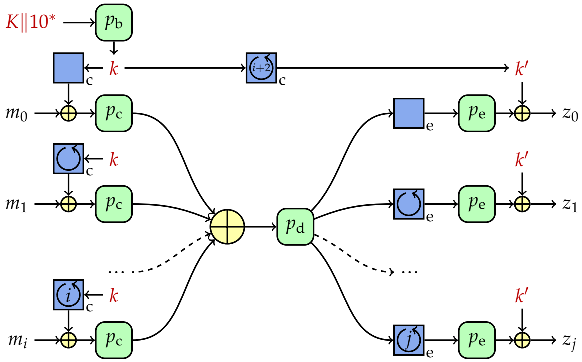
Farfalle: early attempt [KT 2014-2016]



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

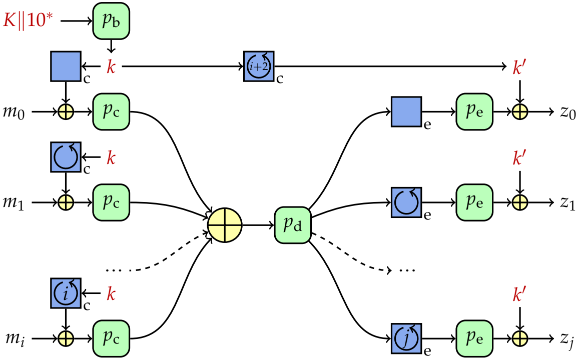
Problem: collisions with higher-order differentials if f has low degree

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



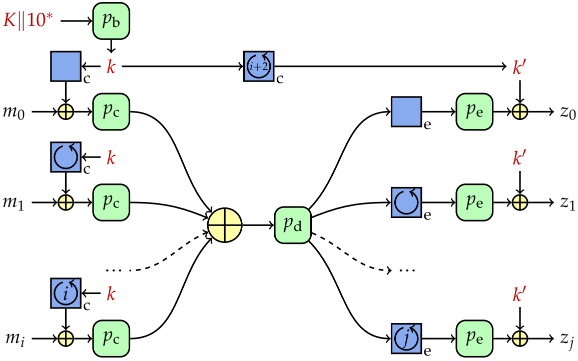
- Input mask rolling and p_c against accumulator collisions
- State rolling, p_e and output mask against state retrieval at output
- Middle p_d against higher-order DC
- Input-output attacks have to deal with $p_e \circ p_d \circ p_c$

KRAVATTE = Farfalle with KECCAK-p as in eprint 2016/1188



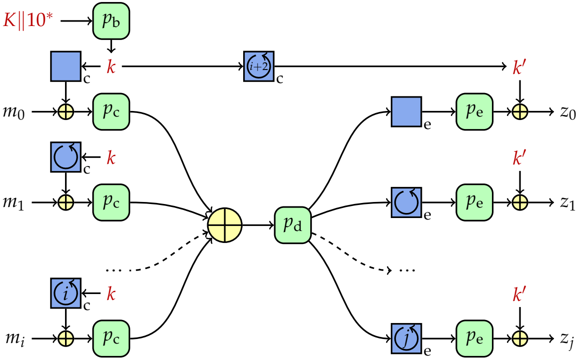
- Target security: 128 bits, incl. multi-target
- p_i = KECCAK-$p[1600]$ with # rounds in p_b, p_c, p_d, p_e being $6, 6, 4, 4$
- Rolling function as in [Granger, Jovanovic, Mennink & Neves, EC 2016], linear with order $2^{320} - 1$

KRAVATTE = Farfalle with KECCAK-p as in eprint 2016/1188



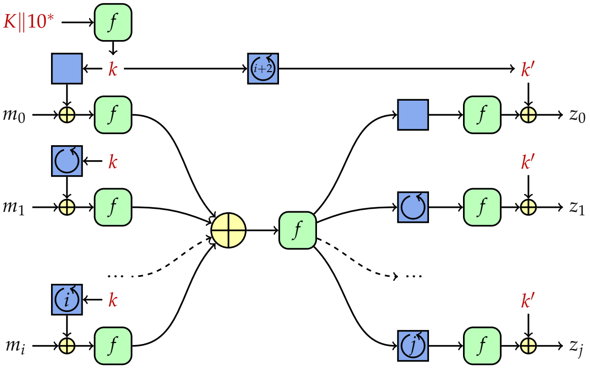
- Target security: 128 bits, incl. multi-target
- p_i = KECCAK-p[1600] with # rounds in p_b, p_c, p_d, p_e being 6, 6, 4, 4
- Rolling function as in [Granger, Jovanovic, Mennink & Neves, EC 2016], linear with order $2^{320} - 1$

KRAVATTE = Farfalle with KECCAK-p as in eprint 2016/1188



- Target security: 128 bits, incl. multi-target
- $p_i = \text{KECCAK-}p[1600]$ with # rounds in p_b, p_c, p_d, p_e being $6, 6, 4, 4$
- Rolling function as in [Granger, Jovanovic, Mennink & Neves, EC 2016], linear with order $2^{320} - 1$

KRAVATTE as in TOSC 2018

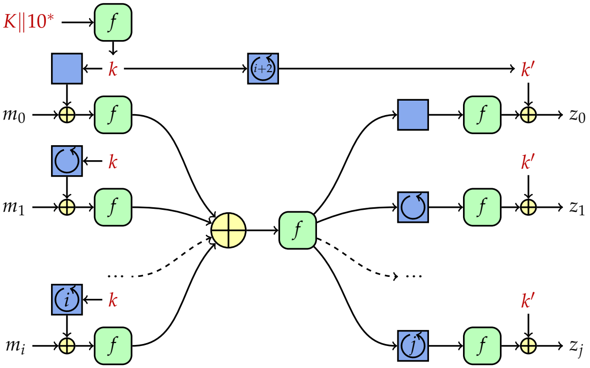


- Due to theoretical attack reversing last rounds, increase # rounds
- p_i = KECCAK-p[1600] with # rounds 6666 : *Achouffe configuration*
- Disadvantage of KRAVATTE: 200-byte granularity

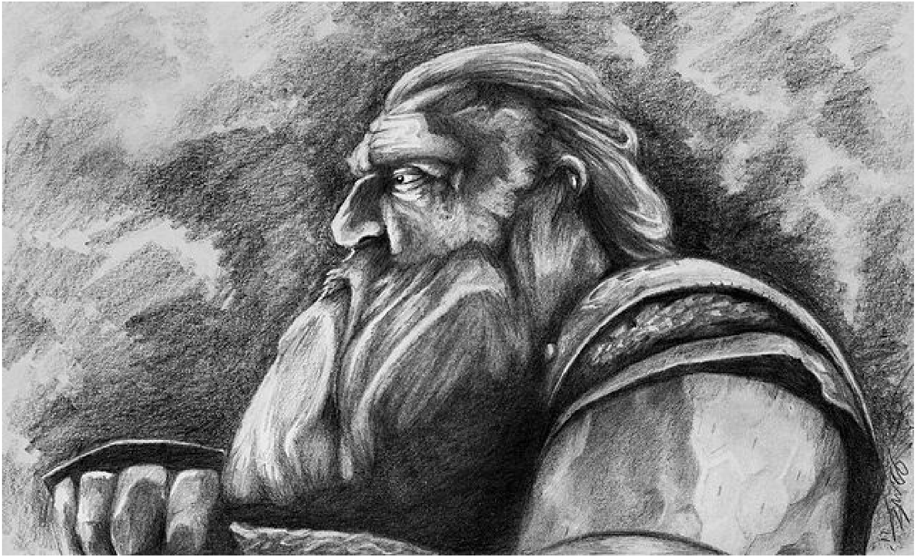
KRAVATTE as in TOSC 2018

- Due to theoretical attack reversing last rounds, increase # rounds
- p_i = KECCAK-$p[1600]$ with # rounds 6666 : *Achouffe configuration*
- Disadvantage of KRAVATTE: 200-byte granularity

KRAVATTE as in TOSC 2018

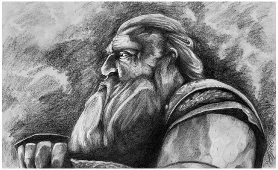


- Due to theoretical attack reversing last rounds, increase # rounds
- p_i = KECCAK-$p[1600]$ with # rounds 6666 : *Achouffe configuration*
- Disadvantage of KRAVATTE: 200-byte granularity



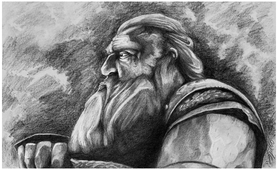
by Perrie Nicholas Smith (perriesmith.deviantart.com)

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



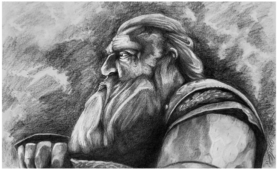
- has ideal size and shape: 48 bytes in 12 words of 32 bits
- fits in registers of ARM Cortex M3/M4 and suitable for SIMD
- For low-end platforms: locality of operations
  - minimizes swapping on AVR, M0, etc.
  - limits diffusion, see e.g. [Mike Hamburg, 2017]
  - no problem for nominal number of rounds: 24
  - not clear how many rounds needed in Farfalle

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



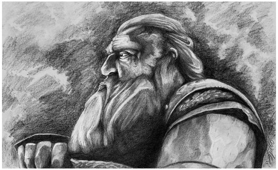
- has ideal size and shape: 48 bytes in 12 words of 32 bits
- fits in registers of ARM Cortex M3/M4 and suitable for SIMD
- For low-end platforms: locality of operations
  - minimizes swapping on AVR, M0, etc.
  - limits diffusion, see e.g. [Mike Hamburg, 2017]
  - no problem for nominal number of rounds: 24
  - not clear how many rounds needed in Farfalle

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



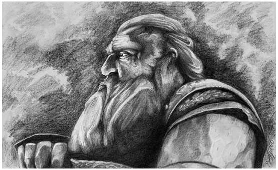
- has ideal size and shape: 48 bytes in 12 words of 32 bits
- fits in registers of ARM Cortex M3/M4 and suitable for SIMD
- For low-end platforms: locality of operations
  - minimizes swapping on AVR, M0, etc.
  - limits diffusion, see e.g. [Mike Hamburg, 2017]
  - no problem for nominal number of rounds: 24
  - not clear how many rounds needed in Farfalle

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



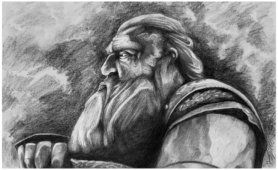
- has ideal size and shape: 48 bytes in 12 words of 32 bits
- fits in registers of ARM Cortex M3/M4 and suitable for SIMD
- For low-end platforms: locality of operations
  - minimizes swapping on AVR, M0, etc.
  - limits diffusion, see e.g. [Mike Hamburg, 2017]
  - no problem for nominal number of rounds: 24
  - not clear how many rounds needed in Farfalle

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



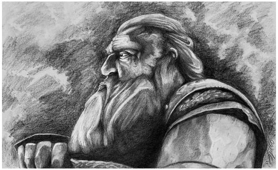
- has ideal size and shape: 48 bytes in 12 words of 32 bits
- fits in registers of ARM Cortex M3/M4 and suitable for SIMD
- For low-end platforms: locality of operations
  - minimizes swapping on AVR, M0, etc.
  - limits diffusion, see e.g. [Mike Hamburg, 2017]
  - no problem for nominal number of rounds: 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
- For low-end platforms: locality of operations
  - minimizes swapping on AVR, M0, etc.
  - 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
- For low-end platforms: locality of operations
  - minimizes swapping on AVR, M0, etc.
  - 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
  - Achouffe configuration
  - linear full-state rolling function of order $2^{384} - 1$
  - 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
  - Achouffe configuration
  - linear full-state rolling function of order $2^{384} - 1$
  - 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
  - Achouffe configuration
  - linear full-state rolling function of order $2^{384} - 1$
  - 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
  - Achouffe configuration
  - linear full-state rolling function of order $2^{384} - 1$
  - 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
  - Achouffe configuration
  - linear full-state rolling function of order $2^{384} - 1$
  - 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 $3 \times 4 \times 32$!

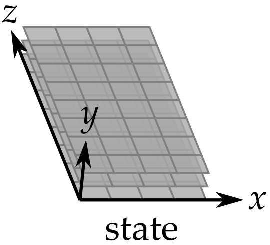
XOODOO [Keccak team with Seth Hoffert and Johan De Meulder]

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

- 384-bit permutation
- Main purpose: usage in Farfalle: XOOPRF
  - Achouffe configuration
  - linear full-state rolling function of order $2^{384} - 1$
  - Efficient on wide range of platforms
- But also for
  - small-state authenticated encryption, KETJE style
  - sponge-based hashing, …

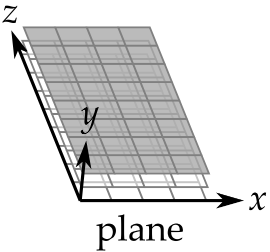
KECCAK-p philosophy ported to Gimli dimensions $3 \times 4 \times 32$*!*

Xoodoo state



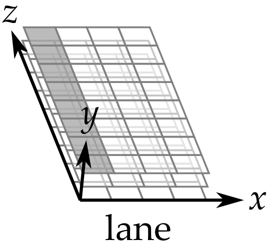
- State: 3 horizontal planes each consisting of 4 lanes

Xoodoo state



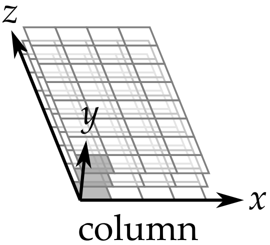
- State: 3 horizontal planes each consisting of 4 lanes

XOODOO state



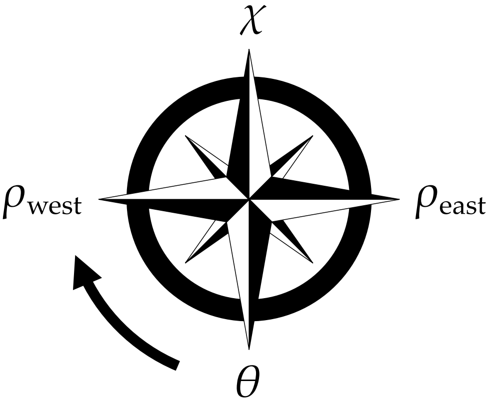
- State: 3 horizontal planes each consisting of 4 lanes

Xoodoo state



- State: 3 horizontal planes each consisting of 4 lanes

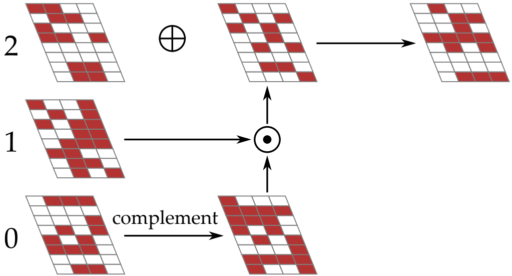
XOODOO round function



Iterated: n_r rounds that differ only by round constant

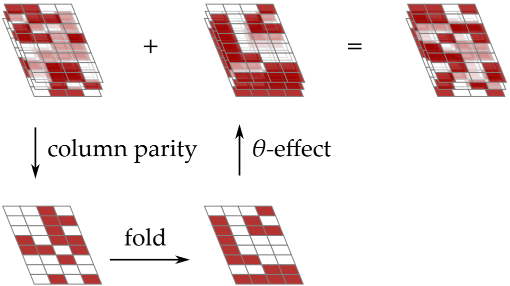
Nonlinear mapping χ

Effect on one plane:



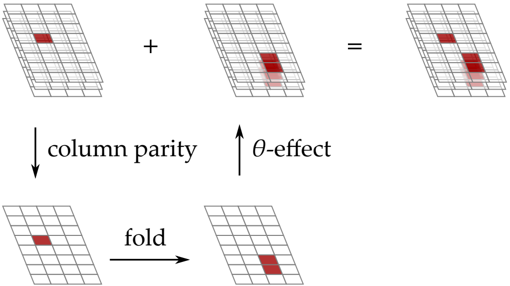
- χ as in Keccak-p, operating on 3-bit columns
- Involution and same propagation differentially and linearly

Mixing layer θ



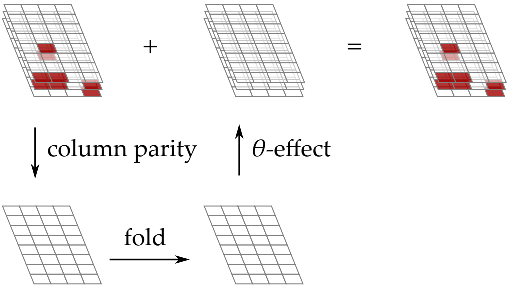
- Column parity mixer: compute parity, fold and add to state
- good average diffusion, identity for states in *kernel*

Mixing layer θ



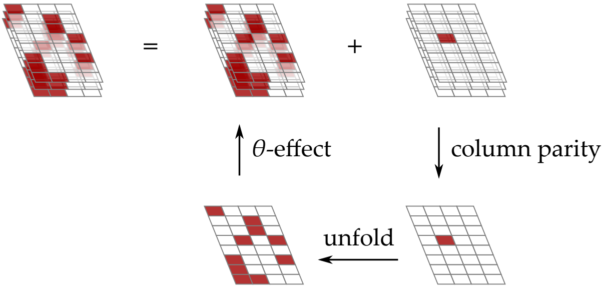
- Column parity mixer: compute parity, fold and add to state
- good average diffusion, identity for states in *kernel*

Mixing layer θ



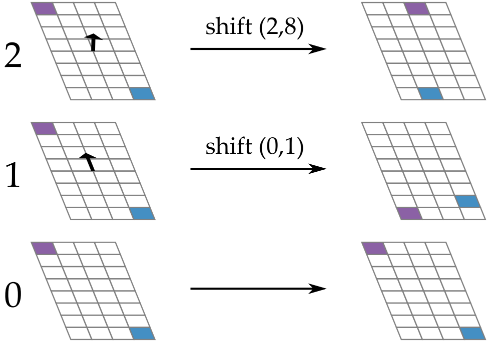
- Column parity mixer: compute parity, fold and add to state
- good average diffusion, identity for states in *kernel*

Mixing layer θ



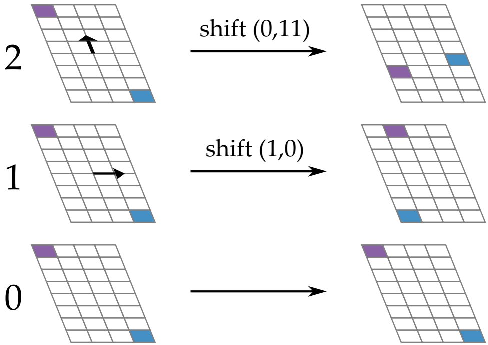
- Column parity mixer: compute parity, fold and add to state
- good average diffusion, identity for states in *kernel*

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]	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

Xoodoo diffusion and confusion

Trail bounds, using [Mella, Daemen, Van Assche, ToSC 2016]:

# rounds	min. trail weights diff.	min. trail weights linear
1	2	2
2	8	8
3	36	36
6	≥ 100	≥ 100

Strict Avalanche Criterion (SAC) [Webster, Tavares, Crypto '85]

A mapping satisfies SAC if flipping an input bit will make each output bit flip with probability close to 1/2

Xoodoo satisfies SAC

- after 3 rounds in forward direction
- after 2 rounds in backward direction

XOODOO diffusion and confusion

Trail bounds, using [Mella, Daemen, Van Assche, ToSC 2016]:

	min. trail weights	
# rounds	diff.	linear
1	2	2
2	8	8
3	36	36
6	≥ 100	≥ 100

Strict Avalanche Criterion (SAC) [Webster, Tavares, Crypto '85]

A mapping satisfies SAC if flipping an input bit will make each output bit flip with probability close to 1/2

XOODOO satisfies SAC

- after 3 rounds in forward direction
- after 2 rounds in backward direction

XOODOO diffusion and confusion

Trail bounds, using [Mella, Daemen, Van Assche, ToSC 2016]:

# rounds	min. trail weights diff.	linear
1	2	2
2	8	8
3	36	36
6	≥ 100	≥ 100

Strict Avalanche Criterion (SAC) [Webster, Tavares, Crypto '85]

A mapping satisfies SAC if flipping an input bit will make each output bit flip with probability close to 1/2

XOODOO satisfies SAC

- after 3 rounds in forward direction
- after 2 rounds in backward direction

XOODOO diffusion and confusion

Trail bounds, using [Mella, Daemen, Van Assche, ToSC 2016]:

# rounds	min. trail weights diff.	linear
1	2	2
2	8	8
3	36	36
6	≥ 100	≥ 100

Strict Avalanche Criterion (SAC) [Webster, Tavares, Crypto '85]

A mapping satisfies SAC if flipping an input bit will make each output bit flip with probability close to 1/2

XOODOO satisfies SAC

- after 3 rounds in forward direction
- after 2 rounds in backward direction

Thanks for your attention!