

# Structuring and optimizing KECCAK software

Gilles VAN ASSCHE<sup>1</sup> Ronny VAN KEER<sup>1</sup>

<sup>1</sup>STMicroelectronics

SPEED-B  
October 19-21, 2016

# Outline

- 1 Properties of KECCAK in software
- 2 Optimizing KECCAK-*f*
- 3 Structuring the KECCAK code package
- 4 KANGAROOTWELVE
- 5 AVX-512

# Outline

- 1** Properties of KECCAK in software
- 2 Optimizing KECCAK-*f*
- 3 Structuring the KECCAK code package
- 4 KANGAROOTWELVE
- 5 AVX-512

# KECCAK- $f$ in pseudo-code

```

KECCAK-F[b](A) {
  forall i in 0..nr-1
    A = Round[b](A, RC[i])
  return A
}

Round[b](A, RC) {
  θ step
  C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4], forall x in 0..4
  D[x] = C[x-1] xor rot(C[x+1],1), forall x in 0..4
  A[x,y] = A[x,y] xor D[x], forall (x,y) in (0..4,0..4)

  ρ and π steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]), forall (x,y) in (0..4,0..4)

  χ step
  A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]), forall (x,y) in (0..4,0..4)

  ι step
  A[0,0] = A[0,0] xor RC

  return A
}

```

[http://keccak.noekeon.org/specs\\_summary.html](http://keccak.noekeon.org/specs_summary.html)

# Choice of parameters

## Rate?

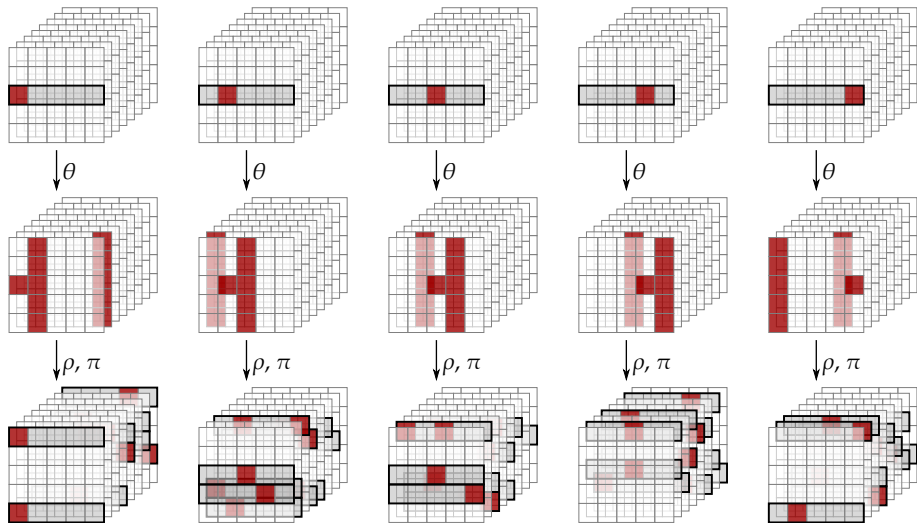
- {1152...576} bits for SHA3-224 to SHA3-512
- {1344...1088} bits for SHAKE128 and SHAKE256
  - Same for cSHAKE, KMAC, TupleHash, ... [NIST SP 800-185 draft]

## Number of rounds?

- 24 rounds for KECCAK, SHA-3, SHAKE
- 12 rounds for KEYAK, KANGAROOTWELVE

# Hardware vs software

Where does the difference come from?

Choice of operations in KECCAK-*f*

# Choice of operations in KECCAK- $f$

- Much dispersion per non-linear operation
- Benefits of **weak alignment** [Hash 2011]
- Wide dispersion: **mov reg, mem** (and back)

Unless there are more registers



# Parallelism

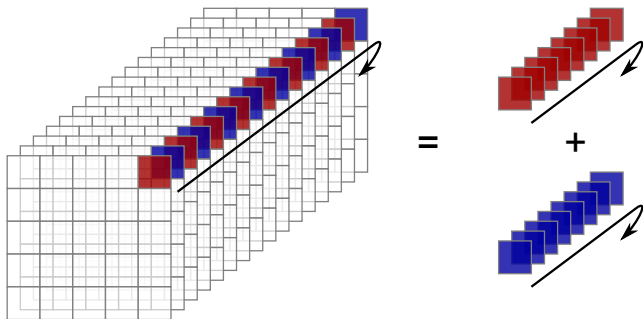
Parallelism inside KECCAK- $f$  goes by  $5 \times 64$ .

- Gain possible on SIMD units
  - E.g.,  $\times 1.7$  speed-up between x86\_64 and XOP (AMD Bulldozer)
- Possible obstacles (SSE and AVX2)
  - No rotation instructions
  - No easy fit of 5-lane structures
- **More processing per clock cycle**
  - Larger SIMD units (e.g., AVX-512)
  - More general parallelism in instruction sets (remember IA64?)
  - Parallelism from the mode

# Outline

- 1 Properties of KECCAK in software
- 2 Optimizing KECCAK-f**
- 3 Structuring the KECCAK code package
- 4 KANGAROOTWELVE
- 5 AVX-512

# Bit interleaving



$$\text{ROT}_{64} \leftrightarrow 2 \times \text{ROT}_{32}$$

# Lane complementing

Remove NOTs using the De Morgan laws:

$$z \leftarrow x \wedge (y \oplus 1)$$

$$\Leftrightarrow$$

$$\bar{z} \leftarrow \bar{x} \vee y$$

Choose an *invariant* pair of complementing masks:

$$s \oplus m \xrightarrow{\chi} \chi(s) \oplus M \xrightarrow{\pi \circ \rho \circ \theta} s' \oplus m$$

with  $m = \pi(\rho(\theta(M)))$ .

80% of NOTs removed

# In-place processing

Store  $A[x, y]$  at round  $i$  in  $(x', y')$  with

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}^i \begin{pmatrix} x \\ y \end{pmatrix}.$$

- Interacts with  $\pi$ : the output of  $\chi$  **can overwrite its input**
- Matrix of order 4
  - $\Rightarrow$  no performance loss if 4 rounds unrolled

[Bertoni et al., KECCAK implementation overview]

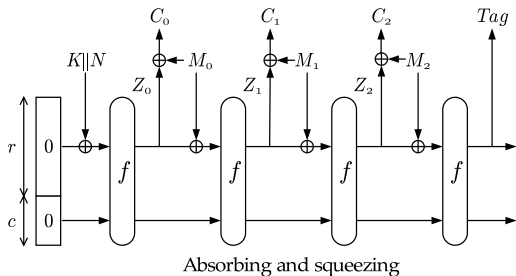
# Outline

- 1 Properties of KECCAK in software
- 2 Optimizing KECCAK-*f*
- 3 Structuring the KECCAK code package**
- 4 KANGAROOTWELVE
- 5 AVX-512

# Inventory

- KECCAK
  - `sponge`
  - KECCAK- $f[1600]$  for SHA-3, SHAKE, cSHAKE, etc.
  - KECCAK- $f[200]$  to KECCAK- $f[800]$
- KEYAK
  - `full-state keyed duplex`
  - 12 rounds
- KETJE
  - `MonkeyDuplex`
  - 1, 6, 12 rounds
- KEYAK, ParallelHash, KANGAROOTWELVE
  - `parallelism`
  - $n \times$  KECCAK- $f[1600]$

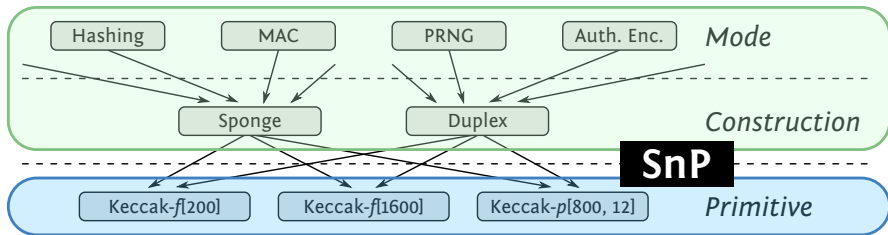
# Ingredients



- initialize the state
- apply the permutation  $f$
- XOR/overwrite bytes into the state
- extract bytes from the state
  - and optionally XOR them



# A layered approach



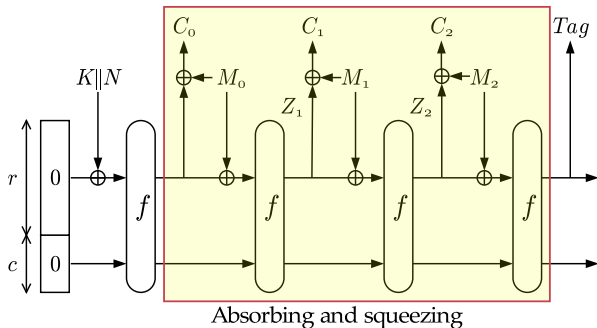
## Generic

- focus on **user**
  - easy to use
  - e.g., message queue
- one implementation
  - pointers and arithmetic

## Specific

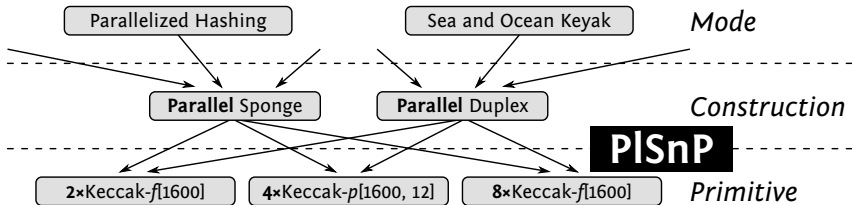
- focus on **developer**
  - limited scope to optimize
  - unit tests
- tailored implementations
  - permutation
  - bulk data processing

# SnP fast loop optimization

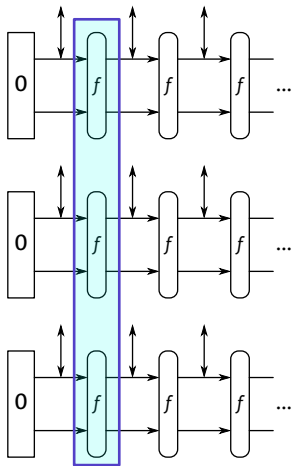


Specialized repeated application of some operations  
(optional)

# Parallel processing



# PlSnP (= Parallel States and Permutations)



- Functions on individual instances
- Functions on **all instances**
  - Parallel application of  $f$
  - XOR blocks into state
- Optional **fast loop**
- Fallback stubs
  - to serial implementation
  - to parallel with lower degree

# The KECCAK code package

GitHub This repository Search Explore Features Enterprise Blog Sign up Sign in

gvanas / KeccakCodePackage ★ Star 45 🍴 Fork 13

Keccak Code Package

90 commits 1 branch 0 releases 6 contributors

branch: master KeccakCodePackage / +

Added AVX2 implementation of Keccak-[1600] by Vladimir Sedach

The Keccak, Keyak and Ketje Teams authored 29 days ago latest commit 6a56d00c0c

Build	Improved permutation and state interface, extended duplex functionali...	7 months ago
CAESAR	Added Keyak reference implementation	6 months ago
Common	Initial version of the Keccak Code Package	2 years ago
Constructions	removed trailing whitespaces at the end of all source files using	4 months ago
Ketje	removed trailing whitespaces at the end of all source files using	4 months ago
Modes	Removed useless includes in Keyak.c	2 months ago
PISnP	removed trailing whitespaces at the end of all source files using	4 months ago
SnP	Added AVX2 implementation of Keccak-[1600] by Vladimir Sedach	29 days ago

Code Issues 0 Pull Requests 0 Pulse Graphs

HTTPS clone URL  
<https://github.com/gvanas/KeccakCodePackage>

You can clone with HTTPS or Subversion.

Clone in Desktop Download ZIP

<https://github.com/gvanas/KeccakCodePackage>

# Outline

- 1 Properties of KECCAK in software
- 2 Optimizing KECCAK- $f$
- 3 Structuring the KECCAK code package
- 4 KANGAROOTWELVE**
- 5 AVX-512

# Safety margin: from *rock-solid* to *comfortable*

- KECCAK's round function unchanged since 2008
- How many rounds?
  - **24** rounds: KECCAK/SHA-3/SHAKE
  - **12** rounds: KEYAK
  - **5** rounds: best collision attacks [Dinur et al.] [Qiao et al.]

⇒ let's do **hashing with 12 rounds**

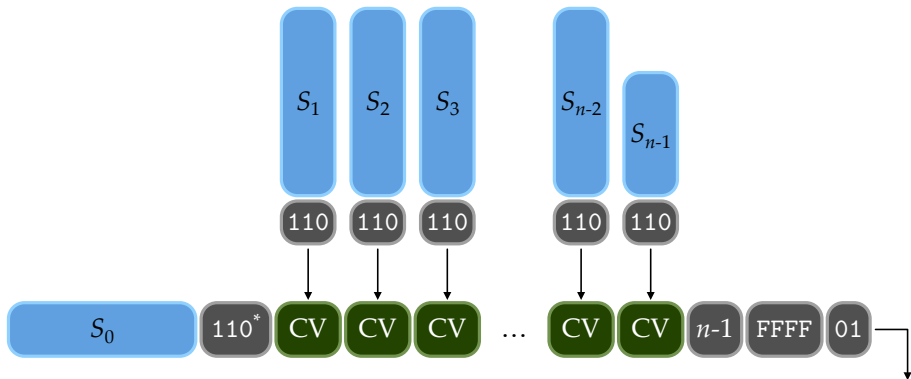
# Better exploit parallelism

- SIMD with growing widths
  - 128, 256 and soon 512 bits
- Multiple cores

⇒ let's **exploit this parallelism**



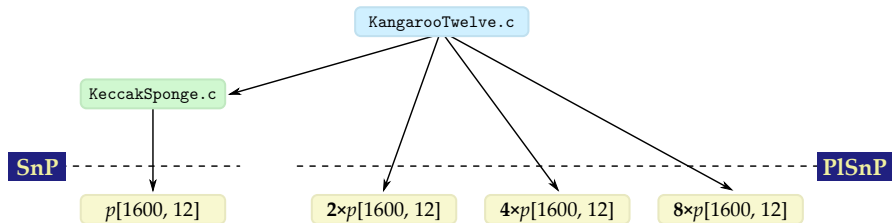
## KANGAROOTWELVE's mode



Final node growing with kangaroo hopping and SAKURA coding

[ACNS 2014]

# Structure in the KECCAK code package



# KANGAROOTWELVE performance

	Short input	Long input
Intel® Core™ i5-4570 (Haswell)	4.15 c/b	1.44 c/b
Intel® Core™ i5-6500 (Skylake)	3.72 c/b	1.22 c/b
Intel® Xeon Phi™ 7250 (Knights Landing)*	(4.56 c/b)	0.74 c/b

\* Thanks to Romain Dolbeau



[Bertoni et al., IACR ePrint 2016/770]

# Outline

- 1 Properties of KECCAK in software
- 2 Optimizing KECCAK- $f$
- 3 Structuring the KECCAK code package
- 4 KANGAROOTWELVE
- 5 AVX-512**

# AVX-512

AVX-512 = SIMD with 512-bit registers

Benefits for KECCAK- $f$ :

- $8 \times \text{KECCAK-}f[1600]$
- Rotation instructions
- **Three-input binary functions**
  - $r \leftarrow a \oplus b \oplus c$
  - $r \leftarrow a \oplus ((b \oplus 1) \wedge c)$
- 32 registers



A near collision...

# Questions?



By Marcello Maria Perongini (flickr.com)

<https://github.com/gvanas/KeccakCodePackage>

<http://keccak.noekeon.org/Keccak-implementation-3.2.pdf>

<http://keccak.noekeon.org/KangarooTwelve.pdf>