The Keccak Code Package

Guido Bertoni[1] Joan Daemen[1]
Michaël Peeters[2] Gilles Van Assche[1] Ronny Van Keer[1]

[1]STMicroelectronics

[2]NXP Semiconductors

SHA-3 2014 Workshop
UC Santa Barbara, August 22, 2014

Outline

Outline

Extending the scope of software implementations?

In KeccakReferenceAndOptimized.zip, there are

- implementations for hashing only
- implementations of KECCAK-$f[1600]$ only

So what about extending this set to

- other applications
- parallelized modes
- KETJE and KEYAK
- KECCAK-$f[800/400/200]$, KECCAK-$p[1600, n_r = 12]$, etc.
    - … and other permutations … ?

Extending the scope of software implementations?

In KeccakReferenceAndOptimized.zip, there are

- implementations for hashing only
- implementations of KECCAK-$f[1600]$ only

So what about extending this set to

- other applications
- parallelized modes
- KETJE and KEYAK
- KECCAK-$f[800/400/200]$, KECCAK-$p[1600, n_r = 12]$, etc.
    - … and other permutations … ?

A heterogenous set of software implementations

In KeccakReferenceAndOptimized.zip, there are

- implementations for various architectures
- with **different** structures
- with hard-coded or flexible capacity
- with or without an input queue

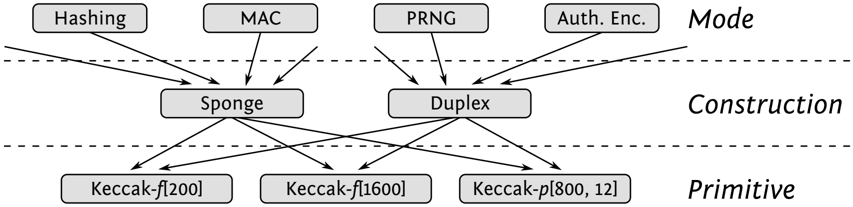
avr8, avr8asm-compact, avr8asm-fast, compact, compact8, inplace, inplace32BI-armgcc-ARMv6M/v7A/v7M, opt32, opt64, reference, reference32BI, xop, simple, simple32BI, simd64, simd128, x86-64, x86-64-shld, Keccakc512-crypto_hash-inplace-armgcc-ARMv7A-NEON.s, ...

A heterogenous set of software implementations

Picture by Magalie L'Abbé (flickr.com)

Outline

Goals of a layered approach



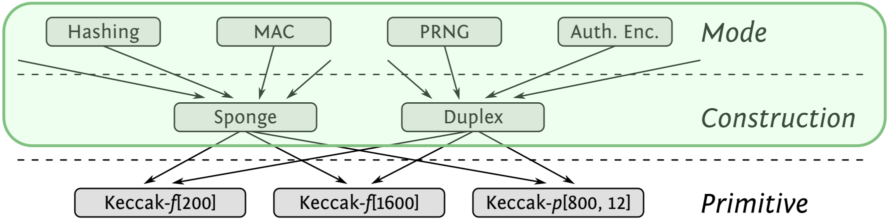
Generic

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

Specific

- focus on **developer**
  - limited scope to optimize
  - bugs caught early
- tailored implementations
  - permutation
  - bulk data processing

Goals of a layered approach



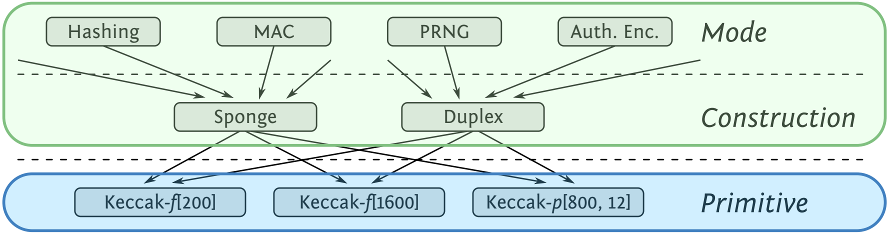
Generic

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

Specific

- focus on **developer**
  - limited scope to optimize
  - bugs caught early
- tailored implementations
  - permutation
  - bulk data processing

Goals of a layered approach



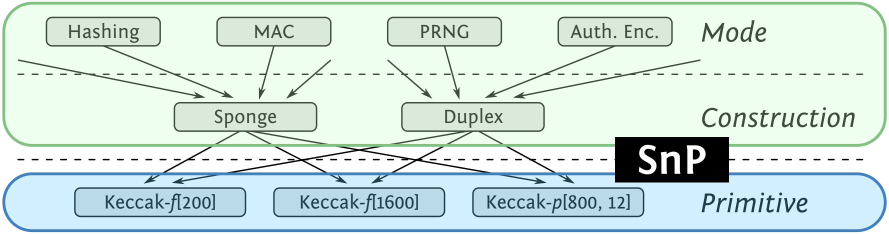
Generic

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

Specific

- focus on **developer**
  - limited scope to optimize
  - bugs caught early
- tailored implementations
  - permutation
  - bulk data processing

Goals of a layered approach



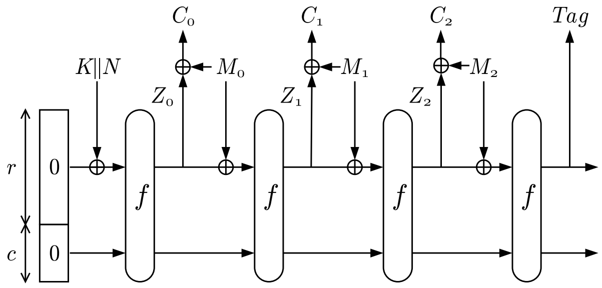
Generic

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

Specific

- focus on **developer**
  - limited scope to optimize
  - bugs caught early
- tailored implementations
  - permutation
  - bulk data processing

SnP (= State and Permutation)

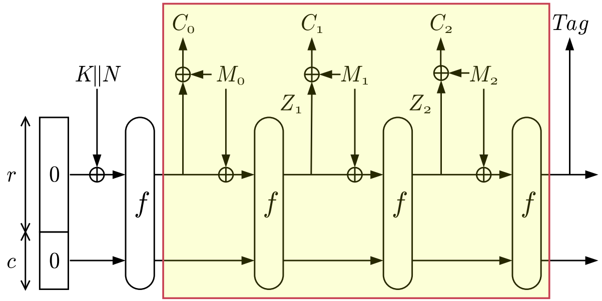


Absorbing and squeezing

- initialize the state to zero
- apply the permutation f

- XOR/overwrite bytes into the state
- extract bytes from the state
  - and optionally XOR them

SnP FBWL (= Full Blocks Whole Lane)

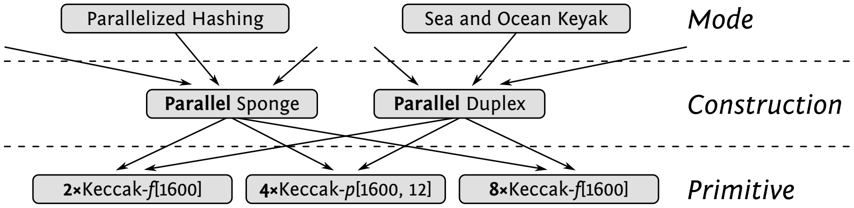


Absorbing and squeezing

Specialized repeated application of some operations
(optional)

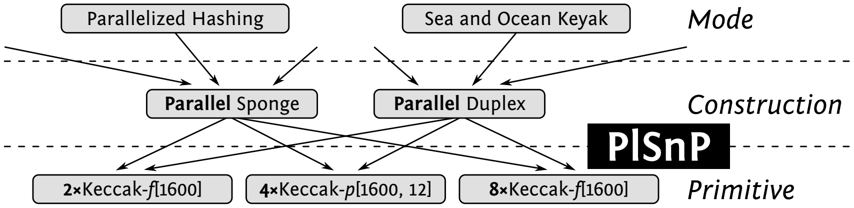
`SnP_FBWL_Absorb/Squeeze/Wrap/Unwrap`

Parallel processing



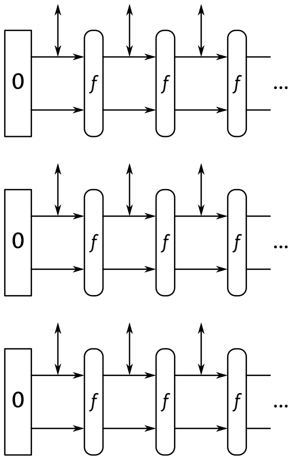
- Some modes exploit parallelism
- To exploit this, we need:
  - sponge functions and duplex objects running in parallel
  - permutation applied on several states in parallel

Parallel processing



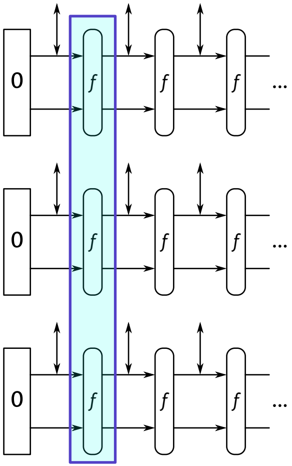
- Some modes exploit parallelism
- To exploit this, we need:
  - sponge functions and duplex objects running in parallel
  - permutation applied on several states in parallel

PlSnP (= Parallel States and Permutations)



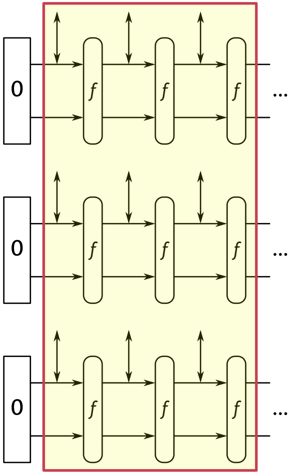
- SnP on individual instances
- Some SnP functions parallelized
    - Parallel application of f
- **PlSnP FBWL** for repeated operations

PlSnP (= Parallel States and Permutations)



- SnP on individual instances
- Some SnP functions parallelized
  - Parallel application of f
- **PlSnP FBWL** for repeated operations

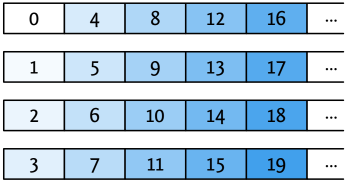
PlSnP (= Parallel States and Permutations)



- SnP on individual instances
- Some SnP functions parallelized
    - Parallel application of f
- **PlSnP FBWL** for repeated operations

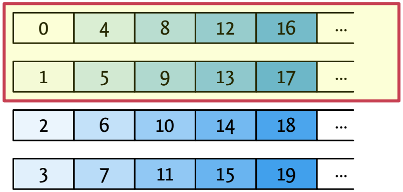
PlSnP FBWL: parameterized block layout

Interleaving (blocks of *r* bits) in 4 lines



PlSnP FBWL: parameterized block layout

Interleaving (blocks of *r* bits) in 4 lines



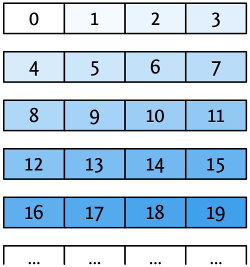
Assuming 2-way parallelism:

$\rightarrow$ 4 blocks

$\downarrow$ 1 block

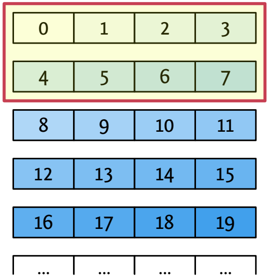
PlSnP FBWL: parameterized block layout

Segmenting in 4 blocks of *r* bits each



PlSnP FBWL: parameterized block layout

Segmenting in 4 blocks of *r* bits each



Assuming 2-way parallelism:

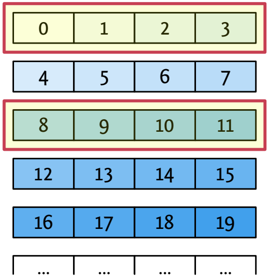
$\rightarrow$ 1 block

$\downarrow$ 4 blocks

(2 consecutive lines)

PlSnP FBWL: parameterized block layout

Segmenting in 4 blocks of r bits each



Assuming 2-way parallelism:

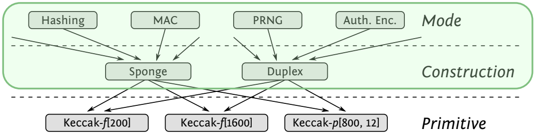
$\rightarrow$ 1 block

$\downarrow$ 8 blocks

(even/odd lines)

Outline

Constructions and modes



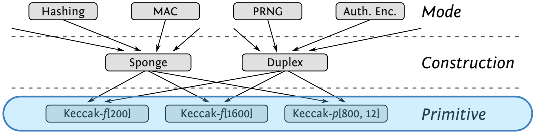
Currently in the KCP

- SHA-3 hashing and XOFs
- River and Lake Keyak
- Ketje (*)
- Anything using sponge or duplex directly

Nice to have

- Pseudo-random bit sequence generator
- Overwrite sponge

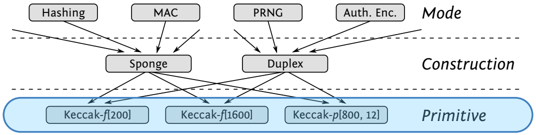
Primitives



KECCAK-f[200 to 1600], KECCAK-p[200 to 1600, n_r]

- Reference implementations
- Optimized impl. in C of KECCAK-f[1600] and -p[1600, $n_r = 12$]
  - using 64-bit words or 32-bit words (bit interleaving)
  - compact, in place, unrolled, lane complemented, etc.
- Assembly optimized for
  - x86_64 (KECCAK-f[1600] and KECCAK-p[1600, $n_r = 12$] only)
  - ARMv6M, ARMv7M, ARMv7A, NEON
  - AVR8

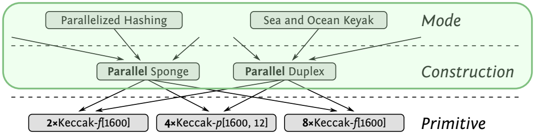
Primitives



On the to-do list

- Some implementations still to be migrated from KeccakReferenceAndOptimized.zip
- Optimized in C for 800-bit width and smaller
- ARMv8, (your favorite platform here)

Parallel constructions and modes



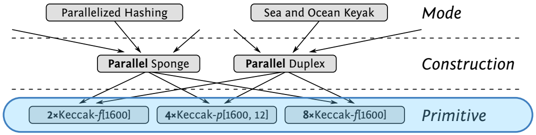
Currently in the KCP

- Sea and Ocean Keyak
- Anything using parallel duplex objects directly

On the to-do list

- Parallel sponge functions
- Parallelized hashing

Parallelized primitives



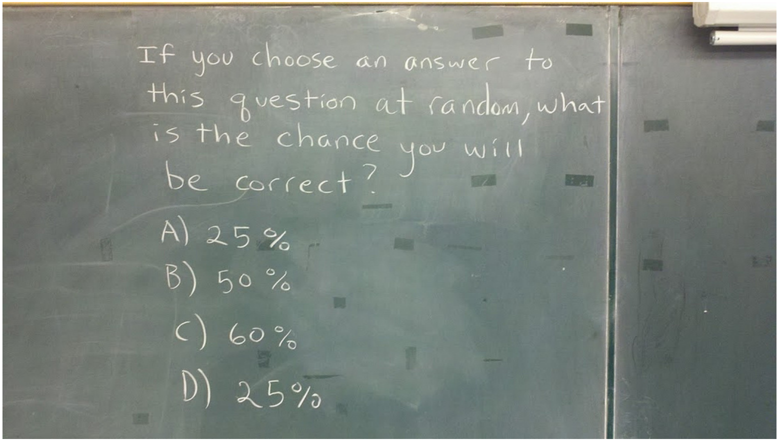
Currently in the KCP

- Serial fallback to SnP
- $2 \times$ KECCAK-$f[1600]/p[1600, n_r = 12]$ on ARMv7M+NEON

Many things on the to-do list

- $2 \times$ KECCAK-$f[1600]/p[1600, n_r = 12]$ using SSE, XOP or AVX (...WIP...)
- $4 \times$ KECCAK-$f[1600]/p[1600, n_r = 12]$ using AVX2 or AVX512
- $8 \times$ KECCAK-$f[1600]/p[1600, n_r = 12]$ using AVX512
- ARMv8 NEON, (your favorite SIMD instruction set here)

Questions?



If you choose an answer to this question at random, what is the chance you will be correct?

A) 25%

B) 50%

C) 60%

D) 25%

Picture by Duncan Hull (dullhunk on flickr.com)

https://github.com/gvanas/KeccakCodePackage