Efficient Implementation of Permutation-Based Hash Functions for the RISC-V Architecture

Issam Jomaa, Hao Cheng, *Johann Großschädl*, Peter Ryan

SnT and DCS, University of Luxembourg

The research described in this presentation was supported by the Fonds National de la Recherche (FNR) Luxembourg under CORE grant C19/IS/13641232

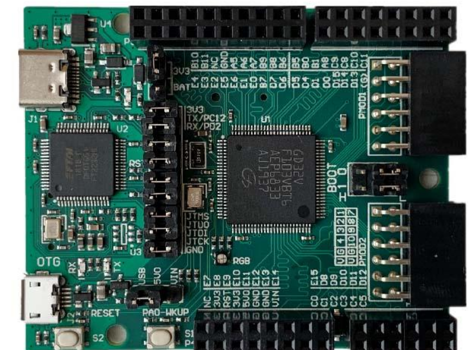
Outline

- RISC-V architecture
- RV32 implementation of Ascon-Hash, Esch256, Xoodoo
- Performance evaluation of permutations
- Auxiliary operations of Ascon-Hash
  - Loading of blocks from RAM
  - Endian conversion
  - Bit-interleaving
- Impact of auxiliary operations on full hash function

RISC-V Architecture

- Free and open instruction set architecture
  - Based on well-established RISC design principle
  - 32 registers (some have special purpose)
  - Minimalist set of 40 core instructions (no rotations)
  - Optional extensions for crypto, bit-manipulation, …
  - BitManip extension: rotations, andn, orn

- Evaluation platform: Nuclei RV-STAR
  - Arduino compatible board
  - GD32VF103VBT6 (RV32IMAC) @ 108 MHz
  - 128 kB flash, 32 kB SRAM (no caches!)



Existing RISC-V Implementations

- Designer teams
  - Ascon: Assembler implementation of full AEAD algorithm (i.e., permutation and mode), with and without BitManip
  - Esch: Assembler implementation of SPARKLE384 permutation (with and without BitManip), rest in C
  - Xoodyak: No Assembler code RISC-V in XKCP
- Other implementers
  - Campos et al (CANS 2020): Assembler implementation of permutations of Ascon, Schwaemm/Esch, and Xoodyak
  - Stoffelen (LATINCRYPT 2019): Assembler code for Keccak f1600

Performance of Permutations

Permutation	Rounds/steps	Time (cycles)	Hash-rate	Cycles/rate-byte
Ascon p8	8	1016 cc	8 bytes	127.00 cc/rb
Ascon p12	12	**1492 cc**	8 bytes	**186.50 cc/rb**
SPARKLE384 slim	7	**1661 cc**	16 bytes	**103.81 cc/rb**
SPARKLE384 big	11	2560 cc	16 bytes	160.00 cc/rb
Xoodoo	12	**1521 cc**	16 bytes	**96.06 cc/rb**
Keccak f1600	25	14014 cc	136 bytes	**103.04 cc/rb**

Ascon, Sparkle, and Xoodoo are speed-optimized (fully unrolled loops)
SPARKLE384 with big steps is only used for initialization and start of squeezing
Keccak f1600 is fast (for large messages!) but not lightweight

Impact of BitManip Extension

- Rotation of 32-bit words, AND/OR with one input inverted
- Ascon p12
  - Saved cycles/round: 40 (bit-interleaving halves lin-layer cycles)
  - Overall permutation: 1492 → **1012** cycles
- SPARKLE384 slim
  - Saved cycles/round: 88 (14 cycles in each of the 6 ARX-boxes)
  - Overall permutation: 1661 → **1045** cycles
- Xoodoo
  - Saved cycles/round: ≈36 (mostly theta and rho)
  - Overall permutation: 1521 → ≈**1089** cycles

Beyond the Permutation

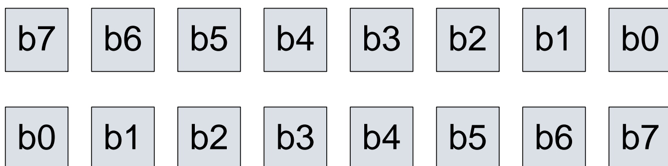
- Auxiliary operations
  - Loading of message blocks (8/16 bytes) from RAM to registers
  - Esch: block injected into state via simple linear Feistel function
  - Ascon: Endian conversion, bit-interleaving (BitManip rotations)
  - Implemented in portable C
- Loading of message blocks
  - High-level API expects message in array of bytes (uint8_t)
  - Permutations operate on 32/64-bit words (uint32_t, uint64_t)
  - Tempting to cast from uint8_t-pointer to uint32/64_t pointer
  - E.g., XKCP-plain-ua: uint32_t *data = (uint32_t *)argdata;

Loading of Message Blocks

- Alignment issues with pointer casts
  - Data in memory is normally aligned at natural boundaries
  - "Upcasting" of pointer can interfere with natural alignment
  - Result of pointer-cast from uint8_t to uint32_t can be undefined
  - Misalignment usually no problem for e.g. x64 and armv7m
  - Misaligned load/store causes bus error on armv6m (Cortex-M0)
  - RISC-V: misaligned accesses are supported, but can be very slow
- How to do it in "clean" C
  - Check alignment at run-time (use memcpy() to copy block into aligned buffer if byte-pointer is not sufficiently aligned)
  - Load/store block byte-by-byte (better for Ascon)

Endian Conversion

- Ascon expects 8-byte blocks in big endian
  - Almost all architectures are little Endian (conversion needed)



- Big Endian: Why, oh why? (was asked on Reddit)
  - Byte-order not mentioned at all in Ascon specification!

jedisct1 · 4 mo. ago · *edited 4 mo. ago*

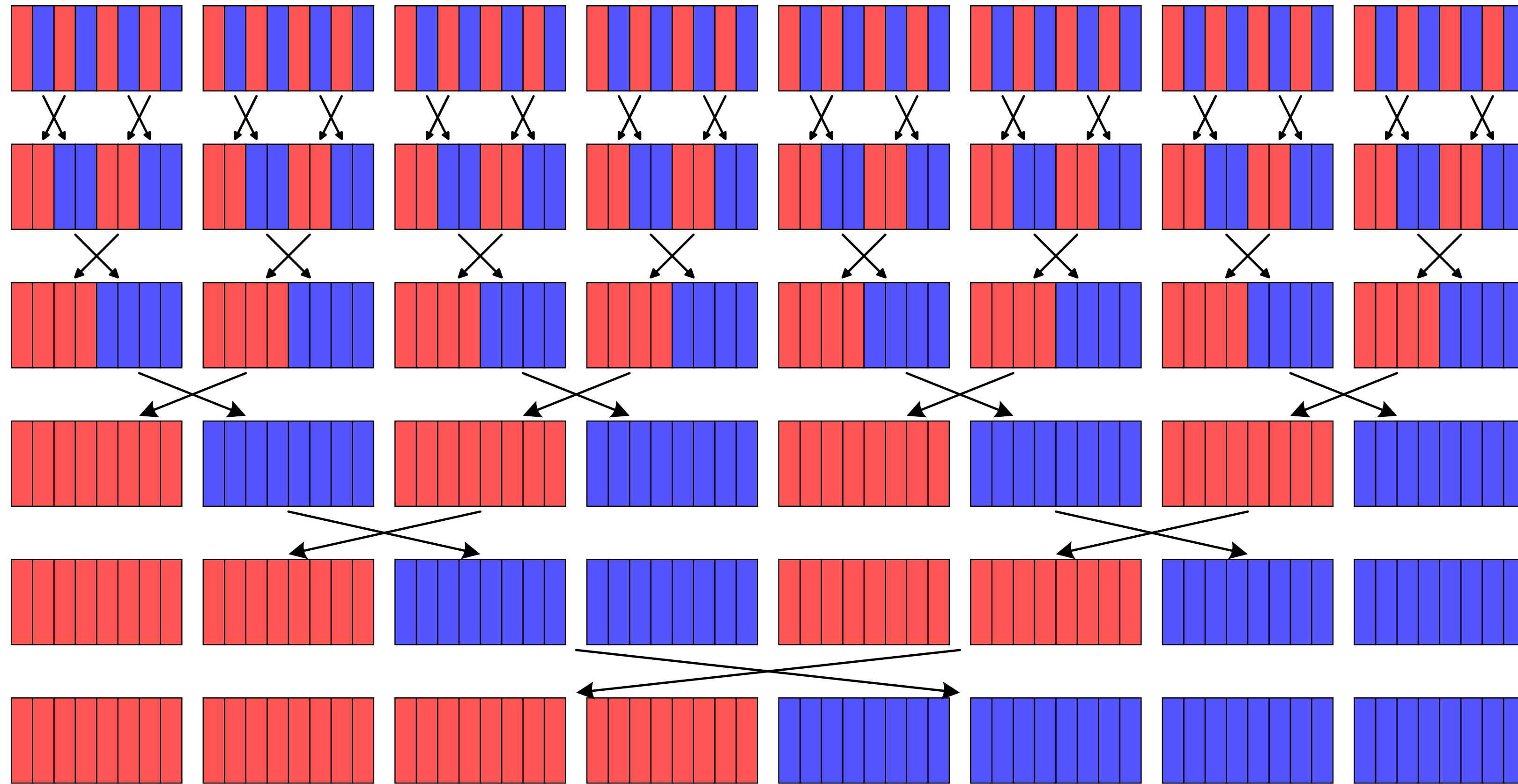
I'm very disappointed as well.
From a usability perspective, Ascon doesn't do anything new. Unlike Xoodyak's Cyclist mode, that makes it very easy to secure entire sessions.
Also, ASCON uses big endian. BIG ENDIAN. Why, oh why?

7 Reply Share ...

Conversion from/to Bit-Interleaved Form

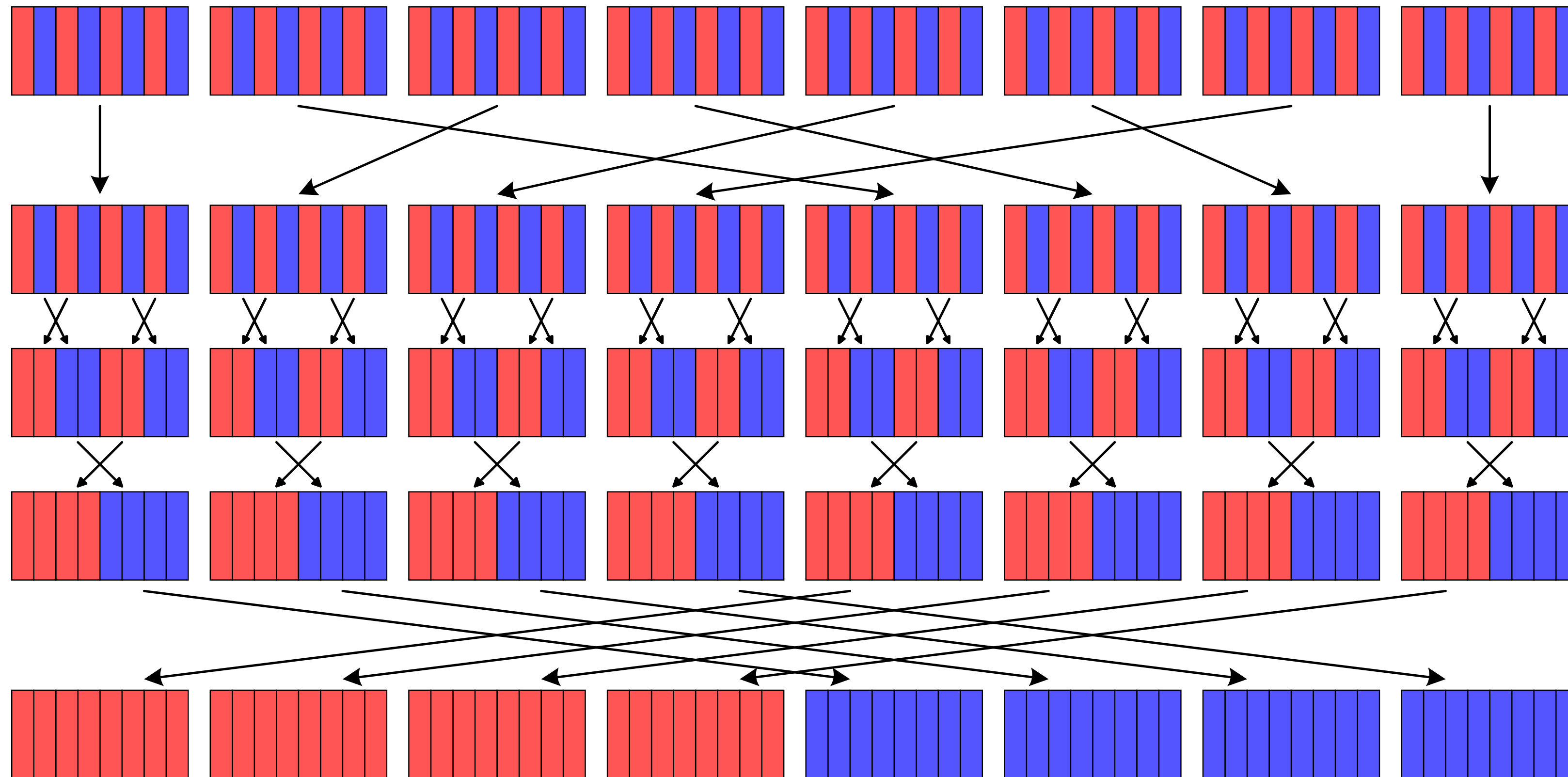
- Bit-Interleaving: Why and how
  - Ascon's linear layer performs rotations on 64-bit words
  - BitManip's rotate instructions operate on 32-bit registers
  - Bit-interleaving: place bits with even index and odd index in two 32-bit registers and rotate these registers
- DeltaSwap and SwapMove
  - DeltaSwap(x,m,d): t = (x ^ (x >> d)) & m; x ^= t ^ (t << d);
  - SwapMove(x,y,m,d): t = (x ^ (y >> d)) & m; x ^= t; y ^= (t << d);
  - Swap (blocks of) bits in one register or across two registers
  - Both take 6 cycles on RV32I (when mask m is already in register)

BitInt Conversion: First Variant



9 DeltaSwaps/SwapMoves (last one takes fewer cycles)

BitInt Conversion: Second Variant



Pre-arrangement of bytes at beginning instead of swaps at end

5 DeltaSwaps/SwapMoves

Putting It All Together

- Byte-wise operations
  - Loading of bytes from RAM and placing them in 2 registers
  - Endian conversion (i.e., moving bytes)
  - Pre-arranging bytes for BitInt conversion (i.e., moving bytes)
  - Combine all the above by loading bytes and placing them at the right position in registers
- Execution-time
  - Loading 8 bytes and placing at correct position in regs: 28 cycles
  - 5 DeltaSwaps/SwapMoves: 30 cycles (excl. loading of masks)
  - ≈90 cycles altogether (function-call overhead, compiler issues)

(Preliminary) Hash Results

- Auxiliary operations impact processing time of a block
  - Processing a message block also requires auxiliary operations and not just the permutation
  - All three permutations take ≈1000 cycles when using BitManip
  - Ascon-Hash: 9% of permutation-time
  - Esch256: 10% of permutation-time
  - Xoodyak: 4% of permutation-time
  - Ascon-AEAD: 32% of p6 (conv to & from BitInt, p6 only 560 cycles)
- Paper and source code
  - Will appear on Eprint and GitHub around mid July