A Fast and Compact RISC-V Accelerator for Ascon and Friends

Stefan Steinegger and Robert Primas

CARDIS 2020, 19th Smart Card Research and Advanced Application Conference

Motivation

- Reduce area overhead compared to dedicated co-processors

Motivation

- Reduce area overhead compared to dedicated co-processors
- Select versatile building block
  - Usable for authenticated encryption, hashing, PRNG,...

Motivation

- Reduce area overhead compared to dedicated co-processors
- Select versatile building block
  - Usable for authenticated encryption, hashing, PRNG,...
- Versatile building block Ascon-p
  - Used by Ascon and Isap
- Accelerator with Isap mode
  - AEAD with hardening against implementation attacks
  - Desired feature in NIST LWC competition
  - Covers both power analysis and fault attacks
  - No need for protected hardware building blocks

Background

- Sponge-based AEAD scheme

Background - Ascon

- Sponge-based AEAD scheme
- First choice in CAESAR's lightweight category

Background - Ascon

- Sponge-based AEAD scheme
- First choice in CAESAR's lightweight category
- Also competes in NIST LWC competition

Background - Ascon

- Sponge-based AEAD scheme
- First choice in CAESAR's lightweight category
- Also competes in NIST LWC competition
- 320-bit state in 5×64 bit lanes

Background - Ascon

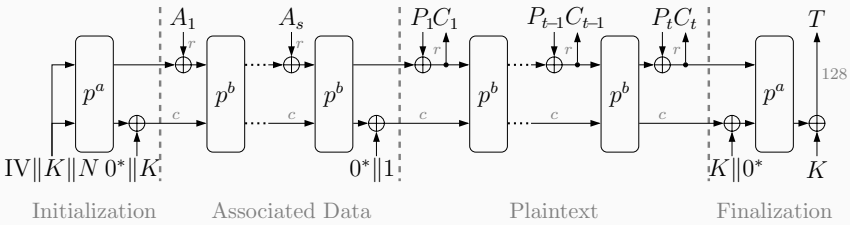
- Sponge-based AEAD scheme
- First choice in CAESAR's lightweight category
- Also competes in NIST LWC competition
- 320-bit state in 5×64 bit lanes
- Rate of 64 or 128 bits

- Sponge-based AEAD scheme
- First choice in CAESAR's lightweight category
- Also competes in NIST LWC competition
- 320-bit state in 5×64 bit lanes
- Rate of 64 or 128 bits
- Permutation function $\textsc{Ascon}$-p

Background - Ascon

- Sponge-based AEAD scheme
- First choice in CAESAR's lightweight category
- Also competes in NIST LWC competition
- 320-bit state in 5×64 bit lanes
- Rate of 64 or 128 bits
- Permutation function $\textsc{Ascon}$-p
- Authenticated encryption, hashing, PRNG

- Ascon-p is called in $p^a = 12$ and $p^b = 6$ or 8 times
- Hashing works in a similar manner



Background - Isap

- Mode for authenticated encryption

Background - Isap

- Mode for authenticated encryption
- Mode-level hardening against various physical attacks
  - DPA, DFA, SFA, SIFA

- Mode for authenticated encryption
- Mode-level hardening against various physical attacks
  - DPA, DFA, SFA, SIFA
- Two-pass scheme

Background - Isap

- Mode for authenticated encryption
- Mode-level hardening against various physical attacks
  - DPA, DFA, SFA, SIFA
- Two-pass scheme
- Currently in the 2nd round of the NIST LWC competition

- Mode for authenticated encryption
- Mode-level hardening against various physical attacks
  - DPA, DFA, SFA, SIFA
- Two-pass scheme
- Currently in the 2nd round of the NIST LWC competition
- 2/4 parametrizations based on Ascon-p

- Mode for authenticated encryption
- Mode-level hardening against various physical attacks
  - DPA, DFA, SFA, SIFA
- Two-pass scheme
- Currently in the 2nd round of the NIST LWC competition
- 2/4 parametrizations based on Ascon-p

- Transforms state in 3 steps:
  - Round constant addition

- Transforms state in 3 steps:
  - Round constant addition
  - Substitution layer

- Transforms state in 3 steps:
  - Round constant addition
  - Substitution layer
  - Linear layer

- Transforms state in 3 steps:
  - Round constant addition
  - Substitution layer
  - Linear layer
- Versatile and flexible:
  - Isap-A-128a, Isap-A-128
  - Ascon, Ascon-Hash, Ascon-Xof

Building an Accelerator

Building an Accelerator

- Typically co-processor designs

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)
  - Great for lots of processing on very little data

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)
  - Great for lots of processing on very little data
  - Less flexible

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)
  - Great for lots of processing on very little data
  - Less flexible
- Our approach:

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)
  - Great for lots of processing on very little data
  - Less flexible
- Our approach:
  - Reuse parts of register file, implement only comb. logic of Ascon-p

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)
  - Great for lots of processing on very little data
  - Less flexible
- Our approach:
  - Reuse parts of register file, implement only comb. logic of Ascon-p
  - 320-bits $\rightarrow$ ten 32 bit or five 64 bit registers
  - ARMv7 and RISC-V RV32E: 16 registers

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)
  - Great for lots of processing on very little data
  - Less flexible
- Our approach:
  - Reuse parts of register file, implement only comb. logic of Ascon-p
  - 320-bits $\rightarrow$ ten 32 bit or five 64 bit registers
  - ARMv7 and RISC-V RV32E: 16 registers
  - State fits in CPU registers (less area)

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)
  - Great for lots of processing on very little data
  - Less flexible
- Our approach:
  - Reuse parts of register file, implement only comb. logic of Ascon-p
  - 320-bits $\rightarrow$ ten 32 bit or five 64 bit registers
  - ARMv7 and RISC-V RV32E: 16 registers
  - State fits in CPU registers (less area)
  - Integrate tightly as new custom instruction

Building an Accelerator

- Typically co-processor designs
  - Loosely coupled and connected over a bus (e.g. AXI)
  - Internal sets of registers (more area)
  - Great for lots of processing on very little data
  - Less flexible
- Our approach:
  - Reuse parts of register file, implement only comb. logic of Ascon-p
  - 320-bits $\rightarrow$ ten 32 bit or five 64 bit registers
  - ARMv7 and RISC-V RV32E: 16 registers
  - State fits in CPU registers (less area)
  - Integrate tightly as new custom instruction
  - Mode remains entirely in software
  - Basic building block for Ascon and Isap

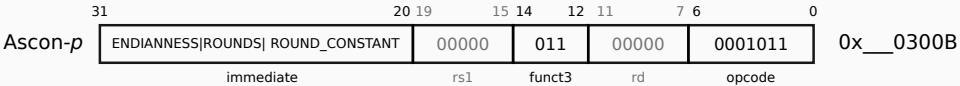
- Add Ascon-*p* on RI5CY/CV32E40P core
- 32-bit RISC-V CPU implementing RV32IM[F]C ISA
  - 32 general purpose registers
  - Additional instructions for DSP
  - 4-stage pipeline

- Add Ascon-*p* on RI5CY/CV32E40P core
- 32-bit RISC-V CPU implementing RV32IM[F]C ISA
  - 32 general purpose registers
  - Additional instructions for DSP
  - 4-stage pipeline
- Necessary changes:
  - Instruction encoding
  - Register file adaptations
  - Decode stage adaptations

Instruction Encoding

- I-type instruction

- I-type instruction
- 12-bit Immediate encodes
  - Round Constant (8 bit)
  - Number of rounds (3 bit)
  - Endianess (1 bit)



- ASCON-p requires ten 32-bit registers

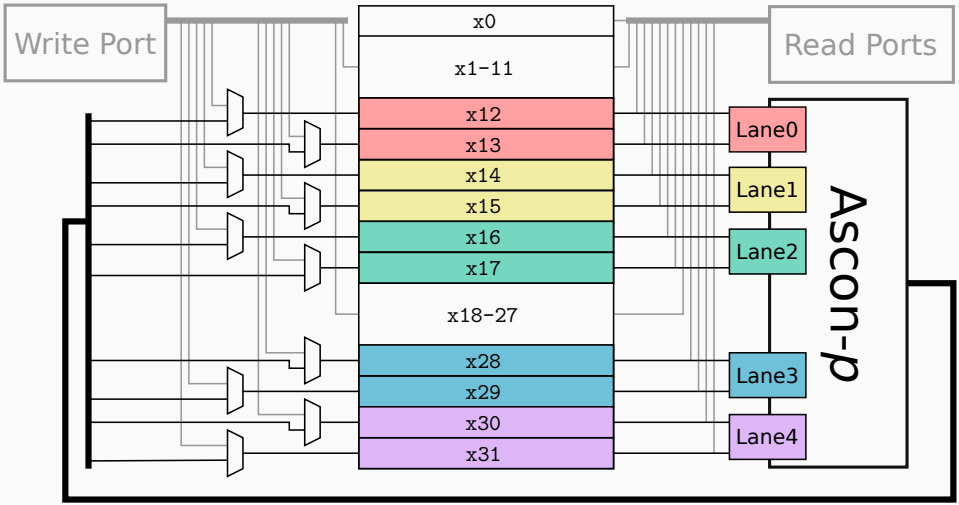
Register File Adaptations

- ASCON-*p* requires ten 32-bit registers
- Solution:
  - Dynamic register selection not strictly needed
  - Define fixed subset as ASCON-*p* registers
  - Toggle between r/w port and accelerator

Register File Adaptations

- ASCON-*p* requires ten 32-bit registers
- Solution:
  - Dynamic register selection not strictly needed
  - Define fixed subset as ASCON-*p* registers
  - Toggle between r/w port and accelerator
- Each ASCON-*p* 64-bit lanes split to two registers

Register File Layout



Decode Stage Adaptations

- Add instruction to decoder

Decode Stage Adaptations

- Add instruction to decoder
- Add single cycle Ascon-p accelerator

Decode Stage Adaptations

- Add instruction to decoder
- Add single cycle Ascon-p accelerator
- Ascon-p instruction:
  - Enables accelerator
  - Toggles register inputs to accelerator output

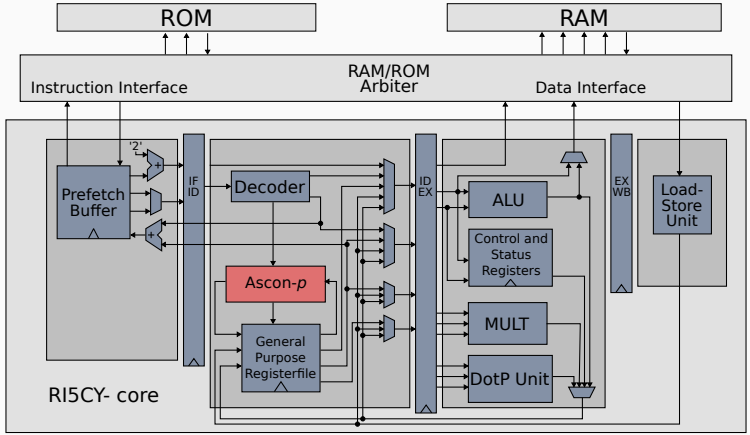
Decode Stage Adaptations

- Add instruction to decoder
- Add single cycle Ascon-*p* accelerator
- Ascon-*p* instruction:
  - Enables accelerator
  - Toggles register inputs to accelerator output
- Limitations:
  - ALU and load-store unit forward results to next instruction

Decode Stage Adaptations

- Add instruction to decoder
- Add single cycle Ascon-p accelerator
- Ascon-p instruction:
  - Enables accelerator
  - Toggles register inputs to accelerator output
- Limitations:
  - ALU and load-store unit forward results to next instruction
  - $\rightarrow$ Ascon-p registers must not be altered the cycle before
  - $\rightarrow$ Ascon-p registers must not be loaded to two cycles before

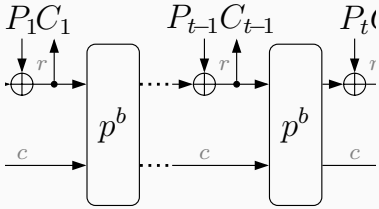
Ascon-p Integration



- Main data processing loop

Code 1 Encrypt Block Loop

```
1:  <encrypt_block_start>:
2:    lw t1,0(t0)
3:    lw s1,4(t0)
4:    xor a2,a2,t1
5:    xor a3,a3,s1
6:    sw a2,0(s8)
7:    sw a3,4(s8)
8:    ASCON-P(ROUND_CONSTANT)
          .
          .
          .
          6-times
14:   addi t0,t0,8
15:   addi s8,s8,8
16:   bgeu t2,t0, <encrypt_block_start>
```



Plaintext

Performance

Table 1: RISC-V RI5CY: Runtime and code size comparison of Ascon and Isap, with/without 1-round Ascon-p hardware acceleration (`HW-A`)

Implementations	Cycles/Byte			Binary Size (B)
	64 B	1536 B	long	
`Ascon-C (-O3)`	164.3	110.6	108.3	11 716
`Ascon-C (-Os)`	269.7	187.1	183.5	2 104
`Ascon-ASM + HW-A`	4.2	2.2	2.1	888
`AsconHash-ASM + HW-A`	4.6	2.6	2.5	484
`ISAP-A-128a-C (-O3)`	1 184.3	386.9	352.3	11 052
`ISAP-A-128a-ASM + HW-A`	29.1	5.2	4.2	1 844

Table 2: Area comparison of the RISC-V RI5CY core and various co-processor designs

Design	kGE	
	Standalone	Integration
RI5CY base design	45.6	-
This work	4.2	0.5
ASCON co-processor [Gro+15]	7.1	?
ASCON co-processor [Gro]	9.4	?
ISAP co-processor (estimated) [Dob+19]	$\leq$ 12.8	?

Implementation Security of ISAP

Effect on Simple Power Analysis (SPA)

- ISAP offers protection/hardening against DPA, DFA, SFA, SIFA
  - See specification for details

Effect on Simple Power Analysis (SPA)

- ISAP offers protection/hardening against DPA, DFA, SFA, SIFA
  - See specification for details
- SPA is possible but limited by highly parallel computation in the accelerator

Effect on Simple Power Analysis (SPA)

- ISAP offers protection/hardening against DPA, DFA, SFA, SIFA
  - See specification for details
- SPA is possible but limited by highly parallel computation in the accelerator
- With accelerator state setup most lucrative target.
- Localized EM analysis could allow successful SPA
  - Add e.g. shuffling

Effect on Simple Power Analysis (SPA)

- ISAP offers protection/hardening against DPA, DFA, SFA, SIFA
  - See specification for details
- SPA is possible but limited by highly parallel computation in the accelerator
- With accelerator state setup most lucrative target.
- Localized EM analysis could allow successful SPA
  - Add e.g. shuffling
- Analyzed in detail in the paper

Summary

Final Remarks

- Ascon-p presents a versatile building block

Final Remarks

- Ascon-p presents a versatile building block
- Acceleration of one block can be used for a broad range of cryptographic primitives
  - With and without implementation security

Final Remarks

- Ascon-p presents a versatile building block
- Acceleration of one block can be used for a broad range of cryptographic primitives
  - With and without implementation security
- One can get more performance *AND* better protection from implementation attacks.

Thank you!

References

[Dob+19] C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, B. Mennink, R. Primas, and T. Unterluggauer. ISAP v2.0. Submission to the NIST Lightweight Crypto Competition. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/isap-spec-round2.pdf. 2019.

[Gro] H. Groß. CAESAR Hacrdware API reference implementation. https://github.com/IAIK/ascon_hardware/tree/master/caesar_hardware_api_v_1_0_3/ASCON_ASCON (accessed 12/2019). (Visited on 12/2019).

[Gro+15] H. Groß, E. Wenger, C. Dobraunig, and C. Ehrenhöfer. Suit up! - Made-to-Measure Hardware Implementations of ASCON. In: DSD. IEEE Computer Society, 2015, pp. 645–652.