Masked Software Implementations of Ascon

Theory and Practice

Barbara Gigerl, Florian Mendel, Robert Primas, Martin Schläffer

Our Contribution

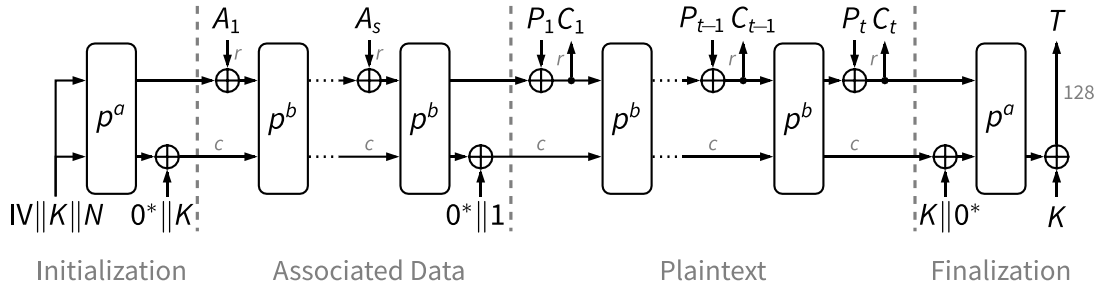
- Efficient second-order masking of Ascon in software
  - Based on first-order masked implementation of Ascon [DEM+22]
- Features / additional SCA hardening
  - Withstand bivariate t-tests
  - Microarchitecture-aware optimizations
  - No online randomness
- Approx. same performance with/without hardware RNG
- Formally verified masking on RISC-V IBEX core
- Code available at: https://github.com/ascon/ascon-c

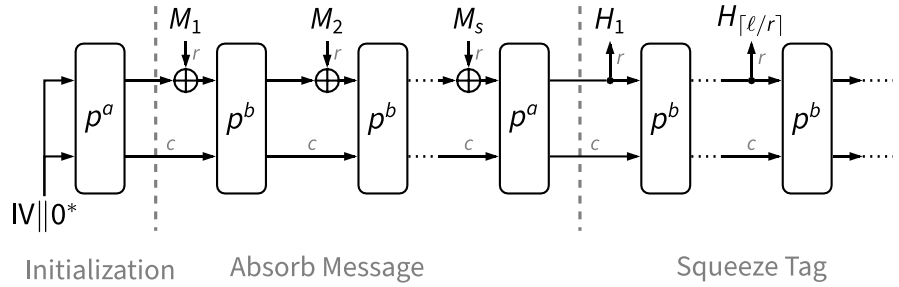
Why Ascon?

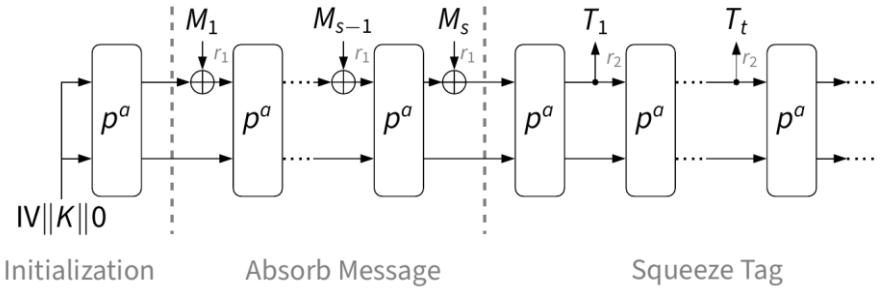
- Selected by **NIST** as **Lightweight Cryptography Standard** for Constrained Devices
- Ascon provides **authenticated** encryption and **hashing** with minimal overhead
- **Comparable security** level as AES-128 and SHA-256
- More efficient on low-end devices (Ascon-128 vs AES128-GCM)
  - Up to **3-5x speed** on microcontrollers
  - Up to **2x throughput** with **0.5x energy** in hardware
- Ascon allows to **secure data**, where this was considered too costly before (energy, area, latency)
- Very **efficient to protect** against physical attacks
  - No table lookups, easy to mask, key used less often

The Ascon design basics

- **Permutation based**
  - Single 320-bit permutation (All)
  - Different number of rounds
- **Sponge based**
  - Duplex-mode (AEAD)
  - Absorb/squeeze (Hash, XOF)
  - High-rate absorption (MAC, PRF)
- **Keyed initialization/finalization**
  - Increases robustness (AEAD)
- **Security proofs** for the constructions

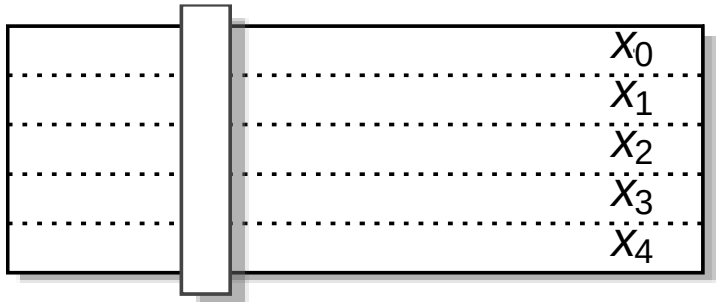


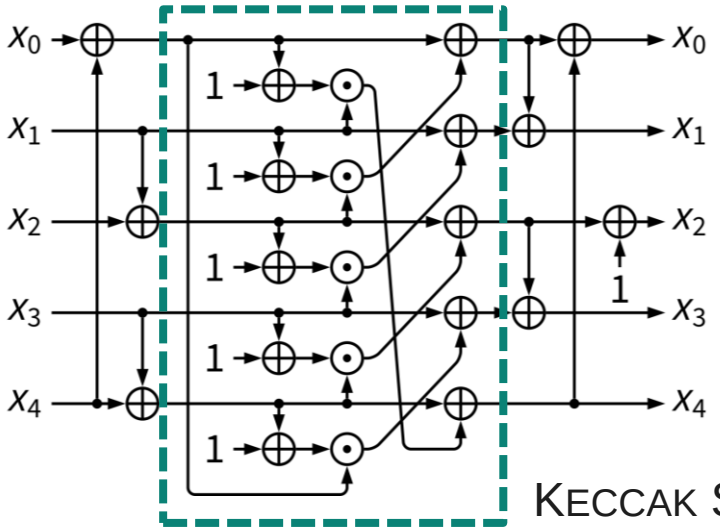




Ascon Permutation: 8/12 rounds

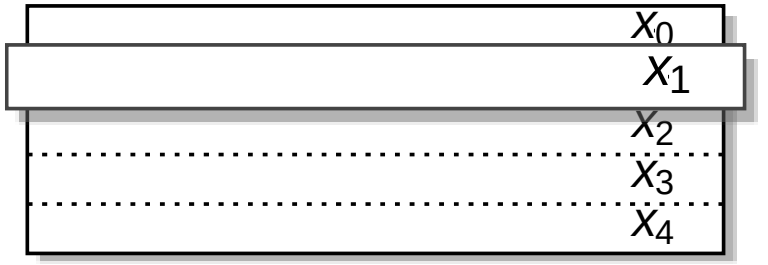
S-box layer





KECCAK S-box

Linear layer

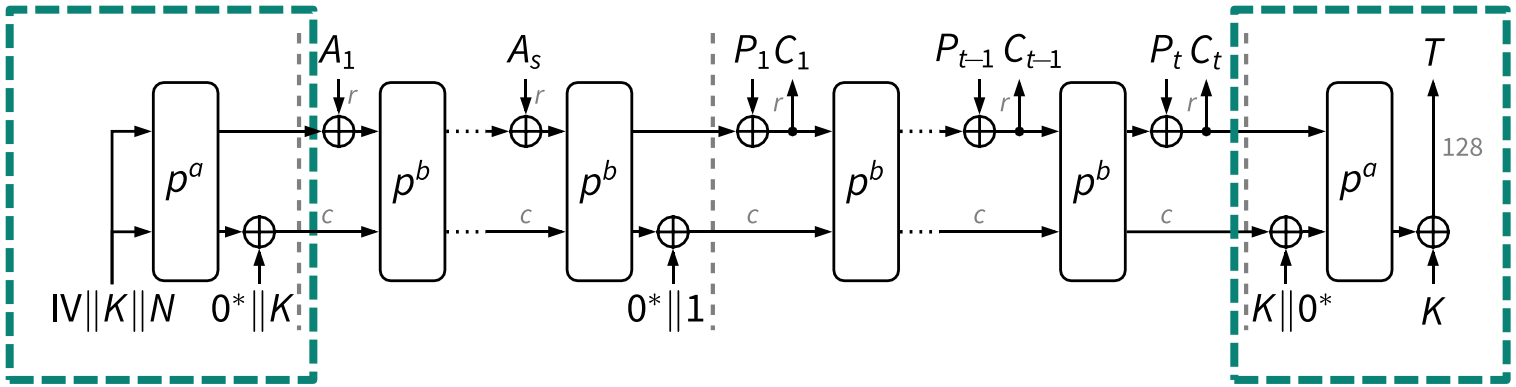


$$
\begin{aligned}
x_0 &= x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \\
x_1 &= x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \\
x_2 &= x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6) \\
x_3 &= x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \\
x_4 &= x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)
\end{aligned}
$$

Ascon: Side-Channel Protection

Ascon was designed with SCA in mind

- Keyed initialization/finalization limit damage if state is recovered



- Levelled implementations [BBC+20]
  - Higher protection order for Init/Final (key)
  - Lower protection order for AD/PT/CT processing (data)
- Algebraic degree 2 of S-box
- Efficient masking using Toffoli gate [DDE+20]

Masking using Toffoli Gate [DDE+20]

- More efficient than masked AND gate
  - Fewer instructions, registers, randomness

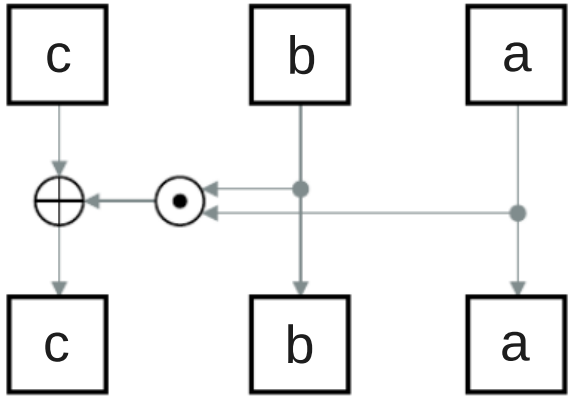
- No fresh randomness needed during round computation
  - Randomness is not lost (invertible shared Toffoli gate)
  - Randomness of previous round can be reused

- Other benefits of invertible shared function:
  - SIFA: Reduced attack surface if used with redundancy [DDE+20]

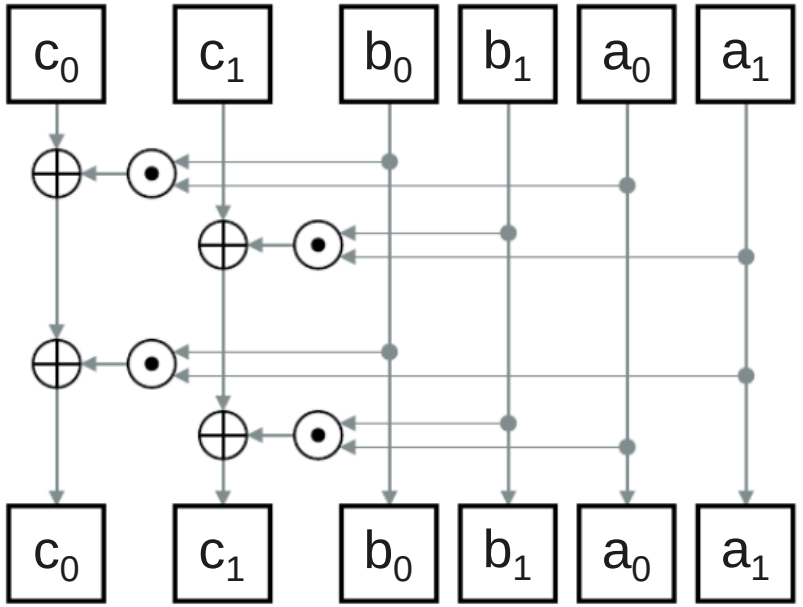
⇨ **This work:** Extend concept to second-order masking

Toffoli Gate $c \leftarrow c \oplus \overline{a}b$

– Simplest invertible non-linear function



Masked Toffoli Gate (2 shares)



Name: $p_{\chi 2S}$

In-/Output: $\{c_0, c_1, a_0, a_1, b_0, b_1\}$

$$c_0 \leftarrow c_0 \oplus \overline{a_0} b_1$$

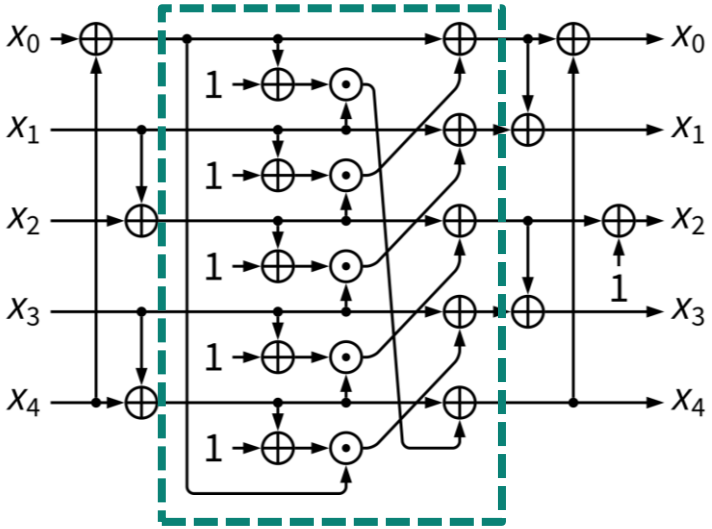
$$c_0 \leftarrow c_0 \oplus \overline{a_0} b_0$$

$$c_1 \leftarrow c_1 \oplus a_1 b_1$$

$$c_1 \leftarrow c_1 \oplus a_1 b_0$$

- Constructed from 4 Toffoli Gates
- Is an invertible function

Masked KECCAK S-box [DDE+20]



Name: χ_{2S}

In-/Output: $\{a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1, e_0, e_1, r_0, r_1\}$

$p_{\chi 2S}(r_0, r_1, e_0, e_1, a_0, a_1)$

$p_{\chi 2S}(a_0, a_1, b_0, b_1, c_0, c_1)$

$p_{\chi 2S}(c_0, c_1, d_0, d_1, e_0, e_1)$

$p_{\chi 2S}(e_0, e_1, a_0, a_1, b_0, b_1)$

$p_{\chi 2S}(b_0, b_1, c_0, c_1, d_0, d_1)$

$d_0 \leftarrow d_0 \oplus r_0$

$d_1 \leftarrow d_1 \oplus r_1$

KECCAK S-BOX

- Five times sequential application of masked Toffoli gate
- Extra random input (r_0,r_1) which is recycled using a changing of the guards construction

⇨ Fewer instructions, registers, randomness

2nd-order Masked Toffoli Gate (3 shares)

Name: $p_{\chi 3S}$

In-/Output: $\{c_0, c_1, c_2, a_0, a_1, a_2, b_0, b_1, b_2, R_0, R_1, R_2\}$

$c_0 \leftarrow c_0 \oplus a_0 b_2$

$c_0 \leftarrow c_0 \oplus a_0 b_1 \oplus R_0$

$c_0 \leftarrow c_0 \oplus \overline{a_0} b_0$

$c_1 \leftarrow c_1 \oplus a_1 b_2$

$c_1 \leftarrow c_1 \oplus \overline{a_1} b_1 \oplus R_1$

$c_1 \leftarrow c_1 \oplus a_1 b_0$

$c_2 \leftarrow c_2 \oplus \overline{b_0} a_2$

$c_2 \leftarrow c_2 \oplus a_2 b_1 \oplus R_2$

$c_2 \leftarrow c_2 \oplus a_2 \curlyvee b_2$

Note: **Additional randomness** needed to achieve 2nd order SCA robustness

Second-order masked KECCAK S-box

Name: χ_{3S}

In-/Output: $\{a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2, r_0, r_1, r_2, R_0, R_1, R_2\}$

$p_{\chi 3S}(r_0, r_1, r_2, e_0, e_1, e_2, a_0, a_1, a_2, R_0, R_1, R_2)$

$\ggg 1$

$p_{\chi 3S}(a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, R_0, R_1, R_2)$

$\ggg 1$

$p_{\chi 3S}(c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2, R_0, R_1, R_2)$

$\ggg 1$

$p_{\chi 3S}(e_0, e_1, e_2, a_0, a_1, a_2, b_0, b_1, b_2, R_0, R_1, R_2)$

$\ggg 1$

$p_{\chi 3S}(b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, R_0, R_1, R_2)$

$d_0 \leftarrow d_0 \oplus r_0$

$d_1 \leftarrow d_1 \oplus r_1$

$d_2 \leftarrow d_2 \oplus r_2$

Note: Reduce randomness requirements by using **changing of the guards**

Further Optimization

Name: $\chi_{3S\star}$

In-/Output: $\{a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2, r_0, r_1, r_2\}$

$(R_0, R_1, R_2) \leftarrow (r_0, r_1, r_2) \ggg 1$

$p_{\chi 3S}(r_0, r_1, r_2, e_0, e_1, e_2, a_0, a_1, a_2, R_0, R_1, R_2); (R_0, R_1, R_2) \leftarrow (R_0, R_1, R_2) \ggg 1$

$p_{\chi 3S}(a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, R_0, R_1, R_2); (R_0, R_1, R_2) \leftarrow (R_0, R_1, R_2) \ggg 1$

$p_{\chi 3S}(c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2, R_0, R_1, R_2); (R_0, R_1, R_2) \leftarrow (R_0, R_1, R_2) \ggg 1$

$p_{\chi 3S}(e_0, e_1, e_2, a_0, a_1, a_2, b_0, b_1, b_2, R_0, R_1, R_2); (R_0, R_1, R_2) \leftarrow (R_0, R_1, R_2) \ggg 1$

$p_{\chi 3S}(b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, R_0, R_1, R_2); (R_0, R_1, R_2) \leftarrow (R_0, R_1, R_2) \ggg 1$

$r_0 \leftarrow r_0 \oplus R_0$

$r_1 \leftarrow r_1 \oplus R_1$

$r_2 \leftarrow r_2 \oplus R_2$

$d_0 \leftarrow d_0 \oplus r_0$

$d_1 \leftarrow d_1 \oplus r_1$

$d_2 \leftarrow d_2 \oplus r_2$

Note: Further reduce randomness requirements by using **changing of the guards** to also replace (R_0,R_1,R_2)

2nd-order Masked ASCON permutation

- Ascon S-box is affine equivalent to KECCAK S-box
- Extension to Ascon permutation is simple using shared linear/affine operations
- **Problem:** Microarchitectural Leakage

Microarchitectural Leakage

- **Problem:** Masking assumes ideal hardware
  - Models assume independent processing of shares
- **Reality**
  - CPUs introduce **hidden interactions**
- **Leakage sources**
  - Register reuse
  - Data transitions
  - Glitches in computation

⇨ Shares can be **recombined unintentionally**

Microarchitecture-aware optimizations

- Assembly implementation of security critical parts of the round function
  - Allows control over the register file usage and order of instructions etc.

- Additional SCA hardening through rotation offsets between shares
  - $x_0 = x_0$
  - $x_1 = x_1 \ggg 5$
  - $x_2 = x_2 \ggg 10$
  - Offsets need to be reversed during non-linear operations
  - Helps avoid transitions/glitches on stack or memory buses

⇨ Efficient second-order masked implementation of Ascon

Performance Evaluation

– Processing one plaintext block in cycles/byte (for long messages)

Implementation	ARM STM32F303	RISC-V IBEX
Plain	59	-
Leveled	89	-
2-shares	318	260*
3-shares	542	500*

*Estimated based on cycle counts of linear and non-linear layer.

Note: Results are for Ascon-128 and **Ascon-128a** (as standardized by NIST) **will be faster**

Practical and Formal Verification

Practical Leakage Evaluation

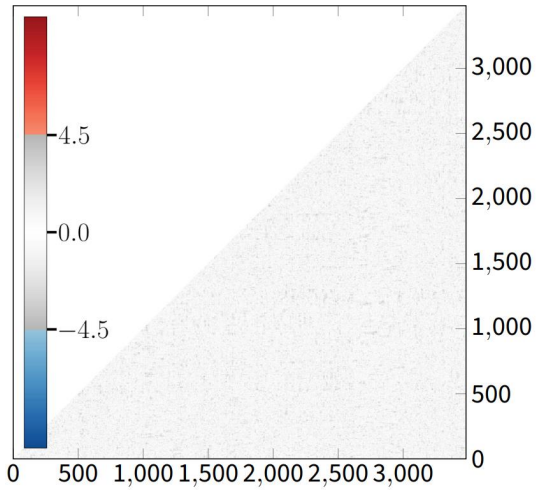
- **Goal:** Practical 2nd-order protection with 3 shares

- **Evaluation setup:**
  - ChipWhisperer-Lite
  - UFO Board
  - STM32F303

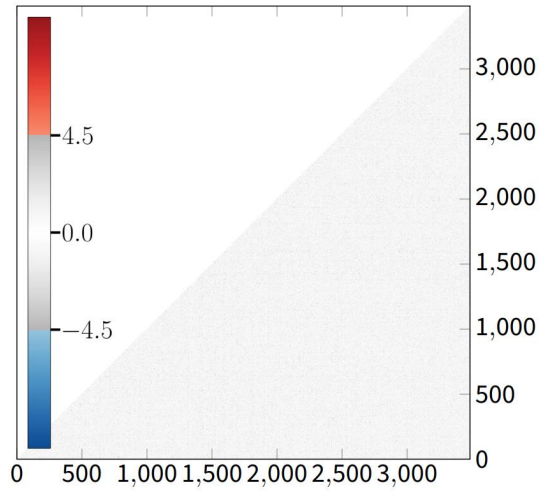
- **Bivariate t-test scenarios:**
  - 3 shares, 1 rounds, share-rotations, 4 samples/clk
  - 3 shares, 4 rounds, share-rotations, 1 samples/clk
  - 3 shares, 1 rounds, **no share-rotations**, 4 samples/clk
  - **2 shares**, 1 rounds, share-rotations, 4 samples/clk

Verification Results

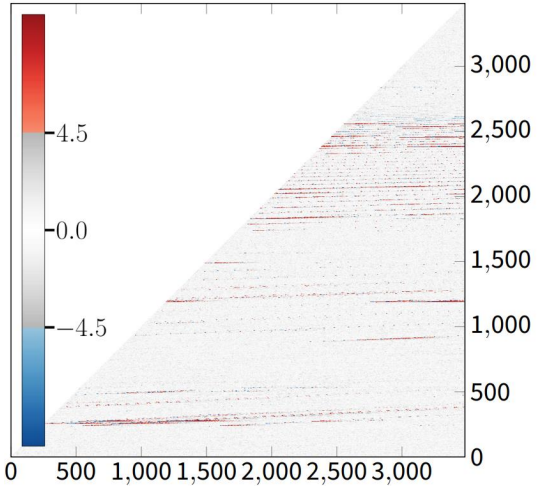
- 3 shares
- 1 rounds
- share-rotations
- 4 samples/clk



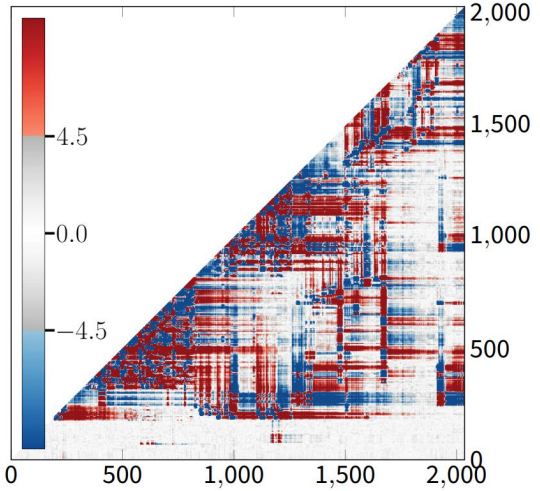
- 3 shares
- 4 rounds
- share-rotations
- 1 sample/clk



- 3 shares
- 1 rounds
- **no share-rotations**
- 4 samples/clk



- **2 shares**
- 1 rounds
- share-rotations
- 4 samples/clk



Formal Masking Verification

- Formal verification of masking in SW/HW using COCO [GHP+21]

- Verifies masked software in “hardware probing model” on CPU netlists
  - Considers stable signals, transitions, glitches
  - RISC-V IBEX core (comparable to ARM Cortex-M0)

- Verification Results

Implementation	Input Labels	Order	Stable		Transient	
			Result	Time	Result	Time
2-share Ascon-*p* round	$5 \times 64 \times 2$ bits	1	✓	3m	✓	5h 20m
3-share Ascon S-box	$5 \times 32 \times 3$ bits	2	✓	26m	✓	1h 17m

* Verification runtimes stem from single-threaded executions on an Intel Core i7 notebook processor with 16GB of RAM.

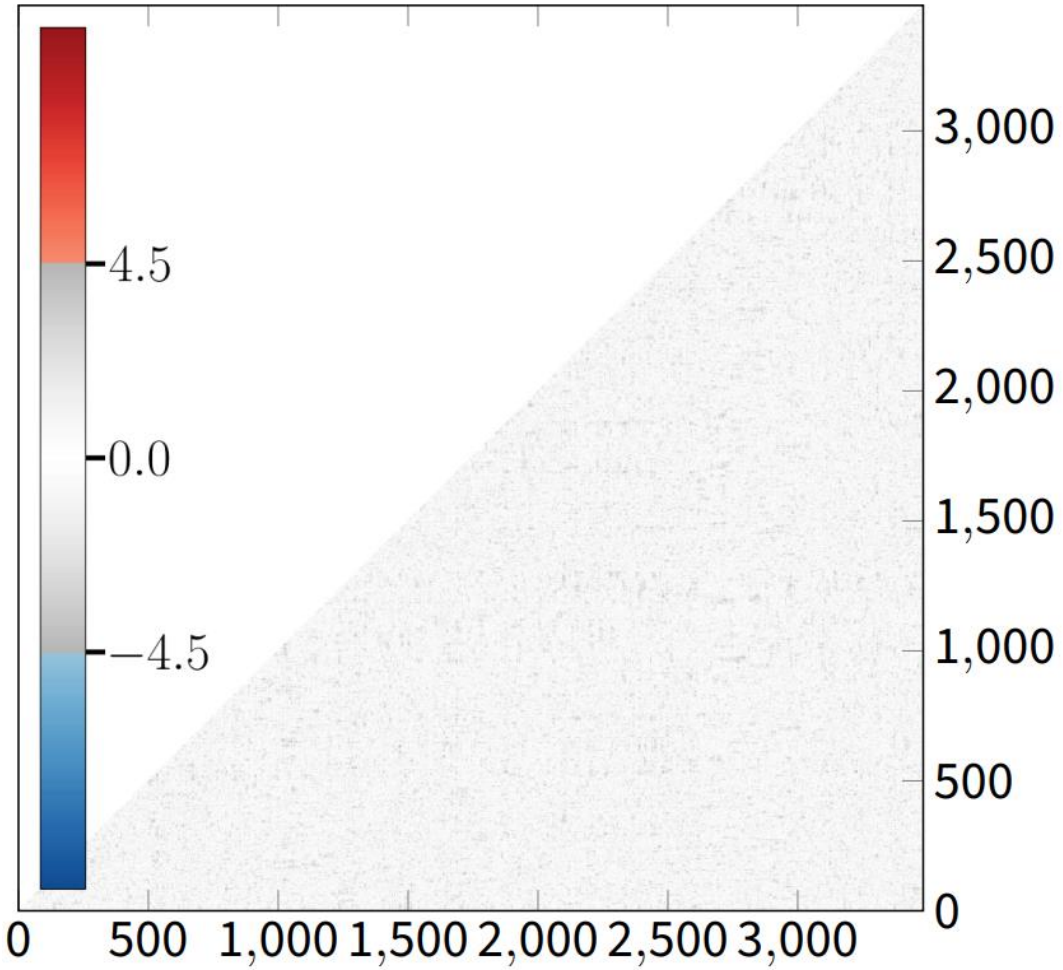
Summary

- Efficient second-order masking of Ascon in software
  - Strong side-channel robustness
  - Comparable low overhead
  - No online randomness required
- Approx. same performance with/without hardware RNG
- Formal and practical verification
- Code available at: https://github.com/ascon/ascon-c

Thank you! Questions?

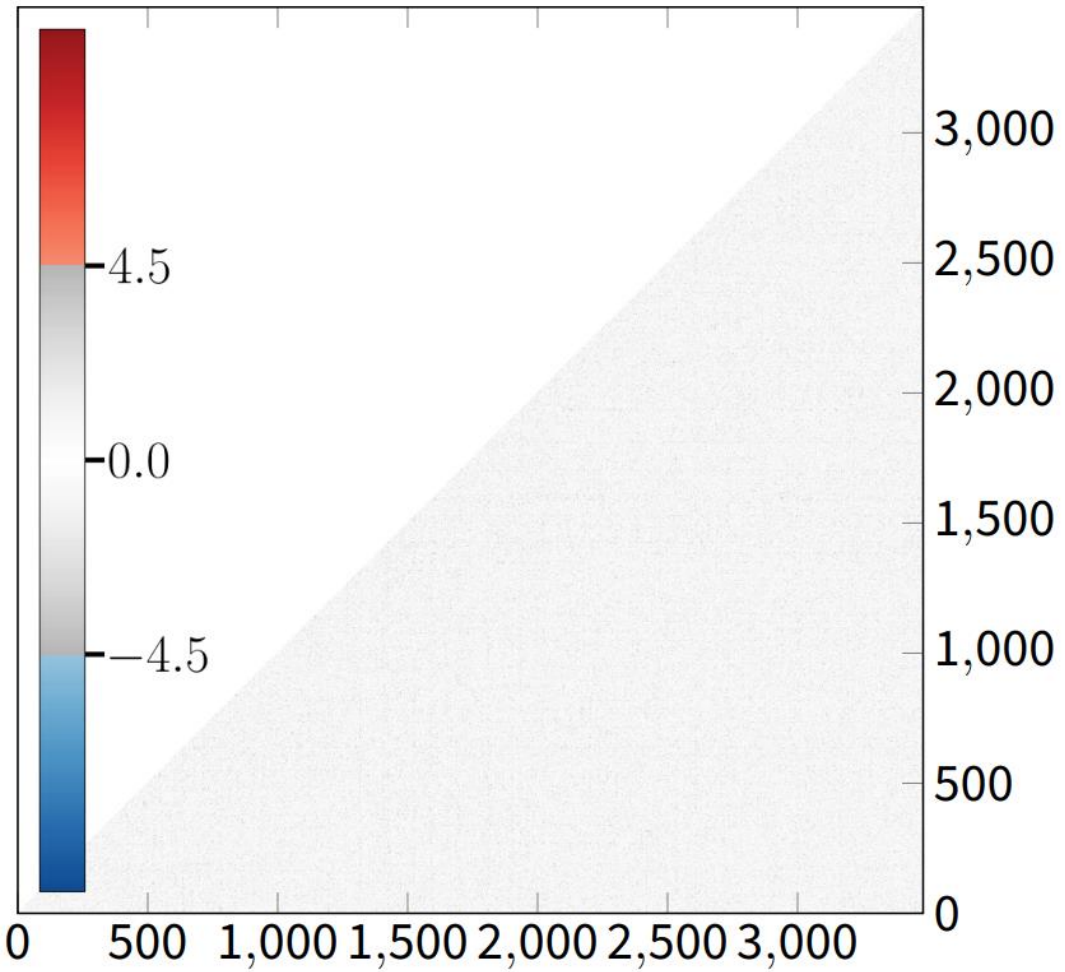
Results

– 3 shares, 1 rounds, share-rotations, 4 samples/clk



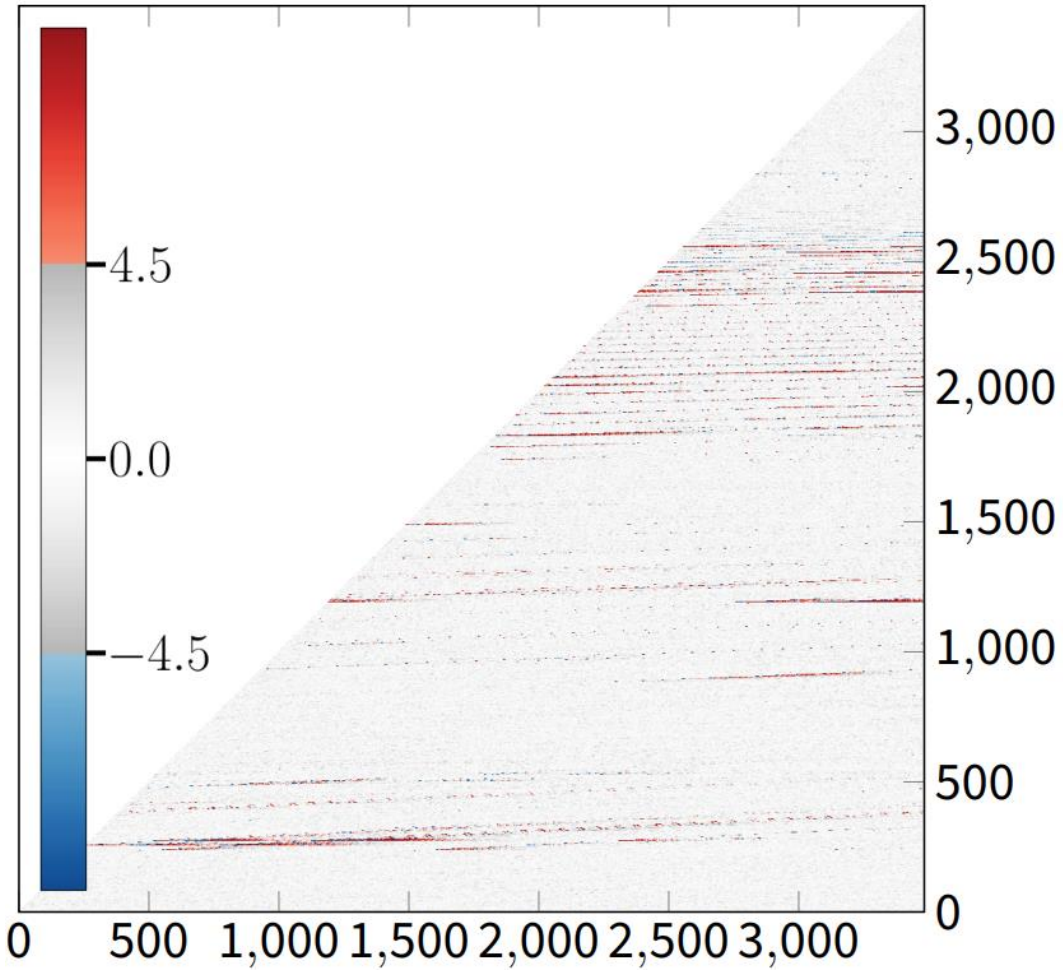
Results

– 3 shares, 4 rounds, share-rotations, 1 samples/clk



Results

– 3 shares, 1 rounds, **no share-rotations**, 4 samples/clk



Results

– **2 shares**, 1 rounds, share-rotations, 4 samples/clk