Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs

Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, Roderick Bloem

2021-05-20

USENIX Security '21

IAIK – Graz University of Technology

Physical Side-Channel Attacks

- Device:
  - Has certain asset, e.g. cryptographic key
  - Examples: Credit card, passport, government IDs, SIM cards, security tokens, ...
  - Microprocessors

Physical Side-Channel Attacks

- Device:
  - Has certain asset, e.g. cryptographic key
  - Examples: Credit card, passport, government IDs, SIM cards, security tokens, ...
  - Microprocessors
- Attacker:
  - Has physical access to device
  - Can observe or manipulate its physical properties, e.g. power consumption

Masking

- Power consumption of CPU depends on:

Masking

- Power consumption of CPU depends on:
  - What instruction is executed?

Masking

- Power consumption of CPU depends on:
    - What instruction is executed?
    - Which data is involved (key)?

Masking

- Power consumption of CPU depends on:
  - What instruction is executed?
  - Which data is involved (key)?

$\longrightarrow$ Break the dependency!

Masking

- Power consumption of CPU depends on:
  - What instruction is executed?
  - Which data is involved (key)?

$\longrightarrow$ Break the dependency!

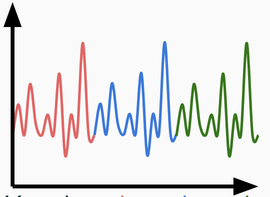
- Masking:
  - Secret sharing technique
  - Split sensitive value into multiple (random) shares
  - Perform computations for each share

Masking

- Power consumption of CPU depends on:
  - What instruction is executed?
  - Which data is involved (key)?

$\longrightarrow$ Break the dependency!

- Masking:
  - Secret sharing technique
  - Split sensitive value into multiple (random) shares
  - Perform computations for each share



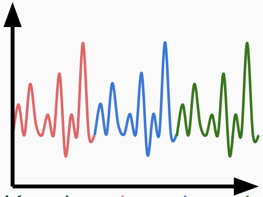
Key $k = k_1 \oplus k_2 \oplus k_3$

Masking

- Power consumption of CPU depends on:
  - What instruction is executed?
  - Which data is involved (key)?

$\longrightarrow$ Break the dependency!

- Masking:
  - Secret sharing technique
  - Split sensitive value into multiple (random) shares
  - Perform computations for each share
- Verification: Check separation of shares
  1. Algorithmically
  2. In a hardware circuit



Key $k = k_1 \oplus k_2 \oplus k_3$

The HW/SW Gap

- So far, formal proofs for masked cryptography exist either:
  - For masked HW circuits (REBECCA[Bloem, 2018])
  - For masked SW
    - Assuming that the underlying HW (CPU netlist) does not cause additional problems

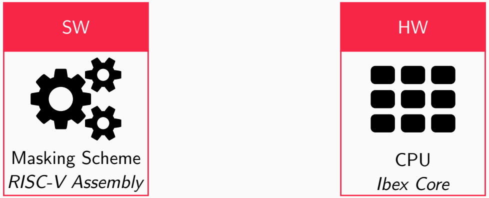
The HW/SW Gap

- So far, formal proofs for masked cryptography exist either:
  - For masked HW circuits (REBECCA[Bloem, 2018])
  - For masked SW
    - Assuming that the underlying HW (CPU netlist) does not cause additional problems

SW
Masking Scheme
RISC-V Assembly

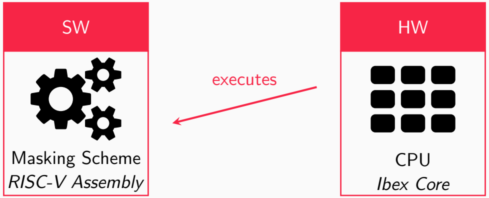
The HW/SW Gap

- So far, formal proofs for masked cryptography exist either:
  - For masked HW circuits (REBECCA[Bloem, 2018])
  - For masked SW
    - Assuming that the underlying HW (CPU netlist) does not cause additional problems



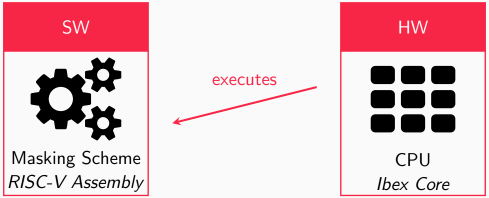
The HW/SW Gap

- So far, formal proofs for masked cryptography exist either:
  - For masked HW circuits (REBECCA[Bloem, 2018])
  - For masked SW
    - Assuming that the underlying HW (CPU netlist) does not cause additional problems



The HW/SW Gap

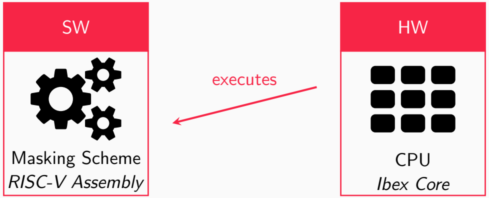
- So far, formal proofs for masked cryptography exist either:
  - For masked HW circuits (REBECCA[Bloem, 2018])
  - For masked SW
    - Assuming that the underlying HW (CPU netlist) does not cause additional problems



- Goal: Co-Verification of SW and HW → Coco

The HW/SW Gap

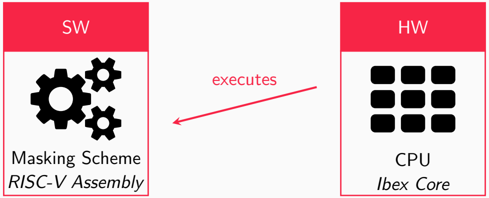
- So far, formal proofs for masked cryptography exist either:
  - For masked HW circuits (REBECCA[Bloem, 2018])
  - For masked SW
    - Assuming that the underlying HW (CPU netlist) does not cause additional problems



- Goal: Co-Verification of SW and HW $\rightarrow$ Coco
  1. Detect leakage of a given masked SW implementation when executed on a given CPU netlist

The HW/SW Gap

- So far, formal proofs for masked cryptography exist either:
  - For masked HW circuits (REBECCA[Bloem, 2018])
  - For masked SW
    - Assuming that the underlying HW (CPU netlist) does not cause additional problems



- Goal: Co-Verification of SW and HW → Coco
  1. Detect leakage of a given masked SW implementation when executed on a given CPU netlist
  2. Construct SCA-hardened CPU components

HW as a Threat to Masked SW

- Attacker observes fluctuations of specific wire for one clock cycle until signal is stable - what leakage could be seen?

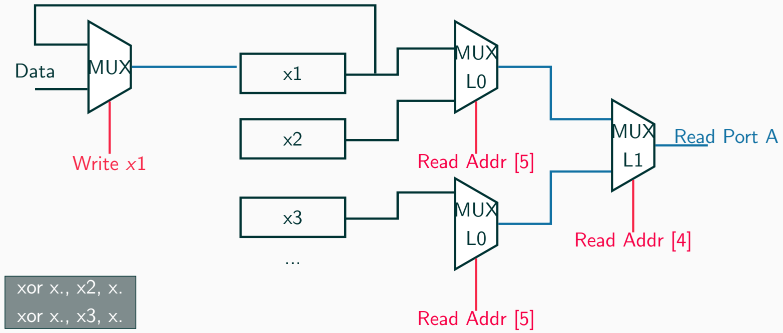
HW as a Threat to Masked SW

- Attacker observes fluctuations of specific wire for one clock cycle until signal is stable - what leakage could be seen?
    - Transitions: leakage depending on both current and previous value

HW as a Threat to Masked SW

- Attacker observes fluctuations of specific wire for one clock cycle until signal is stable - what leakage could be seen?
  - Transitions: leakage depending on both current and previous value
  - Glitches: leakage due to propagation delay variation through combinatorial logic
    - Caused by physical hardware properties, e.g. different wire lengths, gate delays, ...

HW as a Threat to Masked SW



xor x., x2, x.
xor x., x3, x.

- Attacker observes fluctuations of specific wire for one clock cycle until signal is stable - what leakage could be seen?
  - Transitions: leakage depending on both current and previous value
  - Glitches: leakage due to propagation delay variation through combinatorial logic
    - Caused by physical hardware properties, e.g. different wire lengths, gate delays, ...

Co-Verification Flow of Coco

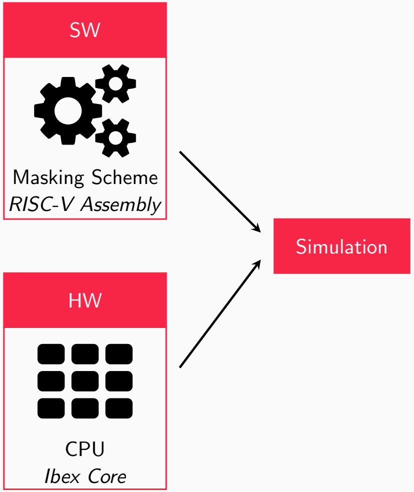
SW

Masking Scheme
RISC-V Assembly

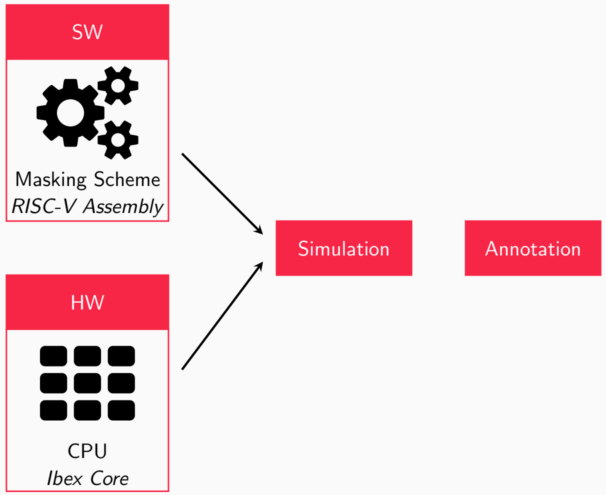
HW

CPU
Ibex Core

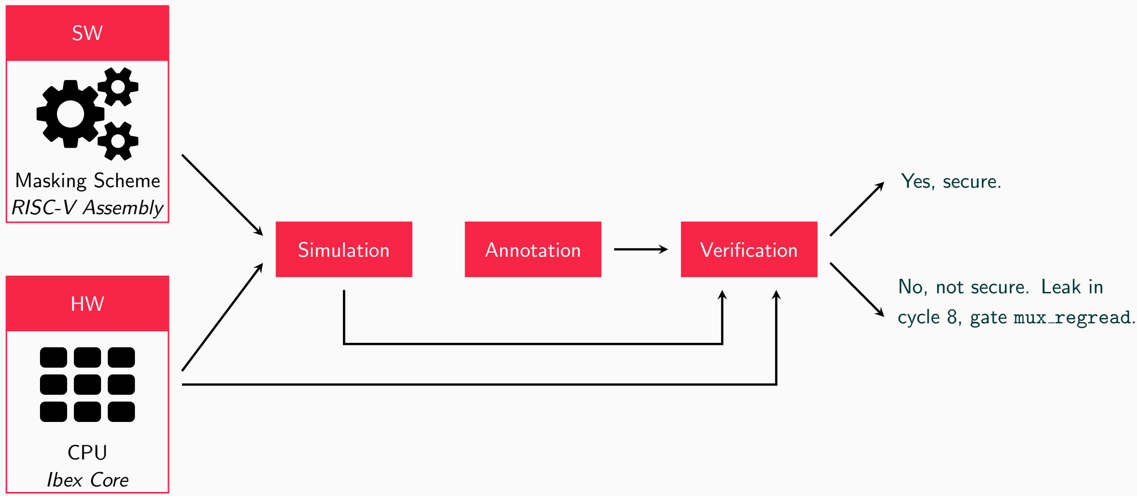
Co-Verification Flow of Coco



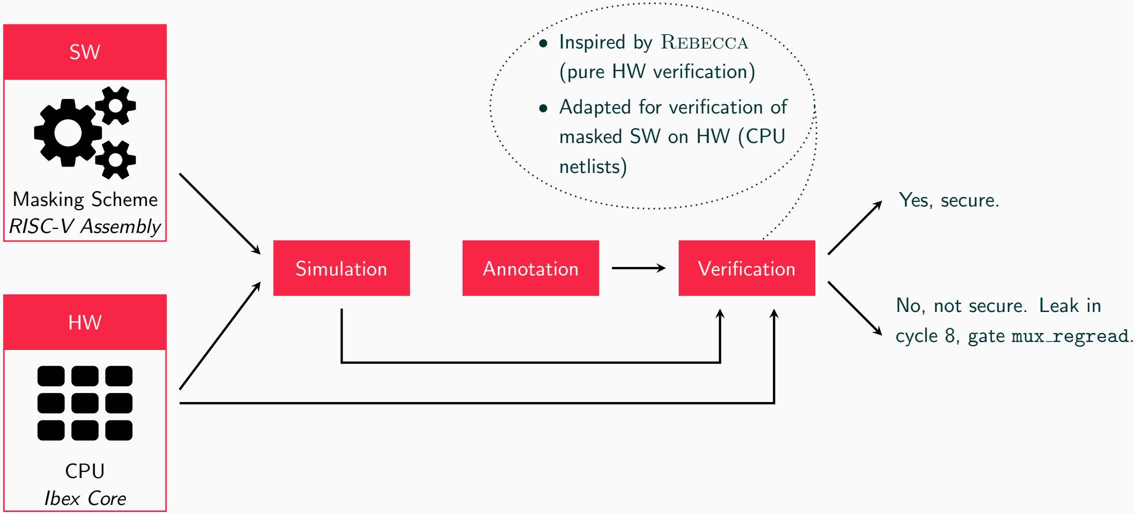
Co-Verification Flow of Coco



Co-Verification Flow of Coco



Co-Verification Flow of Coco


SW
Masking Scheme
RISC-V Assembly
HW
CPU
Ibex Core
Simulation
Annotation
Verification
Inspired by REBECCA (pure HW verification)
Adapted for verification of masked SW on HW (CPU netlists)
Yes, secure.
No, not secure. Leak in cycle 8, gate `mux_regread`.

Co-Design with Coco

- Coco is applicable any processors, as long as netlist ist available

Co-Design with Coco

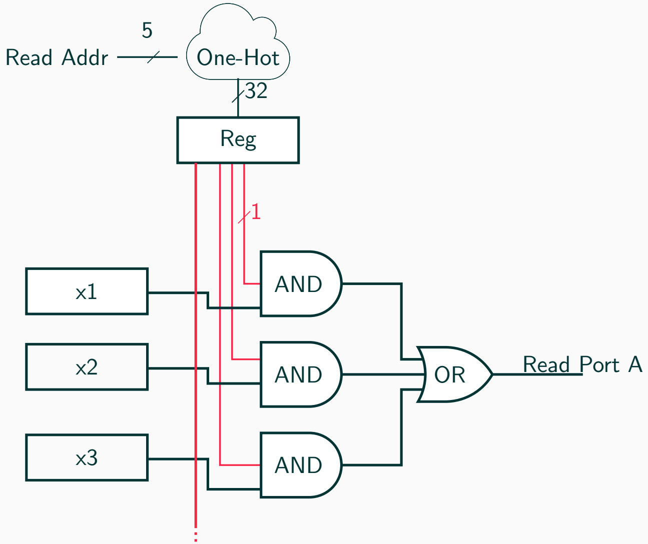
- Coco is applicable any processors, as long as netlist ist available
- Case-study: RISC-V Ibex core
    - 32-bit, 2-stage pipeline, in-order, single-issue

Co-Design with Coco

- Coco is applicable any processors, as long as netlist ist available
- Case-study: RISC-V Ibex core
  - 32-bit, 2-stage pipeline, in-order, single-issue
- Hardening Ibex with Coco
  - Reported leaks in register file, computation units (ALU, Multiplier, CSR Unit), Load-Store Unit, data memory
  - Solution: (1) Hardware fixes and (2) Software Constraints

Example: Hardened Ibex Register File

- Original register file had several problems:
  - Switching wires in multiplexer tree
  - Glitchy address signals
  - Unintended reads



Evaluation

- Area overhead (core excl. SRAM): 9.9% (20.2 kGE vs 22.2 kGE)

Evaluation

- Area overhead (core excl. SRAM): 9.9% (20.2 kGE vs 22.2 kGE)

Name	Runtime (cycles)	Leaking Cycle	Input Shares	Fresh Randomness	Verification Runtime	
					Stable	Transient
Trichina AND reg.	19	-	4×32 bit	32 bit	5 s	19 s
DOM AND reg. ✖	13	12	4×32 bit	32 bit	2 s	12 s
DOM AES S-box	1900	-	16×16 bit	34×16 bit	18 m	4.75 h
DOM Keccak S-box 2nd order	474	-	15×32 bit	15×32 bit	3 m	1.3 h
DOM AND reg. 3rd order	65	-	8×32 bit	6×32 bit	44 s	2.5 m

Evaluation

- Area overhead (core excl. SRAM): 9.9% (20.2 kGE vs 22.2 kGE)

Name	Runtime (cycles)	Leaking Cycle	Input Shares	Fresh Randomness	Verification Runtime	
					Stable	Transient
Trichina AND reg.	19	-	4×32 bit	32 bit	5 s	19 s
DOM AND reg. ✗	13	12	4×32 bit	32 bit	2 s	12 s
DOM AES S-box	1900	-	16×16 bit	34×16 bit	18 m	4.75 h
DOM Keccak S-box 2nd order	474	-	15×32 bit	15×32 bit	3 m	1.3 h
DOM AND reg. 3rd order	65	-	8×32 bit	6×32 bit	44 s	2.5 m

Evaluation

- Area overhead (core excl. SRAM): 9.9% (20.2 kGE vs 22.2 kGE)

Name	Runtime (cycles)	Leaking Cycle	Input Shares	Fresh Randomness	Verification Runtime	
					Stable	Transient
Trichina AND reg.	19	-	4×32 bit	32 bit	5 s	19 s
DOM AND reg. ✖	13	12	4×32 bit	32 bit	2 s	12 s
DOM AES S-box	1900	-	16×16 bit	34×16 bit	18 m	4.75 h
DOM Keccak S-box 2nd order	474	-	15×32 bit	15×32 bit	3 m	1.3 h
DOM AND reg. 3rd order	65	-	8×32 bit	6×32 bit	44 s	2.5 m

Evaluation

- Area overhead (core excl. SRAM): 9.9% (20.2 kGE vs 22.2 kGE)

Name	Runtime (cycles)	Leaking Cycle	Input Shares	Fresh Randomness	Verification Runtime Stable	Verification Runtime Transient
Trichina AND reg.	19	-	4×32 bit	32 bit	5 s	19 s
DOM AND reg. ✖	13	12	4×32 bit	32 bit	2 s	12 s
DOM AES S-box	1900	-	16×16 bit	34×16 bit	18 m	4.75 h
DOM Keccak S-box 2nd order	474	-	15×32 bit	15×32 bit	3 m	1.3 h
DOM AND reg. 3rd order	65	-	8×32 bit	6×32 bit	44 s	2.5 m

Evaluation

- Area overhead (core excl. SRAM): 9.9% (20.2 kGE vs 22.2 kGE)

Name	Runtime (cycles)	Leaking Cycle	Input Shares	Fresh Randomness	Verification Runtime	
					Stable	Transient
Trichina AND reg.	19	-	4×32 bit	32 bit	5 s	19 s
DOM AND reg. ✗	13	12	4×32 bit	32 bit	2 s	12 s
DOM AES S-box	1900	-	16×16 bit	34×16 bit	18 m	4.75 h
DOM Keccak S-box 2nd order	474	-	15×32 bit	15×32 bit	3 m	1.3 h
DOM AND reg. 3rd order	65	-	8×32 bit	6×32 bit	44 s	2.5 m

Summary

- HW/SW Gap: Formal proofs for masked SW wrongly assume that HW is secure

Summary

- HW/SW Gap: Formal proofs for masked SW wrongly assume that HW is secure
- Coco: Co-Verification of SW and HW
    - Co-Verification: Detect leakage in a CPU netlist for masked SW
    - Co-Design: Find HW patches for leaking CPU components

Summary

- HW/SW Gap: Formal proofs for masked SW wrongly assume that HW is secure
- Coco: Co-Verification of SW and HW
  - Co-Verification: Detect leakage in a CPU netlist for masked SW
  - Co-Design: Find HW patches for leaking CPU components
- Case-study: RISC-V Ibex core

Co-Design and Co-Verification of Masked Software Implementations on CPUs

Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, Roderick Bloem

2021-05-20

USENIX Security ’21

IAIK – Graz University of Technology