Secure and Efficient Software Masking on Superscalar Pipelined Processors

Barbara Gigerl, Robert Primas, Stefan Mangard

07/12/2021

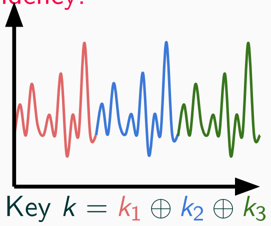
IAIK – Graz University of Technology

Physical Side-Channel Attacks

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

Masking

- Power consumption of CPU depends on:
  - What instruction is executed? $\longrightarrow$ Break the dependency!
  - Which data is involved (key)?
- Masking:
  - Secret sharing technique
  - Split sensitive value into $d+1$ (random) shares
  - Observation of up to d shares does not reveal any information about sensitive value



Masking

ϟ Problem: assumes that independent computations result in independent leakage

- Fine-tune masked implementation for specific microprocessor
- Lazy engineering: use protection order that is higher than theoretically required [BGG+14]

$\Big\}$

- Runtime of masked software is significantly increased
- Still requires manual leakage assessments

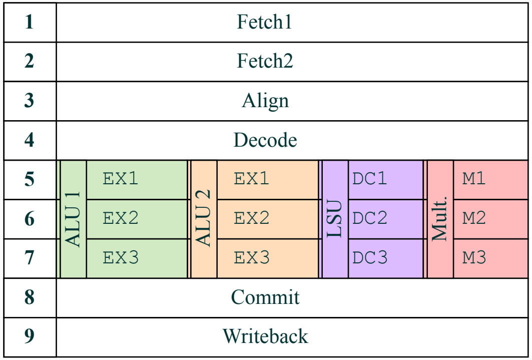
Our Work

- Security of masked software on more complex processors (multiple pipeline stages, forwarding logic, superscalar building blocks, caches, ...)
- Analysis can barely be done manually any more $\rightarrow$ we perform formal analysis
- Case study: RISC-V SweRV core
- Questions:
  - ? Which CPU components cause problems in the context of masking?
  - ? How can we deal with these problems?
  - ? Which general rules need to be fulfilled by masked software running on complex cores?
  - ? How can we still design efficient masked software for complex CPUs?

Target Processor Platform

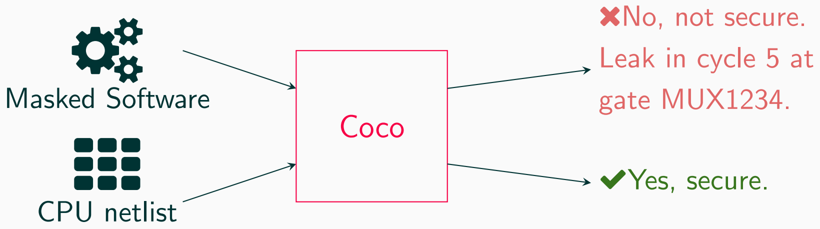
SweRV EH1 Core

- Open-source RISC-V core
- Designed by Western Digital
- Use cases: data-intensive applications (storage controllers, industrial IoT)
- Comparable to ARM Cortex A15
- In-order, dual-issue, load/store buffers
- 9 pipeline stages



Verification Setup

- Goal: investigate security of masked software when executed on a specific CPU
  - Classical probing model for HW: attacker uses d probes to measure specific gate/wire
  - Captures hardware effects like glitches and transitions but too powerful for masked software
  - Time-constrained probing model: Attacker can use d probes to measure specific gate/wire for the duration of one clock cycle
- Coco [GHP+21]: Co-Verification and Co-Design



Verification Setup

- Simple RISC-V Ibex core was analyzed before [GHP+21]
  - Problematic hardware components: register file, ALU, Load-Store unit
  - Needs modification of hardware by applying hardware fixes and software constraints
  - *Secured* Ibex: allows the secure execution of masked software as long as it follows constraints
- Setup Modifications
  - Initial analysis with Coco shows: SweRV has similar problems
  - We map these hardware fixes to SweRV $\rightarrow$ *secured* SweRV as our base point

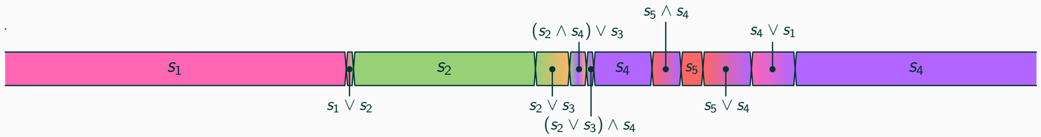
Formal Analysis

- Starting point: Verify software generated by Tornado [BDM+20]
  - Tornado: generates masked C implementation based on unmasked high-level description
  - Security proof in register probing model: attacker places probes on individual registers for one cycle
- Experiment:
  - Generate masked Keccak S-box with Tornado up to 4th-order
  - Verify its execution on *secured* SweRV using Coco
- Result: implementations lose all protection orders due to CPU components causing:
  1. *Big* problems (combination of more than two shares)
  2. *Small* problems (combination of up to two shares)

Example of a big problem

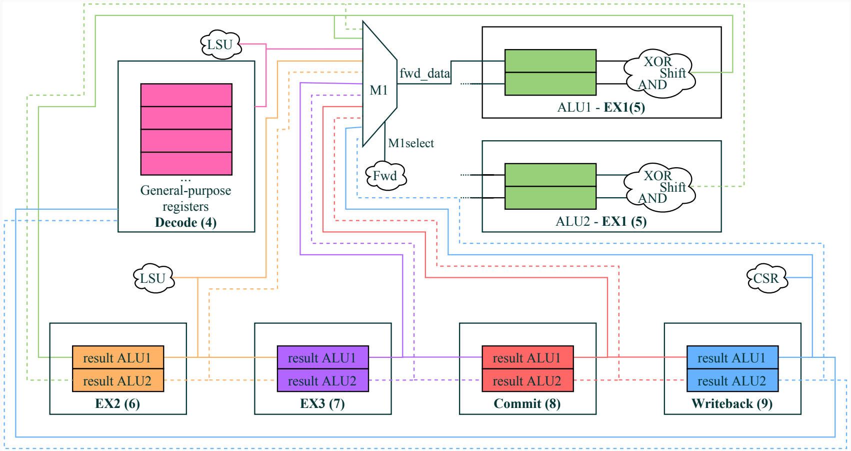
- Software: 10 shares are in the pipeline at the same time (masking is algorithmically correct)
- Gate-level timing simulation of SweRV to visualize possible glitches/transitions on wire
- Based on a specific cell library with concrete timings

Attacker probes a wire in bypass logic for the duration of one cycle - what can be observed?



- Observation of up to five shares is possible (*big* problem)!

Leaks in Pipelines and Execution Units



- M1select is susceptible for glitches
- Leak if multiple shares of the same secret are in different pipeline registers

Leaks in Pipelines and Execution Units

- Possible HW solution
  - Gate output of each pipeline register with a bit indicating whether the value should be forwarded or not
  - Gate-bits need to be glitch-free
  - Impractical due to latency overhead
- Solution in SW
  - Ensure that at no time there are multiple shares of the same native value in different pipeline registers - how?
  - Place enough unrelated instructions between two instructions processing shares of the same native value
  - Unrelated instructions: `nop`, shares from another secret, ALU computations on non-secret data, ...

Leaks in other components

- Management Components of Data Memory:
    - LSU Bus Buffer, Store Pipeline Stages, Data Memory Interface, ...
    - Example: Share is stored in LSU Bus Buffer and gets overwritten by counterpart

Possible HW solutions: again impractical

$\rightarrow$ more SW constraints

- Example: flush LSU Bus Buffer between loading two shares of the same native value

Generic Rules for Masked Software

- Analysis shows: SW constraints are still necessary besides HW fixes
- Effective SW constraint: insertion of unrelated instructions

Generic Rule for Pipelines and Execution Units: For a pipelined processor, the number of unrelated instructions n required is:

$$n = e \times p_d + 1$$

- p Amount of pipeline stages, $p = p_i + p_d$
- p_i Amount of instruction fetch stages
- p_d Amount of data processing stages
- e Amount of execution units

Order reduction when applying lazy engineering: $\left\lfloor \frac{d}{e \times p_d + 1} \right\rfloor$

Efficiency of masked software

If one adapts these rules strictly, the overhead will be huge:

	Masked without constraints		Masked with constraints		
	Cycles	Instructions	Cycles	Instructions	NOPs
DOM AND	10	8	33	48	40
ISW AND	10	8	32	48	40
TI AND	14	15	37	54	39
Trichina AND	9	8	34	46	38
DOM AND 2nd order	20	21	86	148	127
DOM AND 3rd order	33	42	250	295	235

$\rightarrow$ We need specific implementation techniques to reduce overhead.

Serial vs. Parallel Implementations

- Example: Keccak S-box state consists of five lanes (each of d shares)
- Serial: take the d shares of three lanes, process them, store them in the output lane
- Lots of unrelated instructions are needed to separate processing of two shares of the same native value
- Parallel: instead of `NOPs`, use computations of shares of other lanes as unrelated instructions

	Masked without constraints		Masked with constraints		
	Cycles	Instructions	Cycles	Instructions	NOPs
DOM Keccak S-box serial	83	95	240	418	333
DOM Keccak S-box parallel	36	60	81	144	79

Threshold Implementations

- Non-complete component functions: computation is independent of at least one of its input shares
- TI Keccak S-Box: linear layer in sequence for each share, non-linear layer in sequence for each component function
- Ignore *small* problems, but requires three shares for 1st-order security

	Masked without constraints		Masked with constraints		
	Cycles	Instructions	Cycles	Instructions	NOPs
TI Keccak S-box (Input: 15×32 bit)	66	105	72	126	15
TI Ascon (1 round) (Input: 15×64 bit)	721	863	1621	1153	290

Conclusion

Architectural side-effects of complex CPUs can reduce the security of masked software by multiple orders

Problematic components: pipelines, memory management components

Secure and efficient masking requires consideration of HW and SW

Secure and Efficient Software Masking on Superscalar Pipelined Processors

Barbara Gigerl, Robert Primas, Stefan Mangard

07/12/2021

IAIK – Graz University of Technology

Bibliography

GHP^{+}21 Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs. 30th USENIX Security Symposium,USENIX Security 2021, 2021.

BDM^{+}20 Sonia Belaïd, Pierre-Alain Fouque, Darius Mercadier, Matthieu Rivain, and Raphael Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In EUROCRYPT (3), volume 12107 of Lecture Notes in Computer Science, pages 311–341. Springer, 2020.

BGG^{+}14. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers, volume 8968 of Lecture Notes in Computer Science, pages 64–81. Springer, 2014.