eLIMInate: a Leakage-focused ISE for Masked Implementation

Hao Cheng
University of Luxembourg
hao.cheng@uni.lu

Daniel Page
University of Bristol
daniel.page@bristol.ac.uk

Weijia Wang
Shandong University
wjwang@sdu.edu.cn

05/09/24

Context (1)

- Problem: leakage stemming from the **overwriting effect** (see, e.g., [13, Section 3.1])

$$\begin{array}{l} \vdots \\ \mathsf{GPR}[x] \longleftarrow v_i \\ \vdots \\ \mathsf{GPR}[x] \longleftarrow v_j \quad \rightsquigarrow \Lambda \simeq \mathrm{HD}(v_i, v_j) \\ \vdots \end{array}$$

which is an issue if v_i and v_j are Boolean shares of some underlying v.

Context (1)

- Problem: leakage stemming from the **overwriting effect** (see, e.g., [13, Section 3.1])

$$
\begin{array}{lcll}
 & \vdots & & \\
\mathsf{GPR}[x] & \longleftarrow & v_i & \\
 & \vdots & & \\
\mathsf{GPR}[x] & \longleftarrow & 0 & \rightsquigarrow \Lambda \simeq \mathrm{HD}(v_i, 0) \\
\mathsf{GPR}[x] & \longleftarrow & v_j & \rightsquigarrow \Lambda \simeq \mathrm{HD}(0, v_j) \\
 & \vdots & &
\end{array}
$$

  which is an issue if v_i and v_j are Boolean shares of some underlying v.

- Approach #1: alter software implementation to eliminate such overwriting, e.g.,
  1. "flush" resource by pre-zero'ing it,
  2. select different resource allocation,
  3. ...

Context (1)

- Problem: leakage stemming from the **overwriting effect** (see, e.g., [13, Section 3.1])

$$\begin{array}{l} \vdots \\ \mathsf{GPR}[x] \longleftarrow v_i \\ \vdots \\ \mathsf{GPR}[y] \longleftarrow v_j \quad \rightsquigarrow \Lambda \simeq \mathrm{HD}(w, v_j) \\ \vdots \end{array}$$

  which is an issue if v_i and v_j are Boolean shares of some underlying v.

- Approach #1: alter software implementation to eliminate such overwriting, e.g.,
  1. "flush" resource by pre-zero'ing it,
  2. select different resource allocation,
  3. ...

Context (1)

- Problem: leakage stemming from the **overwriting effect** (see, e.g., [13, Section 3.1])

$$\begin{array}{l} \vdots \\ \mathsf{GPR}[x] \longleftarrow v_i \\ \vdots \\ \mathsf{GPR}[x] \longleftarrow v_j \quad \rightsquigarrow \Lambda \simeq \mathrm{HD}(v_i, v_j) \\ \vdots \end{array}$$

  which is an issue if v_i and v_j are Boolean shares of some underlying v.

- Approach #1: alter software implementation to eliminate such overwriting, e.g.,
  1. "flush" resource by pre-zero'ing it,
  2. select different resource allocation,
  3. ...

- Challenge(s):
  1. modulo automation (e.g., Rosita [14, 15]), challenging re. scale, complexity, etc.,
  2. some, e.g., micro-architectural resources cannot be explicitly controlled,
  3. a secure solution may be inefficient,
  4. ...

Context (1)

- Problem: leakage stemming from the **overwriting effect** (see, e.g., [13, Section 3.1])

$$\begin{array}{c} \vdots \\ \mathsf{GPR}[x] \longleftarrow v_i \\ \vdots \\ \mathsf{GPR}[x] \longleftarrow v_j \\ \vdots \end{array}$$

  which is an issue if v_i and v_j are Boolean shares of some underlying v.
- Approach #2: task hardware, i.e., the micro-architecture with overwriting elimination.

Context (1)

- Problem: leakage stemming from the **overwriting effect** (see, e.g., [13, Section 3.1])

$$\begin{array}{c} \vdots \\ \mathsf{GPR}[x] \longleftarrow v_i \\ \vdots \\ \mathsf{GPR}[x] \longleftarrow v_j \\ \vdots \end{array}$$

  which is an issue if v_i and v_j are Boolean shares of some underlying v.
- Approach #2: task hardware, i.e., the micro-architecture with overwriting elimination.
- Challenge(s):
  1. per Gigerl et al. [8, Section 3], "*[w]hile fixing such [overwriting] problems in hardware would, in principle, be possible, it would be very costly*",
  2. the ISA prevents knowledge of, e.g., security-critical vs. security-agnostic operands.

Context (1)

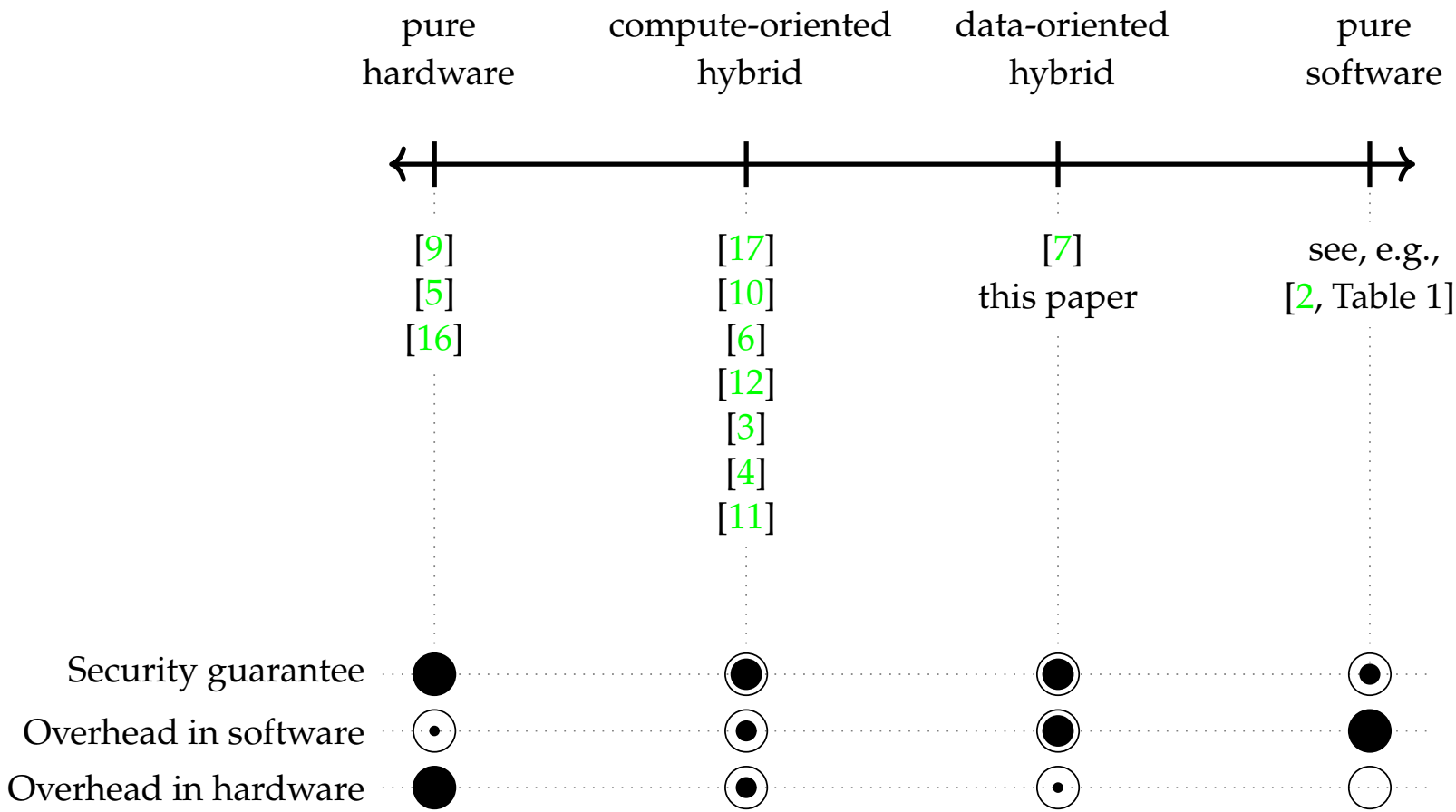
- Problem: leakage stemming from the **overwriting effect** (see, e.g., [13, Section 3.1])

$$\begin{array}{c} \vdots \\ \mathsf{GPR}[x] \longleftarrow v_i \\ \vdots \\ \mathsf{GPR}[x] \longleftarrow v_j \\ \vdots \end{array}$$

  which is an issue if v_i and v_j are Boolean shares of some underlying v.
- Approach #3: hardware/software co-design, i.e., an ISE ...

Context (2)

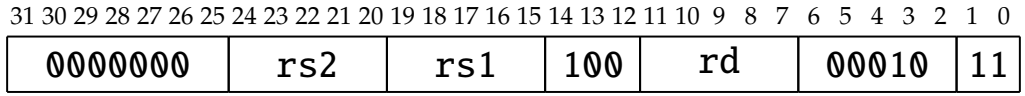
- ... an aside: there's a large design space of approaches related to masking, e.g.,



eLIMInate'ing (only) overwriting-based leakage (1)

- Design:
  - base ISA is RV32I,
  - class-1 instructions: *computation*-related, e.g.,

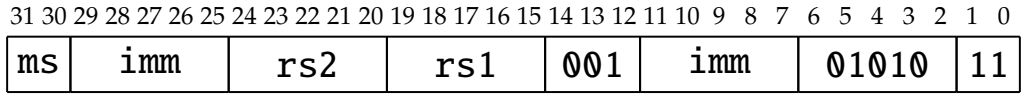
`sec.xor rd, rs1, rs2` :



$$\mapsto \quad \texttt{GPR[rd]} \overset{\Delta}{\longleftarrow} \texttt{GPR[rs1]} \oplus \texttt{GPR[rs2]}$$

  - class-2 instructions: *storage*-related, e.g.,

`sec.sw rs2, rs1, imm, ms` :



$$\mapsto \quad \texttt{MEM[GPR[rs1]} + \texttt{imm]}^4 \overset{\Delta}{\longleftarrow} \texttt{GPR[rs2]}$$

i.e.,

- $\overset{\Delta}{\longleftarrow}$ vs. $\longleftarrow$ denotes a "hint" to eliminate overwriting (noting flexibility re. implementation),
- otherwise functional semantics remain the same,
- so *ideal* outcome is change from, e.g., `xor` to `sec.xor` and that's it.

eLIMInate'ing (only) overwriting-based leakage (2)

- Implementation:
  - base core is (vanilla) Ibex,
  - implementation #1: latency-optimised.

$$\begin{array}{lcl} \text{class-1 instructions} & \simeq & \text{double-buffer} \\ \text{class-2 instructions} & \simeq & \text{mask/unmask at egress/ingress} \end{array}$$

  - implementation #2: area-optimised.

$$\left.\begin{array}{l} \text{class-1 instructions} \simeq \\ \text{class-2 instructions} \simeq \end{array}\right\} \text{double-step (cf. pre-charge then evaluate)}$$

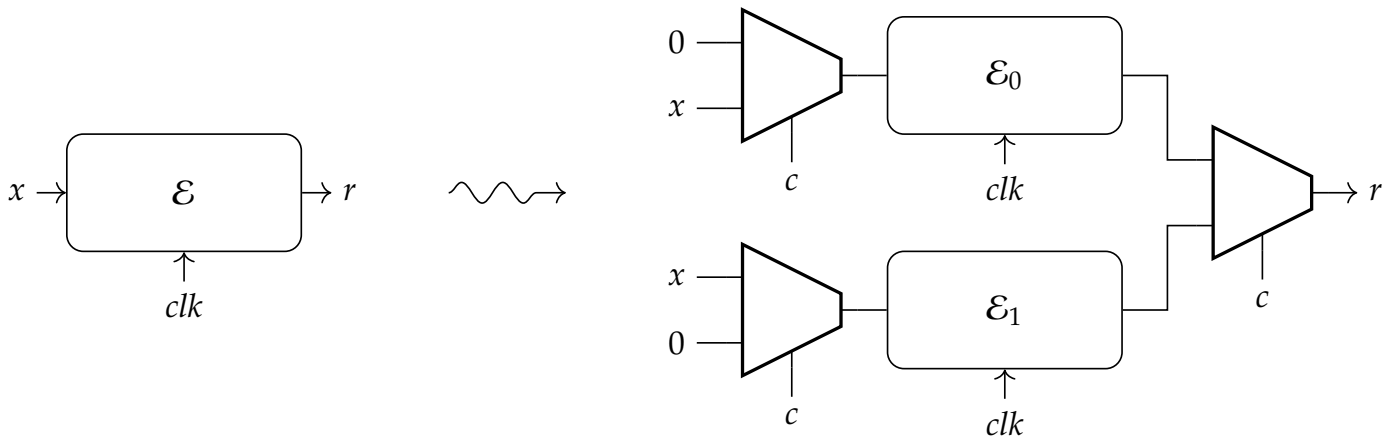
  noting that
  - both adopt conservative assumption re. extra-core resources,
  - latency-optimised class-2 instructions depend on state, i.e., CSRs, indexed by `ms` field.

eLIMInate'ing (only) overwriting-based leakage (3)

- Implementation: latency-optimised, class-1.
  - example instruction:

```
sec.xor x1, x2, x3
```

  - (general-purpose) option #1: double-buffer



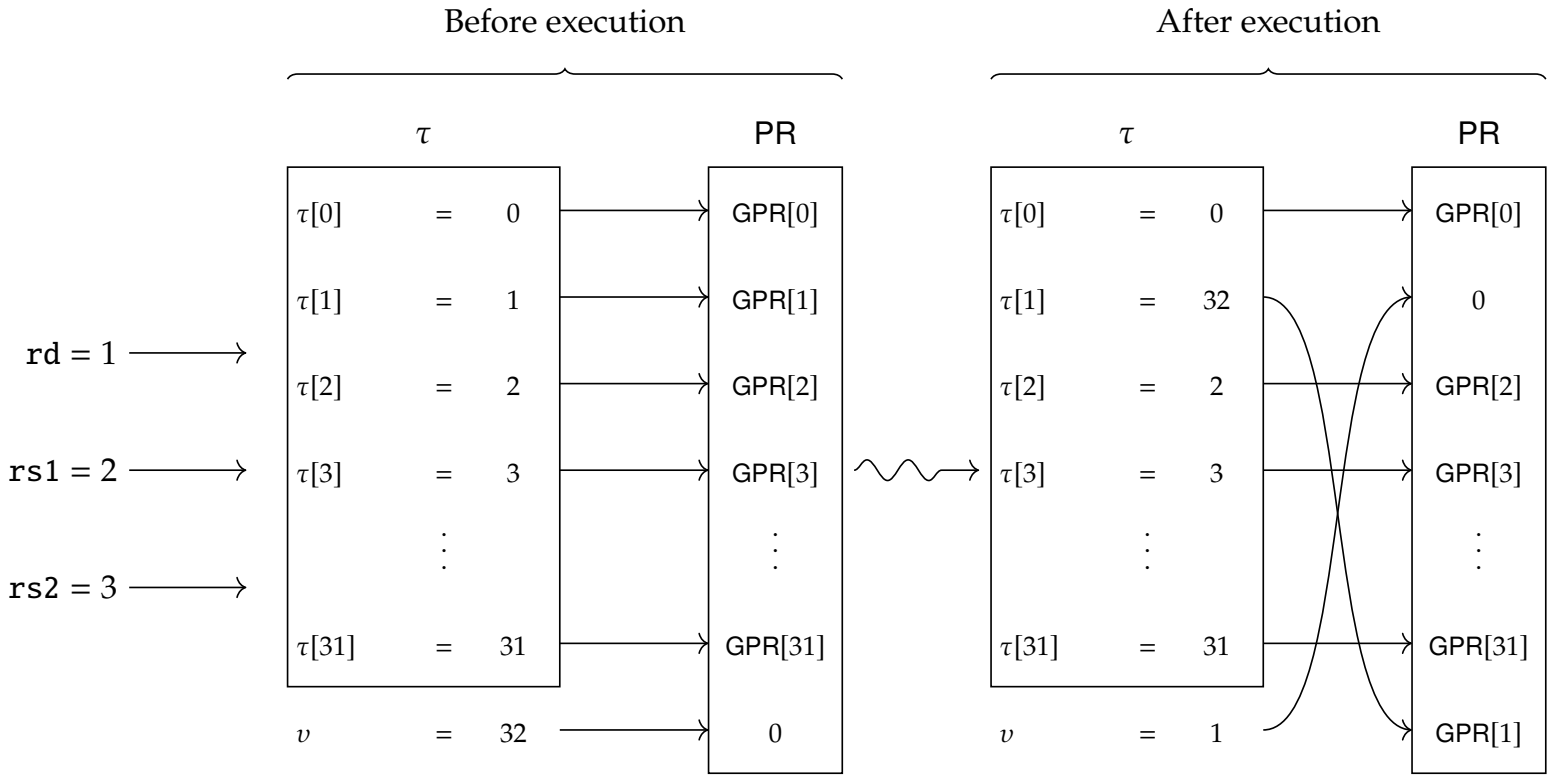
for isolated registers.

eLIMInate'ing (only) overwriting-based leakage (3)

- Implementation: latency-optimised, class-1.
  - example instruction:

```
sec.xor x1, x2, x3
```

  - (special-purpose) option #2: light-weight register renaming



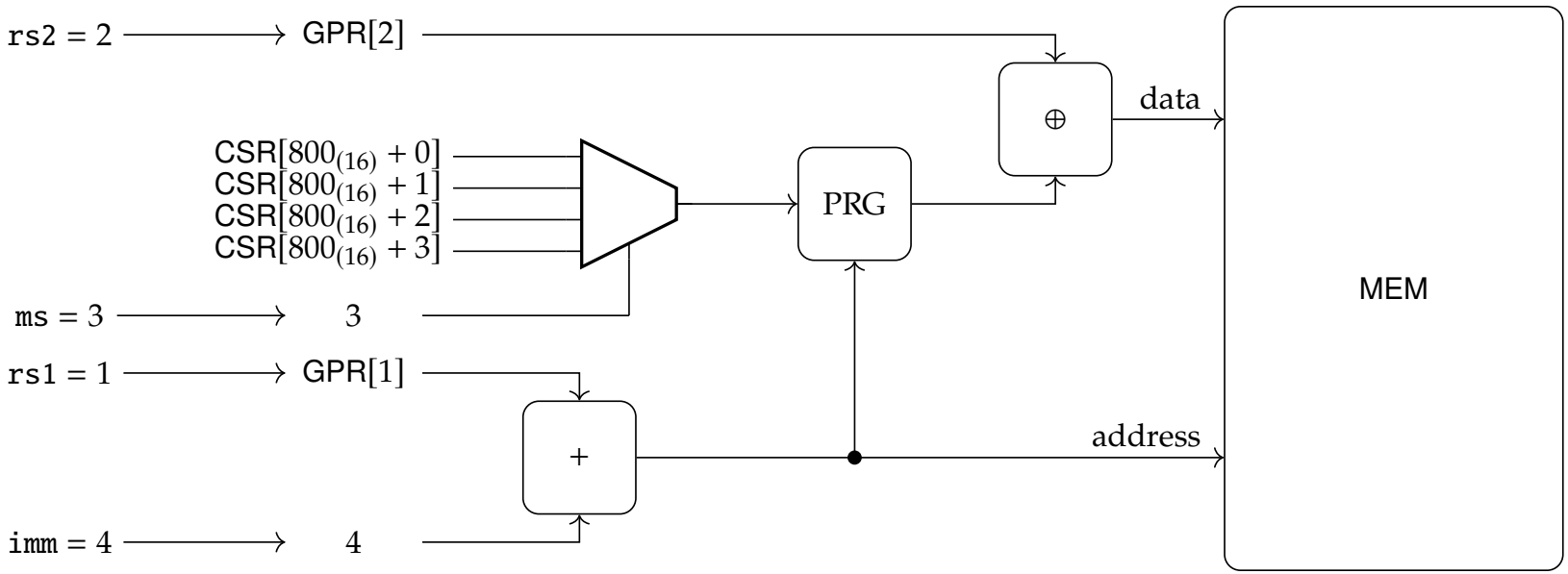
for register file.

eLIMInate'ing (only) overwriting-based leakage (4)

- Implementation: latency-optimised, class-2.
  - example instruction:

```
sec.sw x2, x1, 4, 3
```

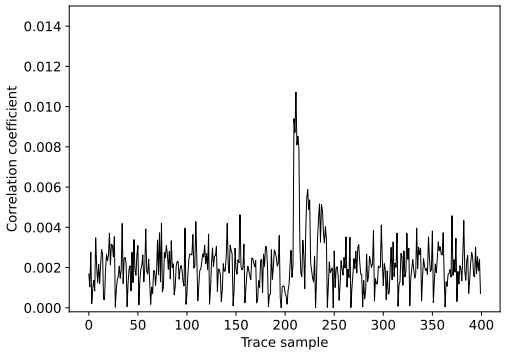
  - executed per



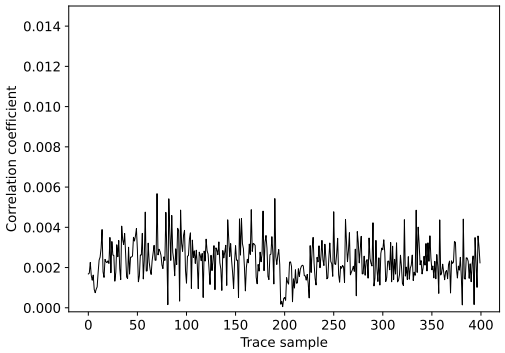
i.e., address-specific *re*masking, so basically the same as [5, Section 4].

eLIMInate'ing (only) overwriting-based leakage (5)

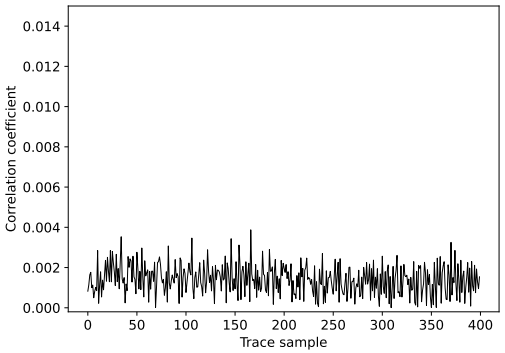
- Evaluation: based on CW305, so Xilinx Artix-7 FPGA.
  1. security-oriented:
     - experimental evidence CPA-based [1] leakage evaluation, e.g.,



`xor`



`sec.xor`
latency-opt.



`sec.xor`
area-opt.

     plus
     - analytical evidence from CoCo-based [8] verification.

---

https://www.newae.com/products/nae-cw305

eLIMInate'ing (only) overwriting-based leakage (5)

- Evaluation: based on CW305, so Xilinx Artix-7 FPGA.

2. efficiency-oriented:
   - area:

	Registers	LUTs
Base core	2364 (1.00×)	3722 (1.00×)
Base core + latency-opt. class-1	2585 (1.09×)	4950 (1.33×)
Base core + latency-opt. class-1+2	2713 (1.15×)	5242 (1.41×)
Base core + area-opt. class-1	2363 (1.00×)	3710 (1.00×)
Base core + area-opt. class-1+2	2364 (1.00×)	3877 (1.04×)

   - latency:

	`[sec.]and`	`[sec.]andi`	`[sec.]or`	`[sec.]ori`	`[sec.]xor`	`[sec.]xori`	`[sec.]slli`	`[sec.]srli`	`[sec.]add`	`[sec.]sub`
Base core	1	1	1	1	1	1	1	1	1	1
Base core + latency-opt. class-1	1	1	1	1	1	1	1	1	1	1
Base core + area-opt. class-1	2	2	2	2	2	2	2	2	2	2

	`[sec.]lw`	`[sec.]sw`	`[sec.]lbu`	`[sec.]sb`
Base core	2	2	2	2
Base core + latency-opt. class-2	2	2	2	2
Base core + area-opt. class-2	6	4	6	4

https://www.newae.com/products/nae-cw305

Conclusions

- Take away points:
  - we pitch eLIMInate more as an exploration than a solution per se,
  - a high-level summary would be

$$\text{latency-optimised} \quad \rightsquigarrow \quad \begin{cases} \text{less} & \text{latency} & \text{overhead} \\ \text{more} & \text{area} & \text{overhead} \\ \text{more} & \text{usability} & \text{overhead} \\ \text{more} & \text{robust} & \end{cases}$$

$$\text{area-optimised} \quad \rightsquigarrow \quad \begin{cases} \text{more} & \text{latency} & \text{overhead} \\ \text{less} & \text{area} & \text{overhead} \\ \text{less} & \text{usability} & \text{overhead} \\ \text{less} & \text{robust} & \end{cases}$$

  with robustness relating to class-2, so *extra*-core resources.

Conclusions

- Take away points:

- in more detail/depth:
  - a more general leakage elimination hint is an attractive idea, but hard to formulate,
  - *extra*-core resources, e.g., memory are a challenge; this hints at problem wrt. interface,
  - usability is often overlooked: it's hard to quantify the impact this has,
  - underlying ethos differs somewhat: given the option of

    1. make efficient, *then* make secure
    2. make secure, *then* make efficient

    the latter may be preferable.

Questions?

References

[1] E. Brier, C. Clavier, and F. Olivier. "Correlation Power Analysis with a Leakage Model". In: *Cryptographic Hardware and Embedded Systems (CHES)*. LNCS 3156. Springer-Verlag, 2004, pp. 16–29 (see p. 15).

[2] A. Beckers et al. "Provable Secure Software Masking in the Real-World". In: *Constructive Side-Channel Analysis and Secure Design (COSADE)*. LNCS 13211. https://doi.org/10.1007/978-3-030-99766-3_10. Springer-Verlag, 2022, pp. 215–235 (see p. 9).

[3] P. Choi et al. "Architectural Supports for Block Ciphers in a RISC CPU Core by Instruction Overloading". In: *IEEE Transactions on Computers* 71.11 (2022). https://doi.org/10.1109/TC.2021.3050515, pp. 2844–2857 (see p. 9).

[4] S. Cui and J. Balasch. "Efficient Software Masking of AES through Instruction Set Extensions". In: *Design, Automation & Test in Europe (DATE)*. https://doi.org/10.23919/DATE56975.2023.10137150. 2023, pp. 1–6 (see p. 9).

[5] E. De Mulder, S. Gummalla, and M. Hutter. "Protecting RISC-V against Side-Channel Attacks". In: *Design Automation Conference (DAC)*. https://doi.org/10.1145/3316781.3323485. 2019, 45:1–45:4 (see pp. 9, 14).

[6] S. Gao et al. "An Instruction Set Extension to Support Software-Based Masking". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 2021.4 (2021). https://doi.org/10.46586/tches.v2021.i4.283-325, pp. 283–325 (see p. 9).

[7] S. Gao et al. "FENL: an ISE to mitigate analogue micro-architectural leakage". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 2020.2 (2020). https://doi.org/10.13154/tches.v2020.i2.73-98, pp. 73–98 (see p. 9).

[8] B. Gigerl et al. "Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs". In: *USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity21/presentation/gigerl. 2021, pp. 1469–1468 (see pp. 6, 7, 15).

[9] H. Gross et al. "Concealing Secrets in Embedded Processors Designs". In: *Smart Card Research and Advanced Applications (CARDIS)*. LNCS 10146. https://doi.org/10.1007/978-3-319-54669-8_6. Springer-Verlag, 2017, pp. 89–104 (see p. 9).

[10] P. Kiaei and P. Schaumont. *Domain-Oriented Masked Instruction Set Architecture for RISC-V*. Cryptology ePrint Archive, Report 2020/465. https://eprint.iacr.org/2020/465. 2020 (see p. 9).

References

[11] F. Lozachmeur and A. Tisserand. "A RISC-V Instruction Set Extension for Flexible Hardware/Software Protection of Cryptosystems Masked at High Orders". In: *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2023 (see p. 9).

[12] B. Marshall and D. Page. *SME: Scalable Masking Extensions*. Cryptology ePrint Archive, Report 2021/1416. https://eprint.iacr.org/2021/1416. 2021 (see p. 9).

[13] K. Papagiannopoulos and N. Veshchikov. "Mind the gap: Towards secure 1st-order masking in software". In: *Constructive Side-Channel Analysis and Secure Design (COSADE)*. LNCS 10348. https://doi.org/10.1007/978-3-319-64647-3_17. Springer-Verlag, 2017, pp. 282–297 (see pp. 2–8).

[14] M.A. Shelton et al. "Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers". In: *Network and Distributed System Security Symposium (NDSS)*. https://doi.org/10.14722/ndss.2021.23137. 2021 (see pp. 3–5).

[15] M.A. Shelton et al. "Rosita++: Automatic Higher-Order Leakage Elimination from Cryptographic Code". In: *Computer and Communications Security (CCS)*. https://doi.org/10.1145/3460120.3485380. 2021, pp. 685–699 (see pp. 3–5).

[16] K. Stangherlin and M. Sachdev. "Design and Implementation of a Secure RISC-V Microprocessor". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.11 (2022). https://doi.org/10.1109/TVLSI.2022.3203307, pp. 1705–1715 (see p. 9).

[17] S. Tillich, M. Kirschbaum, and A. Szekely. "SCA-Resistant Embedded Processors: The next Generation". In: *Annual Computer Security Applications Conference (ACSAC)*. https://doi.org/10.1145/1920261.1920293. 2010, pp. 211–220 (see p. 9).