



Libra

Architectural Support for Principled, Secure and Efficient Balanced Execution on High-End Processors

ACM CCS – Distinguished paper award

Prosecco Seminar – November 25, 2024

Hans Winderix, Marton Bognar, [Lesly-Ann Daniel](#), Frank Piessens



Libra

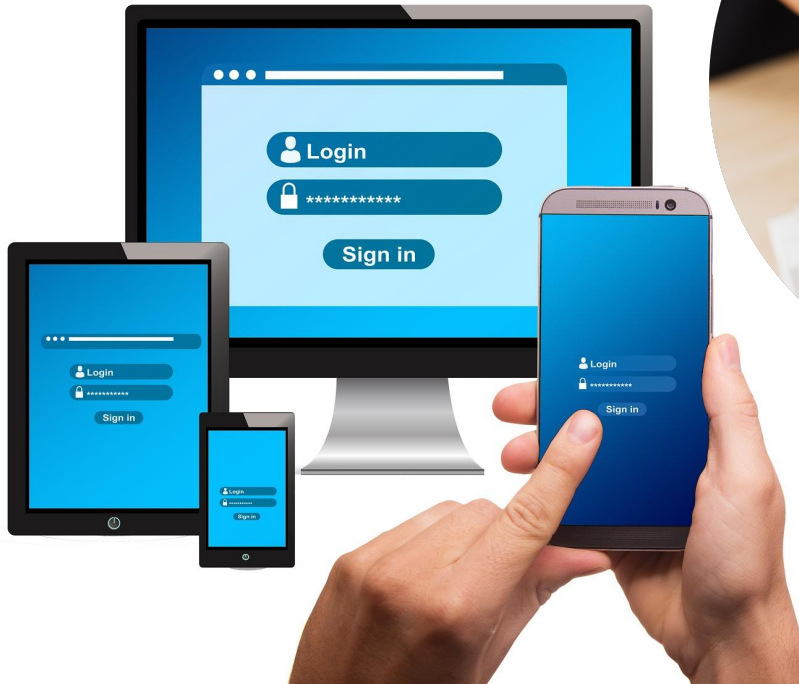
Architectural Support for Principled, Secure and Efficient Balanced Execution on High-End Processors

ACM CCS – Distinguished paper award

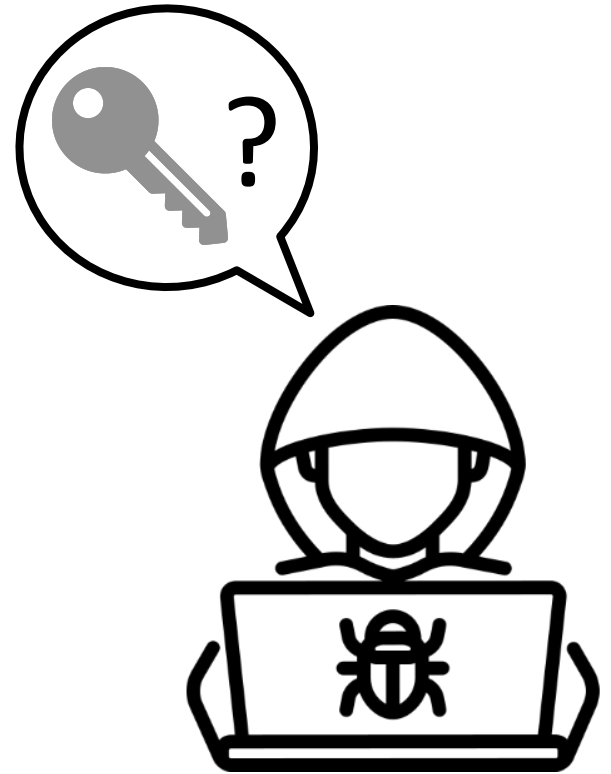
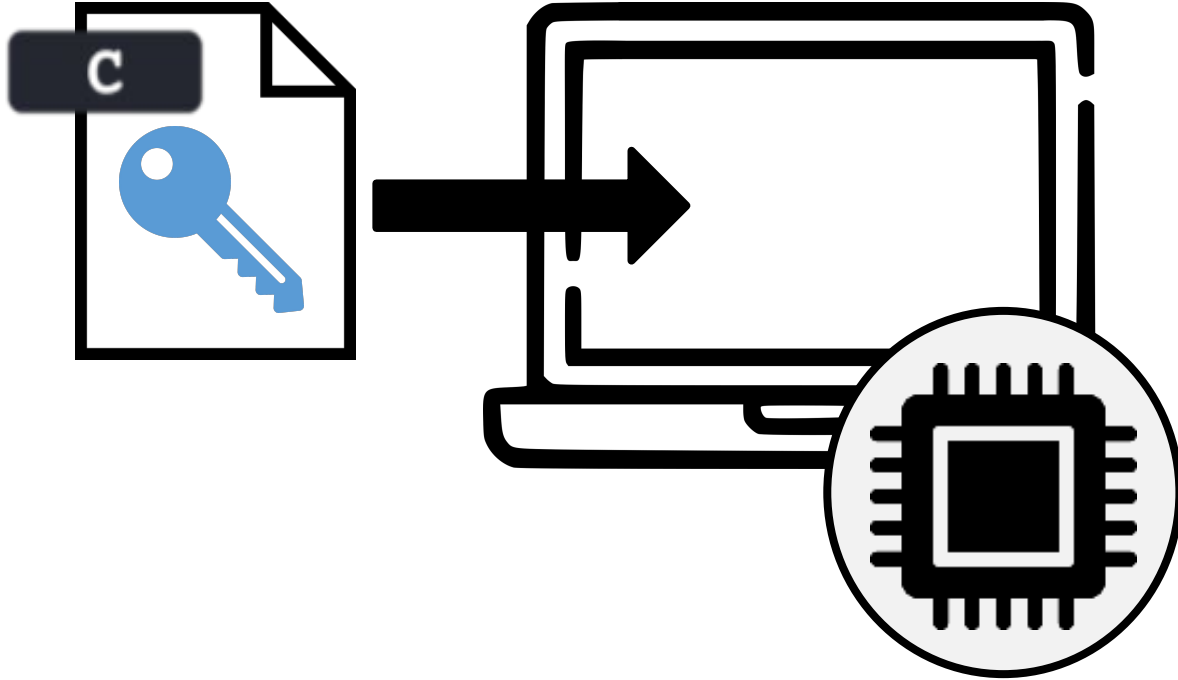
Prosecco Seminar – November 25, 2024

Hans Winderix, Marton Bognar, Lesly-Ann Daniel, Frank Piessens

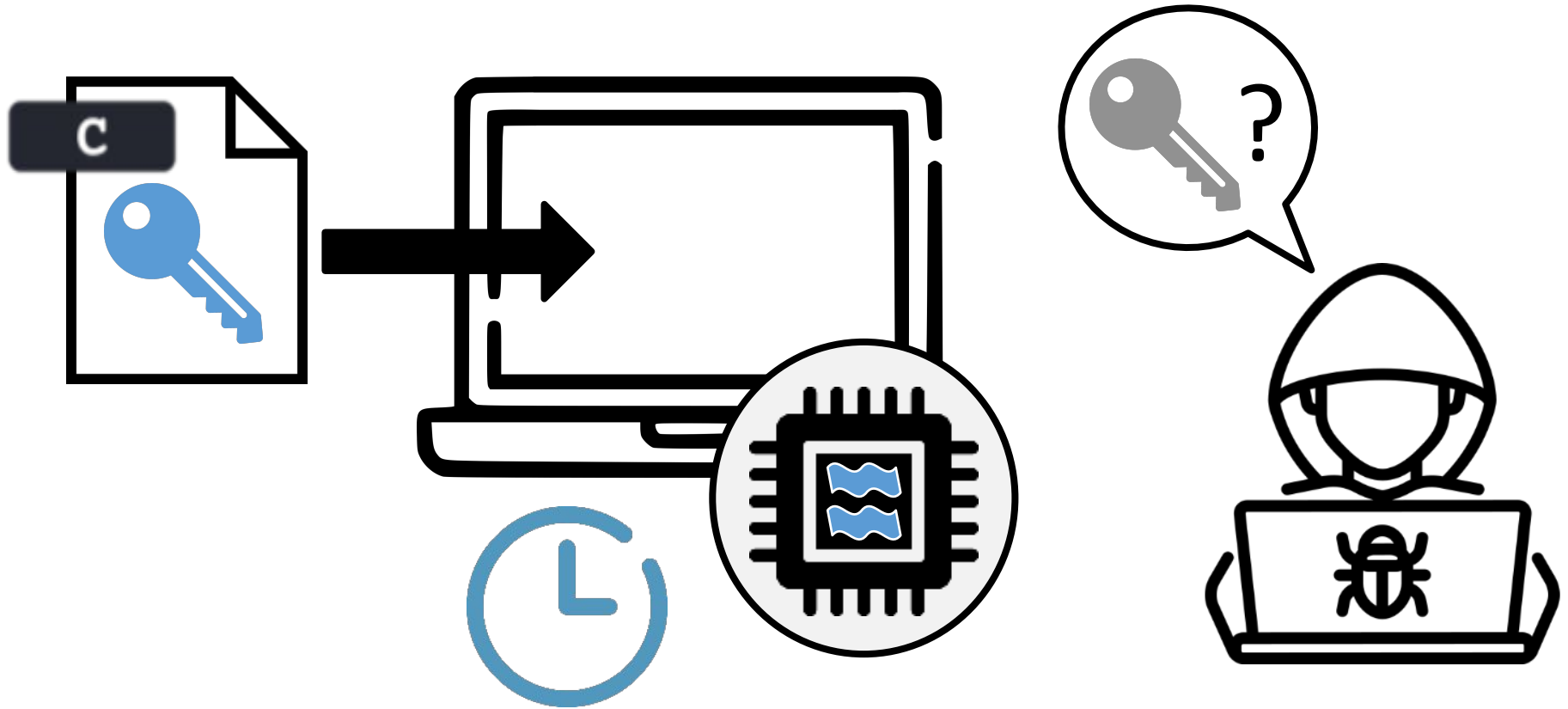
Software process secrets all the time!



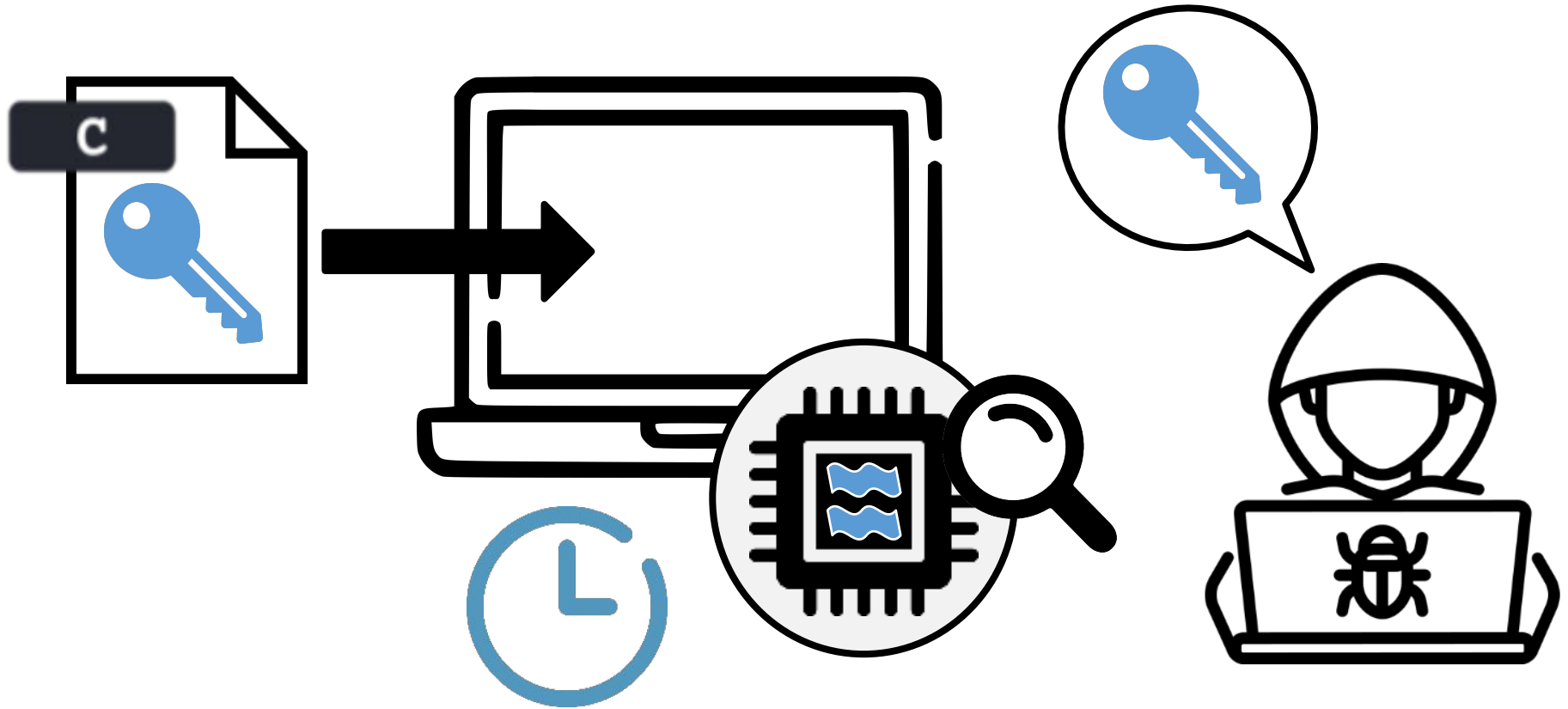
What can go wrong?



What can go wrong?



What can go wrong?



Why does it matter?

Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Paul C. Kocher

Cryptography Research, Inc.
607 Market Street, 5th Floor, San Francisco, CA 94105, USA.
E-mail: paul@cryptography.com.

Abstract. By carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive and often requires only known ciphertext. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements. Techniques for preventing the attack for RSA and Diffie-Hellman are presented. Some cryptosystems will need to be revised to protect against the attack, and new protocols and algorithms may need to incorporate measures to prevent timing attacks.

1996

Cache-timing attacks on AES

Daniel J. Bernstein *

Department of Mathematics, Statistics, and Computer Science (M/C 249)
The University of Illinois at Chicago
Chicago, IL 60607-7045
djb@cr.yp.to

Abstract. This paper demonstrates complete AES key recovery from known-plaintext timings of a network server on another computer. This attack should be blamed on the AES design, not on the particular AES library used by the server; it is extremely difficult to write constant-time high-speed AES software for common general-purpose computers. This paper discusses several of the obstacles in detail.

2005

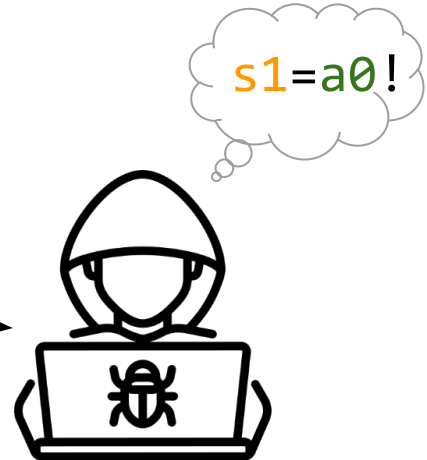
Side-channel leak control flow

```
beq s1 a0 Target  
add a1 a2 a3  
j End  
Target:  
add a2 a3 a4  
End:
```



Side-channel leak control flow

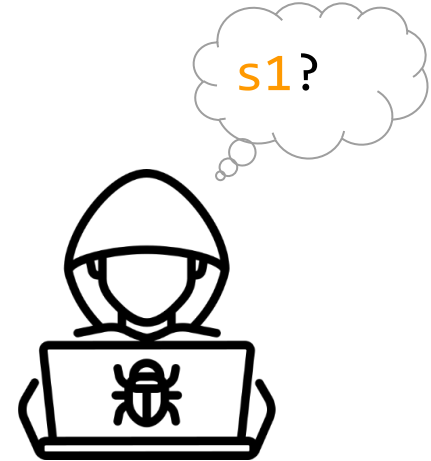
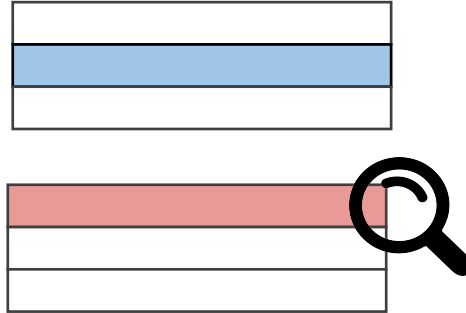
```
beq s1 a0 Target
add a1 a2 a3
j End
Target:
add a2 a3 a4
End:
```



Side-channel leak control flow

```
beq s1 a0 Target
load x a1
j End
Target:
load x a2
End:
```

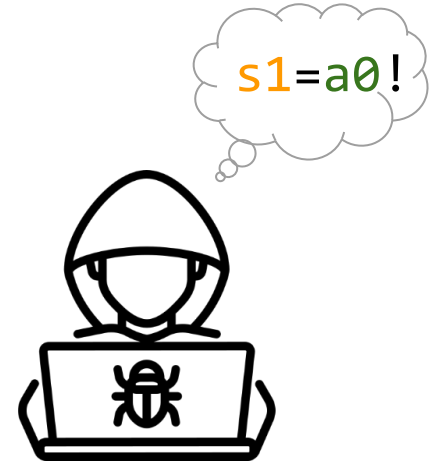
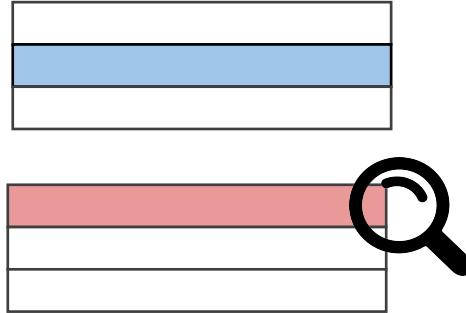
Cache



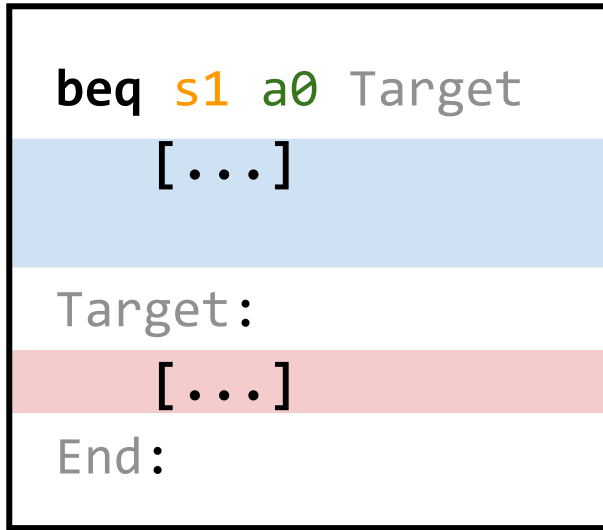
Side-channel leak control flow

```
beq s1 a0 Target
load x a1
j End
Target:
load x a2
End:
```

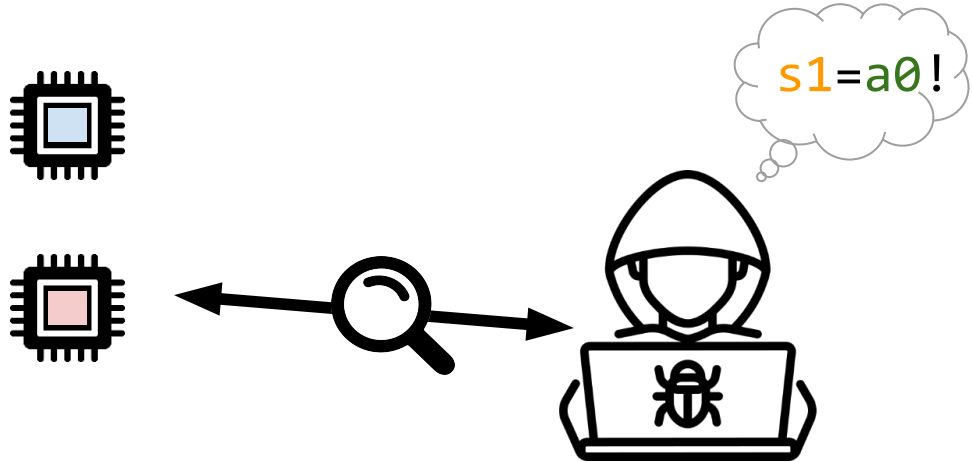
Cache



Side-channel leak control flow



And many more!

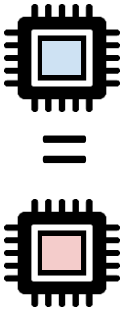
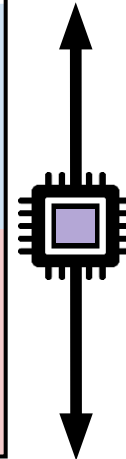


State of the art software countermeasures

Vuln. code
<code>beq s1 a0 Target</code>
<code> add a1 a2 a3</code>
<code> j End</code>
Target:
<code> add a2 a3 a4</code>
End:

Linearization [1]
<code>sub t0 s1 a0</code>
<code>setqz t0 t0</code>
<code>addi t0 t0 -1 ;tt mask</code>
<code>not t1 t0 ;ff mask</code>
<code>and t2 a1 t0</code>
<code>add a1 a2 a3</code>
<code>and a1 a1 t1</code>
<code>or a1 a1 t2</code>
<code>and t2 a2 t1</code>
<code>add a2 a3 a4</code>
<code>and a2 a3 t0</code>
<code>or a2 a2 t2</code>

Balancing
<code>beq s1 a0 Target</code>
<code> add a1 a2 a3</code>
<code> j End</code>
Target:
<code> add a2 a3 a4</code>
<code> j End</code>
End:



Branch balancing, are you kidding me?



“What about branch predictors or instruction caches?”

– Any side-channel expert

“We all know it’s insecure on high-end processors!”

– Any reasonable cryptographer



Antoon Purnal

@PurnalToon



Supreme Court votes 6-3 in favor of branching on secrets in cryptographic code

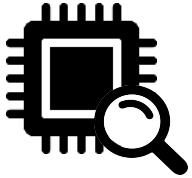
8:01 p.m. · 30 jun. 2022

Branch balancing, are you kidding me?



“But actually why not?”
– Hopeful dreamer

Research questions



What **microarchitectural features** leak control-flow?

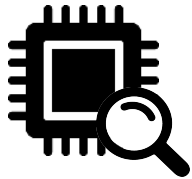


How to **securely balance branches** on high-end CPUs?



Can it improve **performance** over linearization?

Contributions



What **microarchitectural features** leak control-flow?

→ **Characterization** of HW sources of **control-flow leakage**



How to **securely balance branches** on high-end CPUs?

→ **Libra**: Architectural support for balanced execution

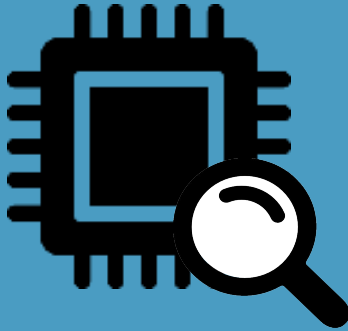


Can it improve **performance** over linearization?

→ HW **implementation** & evaluation (19.3% less overhead)

Characterization

HW sources of control-flow leakage



Literature review

65 attack papers

29 optimizations

Balanceable leakage

Independent of pc

- instruction latency
- data cache
- data TLB
- loads/store buffer dep.
- data dependencies
- ...

→ can be handled in SW 😊
→ but not in a principled way 😞

Unbalanceable leakage

Dependent of pc

- instruction cache
- instruction TLB
- instruction prefetcher
- branch predictors
- μ -op caches
- ...

→ cannot be handled in SW 😞

Balanceable leakage

Independent of pc

Unbalanceable leakage

Dependent of pc

**Disable optims. producing unbalanceable leakage?
Give up on balancing?**

- loads/store buffer dep.
- data dependencies
- ...

- branch predictors
- μ -op caches
- ...

→ can be handled in SW 😊
→ but not in a principled way 😞

→ cannot be handled in SW 😞

Balanceable leakage

Independent of pc

Unbalanceable leakage

Dependent of pc

**Disable optims. producing unbalanceable leakage?
Give up on balancing?**

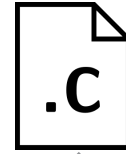
- loads/store buffer dep.

- branch predictors

**No! We handle unbalanceable leakage
with new HW/SW co-design!**

→ can be handled in SW 😊
→ but not in a principled way 😞

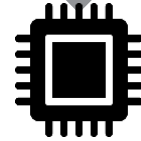
Libra: a new HW/SW co-design for balancing



SW handles **balanceable** leakage



HW/SW Contract for balanced execution



HW support to address **unbalanceable** leakage efficiently



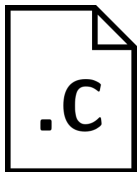
2-D Leakage contract for balanced executions

1. Leakage classes

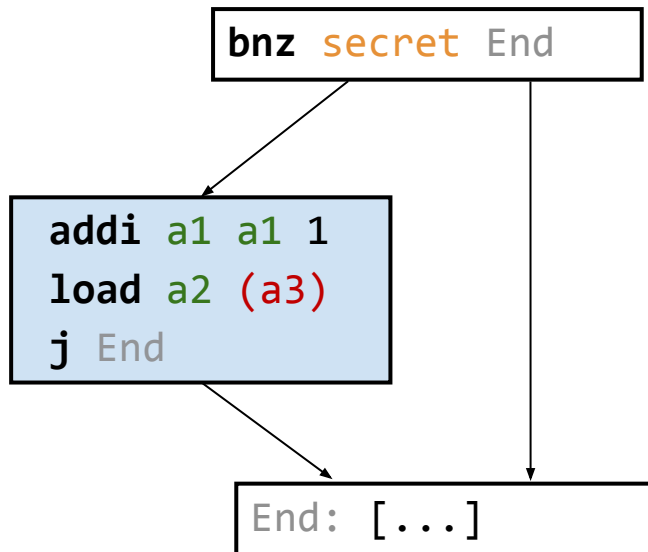
- same observation – `add x1 x1 x2` ~ `sub x1 x1 x2`
- *dummy (no-op)* instruction for each class – `mv x1 x1`

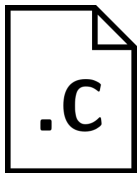
2. Safe/Unsafe instructions

- **Safe**: timing does not depend on operands – `add x1 x1 x2`
- **Unsafe**: timing depends on operands – `load x1 (x2)`

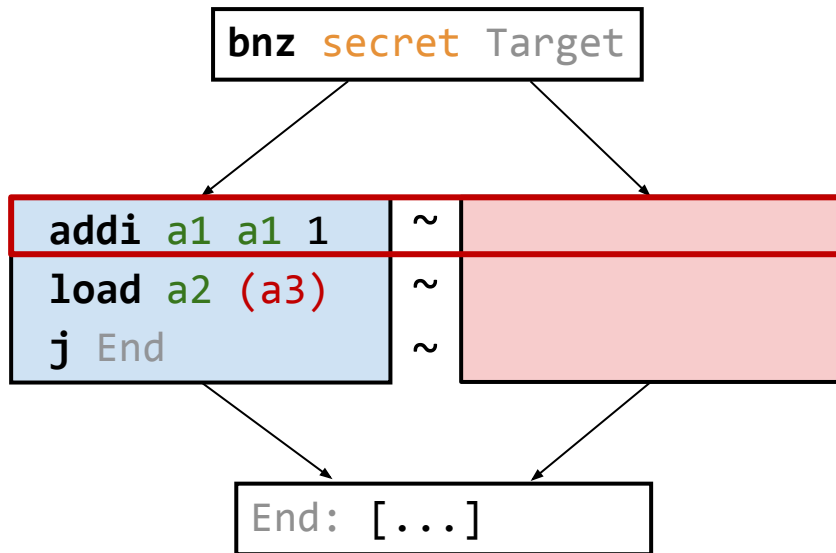


Software balances secret branches w.r.t. contract

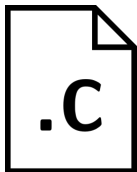




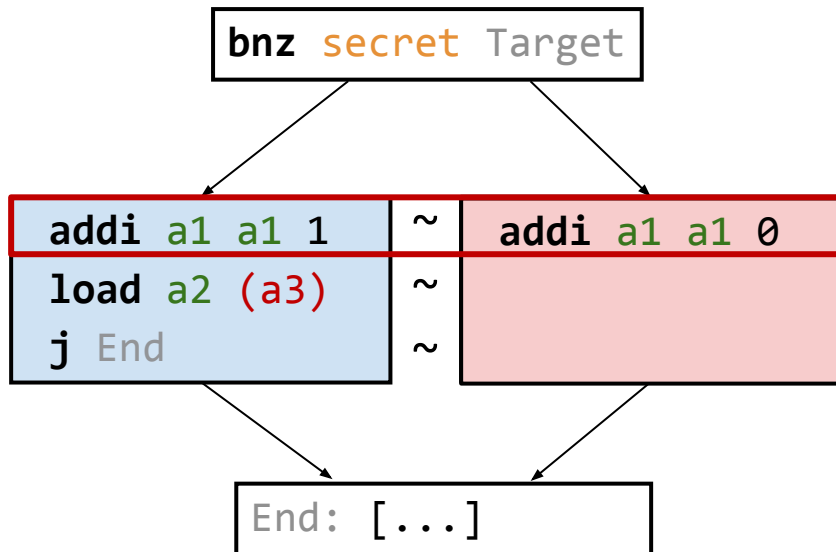
Software balances secret branches w.r.t. contract



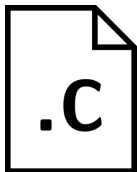
1. Instruction per instruction



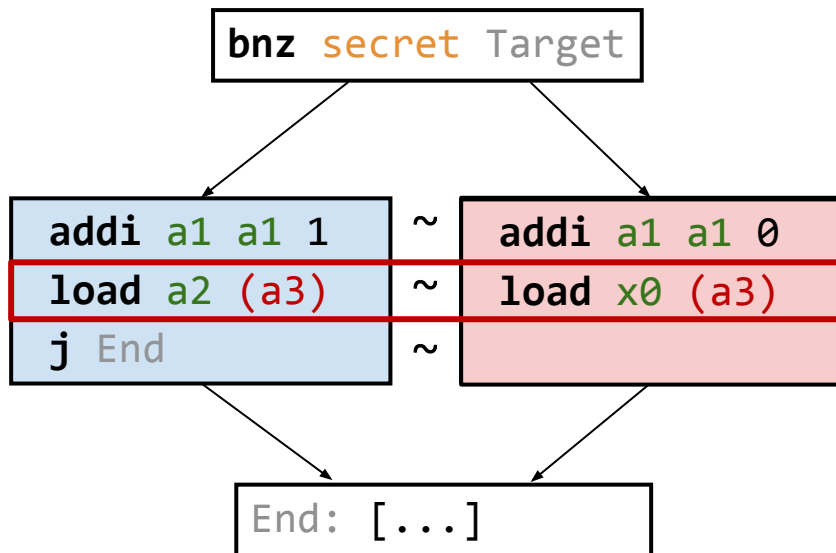
Software balances secret branches w.r.t. contract



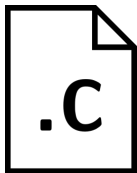
1. Instruction per instruction
2. With dummy instruction in same leakage class



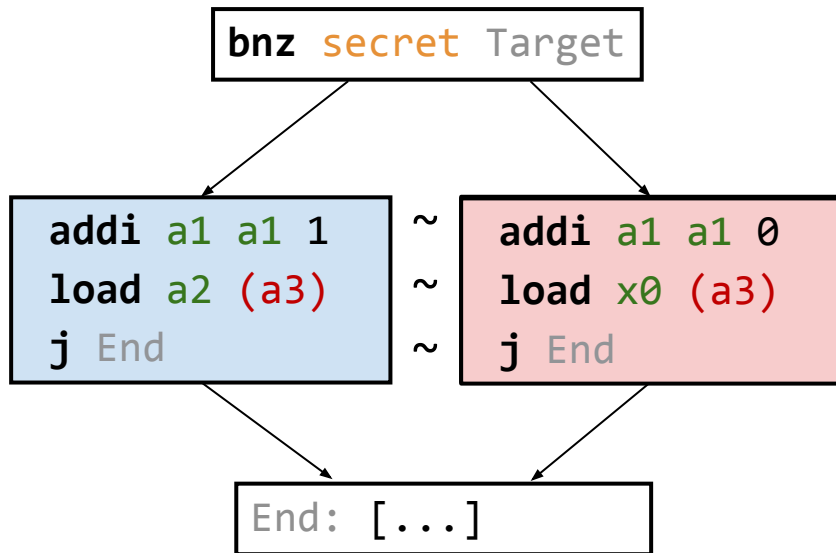
Software balances secret branches w.r.t. contract



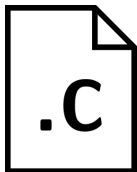
1. Instruction per instruction
2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions



Software balances secret branches w.r.t. contract

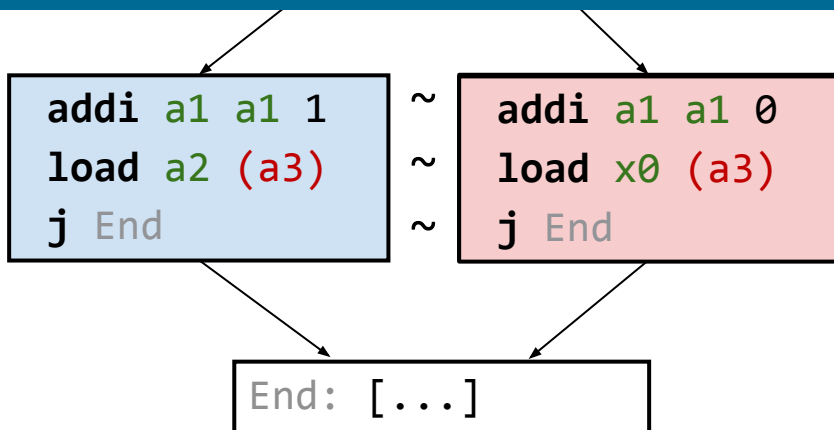


1. Instruction per instruction
2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions

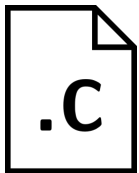


Software balances secret branches w.r.t. contract

Software secure w.r.t. *balanceable* observations



2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions



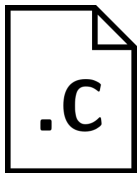
Software balances secret branches w.r.t. contract

Software secure w.r.t. *balanceable* observations

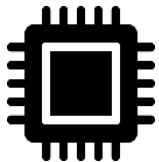
... But still insecure w.r.t. *unbalanceable* observations



I can still see differences
in instruction cache!



+



Folding transformation

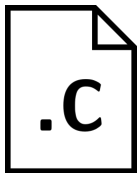
Key Idea: *interleave* secret-dependent branches

```
bnz secret Target
  addi a1 a1 1
  load a2 (a3)
  j End
Target:
  addi a1 a1 0
  load x0 (a3)
  j End
End:
```

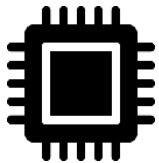


```
  add a1 a1 1
  add a1 a1 0
  load a2 (a3)
  load x0 (a3)
  j End
  j End
```

slice



+



Folding transformation

ISA extension to inform CPU:

- secret region so adapt behavior
- how to navigate folded region

```

bnz secret Target
  addi a1 a1 1
  load a2 (a3)
  j End
Target:
  addi a1 a1 0
  load x0 (a3)
  j End
End:

```



```

lo.bnz secret offT:1 offF:0 #bb:2
  add a1 a1 1 ;pc+2
  add a1 a1 0 ;pc+2
  load a2 (a3) ;pc+2
  load x0 (a3) ;pc+2
lo.beq x0 offT:0 offF:0 #bb:1
lo.beq x0 offT:0 offF:0 #bb:1

```

More in the paper

Advanced features

- Nested secret-dependent regions
- Function calls
 - **lo.call** + fold with dummy function
 - Save/Restore Libra context

And formally?

ISA **semantics** of Libra

Parameterized by **leakage model**

- **weak** observer: obs^- // balanceable observations
- **strong** observer: obs^+ // obs^- + unbalanceable obs.

$$\sigma \xrightarrow{0} \sigma'$$

Libra **transformation** + correctness & security criteria (+proof sketches)

PROPOSITION 2 (SECURITY). For any **asm** program P ,

$$obs^- \text{-ONI}(P) \implies obs^+ \text{-ONI}(\mathcal{F}(P))$$

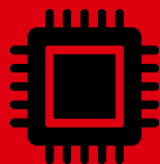
And formally?

ISA semantics of Libra

Parameterized by leakage model

- **weak** observer: obs^- // balanceable observations

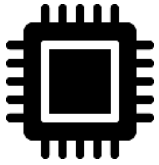
$$\sigma \xRightarrow{0} \sigma'$$



Important requirement: slice-granular leakage

PROPOSITION 2 (SECURITY). For any asm program P ,

$$obs^- \text{-ONI}(P) \implies obs^+ \text{-ONI}(\mathcal{F}(P))$$



Hardware guarantees slice-granular leakage?

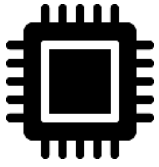


Optimizations producing unbalanceable leakage

5 subcategories

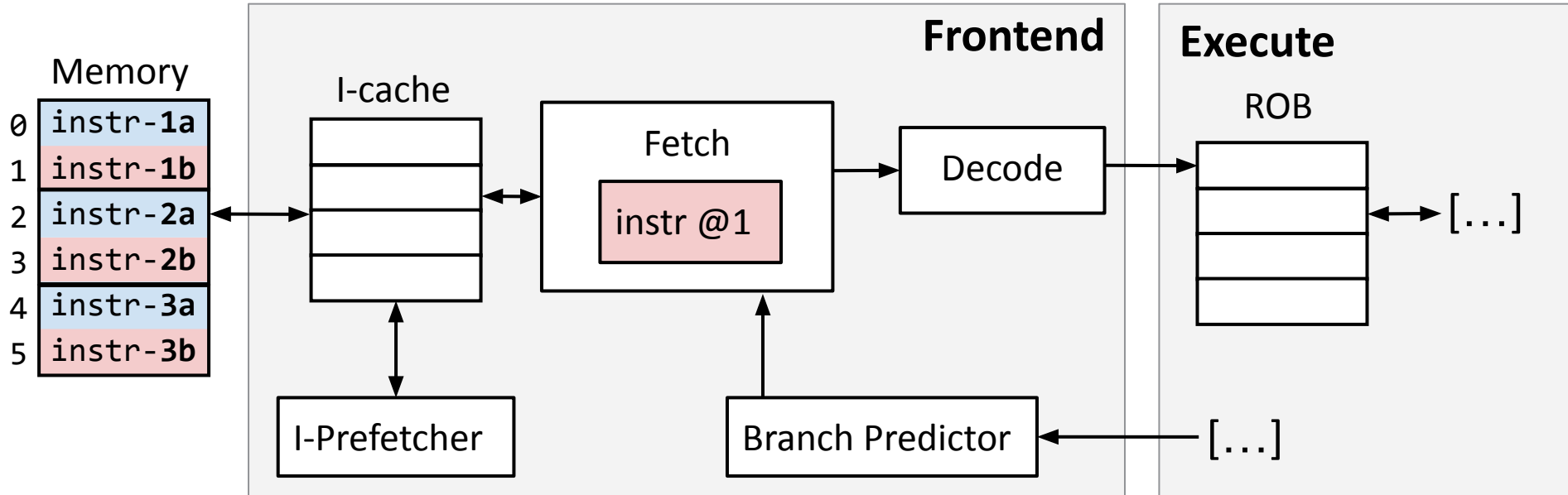
guidelines to adapt for Libra

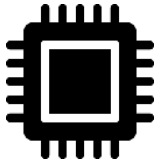




Category: instruction buffering

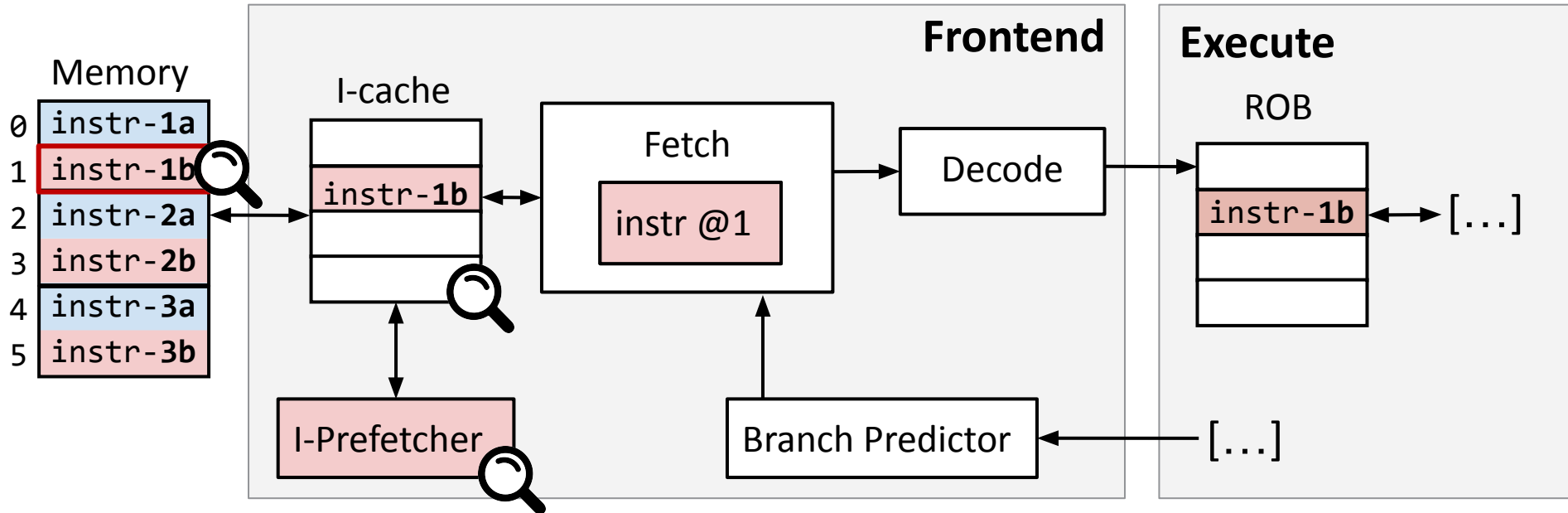
E.g. I-cache, I-prefetcher, MMU, I-TLB, etc.

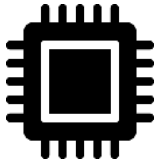




Category: instruction buffering

E.g. I-cache, I-prefetcher, MMU, I-TLB, etc.

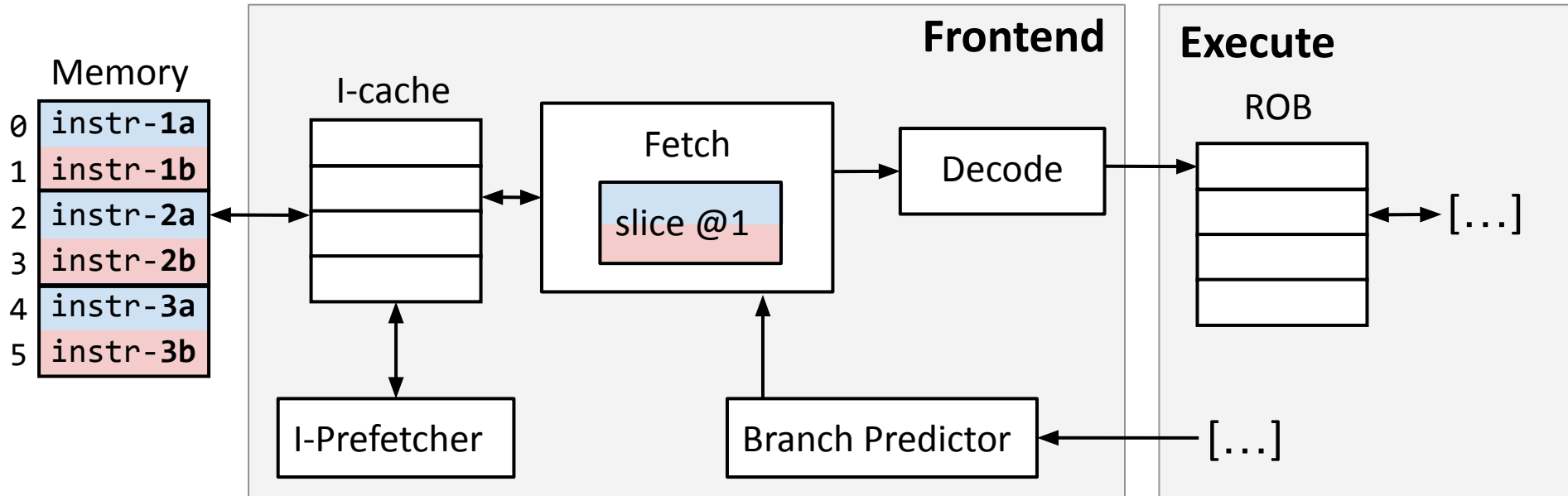


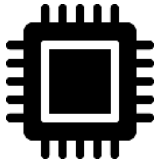


Category: instruction buffering

E.g. I-cache, I-prefetcher, MMU, I-TLB, etc.

Guideline: slice-granular fetch

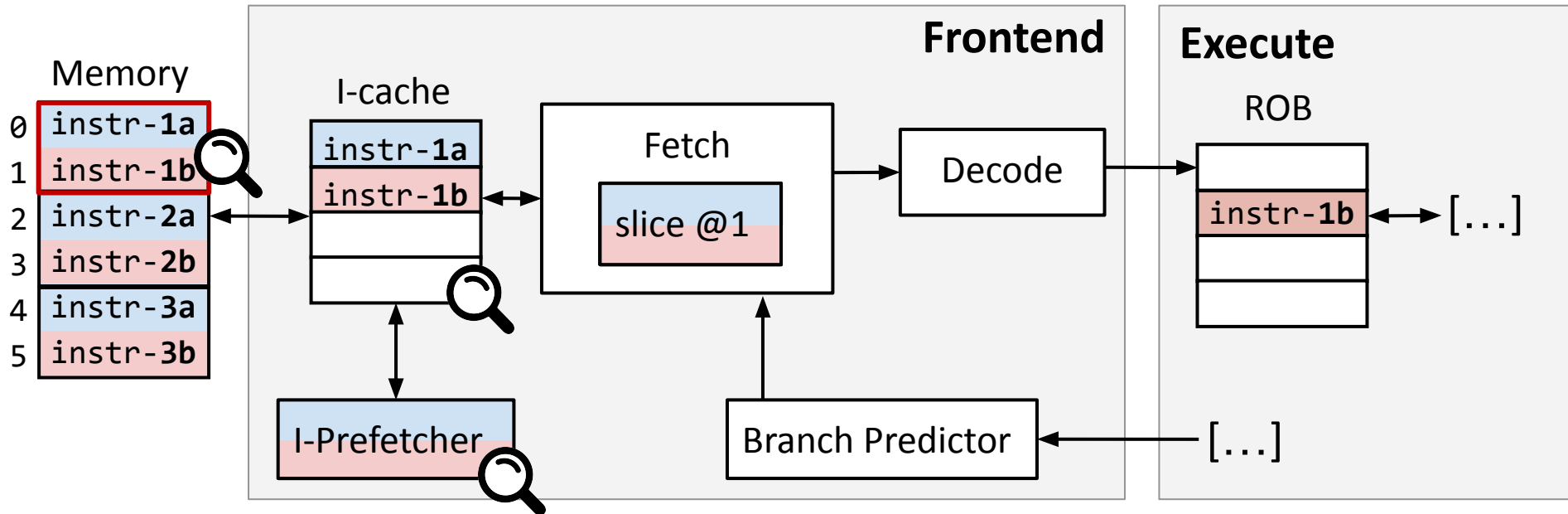


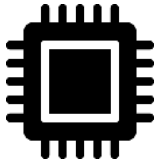


Category: instruction buffering

E.g. I-cache, I-prefetcher, MMU, I-TLB, etc.

Guideline: slice-granular fetch



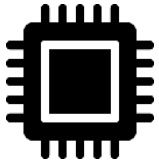


Category: pc-based mappings

E.g. pc-dep prefetcher, branch predictors, etc.

Branch Target Buffer
pc_1 \mapsto target_1
pc_2 \mapsto target_2
pc_n \mapsto target_n

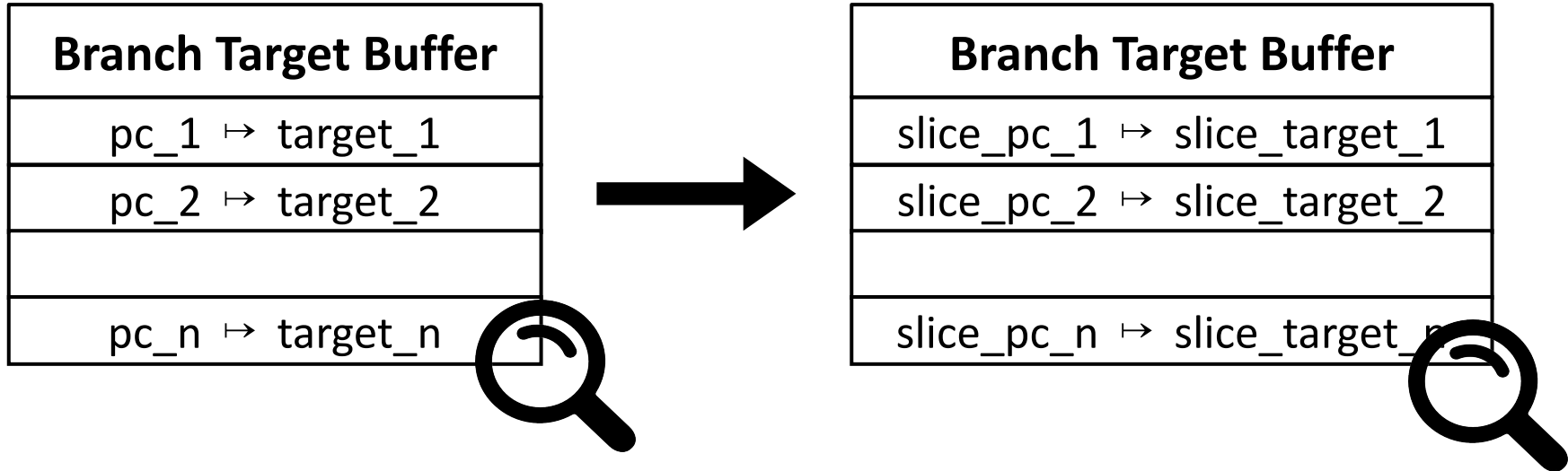


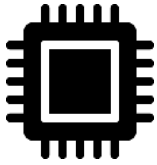


Category: pc-based mappings

E.g. pc-dep prefetcher, branch predictors, etc.

Guideline: slice-based mappings





Other categories



Instruction-specific optimizations

E.g. μ -op cache

In secret region, disable or do operation on whole slice

Inhibit dummy creation

*E.g. *silent-store* opt.*

In secret regions, disable opt. or blacklist stores

Evaluation



Q1. Feasibility

Q2. Security

Q3. Performance

Q4. HW cost

32-bit RISC-V implementation on Proteus



Sources of unbalanceable leakage.

- instruction caches
 - instruction prefetcher
 - branch target predictor
- } Libra-aware fetch unit
- disable in folded regions

Q1. Feasibility



Security evaluation

Benchmark 11 programs [1]

- baseline
- balanced
- linearized
- libra

RTL-level noninterference testing

- Run programs with \neq secret
- Monitor selected signals

Q2. Security 

[1] H. Winderix, J. T. Mühlberg, and F. Piessens, “Compiler-assisted hardening of embedded software against interrupt latency side-channel attacks,” in *EuroS&P*, 2021.

Execution time overhead

	Balanced (insecure)	Linearized (secure)	Libra (secure)
Min	+0%	+8%	-2%
Max	+282%	+225%	+227%
Mean	+42%	+56%	+45%

Compared to linearization
-19.3% overhead

Q3. Performance 

Hardware Cost (FPGA)

	Base	Libra	Increase
LUT	16.5k	18.4k	+11%
Registers	13.6k	14.9k	+9.5%
Critical path	37.4ns	37.4ns	+0%

Small area increase
No impact on CP

Q4. HW cost



Dream of balanced executions come true!



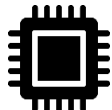
Libra: new HW/SW co-design for balancing



HW/SW Contract balancing



Balance + Fold secret branches



Slice-granular leakage



Keep HW **optimizations**



github.com/proteus-core/libra

A new era for balancing?

Well, there are still challenges!

- HW verif/synthesis for balancing contracts
- Automatic balancing transformation
- Evaluation on larger benchmarks
- Feasibility with more complex optimizations?



Exploring HW-SW Co-Designs

Let's take a dive



Proteus: An Extensible RISC-V Core for Hardware Extensions

(RISC-V Summit '23)

Marton Bogнар, Job Noorman, Frank Piessens



A modular textbook processor to study HW extensions

- **In/Out-of order** pipelines
- **Optimizations:** branch predictors, cache, prefetchers, ...
- **Configurable:** #exec units, ROB size, ...
- **Extensible:** plugin system
- **SpinalHDL** verilog FPGA / simulator
- **Validate:** HW fuzzing (in progress)



ProSpeCT: Provably Secure Speculation for the Constant-Time Policy

(USENIX'23)

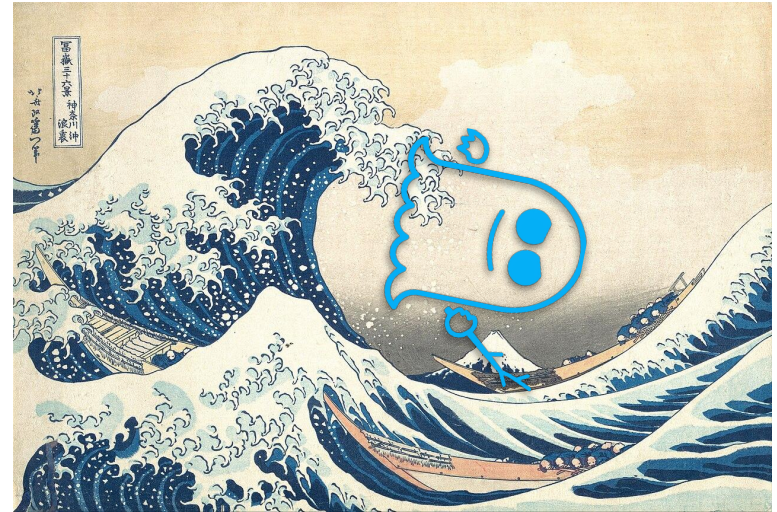
Lesly-Ann Daniel, Marton Bogнар, Job Noorman, Sébastien Bardin, Tamara Rezk, Frank Piessens



SW: annotate secrets

HW: no speculation on secrets

- **CT code secure against Spectre**
- Without sacrificing speculation
- Design proven secure w.r.t. contract
- Holds for many variants of Spectre



Architectural Mimicry: Innovative Instructions to Efficiently Address Control-Flow Leakage in Data-Oblivious Programs

(SP'24)

Hans Winderix, Marton Bogнар, Job Noorman, Lesly-Ann Daniel, Frank Piessens



HW: Mimic execution (imitate μ arch behavior)

SW: ISA extension to control it

- Principled control-flow hardening
- Support for linearization / balancing
- **Accelerate linearized code**



Challenges?

Keep up with
rapidly evolving
threat landscape





Need to adapt side-channel defenses

- Emerging threats (new attacks, new processor optimis., ...)
- Emerging applications (machine learning models, ...)
- New platforms (TEEs, HW accelerators, ...)





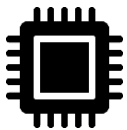
Improve SW integration

- **Compiler support**

- Should be **parametric** in leakage contract to support many targets
- Needed for **adoption** & better **evaluation**
- Challenge: compilers are not security-aware nor really modular

- **Binary analysis**

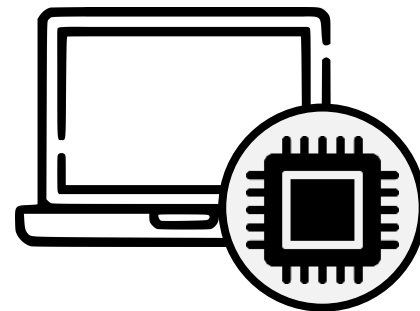
- Should be **parametric** in leakage contract to support many targets
- Designed with **microarchitectural-security** in mind



Validate HW implementations



**Software Security
Property
(e.g. CT/SCT)**



**Actual
Microarchitectural
Leakage**

What I'd like to explore

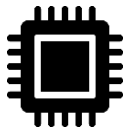


New HW-SW co-designs



New generation of tools for SW side-channel security

- compilers
- binary-analyzers



Hardware verification

