Libra

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

ACM CCS 2024 - October 15th

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

Libra

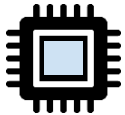
Architectural Support for Principled, Secure and Efficient Balanced Execution on High-End Processors

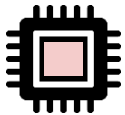
ACM CCS 2024 - October 15th

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

Side-channel leak control flow

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



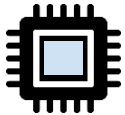


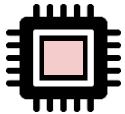
Executions produce *observations*

- End-to-end timing
- Microarchitectural resource usage
  - cache usage
  - port contention
  - etc.

Side-channel leak control flow

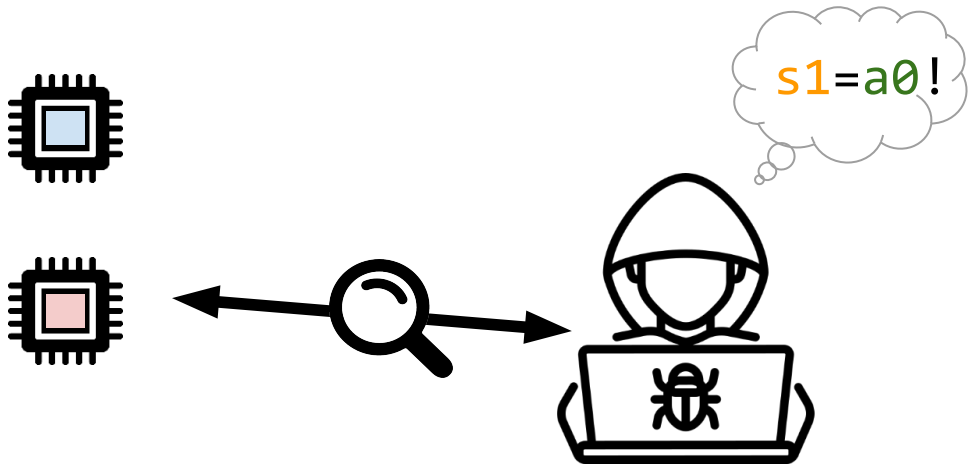
```
beq s1 a0 Target
    add a1 a2 a3
    j End
Target:
    add a2 a3 a4
End:
```





Side-channel leak control flow

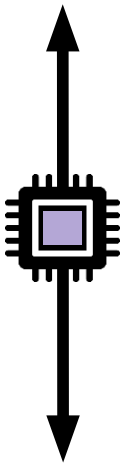
```
beq s1 a0 Target
    add a1 a2 a3
    j End
Target:
    add a2 a3 a4
End:
```



State of the art software countermeasures

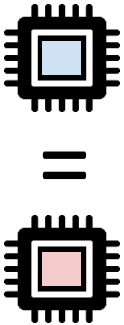
Linearization (Molnar [1])

```
sub t0 s1 a0
setqz t0 t0
addi t0 t0 -1 ;true mask
not t1 t0     ;false mask
and t2 a1 t0
add a1 a2 a3
and a1 a1 t1
or  a1 a1 t2
and t2 a2 t1
add a2 a3 a4
and a2 a3 t0
or  a2 a2 t2
```



Balancing

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

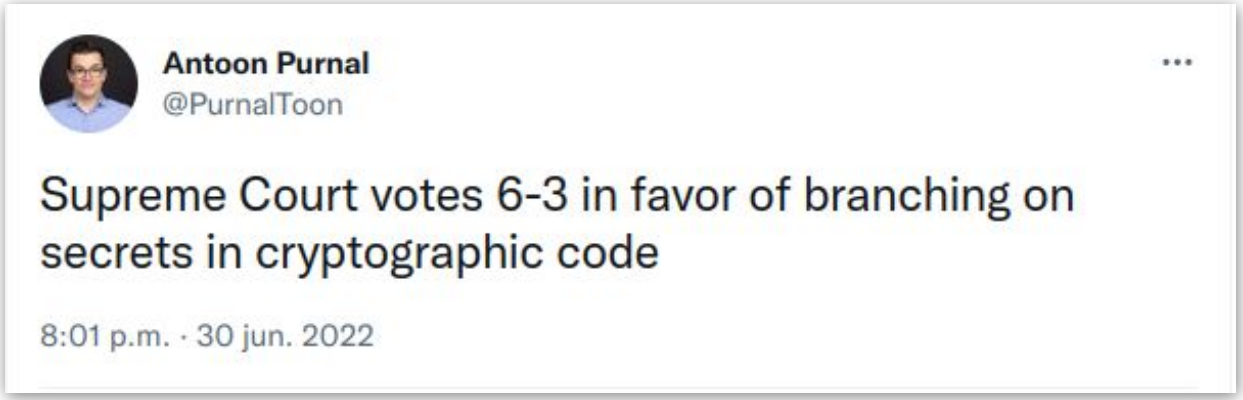


[1] Molnar et al., The program counter security model: Automatic detection and removal of control-flow side channel attacks (ICISC 2005)

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

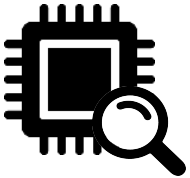


Branch balancing, are you kidding me?

“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**

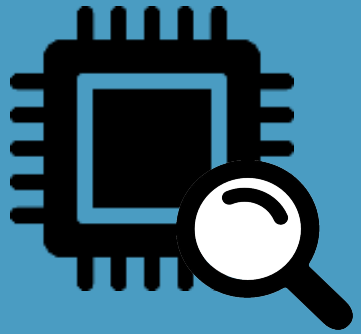
How to **securely balance branches** on high-end CPUs?

➔ **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

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

➔ can be handled in SW ☺️

➔ but not in a principled way ☹️

Unbalanceable leakage

Dependent of pc

- branch predictors
- µ-op caches
- …

➔ cannot be handled in SW ☹️

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

Balanceable leakage

Independent of pc

Unbalanceable leakage

Dependent of pc

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

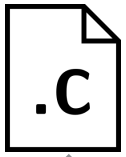
- loads/store buffer dep.
- branch predictors

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

➔ can be handled in SW 😊

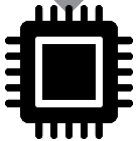
➔ but not in a principled way ☹

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



SW handles **balanceable** leakage

HW/SW Contract for balanced execution

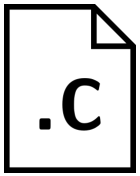


HW support to address **unbalanceable** leakage **efficiently**

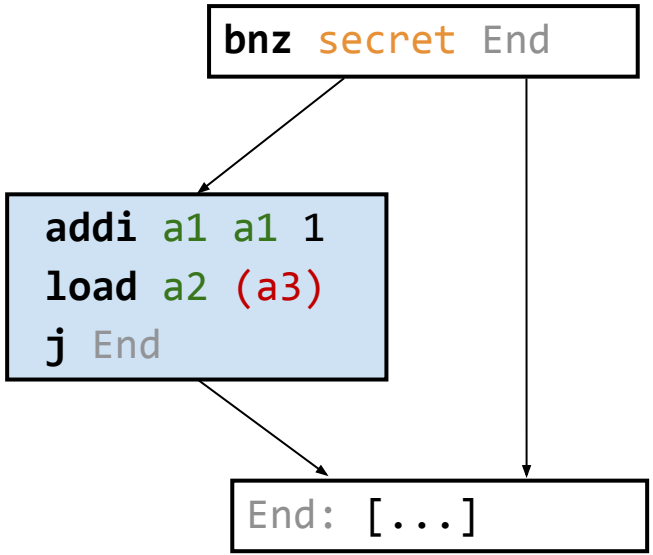
2-D Leakage contract for balanced executions

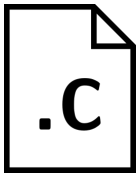
1. **Leakage classes**
   - same observation – `add x1 x1 x2` ~ `sub x1 x1 x2`
   - *dummy (no-op)* instruction for each class – `mv x1 x1`

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

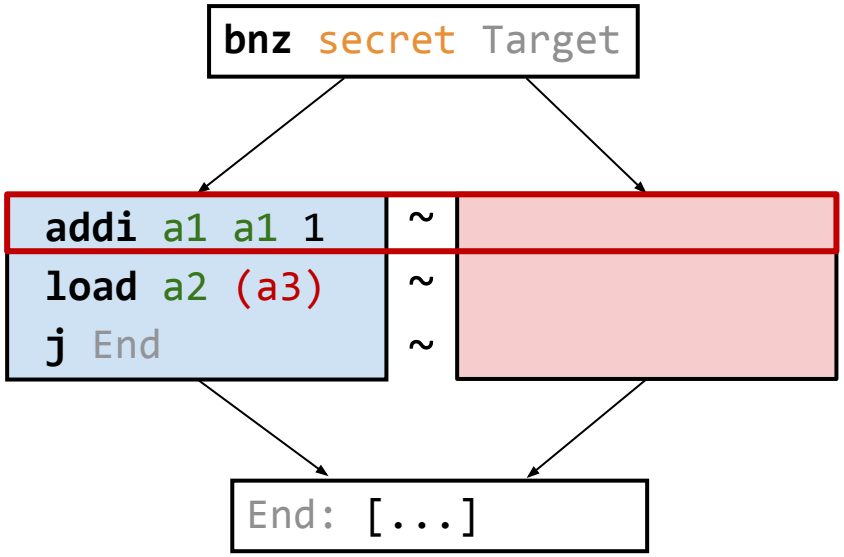


Software balances secret branches w.r.t. contract

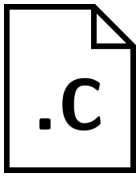




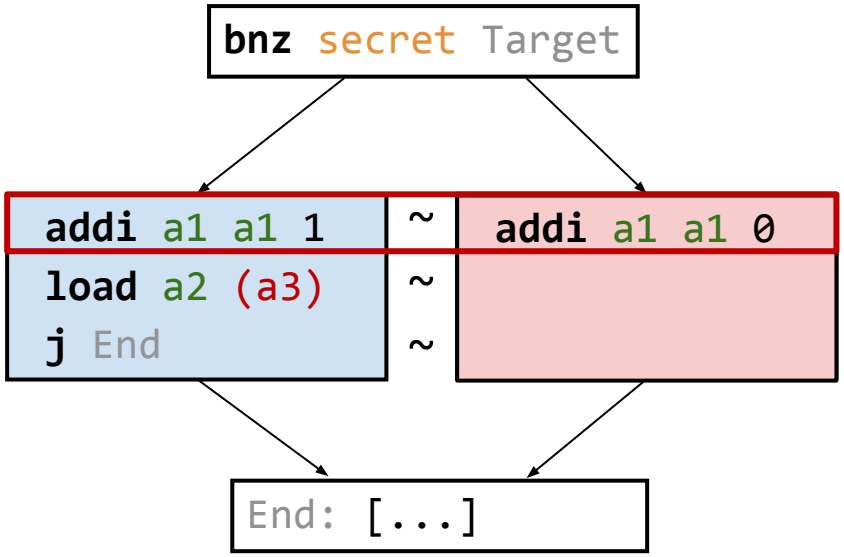
Software balances secret branches w.r.t. contract



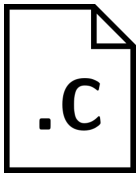
1. Instruction per instruction



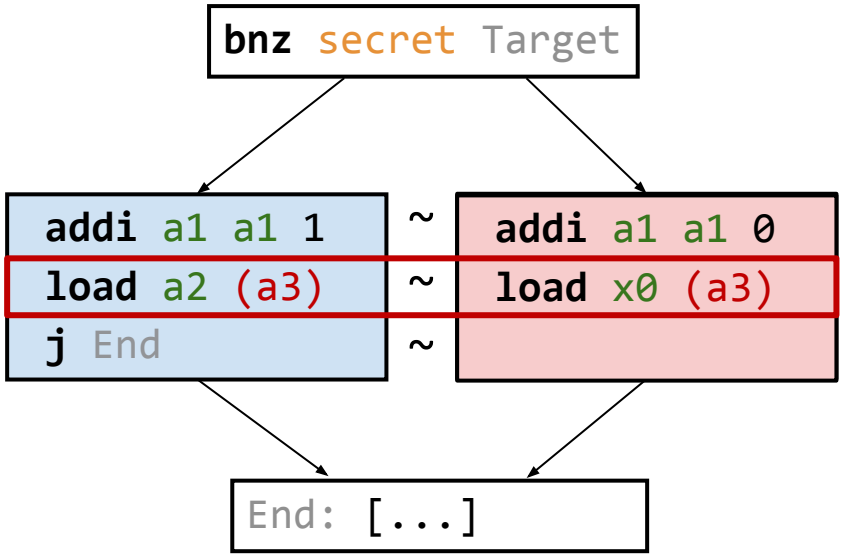
Software balances secret branches w.r.t. contract



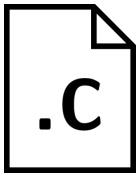
1. Instruction per instruction
2. With dummy instruction in same leakage class



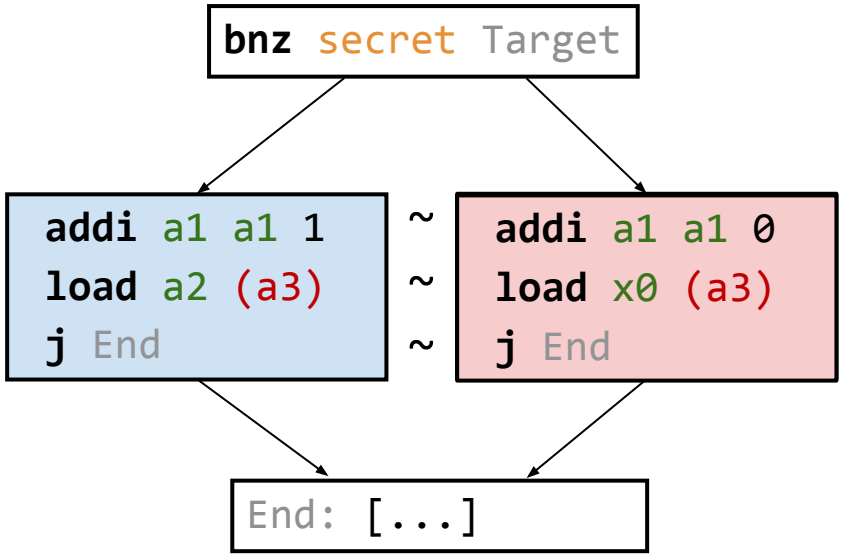
Software balances secret branches w.r.t. contract



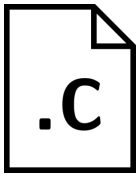
1. Instruction per instruction
2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions



Software balances secret branches w.r.t. contract

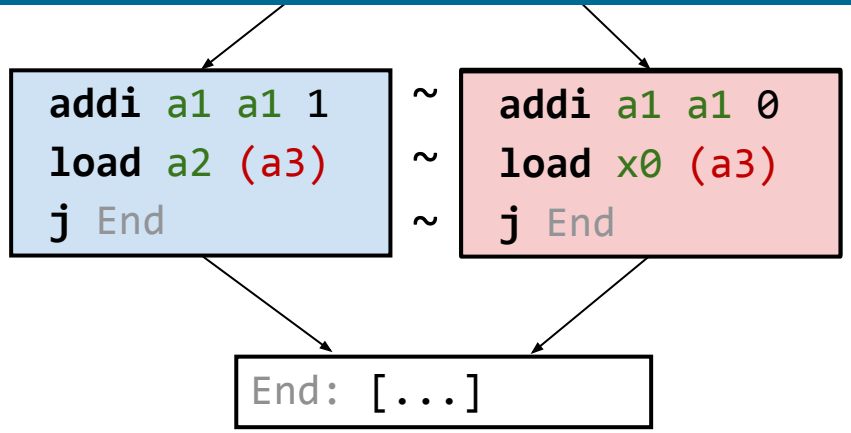


1. Instruction per instruction
2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions

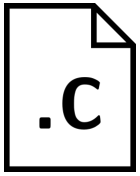


Software balances secret branches w.r.t. contract

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



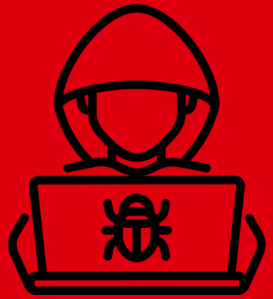
2. With dummy instruction in same leakage class
3. Balance operands of unsafe instructions



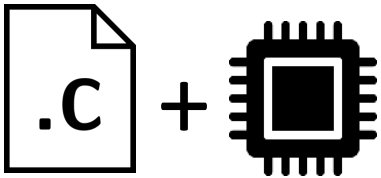
Software balances secret branches w.r.t. contract

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

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



I can still see differences in instruction cache!



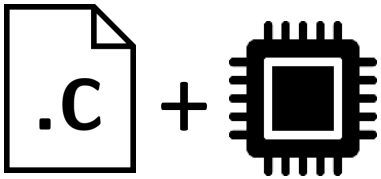
Folding transformation

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



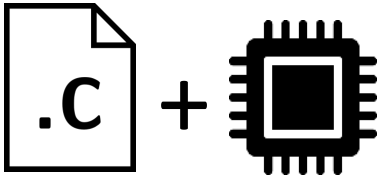
Folding transformation

ISA extension to inform CPU:

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

```
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
```



Folding transformation

ISA extension to inform CPU:

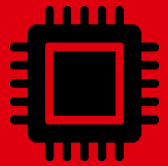
→ how to navigate folded region

→ secret region so adapt behavior

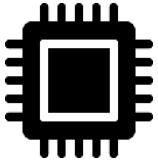
```
bnz secret Target
```

```
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```

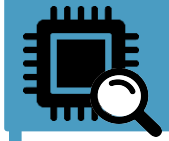
```
    load x0 (a3)                    ;pc+2
    lo.beq x0 offT:0 offF:0 #bb:1
    lo.beq x0 offT:0 offF:0 #bb:1
```



Important requirement: slice-granular leakage



Hardware guarantees slice-granular leakage?



Optimizations producing unbalanceable leakage

- 5 subcategories
  - guidelines to adapt for Libra

Example: Slice-granular fetch-decode

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 ≠ 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

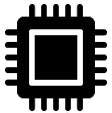
Q4. HW cost ✅

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

Credit icons: Flaticon

Backup

A new era for balancing?

Well, there are still challenges!

- Verif/synthesis for balancing contracts
- Balancing transformation
- Evaluation on larger benchmarks
- Feasibility with more complex optimizations?

Hardware guarantees slice-granular leakage?

pc-dependent buffering

E.g. I-cache, I-prefetcher

Slice-granular fetch

pc-dependent mapping

E.g. pc-dep prefetcher

Slice-based mapping

Unbalanceable leakage

Instr-specific optims.

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

More in the paper

Advanced features

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

Formalization

- ISA semantics
- Security definitions
- Folding transformation
  - Proof of correctness
  - Proof of security