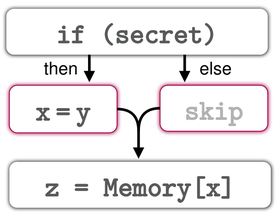
LockOS : Embedded Operating System for Hardware Cache Locking support

Frédéric Besson [1] Pascal Cotret [2] Nicolas Gaudin [2] Guy Gogniat [2]
Jean-Loup Hatchikian-Houdot [1] Guillaume Hiet [3] Pierre Wilke [3]
Vianney Lapôtre [2]
Contact: vianney.lapotre@univ-ubs.fr
[1] EPICURE / IRISA / INRIA, Rennes
[2] Lab-STICC, UBS, Lorient / ENSTA Bretagne, Brest
[3] CIDRE / IRISA / INRIA, CentraleSupélec, Cesson-Sévigné

Context

Timing vulnerabilities caused by behavior depending on a secret

```
x = 0, y = 64
if (secret){
    x = y
}
z = Memory[x]
```



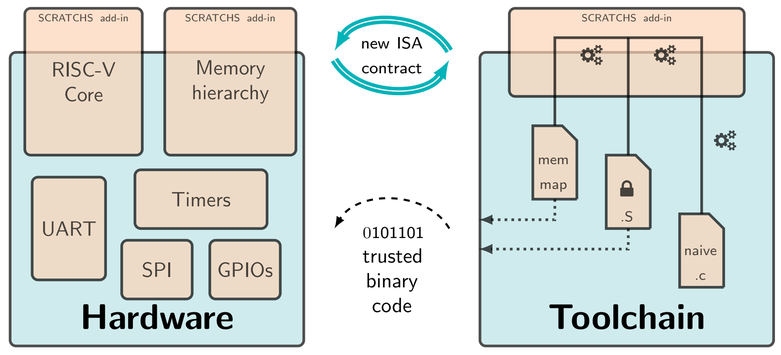
Timing differences :
- in branching
- in memory access

→ Secret exposed

Attacker: observes time ⇒ deduces `secret`

- Behavior duration depends on resource usage (like memory access).
- Timing is observable when resource usage is shared between the victim and the attacker.
- Countermeasures already exist (resources partitioning, Constant-Time programming), but are often costly.

SCRATCHS

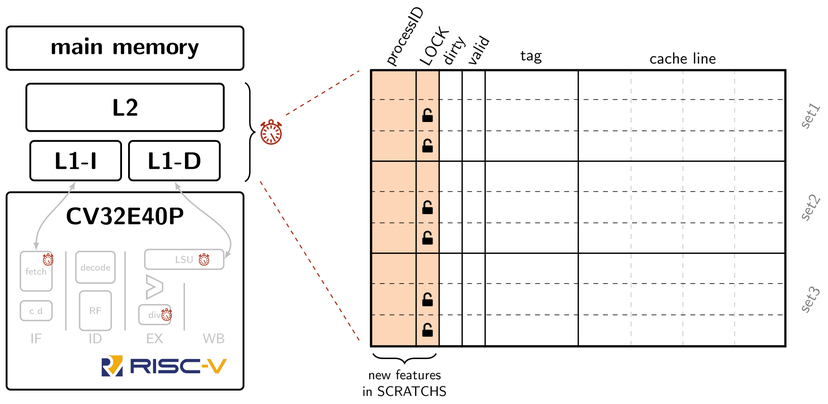


SCRATCHS's goal is to **co-design** a **RISC-V processor** and a **compiler toolchain**:
- Immune sensitive code to **timing side-channel attacks**.
- Minimal overhead on the micro-architecture.
- Considering a small-scale embedded system.

- **Hardware** implements **security mechanisms**.
- **Compiler** produces binaries able to use these mechanisms to be **side-channel resistant**.

Cache lines locking mechanism

Memory hierarchy and some functional units temporal behaviors (e.g. ALU, **LSU**, **division** or branching) can leak **information** ⏱.



We identify three sources of leakage on the CV32E40P RISC-V processor:

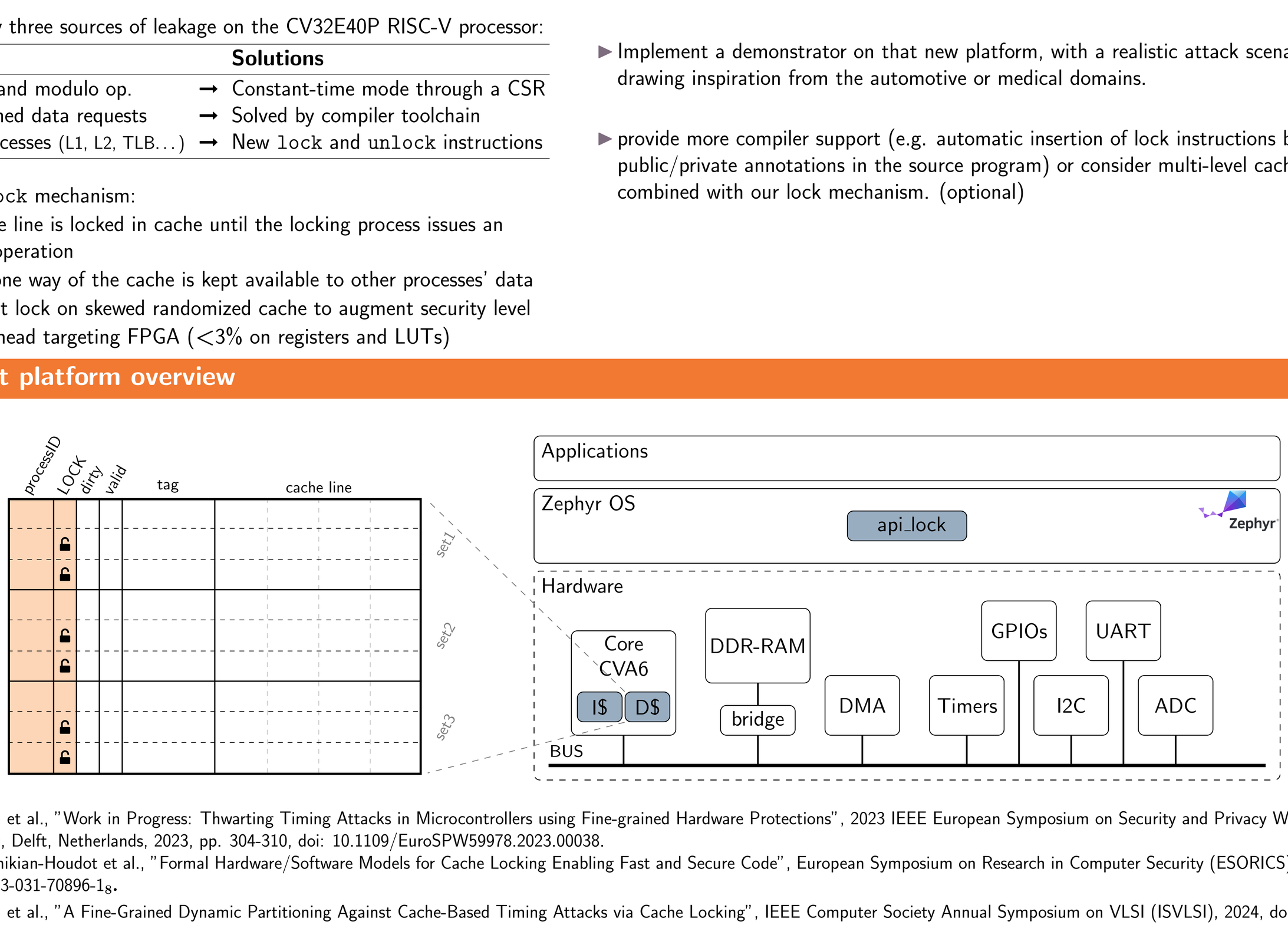
Leak		Solutions
Division and modulo op.	→	Constant-time mode through a CSR
Non-aligned data requests	→	Solved by compiler toolchain
Cache accesses (L1, L2, TLB...)	→	New `lock` and `unlock` instructions

`lock`/`unlock` mechanism:
- The cache line is locked in cache until the locking process issues an `unlock` operation
- At least one way of the cache is kept available to other processes' data
- Implement lock on skewed randomized cache to augment security level
- Low overhead targeting FPGA (<3% on registers and LUTs)

LockOS main Objectives

- Consider a more complex RISC-V processor.
  - additional features to fully support an operating system
  - CVA6 core : This core implements RV32GC or RV64GC extensions with three privilege levels M, S, U. It includes branch prediction, a memory protection unit (MPU) and, optionally, an MMU
- Introduce a small operating system on the platform to
  - configure the MPU
  - manage SCRATCHS Locking mechanism. This includes
    1. handling unsuccessful memory locks
    2. identify processes that requires the use the proposed lock mechanism and grant them permissions to do it
    3. handling killed processes that have locked data in the cache memory
- Implement a demonstrator on that new platform, with a realistic attack scenario, drawing inspiration from the automotive or medical domains.
- provide more compiler support (e.g. automatic insertion of lock instructions based on public/private annotations in the source program) or consider multi-level caches combined with our lock mechanism. (optional)

Project platform overview



[1] N. Gaudin et al., "Work in Progress: Thwarting Timing Attacks in Microcontrollers using Fine-grained Hardware Protections", 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Delft, Netherlands, 2023, pp. 304-310, doi: 10.1109/EuroSPW59978.2023.00038.
[2] J-L. Hatchikian-Houdot et al., "Formal Hardware/Software Models for Cache Locking Enabling Fast and Secure Code", European Symposium on Research in Computer Security (ESORICS), 2024, doi: 10.1007/978-3-031-70896-1$_8$.
[3] N. Gaudin et al., "A Fine-Grained Dynamic Partitioning Against Cache-Based Timing Attacks via Cache Locking", IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2024, doi: 10.1109/ISVLSI61997.2024.00041.