SCRATCHS: Side-Channel Resistant Applications Through Co-designed Hardware/Software

Frédéric Besson [1] Pascal Cotret [2] Nicolas Gaudin [2] Guy Gogniat [2]
Jean-Loup Hatchikian-Houdot [1] Guillaume Hiet [3] Vianney Lapotre [2]
Pierre Wilke [3]

Contact: pierre.wilke@inria.fr

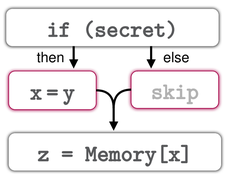
[1] EPICURE / IRISA / INRIA, Rennes
[2] Lab-STICC, UBS, Lorient / ENSTA Bretagne, Brest
[3] CIDRE / IRISA / INRIA, CentraleSupélec, Cesson-Sévigné

https://project.inria.fr/scratchs/

Introduction

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: observe time ⇒ deduce `secret`

- Behavior duration depends on resource usage (like memory access)
- Timing is observable when resource usage is shared between the victim and the attacker.

State of the art

Constant Time Programming in Software

Equivalent data oblivious code (behavior independent of secret)

```
x = 0, y = 64
z = Memory[x]
t = Memory[y]
z = (secret) ? t : z
```

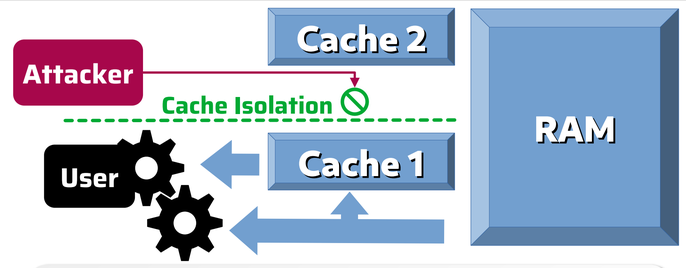
CMOV

- Linear program flow
- Both branches executed

→ No timing variations (that depends on any secret)

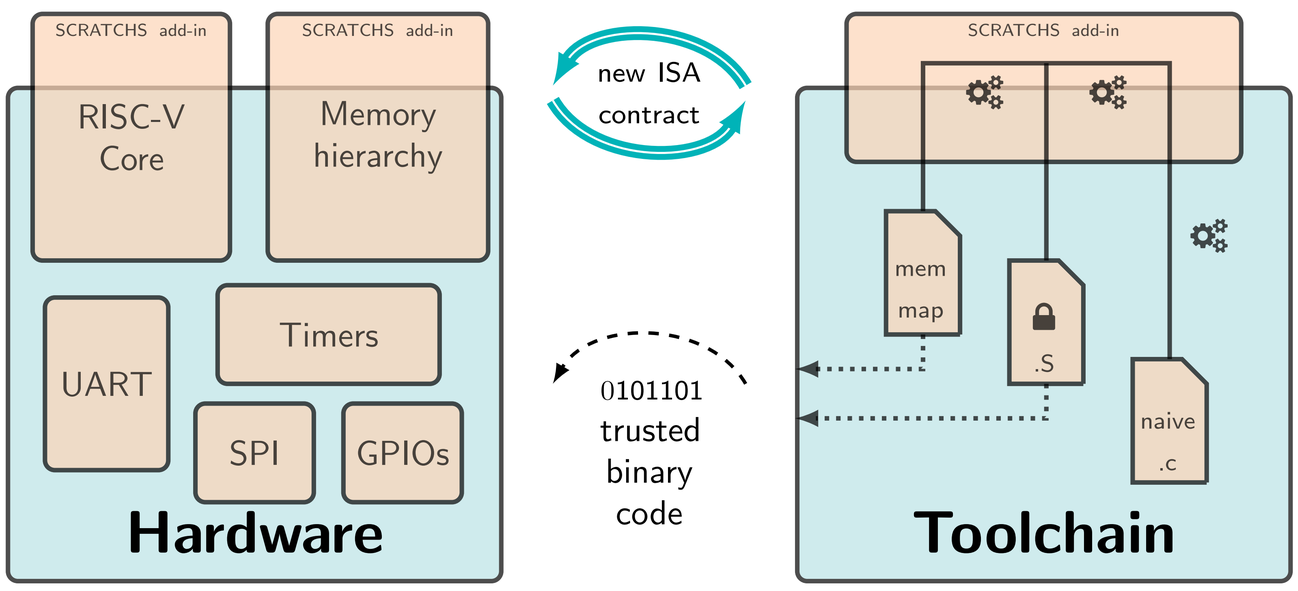
Drawbacks
- Potential micro-architecture vulnerabilities (rely on CMOV)
- More work for the same result (always do both branches)

Resource isolation in Hardware



Drawbacks
- Performance downgrade (one cache partitioned)
- Expensive Hardware (multiple caches)

SCRATCHS

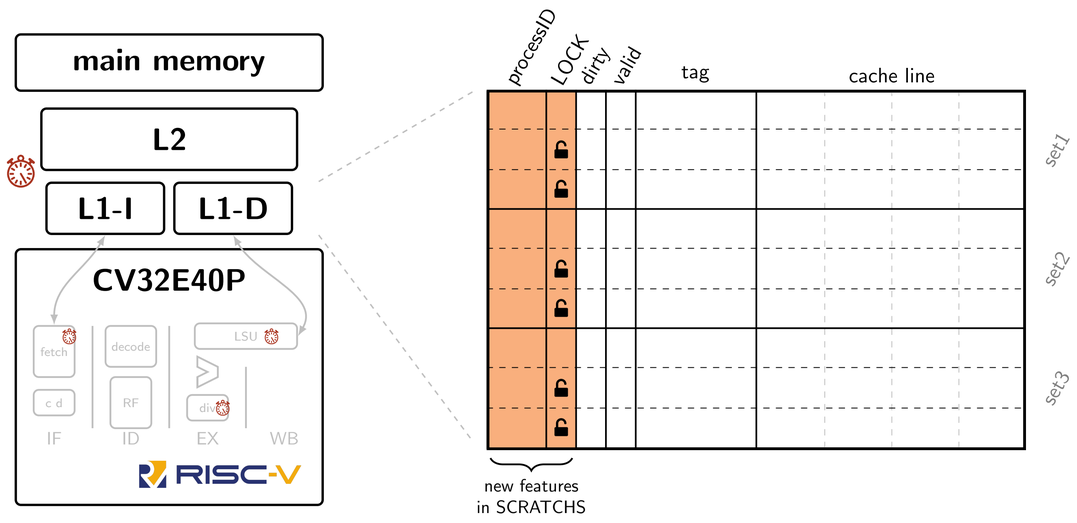


Hardware / Software Co-Design

- **Hardware** implements **security mechanisms**
- **Compiler** and **Operating System** leverage these mechanisms to produce **side-channel resistant** binaries.

Hardware part

Some functional units (e.g. ALU, **LSU**, **division** or branching) can leak a **temporal information** ⏱.



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

Leak		Solutions
Division and modulo op.	→	Constant-time mode through a CSR register
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 OS or the locking process issues a UNLOCK operation.
- At least one way of the cache is kept available to other processes' data.

Software part

Source code with annotations to identify secrets

```
int key; //key∈{0,7}
int choices[8];

int divider = choices[key];
int result = 10/divider;
```

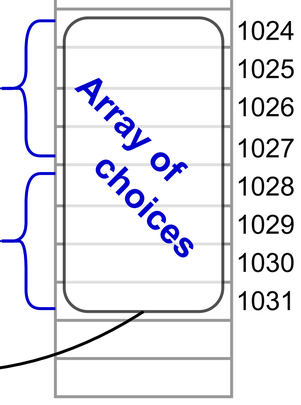
Compilation

Alignement directive

RAM

Assembly code with protections

```
x28 ← LoadWord(1020) //load key
      LockInCache(1024)
      LockInCache(1028)
x28 ← AddImmediate(x28, 1024)
x29 ← LoadWord(x28) //secret choice
      UnlockInCache(1024)
      UnlockInCache(1028)
x30 ← SetConstantTimeMode(1)
x31 ← Division(10, x30) //safe division
x0  ← SetConstantTimeMode(x30)
```



- Divisions on secrets are done in constant time mode
- Secret memory access must be done on locked addresses for constant-time cache-hit
- Locked RAM addresses must be aligned on cache lines.