Thwarting Timing Attacks in Microcontrollers using Fine-grained Hardware Protections

Nicolas GAUDIN

Lab-STICC

Vianney LAPÔTRE, Pascal COTRET & Guy GOGNIAT

July 7, 2023

Summary

Prerequisites : Cache memories

Aim :

- ▶ bring data closer
- ▶ reduce memory latencies

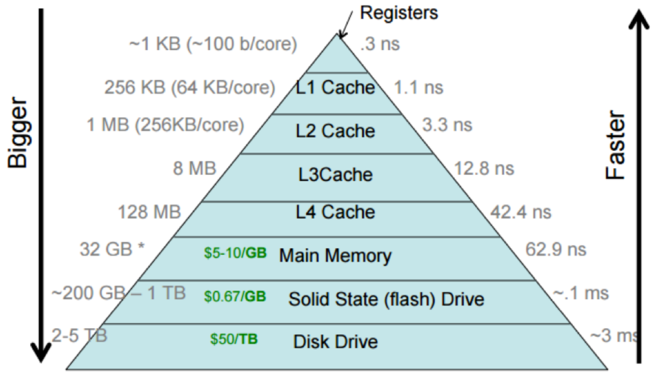


Figure 1: Memory hierarchy

Prerequisites : Cache attacks

How to retrieve information of another process ?

Prerequisites : Cache attacks

How to retrieve information of another process ?

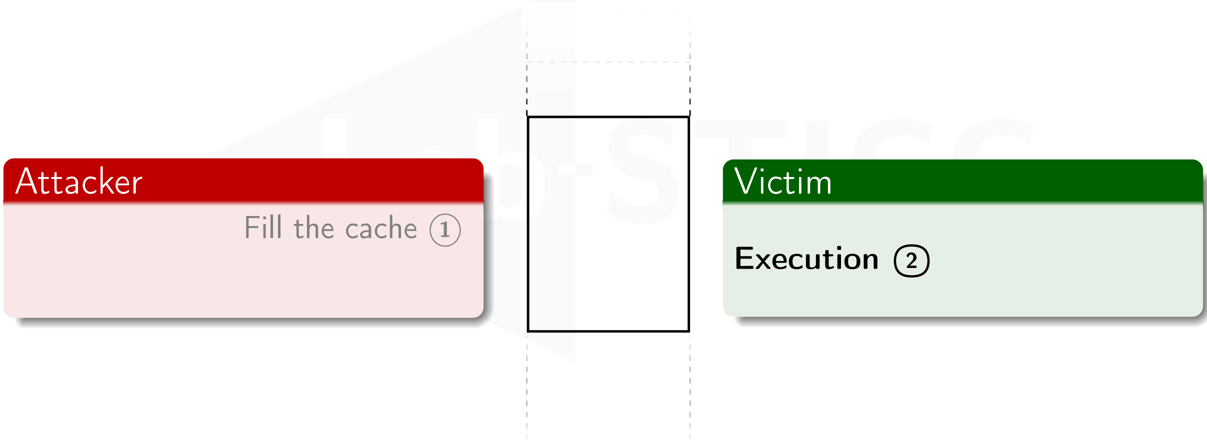
Attacker

Fill the cache ①

Victim

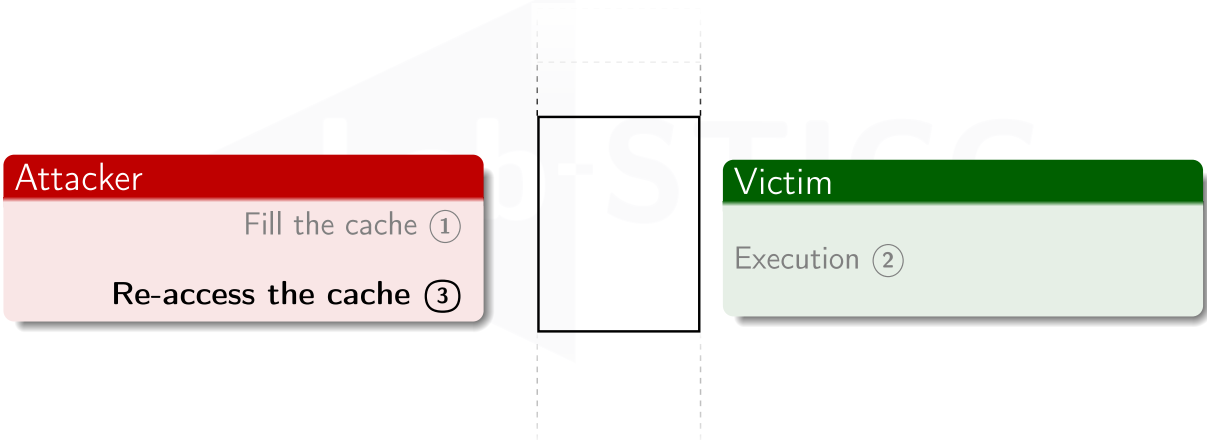
Prerequisites : Cache attacks

How to retrieve information of another process ?



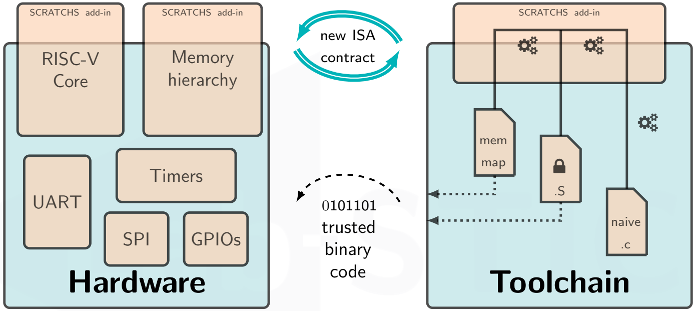
Prerequisites : Cache attacks

How to retrieve information of another process ?



Summary

SCRATCHS project



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

- Aimed attacks : Timing side channel on the microarchitecture
- Ensure efficient constant-time execution and used only when necessary
  - Best convenience between hardware and toolchain contributions

Considered system

Processor

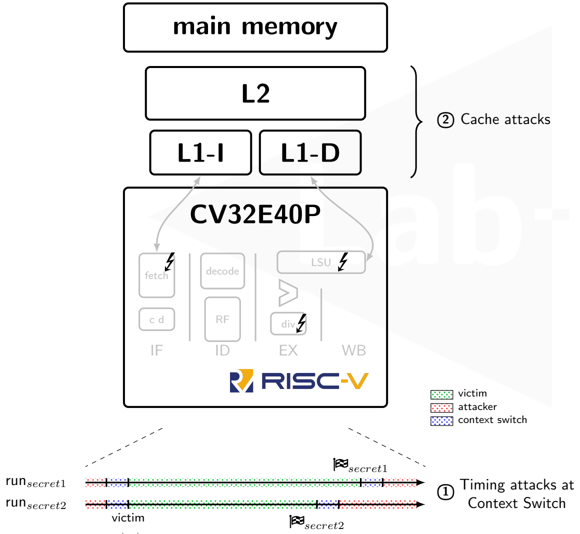
- monocore CV32E40P
  - 4 stage in-order
  - 32 bits
  - no speculation
  - physically addressed

Cache hierarchy

- L1 data cache
  - 4 way set associative
  - 8 KiB cache
  - 16 bytes cache line

To keep in mind : **the simpler the system, the simpler the integration** (theoretically)

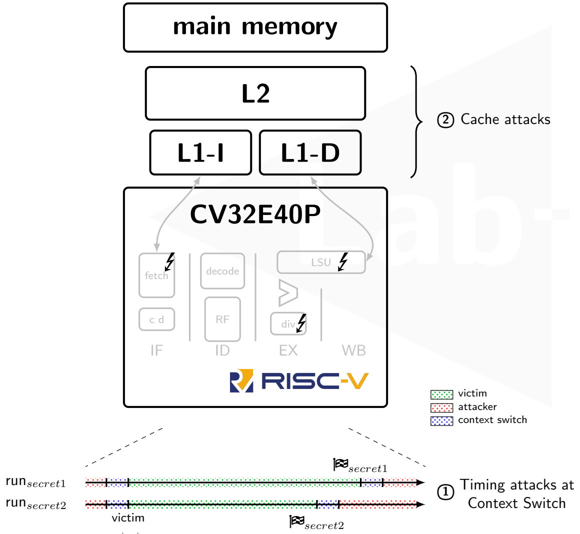
Threat model



An OS schedules and processes.

Only timing side channels are considered.

Threat model



An OS schedules and processes.

Only timing side channels are considered.

The attacker :

- knows the victim program.
- measures time to cycle.
  - victim execution
  - its memory accesses {hit;miss}
- can interrupt.
- shares cache with victim.
- has different memory spaces.

Summary

Considered system

Processor

- monocore CV32E40P
    - 4 stage in-order
    - 32 bits
    - no speculation
    - physically addressed

Cache hierarchy

- L1 data cache
    - 4 way set associative
    - 8 KiB cache
    - 16 bytes cache line

Protection machanisms

- Software controlled mode for constant time execution
- **Lock**ed and **Unlock**ed memory accesses

Summary

State of the Art – Constant time operations

This issue is known and thwarted on x86[1], Arm[2] but also on RISC-V[3] ISAs.

From these existing solutions, we implement it. Our contribution is not the implementation, but the way we use it. The compiler intelligence (the other side of the SCRATCHS project) switches in constant time when necessary.

🔴 : full performance, timings leaks during execution

🟢 : performance loss, constant time execution

[1] Intel, "Data Operand Independent Timing Instruction Set Architecture (ISA) Guidance" ,

[2] arm, "DIT, Data Independent Timing register on Arm Armv8-A Architecture Registers" ,

[3] openHWGroup, "Divider module of CV32E40S RISC-V core" ,

Multi-cycle operations

Table 1: List of multi-cycle instructions (CV32E40P)

Instruction Type	Cycles
Integer Computational	1

Multi-cycle operations

Table 1: List of multi-cycle instructions (CV32E40P)

Instruction Type	Cycles
Integer Computational	1
CSR Access	4 (some CSRs) 1 (the other CSRs)
Load/Store	1 access 2 accesses (if data is non-aligned)
Jump	2
Branch	1 (not-taken) 3 (taken)
Multiplication	1 (32-LSBs computation) 5 (32-MSBs computation)

Multi-cycle operations

Table 1: List of multi-cycle instructions (CV32E40P)

Instruction Type	**Cycles**
Integer Computational	1
CSR Access	4 (some CSRs) 1 (the other CSRs)
Load/Store	1 access 2 accesses (if data is non-aligned)
Jump	2
Branch	1 (not-taken) 3 (taken)
Multiplication	1 (32-LSBs computation) 5 (32-MSBs computation)
Division Remainder	**3-35**

Constant time operations

Involved instructions : div, divu, rem, remu

	Naive		Constant time		
Divider value	Cycles	Leak[1]	Idle cycles	Total	Leak
0x0000 0000	35	0x0000 0000	0	35	∅
0x0000 0001	34	0x0000 0001	1	35	∅
0x0000 0002	33	0x0000 000**2**	2	35	∅
0x0003 F5A2	17	0x000**2** xxxx	18	35	∅
0xBE63 20C1	3	0x**8**xxx xxxx	32	35	∅

[1] **2** = 0b001x, **8** = 0b1xxx

How to implement and use it

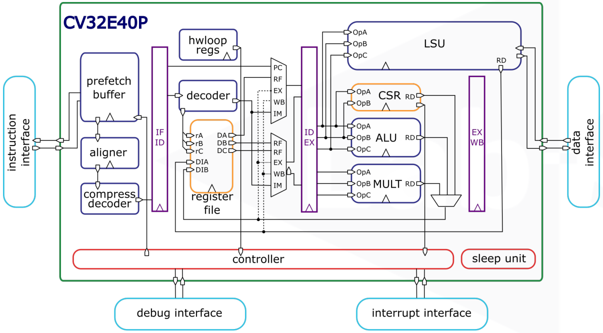


Figure 2: CV32E40P Block diagram

How to implement and use it

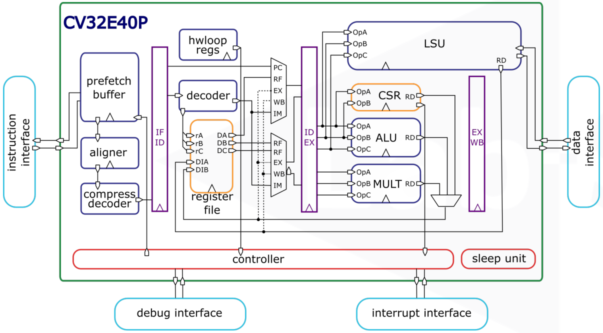


Figure 2: CV32E40P Block diagram

With the swctm pseudo instruction compiled as :
csrr{s|c} rd, CONSTANT_TIME, rs1

```
# block of sensitive code
swctm #set CT mode
add a2, t0, a5
c.addi a3, 82
div a0, a3, a2
rem t1, a5, a2
lw a2, 4(sp)
div a0, a3, a2
swctm #reset CT mode
```

Listing 2: Application example

Commentaires d'améliorations

- expliquer le temps d'exécution avec des valeurs différentes de diviseur (en activant ou non le CTM)
- faire une simulation d''attaque' en mode interruption de l'attaquant...

Summary

State of the Art – Hardware Mitigations of Cache

Randomization based caches

RPcache[a], **ScatterCache**[b] and **Ceaser**[c] propose cache designs based on randomization. Prime+Prune+Probe[d] find eviction sets in randomized caches from only hundred accesses. It requires regular updates of the cache mapping.

• Randomized caches provide a strong security (as long as randomness is randomness) but can be a source of performance loss.

[a] Wang and Lee, "New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks" , 2007

[b] Werner et al., "ScatterCache: Thwarting Cache Attacks via Cache Set Randomization" , 2019

[c] Qureshi, "CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping" , 2018

[d] Purnal et al., "Systematic Analysis of Randomization-based Protected Cache Architectures" , 2021

State of the Art – Hardware Mitigations of Cache

Caches partitioning - *with the support of the software*

NoMo-cache[a] partitions the cache by allocating a set of ways to sensitive applications. Also, **SecDCP**[b] (secure and unsecure ways), or **COLORIS**[c] (memory page allocation) use coarse-grained partitioning.

Wang et al. proposes **PLcache**[d], a lightweight mechanism allowing the lock of process cache lines.

• Cache partitioning is (generally) a lightweight solution, but may have a major impact on performance depending on granularity.

[a] Domnitser et al., "Non-Monopolizable Caches: Low-Complexity Mitigation of Cache Side Channel Attacks" , 2012

[b] Wang et al., "SecDCP: Secure dynamic cache partitioning for efficient timing channel protection" , 2016

[c] Ye et al., "COLORIS: A dynamic cache partitioning system using page coloring" , 2014

[d] Wang and Lee, "New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks" , 2007

State of the Art

PLcache[1] problem :

In accordance with the replacement policy, side effects are introduced accessing in the same cache set of a locked data. It modifies the victim execution behavior on the memory accesses near the locked data. Thus, this approach does not guarantee constant time access to locked cache lines.

PLcache provides cache line reservation rather than cache line locking.

[1] Wang and Lee, "New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks" , 2007

Commentaires d'améliorations

- mettre en forme le problème de plcache (figure de cache où des accès viennent modifié un truc)
- expliquer mieux ce qu'apporte notre solution (aucune modification)

Protection mechanisms

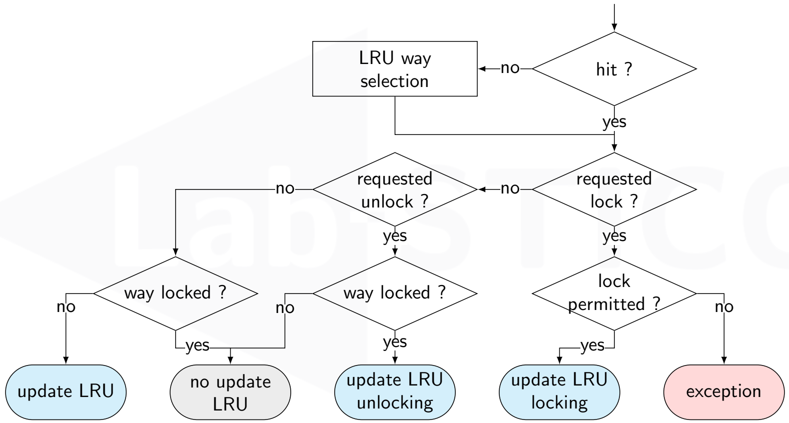


Figure 3: Lock handling procedure

Cache architecture

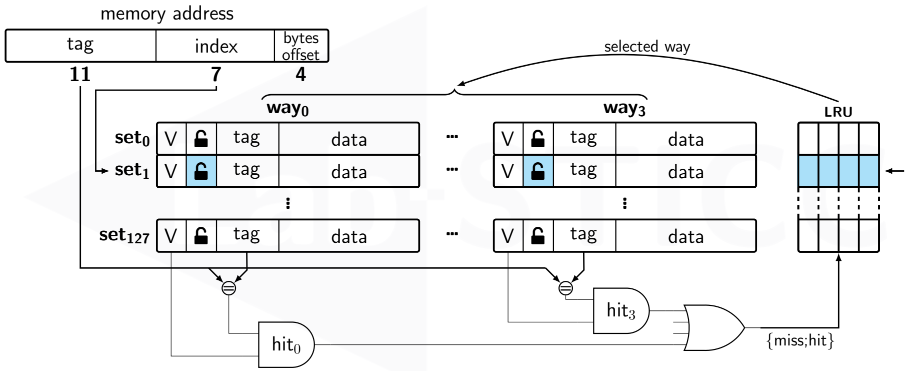


Figure 4: 4-way set associative cache architecture

Behavior of the replacement policy using the `lock` principle

```
lock   a1
lock   a2
lw     a3
lw     a1
unlock a2
lock   a4
lw     a5
lw     a6
```

Behavior of the replacement policy using the `lock` principle

```
lock   a1
lock   a2
lw     a3
lw     a1
unlock a2
lock   a4
lw     a5
lw     a6
```

execution of ①

- cache miss
- locking the data
  - cannot be evicted
  - update the policy

Behavior of the replacement policy using the `lock` principle

```
1  lock    a1
2  lock    a2
3  lw      a3
4  lw      a1
5  unlock  a2
6  lock    a4
7  lw      a5
8  lw      a6
```

execution of ②

- cache miss
- locking the data
  - cannot be evicted
  - update the policy

Behavior of the replacement policy using the `lock` principle

```
lock   a1
lock   a2
lw     a3
lw     a1
unlock a2
lock   a4
lw     a5
lw     a6
```

execution of ③

- cache miss
- standard access
  - update the policy

Behavior of the replacement policy using the lock principle

```
lock    a1
lock    a2
lw      a3
lw      a1
unlock  a2
lock    a4
lw      a5
lw      a6
```

execution of ④

- cache hit
- access to a locked line
    - return the data
    - **no** update the policy

Behavior of the replacement policy using the `lock` principle

```
lock   a1
lock   a2
lw     a3
lw     a1
unlock a2
lock   a4
lw     a5
lw     a6
```

execution of ⑤

- unlocking the data
  - update the policy

Behavior of the replacement policy using the `lock` principle

```
lock   a1
lock   a2
lw     a3
lw     a1
unlock a2
lock   a4
lw     a5
lw     a6
```

execution of ⑥

- cache miss
- locking the data
    - cannot be evicted
    - update the policy

Behavior of the replacement policy using the `lock` principle

```
lock   a1
lock   a2
lw     a3
lw     a1
unlock a2
lock   a4
lw     a5
lw     a6
```

execution of ⑦

- cache miss
  - evince a3
- standard access
  - update the policy

Behavior of the replacement policy using the lock principle

```
1  lock    a1
2  lock    a2
3  lw      a3
4  lw      a1
5  unlock  a2
6  lock    a4
7  lw      a5
8  lw      a6
```

execution of ⑧

- cache miss
  - evince a2 (no longer locked)
- standard access
  - update the policy

Post-synthesis area results [1]

Core: CV32E40P (RISC-V based)
Cache: 8 KiB, 4-way set-associative, L1 data cache

	BRAMs	LUTs	FFs
CV32E40P core	-	4950	2142
Cache (w/o lock)	8.5	489	888
Cache (w/ lock)	8.5	512	894

[1] Synthesis for Kyntex-7 chip using Vivado 2022 tool

Summary

Perspectives

- Evaluate performances and security with many benches
  - Find the best cache parameter (#set, #way, cache line size, #free way)
- Implement an embedded OS
  - support process IDs
  - lock with many processes
- Decrease our current limits
  - involve locking other cache levels

Conclusion

CT Operations

An easily scalable implementation to other RISC-V instructions if needed.

Cache

A promising solution with a strong security. We need more evaluation in both security and performance. Moreover, we can have a different approach to manage locked data with more than one cache level.
ref. Perspectives section

Thank you

further information on : *https://project.inria.fr/scratchs/*

Bibliography

- arm. “DIT, Data Independent Timing register on Arm Armv8-A Architecture Registers”. In: url: https://developer.arm.com/documentation/ddi0595/2021-06/AArch64-Registers/DIT--Data-Independent-Timing.
- Domnitser, Leonid et al. “Non-Monopolizable Caches: Low-Complexity Mitigation of Cache Side Channel Attacks”. In: *ACM Transactions on Architecture and Code Optimization* (Jan. 2012). doi: 10.1145/2086696.2086714.
- Intel. “Data Operand Independent Timing Instruction Set Architecture (ISA) Guidance”. In: url: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/data-operand-independent-timing-isa-guidance.html.

Bibliography

- openHWGroup. "Divider module of CV32E40S RISC-V core". In: url: https://github.com/openhwgroup/cv32e40s/blob/29d44172a6341160a40c3637fa883e311eb6744c/rtl/cv32e40s_div.sv#L246.
- Purnal, Antoon et al. "Systematic Analysis of Randomization-based Protected Cache Architectures". In: *Proc. IEEE Symposium on Security and Privacy (SP)*. May 2021. doi: 10.1109/SP40001.2021.00011.
- Qureshi, Moinuddin K. "CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping". In: *Proc. International Symposium on Microarchitecture (MICRO)*. 2018. doi: 10.1109/MICRO.2018.00068.
- Wang, Yao et al. "SecDCP: Secure dynamic cache partitioning for efficient timing channel protection". In: *53ndDesign Automation Conference (DAC)*. 2016. doi: 10.1145/2897937.2898086.

Bibliography

- Wang, Zhenghong and Ruby B. Lee. "New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks". In: *Proc. International Symposium on Computer Architecture (ISCA)*. 2007. doi: 10.1145/1250662.1250723.
- Werner, Mario et al. "ScatterCache: Thwarting Cache Attacks via Cache Set Randomization". In: *Proc. 28th USENIX Security Symposium (USENIX Security)*. 2019. url: https://www.usenix.org/conference/usenixsecurity19/presentation/werner.
- Ye, Ying et al. "COLORIS: A dynamic cache partitioning system using page coloring". In: *Proc. International Conference on Parallel Architecture and Compilation Techniques (PACT)*. 2014. doi: 10.1145/2628071.2628104.