

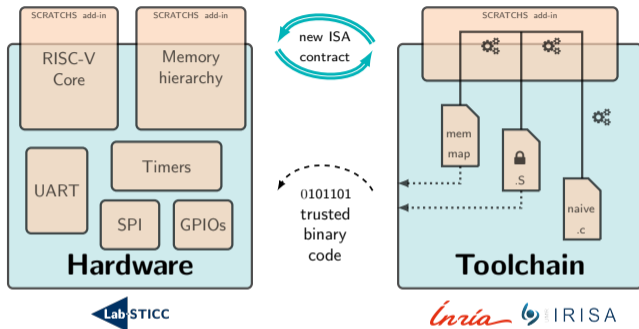
SCRATCHS: Side-Channel Resistant Applications Through Co-designed Hardware/Software

Vianney Lapôte

Journée thématique du club des partenaires sur le RISC-V - GDR SOC²

September 29, 2023

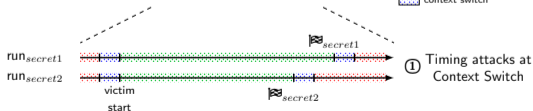
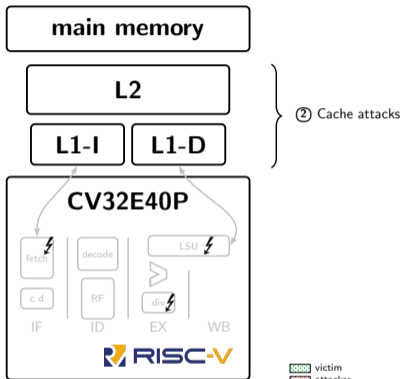




Side-Channel Resistant Applications Through Co-designed Hardware/Software

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

Threat model



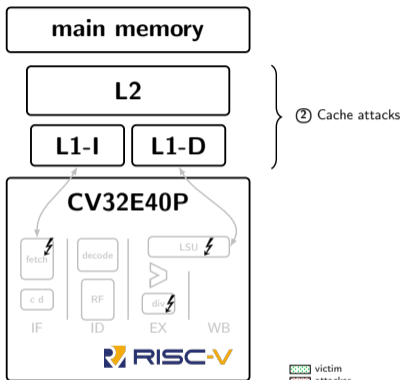
An OS schedules and processes.

Only timing side channels are considered.

The attacker :

- knows the victim program.
- measures time with cycle accuracy.
 - victim execution
 - its memory accesses {hit;miss}
- can interrupt.
- shares cache memories with victim.

Threat model



Ensure efficient and on-demand constant-time execution

Sources of leakages

Branching

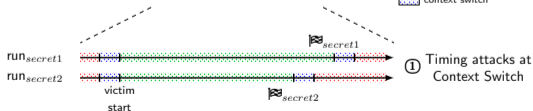
if (condition(secret))

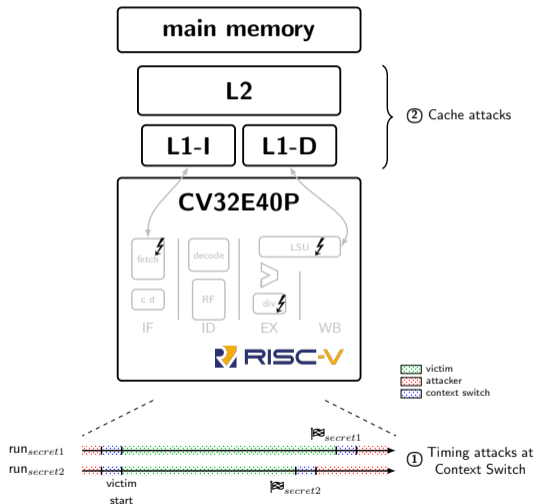
Operation with variable execution time

dividend/secret;

Index for Memory access

array[secret];



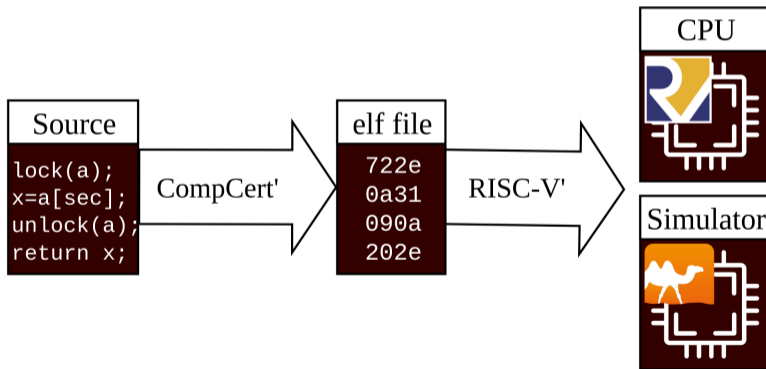


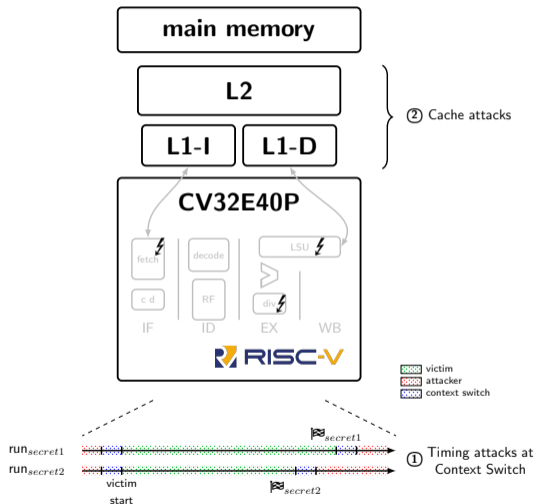
Ensure efficient and on-demand constant-time execution

- Hardware support for
 - software controlled mode for constant time execution
 - software controlled cache lines Locked and Unlocked
- Software tools for
 - constant time programming
 - proposed protections simulation and formal verification

Project toolchain

- CompCert Abs-Int (minor changes for our annotations)
- RISC-V ISA extension
- Simulator to evaluate security
- WIP : formal proof of the proposed security mechanism





Ensure efficient and on-demand constant-time execution

- **Hardware support for**
 - software controlled mode for constant time execution
 - software controlled cache lines Locked and Unlocked
- **Software tools for**
 - constant time programming
 - proposed protections simulation and formal verification

multi-cycle instructions in 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

Software controlled mode for constant time execution

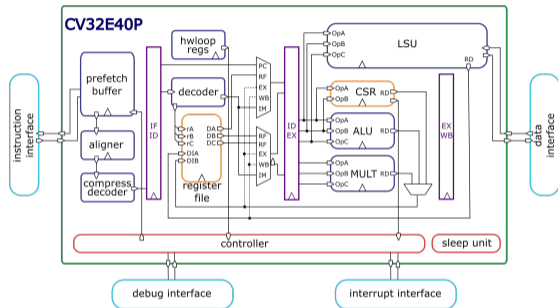


Figure: CV32E40P Block diagram

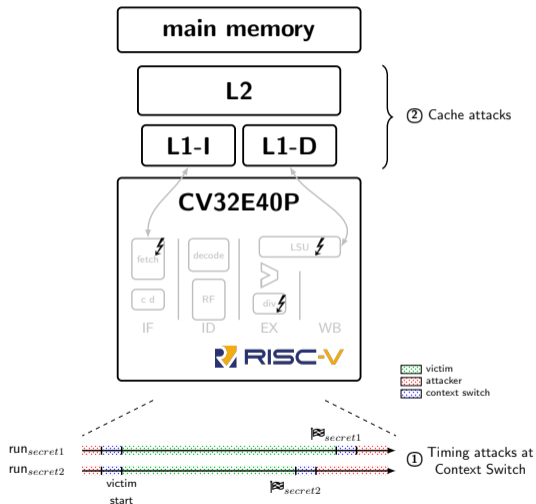
With the `swctm` pseudo instruction compiled as :

```
csrr{s|c} rd, CONSTANT_TIME, rs1
```

```

1  # block of sensitive code
2  swctm #set CT mode
3  add a2, t0, a5
4  c.addi a3, 82
5  div a0, a3, a2
6  rem t1, a5, a2
7  lw a2, 4(sp)
8  div a0, a3, a2
9  swctm #reset CT mode

```





Ensure efficient and on-demand constant-time execution

- **Hardware support for**
 - software controlled mode for constant time execution
 - **software controlled cache lines Locked and Unlocked**
- **Software tools for**
 - constant time programming
 - proposed protections simulation and formal verification

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.

 Randomized caches provide a strong security but it require regular updates of the cache mapping. This 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

Caches partitioning - *with the support of the software*

- **NoMo-cache^a** partitions the cache by allocating a set of ways to sensitive applications.
- **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

PLcache limitations

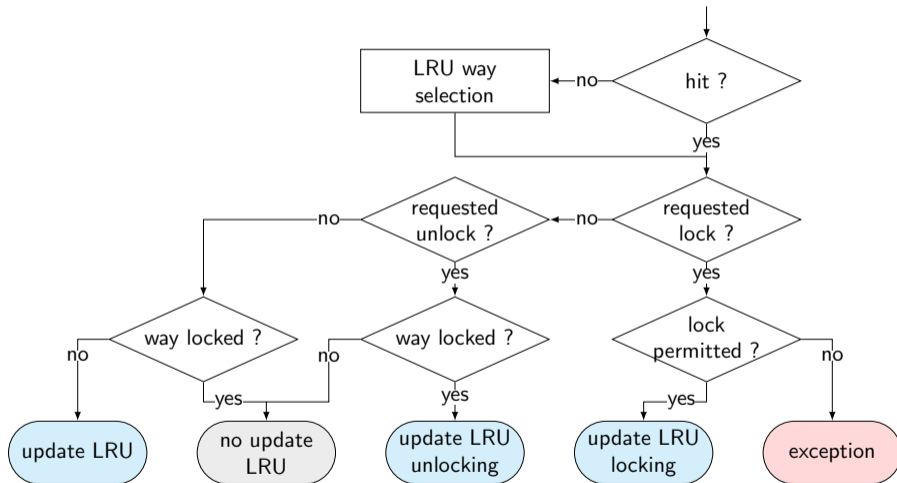
- PLcache¹ does not ensure constant-time accesses
 - some accesses bypass the cache
 - Locked cache lines can be accidentally evicted by the owner process
 - Replacement policy is shared with other processes
- PLcache is not fully secured - it's provides cache line reservation rather than cache line locking.
 - Replacement policy can be manipulated by an attacker for both non-locked and locked cache lines

Our approach

- No accidental unlock → use **Unlock instructions**
- LRU-related meta-data is **not updated for locked lines**.
- At least **one free way**: Lock fail when only one unlocked way left in the cache set

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

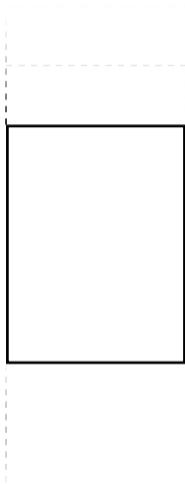
Lock - handling procedure



Lock - a pedagogical example

```

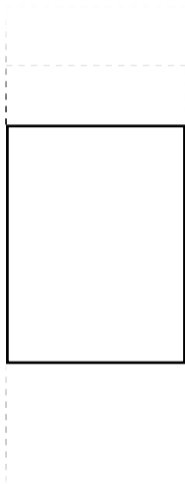
1  lock   a1
2  lock   a2
3  lw     a3
4  lw     a1
5  unlock a2
6  lock   a4
7  lw     a5
8  lw     a6
  
```



Lock - a pedagogical example

```

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



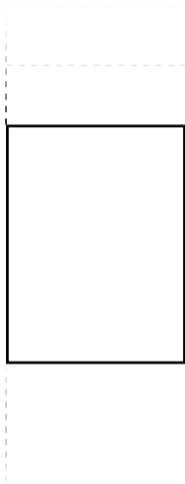
execution of ①

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

Lock - a pedagogical example

```

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



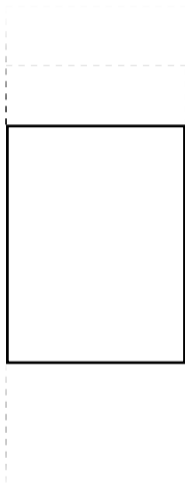
execution of ②

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

Lock - a pedagogical example

```

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



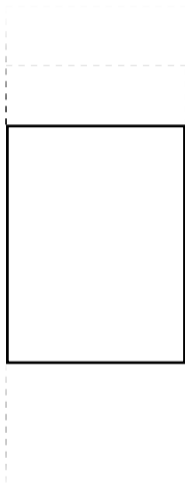
execution of ③

- cache miss
- standard access
 - update the policy

Lock - a pedagogical example

```

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



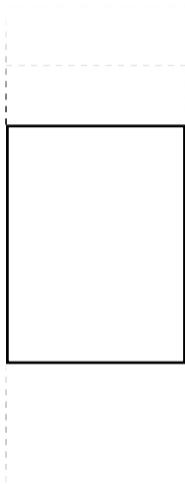
execution of ④

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

Lock - a pedagogical example

```

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



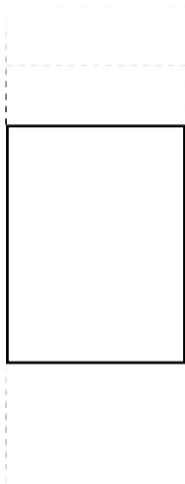
execution of ⑤

- unlocking the data
 - update the policy

Lock - a pedagogical example

```

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



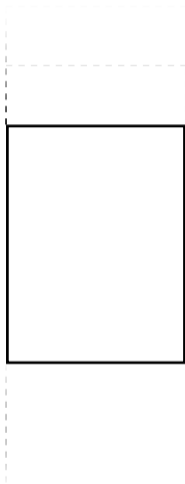
execution of ⑥

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

Lock - a pedagogical example

```

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



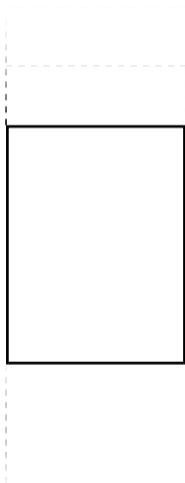
execution of ⑦

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

Lock - a pedagogical example

```

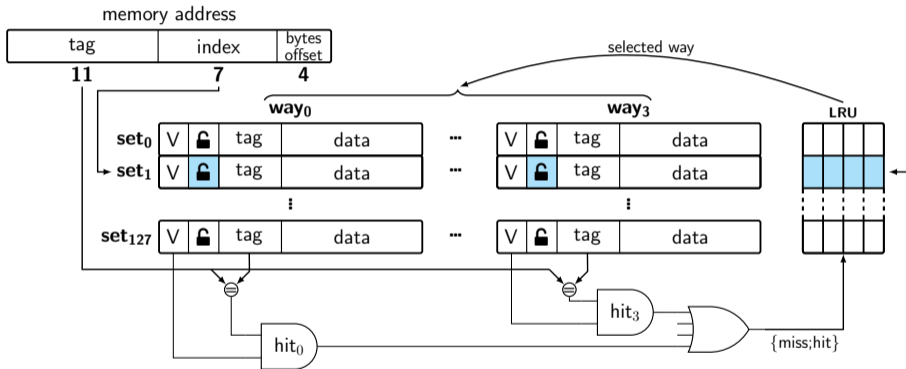
1  lock  a1
2  lock  a2
3  lw    a3
4  lw    a1
5  unlock a2
6  lock  a4
7  lw    a5
8  lw    a6
    
```



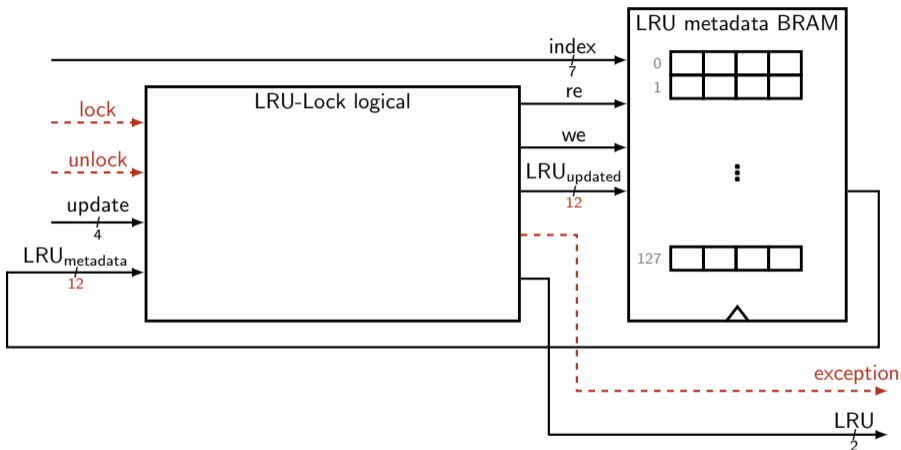
execution of ⑧

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

Lock - 4-way set associative cache architecture



LRU with Lock & Unlock - HW implementation



Preliminary results

Core: CV32E40P (RISC-V based)

Cache: 8 KiB, 4-way set-associative, L1 data cache

	W/o Lock			W/ Lock		
	LUTs	FFs	BRAMs	LUTs	FFs	BRAMs
CPU	5653	3484	8.5	5682 (+0.51%)	3500 (+0.46%)	8.5
CV32E40P core	4655	2260	0	4657 (+0.44%)	2262 (+0.09%)	0
Cache	988	1057	8.5	1015 (+2.66%)	1069 (+1.12%)	8.5

²Synthesis for Kintex-7 chip using Vivado 2022 tool

- We consider a Prime+Probe³ attack targeting the AES SBOX
 - ✗ Lock mechanism is disabled

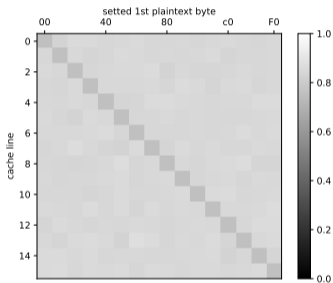


Figure: Prime+Probe on AES (1st round only, 1st plaintext byte settled, key = 0x00).

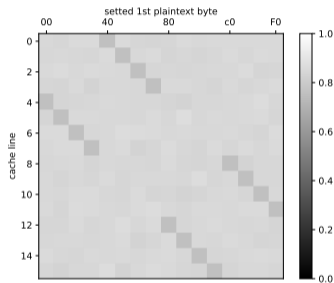



Figure: Prime+Probe on AES (1st round only, 1st plaintext byte settled, key = 0x42).

³Gullasch, Bangerter, and Krenn, "Cache Games - Bringing Access-Based Cache Attacks on AES to Practice", 2011

- We consider a Prime+Probe⁴ attack targeting the AES SBOX
 -  Lock mechanism is enable for the entire SBOX

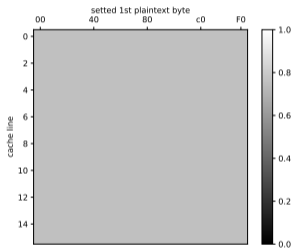
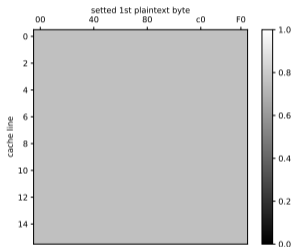


Figure: Prime+Probe on AES locking full table (1st round only, 1st plaintext byte settled, key = 0x00).

⁴Gullasch, Bangerter, and Krenn, "Cache Games - Bringing Access-Based Cache Attacks on AES to Practice", 2011

- We consider a Prime+Probe⁴ attack targeting the AES SBOX
 - ✔ Lock mechanism is enable for the entire SBOX



	Binary size (in ko)
Unprotected AES	60.6
Protected AES	60.9 +0.5%

Figure: Prime+Probe on AES locking full table (1st round only, 1st plaintext byte settled, key = 0x00).

⁴Gullasch, Bangerter, and Krenn, "Cache Games - Bringing Access-Based Cache Attacks on AES to Practice", 2011



- RISC-V CV32E40P extension for constant time execution and cache-based SCA mitigation
- Hardware implementation on FPGA
- Current limits and future works
 - support for multiple cache levels
 - OS support to catch the error and run a back-up solution
 - Study hybrid solutions including cache randomization

<https://project.inria.fr/scratches/>


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

*Many thanks to Jean-Loup Hatchikian-Houdot, Nicolas
Gaudin, Frédéric Besson, Pascal Cotret, Guy Gogniat, Guillaume Hiet,
and Pierre Wilke*

Thank you for listening!

Any questions?

-  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.
-  Gullasch, David, Endre Bangerter, and Stephan Krenn. “Cache Games – Bringing Access-Based Cache Attacks on AES to Practice”. In: *2011 IEEE Symposium on Security and Privacy*. 2011, pp. 490–505. doi: 10.1109/SP.2011.22.
-  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: *53rd Design Automation Conference (DAC)*. 2016. doi: [10.1145/2897937.2898086](https://doi.org/10.1145/2897937.2898086).
-  Wang, Zhenhong 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](https://doi.org/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](https://doi.org/10.1145/2628071.2628104).