Fine-grained dynamic partitioning against cache-based side channel attacks

Nicolas Gaudin, **Vianney Lapôtre**, Guy Gogniat, Pascal Cotret

journées nationales 2025 du GDR Sécurité Informatique

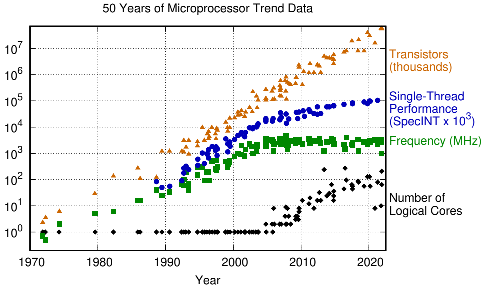
Summary

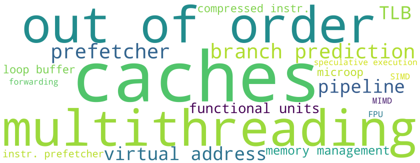
- Context
- State of the art
- Fine-grained locking mechanism
- Implementation with an N-way set associative cache
- Implementation with a randomization-based skewed cache
- Conclusion

Summary

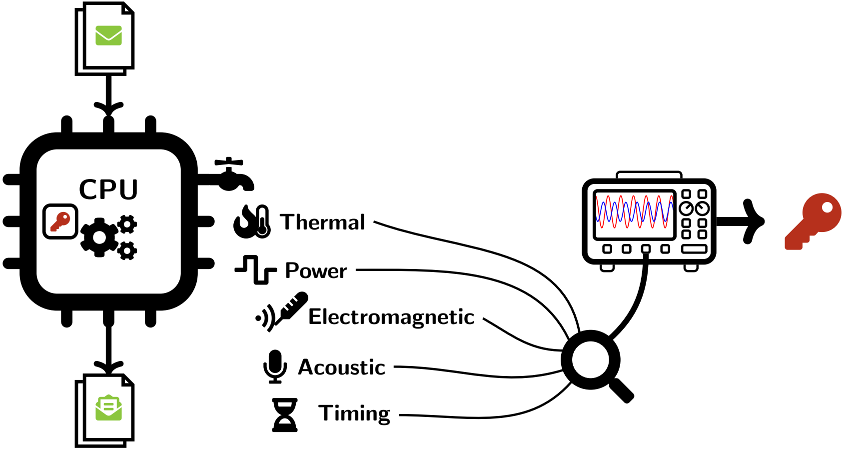
Context

- Number of IoT devices soars
- Wide usage
- Critical applications
  - Health
  - Bank
  - Industry
  - Transport
  - Logistic

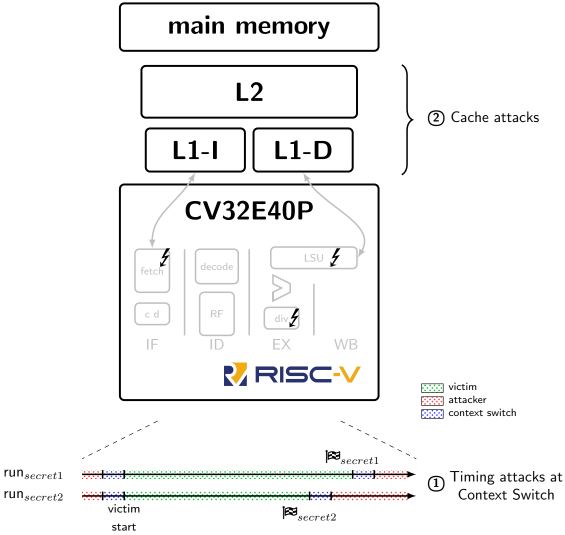




Side Channel Attacks (SCA)



Timing leakage



Sources of leakages exploited by :

Branching

if (condition(**secret**))

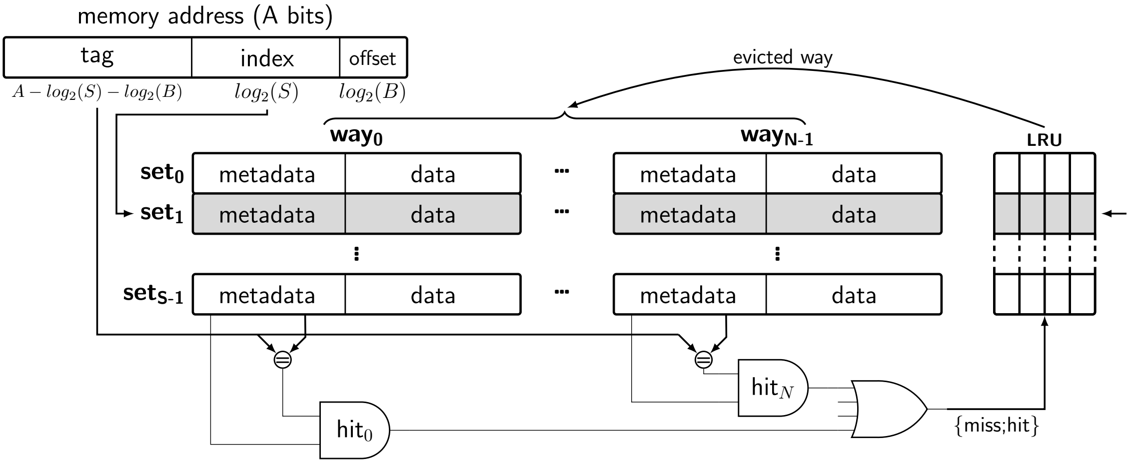
Operation with variable execution time

dividend/**secret**;

Index for Memory access

array[**secret**];

Cache Memories - N-way set associative cache



PRIME+PROBE attack

Objectives :

- Infer which cache set(s) is accessed by the victim
- Observe behavior of the attacker memory accesses

PRIME+PROBE attack

Objectives :

- Infer which cache set(s) is accessed by the victim
- Observe behavior of the attacker memory accesses

Requirements :

- Eviction Set matching with victim addresses
- Shared cache resource
- (High) Precision Timer

Prime+Probe attack

Objectives :

- Infer which cache set(s) is accessed by the victim
- Observe behavior of the attacker memory accesses

Requirements :

- Eviction Set matching with victim addresses
- Shared cache resource
- (High) Precision Timer

How ? :

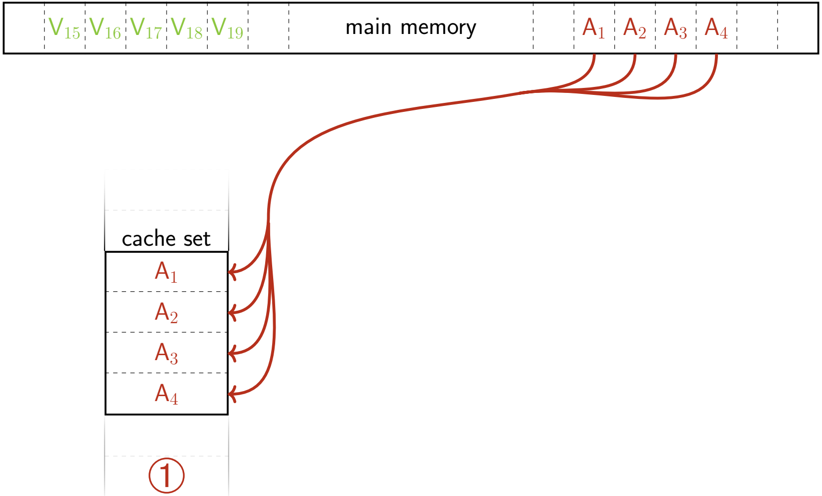
- with a 3-step attack

① fill a cache set using the eviction set

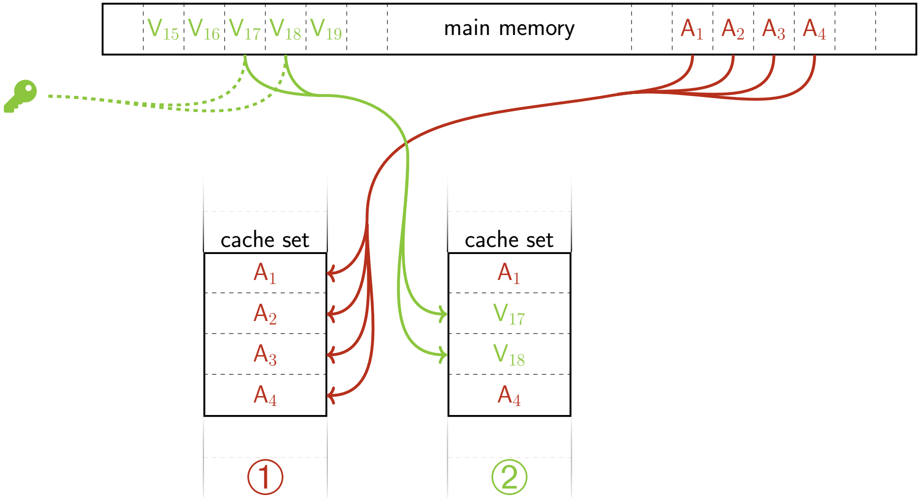
② let the victim execute

③ access and time each address of the eviction set

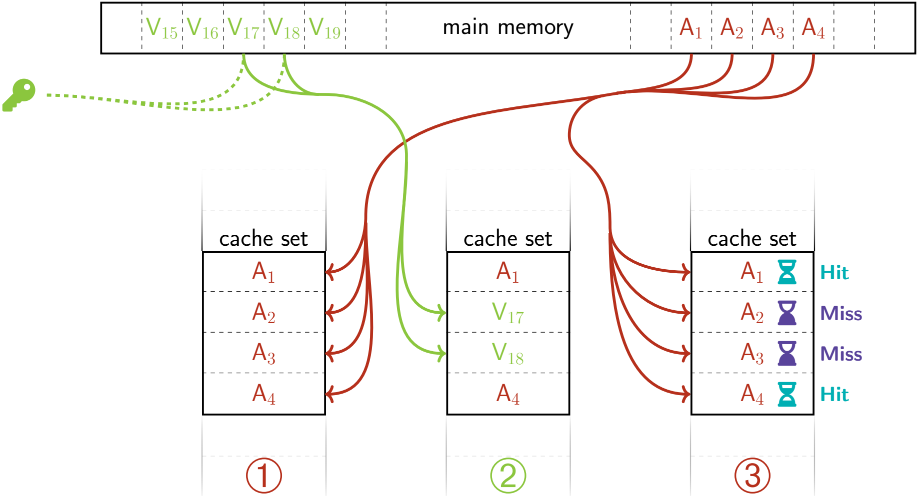
PRIME+PROBE attack



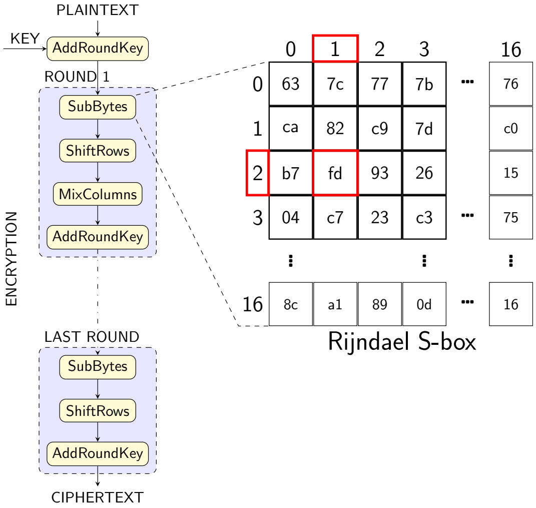
PRIME+PROBE attack



PRIME+PROBE attack



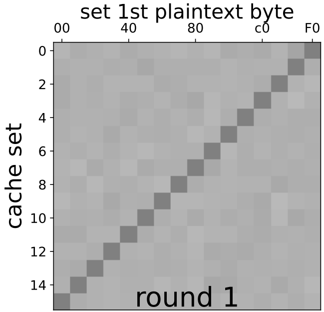
Considered attack on AES-128



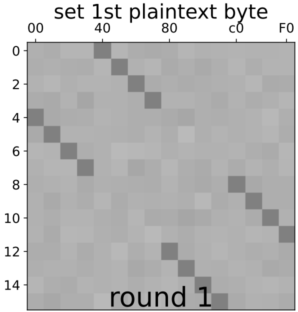
Rijndael S-box

- SubBytes step accesses SBOX$[P_i \oplus K_i]$
- Known plaintext attack

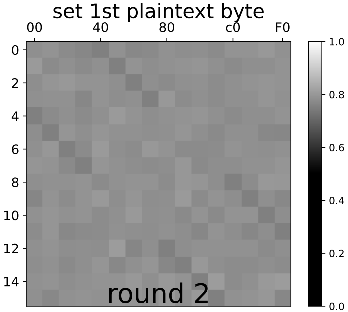
Example with PRIME+PROBE on AES-128



key = 0xFF



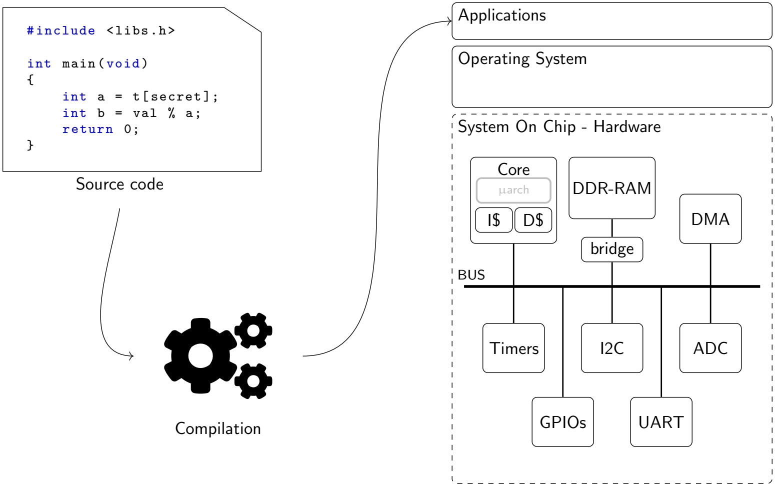
key = 0x42



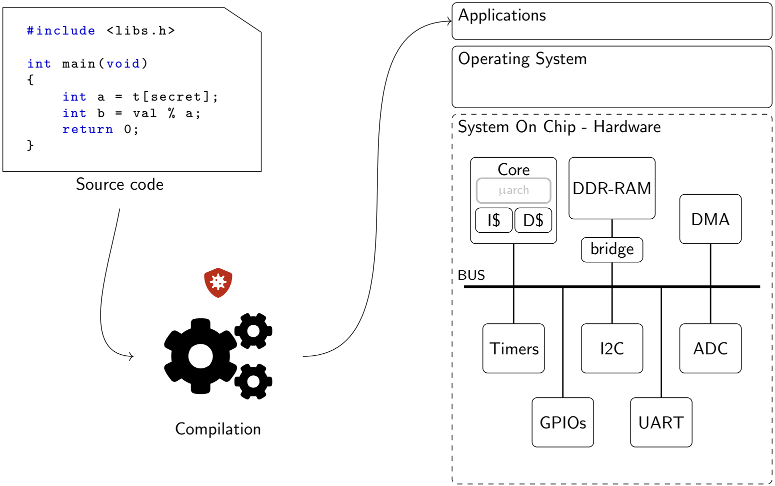
key = 0x42

Summary

Detection-based countermeasures

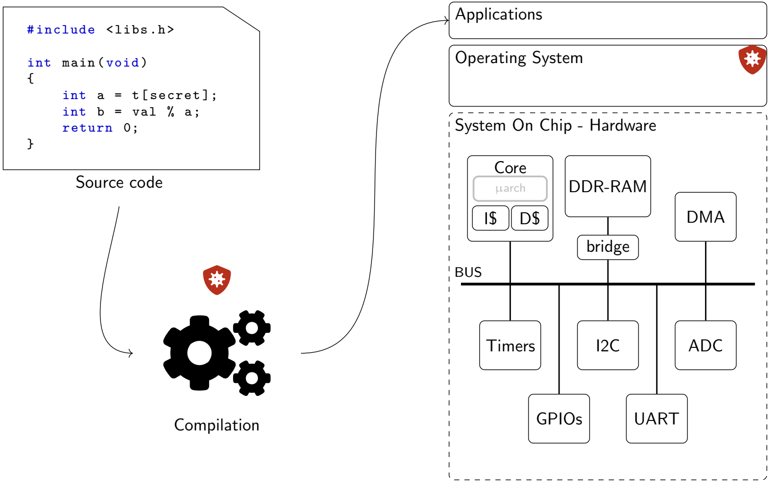


Detection-based countermeasures



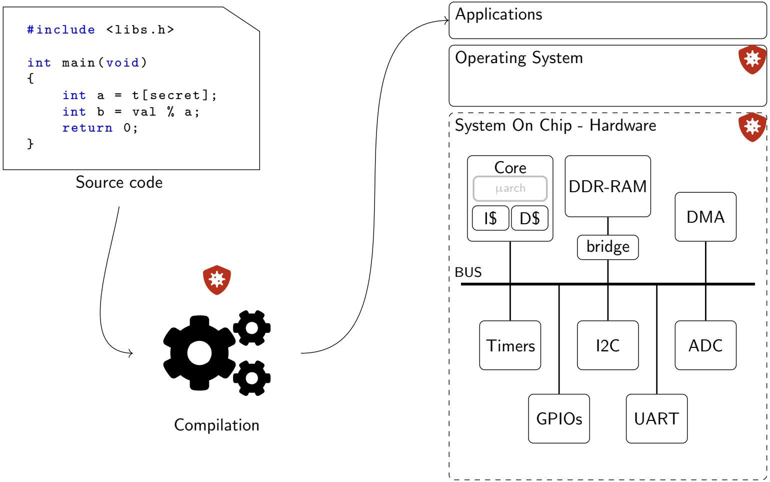
- Winderix *et al.* [1]
- CacheBar [2]

Detection-based countermeasures



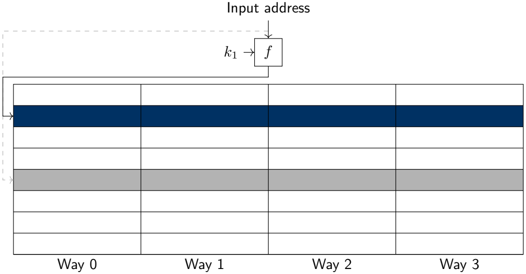
- Winderix *et al.* [1]
- CacheBar [2]
- Nights-Watch [3]
- WHISPER [4]

Detection-based countermeasures



- Winderix *et al.* [1]
- CacheBar [2]
- Nights-Watch [3]
- WHISPER [4]
- täkō [5]

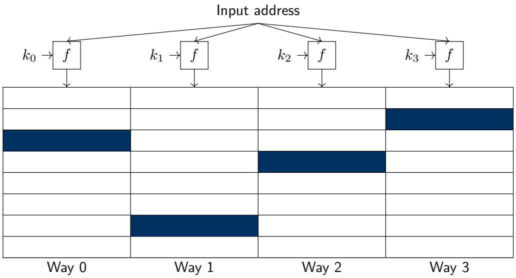
Randomization-based countermeasures - *Associative ways*



- Randomized the accessed cache set
  - Permutation table
  - Index Derivation Function

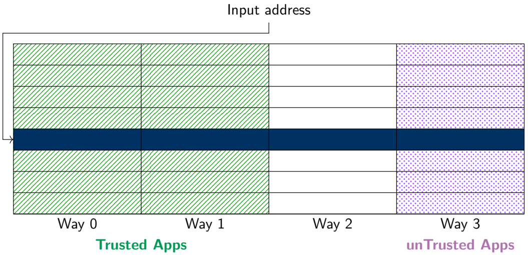
- RPcache [6]
- CEASER [7]
- ScrambleCache [8]
  - Easy to bypass this randomness [9]
  - Need to update primitives frequently
  - Maintain security costs performances

Randomization-based countermeasures - *Skewed ways*



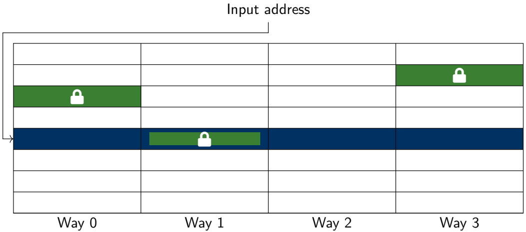
- Randomized accessed cache lines dissociating cache set
  - IDF for each way
- CEASER-S [10]
- ScatterCache [11]
  - Disruptive attack allows to find eviction set
  - PRIME+PRUNE+PROBE [9]
- ClepsydraCache [12]
  - Analog/digital costly design

Partitioning-based countermeasures - *coarse-grained*



- Partition resource to avoid conflicts
- Software-based :
  - COLORIS [13]
  - COTSknight [14]
- Hardware-based :
  - NoMoCache [15]
  - SecDCP [16]
    - Partitions do not meet needs

Partitioning-based countermeasures - *fine-grained*



- Partition resource to avoid conflicts
- Fine-grained :
  - Vantage [17]
  - PLcache [6]
    - Avoid conflict based attacks but leaks on LRU update

State of the Art Synthesis

- **Randomization-based**
  - Autonomous
  - Need to frequently update security (and so invalidate the cache)
- **Coarse Grained Partitionning**
  - Provide a security support for OS/applications
  - Can widely affect performances

State of the Art Synthesis

- **Randomization-based**
  - Autonomous
  - Need to frequently update security (and so invalidate the cache)
- **Coarse Grained Partitionning**
  - Provide a security support for OS/applications
  - Can widely affect performances

- Countermeasures are designed considering complex system
  - Doesn't fit with embedded systems requirements/possibilities
  - Often focus on Last Level Cache (large shared cache)
- Not directly compatible with embedded systems

State of the Art Synthesis

- **Fine Grained Partitionning**
  - Provide a security support for OS/applications
  - Affect slightly performances

State of the Art Synthesis

- **Fine Grained Partitionning**
  - 🚦 Provide a security support for OS/applications
  - 🚦 Affect slightly performances

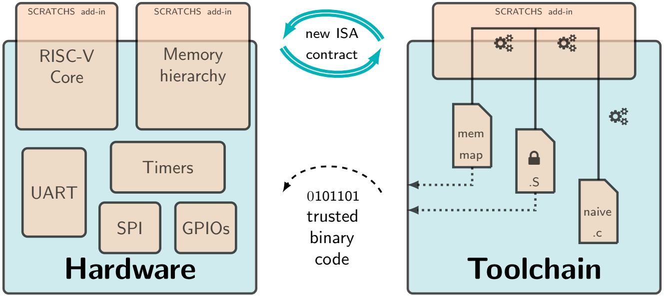
- PLcache is a candidate for our security mechanism
  - Introduce new instructions to reserve cache lines
  - Fit with embedded systems requirement and limitations

Summary

- Context
- State of the art
- Fine-grained locking mechanism
- Implementation with an N-way set associative cache
- Implementation with a randomization-based skewed cache
- Conclusion

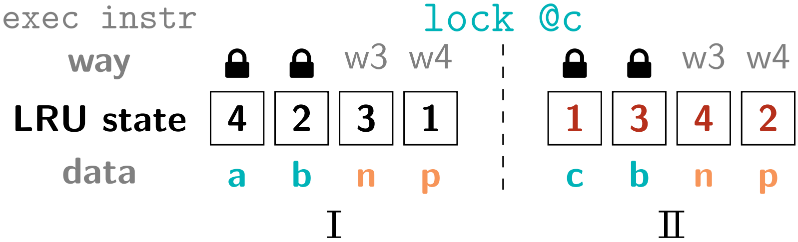
SCRATCHS project

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



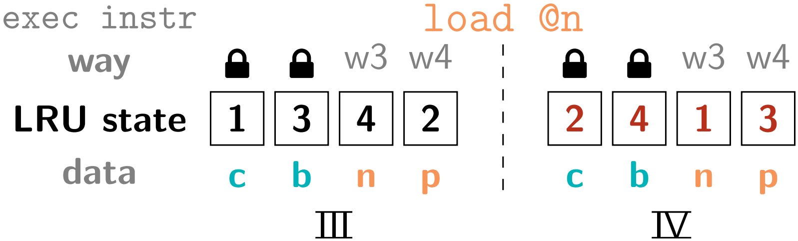
Ensure **efficient** and **on-demand** constant-time execution

PLcache [6] issues



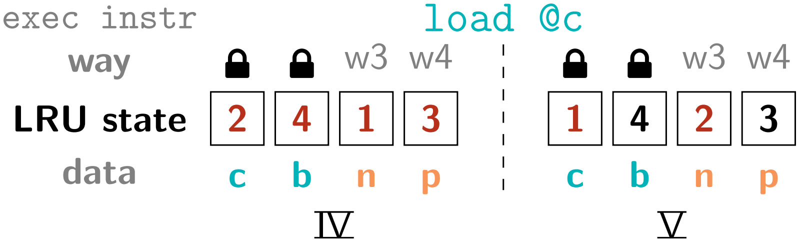
- No contant time accesses on locked data

PLcache [6] issues



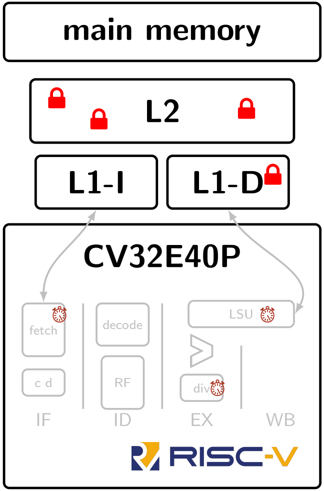
- No contant time accesses on locked data
- Shared LRU state among processes [18]

PLcache [6] issues



- No contant time accesses on locked data
- Shared LRU state among processes [18]

Our lock Mechanism



- Extend the Instruction Set Architecture
  - `lock` and `unlock` instructions

`lock` instr. keeps data cache line in cache

- guarantee constant time access
- locked cache line cannot be evicted
- mitigate EVICT+TIME and PRIME+PROBE

`unlock` instr. releases locked cache line

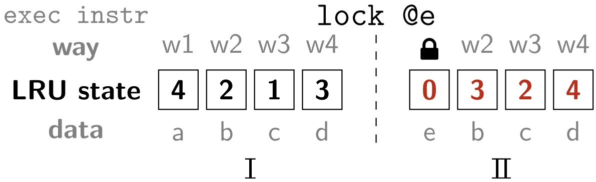
- data can be evicted

Replacement policy update

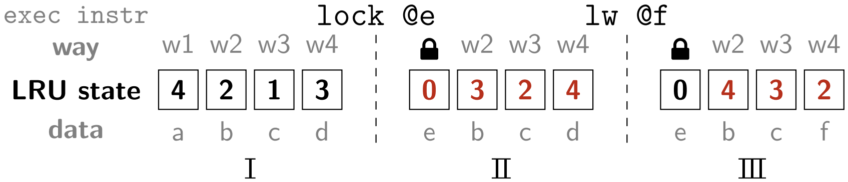
exec instr

way	w1	w2	w3	w4
LRU state	**4**	**2**	**1**	**3**
data	a	b	c	d

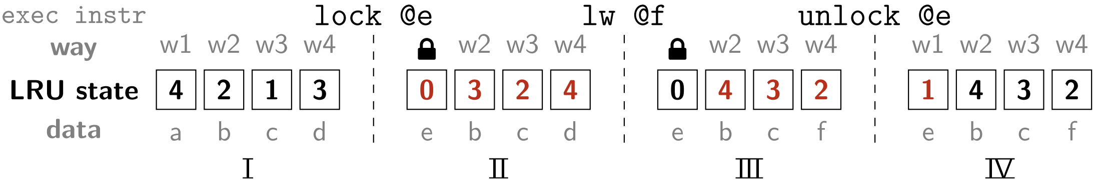
Replacement policy update



Replacement policy update



Replacement policy update



Software Implementation

```c
void fct(int* sensitive_table, int* input){
  //lock phase
  for(int i=0; i<sizeof(sensitive_table); i+=16)
        __LOCK(&sensitive_table, i);

  //algo accessing table depending on secret
  algo(sensitive_table, input);

  //unlock phase
  for(int i=0; i<sizeof(sensitive_table); i+=16)
        __UNLOCK(&sensitive_table, i);
}
```

```asm
c.mv   t4,a4
c.mv   t5,a5
c.add  t4,t5
lock   x0,0(t4)
```

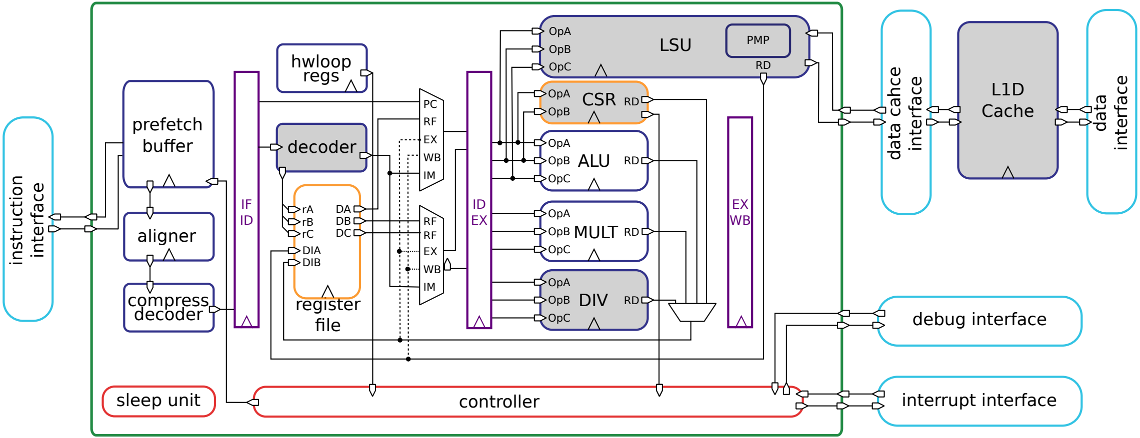
__LOCK macro

Listing 1: Example of use of the cache locking mechanism.

Summary

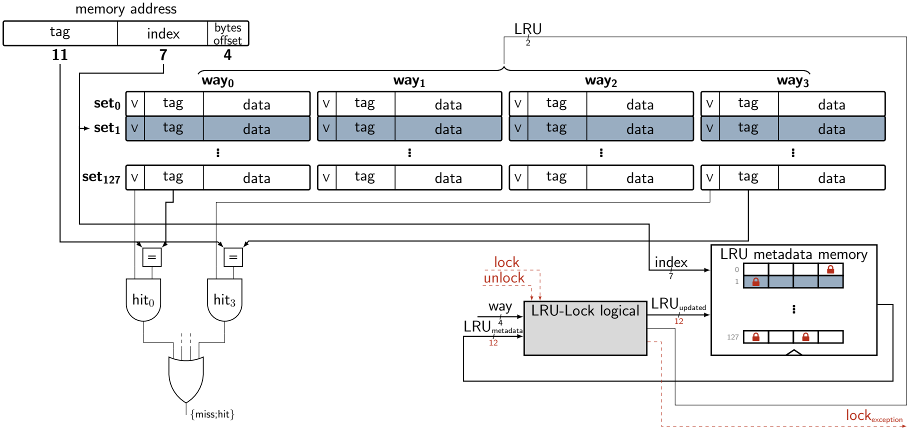
- Context
- State of the art
- Fine-grained locking mechanism
- Implementation with an N-way set associative cache
- Implementation with a randomization-based skewed cache
- Conclusion

Implementation - *core level*



based on the OpenHWgroup CV32E40P core

Implementation - *cache level*



Impact on resource utilization

Post implementation area result

	without lock			with lock			Overhead	
	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF
↪ LRU	26	24	0,5	50	34	0,5	+ 92,31%	+ 41,67%
↪ Cache	980	1 065	8,5	1 007	1 077	8,5	**+ 2,76%**	**+ 1,1%**
↪ Core	4 669	2 233	0	4 666	2 235	0	- 0,06%	+ 0,09%
CPU	5 661	3 467	8,5	5 683	3 481	8,5	**+ 0,39%**	**+ 0,40%**

Considering AMD/Xilinx Kintex-7 FPGA family with Vivado 2022.2

Impact on resource utilization

Post implementation area result

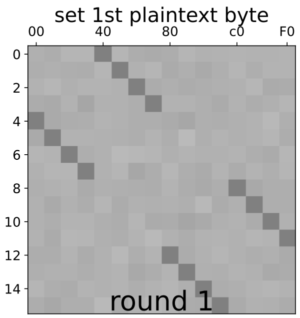
	without lock			with lock			Overhead	
	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF
↪ LRU	26	24	0,5	50	34	0,5	+ 92,31%	+ 41,67%
↪ Cache	980	1 065	8,5	1 007	1 077	8,5	**+ 2,76%**	**+ 1,1%**
↪ Core	4 669	2 233	0	4 666	2 235	0	- 0,06%	+ 0,09%
CPU	5 661	3 467	8,5	5 683	3 481	8,5	**+ 0,39%**	**+ 0,40%**

Considering AMD/Xilinx Kintex-7 FPGA family with Vivado 2022.2

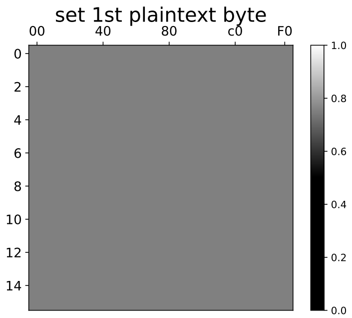
BRAM' Bits overhead

At cache level, baseline stores **72,704 bits** (cache lines + LRU metadata) + **512 bits** to implement our lock mechanism ▸ overhead of 0.7%.

Impact on Security - *considering* PRIME+PROBE *on AES-128*



Unprotected



using `lock`

Takeaway

The number of locked cache lines does not have to depend on secret.

Impact on Performances - *using `lock`*

Binary code size overhead

- AES-128 => **0,28%**
- Camellia => **0,23%**

Overhead induced by the insertion of `lock` and `unlock` instructions.

Impact on Performances - *using* `lock`

Binary code size overhead

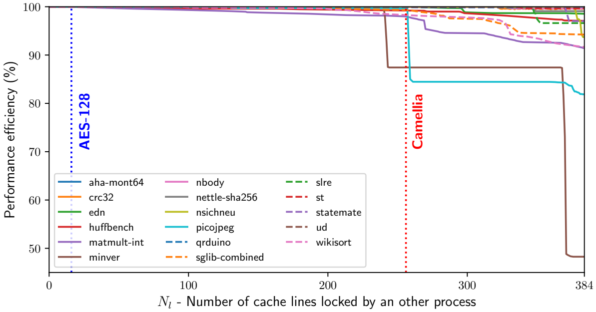
- AES-128 => **0,28%**
- Camellia => **0,23%**

Overhead induced by the insertion of `lock` and `unlock` instructions.

Execution time overhead (+%)

N_{Blocks}	1	4	8	16	64	128	512	1024
Camellia	367,7	99,6	54,22	28,88	7,62	3,85	0,97	0,48
AES-128	2,77	0,71	0,35	0,18	0,04	0,02	-	-

Impact on Performances - *on Embench-IoT benchmark*



Takeaway

AES-128 and Camellia have a **negligeable impact** on Embench-IoT.

Implementation Synthesis

At a glance :

- Locking mechanism implementation implies a **low area** overhead (<3% on cache)
- **Low** (negligeable) **impact** on overall performance

- Locking mechanism provides a **fine-grained** efficient and **on-demand security** against timing SCAs

Implementation Synthesis

At a glance :

- Locking mechanism implementation implies a **low area** overhead (<3% on cache)
- **Low** (negligeable) **impact** on overall performance

- Locking mechanism provides a **fine-grained** efficient and **on-demand security** against timing SCAs

Can a combination with a random skewed cache push back the limits ?

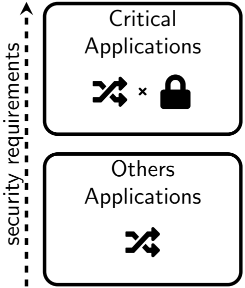
- Identify the exact number of locked cache lines
- Limit usage of lock
- Renew keys frequently to maintain security

Summary

- Context
- State of the art
- Fine-grained locking mechanism
- Implementation with an N-way set associative cache
- Implementation with a randomization-based skewed cache
- Conclusion

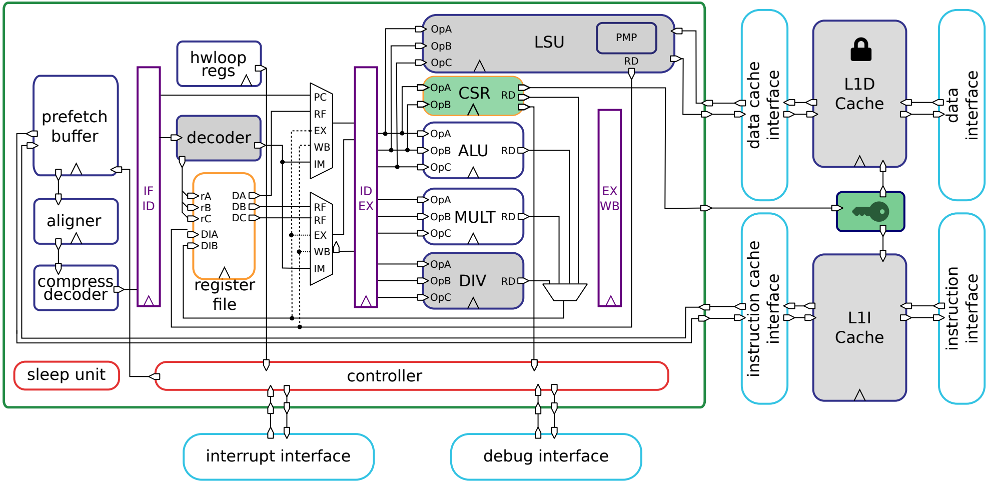
Goal of this implementation

- **Prevent** from a **new disruptive attack** to build eviction set (as PRIME+PRUNE+PROBE releases)
- Provide security guarantees while an eviction set is up
  - keep critical applications unvulnerable
  - avoid attacker infering number of locked cache lines
- Evaluate **hardware resource overhead** to implement such a system



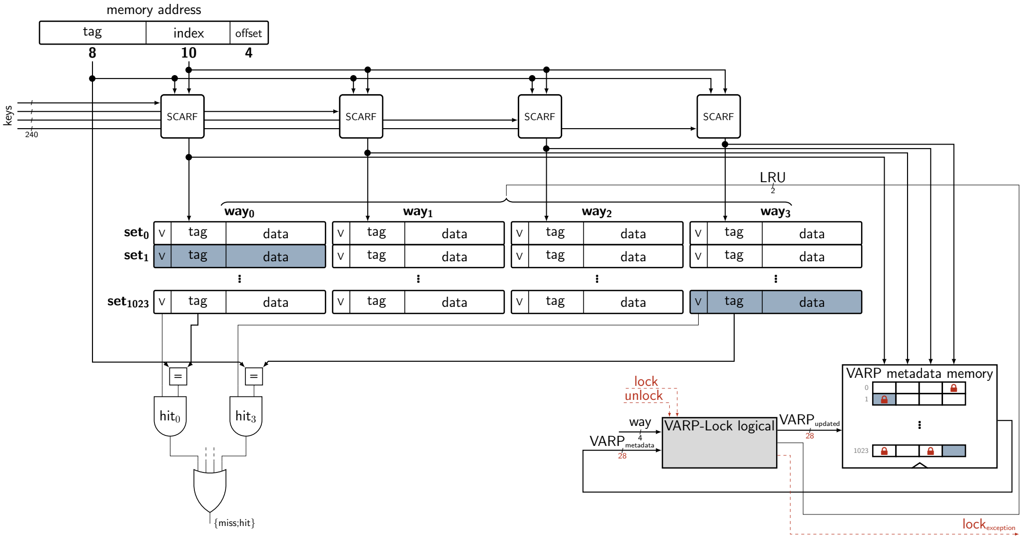
maintains security for 6.8 M of memory accesses against P+P+P [19]

Implementation - *core level*



based on the OpenHWgroup CV32E40P core

Implementation - *cache level*



Upgrades compared with the lock only implementation

Integration of locking mechanism on such a cache system :

- **forbid lock on the last way**
  - keep the cache usable
  - avoid bypass

Integration of **keys renewing module** :

- multiple LFSRs to generate keys
- manage instruction and data caches keys
- invalidate and unlock the whole cache

Post implementation area results

	skewed alone			**skewed with lock**			Overhead	
	LUT	FF	BRAM	LUT	FF	BRAM	LUT	FF
↪ VARP-64	88	85	2	135	138	2	+ 53,41%	+ 62,35%
↪ Data Cache	4 264	2 287	18	4 313	2 342	18	+ 1,15%	+ 2,30%
↪ VARP-64	100	85	2	99	85	2	- 1,0%	0,00%
↪ Instr Cache	3 214	2 178	18	3 212	2 178	18	- 0,06%	0,00%
↪ Key update	628	3 490	0	628	3 490	0	0,00%	0,00%
↪ Core	4 884	2 310	0	4 862	2 310	0	- 0,45%	0,00%
CPU	13 044	10 561	36	13 070	10 616	36	+ 0,20%	+ 0,52%

Considering AMD/Xilinx Kintex-7 FPGA family with Vivado 2022.2

Implementation Synthesis

At a glance :

- Provide **security guarantee** for critical applications
  - Skewed random cache for overall applications
  - Locking mechanism reserved for critical applications
- **Critical applications remain immune** against known timing based CSCAs

- This implementation allows to defend against **new technique** to build eviction set

- Locking mechanism implementation implies a **low area** overhead (<2.5% on cache)

Summary

- Context
- State of the art
- Fine-grained locking mechanism
- Implementation with an N-way set associative cache
- Implementation with a randomization-based skewed cache
- Conclusion

Conclusion

- We extend the RISC-V ISA to :
  - **guarantee cache hit** for memory accesses on locked data
  - mitigate Evict + Time, Prime+Probe

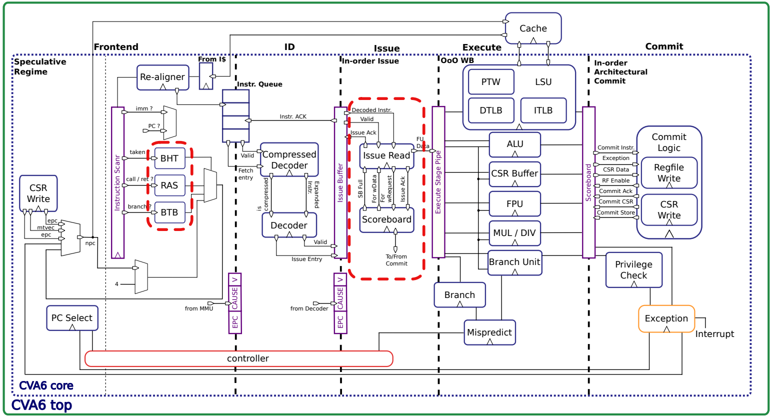
- The locking mechanism implementation implies a **low area** overhead (<3%)
- **Low** (negligeable) **impact** on overall performance

- The locking mechanism provides a **fine-grained** efficient and **on-demand security** against timing SCAs

- We demonstrate the light cost of locking in skewed implementation to harden security for critical application

Perspectives - Axe ①

- Study impact of the microarchitecture

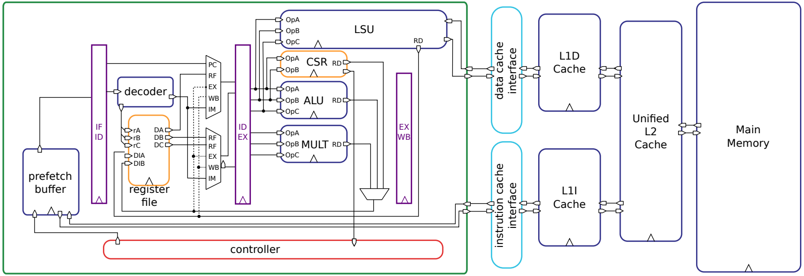


CVA6 Block diagram - *sources: CVA6 & Kévin Q.*

- Force the use of `fence` instruction on out of order
- Speculative execution of `unlock` on locked data

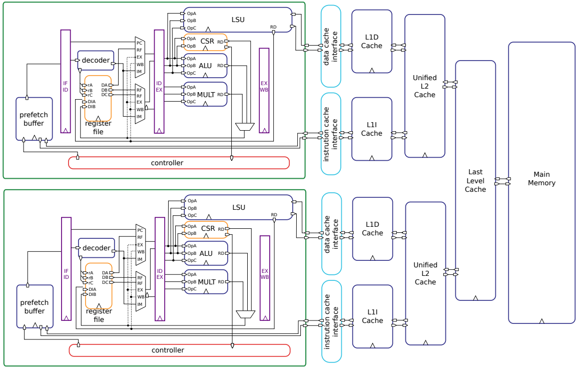
Perspectives - Axe ②

- Manage locking mechanism on more complex CPU
  - Extend the cache memory hierarchy



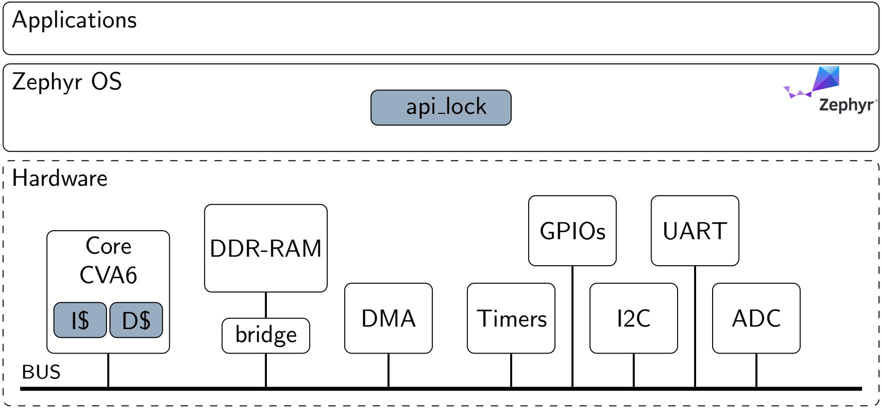
Perspectives - Axe ②

- Manage locking mechanism on more complex CPU
  - Extend the cache memory hierarchy
  - Add cores



Perspectives - Axe ③

- Operating system support
  - LockOS project



Fine-grained dynamic partitioning against cache-based side channel attacks

Nicolas Gaudin, **Vianney Lapôtre**, Guy Gogniat, Pascal Cotret

journées nationales 2025 du GDR Sécurité Informatique

Bibliography

[1] H. Winderix, J. T. Mühlberg, and F. Piessens, "Compiler-assisted hardening of embedded software against interrupt latency side-channel attacks," in *Proc. IEEE European Symposium on Security and Privacy (EuroS&P)*, Sep. 2021. DOI: 10.1109/EuroSP51992.2021.00050.

[2] Z. Zhou, M. K. Reiter, and Y. Zhang, "A software approach to defeating side channels in last-level caches," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016. DOI: 10.1145/2976749.2978324.

[3] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters," in *Proc. International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2018. DOI: 10.1145/3214292.3214293.

Bibliography

[4] M. Mushtaq et al., "Whisper: A tool for run-time detection of side-channel attacks," *IEEE Access*, 2020. DOI: 10.1109/ACCESS.2020.2988370.

[5] B. C. Schwedock, P. Yoovidhya, J. Seibert, and N. Beckmann, "Täkō: A polymorphic cache hierarchy for general-purpose optimization of data movement," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2022. DOI: 10.1145/3470496.3527379.

[6] Z. Wang and R. 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.

[7] M. K. Qureshi, "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.

Bibliography

[8] A. Jaamoum, T. Hiscock, and G. D. Natale, "Scramble cache: An efficient cache architecture for randomized set permutation," in *Proc. Design, Automation & Test in Europe Conference (DATE)*, 2021. DOI: 10.23919/DATE51398.2021.9473919.

[9] A. Purnal, L. Giner, D. Gruss, and I. Verbauwhede, "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.

[10] M. K. Qureshi, "New attacks and defense for encrypted-address cache," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2019. DOI: 10.1145/3307650.3322246.

Bibliography

[11] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: Thwarting cache attacks via cache set randomization," in *Proc. 28th USENIX Security Symposium (USENIX Security)*, 2019. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/werner.

[12] J. P. Thoma et al., "Clepsydracache - preventing cache attacks with time-based evictions," in *Proc. 32th USENIX Security Symposium (USENIX Security)*, 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/thoma.

[13] Y. Ye, R. West, Z. Cheng, and Y. Li, "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.

Bibliography

[14] F. Yao, H. Fang, M. Doroslovački, and G. Venkataramani, "Cotsknight: Practical defense against cache timing channel attacks using cache monitoring and partitioning technologies," in *Proc. International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2019. DOI: 10.1109/HST.2019.8740835.

[15] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *ACM Transactions on Architecture and Code Optimization*, Jan. 2012. DOI: 10.1145/2086696.2086714.

[16] Y. Wang, A. Ferraiuolo, D. Zhang, A. C. Myers, and G. E. Suh, "Secdcp: Secure dynamic cache partitioning for efficient timing channel protection," in *Proc. Design Automation Conference (DAC)*, 2016. DOI: 10.1145/2897937.2898086.

Bibliography

[17] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and efficient fine-grain cache partitioning," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2011. DOI: 10.1145/2000064.2000073.

[18] W. Xiong, S. Katzenbeisser, and J. Szefer, "Leaking information through cache lru states in commercial processors and secure caches," *IEEE Transactions on Computers*, vol. 70, no. 4, pp. 511–523, 2021. DOI: 10.1109/TC.2021.3059531.

[19] M. Peters et al., "On the effect of replacement policies on the security of randomized cache architectures," in *Proc. ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, 2024, pp. 483–497. DOI: 10.1145/3634737.3637677.