IE-Cache
Counteracting Eviction-Based Cache Side Channel Attacks

M. Asim Mukhtar[1] M. Khurram Bhatti[1] Guy Gogniat[2]

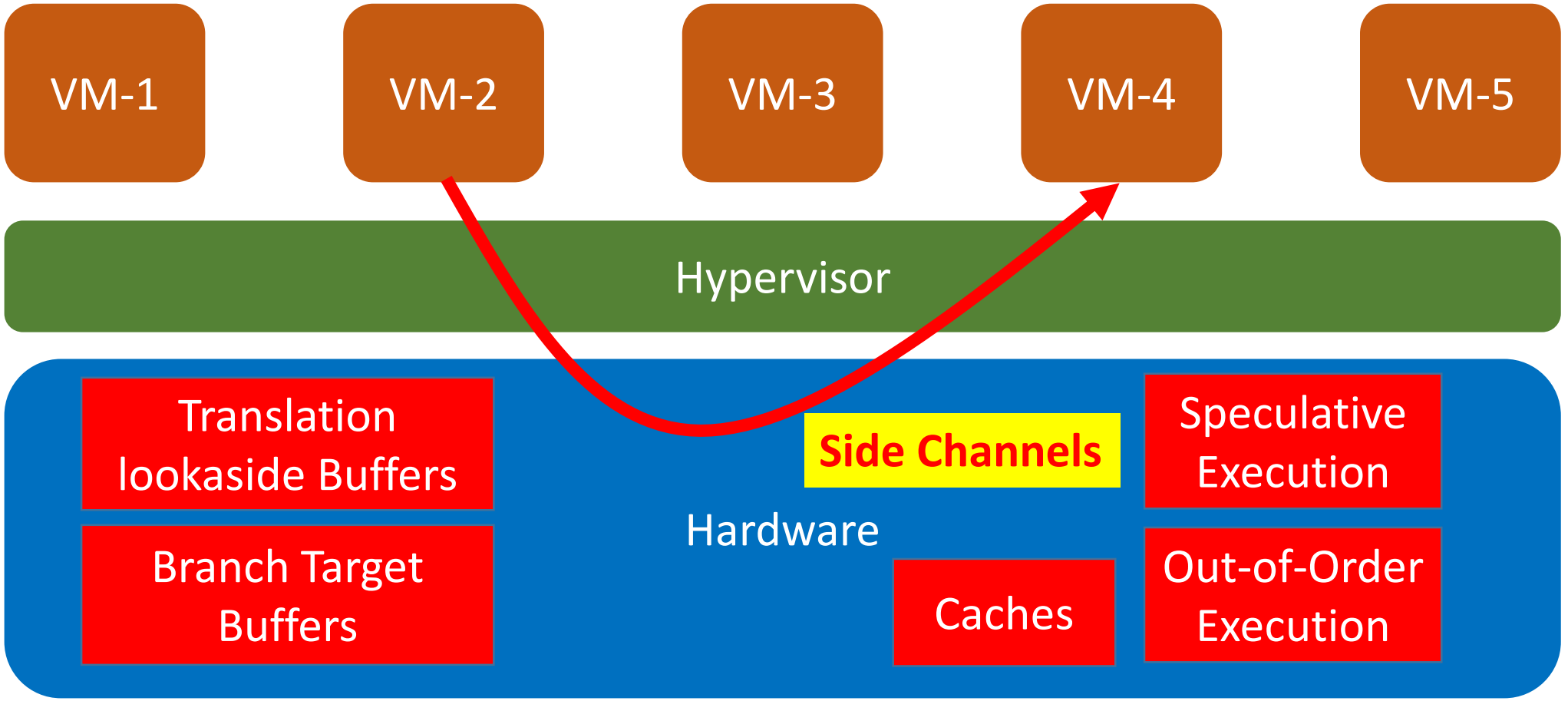
[1] Information Technology University, Lahore, Pakistan

[2] University of South Brittany, Lorient, France

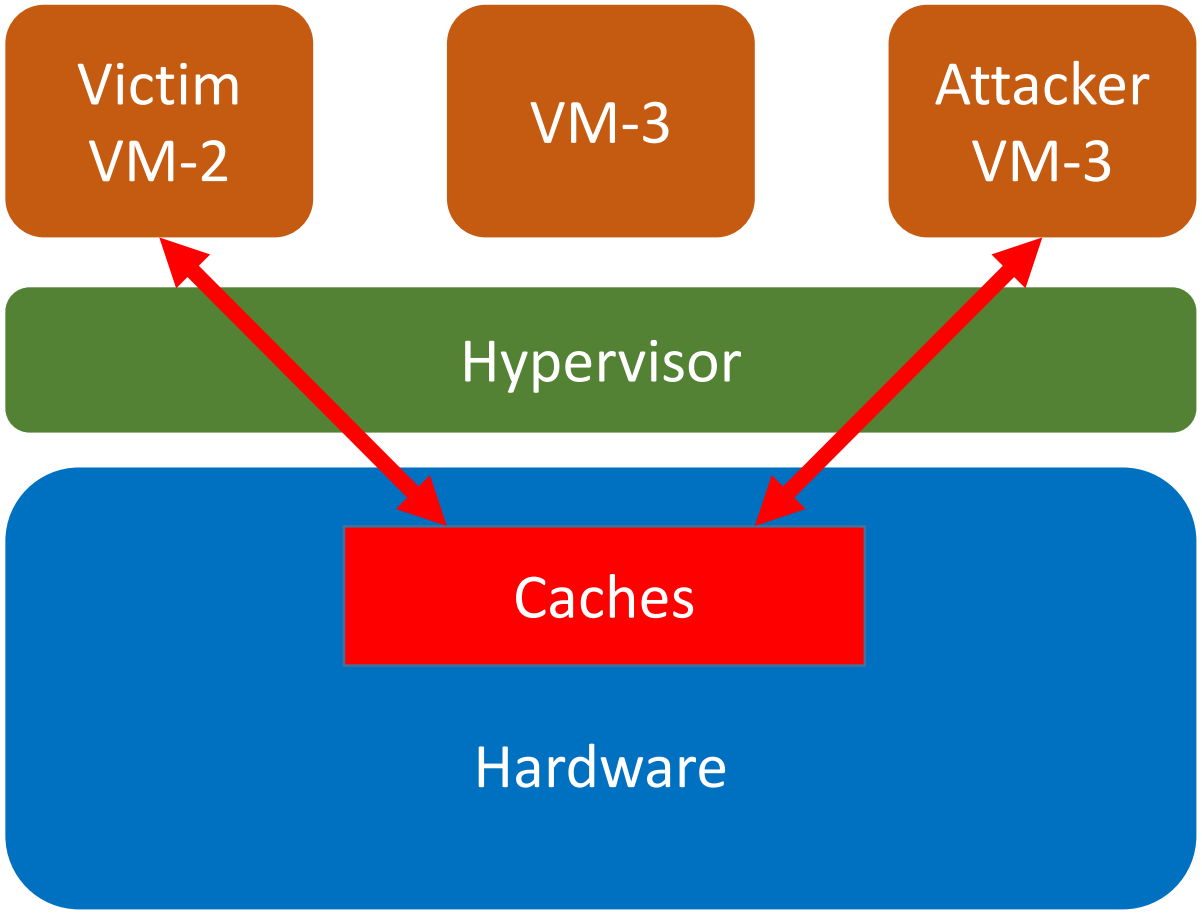
Outline

- Security Issue
- Background
- Prime+Probe Attack
- Prior countermeasures their Limitations
- Our Proposed Countermeasures
- IE-Cache
- Security and Performance Results

Security Issue



Security Issue

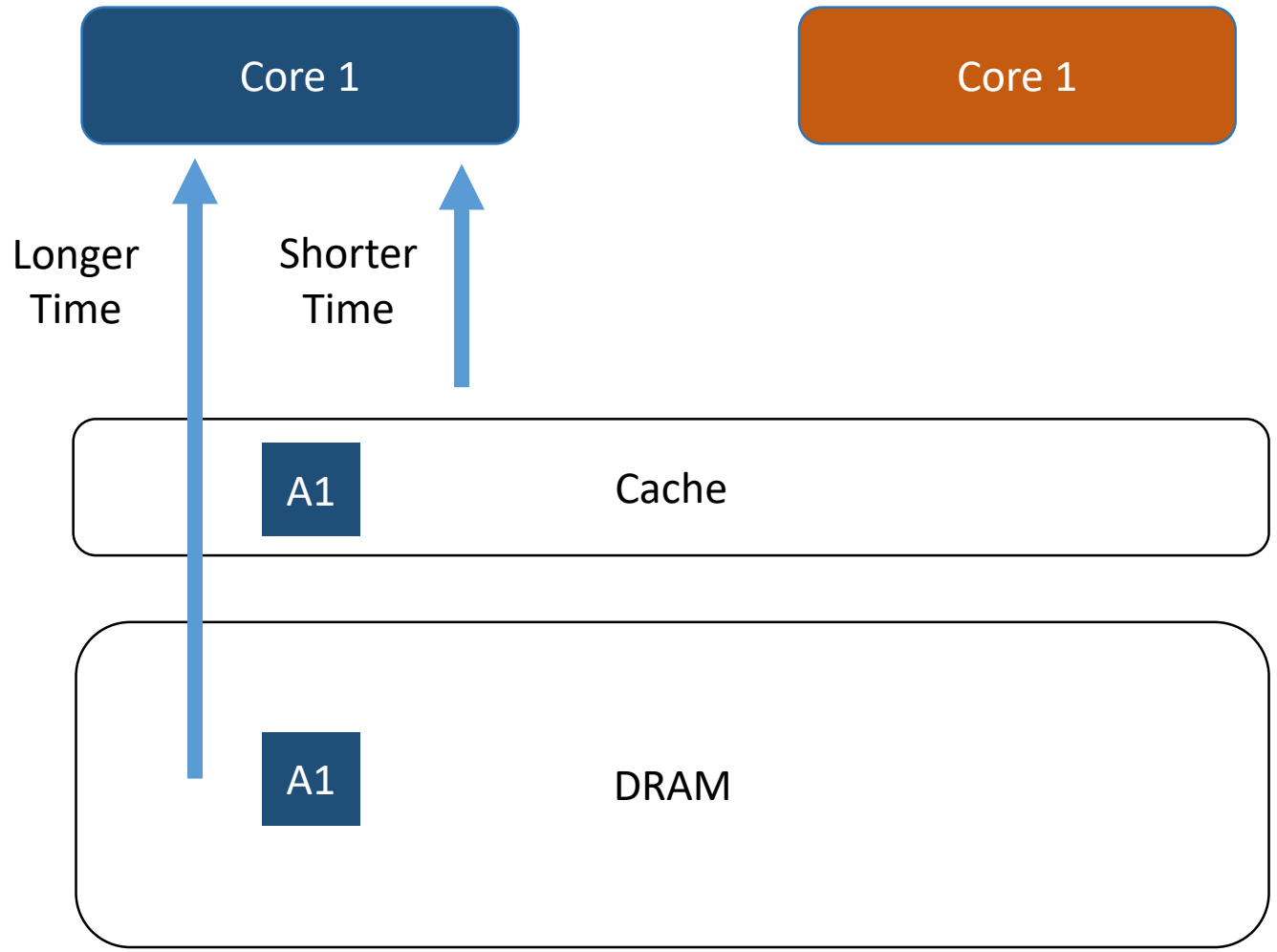


Cache-based side-channel attacks [M. Werner, USENIX Security 2019]

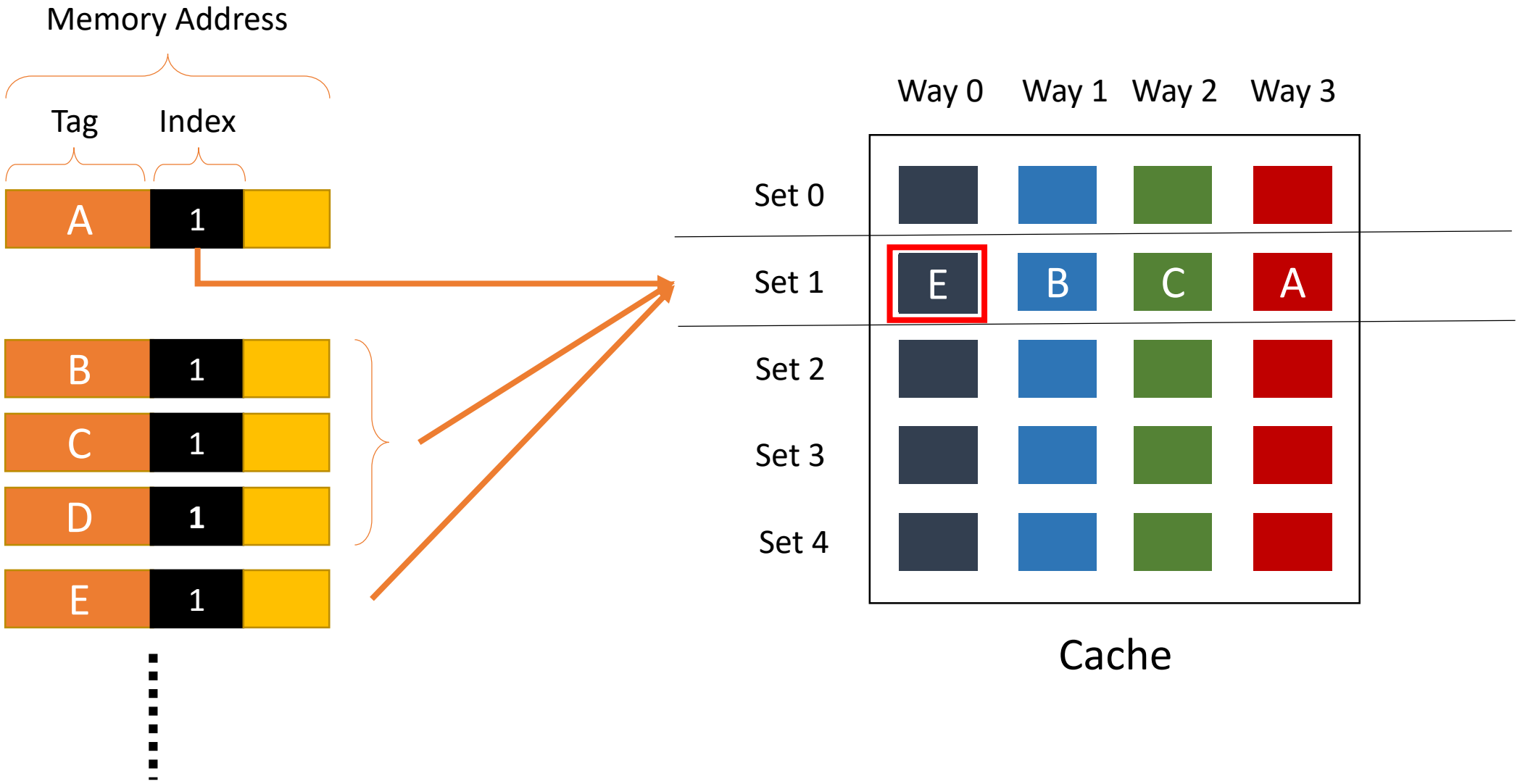
- Extracting keys of cryptographic algorithms (RSA, AES, etc).
- Monitoring keystrokes.
- Reading unauthorized address space.

Background: Caches

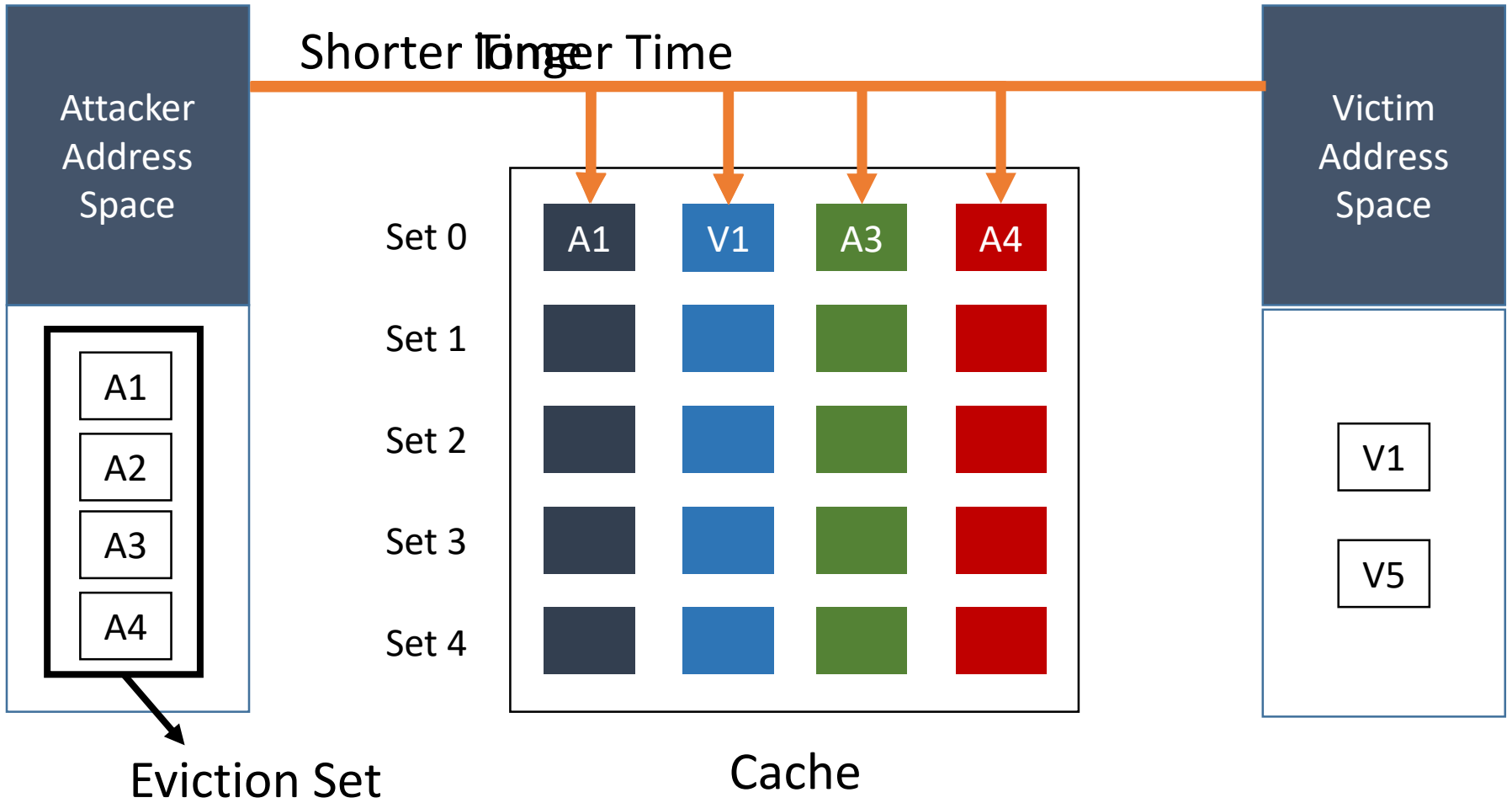
Again Generate Memory Access **A1**



Background: Cache structure and Mapping



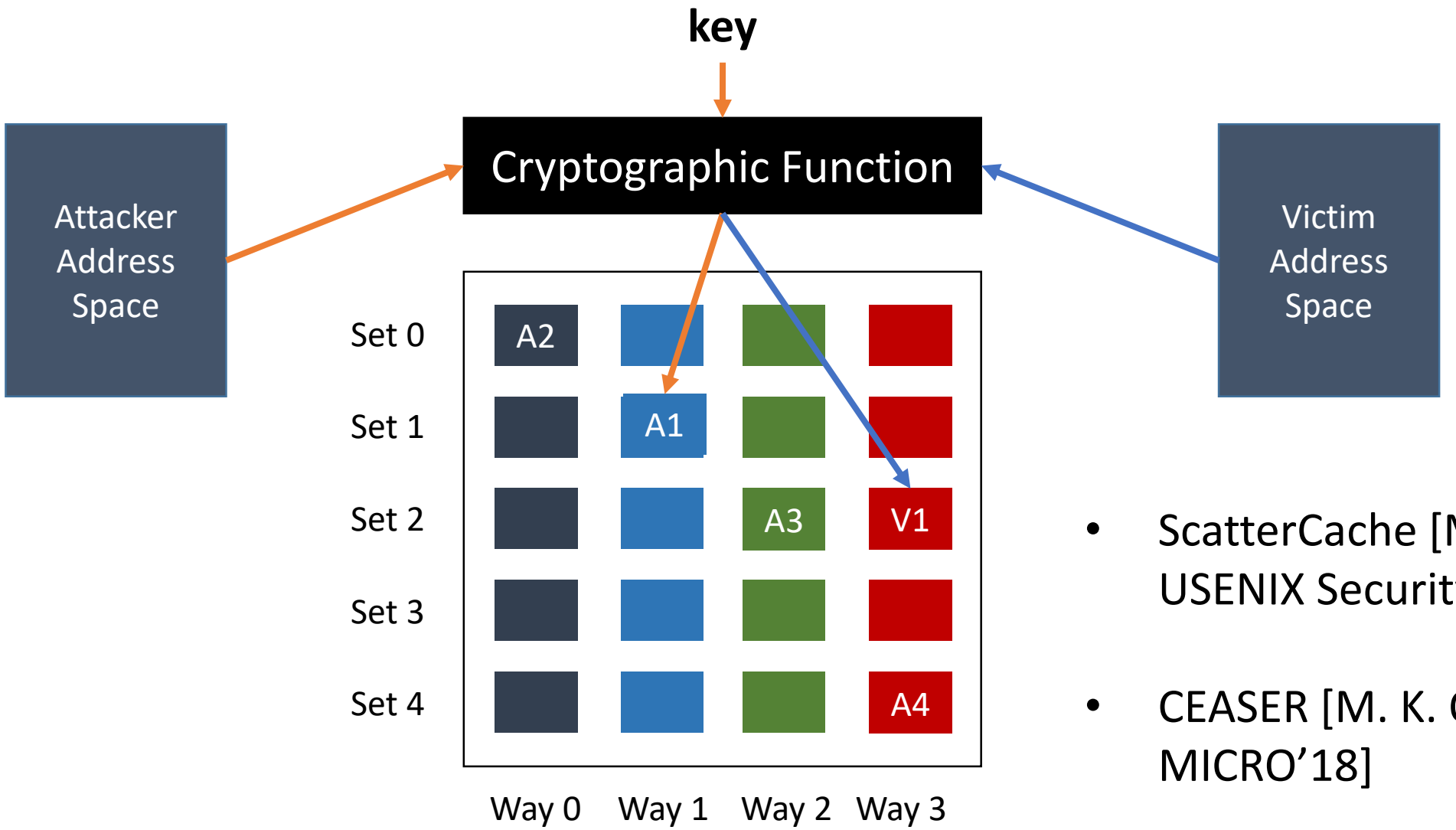
Eviction based Cache Side Channel Attacks: Prime+Probe Attack



Victim Code

```
If (secret == 1)
    func(V1);
else
    func(V5);
```

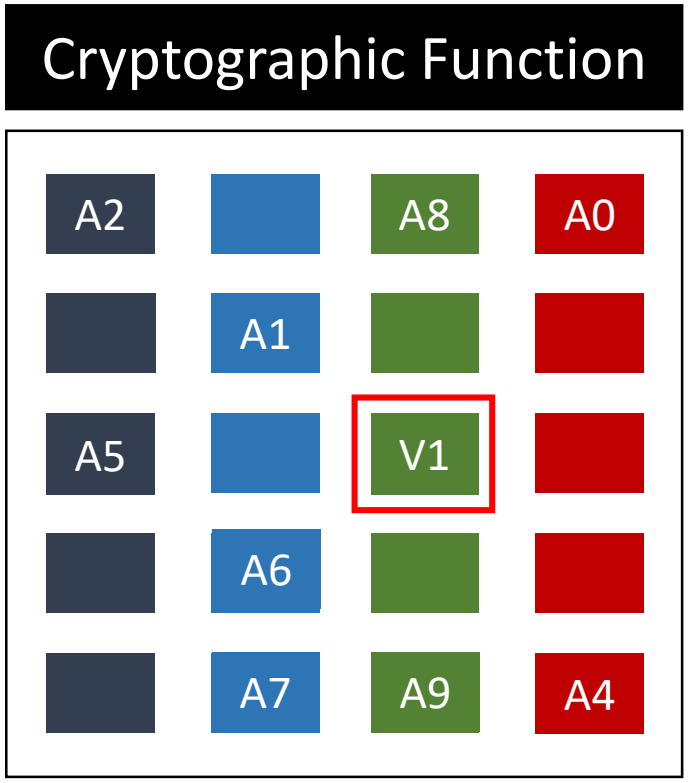
Prior Countermeasures



- ScatterCache [M. Werner, USENIX Security 2019]
- CEASER [M. K. Qureshi, MICRO'18]

Limitation in ScatterCache and CEASER

Prime+Prune+Probe technique can reveal eviction sets [A. Purnal et al. , S&P' 2020]



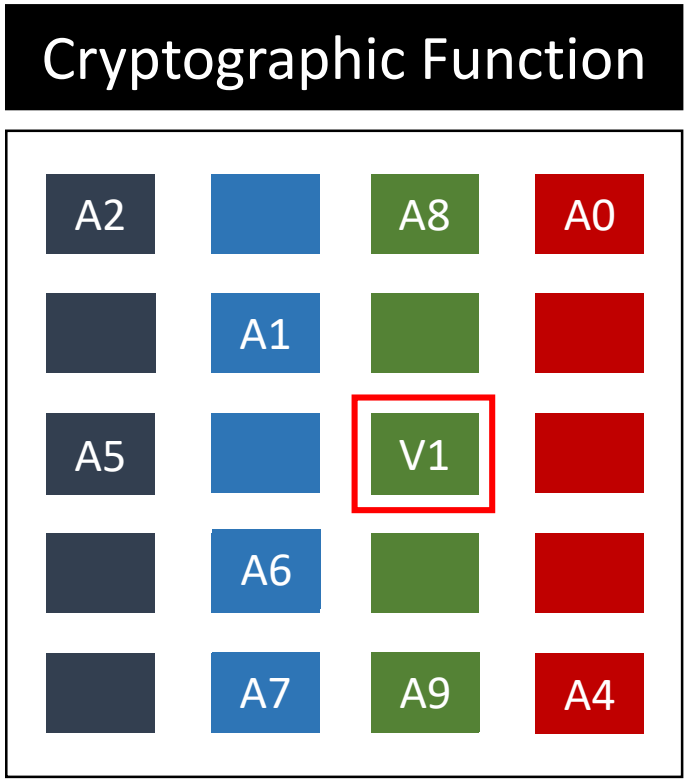
Cache

1) Prime - Attacker randomly chooses memory addresses and places them in cache
   (let us say attacker accesses A0 to A9)
2) Prune - Attacker ensures that all accessed addresses are in cache by re-accessing.
3) Call the victim to execute
   (victim accesses V1)
4) Probe - Attacker accesses again all addresses and observes access latency
   (A3 is a member of eviction set)

This technique can find eviction set in 4 seconds in ScatterCache [A. Purnal et al. , S&P' 2020]

Limitation in ScatterCache and CEASER

Prime+Prune+Probe technique can reveal eviction sets [A. Purnal et al. , S&P' 2020]



Cache

1) Prime - Attacker randomly chooses memory addresses and places them in cache
   (let us say attacker accesses A0 to A9)
2) Prune - Attacker ensures that all accessed addresses are in cache by re-accessing.
3) Call the victim to execute
   (victim accesses V1)
4) Probe - Attacker accesses again all addresses and observes access latency
   (A3 is a member of eviction set)

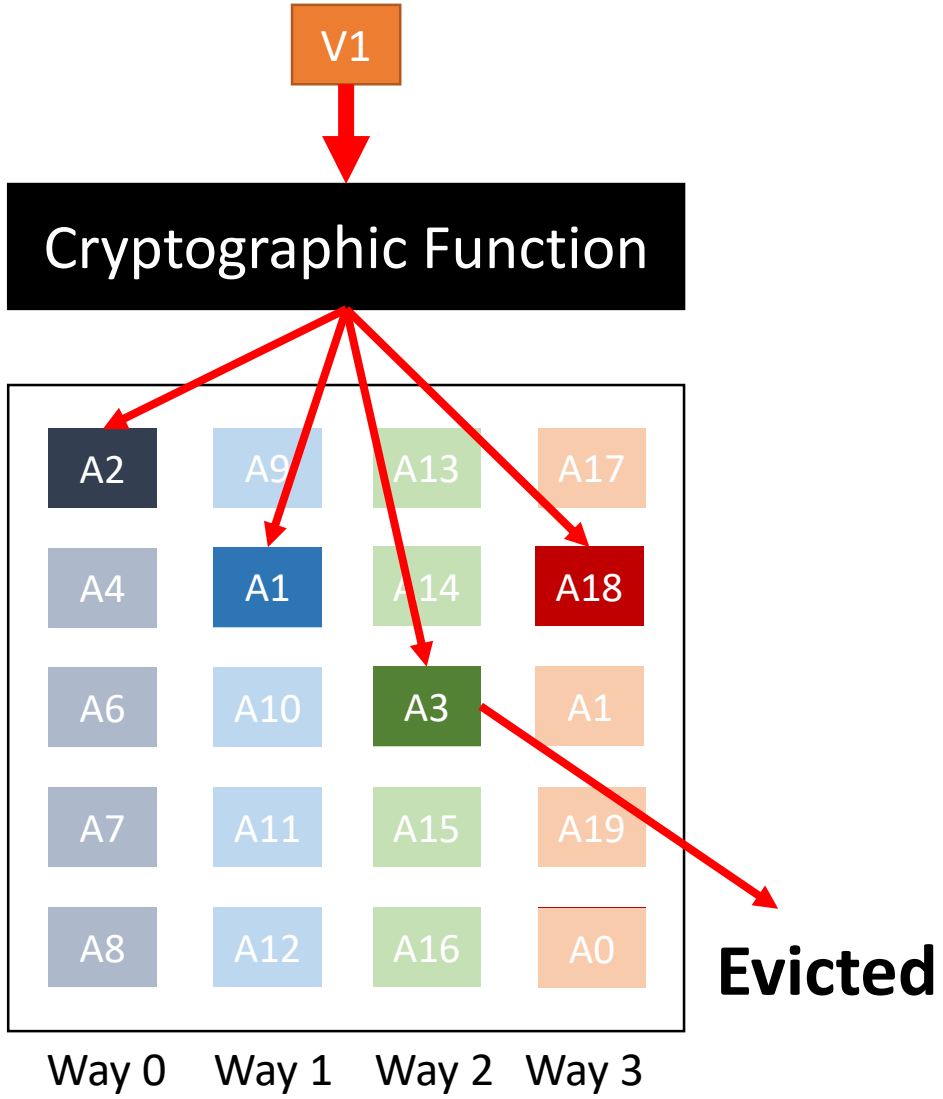
This technique can find eviction set in 50 seconds in CEASER [A. Purnal et al. , S&P' 2020]

Our Proposed Countermeasures

- Random Replacement Policy based Caches
  - IE-Cache : Indirect Eviction Cache
  - PCache: Permutation and Indirect Eviction based Cache
  - OE-Cache: Fully Associative cache using Indirect Eviction

- Least Recently used based Cache
  - 3D-Cache: eviction of least recently used cache line form randomly selected cache lines using indirect eviction.

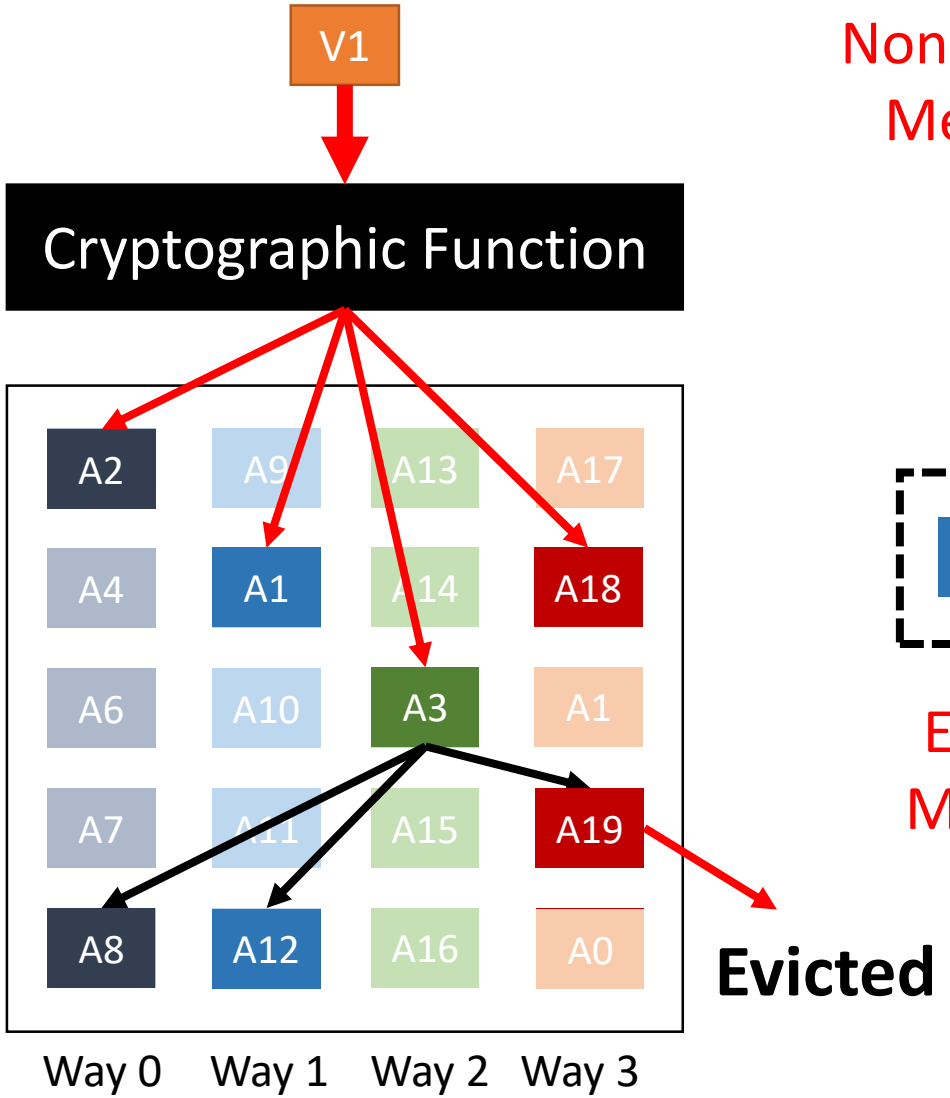
Our Perspective About Problem



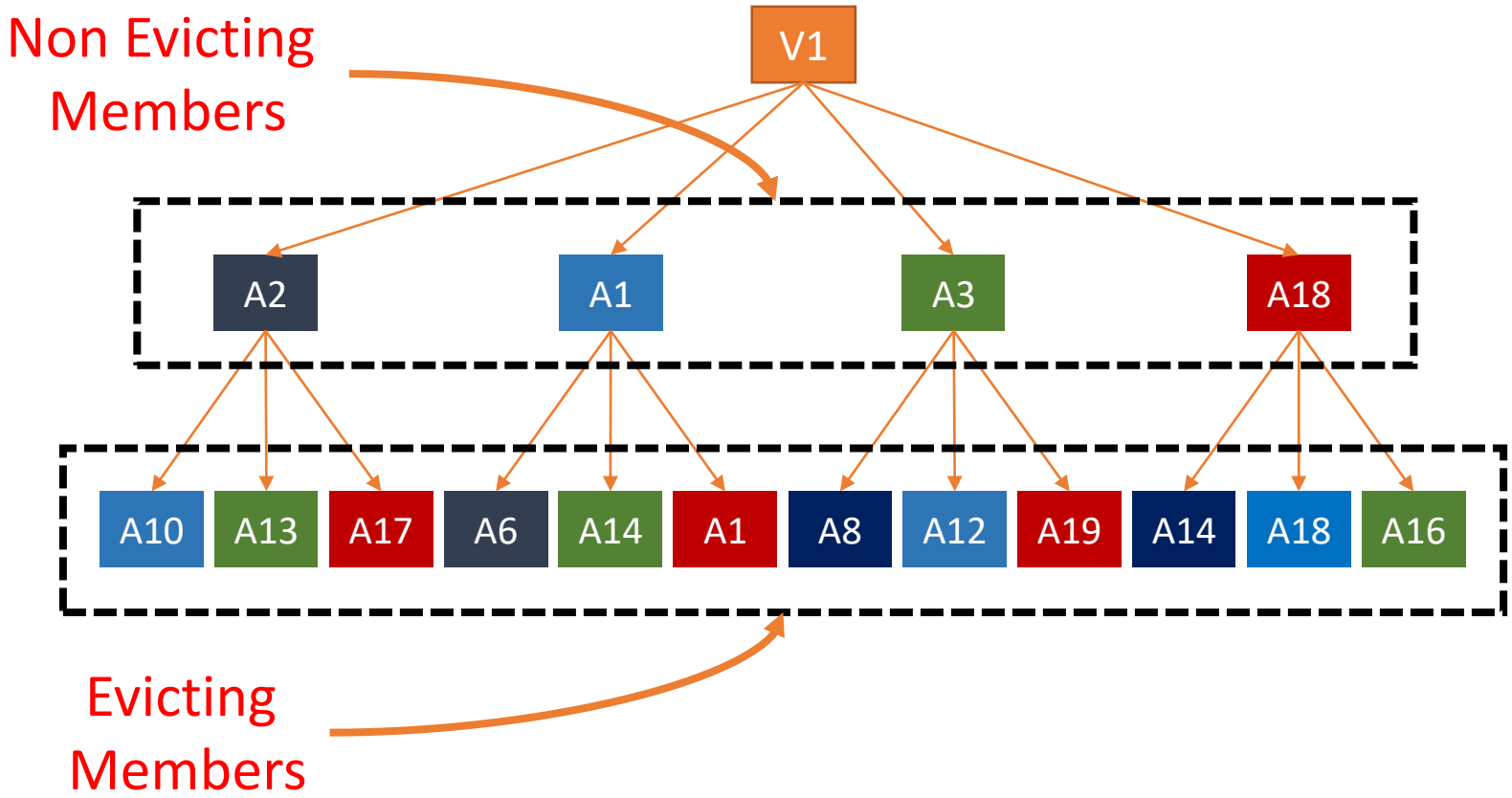
- Direct Relation Problem

- Our Hypothesis
  - Indirect Eviction of cache line will make Prime+Prune+Probe attack impractical

Proposed Solution : IE-Cache

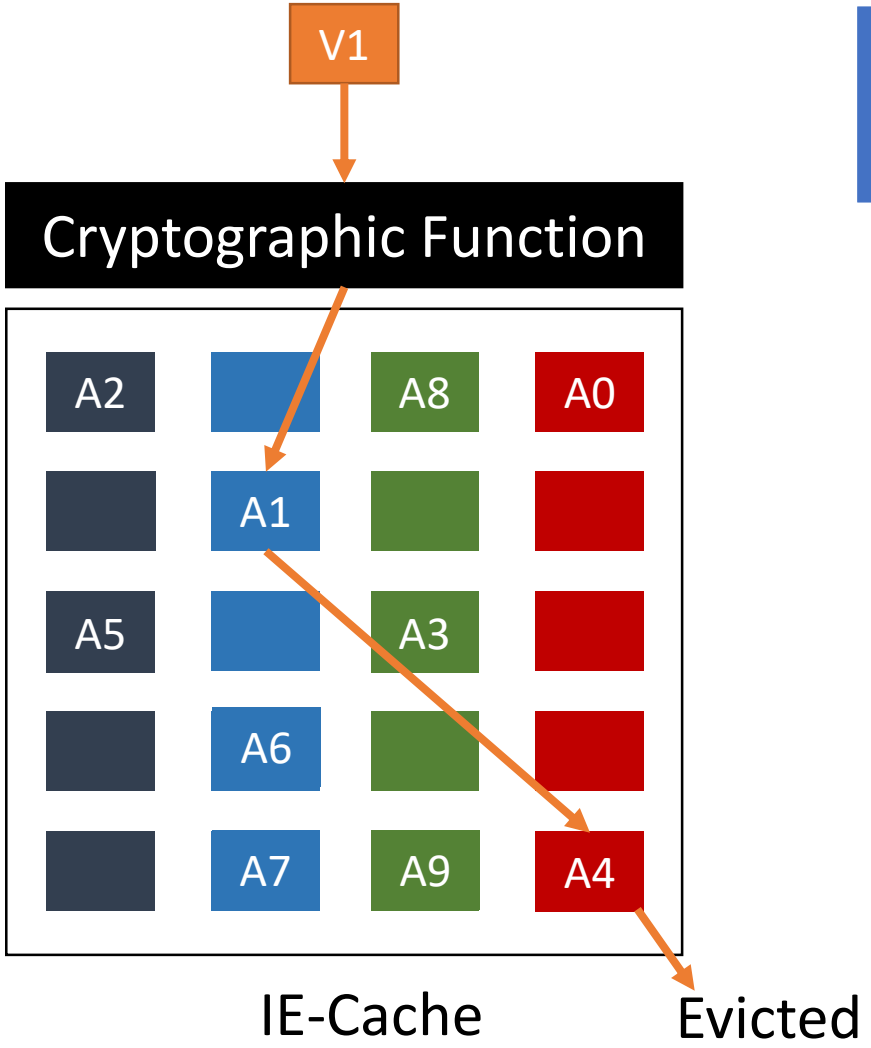


Evicted



Non-Evicting cache lines are impractical to find using Prime+Prune+Probe attack

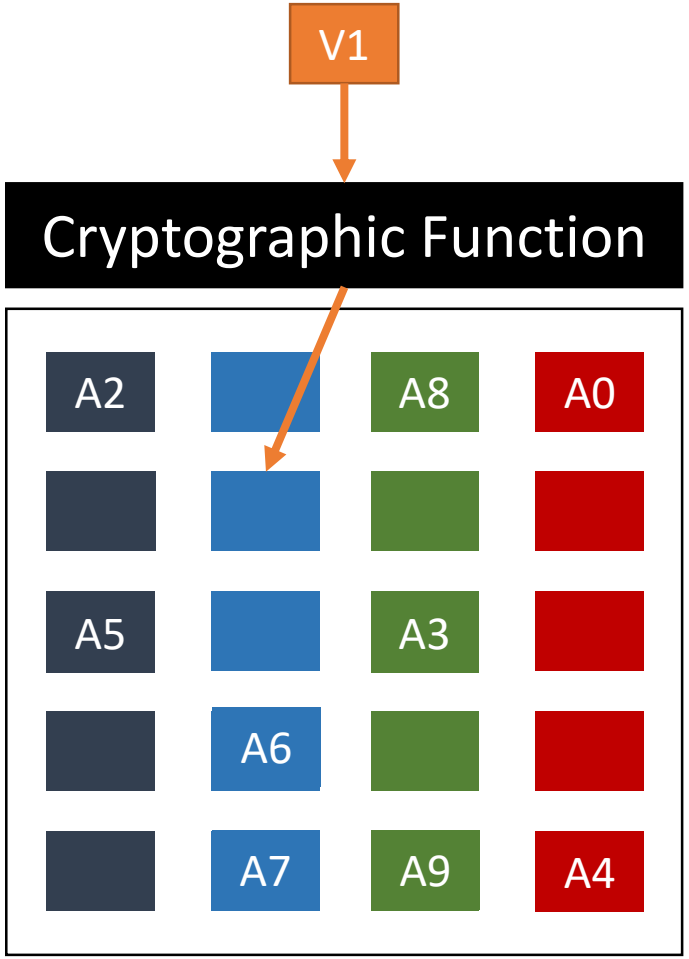
Lets Break IE-Cache: Breaking-Branch Technique



First use Prime+Prune+Probe technique to find evicting members

1) Attacker randomly chooses memory addresses and places them in cache
   (let us say attacker accesses A0 to A9)
2) Attacker ensures that all accessed addresses are in cache by re-accessing.
3) Call the victim to execute
   (victim accesses V1)
4) Attacker accesses again all addresses and observes access latency
   (A4 is a member of eviction set)

Breaking-Branch Technique



IE-Cache

Break Branch to find Non-Evicting Member

1) Attacker again accesses addresses except one
   (let us say attacker accesses all except A1)
2) Attacker ensures that all accessed addresses are in cache by re-accessing.
3) Call the victim to execute
   (victim accesses V1)
4) Attacker accesses again all addresses and observes access latency of A4
   (If A4 does not evict then A1 is the non-evicting member)

Security Evaluation

- We build functional model of IE-Cache using Python.

- We extracted victim and attacker access to find 1000 evicting and non-evicting members using Prime+Prune+Probe and breaking-branch technique. Then, we have averaged 1000 samples to form finding time of one member of eviction set.

- For time analysis, we used following data [A. Purnal et al. , S&P' 2020]
  - cache hit time = 9.5ns,
  - cache miss time = 50ns ,
  - victim execution time = 0.5ms and
  - cache flush time = 3.6ms.

Time Analysis of Finding Evicting Members

Cache Configuration: 2^{11} cache lines, 4 ways and 2 levels

Group Size	Victim Accesses	Attacker Accesses	Time (ms) per Evicting Member	Time (hr) Evicting Members of Eviction set
7000	80.07	1.95E+08	7.16	22
6000	64.12	1.08E+08	4.23	13
5000	442.70	5.76E+07	2.27	7
4000	30143.22	1.65E+09	69.33	213

Increased from 50 seconds to 7 hours

For 3 levels, time to find non-evicting members is 191 hours

Time Analysis of Finding Non-Evicting Members

Cache Configuration: 2^{11} cache lines, 4 ways and 2 levels

Group Size	Victim Accesses	Attacker Accesses	Time (months) per Non-Evicting Member	Time (years) Non-Evicting Members of Eviction set
7000	3.95E+10	1.67E+14	58.37674	311.3426
6000	3.59E+10	5.20E+10	52.73438	281.25
5000	8.47E+08	1.83E+10	1.291233	6.886574
4000	1.69E+08	4.41E+09	0.279948	1.493056

6.8 years required to find one eviction set. Usually attacker requires more than one eviction set to launch attack.

For 3 levels, time to find evicting members is 100+ years

IE-Cache: Prime+Probe Complexity

Way	Levels	Non-Evicting Members	Evicting Members	Memory accesses required in Prime or Probe phase	Increased compared to w/o indirection	Capacity required in main memory by an attacker
4	2	64	3072	3136	×49	196 KB
	3	3072	147456	150528	×2352	9.1 MB
8	2	272	64736	65008	×239	3.9 MB
	3	65008	15407168	15472176	×56883	944.3 MB
16	2	1136	1209840	1210976	×1066	73.9 MB
	3	1209840	1288479600	1289690576	×1135291	76.8 GB
32	2	4640	20856800	20861440	×4496	1.2 GB
	3	20856800	93751316000	93772177440	×20209521	5.4 TB

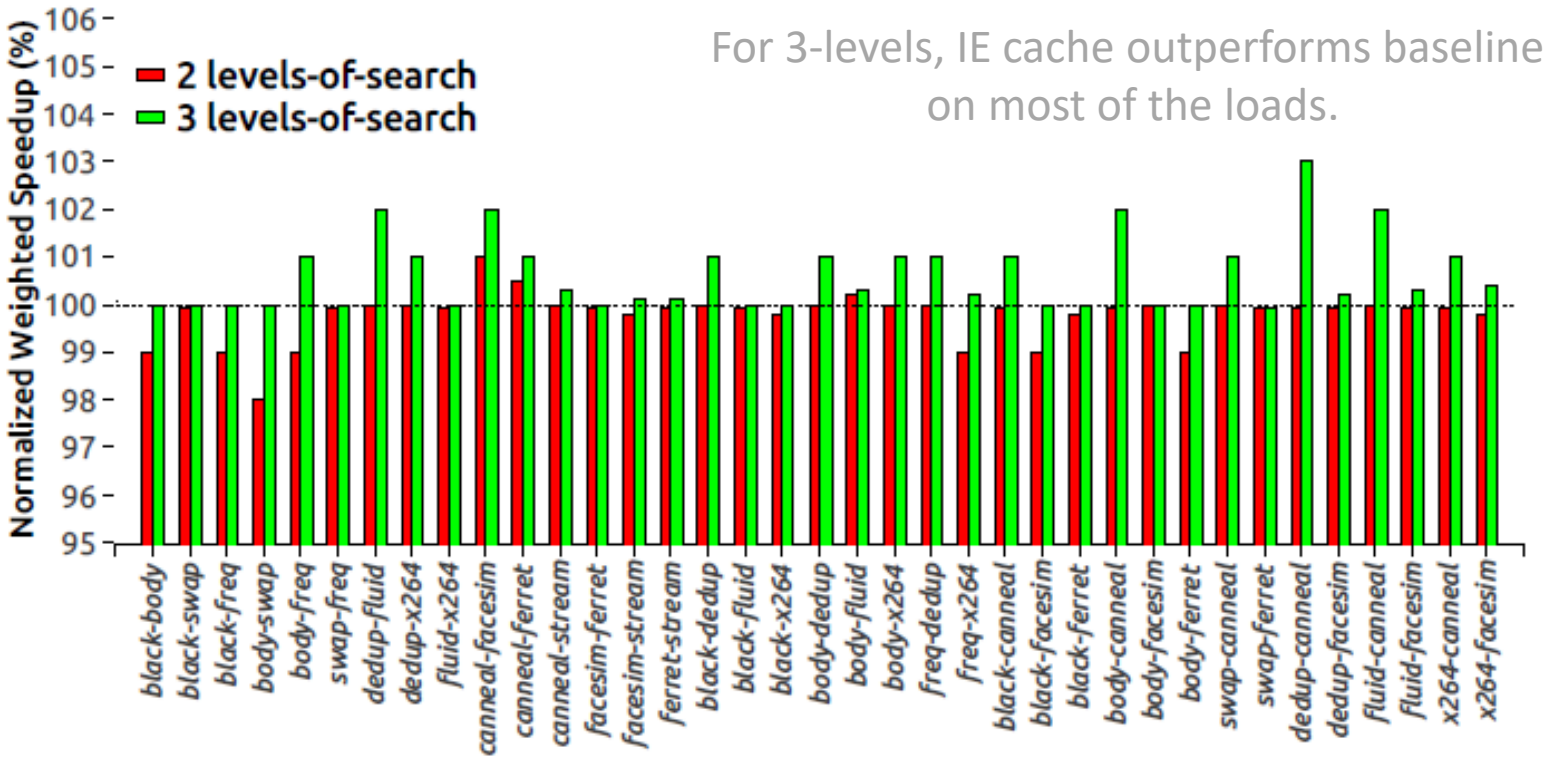
Assumption: Attacker memory address overlap with the evicting lines of a victim address in a single cache way.

Performance Evaluation

- We have build the IE-Cache in Zsim simulator [D. Sanchez ISCA 2013].
- We have used weighted speed-up metric to quantify performance.
- We have normalized 4-way IE-Cache performance relative to baseline architecture.

Baseline Configuration	
Cores	2 cores , 2.2 GHz, OoO model
L1 Cache	Private, 32kB, 8-way set associative, split D/I
L2 Cache	Private, 256kB, 8-way set associative
L3 Cache	Shared, 1MB, 16-way set associative

Performance Results



For 3-levels, IE cache outperforms baseline on most of the loads.

PARSEC Benchmark 3.0

Increasing level of indirection provides high security without compromising the performance.

Conclusion

- Profiling of eviction set becomes impractical if we build cache based on indirect eviction.
- IE-Cache provides both high-security and better performance.
- We have applied indirection on caches but these can be extended to other components of computers such as translation look aside buffers to prevent side channels.

Our Contributions

- We have figured out the direct relation problem.
- We have solved the problem using indirect eviction and designed the cache on it – IE-Cache.
- We also have found the possible attack on IE-Cache.
- We have evaluated the security and performance of IE-Cache.

Thank you