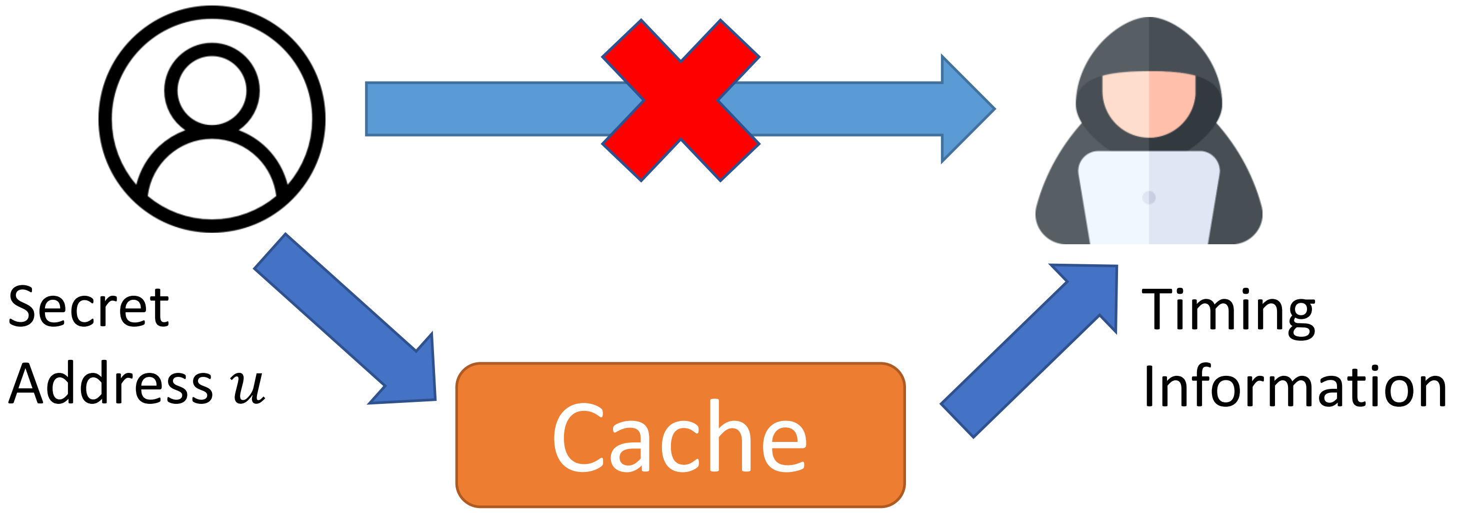
CacheGuard: A Behavior Model Checker for Cache Timing Side-Channel Security

Zihan Xu[1], Lingfeng Yin[1], Yongqiang Lyu[1], Haixia Wang[1], Gang Qu[2], and Dongsheng Wang[1]

[1]Tsinghua University, [2]University of Maryland

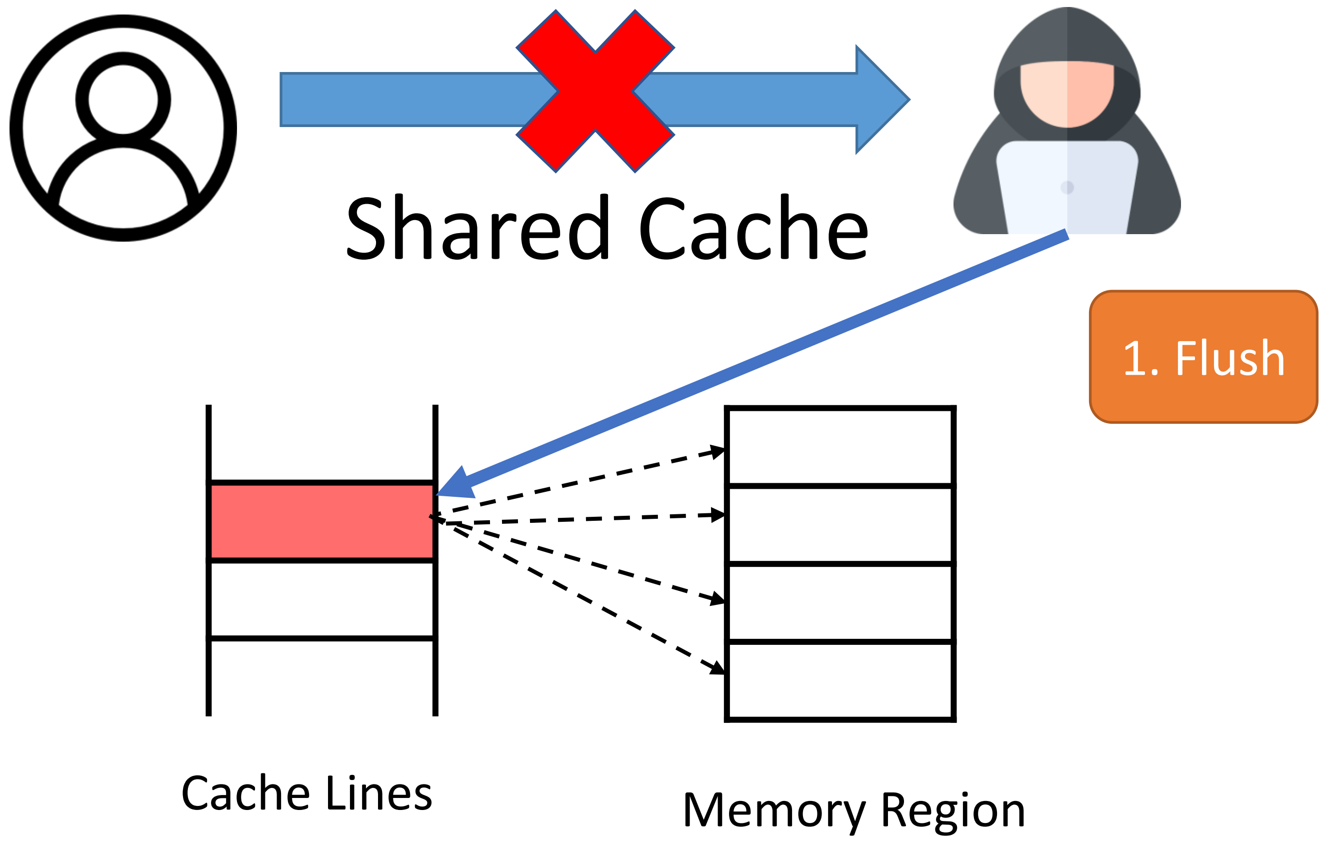
ASPDAC-2022, Jan 17th

Threats of Cache Side-Channel Attacks



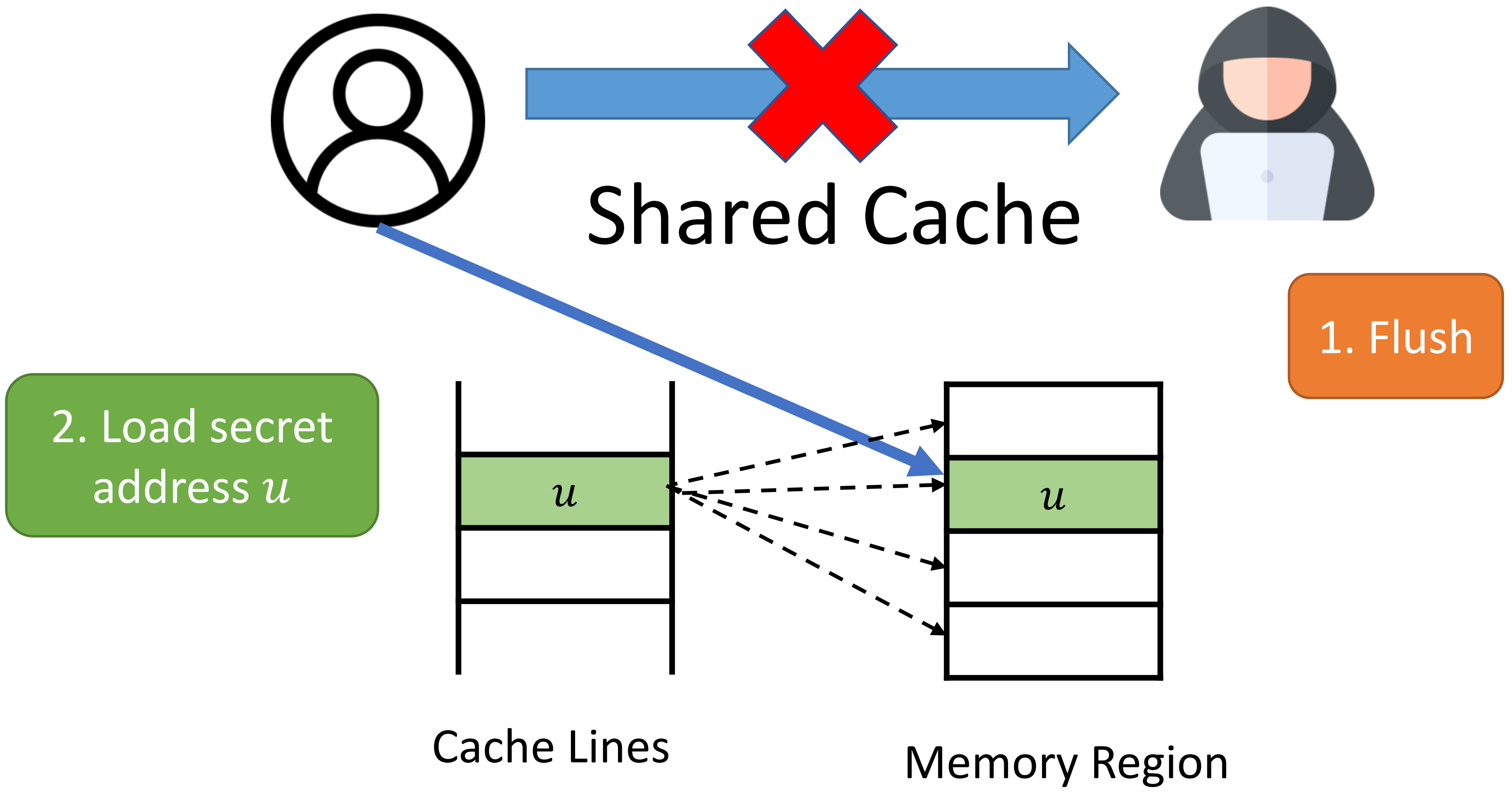
- Emerging threats
  - Spectre, Meltdown, Foreshadow…
- ... and long-lasting vulnerabilities

Example of Cache Side-Channel Attacks



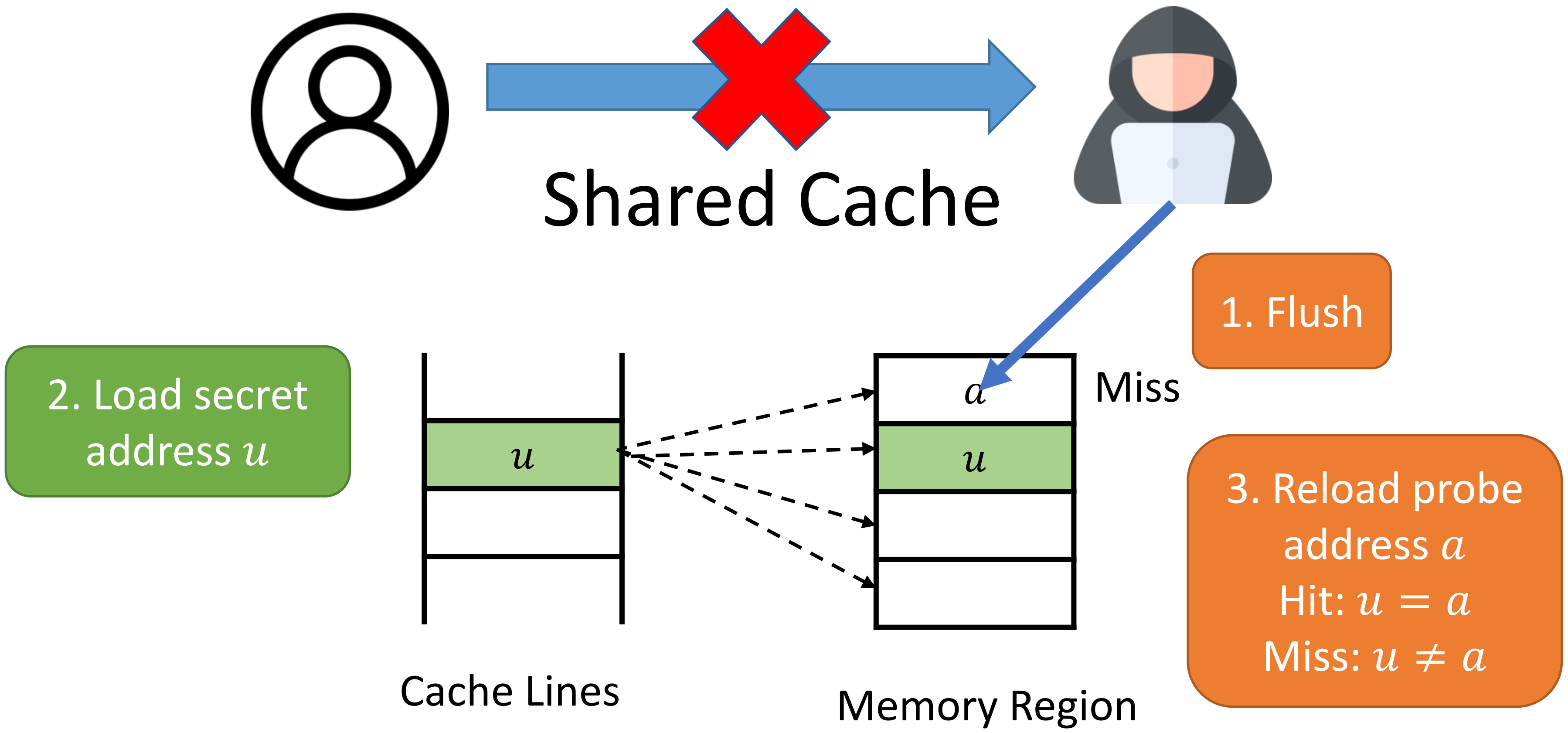
- Flush + Reload [USENIX-SEC’ 14]

Example of Cache Side-Channel Attacks



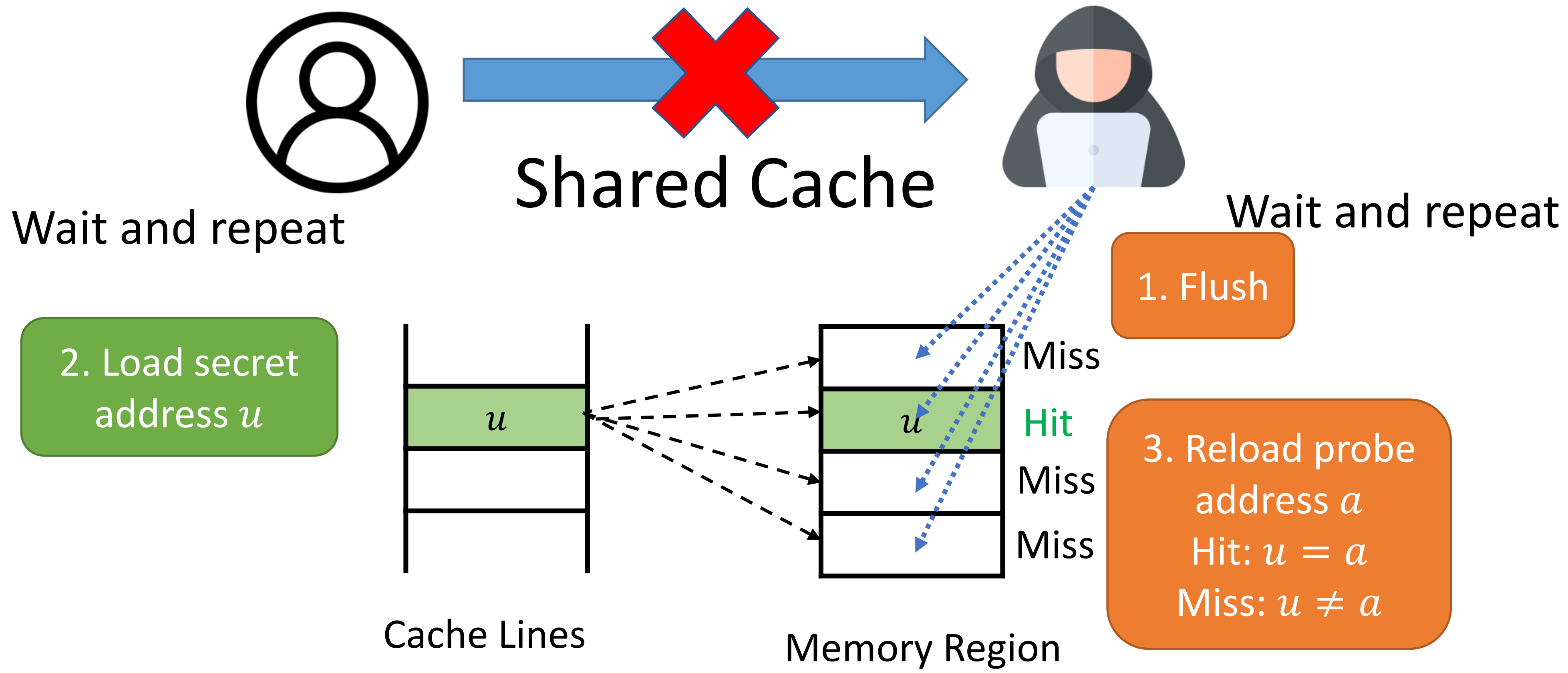
- Flush + Reload [USENIX-SEC' 14]

Example of Cache Side-Channel Attacks



- Flush + Reload [USENIX-SEC' 14]

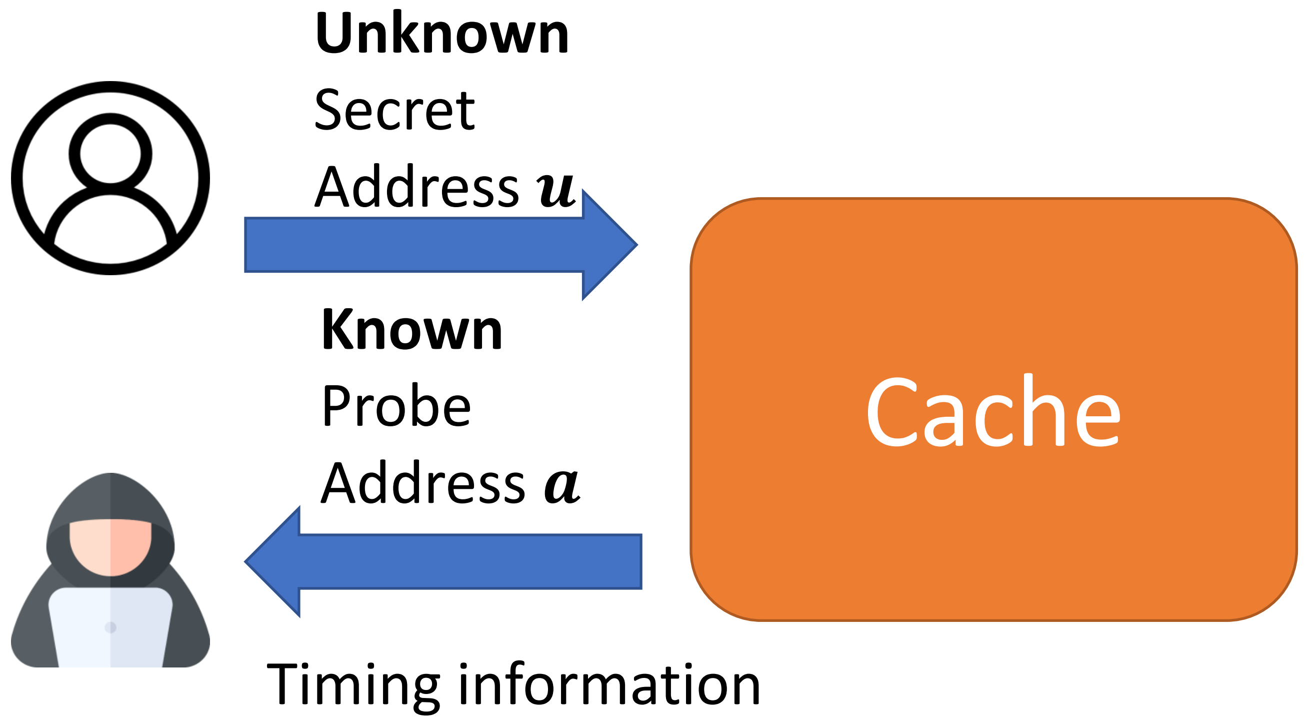
Example of Cache Side-Channel Attacks



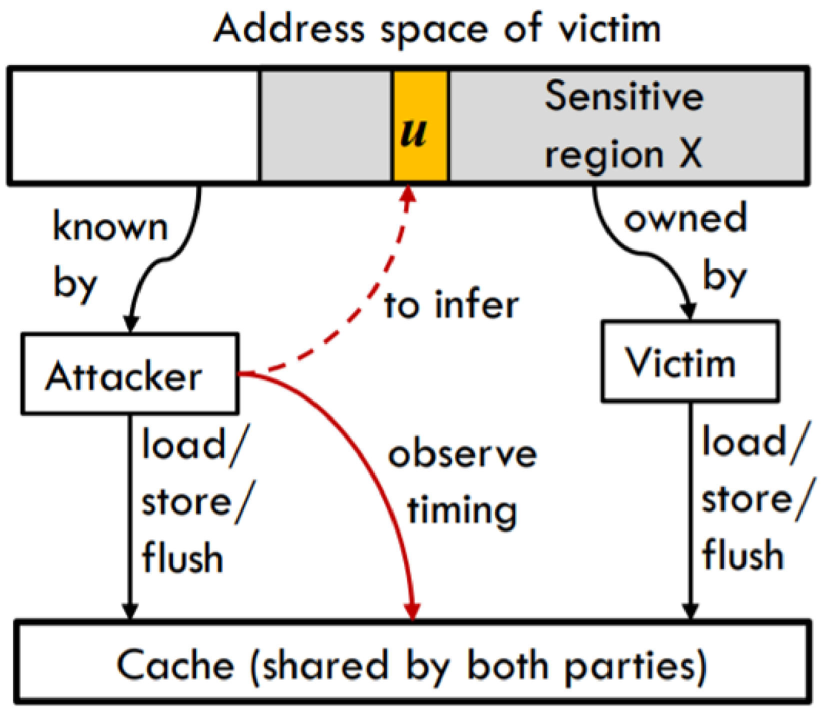
- Flush + Reload [USENIX-SEC’ 14]

Key aspects of Cache Side-channel Threat models

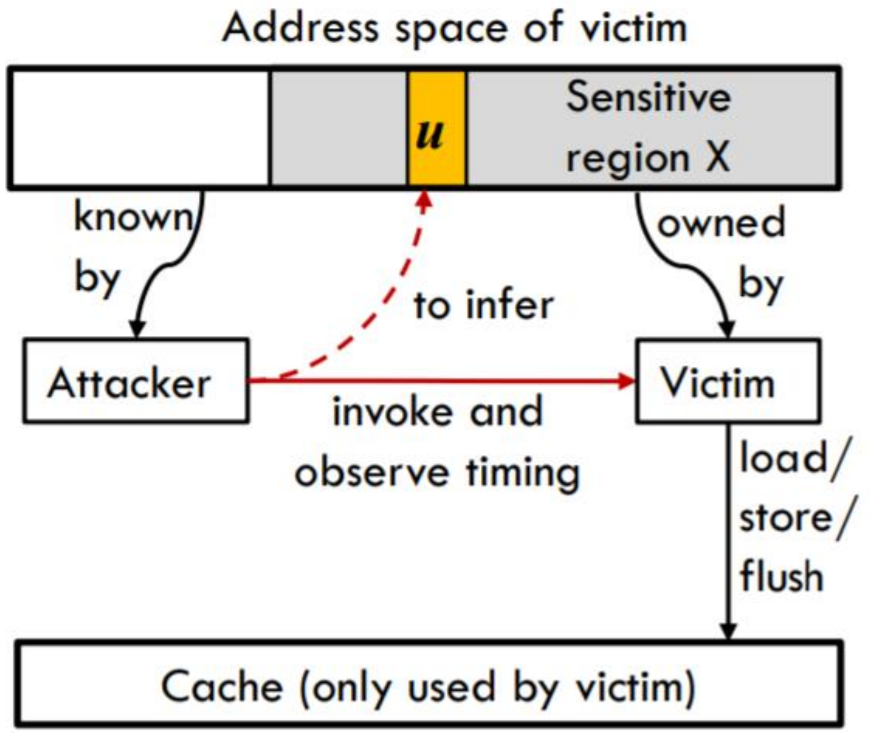
- Secret address u
- Known address a
- A series of operations
- Timing observation



Threat Models of Side-Channel Security



(a) External-interference model



(b) Internal-interference model

Different Cache Side-Channel Attacks

- Evict + Time, Prime + Probe, Bernstein's Attacks…
- Exploits different aspect of cache behavior
  - Directly observe timing of cache hit
  - Write-back policy on flush operations
  - Eviction by confliction
- With different threat model
  - External interference or internal interference

Secure Cache Designs

- Partitioned cache
    - SP* Cache, DAWG Cache , SecVerilog Cache, …
- Randomized cache
    - Random Fill cache, CEASER Cache, …
- Hybrid cache
    - HYBCACHE, …
- Can those caches defend all side-channel threats?
    - Need fully check!

Behavior Analysis of Secure Caches

- Finite-state machine model of cache side-channel [Zhang, 2014] and statistical cache behavior analysis [He, 2017].
- Build model for known attack behavior, and analyze cache behavior based on certain practical attacks.
- Limitation:
  - Can only evaluate well-known side-channel attacks.

T. Zhang and R. B. Lee. "New models of cache architectures characterizing information leakage from cache side channels." *Proceedings of the 30th annual computer security applications conference*, pp. 96–105, 2014.
Z. He and R. B. Lee. "How secure is your cache against side-channel attacks?" *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 341–353, 2017

Formal Verification of Secure Caches

- Information flow-based Secure Cache designs
- E.g. SecVerilog Cache

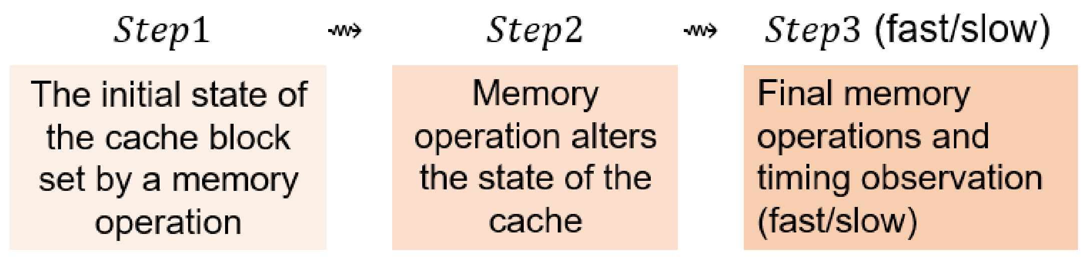
- Formal verification of specification of information flow through processor, based on security labels.

- Limitation:
  - Can not defend attacks based on internal interference.

Zhang, Danfeng, et al. "A hardware design language for timing-sensitive information-flow security." *Acm Sigplan Notices* 50.4 (2015): 503-516.

Three-step Model for Cache Security

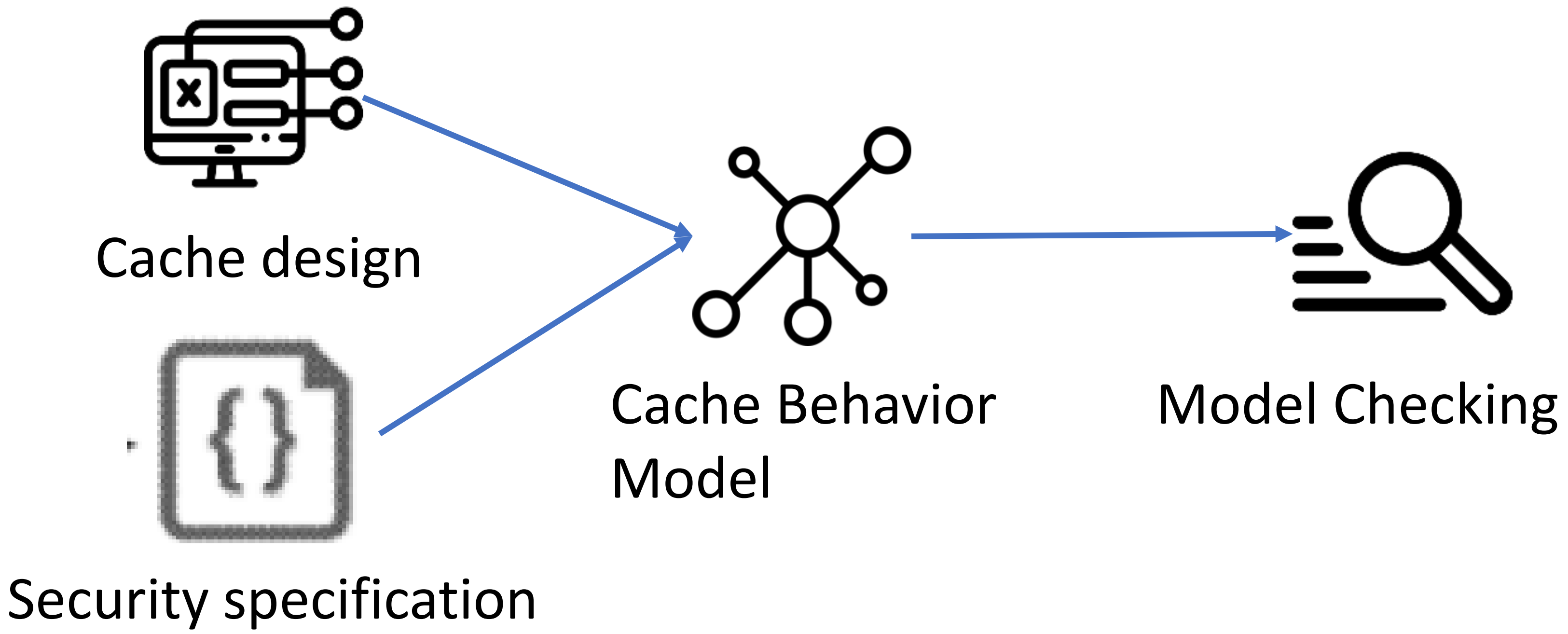
- Use three memory operations to model current cache attacks


$Step1$
The initial state of the cache block set by a memory operation
$Step2$
Memory operation alters the state of the cache
$Step3$ (fast/slow)
Final memory operations and timing observation (fast/slow)

- Enumerate attacks within three steps to discover new attacks.
- Limitation:
  - Relies prose simplification rules that have defects.
  - Threat model based on standard cache, not appliable to secure cache designs.

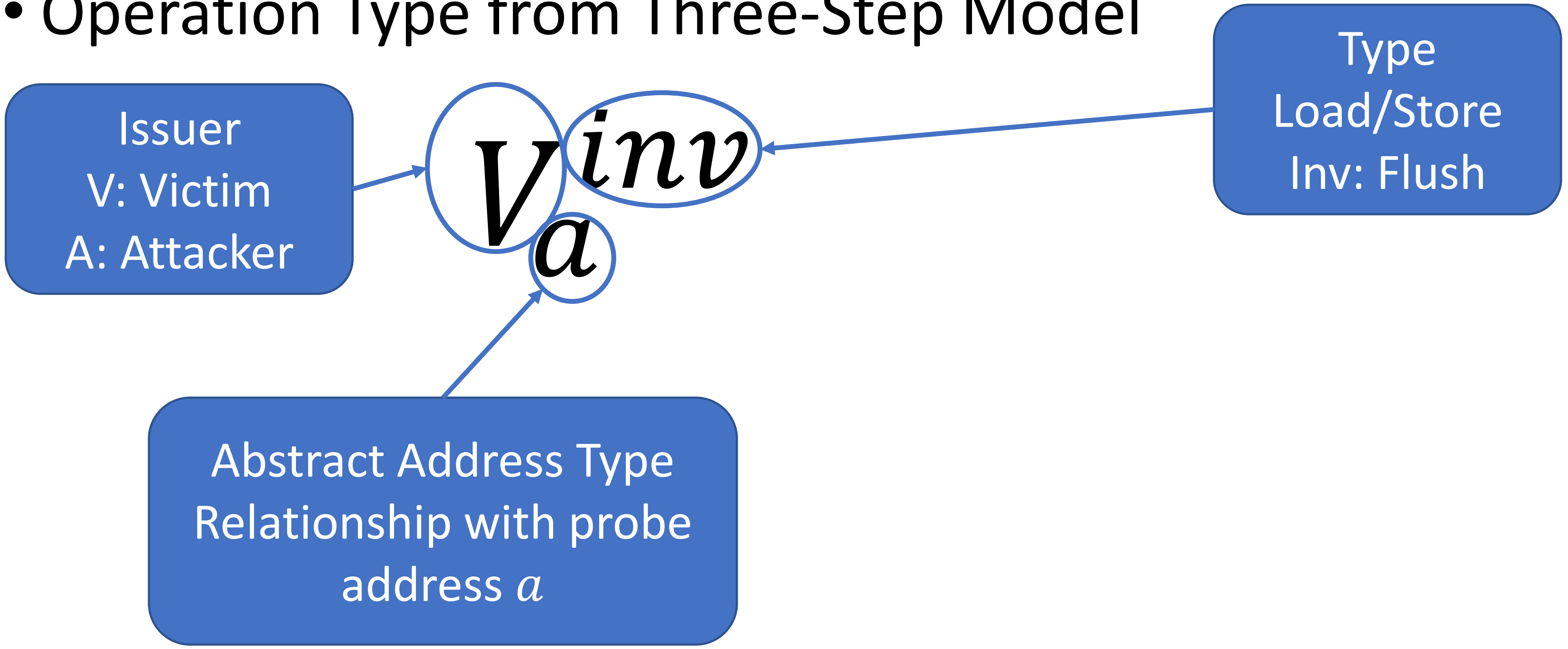
Deng, W. Xiong, and J. Szefer, "A benchmark suite for evaluatingcaches' vulnerability to timing attacks," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 683–697

Checking Secure Caches

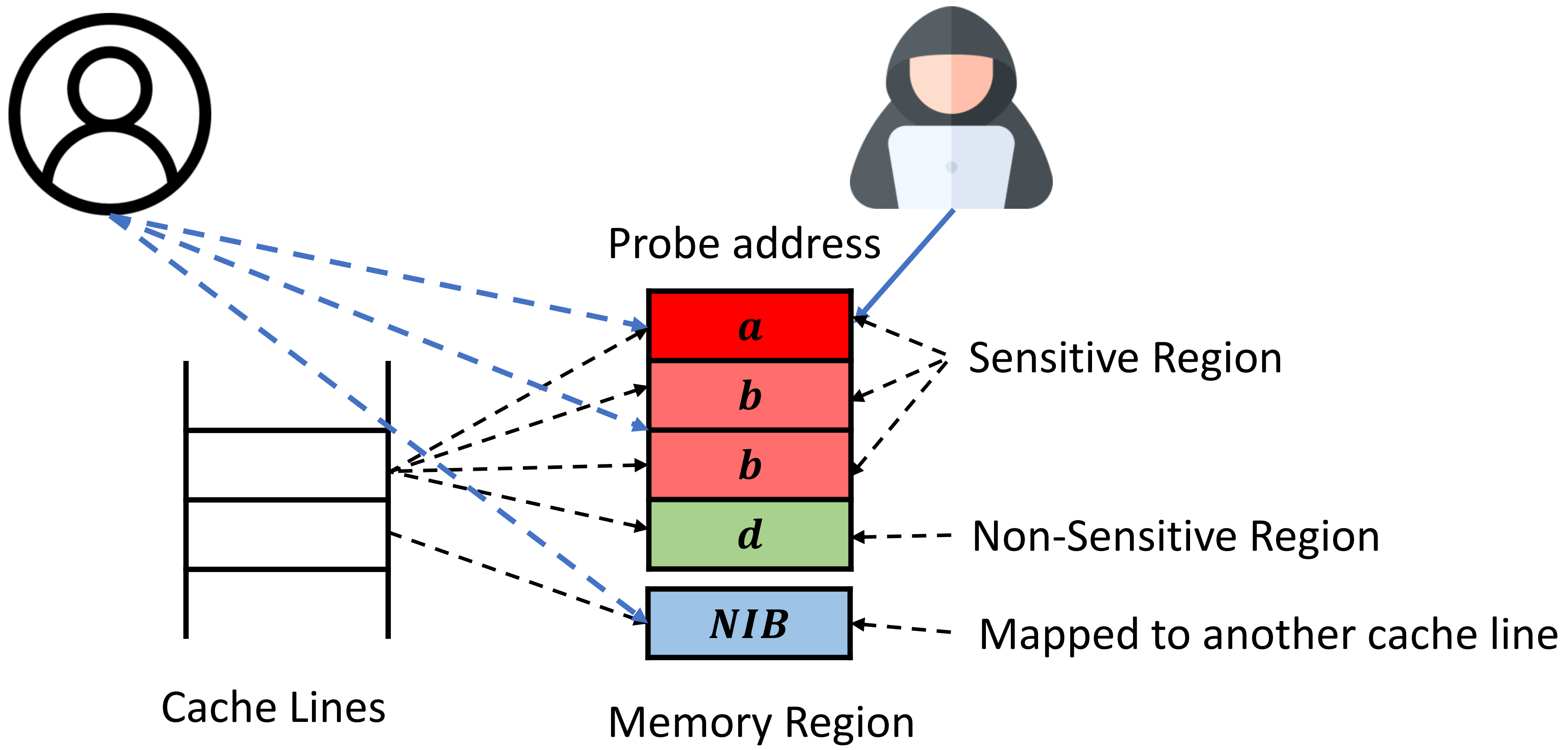


Modeling Cache Behavior

- Formally Modeling Cache Behavior
  - Operation Type from Three-Step Model



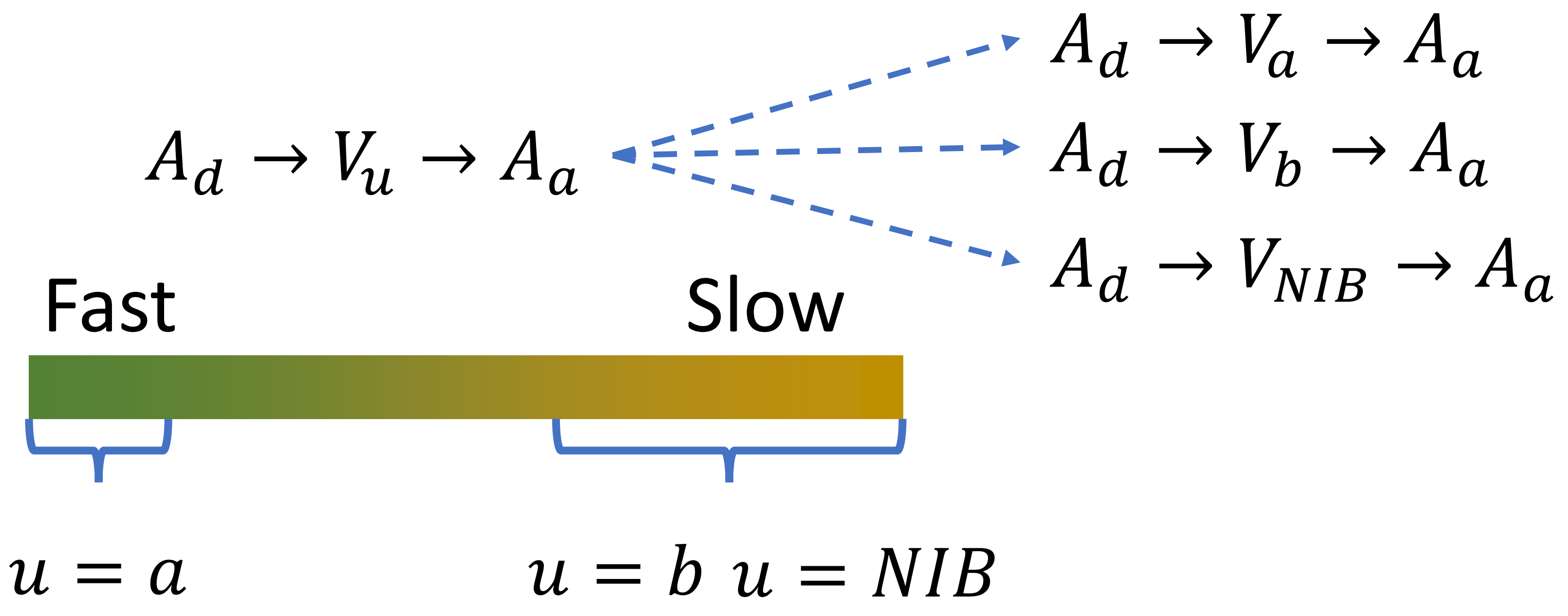
Modeling Cache Behavior



- Possible relationship with address a .

Modeling Cache Behavior

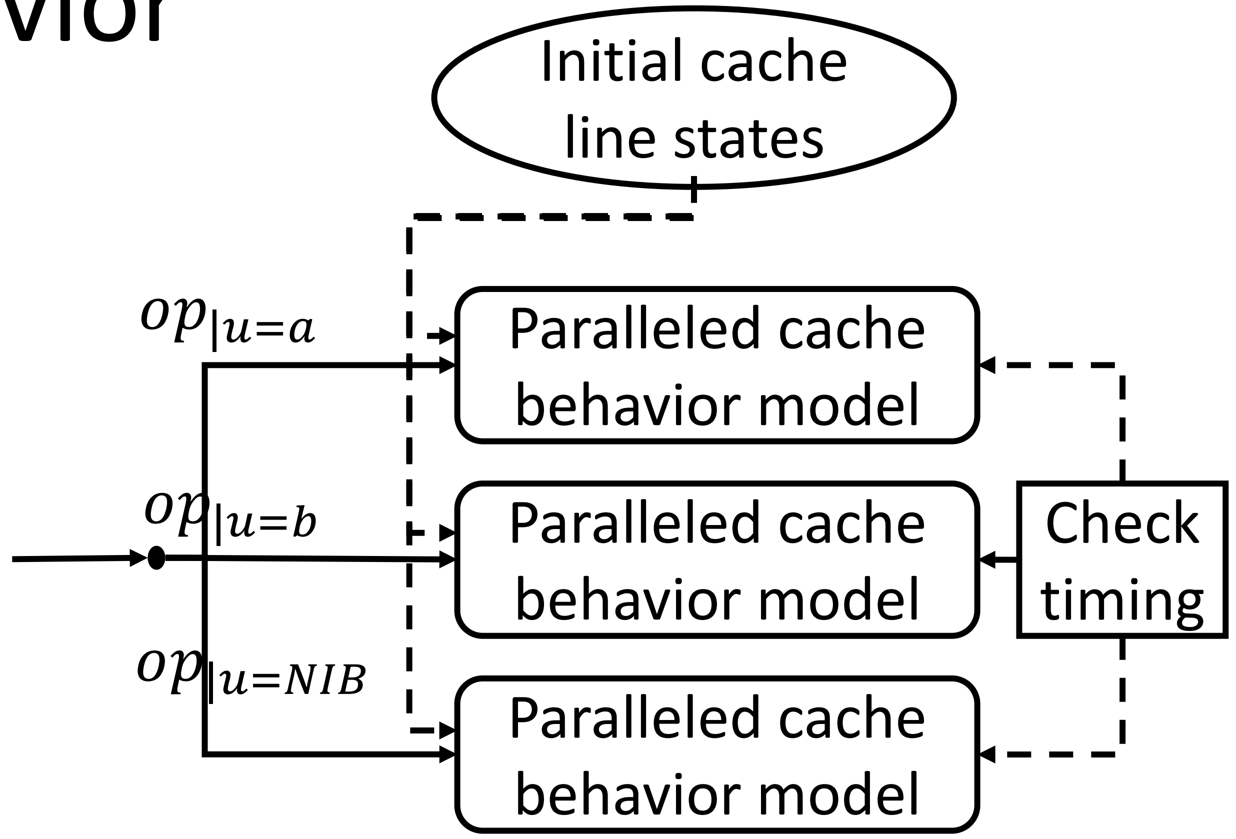
- Attacker's objective
- Three possible u types have three timing results
- **Distinguish** timing difference for **different** u in the **same** operation sequence



$A_d \rightarrow V_u \rightarrow A_a$
$A_d \rightarrow V_a \rightarrow A_a$
$A_d \rightarrow V_b \rightarrow A_a$
$A_d \rightarrow V_{NIB} \rightarrow A_a$
Fast
Slow
$u = a$
$u = b$ $u = NIB$

Modeling Cache Behavior

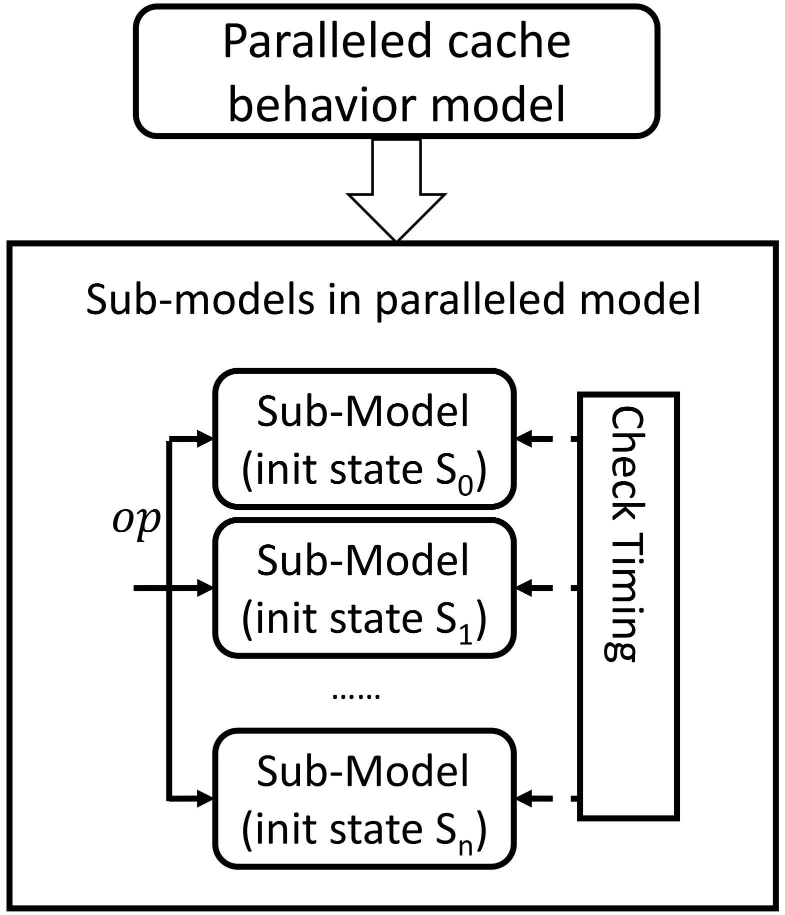
- Build paralleled cache behavior model for each u value.
- Address u in the raw operation sequence will be replaced into a, b and NIB respectively.
- Timing from three paralleled models is checked against the timing specification.


Initial cache line states
$op_{|u=a}$
$op_{|u=b}$
$op_{|u=NIB}$
Paralleled cache behavior model
Paralleled cache behavior model
Paralleled cache behavior model
Check timing

Modeling Cache Behavior - Initial States

- If the initial cache state is known, the timing will be deterministic.
- In practical attack, the cache state is **unknown** to the attacker

- Solution for **ambiguous** timing:
  - Build **sub-models** for each possible initial cache states.
  - Timings from sub-models will be **summarized** to fast/slow/ambiguous timing.

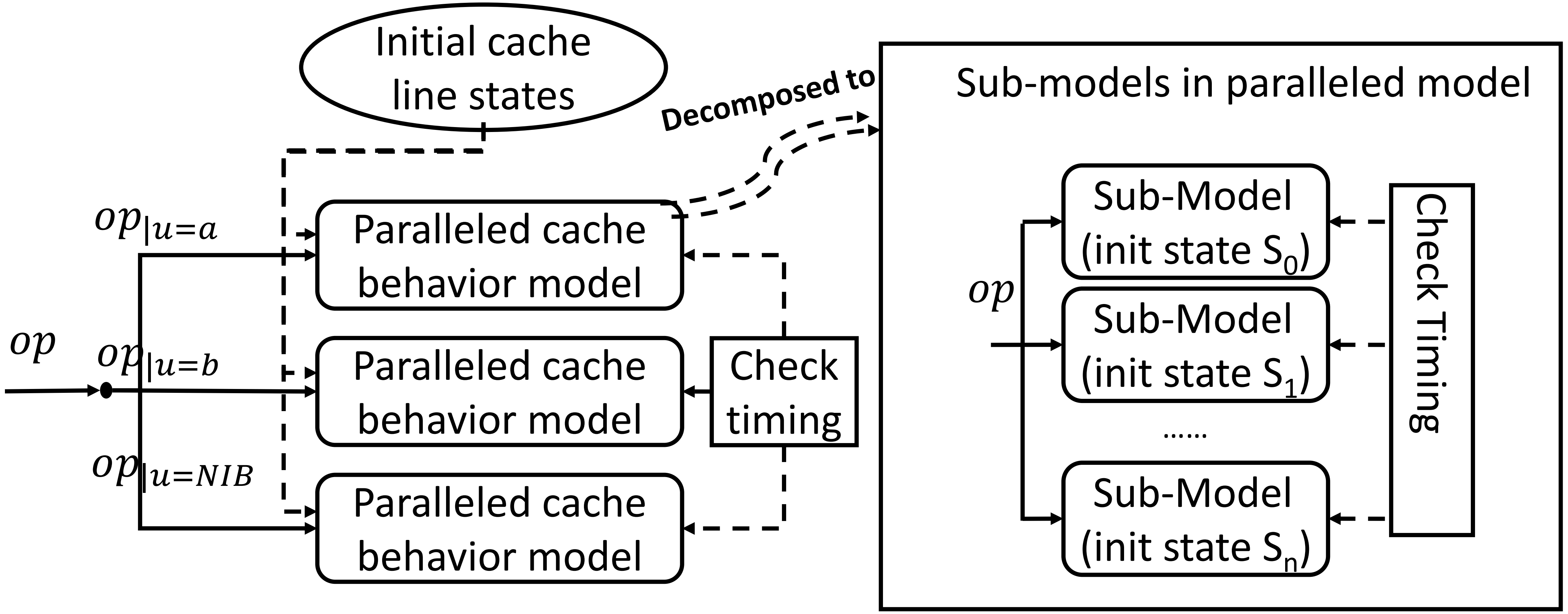

Paralleled cache behavior model
Sub-models in paralleled model
op
Sub-Model (init state S_0)
Sub-Model (init state S_1)
……
Sub-Model (init state S_n)
Check Timing

Timing Check Principle

- Timing checking principle
- If three timing is **same -> no vulnerability**.
- If at least one timing is **ambiguous**, the timing overlaps -> **weak vulnerability**.
- Otherwise, the timing is **different** and **non-ambiguous -> strong exploitable vulnerability.**
- Use model checker to check security specification.

```
attack := case
    ( PM0_timing = PM1_timing &
      PM0_timing = PM2_timing)
       : notVul;
    ( PM0_timing = ambiguous |
      PM1_timing = ambiguous |
      PM2_timing = ambiguous )
       : weakVul;
    TRUE: strongVul;
esac;
SPEC AG !(attack = strongVul)
```

Overall Cache Model


Initial cache line states
Decomposed to
Sub-models in paralleled model
op
$op_{|u=a}$
$op_{|u=b}$
$op_{|u=NIB}$
Paralleled cache behavior model
Paralleled cache behavior model
Paralleled cache behavior model
Check timing
op
Sub-Model (init state S_0)
Sub-Model (init state S_1)
……
Sub-Model (init state S_n)
Check Timing

Experiment Setup

- NuSMV 2.6.0 model checker with our path enumeration extension
- Specification
  - Quad-core 3.9GHz I5-8300H CPU (2 cores used)
  - 8GB memory
  - Ubuntu 18.04.5 LTS
- Cache designs
  - Standard cache
  - Partition cache: SP* cache
  - Randomized cache: Random Fill cache, CEASER cache.

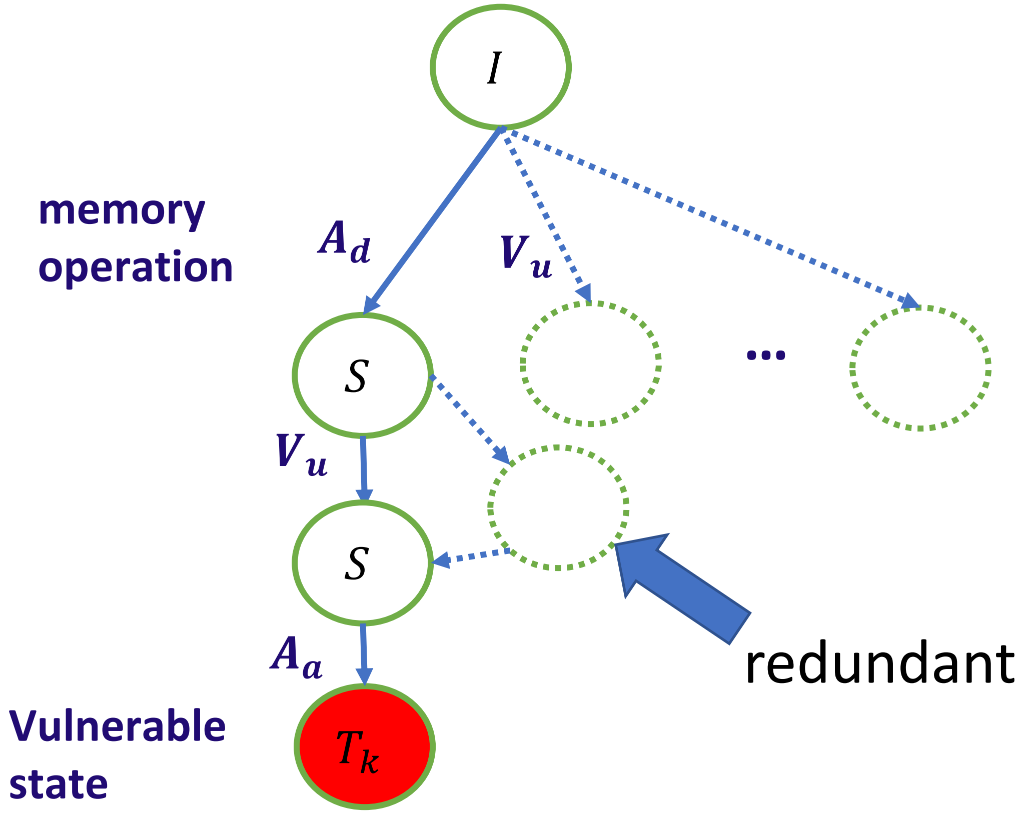
Finding Side-channel Attacks

- Covering all possible attacks
  - Operations path -> Attack sequence

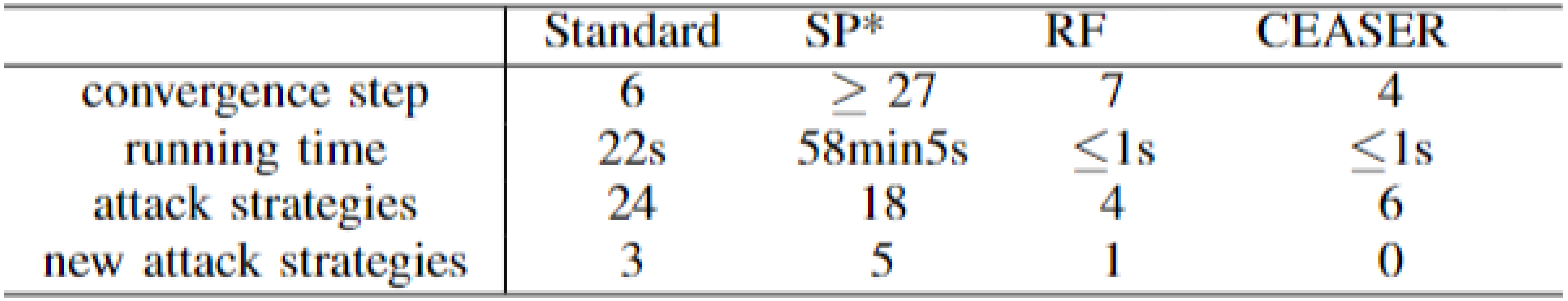
- All vulnerable states must be covered.

- Non-redundant representation.

- Find all shortest paths



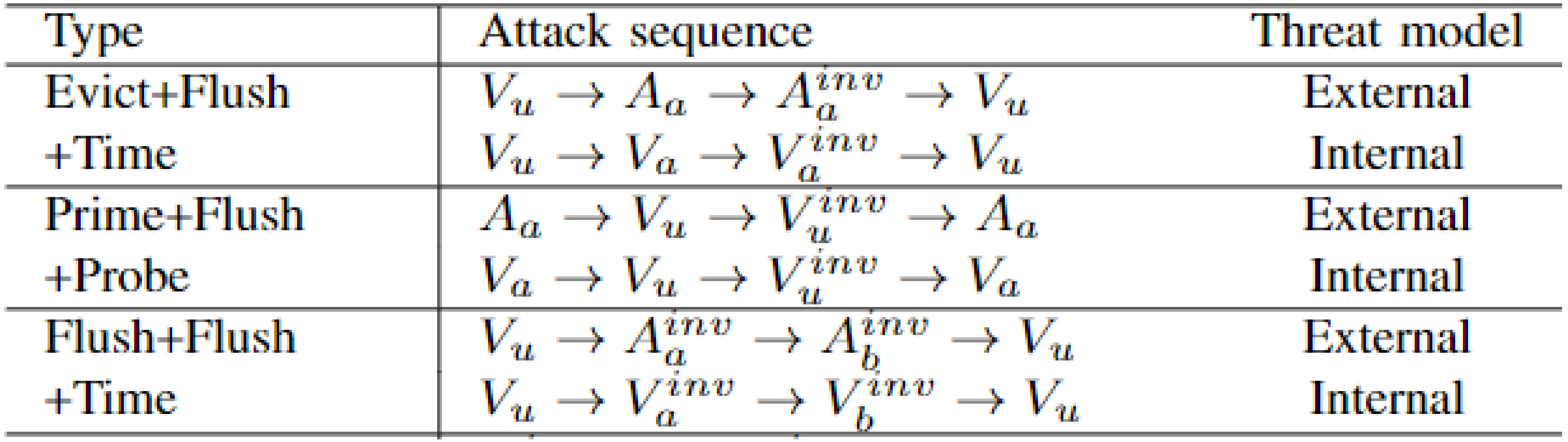
Verification results



	Standard	SP*	RF	CEASER
convergence step	6	≥ 27	7	4
running time	22s	58min5s	$\leq$1s	$\leq$1s
attack strategies	24	18	4	6
new attack strategies	3	5	1	0

- Attacks found are generalized into 26 attack strategies based on their interference and operation type.
- None of those cache are fully safe.

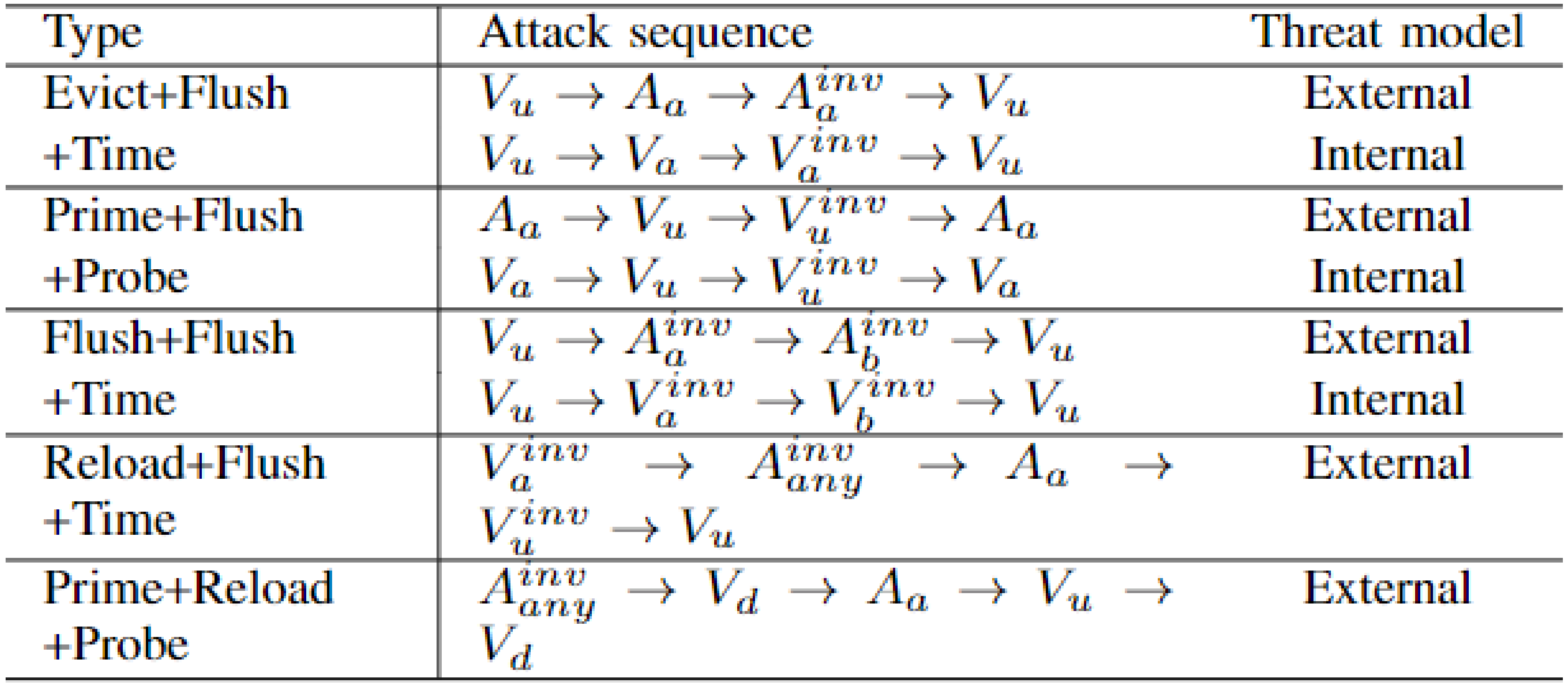
Findings on Standard Cache



Type	Attack sequence	Threat model
Evict+Flush	$V_u \rightarrow A_a \rightarrow A_a^{inv} \rightarrow V_u$	External
+Time	$V_u \rightarrow V_a \rightarrow V_a^{inv} \rightarrow V_u$	Internal
Prime+Flush	$A_a \rightarrow V_u \rightarrow V_u^{inv} \rightarrow A_a$	External
+Probe	$V_a \rightarrow V_u \rightarrow V_u^{inv} \rightarrow V_a$	Internal
Flush+Flush	$V_u \rightarrow A_a^{inv} \rightarrow A_b^{inv} \rightarrow V_u$	External
+Time	$V_u \rightarrow V_a^{inv} \rightarrow V_b^{inv} \rightarrow V_u$	Internal

- Three new side-channel strategies that can not be covered by the three-step model.

Findings on SP Cache



Type	Attack sequence	Threat model
Evict+Flush +Time	$V_u \to A_a \to A_a^{inv} \to V_u$	External
	$V_u \to V_a \to V_a^{inv} \to V_u$	Internal
Prime+Flush +Probe	$A_a \to V_u \to V_u^{inv} \to A_a$	External
	$V_a \to V_u \to V_u^{inv} \to V_a$	Internal
Flush+Flush +Time	$V_u \to A_a^{inv} \to A_b^{inv} \to V_u$	External
	$V_u \to V_a^{inv} \to V_b^{inv} \to V_u$	Internal
Reload+Flush +Time	$V_a^{inv} \to A_{any}^{inv} \to A_a \to V_u^{inv} \to V_u$	External
Prime+Reload +Probe	$A_{any}^{inv} \to V_d \to A_a \to V_u \to V_d$	External

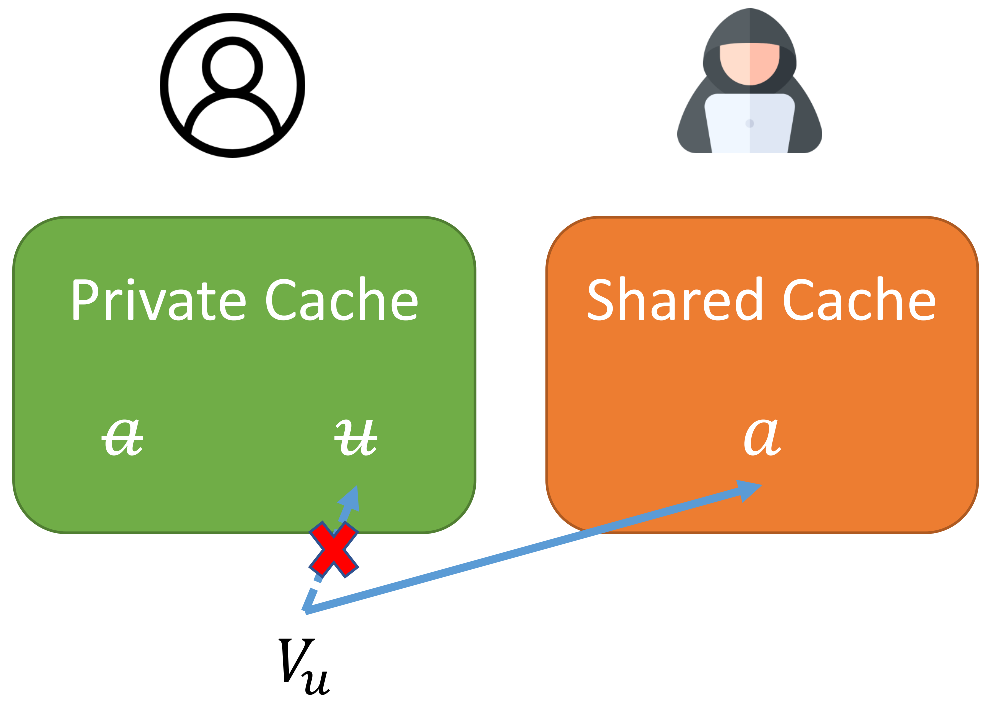
- Additional side-channel attacks based on the design defects of SP* cache.
- Undetectable by benchmarks based on standard cache.

Case Study: New attacks on SP* cache

- Reload + Flush + Time
  - $\boldsymbol{V_a^{inv} \rightarrow A^{inv} \rightarrow A_a \rightarrow V_u^{inv} \rightarrow V_u}$

- Address u had been evicted from the private cache region, so it can only hit in the attacker-controlled region.

- Do not work on standard cache.



Evaluation on Random Fill Cache

- Uses randomized fill requests with data from a range of addresses to fill the cache at cache miss.
- Vulnerable to 4 attack strategies
- i.e., Flush + Time, Evict + Time, Bernstein's Attack, and newly discovered Flush + Flush + Time.
- All discovered attacks must start with the operation of V_u.
- Attack will only happen if secret u is in the cache before random filling strategy starts.

Evaluation on CEASER cache

- Use encrypted address to calculate the mapping from address to cache set.
- Can successfully defend against conflict-based attacks.
- Still vulnerable to 6 eviction-based attack strategies
- i.e., Flush + Reload, Evict + Reload, Flush + Time, Reload + Time, Cache Internal Collision and Flush + Probe.

Conclusion

- Presents a model checking technique for cache security.
- Discovered attacks belongs to 26 attack strategies, including 5 new attack strategies.
- A complete evaluation of secure cache designs for their protection capabilities.
- Help develop secure cache.