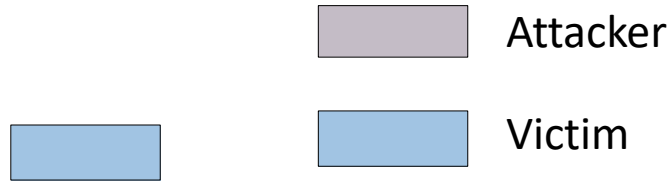
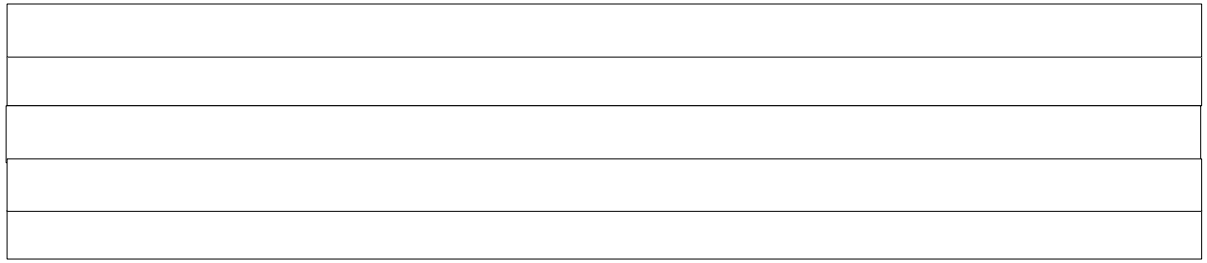
Systematic Evaluation of Randomized Cache Designs against Cache Occupancy

Anirban Chakraborty[1], Nimish Mishra[2], Sayandeep Saha[3], Sarani Bhattacharya[2] and Debdeep Mukhopadhyay[2]

[1]Max Plank Institute for Security and Privacy, Germany
[2]Indian Institute of Technology Kharagpur, India
[3]Indian Institute of Technology Bombay, India

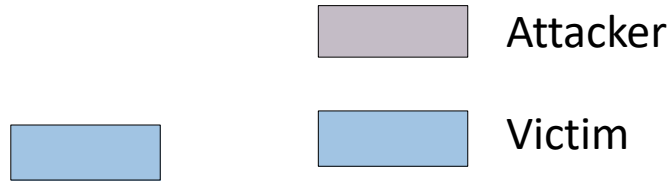
Prime+Probe Attack

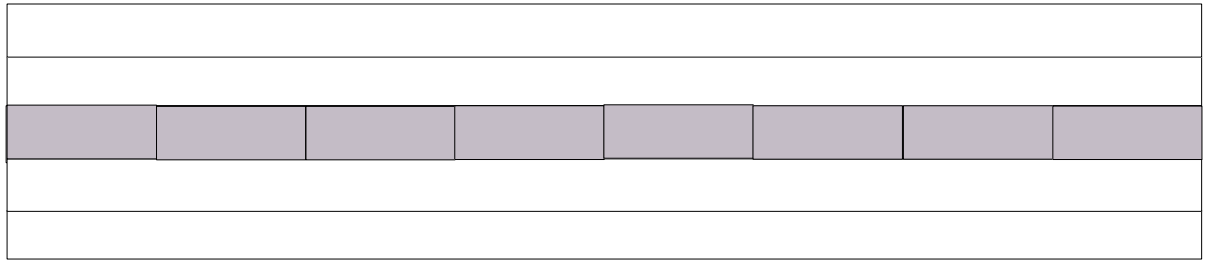




Prime+Probe in a cache set

Prime+Probe Attack

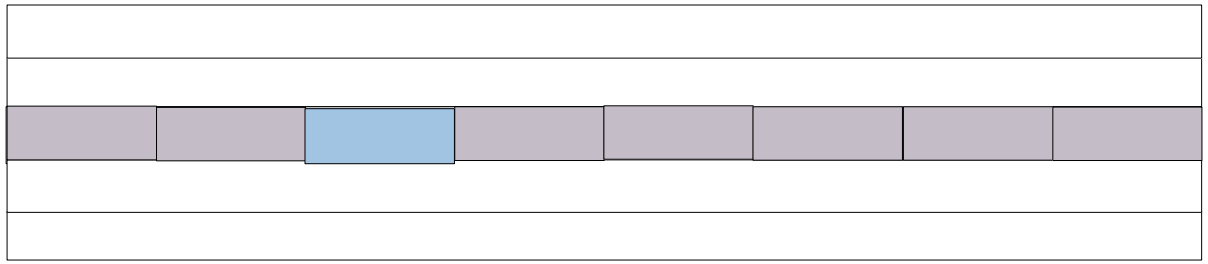




Prime+Probe in a cache set

Prime+Probe Attack

2

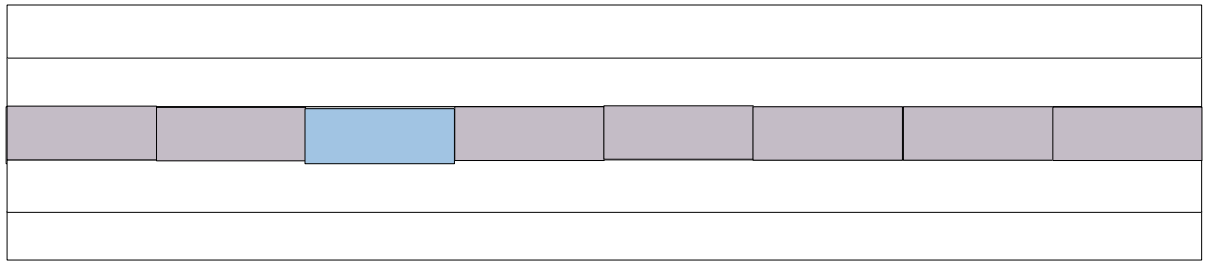


Attacker

Victim

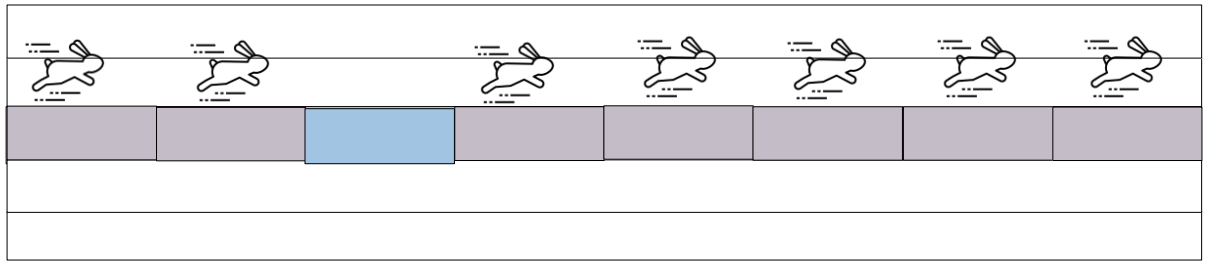
Prime+Probe in a cache set

Prime+Probe Attack



Prime+Probe in a cache set

Prime+Probe Attack

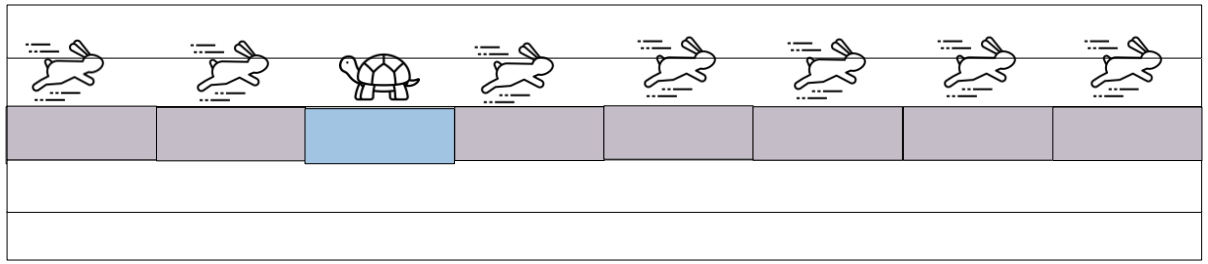


Attacker

Victim

Prime+Probe in a cache set

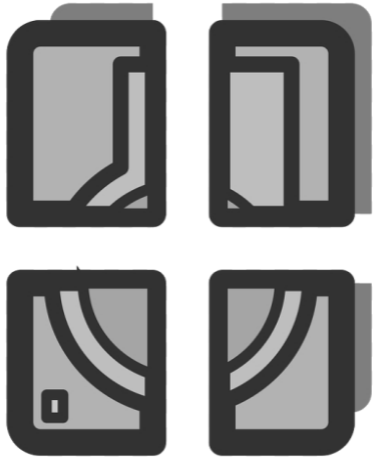
Prime+Probe Attack



Prime+Probe in a cache set

Randomized Caches

Randomization of addresses



Partitioning of Cache

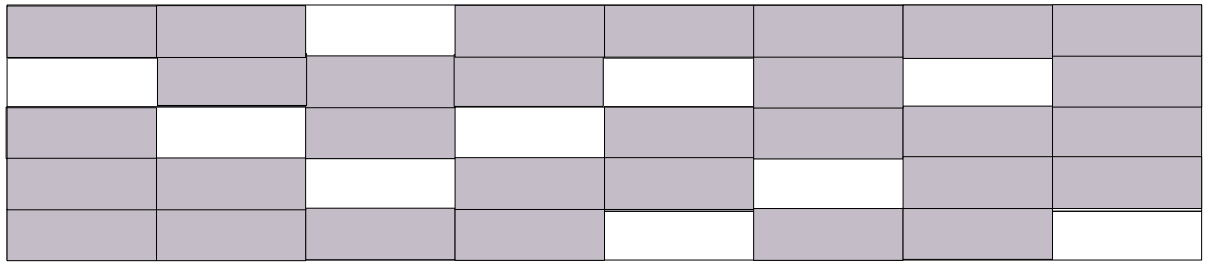
Cache Occupancy Attacks

Occupancy Attack in entire cache

Attacker

Victim

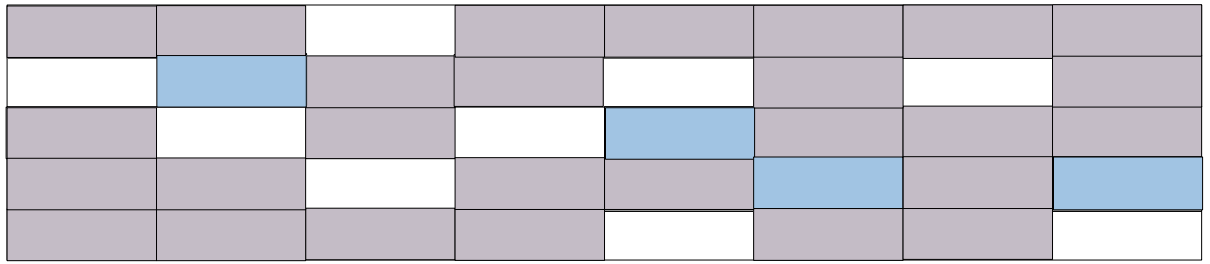
Cache Occupancy Attacks



Occupancy Attack in entire cache

Cache Occupancy Attacks

2



Attacker

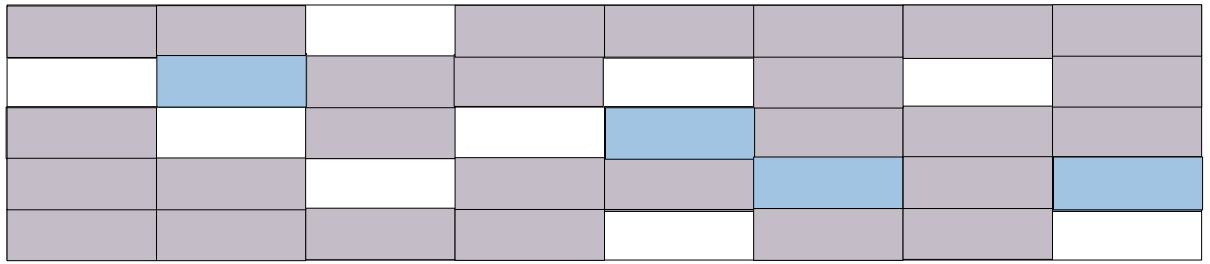
Victim

Occupancy Attack in entire cache

Cache Occupancy Attacks

Attacker

Victim



Occupancy Attack in entire cache

Motivation

State of the Art

- **Randomized Caches** – obfuscate address to cache-set mapping
- **Multiple design principles** – constantly evolving field

Motivation

State of the Art

- **Randomized Caches** – obfuscate address to cache-set mapping
- **Multiple design principles** – constantly evolving field

This Work

- **Performance Impact** – Evaluation across various replacement policies
- **Security Impact –** Testing resilience against cache occupancy attacks

Performance Evaluation

Problems

Performance Evaluation

Problems

- Different works consider different benchmarks, making it difficult to evaluate all designs against a common baseline

Performance Evaluation

Problems

- Different works consider different benchmarks, making it difficult to evaluate all designs against a common baseline
- Certain cache designs make inherent assumptions in their implementation which contemporary benchmarking strategies violate

Performance Evaluation

Problems

- Different works consider different benchmarks, making it difficult to evaluate all designs against a common baseline
- Certain cache designs make inherent assumptions in their implementation which contemporary benchmarking strategies violate
- MIRAGE[1] gem5 implementation assumes uniform distribution of loads/stores across lifetime of benchmark workloads, providing optimistic view of the performance overhead.

1. Saileshwar and Qureshi, MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design, USENIX Security 2021

Performance Evaluation

Problems

- Different works consider different benchmarks, making it difficult to evaluate all designs against a common baseline
- Certain cache designs make inherent assumptions in their implementation which contemporary benchmarking strategies violate
- MIRAGE[1] gem5 implementation assumes uniform distribution of loads/stores across lifetime of benchmark workloads, providing optimistic view of the performance overhead.

Solution

1. Saileshwar and Qureshi, MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design, USENIX Security 2021

Performance Evaluation

Problems

- Different works consider different benchmarks, making it difficult to evaluate all designs against a common baseline
- Certain cache designs make inherent assumptions in their implementation which contemporary benchmarking strategies violate
- MIRAGE[1] gem5 implementation assumes uniform distribution of loads/stores across lifetime of benchmark workloads, providing optimistic view of the performance overhead.

Solution

- Benchmarking *under spurious cache occupancy* that removes this implementation assumption of MIRAGE and provides a fair baseline for all cache design

1. Saileshwar and Qureshi, MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design, USENIX Security 2021

Performance Evaluation

Problems

- Different works consider different benchmarks, making it difficult to evaluate all designs against a common baseline
- Certain cache designs make inherent assumptions in their implementation which contemporary benchmarking strategies violate
- MIRAGE[1] gem5 implementation assumes uniform distribution of loads/stores across lifetime of benchmark workloads, providing optimistic view of the performance overhead.

Solution

- Benchmarking *under spurious cache occupancy* that removes this implementation assumption of MIRAGE and provides a fair baseline for all cache design
- Evaluate the performance five different replacement policies - RandomRP, TreePLRURP, WeightedLRURP, RRIPRP, and FIFORP

1. Saileshwar and Qureshi, MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design, USENIX Security 2021

Comparative Performance Evaluation

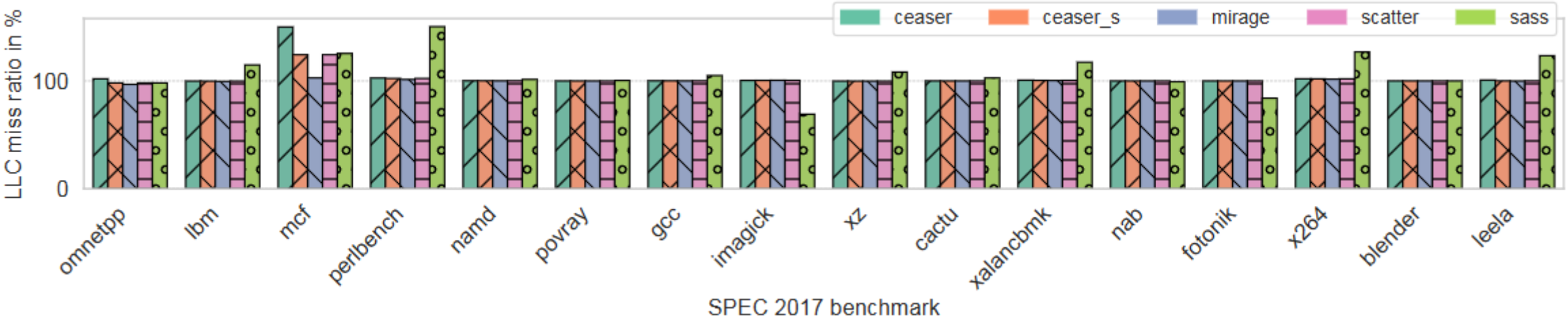
- **Platform**: gem5
- **Workload**: Spurious occupancy followed by SPEC2017
- **LLC size**: 16 MB
- **Skews** (wherever applicable): 2

L1 size: L1d : 512 KB, L1i : 32 KB
CPU model: TimingSimpleCPU
LLC replacement policy: various
Encryption latency: 3

Comparative Performance Evaluation

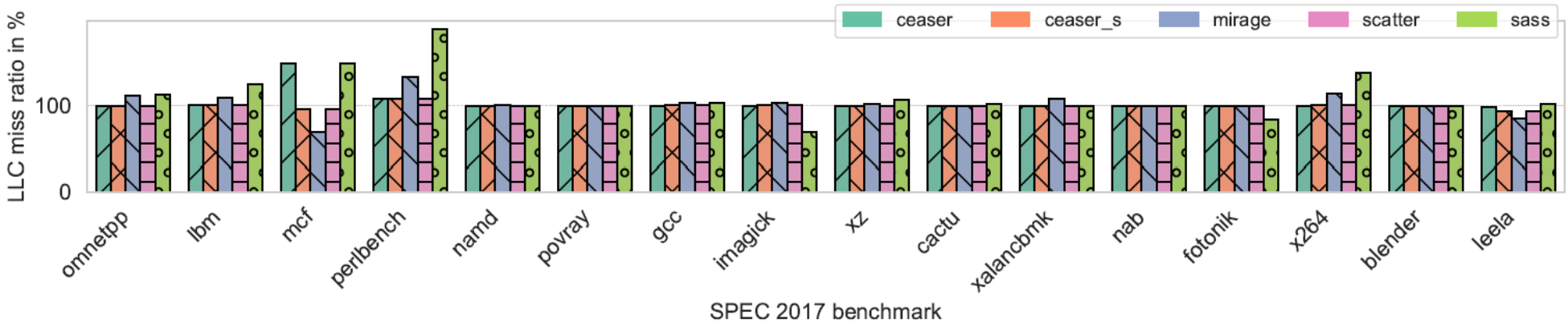
- **Platform**: gem5
- **Workload**: Spurious occupancy followed by SPEC2017
- **LLC size**: 16 MB
- **Skews** (wherever applicable): 2

L1 size: L1d : 512 KB, L1i : 32 KB
CPU model: TimingSimpleCPU
LLC replacement policy: various
Encryption latency: 3



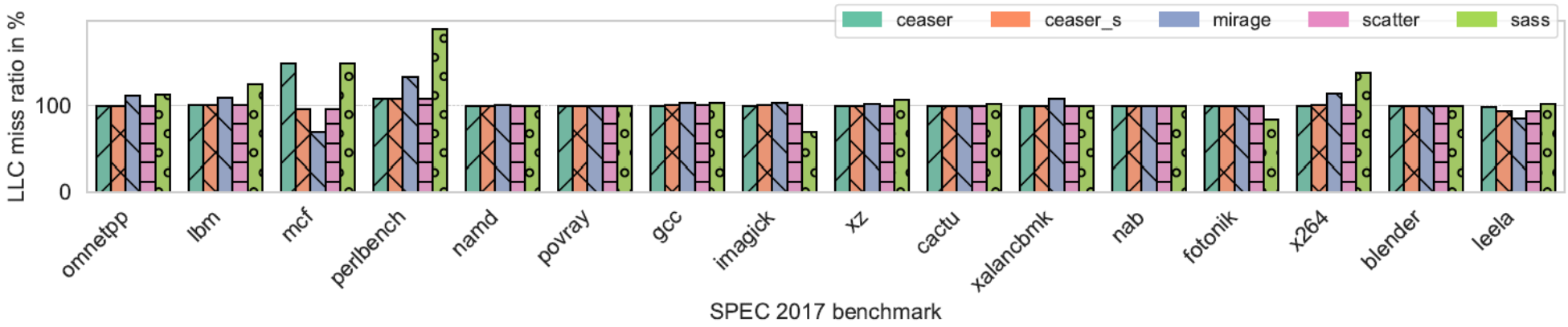
Performance evaluation of different cache designs with RandomRP replacement policy

Comparative Performance Evaluation



Performance evaluation of different cache designs with RRIPRP replacement policy

Comparative Performance Evaluation



Performance evaluation of different cache designs with RRIPRP replacement policy

Takeaway

- While most prior works are evaluated mostly with random replacement, we show that different cache designs have varied effects of replacement policies
- For some replacement policies, certain randomized cache designs actually outperform baseline set-associative cache

Cache Occupancy Attack: covert channel

Receiver

Sender

Cache Occupancy Attack: covert channel

Receiver

① Allocates sufficient memory space and randomly accesses ℓ number of addresses.

Sender

Cache Occupancy Attack: covert channel

Receiver

① Allocates sufficient memory space and randomly accesses ℓ number of addresses.

Sender

② Allocates a memory space and depending on the value of the bit (0 or 1) it wants to transmit, either makes x or y memory accesses

Cache Occupancy Attack: covert channel

Receiver

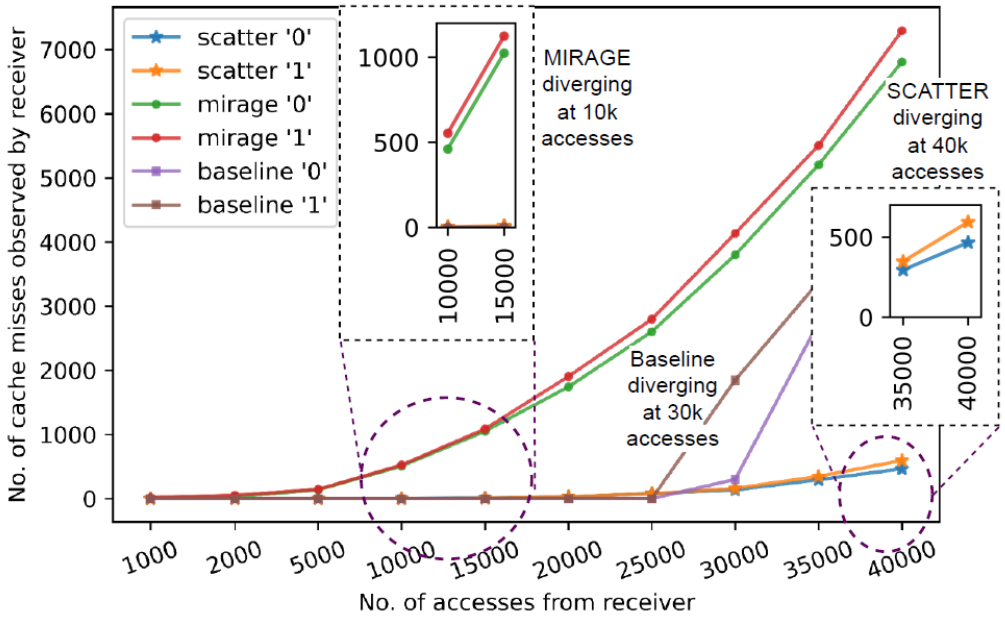
① Allocates sufficient memory space and randomly accesses ℓ number of addresses.

③ Re-accesses all the ℓ addresses it has previously installed in the cache.

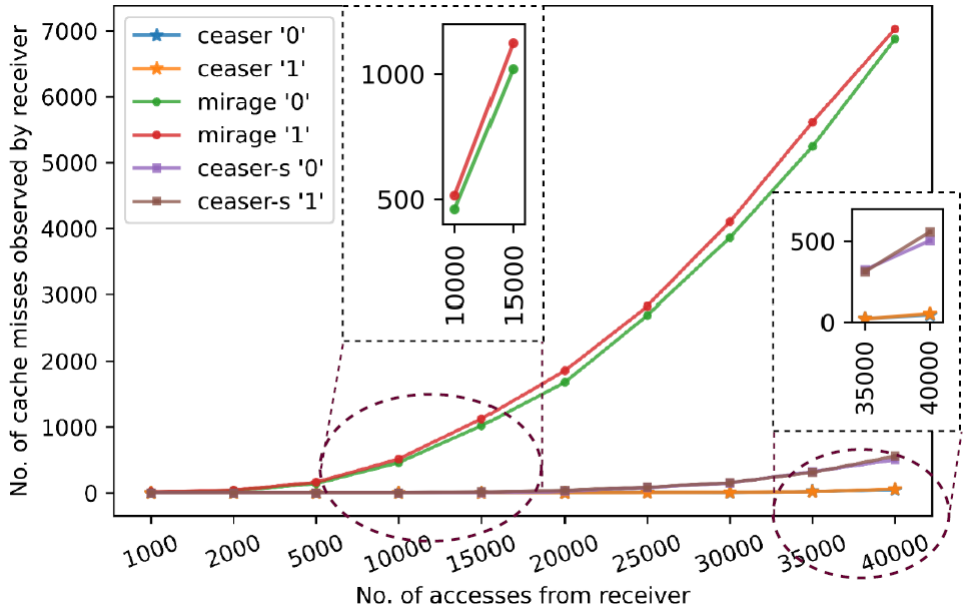
Sender

② Allocates a memory space and depending on the value of the bit (0 or 1) it wants to transmit, either makes x or y memory accesses

Cache Occupancy Attack: covert channel

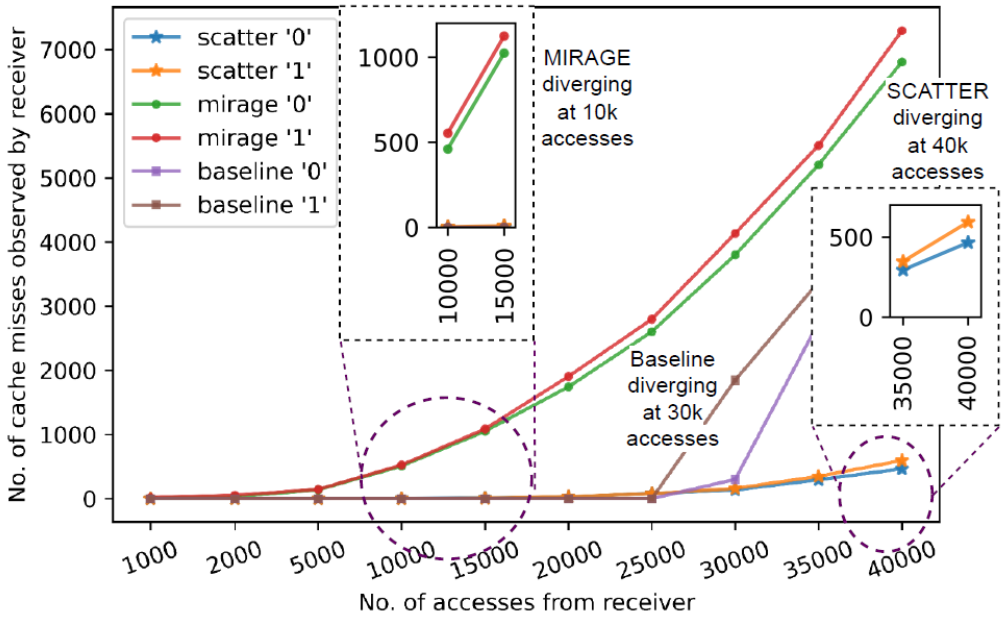


Cache occupancy channel in MIRAGE, ScatterCache and baseline set-associative cache

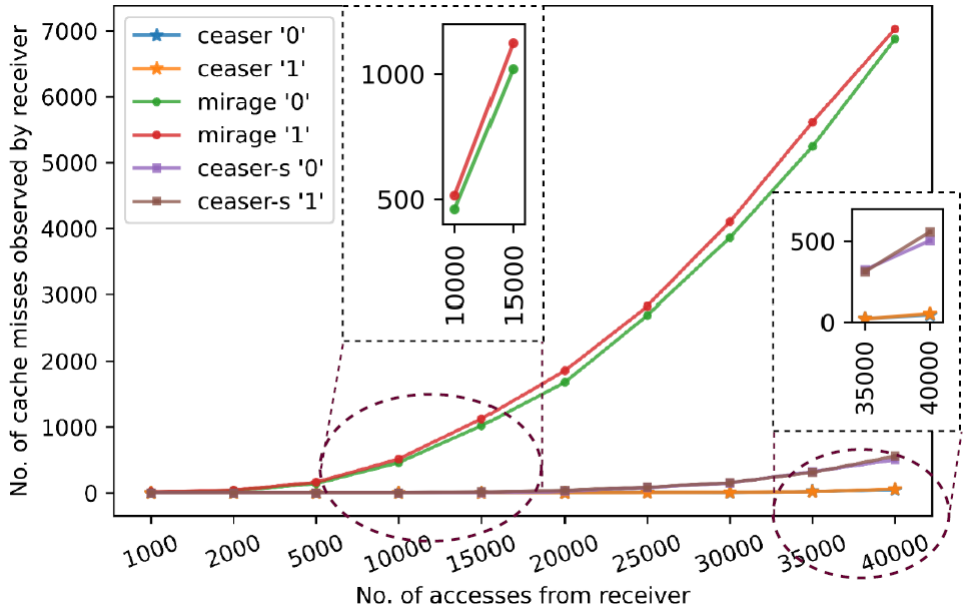


Cache occupancy channel in MIRAGE, CEASER and CEASER-S cache designs

Cache Occupancy Attack: covert channel



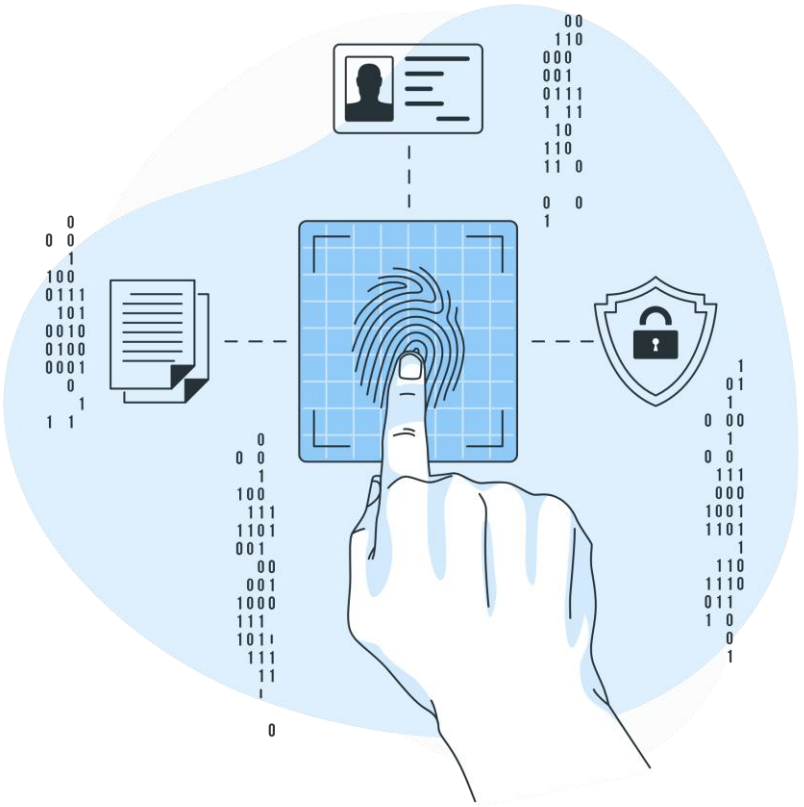
Cache occupancy channel in MIRAGE, ScatterCache and baseline set-associative cache



Cache occupancy channel in MIRAGE, CEASER and CEASER-S cache designs

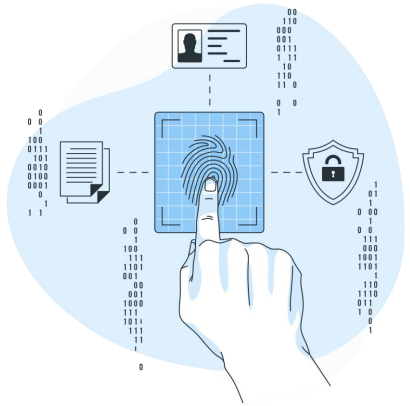
Takeaway

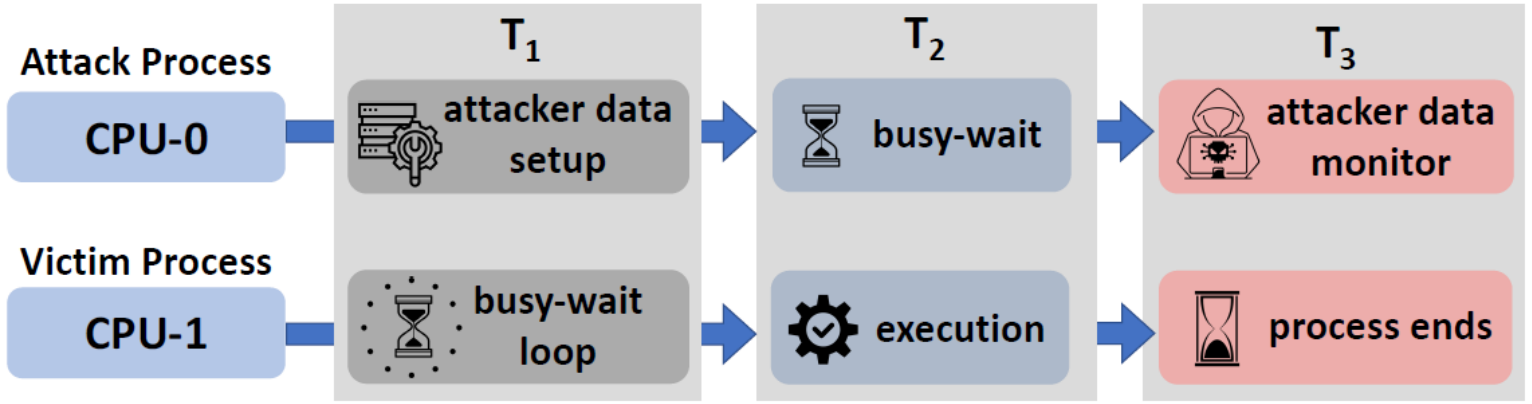
MIRAGE, due to its policy of compulsory eviction from the data store for every new address installation, is more vulnerable to cache occupancy attacks than other designs, including baseline set-associative cache.



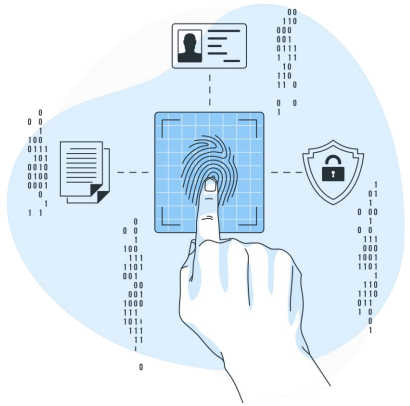
Cache Occupancy Attack: fingerprinting of processes

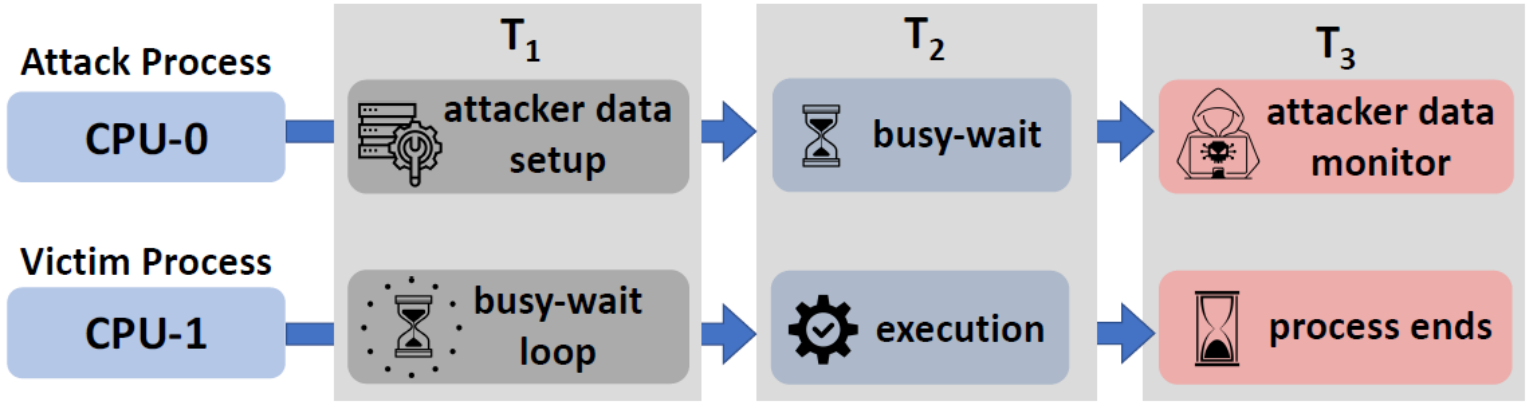
Cache Occupancy Attack: fingerprinting of processes

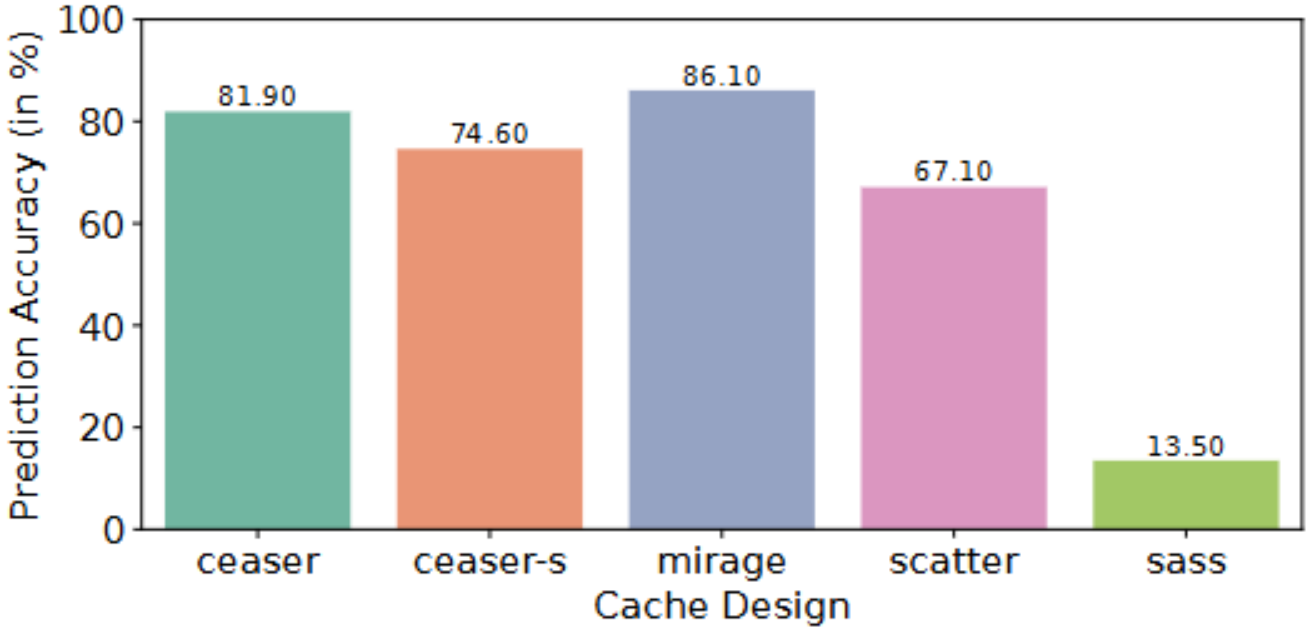




Cache Occupancy Attack: fingerprinting of processes

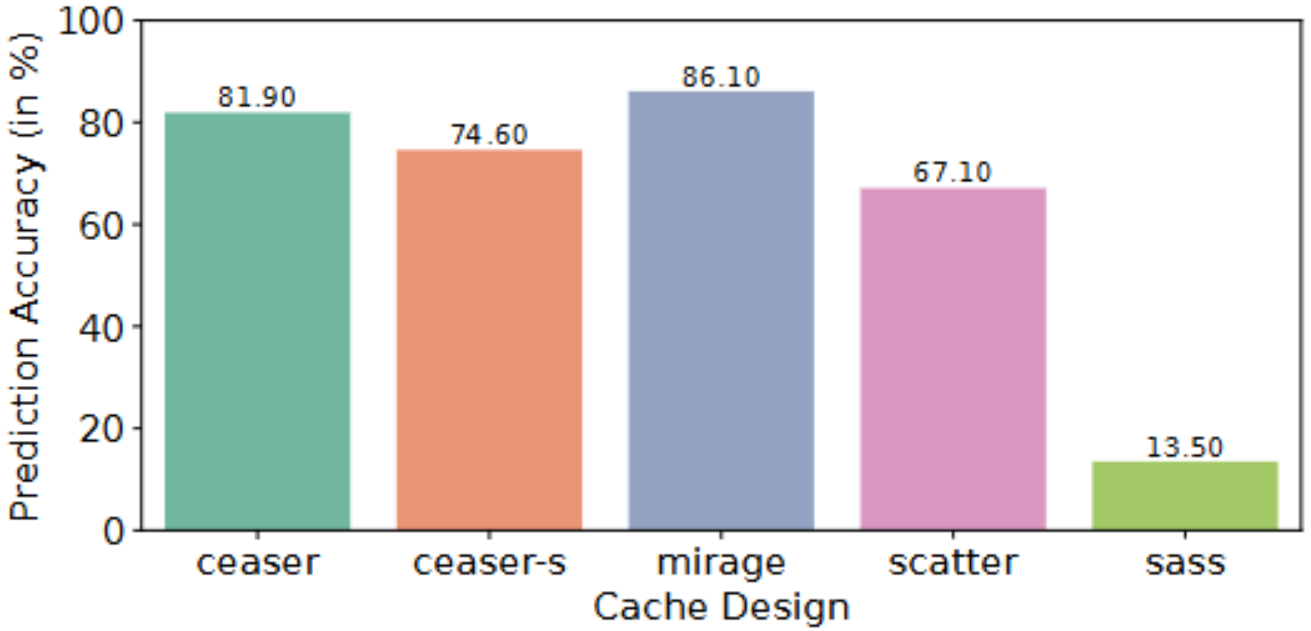






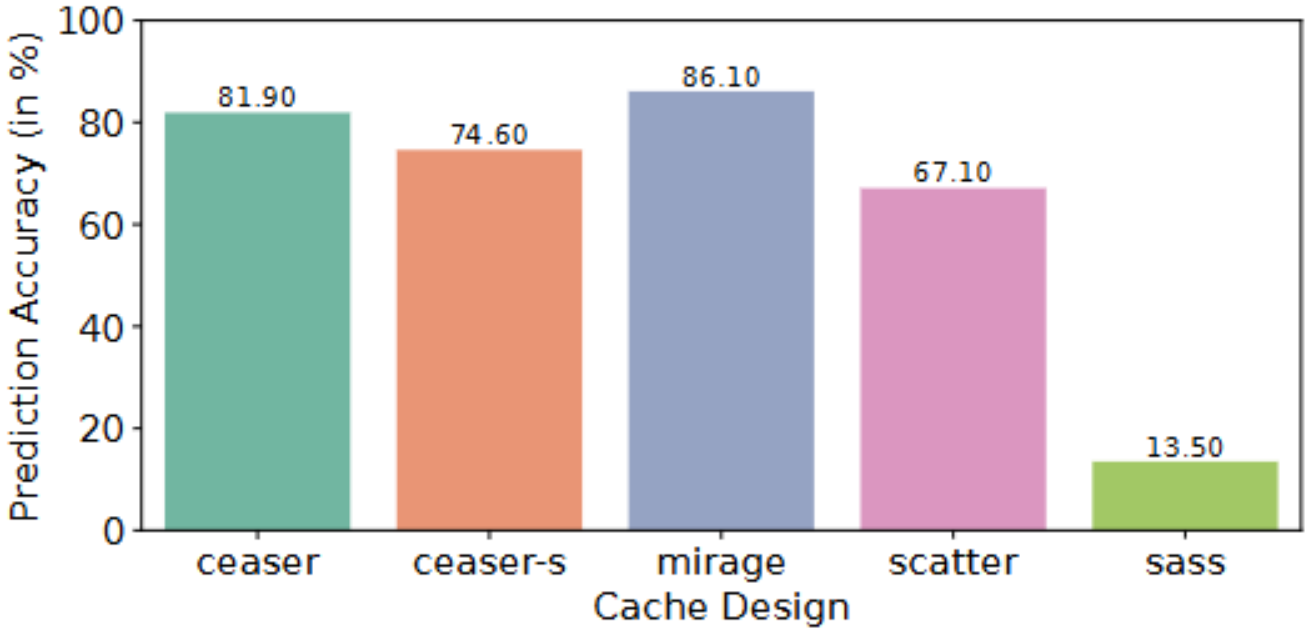
Fingerprinting accuracy across different cache designs of SPEC2017 workloads.

Cache Occupancy Attack: fingerprinting of processes



Fingerprinting accuracy across different cache designs of SPEC2017 workloads.

Cache Occupancy Attack: fingerprinting of processes



Fingerprinting accuracy across different cache designs of SPEC2017 workloads.

Takeaway

- SassCache exhibits a remarkably less prediction accuracy, implying no statistically significant adversarial advantage wrt. process fingerprinting.

Cache Occupancy Attack: AES key recovery

Cache Occupancy Attack: AES key recovery

1. The attacker fills the LLC with spurious occupancy. This ensures that the AES victim T-Tables are flushed from the LLC.

Cache Occupancy Attack: AES key recovery

1. The attacker fills the LLC with spurious occupancy. This ensures that the AES victim T-Tables are flushed from the LLC.
2. Given a fixed occupancy $X\%$, the attacker accesses it to ensure occupancy of complete L1d and X% of LLC.

Cache Occupancy Attack: AES key recovery

1. The attacker fills the LLC with spurious occupancy. This ensures that the AES victim T-Tables are flushed from the LLC.
2. Given a fixed occupancy $X\%$, the attacker accesses it to ensure occupancy of complete L1d and X% of LLC.
3. Chosen Plaintext Attack - AES victim run a single encryption of P and obtains the ciphertext C

Cache Occupancy Attack: AES key recovery

1. The attacker fills the LLC with spurious occupancy. This ensures that the AES victim T-Tables are flushed from the LLC.
2. Given a fixed occupancy $X\%$, the attacker accesses it to ensure occupancy of complete L1d and X% of LLC.
3. Chosen Plaintext Attack - AES victim run a single encryption of P and obtains the ciphertext C
4. the attacker measures access-time to its previously allocated memory -> this forms a trace T

Cache Occupancy Attack: AES key recovery

1. The attacker fills the LLC with spurious occupancy. This ensures that the AES victim T-Tables are flushed from the LLC.
2. Given a fixed occupancy $X\%$, the attacker accesses it to ensure occupancy of complete L1d and X% of LLC.
3. Chosen Plaintext Attack - AES victim run a single encryption of P and obtains the ciphertext C
4. the attacker measures access-time to its previously allocated memory -> this forms a trace T
5. A single observation point for the attacker is the tuple (P, C, T): the time T taken to access its own cache occupancy after the AES victim operates upon plaintext P for an unknown key K

Cache Occupancy Attack: AES key recovery

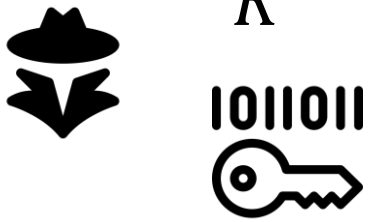
Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery

Creates templates of access time T_j^* for each byte of known key K^* for last round attack

Total number of templates created by the adversary = 256×16

Cache Occupancy Attack: AES key recovery



Creates templates of access time T_j^* for each byte of known key K^* for last round attack

Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery

K^*

Creates templates of access time T_j^* for each byte of known key K^* for last round attack

Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery

Creates templates of access time T_j^* for each byte of known key K^* for last round attack

Total number of templates created by the adversary = 256×16

Cache Occupancy Attack: AES key recovery

K^*

T_j^*

Creates templates of access time T_j^* for each byte of known key K^* for last round attack

Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery

K^*

T_j^*

Creates templates of access time T_j^* for each byte of known key K^* for last round attack

Victim runs AES with unknown key K and attacker creates observations T_j^i s.t. $0 \leq i \leq 15, 0 \leq j \leq 255$

Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery

K^*

T_j^*

Creates templates of access time T_j^* for each byte of known key K^* for last round attack

K

Victim runs AES with unknown key K and attacker creates observations T_j^i s.t. $0 \leq i \leq 15, 0 \leq j \leq 255$

Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery

K^*

T_j^*

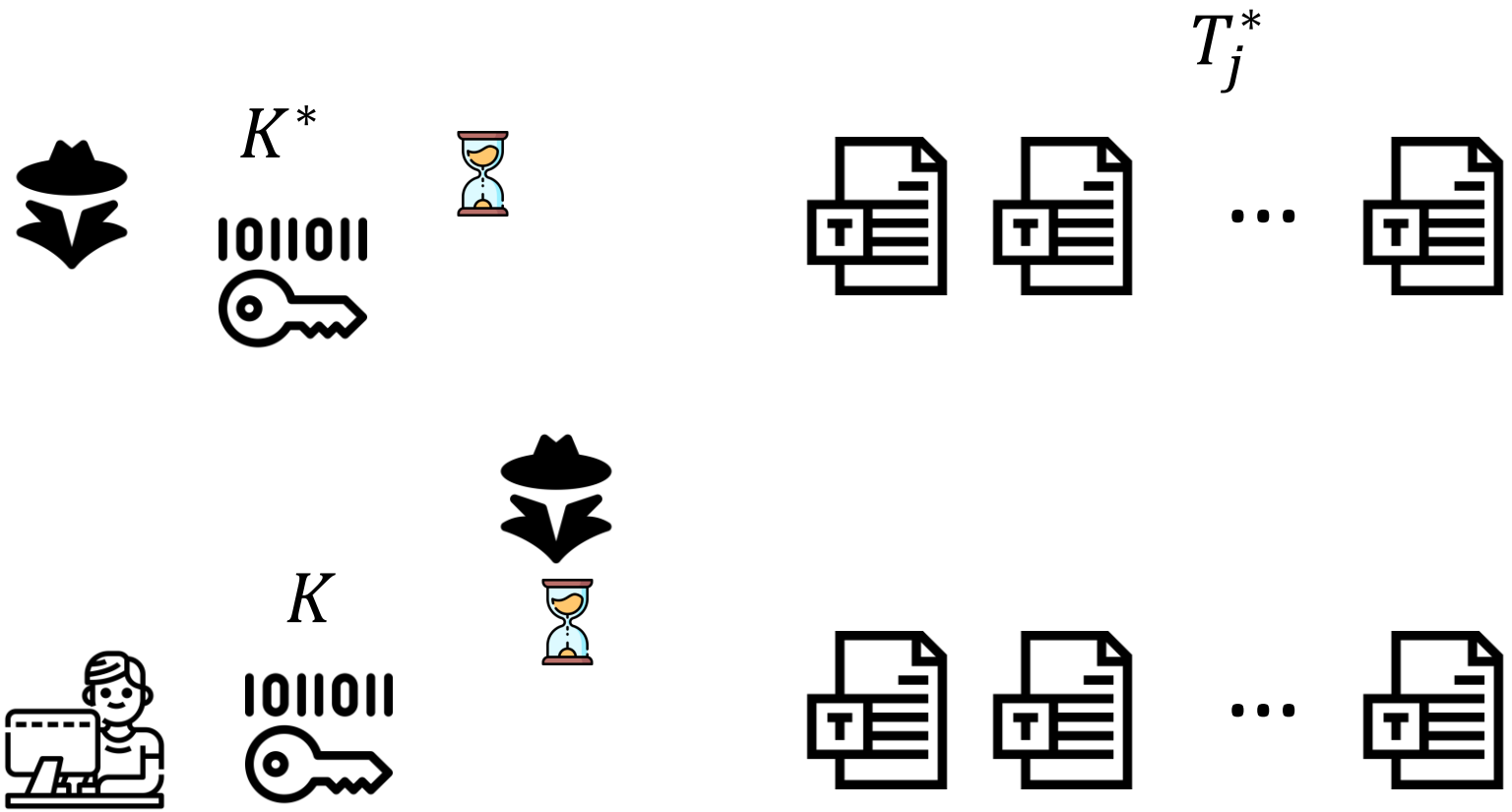
Creates templates of access time T_j^* for each byte of known key K^* for last round attack

K

Victim runs AES with unknown key K and attacker creates observations T_j^i s.t. $0 \leq i \leq 15, 0 \leq j \leq 255$

Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery

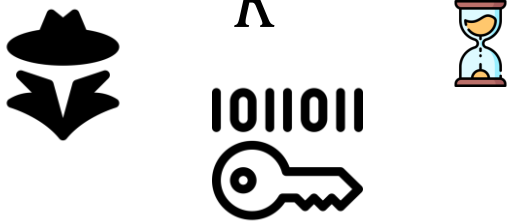


Creates templates of access time T_j^* for each byte of known key K^* for last round attack

Victim runs AES with unknown key K and attacker creates observations T_j^i s.t. $0 \leq i \leq 15, 0 \leq j \leq 255$

Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery



K^*

T_j^*

Creates templates of access time T_j^* for each byte of known key K^* for last round attack

K

T_j^i

Victim runs AES with unknown key K and attacker creates observations T_j^i s.t. $0 \leq i \leq 15, 0 \leq j \leq 255$

Total number of templates created by the adversary $= 256 \times 16$

Cache Occupancy Attack: AES key recovery

Cache Occupancy Attack: AES key recovery

- For each key byte index $i \in \{0, 1, 2, \cdots, 15\}$, the attacker correlates $\{T_0^i, T_1^i, T_2^i, \cdots T_{255}^i\}$ with T_i^*

Cache Occupancy Attack: AES key recovery

- For each key byte index $i \in \{0, 1, 2, \cdots, 15\}$, the attacker correlates $\{T_0^i, T_1^i, T_2^i, \cdots T_{255}^i\}$ with T_i^*
- Sort and calculate ranks of the correct key bytes as $R = R_0, R_1, R_2, \cdots, R_{15}$

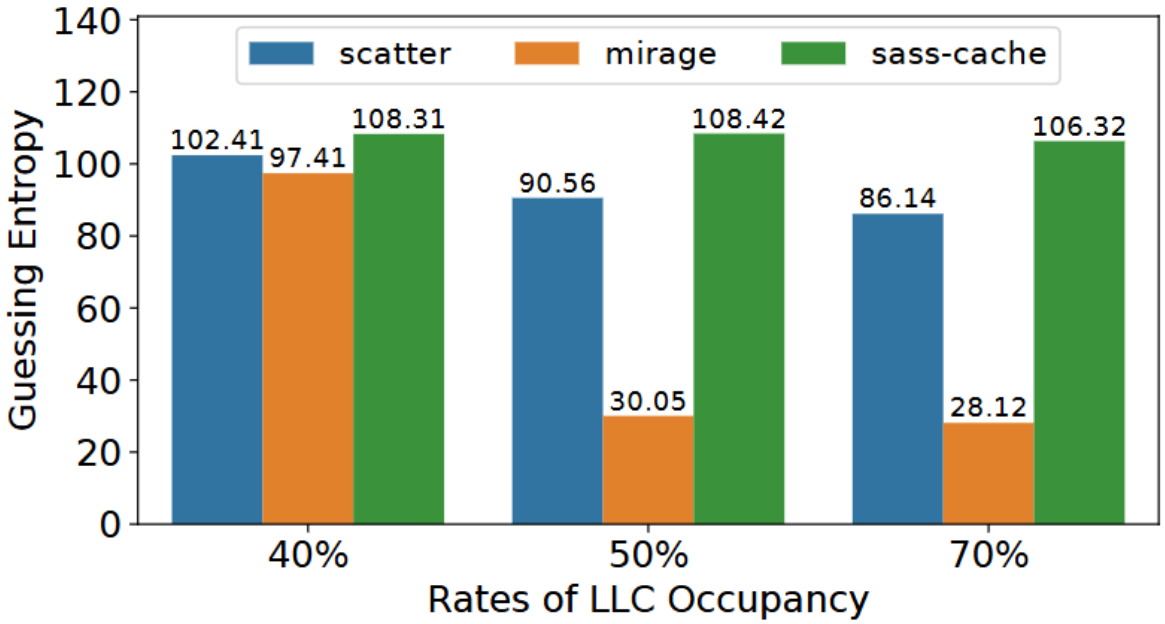
Cache Occupancy Attack: AES key recovery

- For each key byte index $i \in \{0, 1, 2, \cdots, 15\}$, the attacker correlates $\{T_0^i, T_1^i, T_2^i, \cdots T_{255}^i\}$ with T_i^*
- Sort and calculate ranks of the correct key bytes as $R = R_0, R_1, R_2, \cdots, R_{15}$
- Calculate Guessing Entropy $(\text{GE}) = \sum_{i=0}^{16} \log_2(R_i)$

Cache Occupancy Attack: AES key recovery

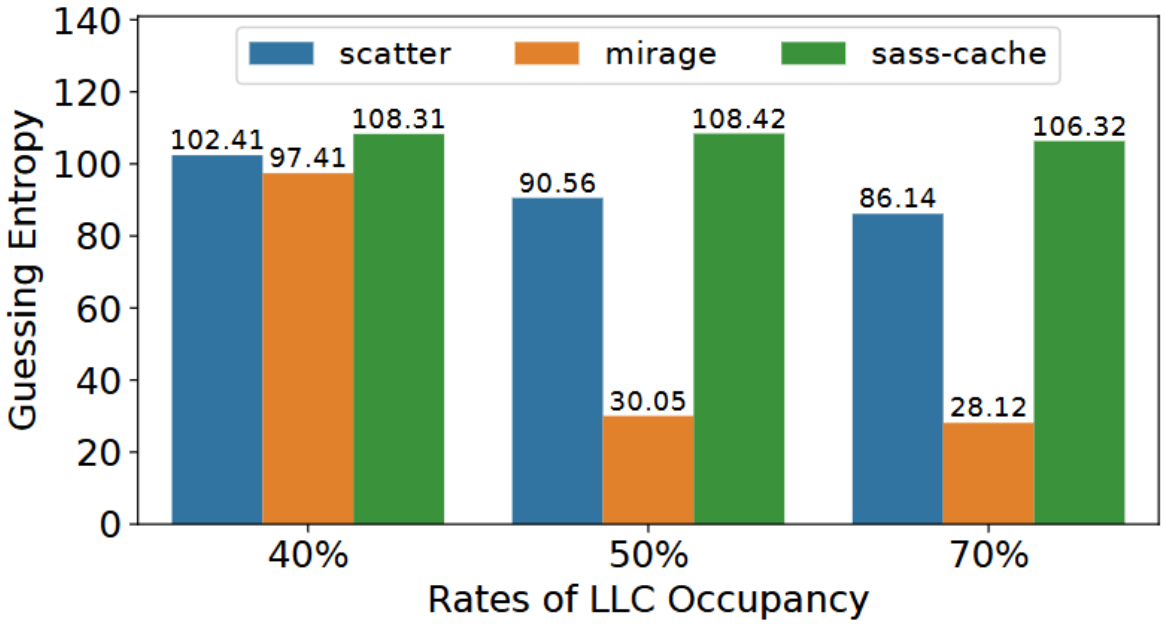
- For each key byte index $i \in \{0, 1, 2, \cdots, 15\}$, the attacker correlates $\{T_0^i, T_1^i, T_2^i, \cdots T_{255}^i\}$ with T_i^*
- Sort and calculate ranks of the correct key bytes as $R = R_0, R_1, R_2, \cdots, R_{15}$
- Calculate Guessing Entropy $(\text{GE}) = \sum_{i=0}^{16} \log_2(R_i)$
- Lower GE implies the key bytes are lower ranked post correlation, thereby enabling faster key recovery

Cache Occupancy Attack: AES key recovery

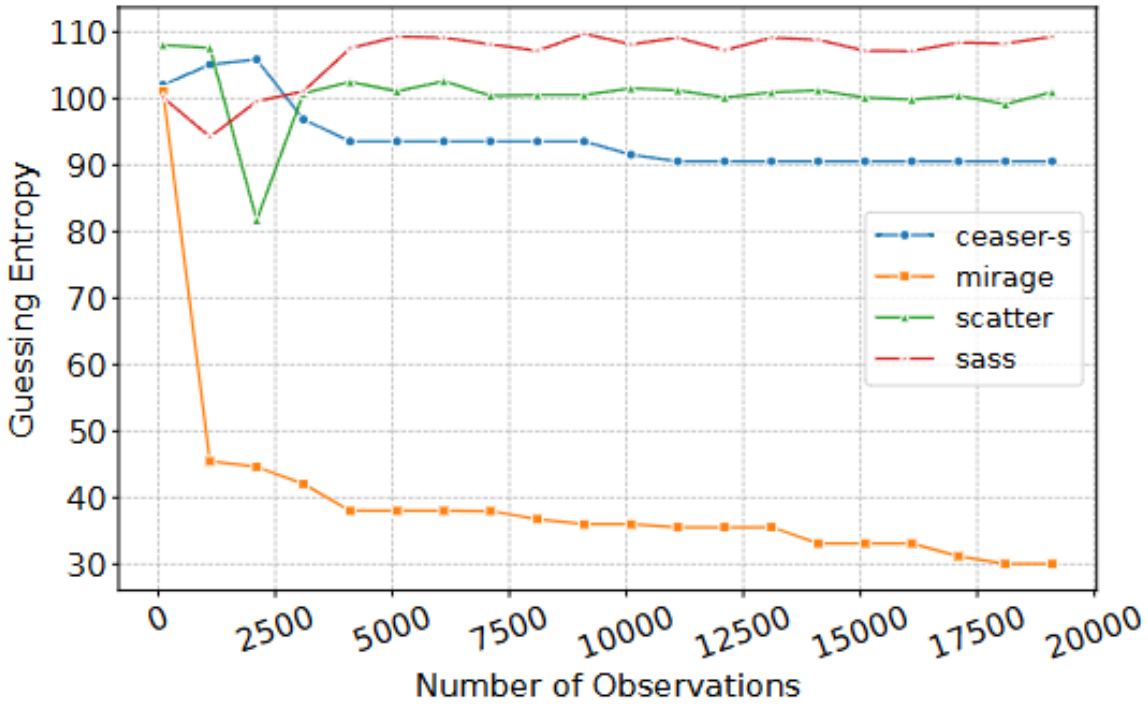


Key Recovery experiment across three occupancy rates: 40%, 50%, 70%

Cache Occupancy Attack: AES key recovery

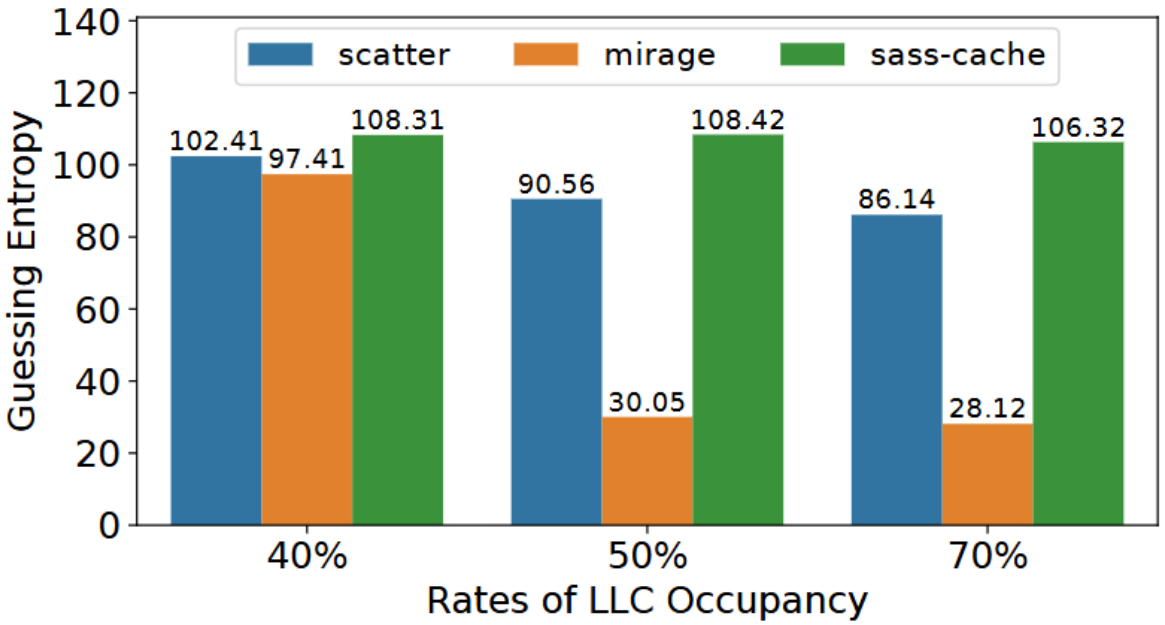


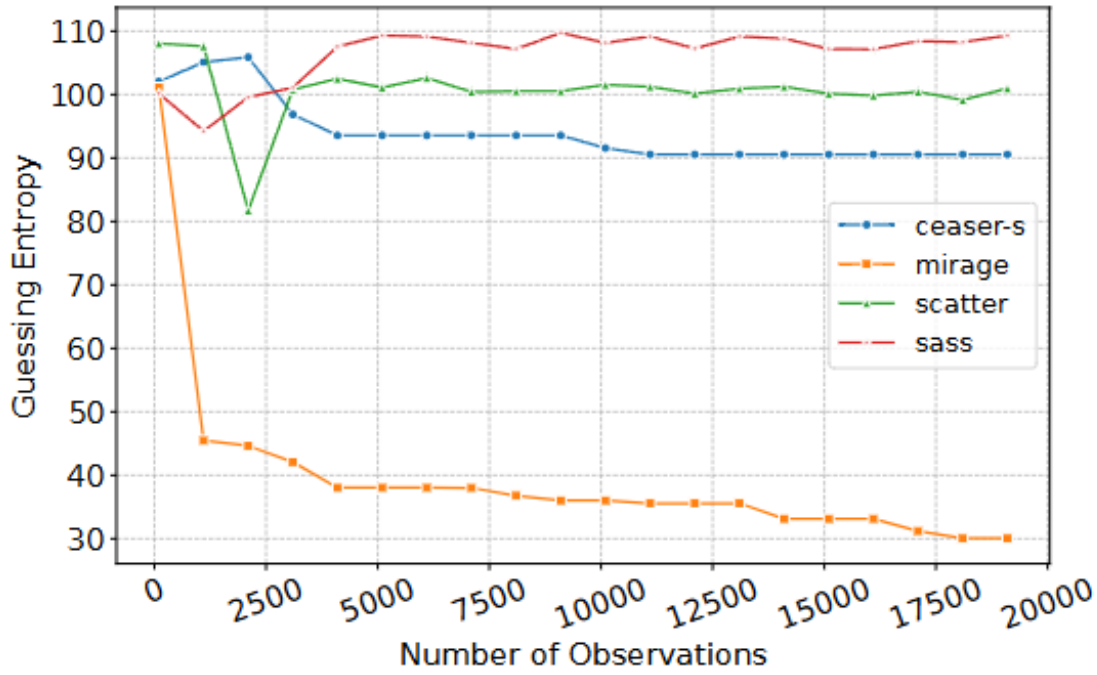
Key Recovery experiment across three occupancy rates: 40%, 50%, 70%



Key Recovery experiment across a fixed occupancy rate: 50%

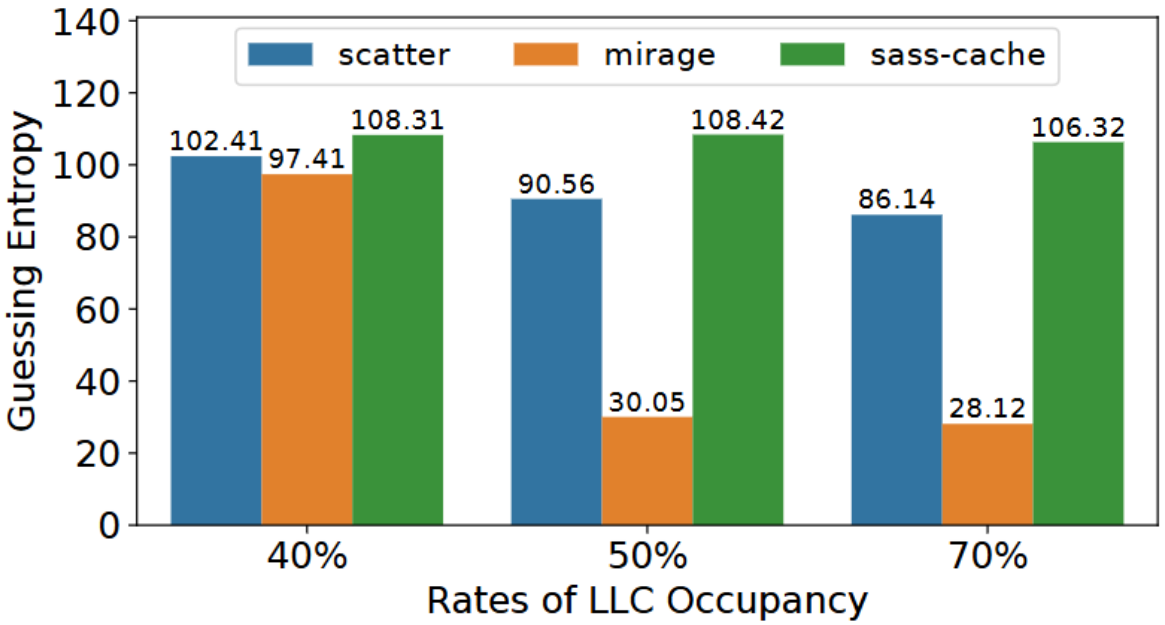
Cache Occupancy Attack: AES key recovery

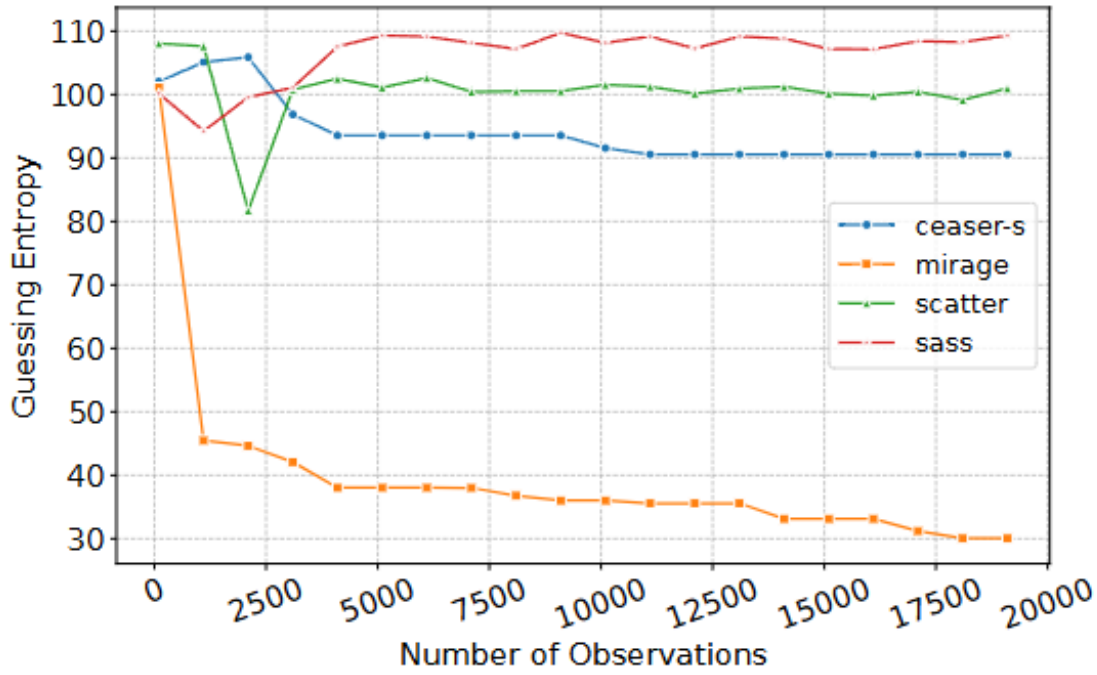




Takeaway

Cache Occupancy Attack: AES key recovery





Takeaway

Designs based on set-associative caches are more resilient to cache occupancy attacks compared to designs based on pseudo-fully associative caches.

Summary

- **Performance and security analysis** of randomized cache designs across multiple replacement policies and Cache Occupancy attacks.
- **First Key recovery** attacks on T-table based **AES-128** using different occupancy levels and guessing entropy
- Establishes Cache Occupancy attack as a **serious concern** like contention attacks for secured cache designs.

Summary

- **Performance and security analysis** of randomized cache designs across multiple replacement policies and Cache Occupancy attacks.
- **First Key recovery** attacks on T-table based **AES-128** using different occupancy levels and guessing entropy
- Establishes Cache Occupancy attack as a **serious concern** like contention attacks for secured cache designs.

Thank you for your attention!