

A Security Model for Randomization-based Protected Caches

Jordi Ribes-González, Oriol Farràs
Universitat Rovira i Virgili

Carles Hernández
Universitat Politècnica de València

Vatistas Kostalabros, Miquel Moretó
Barcelona Supercomputing Center

Cryptographic Hardware and Embedded Systems (CHES)

19th September 2022

Table of Contents

1 Introduction to Cache Side-channels and RPCs

2 Our Model for RPCs

3 Security Definition and Analysis

4 Pseudo-random and Multi-epoch Cases

5 Performance Analysis

6 Conclusions

Table of Contents

1 Introduction to Cache Side-channels and RPCs

2 Our Model for RPCs

3 Security Definition and Analysis

4 Pseudo-random and Multi-epoch Cases

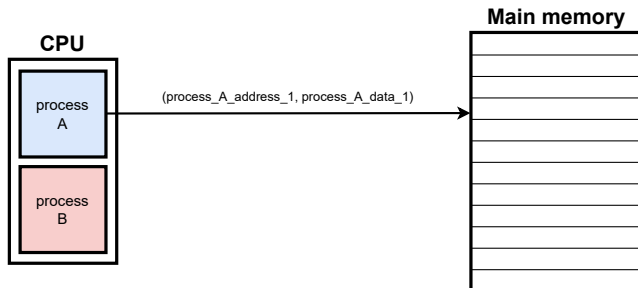
5 Performance Analysis

6 Conclusions

An Overview of Cache Side-channel Attacks



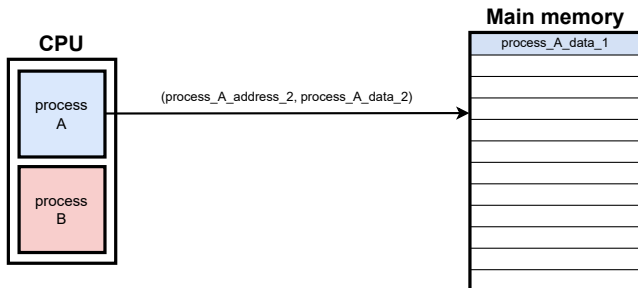
An Overview of Cache Side-channel Attacks



An Overview of Cache Side-channel Attacks



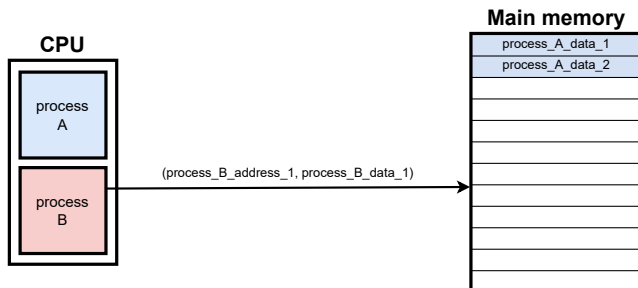
An Overview of Cache Side-channel Attacks



An Overview of Cache Side-channel Attacks



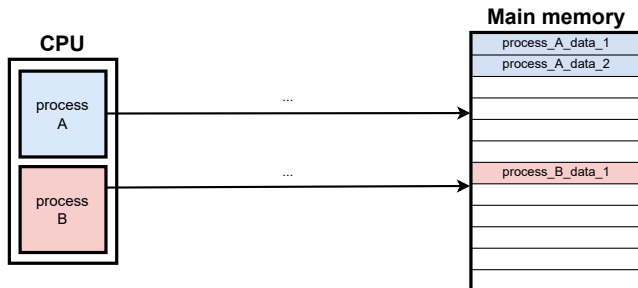
An Overview of Cache Side-channel Attacks



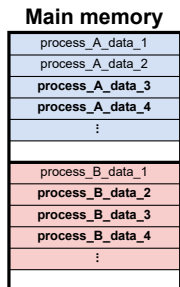
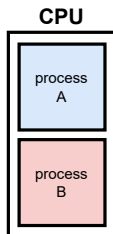
An Overview of Cache Side-channel Attacks



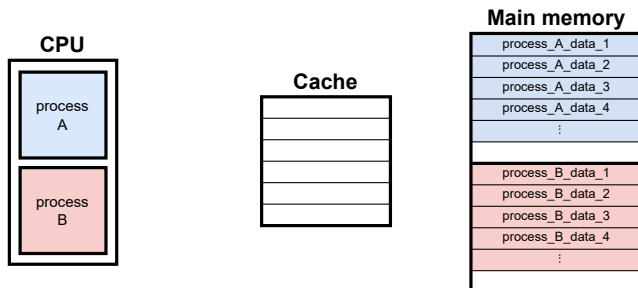
An Overview of Cache Side-channel Attacks



An Overview of Cache Side-channel Attacks

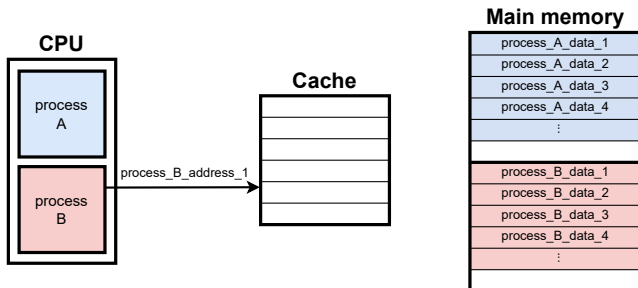


An Overview of Cache Side-channel Attacks



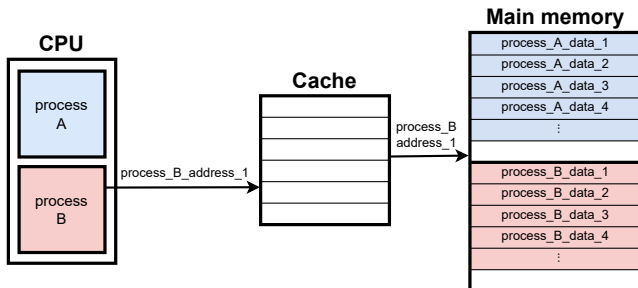
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



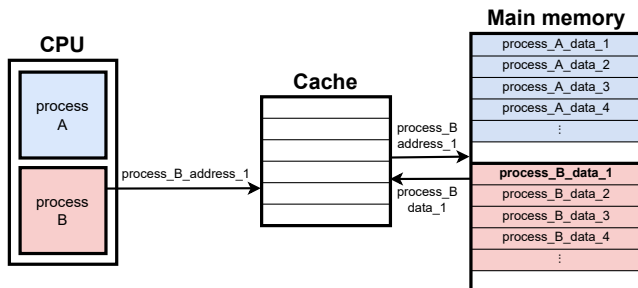
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



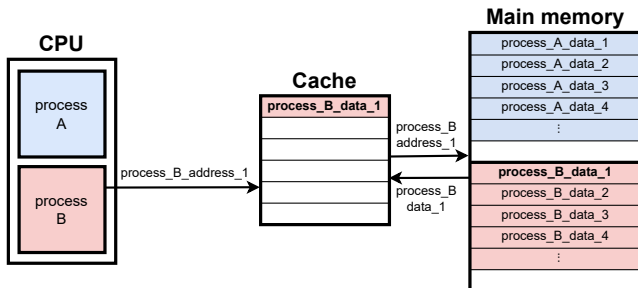
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



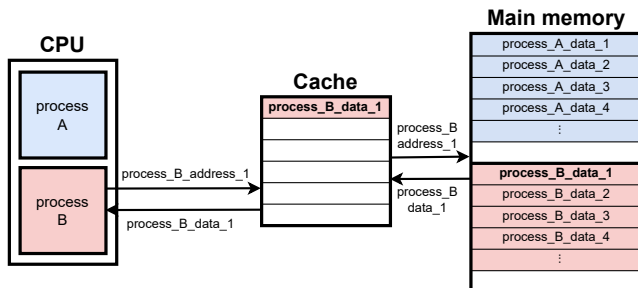
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



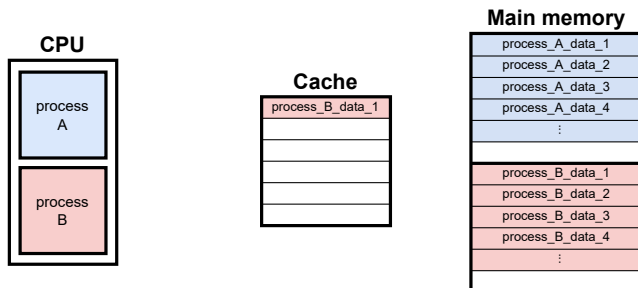
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



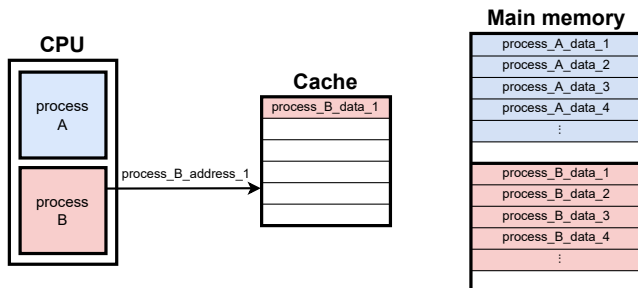
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



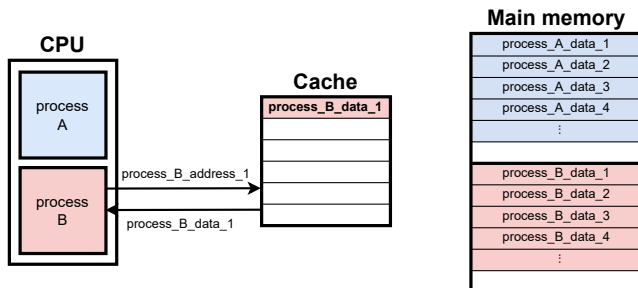
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



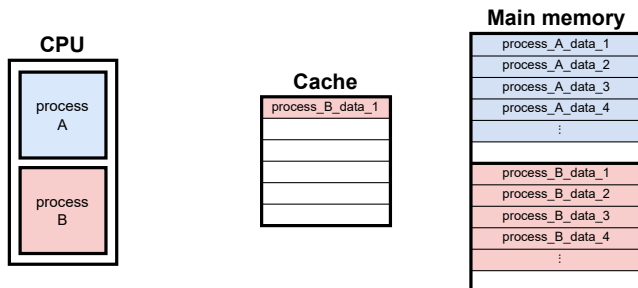
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



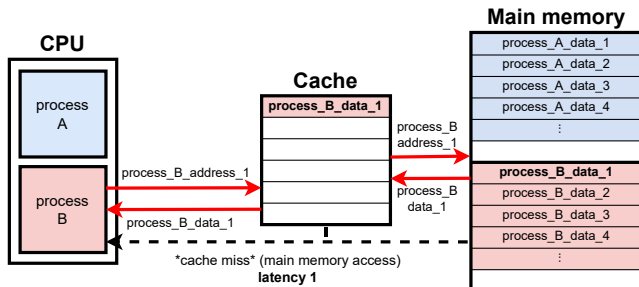
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



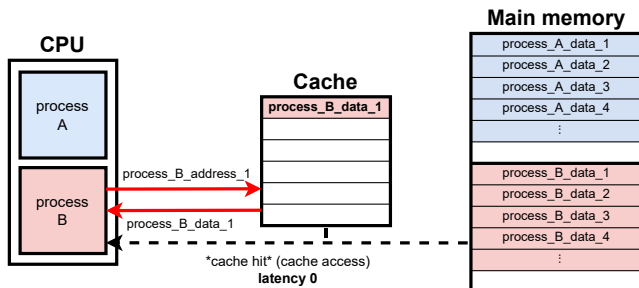
- Cache memories **reduce the latency** of memory accesses

An Overview of Cache Side-channel Attacks



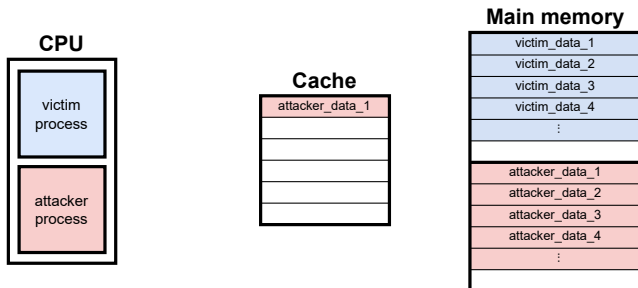
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



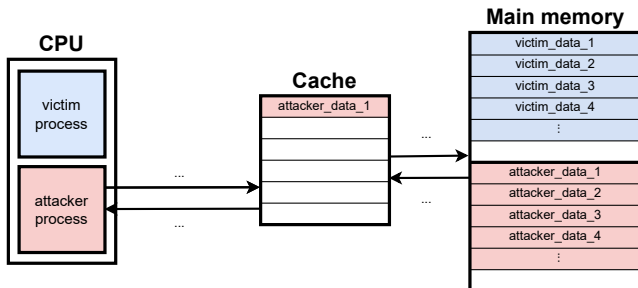
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



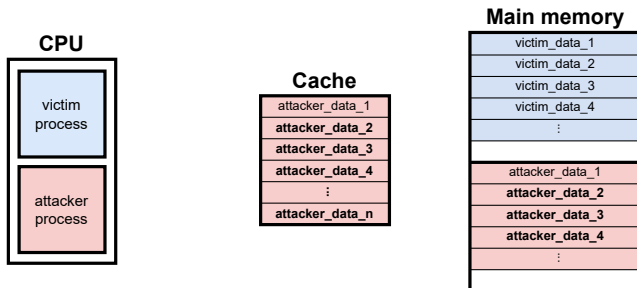
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



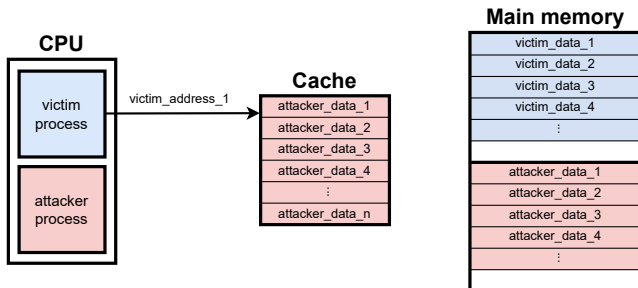
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



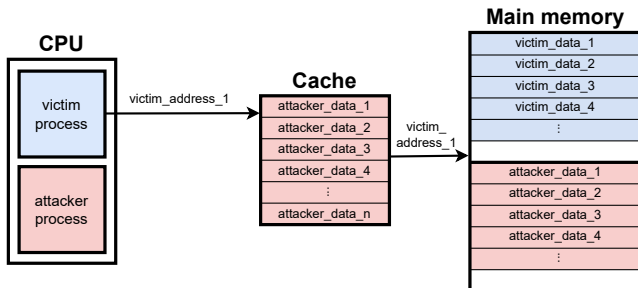
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



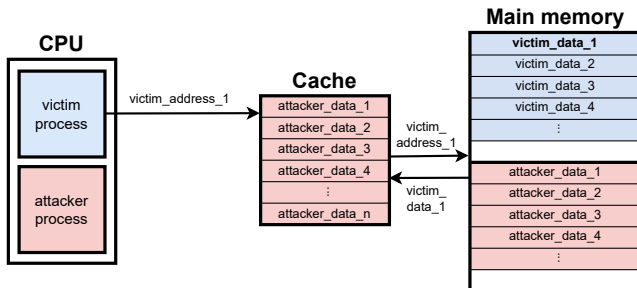
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



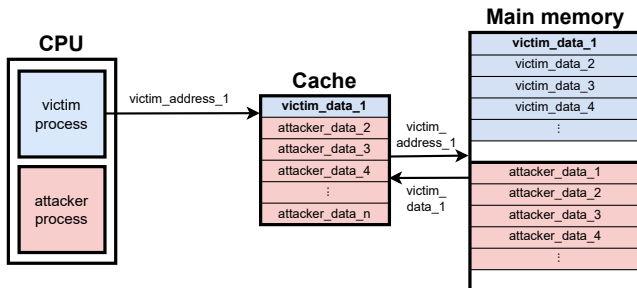
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



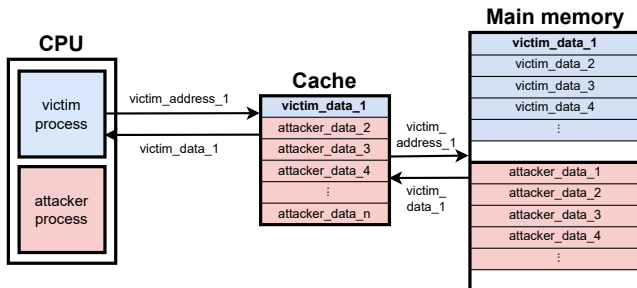
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



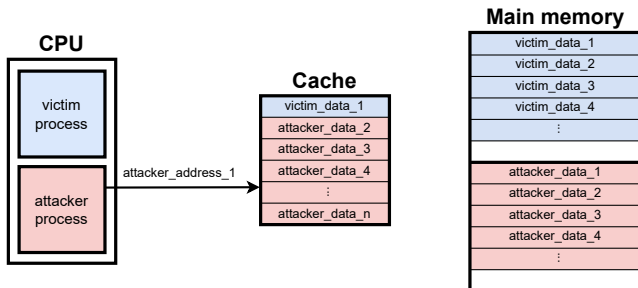
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



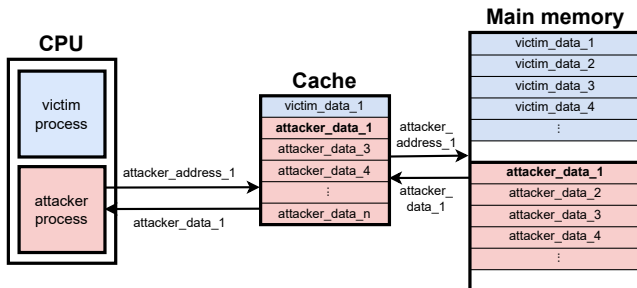
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



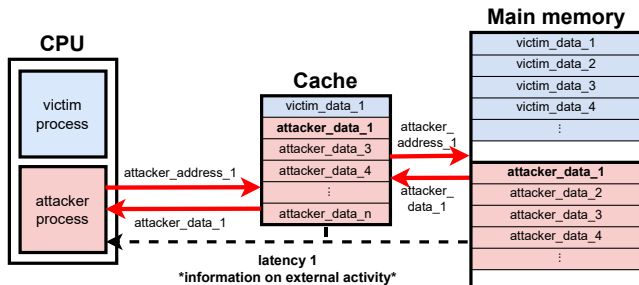
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



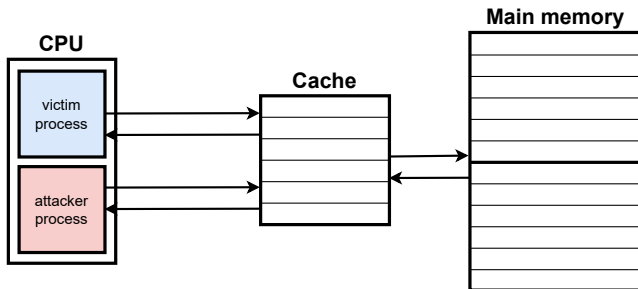
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**

An Overview of Cache Side-channel Attacks



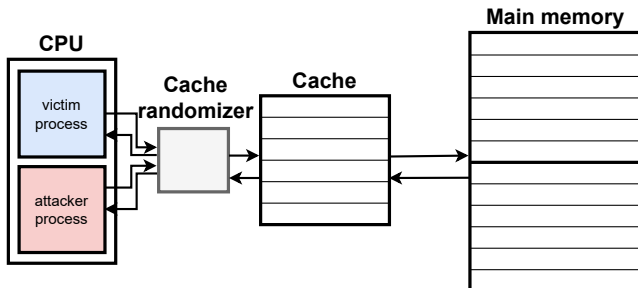
- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**
- Exploit access latency to eavesdrop on **external** processes

An Overview of Cache Side-channel Attacks



- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**
- Exploit access latency to eavesdrop on **external** processes

An Overview of Cache Side-channel Attacks



- Cache memories **reduce the latency** of memory accesses
- Cache side-channel: access **latency** reveals if data is **already cached**
- Exploit access latency to eavesdrop on **external** processes
- Randomization-based Protected Caches: **randomize** cache addresses

Outline of Our Work

Study the security of RPCs

- against **single-target access-based** attacks in
- **shared large** caches (LLC)

In particular, resistance against **Prime+Probe** and **Evict+Probe** attacks.

Outline of Our Work

Study the security of RPCs

- against **single-target access-based** attacks in
- **shared large** caches (LLC)

In particular, resistance against **Prime+Probe** and **Evict+Probe** attacks.

Several previous RPCs have been found **insecure**. Aims:

- address the **break-and-repair cycle**
- analyze the **impact of access-based attacks**

Outline of Our Work

Study the security of RPCs

- against **single-target access-based** attacks in
- **shared large** caches (LLC)

In particular, resistance against **Prime+Probe** and **Evict+Probe** attacks.

Several previous RPCs have been found **insecure**. Aims:

- address the **break-and-repair cycle**
- analyze the **impact of access-based attacks**

Provable security approach to RPCs:

1. **model RPCs**
2. characterize security through **game-based definitions**
3. analyze security through **security proofs** and **attacks**
4. evaluate performance through a **simulation**

Cache side-channel attacks exploit:

- Own **access** latency
- **Timing** external processes
- Cache **flushing**
- Cache **collisions**
- Cache **coherence**

Cache Side-channels and Mitigation Measures

Cache side-channel attacks exploit:

- Own **access latency**
- **Timing** external processes
- Cache **flushing**
- Cache **collisions**
- Cache **coherence**

Focus on **access-based attacks** as Prime+Probe and Evict+Probe.

Cache side-channel attacks exploit:

- Own **access** latency
- **Timing** external processes
- Cache **flushing**
- Cache **collisions**
- Cache **coherence**

Focus on **access-based attacks** as Prime+Probe and Evict+Probe.

Mitigation strategies:

- cache **partitioning**
- **table-based** randomization
- **randomization**-based protected caches

Cache side-channel attacks exploit:

- Own **access** latency
- **Timing** external processes
- Cache **flushing**
- Cache **collisions**
- Cache **coherence**

Focus on **access-based attacks** as Prime+Probe and Evict+Probe.

Mitigation strategies:

- cache **partitioning** (bad for performance)
- **table-based** randomization (inefficient for LLC)
- **randomization-based protected caches**

Table of Contents

1 Introduction to Cache Side-channels and RPCs

2 Our Model for RPCs

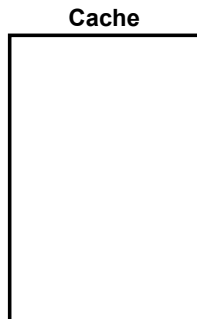
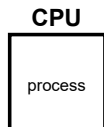
3 Security Definition and Analysis

4 Pseudo-random and Multi-epoch Cases

5 Performance Analysis

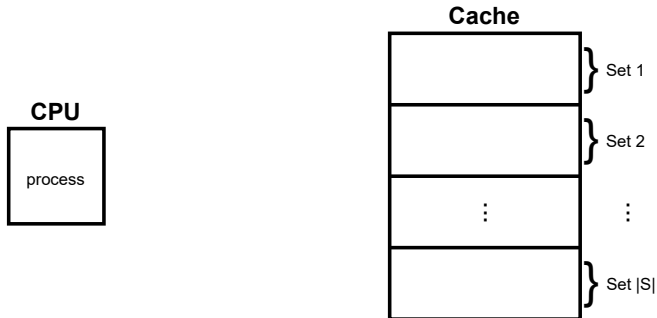
6 Conclusions

Our Cache Model and Access-based Attacks



Cache memories

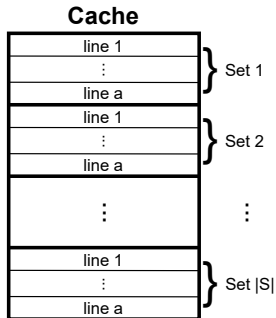
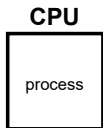
Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ cache sets,

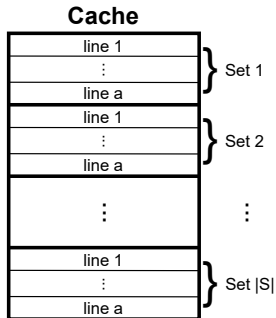
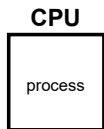
Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),

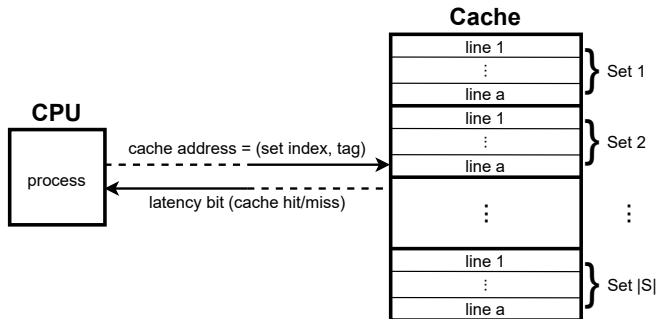
Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Our Cache Model and Access-based Attacks



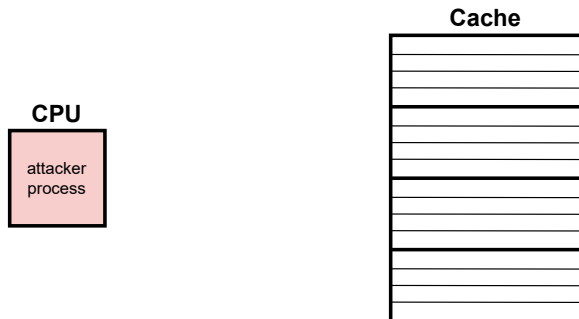
Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

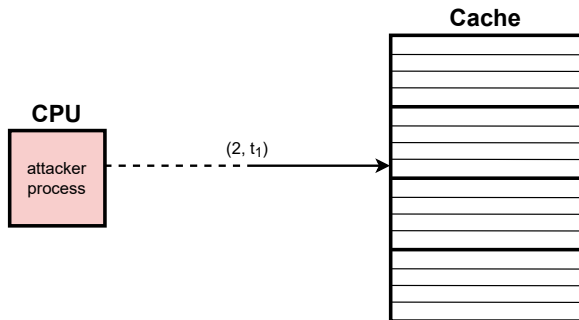
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

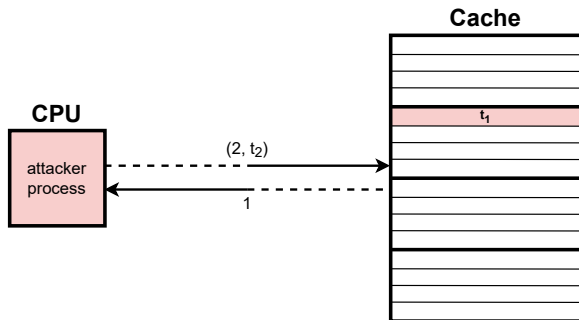
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



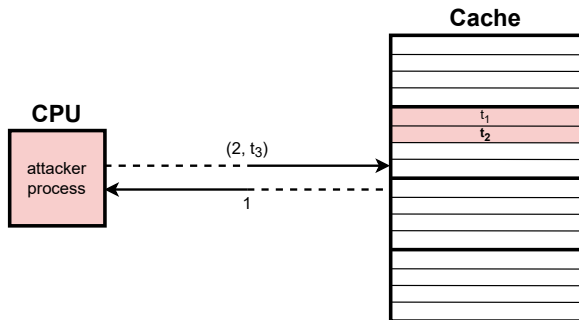
Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

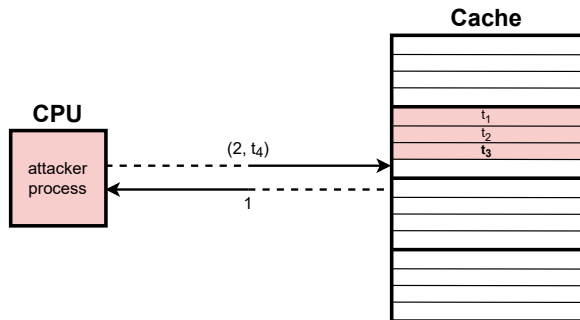
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses

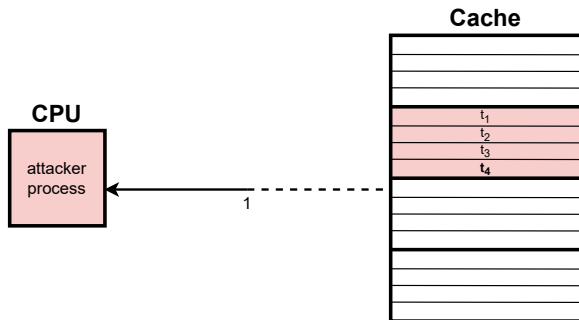
 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack

 in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

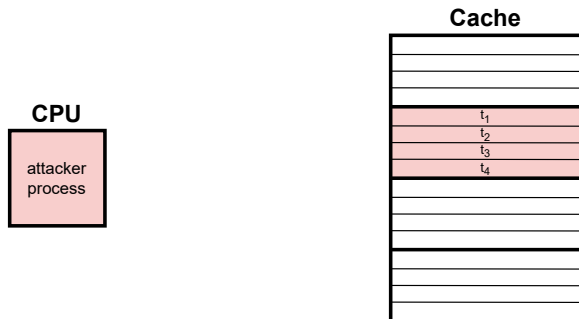
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

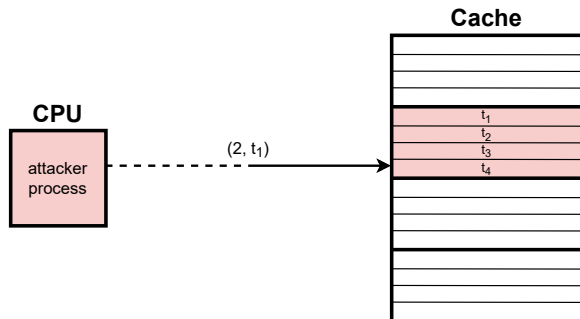
- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).



Cache memories

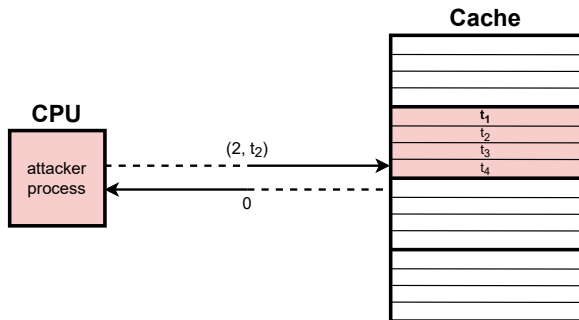
- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

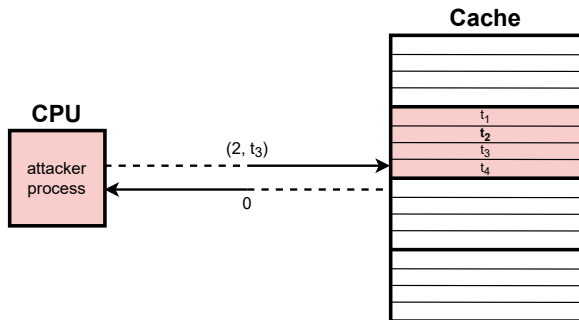
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

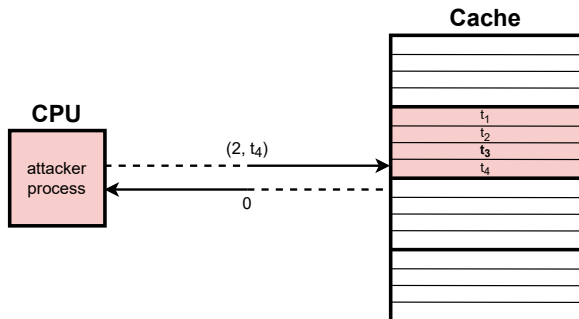
- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

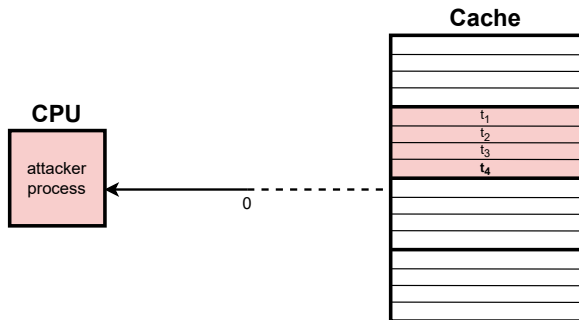
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses

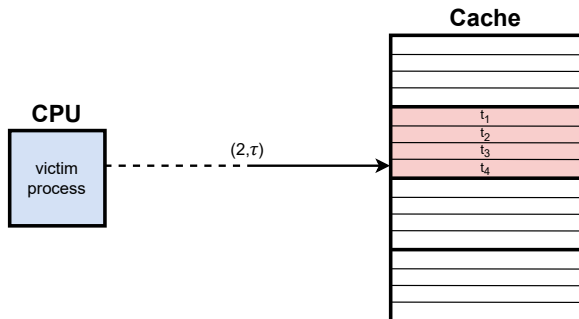
 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack

 in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses

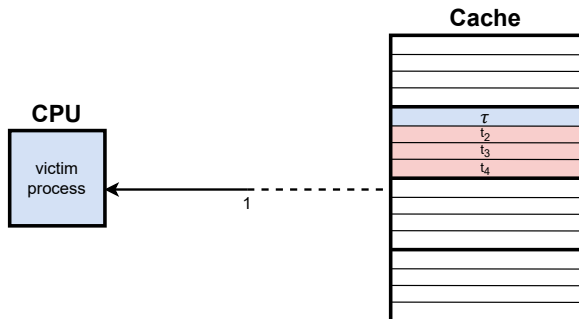
 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack

 in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack

 in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

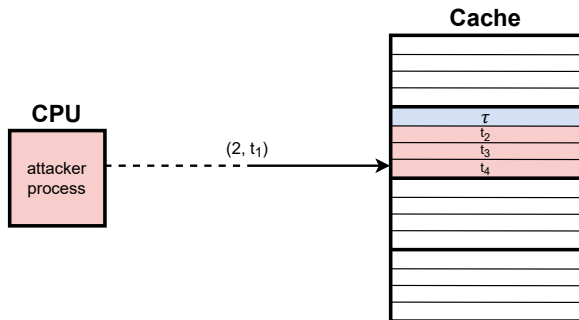
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

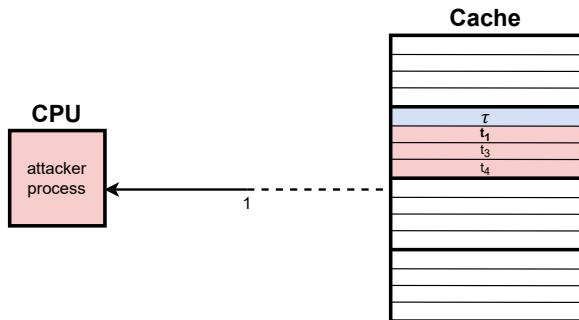
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

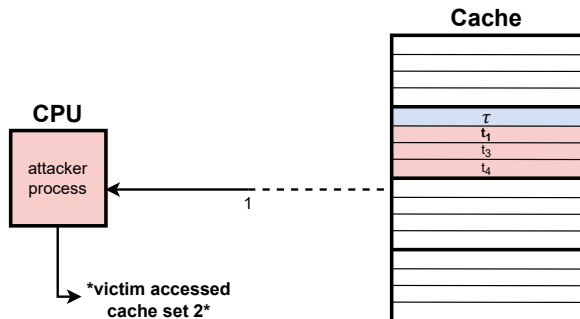
Cache addresses

 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack in this context (with $|S| = a = 4$).

Our Cache Model and Access-based Attacks



Cache memories

- consist of $|S|$ **cache sets**, with a **lines** each (associativity),
- and a **replacement policy** RP.

Cache addresses

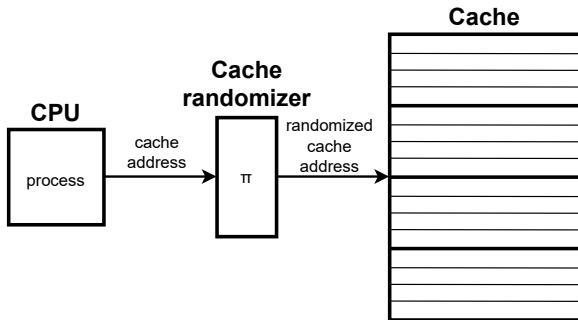
 comprise

- a **set index** which addresses cache sets, and
- a **tag** which is stored in some line according to RP.

Access-based attack

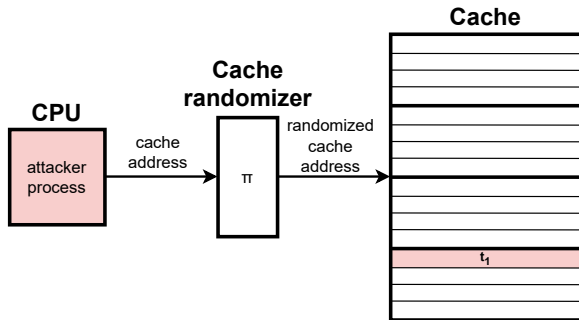
 in this context (with $|S| = a = 4$).

Randomization-based Protected Caches



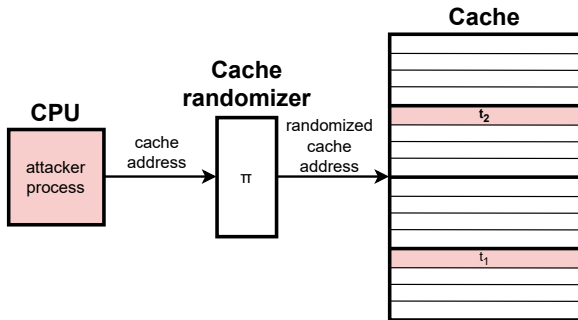
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



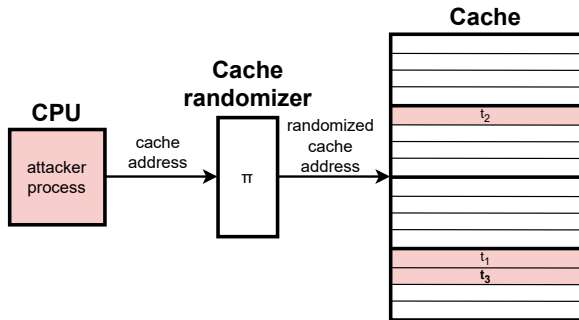
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



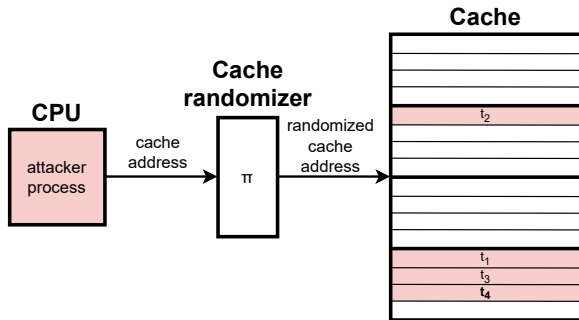
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



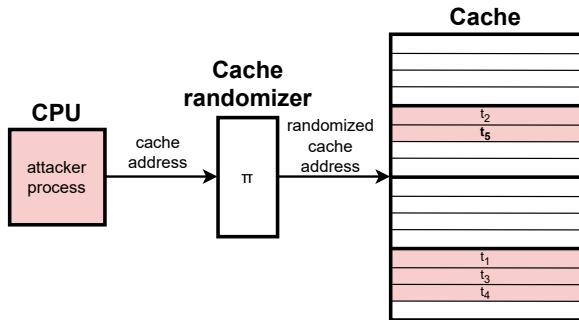
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



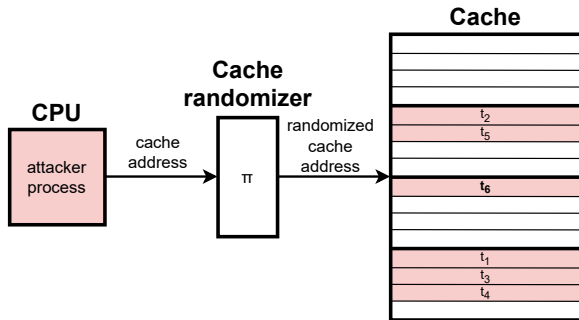
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



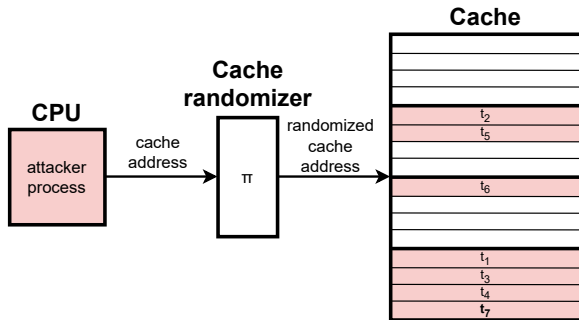
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



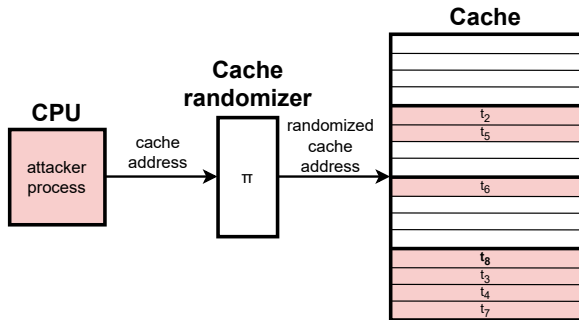
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



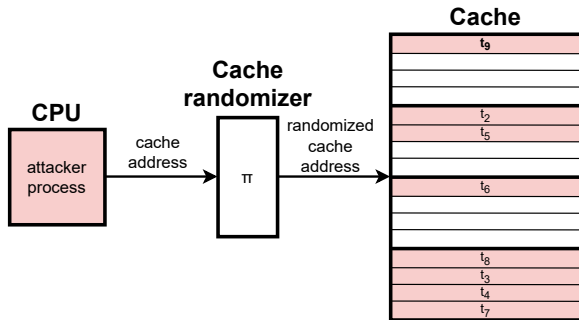
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



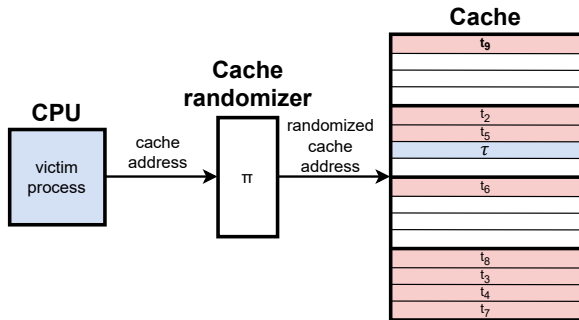
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



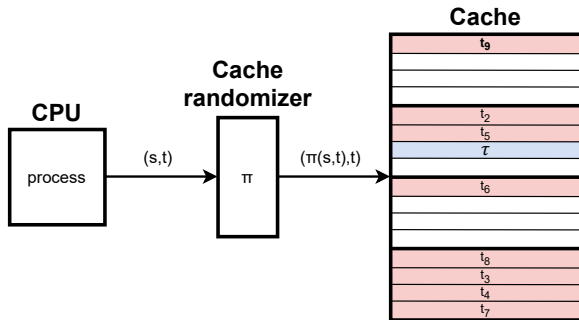
Address randomization: Hamper attacks by **scattering** accesses.

Randomization-based Protected Caches



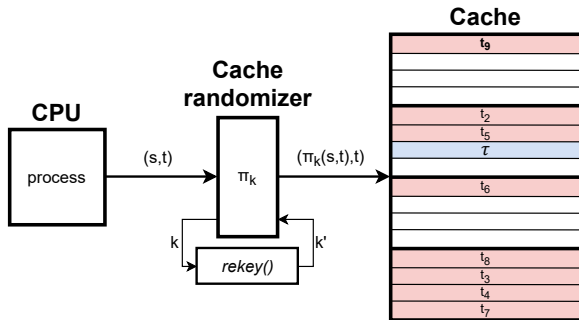
Address randomization: Hamper attacks by **scattering** accesses.
Access-based attacks *take longer* and are *more difficult*!

Randomization-based Protected Caches



Address randomization: Hamper attacks by **scattering** accesses.
Access-based attacks *take longer* and are *more difficult*!

- **Set-index randomization** $(s, t) \rightarrow (\pi(s, t), t)$



Address randomization: Hamper attacks by **scattering** accesses.
Access-based attacks *take longer* and are *more difficult*!

- **Set-index randomization** $(s, t) \rightarrow (\pi_k(s, t), t)$
- Modeled as a keyed **pseudo-random function** (rekey, π)

Table of Contents

1 Introduction to Cache Side-channels and RPCs

2 Our Model for RPCs

3 Security Definition and Analysis

4 Pseudo-random and Multi-epoch Cases

5 Performance Analysis

6 Conclusions

Questions Addressed in Our Work

Even with RPCs, **attacks are possible** given enough cache accesses [Bourgeat et al.'20, Purnal et al.'21, our work].

RPCs establish a **rekeying period**.

Questions Addressed in Our Work

Even with RPCs, **attacks are possible** given enough cache accesses [Bourgeat et al.'20, Purnal et al.'21, our work].

RPCs establish a **rekeying period**.

Up until now, **rekeying has been set heuristically** to thwart **particular attacks**, leading to **insecure RPCs**.

Can some rekeying periods provide provable **security guarantees**?

Questions Addressed in Our Work

Even with RPCs, **attacks are possible** given enough cache accesses [Bourgeat et al.'20, Purnal et al.'21, our work].

RPCs establish a **rekeying period**.

Up until now, **rekeying has been set heuristically** to thwart **particular attacks**, leading to **insecure RPCs**.

Can some rekeying periods provide provable **security guarantees**?

Moreover, **key-invariant information** about the cache randomizer can be exploited [Bourgeat et al.'20].

Can security be enforced **across different epochs**?

Formally **define and prove security** against **all attacks** that

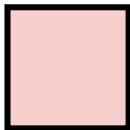
- aim to detect a victim access to a **single target address**
- are considered to succeed if their **advantage crosses some threshold**
- only **exploit access latency** information

This approach allows

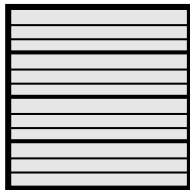
- to provide **concrete security guarantees**
- to **quantify the success of an attack** under specific conditions

Security Definition

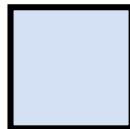
Attacker



RPC

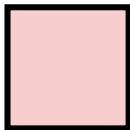


Victim

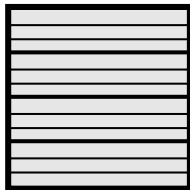


Security Definition

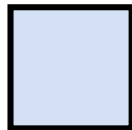
Attacker



RPC
flush

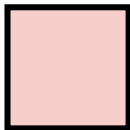


Victim



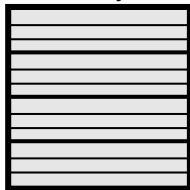
Security Definition

Attacker

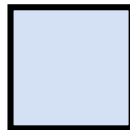


RPC

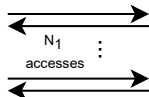
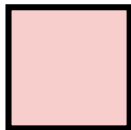
flush
rekey



Victim

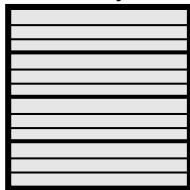


Attacker

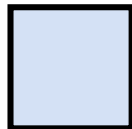


RPC

flush
rekey

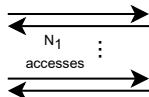
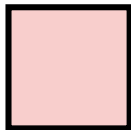


Victim

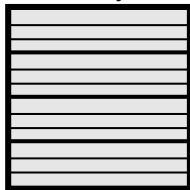


Security Definition

Attacker

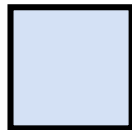


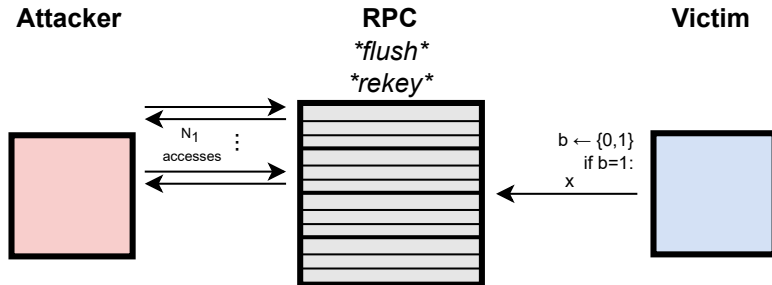
RPC
flush
rekey

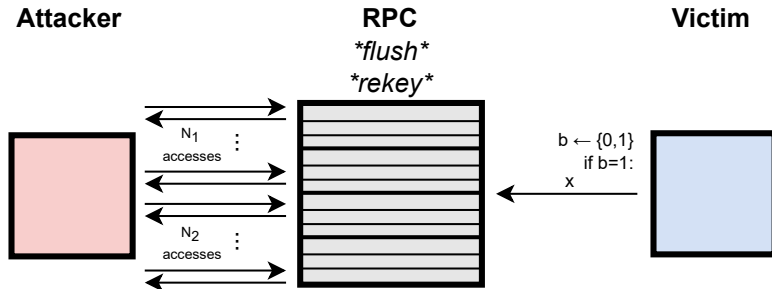


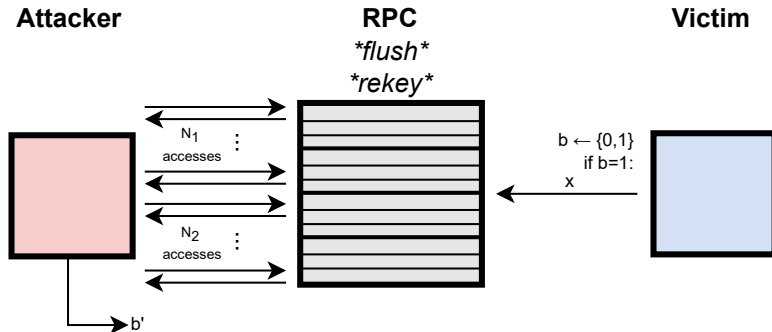
Victim

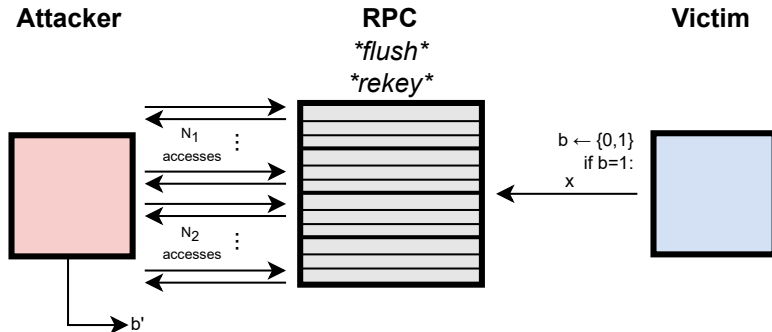
$b \leftarrow \{0,1\}$











The RPC \mathcal{C} is N -access secure with advantage at most p if, for every x , for every N_1, N_2 such that $N_1 + N_2 = N$, and for every adversary \mathcal{A} ,

$$\text{Adv}_{\mathcal{C}, \mathcal{A}}^{\text{RPC}}(N_1, N_2) := 2 \cdot |\Pr [b' = b] - 1/2| \leq p.$$

Security Analysis: Ideal Cache Randomizer

As a first step, we assume an **ideal cache randomizer** ($\overline{\text{rekey}}, \bar{\pi}$) that behaves as a random oracle for functions from addresses to set indexes.

Ideal Cache Randomizer

For every $k \leftarrow \overline{\text{rekey}}()$, choose $\bar{\pi}_k$ uniformly at random.

Security Analysis: Ideal Cache Randomizer

As a first step, we assume an **ideal cache randomizer** ($\overline{\text{rekey}}, \bar{\pi}$) that behaves as a random oracle for functions from addresses to set indexes.

Ideal Cache Randomizer

For every $k \leftarrow \overline{\text{rekey}}()$, choose $\bar{\pi}_k$ uniformly at random.

We obtain:

Let $p \in [0, 1]$. Then \bar{C} is N -access secure with advantage at most p for

$$N = \max \left\{ N' : \sum_{i=0}^{N'-a} \binom{N'}{i} (1/|S|)^{N'-i} (1 - 1/|S|)^i \leq p \right\}.$$

Security Analysis: Ideal Cache Randomizer

As a first step, we assume an **ideal cache randomizer** ($\overline{\text{rekey}}, \bar{\pi}$) that behaves as a random oracle for functions from addresses to set indexes.

Ideal Cache Randomizer

For every $k \leftarrow \overline{\text{rekey}}()$, choose $\bar{\pi}_k$ uniformly at random.

We obtain:

Let $p \in [0, 1]$. Then \bar{C} is N -access secure with advantage at most p for

$$N = \max \left\{ N' : \sum_{i=0}^{N'-a} \binom{N'}{i} (1/|S|)^{N'-i} (1 - 1/|S|)^i \leq p \right\}.$$

This result has been slightly improved in a scenario with **noise**.

Application Example: Setting and Ideal Case

We consider the Intel[®] Core **i7-8700K (Coffee Lake)** 12MB LLC:

associativity a : 16

slices: 12

cache sets per slice: 1024

cache sets $|S|$: $12 \cdot 1024 = 12288$



Application Example: Setting and Ideal Case

We consider the Intel[®] Core **i7-8700K (Coffee Lake) 12MB LLC:**

associativity a : 16

slices: 12

cache sets per slice: 1024

cache sets $|S|$: $12 \cdot 1024 = 12288$



Suppose we want to thwart attacks with advantage **bigger than 1%**.

Application Example: Setting and Ideal Case

We consider the Intel[®] Core **i7-8700K (Coffee Lake)** 12MB LLC:

associativity a : 16

slices: 12

cache sets per slice: 1024

cache sets $|S|$: $12 \cdot 1024 = 12288$



Suppose we want to thwart attacks with advantage **bigger than 1%**.

Ideal Case:

The ideal RPC \bar{C} is N -access secure with advantage at most 0.01 for

$$N = \max \left\{ N' : \sum_{i=0}^{N'-a} \binom{N'}{i} (1/|S|)^{N'-i} (1 - 1/|S|)^i \leq 0.01 \right\}.$$

Application Example: Setting and Ideal Case

We consider the Intel[®] Core **i7-8700K (Coffee Lake)** 12MB LLC:

associativity a : 16

slices: 12

cache sets per slice: 1024

cache sets $|S|$: $12 \cdot 1024 = 12288$



Suppose we want to thwart attacks with advantage **bigger than 1%**.

Ideal Case:

The ideal RPC \bar{c} is 100532-access **secure** with advantage at most 0.01.

Table of Contents

1 Introduction to Cache Side-channels and RPCs

2 Our Model for RPCs

3 Security Definition and Analysis

4 Pseudo-random and Multi-epoch Cases

5 Performance Analysis

6 Conclusions

Security Analysis: PRF Cache Randomizer

Cache randomizers are **not ideal** in practice.

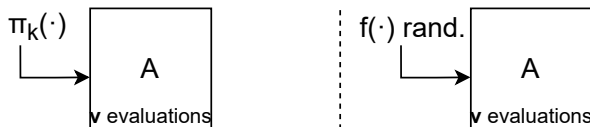
We use **pseudo-random** cache randomizers (π_k , rekey):



Security Analysis: PRF Cache Randomizer

Cache randomizers are **not ideal** in practice.

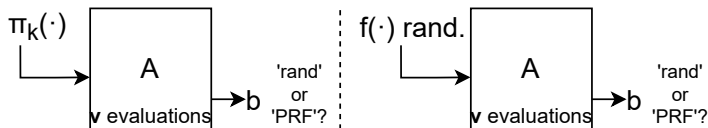
We use **pseudo-random** cache randomizers (π_k , rekey):



Security Analysis: PRF Cache Randomizer

Cache randomizers are **not ideal** in practice.

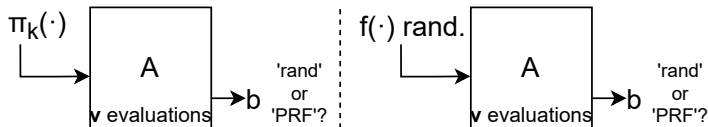
We use **pseudo-random** cache randomizers (π_k , rekey):



Security Analysis: PRF Cache Randomizer

Cache randomizers are **not ideal** in practice.

We use **pseudo-random** cache randomizers (π_k, rekey) :

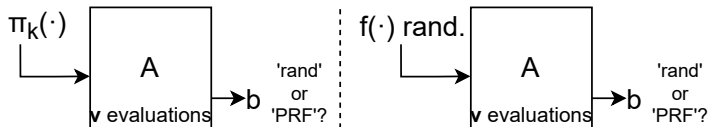


We say (π_k, rekey) is (ν, ε) -**pseudo-random** if every A has advantage at most ε in distinguishing the ν outputs of the oracle from random.

Security Analysis: PRF Cache Randomizer

Cache randomizers are **not ideal** in practice.

We use **pseudo-random** cache randomizers (π_k, rekey) :



We say (π_k, rekey) is (ν, ε) -**pseudo-random** if every A has advantage at most ε in distinguishing the ν outputs of the oracle from random.

Extend previous result to **PRF cache randomizers**: advantages add up.

Suppose that

- the ideal RPC \bar{C} is N -access secure with advantage at most p ,
- the cache randomizer is (N, ε) -**pseudo-random**.

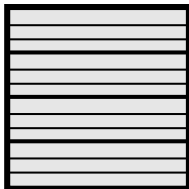
Then C is N -access secure with advantage at most $p + \varepsilon$.

Multi-epoch Security Definition

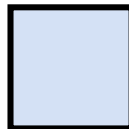
Attacker



RPC

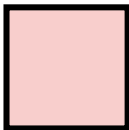


Victim

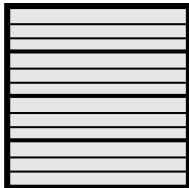


Multi-epoch Security Definition

Attacker

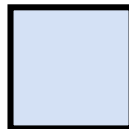


RPC



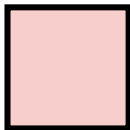
Victim

$b \leftarrow \{0,1\}$

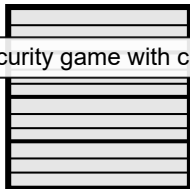


Multi-epoch Security Definition

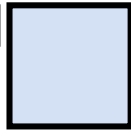
Attacker



RPC



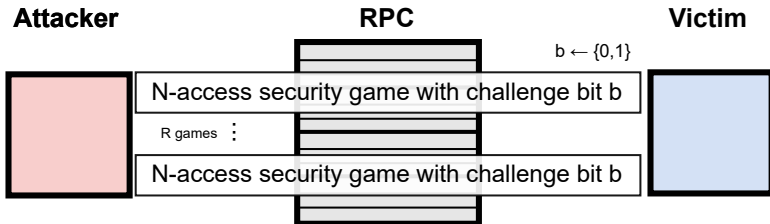
Victim



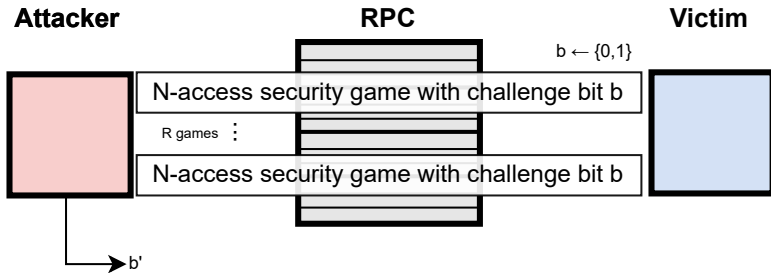
N-access security game with challenge bit b

$b \leftarrow \{0,1\}$

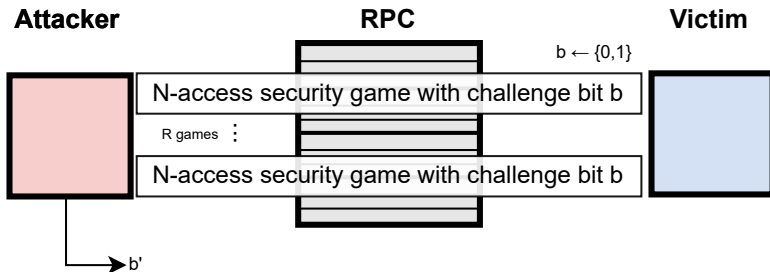
Multi-epoch Security Definition



Multi-epoch Security Definition



Multi-epoch Security Definition



The RPC \mathcal{C} is **R -Epoch N -Access Secure with advantage at most p** if, for all target addresses, every $N_{1,i} + N_{2,i} = N$, and every adversary \mathcal{A}

$$\text{Adv}_{\mathcal{C}, \mathcal{A}}^{\text{ME-RPC}}(R, N) := 2 |\Pr [b' = b] - 1/2| \leq p.$$

Multi-epoch Security Analysis

Following [Abdalla-Bellare'00], we **reduce multi-epoch security** to

- **single-epoch** security and
- the **pseudo-randomness** of the rekeying algorithm.

Multi-epoch Security Analysis

Following [Abdalla-Bellare'00], we **reduce multi-epoch security** to

- **single-epoch** security and
- the **pseudo-randomness** of the rekeying algorithm.

As before, all advantages add up.

Suppose

- \mathcal{C} is N -access secure with advantage at most p ,
- rekey is (R, ε) -pseudo-random.

Then \mathcal{C} is R -epoch N -access secure with advantage at most $R \cdot p + \varepsilon$.

Multi-epoch Security Analysis

Following [Abdalla-Bellare'00], we **reduce multi-epoch security** to

- **single-epoch** security and
- the **pseudo-randomness** of the rekeying algorithm.

As before, all advantages add up.

Suppose

- \mathcal{C} is N -access secure with advantage at most p ,
- rekey is (R, ε) -pseudo-random.

Then \mathcal{C} is R -epoch N -access secure with advantage at most $R \cdot p + \varepsilon$.

Rekeying **expands the time window** where security is **provably enforced**.

Application Example: PRF and Multi-epoch Cases

Suppose we want to thwart attacks with advantage **bigger than 1%**.

PRF case:

Assume the cache randomizer is $(100000, 0.001)$ -pseudo-random.

Application Example: PRF and Multi-epoch Cases

Suppose we want to thwart attacks with advantage **bigger than 1%**.

PRF case:

Assume the cache randomizer is $(100000, 0.001)$ -pseudo-random.

The ideal RPC \bar{C} is 99317-access secure with advantage at most 0.009.

The RPC C is 99317-**access secure** with advantage at most 0.01.

Application Example: PRF and Multi-epoch Cases

Suppose we want to thwart attacks with advantage **bigger than 1%**.

PRF case:

Assume the cache randomizer is $(100000, 0.001)$ -pseudo-random.
The ideal RPC \bar{C} is 99317-access secure with advantage at most 0.009.
The RPC C is 99317-**access secure** with advantage at most 0.01.

Multi-epoch case:

Assume the rekeying algorithm is $(10, 0.00001)$ -pseudo-random.

Application Example: PRF and Multi-epoch Cases

Suppose we want to thwart attacks with advantage **bigger than 1%**.

PRF case:

Assume the cache randomizer is $(100000, 0.001)$ -pseudo-random.
The ideal RPC \bar{C} is 99317-access secure with advantage at most 0.009.
The RPC C is 99317-**access secure** with advantage at most 0.01.

Multi-epoch case:

Assume the rekeying algorithm is $(10, 0.00001)$ -pseudo-random.
The RPC C is 9-**epoch**, 64033-**access** secure with advantage at most 0.01.

Security is **provably enforced for $RN = 576297$ accesses**

Table of Contents

1 Introduction to Cache Side-channels and RPCs

2 Our Model for RPCs

3 Security Definition and Analysis

4 Pseudo-random and Multi-epoch Cases

5 Performance Analysis

6 Conclusions

Performance Analysis

We use **ChampSim** to simulate the RPC in our running example, with

- cache randomizer: xor-based parametric randomizer [Trilla et al'18].
- L1 and L2 private caches: 8 ways, 64 and 1024 cache sets.
- replacement policy: PLRU.
- workload: **SPEC2006 benchmark suite**.

IPC for a randomized cache for different workloads and rekeying periods, normalized to a non-randomized setting

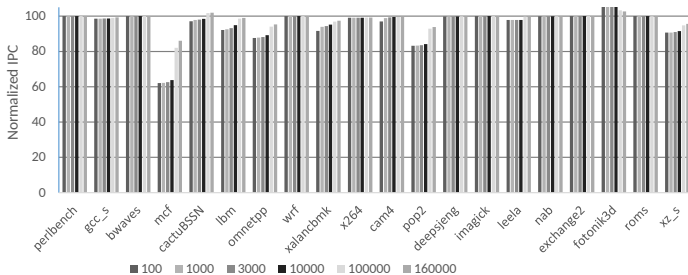


Table of Contents

1 Introduction to Cache Side-channels and RPCs

2 Our Model for RPCs

3 Security Definition and Analysis

4 Pseudo-random and Multi-epoch Cases

5 Performance Analysis

6 Conclusions

In this work, we introduce a **security model** for RPCs.

- We present game-based **security definitions**
- We show **how to design** RPCs to obtain **security guarantees**
- We provide a **performance evaluation**

Further research in this line

- improve security through **additional hardware techniques**
- **broaden the scope** of security definitions
- **tighten the bounds** for particular replacement policies

Thank you! Any questions?