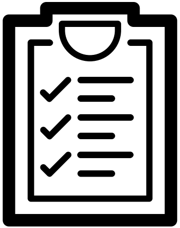
ScatterCache: Thwarting Cache Attacks via Cache Set Randomization

Werner, Unterluggauer, Giner, Schwarz, Gruss, Mangard

August 15, 2019

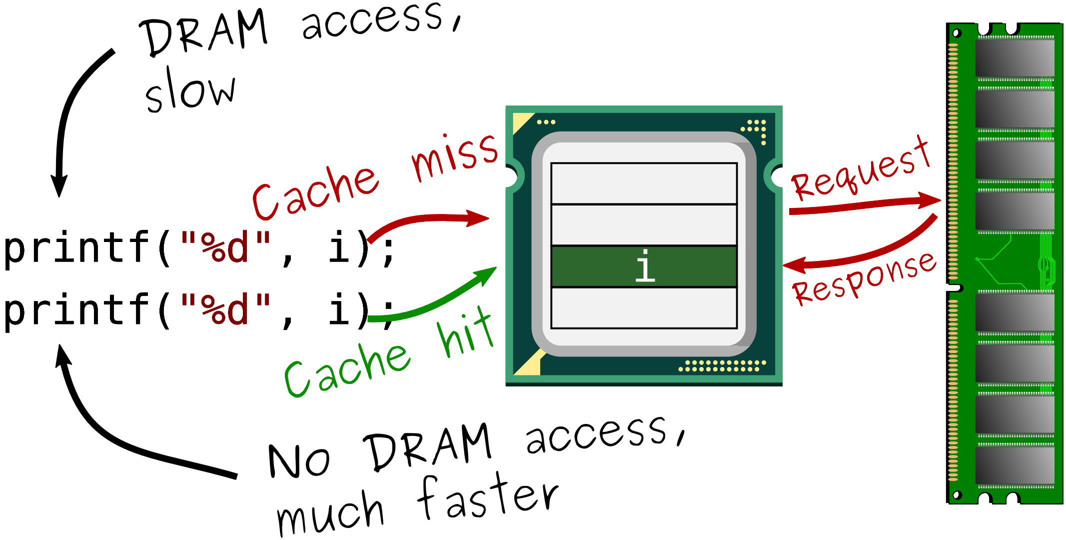
Graz University of Technology

What is ScatterCache?

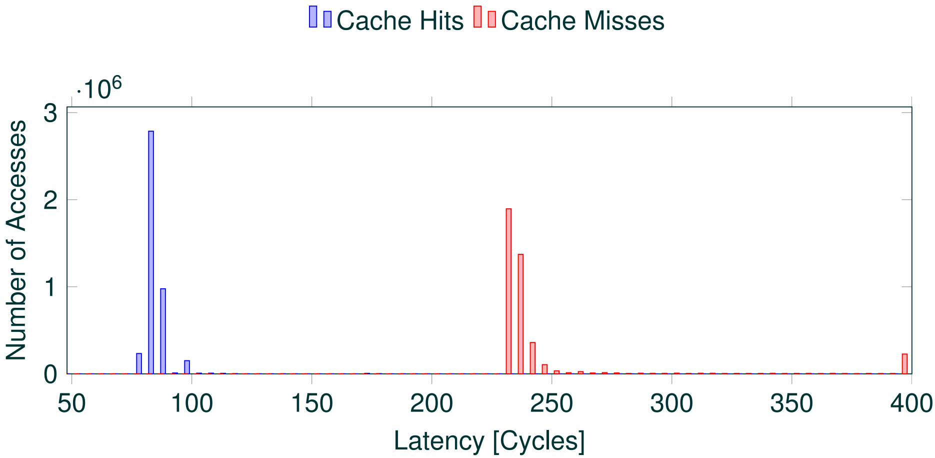


- Alternative design for n-way set associative caches
- Designed as countermeasures against cache attacks
  - Breaks the fixed link between addresses and cache sets
  - Increases the number of possible cache sets
  - IDs to change the mapping between security domains
  - → Exploitation of side channel information is much harder
- Reuses established concepts
  - Skewed caches [Sez93]
  - Low latency cryptography (e.g., QARMA-64 [Ava17])
- Still similar to existing cache designs (usability, hardware)

Motivation and Background

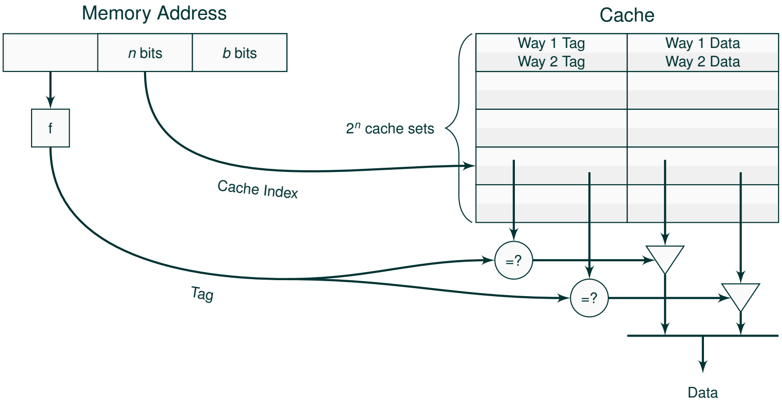


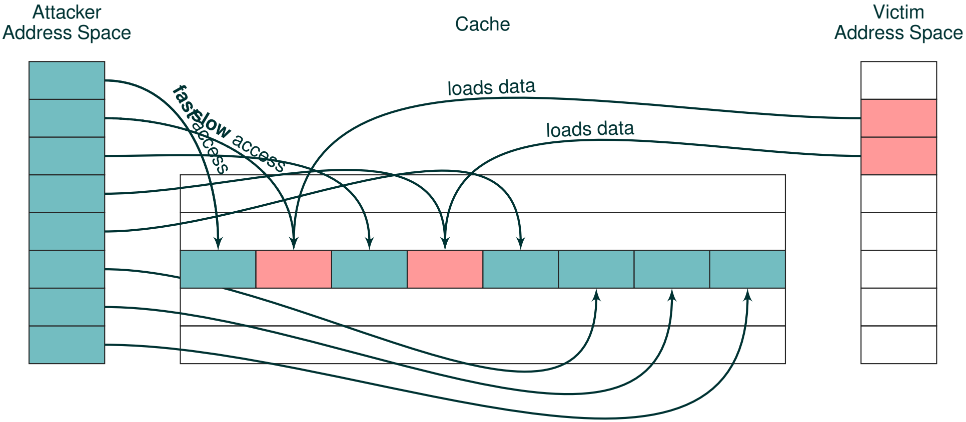
DRAM access,
slow
Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
i
Request
Response
No DRAM access,
much faster



generated using the CTA calibration tool [GSM15] on my i5-4200U laptop

Regular 2-way Set Associative Cache





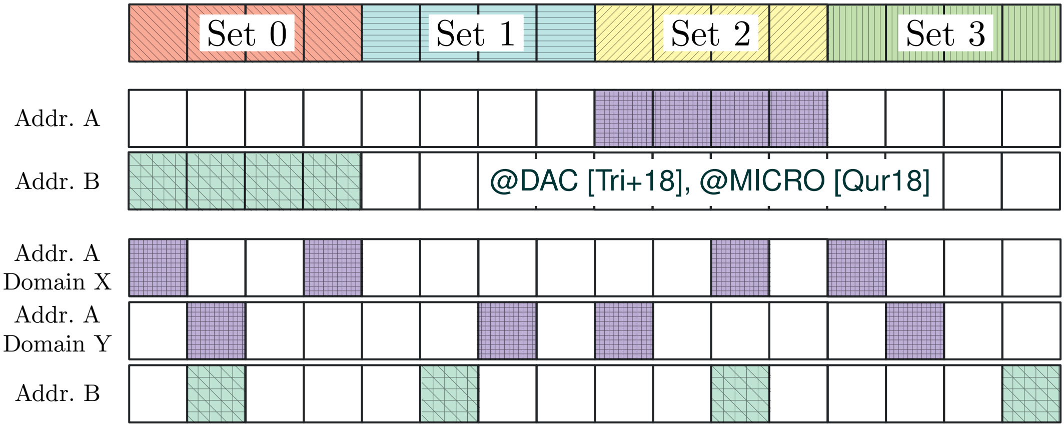
Attacker Address Space
Cache
Victim Address Space
loads data
loads data
fast access
slow access

Why should we care?

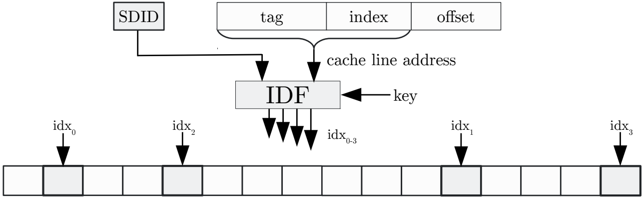
- Cache attacks are powerful and break isolation boundaries
- Many attacking techniques
  - Flush+Reload, Evict+Reload, Flush+Flush
  - Prime+Probe, Evict+Time
- Numerous attack scenarios
  - Extracting cryptographic keys
  - Keyloggers
  - Breaking of ASLR
  - Collection of private information
- Often used building block for further microarchitectural attacks

ScatterCache

SCATTERCACHE - Idea



How can we build such a SCATTERCACHE?



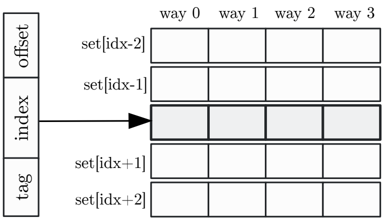
$\binom{n_{ways} \cdot 2^{b_{indices}} + n_{ways} - 1}{n_{ways}}$ possible cache sets

512 KiB (32 B lines), $n_{ways} = 8$, $b_{indices} = 11$
$\rightarrow 2^{96.7}$ sets

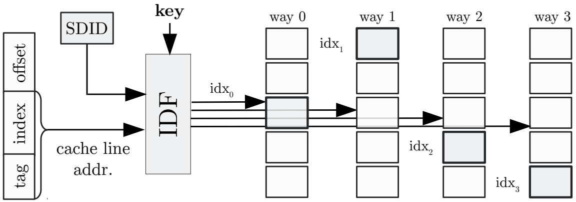
- Index Derivation Function (IDF) takes an address and returns a cache set
- Depends on hardware key and optional Security Domain ID (SDID)
- $\rightarrow$ Unique combination of cache lines for each address
  - Potential index collisions
  - One n_{ways} multi-port memory

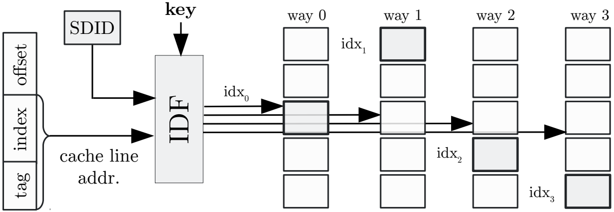
We want something that is closer to a traditional cache!

instead of this:



let's do this:



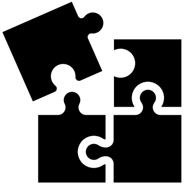


$2^{b_{indices} \cdot n_{ways}}$ possible cache sets

512 KiB (32 B lines), $n_{ways} = 8$, $b_{indices} = 11$
$\rightarrow 2^{88}$ sets

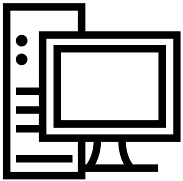
- Skewed cache [Sez93] (*i.e.*, traditional cache with additional addressing logic) and an IDF
- Similar to building larger caches from smaller cache slices
- We use random replacement policy (for now)

ScatterCache - Selecting the IDF



- Inputs: cache line address, SDID, key
- Outputs: n_{ways} indices with $b_{indices}$ bits
- Reuse concepts and existing cryptographic primitives
- SCv1: hashing variant
  - Block ciphers (e.g., PRINCE [Bor+12])
  - Tweakable block ciphers (e.g., QARMA [Ava17])
  - Permutation-based primitives (e.g., Keccak-*p* [Ber+11])
- SCv2: permutation variant
  - Prevents birthday-bound index collisions
  - No off-the-shelf primitives

System Integration



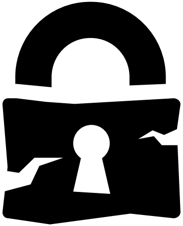
- SCATTERCACHE as last level cache
- Hardware managed key
  - Randomly generated at boot time
  - Rekeying with full cache flush
  - Potential for iterative rekeying
    → concurrently developed CEASER-S @ISCA [Qur19]
- SDID management via page table (indirection)
  - x86: Page Attribute Tables (PATs)
  - ARM: Memory Attribute Indirection Register (MAIRs)

- SCATTERCACHE requires no software support, default SDID = 0
- But - OS support enables page-wise security domains
  → shared read-only pages can be private in the cache!
- OS can define domains as needed
  (pages, processes, containers, VMs, . . .)
- Software-based page “rekeying” by changing the SDID

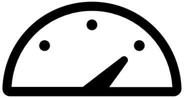
Security and Evaluation

Applicable Cache Attacks

- **Unshared memory** has no shared (physical) addresses
  - → No Flush+Reload, Evict+Reload, Flush+Flush
  - → Specialized Prime+Probe is possible
- **Shared, read-only memory**
  - → Like unshared memory given OS support
  - → Otherwise, eviction-based attacks are hindered
- **Shared, writable memory** can't be separated
  - → Eviction-based attacks are hindered

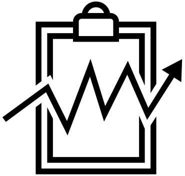


- No end-to-end attack yet
  - → Simplified setting: perfect control, single access, no noise
  - → Investigate the building blocks in simulation and analytically
- Finding congruent addresses ($n_{ways} = 8, b_{indices} = 11$)
  - Full collisions are unlikely → use partial collisions
  - Approach in the paper: $\approx 2^{25}$ profiled victim accesses
  - Generalized by Purnal and Verbauwhede [PV19]: $\approx 2^{10}$
- Evicting one set with 99 % needs 275 addresses
- Two PRIME+PROBE variants ($n_{ways} = 8, b_{indices} = 12$)
  - 99 % confidence: 35 to 152 victim accesses (repetitions)
  - Between 9870 and 1216 congruent addresses
- Investigate the effect of noise (coupon collector problem)



- Micro benchmarks using the gem5 full system simulator (ARM)
  - Poky Linux from Yocto 2.5 (kernel version 4.14.67)
  - GAP, MiBench, lmbench, scimark2
- SPEC CPU 2017 on custom cache simulator
- Cache hit rate always at or above levels of set-associative cache with random replacement
- Typically 2 % – 4 % below LRU on micro benchmarks, 0 % – 2 % for SPEC

Conclusion



- SCATTERCACHE builds upon skewed caches and low latency cryptographic primitives
  - Breaks the fixed link between addresses and cache sets
  - Removes the rigid assignment of cache lines to sets
  - Enables software control over the cache congruencies via SDIDs
- Comparable performance to contemporary caches
- Harder to attack even in very strong attack models
- Attacks are probabilistic and demand new approaches
- Still, more analysis is required in more realistic models to determine if and how often rekeying is needed

Acknowledgements - We want to thank ...

- the anonymous USENIX reviewers.
- our shepherd Yossi Oren.
- Antoon Purnal and Ingrid Verbauwhede from KU Leuven for their analysis.
- Our funding partners:
  - European Research Council (ERC)
    Horizon 2020 grant agreement No 681402
  - Intel

ScatterCache: Thwarting Cache Attacks via Cache Set Randomization

Werner, Unterluggauer, Giner, Schwarz, Gruss, Mangard

August 15, 2019

Graz University of Technology

References

[Ava17] Roberto Avanzi. “The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes”. In: *IACR Trans. Symmetric Cryptol.* (2017), pp. 4–44. DOI: 10.13154/tosc.v2017.i1.4-44.

[Ber+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *The KECCAK reference*. https://keccak.team/files/Keccak-reference-3.0.pdf. 2011.

[Bor+12] Julia Borghoff et al. “PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract”. In: *Advances in Cryptology – ASIACRYPT*. 2012, pp. 208–225. DOI: 10.1007/978-3-642-34961-4_14.

[GSM15] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. *Cache Template Attacks Repository*. https://github.com/IAIK/cache_template_attacks. 2015.

[PV19] Antoon Purnal and Ingrid Verbauwhede. “Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache”. In: *arXiv abs/1508.03619* (2019). URL: `http://arxiv.org/abs/1908.03383`.

[Qur18] Moinuddin K. Qureshi. “CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping”. In: *IEEE/ACM International Symposium on Microarchitecture – MICRO*. 2018, pp. 775–787. DOI: `10.1109/MICRO.2018.00068`.

[Qur19] Moinuddin K. Qureshi. “New attacks and defense for encrypted-address cache”. In: *International Symposium on Computer Architecture – ISCA*. 2019, pp. 360–371. DOI: `10.1145/3307650.3322246`.

[Sez93] André Seznec. “A Case for Two-Way Skewed-Associative Caches”. In: *International Symposium on Computer Architecture – ISCA*. 1993, pp. 169–178. DOI: `10.1145/165123.165152`.

[Tri+18] David Trilla, Carles Hernández, Jaume Abella, and Francisco J. Cazorla. “Cache side-channel attacks and time-predictability in high-performance critical real-time systems”. In: *Design Automation Conference – DAC*. 2018, 98:1–98:6. DOI: `10.1145/3195970.3196003`.