HybCache:
Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments

Ghada Dessouky, Tommaso Frassetto, Ahmad-Reza Sadeghi
Systems Security Lab, Technische Universität Darmstadt, Germany
Architecture is Only the Tip of the Iceberg

The myth of the hardware-based trust anchor

Software

Architecture

Dedicated focus on software security

ISA
registers
main memory
Architecture is Only the Tip of the Iceberg
The Performance – Security Trade-Off

- shared caches
- branch prediction units
- speculative execution
- shared buffers contention
- microarchitecture
- store-to-load/load-to-load forwarding
- transient execution
HybCache: The Big Picture

- On-demand & configurable cache-based side-channel resilience \(\rightarrow\) tunable performance/security trade-off knob

- Hybrid set-associative cache:
  - Non-critical workload: behaves typically and utilizes full cache capacity
  - Security-critical workload: utilizes only a cache subset fully-associatively with random eviction

Incur performance costs for side-channel resilience only when the additional security guarantees are required
How are Shared Caches Accessed?

Address:
- remaining bits – tag
- $\log_2 S$ bits – set index
- $\log_2 B$ bits – block offset

$S$ sets

$W$ ways

B-byte cache line

Access the required byte block from the cache line using block offset bits

Tag match found
How are Shared Caches Exploited?

Because of the inherent principles of caches operation and their performance advantage

CPU (+cache) speed > DRAM access latency

- Cache miss: request data from DRAM upon first access → much slower
- Cache hit: No DRAM access required for subsequent accesses → much faster

inherent timing channel
Our Insights & Requirements for HybCache

Insights
• Majority of the execution workload is not security-critical
• Security-critical code is already isolated, e.g., in a TEE
• Root causes for conflict/contention & access-based cache attacks are set-associative eviction & shared cache lines
• Only approach to complete non-interference is strict cache partitioning → impractical

Requirements
• Strong side-channel resilience guarantees between security-critical and non-critical execution
• Side-channel resilience and dynamic utilization among mutually distrusting critical workloads
• Selective/configurable cache side-channel-resilience
• Usability and backwards compatibility
System Assumptions

• Security-critical code (requiring side-channel-resilient cache utilization) is in an isolated component or domain (I-Domains), e.g., a process or a TEE (enclave).

• Mutually distrusting code is allocated to different I-Domains.

• Isolated execution (in I-Domains) is the minority of the execution workload. Rest of the workload is non-isolated (NI-Domain).

• The attacker is not in the same I-Domain as the victim.
High-Level Architecture

Core 0
- L1 Data Cache
- L1 Instr Cache
- L2 Unified Cache

Core N
- L1 Data Cache
- L1 Instr Cache
- L2 Unified Cache

L3 Shared Cache
High-Level Architecture

Core 0
- L1 Data Cache
- L1 Instr Cache
- L2 Unified Cache

Core N
- L1 Data Cache
- L1 Instr Cache
- L2 Unified Cache

L3 Shared Cache

Orthogonal subcache of fully-associative cache ways

non-isolated cache access request from NI-Domain
full cache capacity including subcache ways
High-Level Architecture

Core 0
- L1 Data Cache
- L1 Instr Cache
- L2 Unified Cache

Core N
- L1 Data Cache
- L1 Instr Cache
- L2 Unified Cache

L3 Shared Cache

Orthogonal subcache of fully-associative cache ways

isolated cache access request (from any I-Domain):
restricted to only use subcache ways
Controller Flow

incoming cache request

isolated or non-isolated mode?

isolated (ID!=0)

Query the subcache ways fully-associatively using extended tag bits

yes

no

cache miss

no

yes

way with matching tag found?

no

cache miss

line ID non-isolated (0) if the way is in subcache?

no

cache miss

yes

cache hit

cache miss

cached line occupied with line of matching ID?

no

cache miss

yes

cache hit

Query cache set-associatively using set index and tag bits to locate the way

Query the subcache ways fully-associatively using extended tag bits

no

yes

way with matching tag found?

no

cache miss

way with matching tag found?

no

cache miss

yes

cache hit

cached line occupied with line of matching ID?

no

cache miss

yes

cache hit

cached line occupied with line of matching ID?

no

cache miss

Evict and replace (via LRU/pseudo-LRU policy) cache line (including these occupying subcache ways)

non-isolated (ID=0)

cache miss

no

way with matching tag found?

yes

cache hit

cache miss

Randomly evict and replace any of the cache lines occupying the subcache ways (irrespective of their line-IDID)
Tag-Store Extensions

Incoming cache access request

<table>
<thead>
<tr>
<th>cache ways</th>
<th>line-IDID</th>
<th>extended tag bits</th>
<th>subcache ways</th>
<th>set 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cache ways</th>
<th>line-IDID</th>
<th>extended tag bits</th>
<th>subcache ways</th>
<th>set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tag-Store Extensions

Incoming cache access request

<table>
<thead>
<tr>
<th>Tag Store</th>
<th>Cache Line Store</th>
</tr>
</thead>
</table>

Hardware design decisions/trade-off:
- Size of *subcache*
- Maximum number of concurrent isolation domains that can be supported
Security Analysis

Direct access to cache lines of another I-Domain not possible
- Shared cache lines are disallowed between different domains by design: cannot flush/hit on cache lines of another domain

Pre-computing and constructing an eviction set not possible
- Disabling the set-associative eviction of the subcache and random replacement → no reproducible and consistent mapping of a given memory access to a cache entry

Observing cache hits and misses of another I-Domain
- Cache hits not observable by another domain
- Attacker can still infer size of victim’s working set by observing its own line evictions → cache occupancy channel remains by design
Evaluation
### Performance: Two Processes

<table>
<thead>
<tr>
<th>Core</th>
<th>Process A</th>
<th>Process B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L1 Cache</strong> – 64 KB</td>
<td>8-way associative</td>
<td></td>
</tr>
<tr>
<td><strong>L2 Cache</strong> – 256 KB</td>
<td>8-way associative</td>
<td></td>
</tr>
<tr>
<td><strong>L3 Cache</strong> – 4 MB</td>
<td>16-way associative</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mix</th>
<th>SPEC Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>pov+mcf</td>
<td>povray, mcf</td>
</tr>
<tr>
<td>lib+sje</td>
<td>libquantum, sjeng</td>
</tr>
<tr>
<td>gob+mcf</td>
<td>gobmk, mcf</td>
</tr>
<tr>
<td>ast+pov</td>
<td>astar, povray</td>
</tr>
<tr>
<td>h26+gob</td>
<td>h264ref, gobmk</td>
</tr>
<tr>
<td>bzi+sje</td>
<td>bzip2, sjeng</td>
</tr>
<tr>
<td>h26+per</td>
<td>h264ref, perlbench</td>
</tr>
<tr>
<td>cal+gob</td>
<td>calculix, gobmk</td>
</tr>
</tbody>
</table>
Performance Overhead of Isolated Processes
Overhead of Non-Isolated Processes
Performance: Four Processes

Core 0
- Process A
- Process B
- L1 Cache – 64 KB
  8-way associative
- L2 Cache – 256 KB
  8-way associative
- L3 Cache – 4 MB
  16-way associative

Core 1
- Process C
- Process D
- L1 Cache – 64 KB
  8-way associative
- L2 Cache – 256 KB
  8-way associative
Deployment and Future Work

• Generic concept of “soft” partitioning:
  • which structures and caches to apply it to and the subcache size is a hardware design decision/trade-off

• CAM tag-store lookups are expensive (power consumption and timing/routing)
  • emerging memory technologies such as DRAM-based and STT-MRAM caches to alleviate the overheads

• Ongoing work: improved design to achieve the same full-associativity (+ stronger isolation) without expensive lookups
Thank you!

Ghada Dessouky
ghada.dessouky@trust.tu-darmstadt.de