Scalable Memory Protection in the PENGLAI Enclave

Erhu Feng, Xu Lu, Dong du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, Haibo Chen
(fengerhu1@sjtu.edu.cn)

Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
Shanghai AI Laboratory
Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China
Enclave / TEE (Trusted Execution Environment)

1. TEE protects enclaves from untrusted software
   - Hypervisor
   - Operating system
   - Other applications

2. TEE contains secure hardware resources
   - Secure CPU cores
   - Trusted on-chip modules
     - Memory controller, etc.
Existing Enclaves and Usages in Cloud

- Intel SGX
- AMD SEV
- ARM TrustZone
- Keystone, Penglai

- Enclaves are widely-used in various cloud scenarios

- Secure ML
  - Protect AI model
  - Protect input privacy
  - Protect output result

- Secure Service
  - Transaction
  - Confidential computing
  - Key/CA management

- Secure DB
  - Protect user private data in the database
Restrictions of Current Enclaves

**Restricted enclave size**
- Intel SGX (with hardware integrity protection) only has 128/256M secure memory
- Secure memory regions used in Keystone / TrustZone are hard to expand

**Restricted enclave number**
- Enclave number of keystone is restricted by the protected memory regions
- Enclave numbers of AMD SEV/Intel TDX are restricted by the encryption keys

**Restricted startup latency**
- Creating an enclave (64M) in SGX needs about 0.6s
- Creating runtime and enclave (64M) in the keystone needs 60 million cycles

**Root cause:** Non-scalable secure memory protection mechanisms
Non-scalable Secure Memory Protection

1. Non-scalable secure memory isolation
2. Non-scalable secure memory integrity protection
3. Non-scalable secure memory initialization
## Scalable Memory Protection: Overview

<table>
<thead>
<tr>
<th>Problems</th>
<th>Solutions</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-scalable Memory isolation?</td>
<td>Guarded Page Table</td>
<td>Fine-grained isolation</td>
</tr>
<tr>
<td>Non-scalable Memory Integrity protection?</td>
<td>Mountable Merkle Tree</td>
<td>Large-scale memory integrity protection</td>
</tr>
<tr>
<td>Non-scalable Memory initialization?</td>
<td>Shadow Fork</td>
<td>Fast memory initialization</td>
</tr>
</tbody>
</table>
1. Non-scalable Memory Isolation

- **Region-based memory isolation**
  - Protected memory can only reside in few contiguous ranges
    - SGX uses a *fixed-size* processor reserved memory range
    - TrustZone configures *restricted* secure memory regions
    - Keystone leverages *limited* PMP registers to isolate physical memory regions

- **Page-based memory isolation (our choice)**
  - Consistent with the fine-grained memory management
  - Easy to scale, no need of GC for normal memory
Fine-grained Flexible Memory Isolation

- **Approach**
  - Using bitmap to record each page’s ownership (like memory tagging)

- **Challenge**
  - **Double-access issue**: check the page ownership during memory access
  - Incur non-trivial overhead (e.g., 25% on average for TIMBER-V[1])

- **Observation**
  - Memory mapping is far less frequent than memory access
  - **Check during mapping, not accessing**

---

[1]: *Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v*
Guarded Page Table (GPT)

• We restrict the host page table in a specific memory region, named HPT_Area
  – HPT_Area: a monitor-write-only region, enforced by the hardware (e.g., PMP)

• 1. An extended MMU checks during page table walking to ensure:
  – All the page table pages must reside in the HPT_Area
  – So that only the monitor can update the page tables

• 2. The monitor checks mapping according to the ownership bitmap to ensure:
  – None of the secure page will be mapped in the host page tables

Guarded page table maintains all non-secure pages
2. Non-scalable Integrity Protection

- **Restrictions of non-scalable integrity tree (Merkle Tree)**
  - Store topmost tree nodes in SoC ➞ SoC storage is restricted
  - Pre-allocate all intermediate nodes ➞ Fixed memory storage overhead
  - Protect contiguous physical memory ➞ Violate with fine-grained memory isolation
Scale the Traditional Merkle Tree

- Enlarge the SoC storage

Protected physical memory

- Protected memory: 128MB
- Protected memory: 512GB

Node

- SoC: >512B
- SoC: >2MB
- SoC: >2MB

Protected physical memory

- About 32MB
- About 128GB
Mountable Merkle Tree (MMT)

• Divide one tree into several SubTrees
  – Three states for Subtrees
    1. Mounted (In-SoC)
    2. Allocated (In-memory)
    3. Unallocated

![Diagram showing division of a tree into sub-trees with states: Mounted, Allocated, Unallocated, Protected memory, Hot secure, Non-secure, Secure.]
Mountable Merkle Tree (MMT)

- Unallocated state

![Diagram of MMT with nodes and memory zones]

- In-SoC node
- In-memory node
- Unallocated node
- Non-secure memory
- Secure memory
- Tree node memory
- Hot tree node memory

Protected memory: 512GB

MMT meta-zone: 2MB
Mountable Merkle Tree (MMT)

- Allocated state

Protected memory: 512GB

MMT meta-zone: 2MB

- In-SoC node
- In-memory node
- Unallocated node
- Non-secure memory
- Secure memory
- Tree node memory
- Hot tree node memory
Mountable Merkle Tree (MMT)

- Mounted state

Protected memory: 512GB

SoC: 512B

In-SoC node
In-memory node
Unallocated node
Non-secure memory
Secure memory
Tree node memory
Hot tree node memory

Mounted SubTree

Hot secure Non-secure Hot secure Secure

Mounted SubTree

MMT meta-zone: 2MB
3. Non-scalable Memory Initialization

- Initialize enclave memory with its measurement
  - Calculate hash of each secure page
    - Secure hardware/firmware calculate the hash sequentially
    - Used as the measurement in the attestation report
    - Takes up over 90% time during enclave startup
Boost Secure Memory Initialization

- **Introducing** **shadow fork** to reduce hash calculating
  - *Shadow enclave*: a clean template contains code and data of enclaves
    - Code and ro-data are initialized by *shadow fork* with *shadow enclave*
  - Initializing memory without hash calculating
    - Only copy writable parts from shadow enclave to the new enclave
    - Assign zero-filled memory as stack and heap
Evaluation

• **Hardware platforms**
  – SiFive Freedom U500 + Xilinx VC707 FPGA board
  – QEMU
  – Gem5

• **Software environment**
  – OpenSBI
    • Add 6,399 LoC
  – Linux-5.10.2
Guarded Page Table (GPT)

- **GPT shifts the overhead from memory access to mapping**
  - No overhead for memory access
  - For both latency and bandwidth

- **Evaluate the mmap() overhead**
  - 45% overhead for latency
  - No overhead for bandwidth
Guarded Page Table

- Real-world memory-intensive workload
  - Choose the Redis benchmark
    - Max 5% overhead for SET/GET
Mountable Merkle Tree

• Evaluate MMT on SPECCPU benchmark
  – If used memory size is larger than SoC protected size
    • SGX uses paging mechanism
      – 0.46x ~ 7.5x
    • MMT uses mounting mechanism
      – 0.15x ~ 0.64x
Enclave Startup Latency

• **Shadow fork reduces hash calculating during startup**
  - Micro-benchmark (startup latency)
    • 4~989x faster than normal enclave creation
  - End-to-end latency (MapReduce)
    • 2~3.6x faster when using shadow fork to initialize MapReduce workers
Conclusion

- Achieving scalable memory protection in PENGLAI enclave
  - Fine-grained and flexible memory isolation
    - Using guarded page table to protect the page mapping
  - Scalable memory integrity protection
    - Using mountable Merkle Tree to scale the secure memory
  - Fast secure memory initialization
    - Using shadow fork to reduce hash calculation during enclave startup

Code available: https://github.com/Penglai-Enclave/

Thanks!