Secure Processors

Typical Computer Trusted Computing Base (TCB)

Software

Hardware

100's of millions of LOC, billions of logic gates!
Single-Chip Secure Processor

- Shrink software TCB to thousands of LOC.
- Still trust the hardware processor.
Untrusted Memory

Confidentiality: Encryption

Memory latency increase hurts performance (most programs are memory bottle necked!)
Direct-Block Encryption

Cipher Block Chaining (CBC) mode

Cache Block

Diagram showing the process of encryption and decryption using AES and CBC mode.
One-Time-Pad Encryption

One-time-pad (OTP)

Encryption

Decryption

XOR'ing is fast!

How to store OTP[i]'s?

- can't store in untrusted DRAM!
Counter-Mode Encryption

Generate OTPs from timestamps with AES! Store timestamps with encrypted blocks in memory.

L2- Cache-Writeback:
1. $TS = TS + 1$
2. a. $OTP = AES_k(Addr, TS)$
   b. $EB = B \oplus OTP$
3. Write $TS, EB$ to memory

L2- Cache-Miss:
1. Read $TS$ from memory.
2. in parallel:
   - $OTP = AES_k(Addr, TS)$
   - Read $EB$ from Addr in Memory
3. $B = EB \oplus OTP$
Active Attacker / Integrity

Just use a MAC:

<table>
<thead>
<tr>
<th>addr1</th>
<th>addr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>EB</td>
<td>MAC(EB, addr1)</td>
</tr>
</tbody>
</table>

Problem: Replay attacks

Suppose program writes:

- EB₁, MAC(EB₁, addr₁)
- EB₂, MAC(EB₂, addr₁)

Adversary ignores this write. Return this value on next read ⇒ no failure/detection.
Merkle/Integrity Tree

$h_1 = h(V_1, V_2)$

$h_2 = h(V_3, V_4)$

$\text{root} = h(h_1, h_2)$

Store root in processor secure memory cannot be tampered with. root checked on reads, updated on writes.
Tree Operations: Read

Read \( V_3 \)
- Read \( V_4 \)
- Read \( h_2 \)
- Compute \( h_2' = h(V_3, V_4) \)
- Check \( h_2' = h_2 \)
- Read \( h_1 \)
- Compute \( \text{root}' = h(h_1, h_2') \)
- Check \( \text{root}' = \text{processor root} \)

Collision-resistant \( h() \) provides security. Adversary can't modify \( V_3, V_4, h_2, h_1 \) to match \( \text{root}' \).
Tree Operations: Write

Write $v_2$
- read $v_1$
- compute $h(v_1, v_2') = h_1'$
- write $h_1'$
- read $h_2$
- compute $h(h_1', h_2) = \text{root}'$
- update processor root with root'

Caching intermediate hashes in trusted processor cache improves performance significantly.
Intel SGX
- Memory encryption & integrity verif.

BIOS
Hypervisor
OS Kernel
App SGX enclave

← Trusted + Microcode ROM

DRAM
Processor Reserved Memory

PRM
Enclave Page Cache

EPC
4KB page
4KB page

CPU ensures each EPC page belongs to exactly one enclave.
SGX Leaks

- Untrusted OS/app can attack enclave via (shared) cache timing attacks.

- Address Translation Leak

Untrusted OS manages its and enclaves' page tables!
Malicious Hypervisor sets P (present) flag to 0 on all page table entries, lets enclave execute.

First memory access causes a page fault. Hypervisor maps faulty page & resumes enclave execution. enzyme page address

Next page fault, hypervisor maps in new page, and unmaps previous page, so it can see enclave's memory access pattern at page granularity (minus offset)

Instruction execution $\rightarrow$ pages mapping $\rightarrow$ gives control/secret data.
Sanctum Design

Partitioned Cache

Address

<table>
<thead>
<tr>
<th>Tag</th>
<th>Set Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 0, Way 0</td>
<td>Set 0, Way 1</td>
<td></td>
</tr>
<tr>
<td>Set 1, Way 0</td>
<td>Set 1, Way 1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Set $S-1$, Way 0</td>
<td>Set $S-1$, Way 1</td>
<td></td>
</tr>
</tbody>
</table>

Enclave

OS

DRAM

OS memory

Enclave mem
Sanctum Design

Page table isolation

Enclave A
Virtual Addr Space

Enclave B
Virtual Addr Space

Enclave A page tables

Enclave B page tables

OS Region

OS page tables

Host application space

Host app space

Enclave page tables (PTs) inside enclave memory (isolated from OS) - need some hardware to multiplex PTs
Enclave Life cycle (simplified)

1. Create enclave grant resources
2. Load enclave
3. Seal enclave
4. Security monitor
5. Exit enclave

- Enclave executes isolated memory, page tables, cache, etc.
- Enclave binary image