HyperEnclave: An Open and Cross-platform Trusted Execution Environment

Yuekai Jia\textsuperscript{1}, Shuang Liu\textsuperscript{2}, Wenhao Wang\textsuperscript{3,4}, Yu Chen\textsuperscript{1}, Zhengde Zhai\textsuperscript{2}, Shoumeng Yan\textsuperscript{2}, Zhengyu He\textsuperscript{2}

\textsuperscript{1}Tsinghua University
\textsuperscript{2}Ant Group
\textsuperscript{3}SKLOIS, Institute of Information Engineering, CAS
\textsuperscript{4}School of Cyber Security, University of Chinese Academy of Sciences

USENIX ATC '22, July 11-13, 2022
Background

- **Trusted execution environments (TEEs):**
  - Protected sensitive data in hardware-enforced isolated environments (enclaves)
  - Thwart OS-level and physical attackers
Motivation

• Most of today’s TEEs:
  • Require **specific hardware** changes: slow to evolve
  • **Close-sourced**: difficult to audit
  • Enclaves run in **fixed mode**: inflexible to adapt to diverse workloads
    (e.g., applications that want to access privileged resources)
Design Goals

• **G1: Minimum hardware requirements**
  • Hardware virtualization + TPM
  • Cross-platform

• **G2: Easy to develop**
  • SGX compatible
    ‣ Port existing SGX programs with little or no code changes

• **G3: Flexible enclave modes**
  • Better fulfill the needs for specific enclave workloads
Threat Model

• System is initially benign
  ✔ Can be compromised after system boot (cloud scenario)

• Malicious software
  ✔ Application, OS kernel, enclave malware

• Certain physical memory attacks
  ✔ cold boot attacks, bus snooping attacks

• Certain side channel attacks
  ✔ page-table-based attacks

• Out of scope
  ✘ DoS, other side channel attacks (cache timing, speculative execution)
HyperEnclave Overview

Hardware
- CPU
- RAM
- EPC
- Device
- APIC
- Disk
- NIC
- TPM

Enclave-A
- SDK irtS
- Enclave Management

Enclave-B
- SDK irtS
- Enclave Management

Enclave-A
- SDK irtS
- Enclave Management

Enclave-B
- SDK irtS
- Enclave Management

RustMonitor
- Memory Management
- Hypercall/Syscall Handler
- Initialization & Demotion

Normal VM
- Primary OS
- Kernel Module
- App-B
  - SDK urTS
- App-A
  - SDK urTS

Monitor Mode
- Normal Mode
- Secure Mode

Secure Mode
- VMX root, ring-0

Normal Mode
- VMX root, ring-0

Monitor Mode
- VMX root, ring-0
HyperEnclave Overview

- **Hardware**
  - CPU
  - Device
  - RAM
  - EPC
  - APIC
  - Disk
  - NIC
  - TPM

- **Enclave Management**
  - Initialization & Demotion
  - Memory Management
  - Hypercall/Syscall Handler
  - RustMonitor

- **Modes**
  - Monitor Mode
  - Normal Mode
  - Secure Mode

- **VMX non-root, ring-0 & ring-3**
HyperEnclave Overview

- **Hardware**: CPU, Device, RAM, EPC, APIC, Disk, NIC, TPM
- **CPU**: Hardware component
- **Device**: Hardware component
- **RAM**: Memory Management
- **EPC**: Memory Management
- **APIC**: Memory Management
- **Disk**: Memory Management
- **NIC**: Memory Management
- **TPM**: Memory Management

**Hypercall/Syscall Handler**

- **vCPU**: Initialization & Demotion
- **Normal VM**: Primary OS
- **Primary OS**: Kernel Module

**Rust Monitor**

- **Enclave-A**: SDK iRTS
- **Enclave-B**: SDK iRTS

**Secure Mode**: VMX non-root, ring-3 (GU-Enclave)
- **Normal Mode**: VMX non-root, ring-0 (P-Enclave)
- **Monitor Mode**: VMX root, ring-3 (HU-Enclave)
HyperEnclave Overview

Hardware

- CPU
- RAM
- EPC
- Device
- APIC
- Disk
- NIC
- TPM

CPU

Device

RAM

EPC

APIC

Disk

NIC

TPM

vCPU

Memory Management

Hypercall/Syscall Handler

Enclave Management

RustMonitor

Initialization & Demotion

ENCLS

ECALL

Normal VM

Normal Mode

Monitor Mode

Secure Mode

Primary OS

Kernel Module

App-A

SDK uRTS

App-B

SDK uRTS

Enclave-A

SDK iRTS

Enclave-B

SDK iRTS

ENCLU

VMX non-root, ring-3 (GU-Enclave)

VMX non-root, ring-0 (P-Enclave)

VMX root, ring-3 (HU-Enclave)

flexible!
Design Details

1. Memory management and protection
2. Flexible enclave operation mode
3. Trusted Boot
4. The enclave SDK
1. Memory Management and Protection

• Existing designs:

1. Enclave’s page table is managed by untrusted OS

2. Enclave can access the application’s entire address space
1. Memory Management and Protection

• Existing designs:

  1. Enclave’s page table is managed by untrusted OS

  2. Enclave can access the application’s entire address space
1. Memory Management and Protection

• Existing designs:

  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space
1. Memory Management and Protection

• Existing designs:

1. Enclave’s page table is managed by untrusted OS

2. Enclave can access the application’s entire address space
1. Memory Management and Protection

• Existing designs:

1. Enclave’s page table is managed by untrusted OS

2. Enclave can access the application’s entire address space

 saldo

Without EPCM hardware

SGX hardware

Intel SGX

Untrusted OS

Page table

validate

SGX hardware

EPCM

Enclave

RustMonitor

Primary OS

Page table

write-protected

Significant overhead (A/D bit updates)

☹
1. Memory Management and Protection

- Existing designs:
  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space

- Significance of overhead: Page-table-based side channel attacks

- Without EPCM hardware:

  ![Diagram showing page table management without EPCM hardware]

- With EPCM hardware:

  ![Diagram showing page table management with EPCM hardware]

- Intel SGX

- RustMonitor

- #PF
1. Memory Management and Protection

- Existing designs:

  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space

- Significant overhead (A/D bit updates)

- Page-table-based side channel attacks

- Without EPCM hardware
1. Memory Management and Protection

- Existing designs:
  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space

 disables page-table-based side channel attacks

Diagram:
- App
- Enclave
- Untrusted OS
- Page table
- SGX hardware
- EPCM
- Intel SGX
1. Memory Management and Protection

- Existing designs:
  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space
1. Memory Management and Protection

- Existing designs:
  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space

![Diagram showing memory management and protection concepts]
1. Memory Management and Protection

• Existing designs:

1. Enclave’s page table is managed by untrusted OS

2. Enclave can access the application’s entire address space
1. Memory Management and Protection

- Existing designs:
  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space
1. Memory Management and Protection

- Existing designs:
  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space

![Diagram showing memory management and protection with Intel SGX and HyperEnclave.]
1. Memory Management and Protection

- Existing designs:
  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space

---

![Diagram showing memory management and protection concepts]
1. Memory Management and Protection

- Existing designs:
  1. Enclave’s page table is managed by untrusted OS
  2. Enclave can access the application’s entire address space

Enclave malware attacks: arbitrary memory accesses to App

Synchronize mapping updates in RustMonitor

- Marshaling buffer
- Pinned to physical memory
- Fixed mapping
1. Memory Management and Protection

HyperEnclave memory isolation

- App's code & data
- Marshalling buffer
- Enclave's code & data
- Encrypted memory

Diagram:

- Normal VM
  - App-B
  - App-A
- Primary OS
- Peripherals
- Normal memory
- EPC memory
- RustMonitor memory
1. Memory Management and Protection

HyperEnclave memory isolation

R1: The primary OS and applications are not allowed to access the physical memory belonging to RustMonitor and the enclaves.
1. Memory Management and Protection

HyperEnclave memory isolation

R2: The enclave is not allowed to access physical memory belonging to RustMonitor and other enclaves.

Nested paging, normal 1-level paging
1. Memory Management and Protection

HyperEnclave memory isolation

R3: DMA accesses from malicious peripherals to the physical memory belonging to RustMonitor and the enclaves are not allowed.
2. Flexible Enclave Operation Mode

- A wide range of existing applications can be offloaded to TEEs:
2. Flexible Enclave Operation Mode

• A wide range of existing applications can be offloaded to TEEs:

  TEEs need to better fulfill the requirements for specific enclave workloads

- Web Server
- Machine learning
- Computing-intensive
- I/O-intensive
- Frequent world switches
- Infrequent world switches
- High TLB pressure
- TEE
- Memory-intensive
- Exception-driven
- In-memory database
- Garbage Collector (GC)
- In-enclave exception handling
- Machine learning
- Computing-intensive
- I/O-intensive
- Frequent world switches
- Infrequent world switches
- High TLB pressure
- TEE
- Memory-intensive
- Exception-driven
- In-memory database
- Garbage Collector (GC)
- In-enclave exception handling
2. Flexible Enclave Operation Mode
2. Flexible Enclave Operation Mode

😊 Slow to handle enclave exception
😊 Not adapt well to various enclave workloads
2. Flexible Enclave Operation Mode

 dém: Slow to handle enclave exception
 dém: Not adapt well to various enclave workloads

Flexible modes!
2. Flexible Enclave Operation Mode

GU-Enclave

- Basic form
2. Flexible Enclave Operation Mode

GU-Enclave

- Basic form
2. Flexible Enclave Operation Mode

P-Enclave

- Access privileged resources:
  - IDT
  - Page tables
- Process privileged events:
  - Interrupts
  - Exceptions
2. Flexible Enclave Operation Mode

P-Enclave

- Access privileged resources:
  - IDT
  - Page tables
- Process privileged events:
  - Interrupts
  - Exceptions

sandboxing untrusted libraries

GARbage collector
RustMonitor
HyperEnclave

Host User
Guest User
Guest Priv.
OS
App

GU-Enclave
Encl.
Encl.
P-Enclave

①
②
③
2. Flexible Enclave Operation Mode

HU-Enclave

- Fast world switches
  - hypercall (~880 cycles)
  - syscall (~120 cycles)
- No virtualization overhead
2. Flexible Enclave Operation Mode

HU-Enclave

- Fast world switches
  - hypercall (~880 cycles)
  - syscall (~120 cycles)
- No virtualization overhead

I/O-intensive workloads

memory-intensive workloads

RustMonitor

HyperEnclave
### 3. Trusted Boot

**Measured Late Launch**

More details about trusted boot and attestation are in the paper.
4. The Enclave SDK

- Retrofit the official SGX SDK¹
- Supported SGX applications:
  - Rust SGX SDK²
  - Occulm library OS³
- Run **existing SGX programs** with little or no code changes

![Rust](https://github.com/intel/linux-sgx)
![Occlum](https://github.com/occlum/occlum)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Leaf</th>
<th>Description</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCLU (user)</td>
<td>EENTER</td>
<td>Enter the enclave</td>
<td>hypercall/syscall</td>
</tr>
<tr>
<td>ENCLU (user)</td>
<td>EEXIT</td>
<td>Exit the enclave</td>
<td>hypercall/syscall</td>
</tr>
<tr>
<td>ENCLU (user)</td>
<td>EREPORT</td>
<td>Create a cryptographic report</td>
<td>hypercall/syscall</td>
</tr>
<tr>
<td>ENCLS (privileged)</td>
<td>ECREATE</td>
<td>Create an enclave</td>
<td>ioctl() ↓ kernel module ↓ hypercall</td>
</tr>
<tr>
<td>ENCLS (privileged)</td>
<td>EADD</td>
<td>Add an enclave page</td>
<td>hypercall/syscall</td>
</tr>
<tr>
<td>ENCLS (privileged)</td>
<td>EEXTEND</td>
<td>Extend enclave measurement</td>
<td>hypercall/syscall</td>
</tr>
<tr>
<td>ENCLS (privileged)</td>
<td>EINIT</td>
<td>Initialize an enclave</td>
<td>hypercall/syscall</td>
</tr>
</tbody>
</table>

¹[https://github.com/intel/linux-sgx](https://github.com/intel/linux-sgx)
²[https://github.com/apache/incubator-teaclave-sgx-sdk](https://github.com/apache/incubator-teaclave-sgx-sdk)
³[https://github.com/occlum/occlum](https://github.com/occlum/occlum)
Implementation

• **AMD** EYPC 7601
  • Hardware virtualization (SVM)
  • Memory encryption (SME)
• Future: ARM, RISC-V, ...
• **Rust** Monitor

<table>
<thead>
<tr>
<th>Component</th>
<th>Language</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RustMonitor</td>
<td>Rust</td>
<td>+7,500</td>
</tr>
<tr>
<td>Kernel module</td>
<td>C</td>
<td>+3,500</td>
</tr>
<tr>
<td>Enclave SDK</td>
<td>C++</td>
<td>+2,000</td>
</tr>
</tbody>
</table>
Evaluation

Methodology

- Platform A: **AMD EPYC 7601**, 512 GB RAM, with **HyperEnclave** and **SME**
- Platform B: **Intel Xeon E3-1270 v6**, 64 GB RAM, with **SGX**
Evaluation
Methodology

• Platform A: **AMD EPYC 7601, 512 GB RAM, with HyperEnclave and SME**

• Platform B: **Intel Xeon E3-1270 v6, 64 GB RAM, with SGX**

*How to compare performance on the two different platforms?*
Evaluation

Methodology

- Platform A: **AMD EPYC 7601**, 512 GB RAM, with **HyperEnclave** and **SME**
- Platform B: **Intel Xeon E3-1270 v6**, 64 GB RAM, with **SGX**

How to compare performance on the two different platforms?

Relative slowdown ↓

Baseline: SDK simulation mode (no security protections)
Evaluation

Methodology

• Platform A: **AMD EPYC 7601, 512 GB RAM, with HyperEnclave and SME**

• Platform B: **Intel Xeon E3-1270 v6, 64 GB RAM, with SGX**

How to compare performance on the two different platforms?

Relative slowdown ↓

Baseline: SDK simulation mode (no security protections)


Evaluation
World switches performance

<table>
<thead>
<tr>
<th></th>
<th>EENTER</th>
<th>EEXIT</th>
<th>ECALL</th>
<th>OCALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel SGX</td>
<td>—</td>
<td>—</td>
<td>14,432</td>
<td>12,432</td>
</tr>
<tr>
<td>GU-Enclave</td>
<td>1,704</td>
<td>1,319</td>
<td>9,480</td>
<td>4,920</td>
</tr>
<tr>
<td>HU-Enclave</td>
<td>1,163</td>
<td>1,144</td>
<td>8,440</td>
<td>4,120</td>
</tr>
<tr>
<td>P-Enclave</td>
<td>1,649</td>
<td>1,401</td>
<td>9,700</td>
<td>5,260</td>
</tr>
</tbody>
</table>

Latency of SGX primitives on HyperEnclave and Intel (in CPU cycles)

1We are unable to measure the instruction latencies on the SGX hardware since the RDTSCP instruction is not supported in the SGX enclaves.
## Evaluation

### P-Enclave use cases: exception handling

The diagram illustrates the process of handling an #UD exception inside the enclaves. The code snippet shown is:

```
0F 0B UD2;
```

and

```
RIP += 2;
```

The table below shows the average CPU cycles of handling an #UD exception inside the enclaves:

<table>
<thead>
<tr>
<th></th>
<th>Intel SGX</th>
<th>GU-Enclave</th>
<th>P-Enclave</th>
</tr>
</thead>
<tbody>
<tr>
<td>#UD</td>
<td>28,561</td>
<td>17,490</td>
<td>258</td>
</tr>
</tbody>
</table>

*Average CPU cycles of handling an #UD exception inside the enclaves*
Evaluation
P-Enclave use cases: exception handling

Two-phase exception handling

<table>
<thead>
<tr>
<th></th>
<th>Intel SGX</th>
<th>GU-Enclave</th>
<th>P-Enclave</th>
</tr>
</thead>
<tbody>
<tr>
<td>#UD</td>
<td>28,561</td>
<td>17,490</td>
<td>258</td>
</tr>
</tbody>
</table>

Average CPU cycles of handling an #UD exception inside the enclaves

Evaluation

P-Enclave use cases: exception handling

Two-phase exception handling

Handled in the enclave!

<table>
<thead>
<tr>
<th></th>
<th>Intel SGX</th>
<th>GU-Enclave</th>
<th>P-Enclave</th>
</tr>
</thead>
<tbody>
<tr>
<td>#UD</td>
<td>28,561</td>
<td>17,490</td>
<td>258</td>
</tr>
</tbody>
</table>

Average CPU cycles of handling an #UD exception inside the enclaves

---

Evaluation

P-Enclave use cases: exception handling

<table>
<thead>
<tr>
<th></th>
<th>Intel SGX</th>
<th>GU-Enclave</th>
<th>P-Enclave</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PF</td>
<td>—¹</td>
<td>2,660</td>
<td>1,132</td>
</tr>
</tbody>
</table>

Average CPU cycles of handling a #PF exception inside the enclaves

¹Our SGX1 platform does not support page permission modifications.
Evaluation

P-Enclave use cases: exception handling

Our SGX1 platform does not support page permission modifications.

<table>
<thead>
<tr>
<th></th>
<th>Intel SGX</th>
<th>GU-Enclave</th>
<th>P-Enclave</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PF</td>
<td>—¹</td>
<td>2,660</td>
<td>1,132</td>
</tr>
</tbody>
</table>

Average CPU cycles of handling a #PF exception inside the enclaves.

¹Our SGX1 platform does not support page permission modifications.
Evaluation

P-Enclave use cases: exception handling

---

**Exception handling**

Enclave

```
write(buf, data);
```

Exception handler

```
buf: R → RW
```

**#PF**

handled in the enclave!

**Handle into RustMonitor**

---

<table>
<thead>
<tr>
<th></th>
<th>Intel SGX</th>
<th>GU-Enclave</th>
<th>P-Enclave</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PF</td>
<td>–¹</td>
<td>2,660</td>
<td>1,132</td>
</tr>
</tbody>
</table>

**Average CPU cycles of handling a #PF exception inside the enclaves**

²³x

---

¹Our SGX1 platform does not support page permission modifications.
Evaluation
Real-world workloads

- SQLite (memory-intensive)
  - < 5%
  - Support large EPC size to avoid page swapping
Evaluation
Real-world workloads

- Lighttpd (I/O-intensive)
  - GU-Enclave: 22% ↓
  - HU-Enclave: 12% ↓
- Redis (I/O and memory intensive)
  - GU-Enclave: 28% ↓
  - HU-Enclave: 11% ↓
Conclusion

• An open process-based TEE
  ▶ Minimum hardware requirements

• SGX compatible
  ▶ Run existing SGX programs with little or no code changes

• Flexible enclave modes
  ▶ Better to fulfill various enclave workloads
Thanks!

• Source code will be available at (still in progress):

  https://github.com/HyperEnclave

• Contact us:

  • Yuekai Jia, equation618@gmail.com
  • Shuang Liu, ls123674@antgroup.com
  • Wenhao Wang, wangwenhao@iie.ac.cn