OBFUSCURO: A Commodity Obfuscation Engine for Intel SGX

Adil Ahmad*, Byunggill Joe*, Yuan Xiao
Yinqian Zhang, Insik Shin, Byoungyoung Lee

(* denotes equal contribution)
Program Obfuscation
**Program Obfuscation**

- **Trusted**
- **Untrusted (except the Black box)**

**Sender’s Goal**
Protect the internals of private program $P_{priv}$
Program Obfuscation

Sender’s Goal
Protect the internals of private program $P_{\text{priv}}$

Trusted

Untrusted (except the Black box)
Program Obfuscation

Sender’s Goal
Protect the internals of private program $P_{priv}$

Encryption Engine

Trusted

Untrusted (except the Black box)
Program Obfuscation

Sender’s Goal
Protect the internals of private program $P_{priv}$

Trusted

Untrusted (except the Black box)

$P_{priv}$

Encryption Engine

Attacker chooses inputs $I^0, I^1, \ldots, I^N$
Program Obfuscation

**Sender's Goal**
Protect the internals of private program $P_{priv}$

$P_{priv}$

Encryption Engine

Attacker chooses inputs $I^0, I^1, \ldots, I^N$

Black box

Untrusted System

Untrusted (except the Black box)
Program Obfuscation

**Sender’s Goal**
Protect the internals of private program $P_{priv}$

**Receiver’s Goal**
Disclose the internals of program $P_{priv}$

$P_{priv}$

Encryption Engine

Attacker chooses inputs $I^0, I^1, ..., I^N$

Black box
Untrusted System

Trusted

Untrusted (except the Black box)
Program Obfuscation

Trusted

Sender’s Goal
Protect the internals of private program $P_{priv}$

$P_{priv}$

Encryption Engine

Untrusted (except the Black box)

Receiver’s Goal
Disclose the internals of program $P_{priv}$

$P_{priv}$

Black box

Untrusted System

Attacker chooses inputs $I^0, I^1, \ldots, I^N$

If the black box is “secure”?
Program Obfuscation

Sender’s Goal
Protect the internals of private program $P_{priv}$

Receiver’s Goal
Disclose the internals of program $P_{priv}$

Sender's Goal

Receiver's Goal

If the black box is “secure”?
After constant time $T$

Untrusted (except the Black box)

Trusted

Encryption Engine

$P_{priv}$

Attacker chooses inputs $I_0, I_1, ..., I_N$

Black box

Untrusted System

Output
Program Obfuscation

Sender’s Goal
Protect the internals of private program $P_{priv}$

Receiver’s Goal
Disclose the internals of program $P_{priv}$

Trusted

Untrusted (except the Black box)

Encryption Engine

Attacker chooses inputs $I^0, I^1 \ldots I^N$

Untrusted System

Black box

Observable execution traces $\Phi^0, \Phi^1 \ldots \Phi^N$

If the black box is “secure”? After constant time $T$

Output

Execution traces should not leak information about $P_{priv}$
Wait, isn’t that what Intel SGX does?
Wait, isn’t that what Intel SGX does?
Wait, isn’t that what Intel SGX does?
Wait, isn’t that what Intel SGX does?

**Trusted** execution region

**Confidentiality** and **integrity** guarantees

Program

Enclave

Non-Enclave
Wait, isn’t that what Intel SGX does?

Trusted execution region

Confidentiality and integrity guarantees

Restricted by the processor

Operating System (and other untrusted software)
Intel SGX is not perfect!
Intel SGX is **not** perfect!
Intel SGX is **not** perfect!

Memory *accessed* by the enclave
Intel SGX is not perfect!

Visible traces on untrusted/shared components!

Memory **accessed** by the enclave

---

Page Table

<table>
<thead>
<tr>
<th>Access</th>
<th>Frame #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU Cache

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Branch Target Buffer

<table>
<thead>
<tr>
<th>Taken</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Timing

---
Intel SGX is not perfect!

Visible traces on untrusted/shared components!

Granularity: 4KB (1 page)
Intel SGX is **not** perfect!

During enclave execution, memory accessed by the enclave can lead to visible traces on untrusted/shared components.

- **Page Table**
  - Granularity: 4KB (1 page)
  - Access
  - Frame #

- **CPU Cache**
  - Granularity: 64B (1 line)

- **Branch Target Buffer**
  - Taken
  - Address

- **Timing**
Intel SGX is not perfect!

Memory *accessed* by the enclave

Visible traces on untrusted/shared components!

Granularity:
- Page Table: 4KB (1 page)
- CPU Cache: 64B (1 line)
- Branch Target Buffer: Jmp address

Operating System
Intel SGX is not perfect!

Visible traces on untrusted/shared components!

Granularity: 4KB (1 page)
Granularity: 64B (1 line)
Granularity: Jmp address
Granularity: Execution Time

Page Table
CPU Cache
Branch Target Buffer
Timing

Granularity: 4KB (1 page)
Granularity: 64B (1 line)
Granularity: Jmp address
Granularity: Execution Time

Operating System
Intel SGX is **not** perfect!

Visible traces on untrusted/shared components!

**Granularity:**
- **Page Table:** 4KB (1 page)
- **CPU Cache:** 64B (1 line)
- **Branch Target Buffer:** Jmp address
- **Timing:** Execution Time

**Operating System**

**Paging, Branch-prediction and Cache attacks!**
[S&P14, SEC17, ASPLOS18, DIMVA17, WOOT17]
Learning from **existing solutions**!
Learning from existing solutions!

Access patterns attacks!
Learning from existing solutions!

Access patterns attacks!

Possible Soln.

Transactional Memory
[NDSS17, SEC17]

Incomplete
Learning from existing solutions!

Access patterns attacks!

Possible Soln.

- Transactional Memory
  - Incomplete
  - [NDSS17, SEC17]

- Cache Partitioning
  - ring-0 required
  - [SEC18]
Learning from existing solutions!

Access patterns attacks!

Possible Soln.

- Transactional Memory [NDSS17, SEC17]
  - Incomplete

- Cache Partitioning [SEC18]
  - ring-0 required

- Address Randomization [NDSS17]
  - Insecure
Lesson #1

**Ring-3** enclaves cannot hide access patterns through side-channels!
Lesson #1

**Ring-3** enclaves cannot hide access patterns through side-channels!
Learning from existing solutions!

Access patterns attacks!
- Transactional Memory [NDSS17, SEC17] - Incomplete
- Cache Partitioning [SEC18] - ring-0 required
- Address Randomization [NDSS17] - Insecure

Timing attacks!
- RDTSC - OS-controllable

Lesson #1
*Ring-3* enclaves cannot hide access patterns through side-channels!
Learning from existing solutions!

Access patterns attacks!

Possible Soln.

Transaction Memory [NDSS17, SEC17]
Cache Partitioning [SEC18]
Address Randomization [NDSS17]

Incomplete
ring-0 required
Insecure

Timing attacks!

Possible Soln.

RDTSC
Network timers

OS-controllable
OS-controllable

Lesson #1

Ring-3 enclaves cannot hide access patterns through side-channels!
Learning from existing solutions!

**Access patterns attacks!**
- Transactional Memory [NDSS17, SEC17]
- Cache Partitioning [SEC18]
- Address Randomization [NDSS17]

- Incomplete
- ring-0 required
- Insecure

**Timing attacks!**
- RDTSC
- Network timers
- Thread timers

- OS-controllable
- OS-controllable
- OS-controllable

**Lesson #1**
*Ring-3* enclaves cannot hide access patterns through side-channels!
Learning from existing solutions!

**Lesson #1**
*Ring-3* enclaves cannot hide access patterns through side-channels!

**Lesson #2**
*Unreliable* timers for SGX enclaves!
Our approach
Our approach

• Indistinguishable enclave program(s)
Our approach

- Indistinguishable enclave program(s)
  - A code block executed \textbf{N times} on C-Pad, and data block accessed from D-Pad
Our approach

• **Indistinguishable enclave program(s)**
  • A code block executed **N times** on C-Pad, and data block accessed from D-Pad
  • C-Pad and D-Pad are **one cache-line** (64B) in size!
Our approach

• Indistinguishable enclave program(s)
  • A code block executed $N$ times on C-Pad, and data block accessed from D-Pad
  • C-Pad and D-Pad are one cache-line (64B) in size!

# of executions: 0
Our approach

• Indistinguishable enclave program(s)
  • A code block executed **N times** on C-Pad, and data block accessed from D-Pad
  • C-Pad and D-Pad are **one cache-line** (64B) in size!

# of executions: 0
Our approach

- Indistinguishable enclave program(s)
  - A code block executed **N times** on C-Pad, and data block accessed from D-Pad
  - C-Pad and D-Pad are **one cache-line** (64B) in size!

`# of executions: 1`

**Branch** to the start of C-Pad

**Single** data access
Our approach

- Indistinguishable enclave program(s)
  - A code block executed **N times** on C-Pad, and data block accessed from D-Pad
  - C-Pad and D-Pad are **one cache-line** (64B) in size!

**# of executions:** 1

**Branch** to the start of C-Pad

**Single** data access
Our approach

- Indistinguishable enclave program(s)
  - A code block executed **N times** on C-Pad, and data block accessed from D-Pad
  - C-Pad and D-Pad are **one cache-line** (64B) in size!

**# of executions: N**

**Branch** to the start of C-Pad

**Single** data access

C-Pad 64B

D-Pad 64B
Our approach

• Indistinguishable enclave program(s)
  • A code block executed **N times** on C-Pad, and data block accessed from D-Pad
  • C-Pad and D-Pad are **one cache-line** (64B) in size!

# of executions: $N$

**Branch** to the start of C-Pad

**Single** data access

What do the attacks reveal?
Our approach

- Indistinguishable enclave program(s)
  - A code block executed \( N \) times on C-Pad, and data block accessed from D-Pad
  - C-Pad and D-Pad are **one cache-line** (64B) in size!

**Paging Attack:** Same page
Our approach

- Indistinguishable enclave program(s)
  - A code block executed $N$ times on C-Pad, and data block accessed from D-Pad
  - C-Pad and D-Pad are one cache-line (64B) in size!

# of executions: $N$

**Branch** to the start of C-Pad

**Single** data access

What do the attacks reveal?

- Paging Attack: Same page
- Cache Attack: Same cache-lines
Our approach

- **Indistinguishable enclave program(s)**
  - A code block executed \(N\) times on C-Pad, and data block accessed from D-Pad
  - C-Pad and D-Pad are one cache-line (64B) in size!

**What do the attacks reveal?**

- **Paging Attack**: Same page
- **Cache Attack**: Same cache-lines
- **Branch Attack**: Same branch

**Branch** to the start of C-Pad

\(\# \text{ of executions: } N\)

**Single** data access
Our approach

- Indistinguishable enclave program(s)
  - A code block executed **N times** on C-Pad, and data block accessed from D-Pad
  - C-Pad and D-Pad are **one cache-line** (64B) in size!

**What do the attacks reveal?**

**Paging Attack:** Same page
**Cache Attack:** Same cache-lines
**Branch Attack:** Same branch
**Timing Attack:** Same time to execute N code blocks

# of executions: **N**
Our approach

- Indistinguishable enclave program(s)
  - A code block executed \( N \) times on C-Pad, and data block accessed from D-Pad
  - C-Pad and D-Pad are one cache-line (64B) in size!

Instead of trying to hide traces, all enclaves should leak the same traces!
Let **Hermione** explain!
Let Hermione explain!
Let Hermione explain!

Operating System

Enclave 1

Enclave 2

Before (Native)
Let Hermione explain!

Operating System

Obfuscuro

Enclave₁

Enclave₂

Before
(Native)

Pattern
Let **Hermione** explain!

**Obfuscuro**

**Operating System**

**Enclave**

- Enclave$_1$
- Enclave$_2$

**Before** (Native)

**After** (Obfuscuro)
Cool, what’s the challenge?
Cool, what’s the challenge?

• Naïve solution
  • Use a software-translator to copy all code and data onto C/D-Pad
Cool, what’s the **challenge**?

- **Naïve solution**
  - Use a *software-translator* to copy all code and data onto C/D-Pad
Cool, what’s the challenge?

• Naïve solution
  • Use a software-translator to copy all code and data onto C/D-Pad

C1. Native code is not in 64B blocks!
Cool, what’s the **challenge**?

- **Naïve solution**
  - Use a [software-translator](#) to copy all code and data onto C/D-Pad

**C1.** Native code is **not in 64B blocks!**

**C2.** Access patterns **leaked while copying!**
Cool, what’s the challenge?

- Naïve solution
  - Use a software-translator to copy all code and data onto C/D-Pad

  **C1.** Native code is *not in 64B blocks!*

  **C2.** Access patterns *leaked while copying!*

  **C3.** Code can have *different branches!*

---

- **C1.** Native code is *not in 64B blocks!*
- **C2.** Access patterns *leaked while copying!*
- **C3.** Code can have *different branches!*
Cool, what’s the challenge?

- Naïve solution
  - Use a software-translator to copy all code and data onto C/D-Pad

C1. Native code is not in 64B blocks!

C2. Access patterns leaked while copying!

C3. Code can have different branches!

C4. Timing issues not even discussed!
Obfuscuro

- Program obfuscation on Intel SGX
  - All programs should exhibit same patterns irrespective of logic/input.
  - Adapted from Harry Potter spell “Obscuro” (translation: > Darkness)
C1. Enforce code blocks of identical sizes
C1. Enforce code blocks of identical sizes

- Break code blocks into 64 bytes and pad using `nop`
C1. Enforce code blocks of identical sizes

• Break code blocks into 64 bytes and pad using `nop`
C1. Enforce code blocks of identical sizes

- Break code blocks into 64 bytes and pad using \texttt{nop}
C1. Enforce code blocks of identical sizes

- Break code blocks into 64 bytes and pad using `nop`
C1. Enforce code blocks of identical sizes

- Break code blocks into 64 bytes and pad using `nop`

**64B (single cache-line) code blocks** can be loaded onto the C-Pad!
C2. Securely loading C/D-Pad
C2. Securely loading C/D-Pad

• Fetch code and data using **Oblivious RAM (ORAM)**
  • The code and data is fetched onto C-Pad and D-Pad resp.
C2. Securely loading C/D-Pad

- Fetch code and data using **Oblivious RAM (ORAM)**
  - The code and data is fetched onto **C-Pad** and **D-Pad** resp.
C2. Securely loading C/D-Pad

- Fetch code and data using **Oblivious RAM (ORAM)**
  - The code and data is fetched onto **C-Pad** and **D-Pad** resp.
C2. Securely loading C/D-Pad

- Fetch code and data using **Oblivious RAM (ORAM)**
  - The code and data is fetched onto **C-Pad** and **D-Pad** resp.
C2. Securely loading C/D-Pad

- Fetch code and data using **Oblivious RAM (ORAM)**
  - The code and data is fetched onto **C-Pad** and **D-Pad** resp.
C2. Securely loading C/D-Pad

- Fetch code and data using **Oblivious RAM (ORAM)**
  - The code and data is fetched onto **C-Pad** and **D-Pad** resp.
C2. Securely loading C/D-Pad

• Fetch code and data using **Oblivious RAM (ORAM)**
  • The code and data is fetched onto **C-Pad** and **D-Pad** resp.

1. **Execute** old code block
2. **Request** new code block
3. **Retrieve** the block using ORAM
4. **Update** C-Pad with new code block

Instrumented code is located in C-Tree
C2. Securely loading C/D-Pad

- Fetch code and data using **Oblivious RAM (ORAM)**
  - The code and data is fetched onto **C-Pad** and **D-Pad** resp.

![Diagram](image)

- **Execute** new code block
- **Update** C-Pad with new code block
- **Retrieve** the block using ORAM
- **Request** new code block

Instrumented code is located in C-Tree
C2. Securely loading C/D-Pad

- Fetch code and data using Oblivious RAM (ORAM)
  - The code and data is fetched onto C-Pad and D-Pad resp.

Side-channel-resistant ORAM scheme ensures no leakage as C/D-Pad are loaded!
C3. Align branches to/from C-Pad
C3. Align branches to/from C-Pad

• Each instrumented code block has **two branches** to fixed locations
  • C-Pad → Code-Controller
  • C-Pad → Data-Controller
C3. Align branches to/from C-Pad

- Each instrumented code block has two branches to fixed locations
  - C-Pad → Code-Controller
  - C-Pad → Data-Controller

CPU-bound instructions

Data Controller
- stash
- pos. map

Code Controller
- stash
- pos. map
C3. Align branches to/from C-Pad

• Each instrumented code block has **two branches** to fixed locations
  • C-Pad → Code-Controller
  • C-Pad → Data-Controller
C3. Align branches to/from C-Pad

- Each instrumented code block has two branches to fixed locations
  - C-Pad → Code-Controller
  - C-Pad → Data-Controller

**Code execution model**

**Data access model**

**C/D-Controller have no conditional branches!**
C3. Align branches to/from C-Pad

- Each instrumented code block has **two branches** to fixed locations
  - C-Pad $\rightarrow$ Code-Controller
  - C-Pad $\rightarrow$ Data-Controller

All Obfuscuro programs execute the **same sequence of branches**!
C4. Ensuring execution time consistency
C4. Ensuring execution time consistency

• The program executes fixed number of code blocks
C4. Ensuring execution time consistency

- The program executes **fixed** number of code blocks
C4. Ensuring execution time consistency

- The program executes **fixed** number of code blocks
C4. Ensuring execution time consistency

- The program executes **fixed** number of code blocks
C4. Ensuring execution time consistency

- The program executes **fixed** number of code blocks
C4. Ensuring execution time consistency

- The program executes **fixed** number of code blocks

![Diagram showing code block requests and ORAM Bank]

Contains **dummy but indistinguishable** code blocks
C4. Ensuring execution time consistency

- The program executes **fixed** number of code blocks

![Diagram](image-url)

- **Request** next code block
- **Retrieves** the next block
- **Fetches output and exits** enclave!
- **Contains dummy but indistinguishable code blocks**

**Diagram Description**

- **C-Pad**: 64B
- **Code Controller**: stash, pos. map
- **C-Tree**: ORAM Bank
- **Term**: After \(N\) blocks
- **Return to C-Pad**: After \(N\) blocks
- **ORAM Bank**: Contains dummy but indistinguishable code blocks
C4. Ensuring execution time consistency

- The program executes fixed number of code blocks

Execute $N$ code blocks to ensure all programs terminate consistently!
Faster memory store for enclaves
Faster memory store for enclaves

• Use AVX registers as store instead of "Oblivious" store
Faster memory store for enclaves

- Use **AVX registers** as store instead of "Oblivious" store
Faster memory store for enclaves

- Use **AVX registers** as store instead of "Oblivious" store

Have to **sequentially** access all memory indices

**C-Pad**

| 64B |

**Code Controller**

- stash
- pos. map

**DRAM**

**CPU**

**AVX registers**

**DRAM-based store**
Faster memory store for enclaves

- Use AVX registers as store instead of "Oblivious" store

- Have to sequentially access all memory indices

- Can access individual registers obliviously!

- Use AVX registers as store instead of "Oblivious" store
Faster memory store for enclaves

- Use AVX registers as store instead of "Oblivious" store

AVX registers can be used as a **faster, oblivious storage** for SGX enclaves!
Implementation
Implementation

• LLVM compiler suite (3117 LoC)
  • Breaks all code into similar blocks (C1)
  • Instrument and align all control and data-flow instructions (C3)
Implementation

• LLVM compiler suite (3117 LoC)
  • Breaks all code into similar blocks (C1)
  • Instrument and align all control and data-flow instructions (C3)

• Runtime library (2179 LoC)
  • Initializes ORAM trees and performs secure ORAM operations (C2)
  • Terminate program and fetch output (C4)
Implementation

• LLVM compiler suite (3117 LoC)
  • Breaks all code into similar blocks (C1)
  • Instrument and align all control and data-flow instructions (C3)

• Runtime library (2179 LoC)
  • Initializes ORAM trees and performs secure ORAM operations (C2)
  • Terminate program and fetch output (C4)

• Intel SGX SDK (25 LoC)
  • Assign memory regions for C/D-Pad (support)
Performance Evaluation

Overhead (times)

Programs

- SUM: 16
- PALINDROME: 27
- BINSEARCH: 68
- MATTRANS: 85
- ADNAGRAM: 121
- MATMUL: 231
We ported ~10 simple applications to Obfuscuro!
Average overhead observed is **81 times over native programs!**

We ported ~**10 simple applications** to Obfuscurso!
Performance Evaluation

Average overhead observed is **81 times over native programs**!

The overhead is *highly dependent on input size and program type*!

We ported ~10 simple applications to Obfuscuro!
Ending Remarks!
Ending Remarks!

1. Program obfuscation is a remarkable dream to achieve
Ending Remarks!

1. Program obfuscation is a remarkable dream to achieve.
2. Various software/hardware limitations hinder the realization of program obfuscation on Intel SGX.
Ending Remarks!

1. Program obfuscation is a **remarkable dream** to achieve
2. Various **software/hardware limitations** *hinder* the realization of program obfuscation on Intel SGX
3. Existing solutions have a **limited approach** towards side-channel mitigation in Intel SGX
Ending Remarks!

1. Program obfuscation is a remarkable dream to achieve
2. Various software/hardware limitations hinder the realization of program obfuscation on Intel SGX
3. Existing solutions have a limited approach towards side-channel mitigation in Intel SGX
4. Obfuscuro is compiler-based scheme which addresses this issue by ensuring all programs leak same access patterns

Contact: ahmad37@purdue.edu
감사합니다
(Translation ~ Thanks!) ;}
Execution Time Evaluation

ORAM access time dominates the time of code block execution!