ZeroTrace: Oblivious Memory Primitives from Intel SGX

Sajin Sasy\textsuperscript{1}, Sergey Gorbunov\textsuperscript{1} and Christopher Fletcher\textsuperscript{2}

\textsuperscript{1} - University of Waterloo, \textsuperscript{2} - University of Illinois at Urbana-Champaign
Secure Remote Computations

Alice with Program($P$) and Data($D$) wishes to compute $P(D)$

Desirable Properties:

- **Confidentiality**: The server learns nothing about $D$
- **Integrity**: The server can only return $P(D)$ and no other function of $D$
- **Efficiency**: It executes in time close to natively executing $P(D)$
Solution in the ideal world

Alice with Program(P) and Data(D) wishes to compute P(D)

Desirable Properties:
● **Confidentiality**: The server learns nothing about D
● **Integrity**: The server can only return P(D) and no other function of D
● **Efficiency**: It executes in time close to natively executing P(D)
Real world tools and techniques

Software Solutions:

FHE [1]:

Real world tools and techniques

Software Solutions:

FHE [1]:
- Confidentiality ✓
- Integrity ✗
- Efficiency ✗

Real world tools and techniques

Software Solutions:

FHE [1]:
- Confidentiality: ✓
- Integrity: ✗
- Efficiency: ✗

ORAM [2]:
- Confidentiality: ✓
- Integrity: ✓
- Efficiency: ✗

Real world tools and techniques

**Software Solutions:**

FHE [1]:
- Confidentiality ✓
- Integrity ×
- Efficiency ×

ORAM [2]:
- Confidentiality ✓
- Integrity ✓
- Efficiency ×

**Hardware Solutions:**

Intel TPM+TXT [3]:
- Confidentiality ×
- Integrity ✓
- Efficiency ✓

Real world tools and techniques

**Software Solutions:**

FHE [1]:
- Confidentiality: ✓
- Integrity: ×
- Efficiency: ×

ORAM [2]:
- Confidentiality: ✓
- Integrity: ✓
- Efficiency: ×

**Hardware Solutions:**

Intel TPM+TXT [3]:
- Confidentiality: ×
- Integrity: ✓
- Efficiency: ✓

Intel SGX [4]:
- Confidentiality: ✓
- Integrity: ✓
- Efficiency: ✓

---

# Real world tools and techniques

## Software Solutions:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Confidentiality</th>
<th>Integrity</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>FHE [1]</td>
<td>✅</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>ORAM [2]</td>
<td>✅</td>
<td></td>
<td>❌</td>
</tr>
</tbody>
</table>

## Hardware Solutions:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Confidentiality</th>
<th>Integrity</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel TPM+TXT [3]</td>
<td>❌</td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>Intel SGX [4]</td>
<td>✅</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### References:

Our Goal

Can we design a solution that meets all the three desirable properties of secure remote computation?

Yes, ZeroTrace.

Our Approach:

Privacy of ORAM
+  
Efficiency of SGX
Outline

1. Secure Remote Computation ✓
2. Preliminaries:
   ● Intel SGX
   ● ORAM
3. ZeroTrace Architecture
4. Evaluation
Preliminaries

Intel SGX Background
Intel SGX - Software Guard eXtensions

- x86 instructions extension
• x86 instructions extension
• Trusted processor fused with secret keys
Intel SGX - Software Guard eXtensions

- x86 instructions extension
- Trusted processor fused with secret keys
- Processor Reserved Memory (PRM) set aside securely at boot
Intel SGX - Software Guard eXtensions

- x86 instructions extension
- Trusted processor fused with secret keys
- Processor Reserved Memory (PRM) set aside securely at boot
- Secure virtual containers called **enclaves**
- x86 instructions extension
- Trusted processor fused with secret keys
- Processor Reserved Memory (PRM) set aside securely at boot
- Secure virtual containers called **enclaves**
- “Secure as long as processor isn’t physically broken into.”
Secure Remote Computations with Intel SGX

Properties:
- Confidentiality ✓
- Integrity ✓
- Efficiency ✓
Secure Remote Computations with Intel SGX

Properties:
- Confidentiality ✗
- Integrity ✓
- Efficiency ✓
Security Limitations of SGX

Research has shown that SGX is susceptible to side channel attacks. These attacks enable an adversary to extract secret data from enclaves!

- Page Fault Attacks [1,2]
- Branch Shadowing Attack [3]
- Cache Attacks [4,5]
- Data Access Pattern Attacks [1,6]

Functional Limitations of SGX

- Effective PRM limited to 90 MB (with expensive cost for paging)
- No direct IO / syscalls
- Expensive context switching due to Asynchronous Enclave Exits (AEX)
1. Secure Remote Computation ✓
2. Preliminaries:
   ● Intel SGX - Lightning Tour ✓
   ● ORAM
3. ZeroTrace Architecture
4. Evaluation
Preliminaries

ORAM or Oblivious RAM
Oblivious RAM [1]

- Cryptographic primitive designed to hide memory access patterns
- All ORAMs fundamentally require a probabilistic encryption schema

Tree based ORAMs [1,2,3]

Data Blocks  

Tree based ORAMs [1,2,3]

Tree based ORAMs [1,2,3]

Invariant: A block in the system will always reside in either the local stash, or in its path to the leaf label on the server tree.

Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>l</td>
</tr>
</tbody>
</table>

Server

1 Fetch leaf for id = 7

N leaves

1 2 .... l .. N-1 N
Tree based ORAMs - Access

**Client**

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ℓ'</td>
</tr>
</tbody>
</table>

1. Fetch leaf for id = 7
2. Sample new leaf ℓ'

**Server**

N leaves
Tree based ORAMs - Access

Client
Request ID : 7
Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ℓ'</td>
</tr>
</tbody>
</table>

1. Fetch leaf for id = 7
2. Sample new leaf ℓ'
3. Request path to leaf ℓ

Server

N leaves

1 2 .... N-1 N
Tree based ORAMs - Access

1. Fetch leaf for id = 7
2. Sample new leaf $\ell'$
3. Request path to leaf $\ell$

**Client**

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>$\ell'$</td>
</tr>
</tbody>
</table>

**Server**

$\ell$

$\ell$

1 2 ... N leaves
Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ℓ'</td>
</tr>
</tbody>
</table>

1. Fetch leaf for id = 7
2. Sample new leaf ℓ'
3. Request path to leaf ℓ
4. Return path to leaf path

Server

```
<id,leaf,data>
...
...
...
<id,leaf,data>
...
...
...
<id,leaf,data>
...
...
...
```

ℓ
Tree based ORAMs - Access

Client
Request ID: 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>( \ell' )</td>
</tr>
</tbody>
</table>

Stash

<table>
<thead>
<tr>
<th>&lt;id,leaf,data&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;id,leaf,data&gt;</td>
</tr>
<tr>
<td>&lt;id,leaf,data&gt;</td>
</tr>
</tbody>
</table>

Server

\[ \langle \text{id}, \text{leaf}, \text{data} \rangle \]

\[ \ldots \]

\[ \langle \text{id}, \text{leaf}, \text{data} \rangle \]

\[ \ldots \]

\[ \langle \text{id}, \text{leaf}, \text{data} \rangle \]

\[ \ldots \]

\[ \langle \text{id}, \text{leaf}, \text{data} \rangle \]

1. Fetch leaf for id = 7
2. Sample new leaf \( \ell' \)
3. Request path to leaf \( \ell \)
4. Return path to leaf path
5. Push real blocks into stash
Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ℓ'</td>
</tr>
</tbody>
</table>

Stash

<id,leaf,data>
<id,leaf,data>
<id,leaf,data>

Server

<id,leaf,data>
...
...
...
<id,leaf,data>
...
...
...
<id,leaf,data>
...
...
...

1. Fetch leaf for id = 7
2. Sample new leaf ℓ'
3. Request path to leaf ℓ
4. Return path to leaf path
5. Push real blocks into stash
6. Rebuild path from stash
Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ℓ'</td>
</tr>
</tbody>
</table>

Stash

<table>
<thead>
<tr>
<th>&lt;id,leaf,data&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>&lt;id,leaf,data&gt;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>&lt;id,leaf,data&gt;</td>
</tr>
</tbody>
</table>

Server

<table>
<thead>
<tr>
<th>&lt;id,leaf,data&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>&lt;id,leaf,data&gt;</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>&lt;id,leaf,data&gt;</td>
</tr>
</tbody>
</table>

1. Fetch leaf for id = 7
2. Sample new leaf ℓ'
3. Request path to leaf ℓ
4. Return path to leaf path
5. Push real blocks into stash
6. Rebuild path from stash
7. Return new path to leaf new_path
Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ℓ'</td>
</tr>
</tbody>
</table>

Stash

<id,leaf,data>
<id,leaf,data>
<id,leaf,data>

1. Fetch leaf for id = 7
2. Sample new leaf ℓ'
3. Request path to leaf ℓ
4. Return path to leaf path
5. Push real blocks into stash
6. Rebuild path from stash
7. Return new path to leaf new_path

Server

<id,leaf,data>
<id,leaf,data>
<id,leaf,data>

ORAM Controller

12
1. Secure Remote Computation ✔
2. Preliminaries :
   ● Intel SGX - Lightning Tour ✔
   ● ORAM ✔
3. ZeroTrace Architecture
4. Evaluation
ZeroTrace Architecture
By default we consider a malicious active adversary
Everything on the server stack except the processor is untrusted
Problems:
1. Controller code susceptible to side channel leakages
Problems:
1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
Problems:
1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
3. Position map could exceed available PRM
Problems:
1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
3. Position map could exceed available PRM
4. Enclaves do not have IO support
Problems:
1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
3. Position map could exceed available PRM
4. Enclaves do not have IO support
Problems:
1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
3. Position map could exceed available PRM
Attempt 2: Recursive ORAMs

Position Map X

\[ \text{N / c}^x \text{ entries} \]

Position Map 2

\[ \text{N/c leaves} \]

Position Map 1

\[ \text{N leaves} \]
Attempt 2

Problems:
1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
Evaluation: Recursion

Data blocks are of 1 KB size in this experiment.
Attempt 3: Solving the security issues

Problems:
1. Controller code susceptible to side channel leakages
2. Access pattern leakages of position map and stash accesses
1) Oblivious functions at assembly level
   ● Library of assembly-level functions for oblivious operations.
   ● Wrapper functions over CMOV instruction [14,15]
   ● Example function:

```
outupdate <srcT, destT> (bool cond, srcT *src, destT *dest, sizeT sz)
```

2) Constant time code for the underlying ORAM schema
   ● Code branches must be data independent
   ● Access to stash and position map are made through linear scans
ZeroTrace
ZeroTrace

- First oblivious memory controller on a real secure hardware platform
- Flexible storage backends
- ZeroTrace is secure against ALL software side-channel attacks since it realizes the oblivious enclave execution definition.
1) Memory Protection for Secure Computation
   - Memory controller for other enclaves
   - Data accesses are now side-channel secure

![Diagram showing memory protection for secure computation](image)
Evaluation: ZeroTrace performance with small data size

Data blocks are of 8 bytes size in this experiment.
2) Remote Oblivious Data Storage
   - Order of magnitude network bandwidth saving
   - Order of magnitude decrease in access latency
Evaluation: ZeroTrace performance with increasing data sizes

![Graph showing time per query vs block size for Circuit ORAM and Path ORAM.](image)
How to use ZeroTrace

In order to use oblivious memory via ZeroTrace, where necessary:

- Create an oblivious memory abstraction by:
  \[\text{ZeroTrace\_New (label, N, block\_size, \langle\text{params}\rangle)}\]

- Access this oblivious memory by:
  \[\text{ZeroTrace\_Access (label, id, op, data*)}\]
Summary

- Illustrated design and evaluation of ZeroTrace
- Showed how to achieve efficient secure remote computation through ZeroTrace
- Go play with ZeroTrace: https://github.com/Sajin7/ZT
Summary

- Illustrated design and evaluation of ZeroTrace
- Showed how to achieve efficient secure remote computation through ZeroTrace
- Go play with ZeroTrace: [https://github.com/Sajin7/ZT](https://github.com/Sajin7/ZT)
Bonus Slides !
Comparing with Hardware ORAM solutions:

- No deployed / practically available solution
  Since H/W required is custom and not commercially available
- Typically tied to DRAM storage
- All or nothing, no flexibility
Comparing with Raccoon:

- Experiments on Xeon processors (No SGX Support)
- Parameterized to fit recursion within the register space, and discarded recursive ORAMs for SGX setting
- Streaming doesn’t account for encryption/decryption overhead
How does Meltdown/Spectre affect ZeroTrace

- Meltdown does not effect ZeroTrace. No PoC currently
- Spectre_1 doesn’t pan out since there are no branches
- Spectre_2 has been patched by Intel already
- We are still investigating this
High-level Security of ZeroTrace

Side-channel proofed using oblivious techniques

Inherits traditional ORAM security
Future Steps:

- Deploying ZeroTrace as an open source library
- Optimizing data structure support
- Optimizing initialization costs
- Asynchronous ORAM
When a program $P$ is loaded in an enclave, and a set of inputs $y = (\text{in}_1, \ldots, \text{in}_M)$ are executed by this enclave it results in an adversarial view $V(y) = \text{trace}((E_p, \text{in}_1), \ldots, (E_p, \text{in}_M))$. We say that the enclave execution is oblivious if given two sets of inputs $y$ and $z$, their adversarial views $V(y)$ and $V(z)$ are computationally indistinguishable.

Here $\text{trace}(E_p, \text{in})$ captures the execution trace induced by running the enclave $E_p$ with input in. This $\text{trace}(E_p, \text{in})$ contains all the powerful side channel artifacts that the adversary can view such as cache usage, page faults, etc.
Non-Oblivious Leaf-label Retrieval:

```python
newleaf = random(N)
leaf = position_map[x]
position_map[x] = newleaf
```

Oblivious Leaf-label Retrieval:

```python
newleaf = random(N)
for i in range(0, N):
    cond = (i == x)
    update(cond, position_map[i], leaf, size)
    update(cond, newleaf, position_map[i], size)
```
Other Graphs
Evaluation: Memory Primitives

![Graph showing the time per query in ms against the number of blocks for different memory primitives: Array, List, Set, Dictionary with different data sizes. The graph illustrates the performance comparison of these structures as the number of blocks increases.]
Evaluation: Flexibility of Controller

Data blocks are of 1 KB size in this experiment.
Evaluation: Breakdown of Request Time
Overhead of EPC memory accesses
oupdate() in depth
update <srcT, destT> (bool cond, uint32_t src, uint32_t dest, sizeT sz)
Building blocks for side-channel proofing

\texttt{update \langle srcT, destT\rangle (bool cond, uint32_t src, uint32_t dest, sizeT sz)}
update <srcT, destT> (bool cond, uint32_t src, uint32_t dest, sizeT sz)
Building blocks for side-channel proofing

```c
update <srcT, destT> (bool cond, uint32_t src, uint32_t dest, sizeT sz)
```

```
mov eax, src
mov ebx, dst
mov ecx, flag
```

```
cmp ecx, 1
```

```
cmovz ebx, eax
```

```
mov src, eax
mov dst, ebx
```

```
Processor Reserved
Memory
```

```
src
dst
cond
```

```
 Trusted Processor
```

```
 eax
 ebx
 ecx
```

```
 1
mov eax, src
mov ebx, dst
mov ecx, flag
```

```
 2
cmp ecx, 1
cmovz ebx, eax
```

```
 3
```
Access Protocol for Tree based ORAM schemes
Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>l</td>
</tr>
</tbody>
</table>

1. Fetch leaf for id = 7 from Position Map

Server

N leaves

1 2 .... l .... N-1 N
Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>l</td>
</tr>
</tbody>
</table>

1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf l

Server

N leaves

1 2 .... l .. N-1 N
Tree based ORAMs - Access

**Client**

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>p</td>
</tr>
</tbody>
</table>

**Server**

1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf \( l \)
3. Sample new leaf \( p \)

N leaves

```
1 2 .... N-1 N
```

```
\( l \)
```
Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>p</td>
</tr>
</tbody>
</table>

Server

1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf \( l \)
3. Sample new leaf \( p \)

N leaves

1 2 .... N-1 N
Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>p</td>
</tr>
</tbody>
</table>

Server

1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf $l$
3. Sample new leaf $p$
4. Return path to leaf

path
Tree based ORAMs - Access

**Client**

Request ID: 7

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>p</td>
</tr>
</tbody>
</table>

Position Map

1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf \( l \)
3. Sample new leaf \( p \)
4. Return path to leaf
5. Push real blocks on path into stash

**Stash**

- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)

**Server**

- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)
- \(<id,leaf,\text{data}>\)
Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>p</td>
</tr>
</tbody>
</table>

Stash

- `<id,leaf,data>`
- `<id,leaf,data>`
- `<id,leaf,data>`

Server

1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf \( l \)
3. Sample new leaf \( p \)
4. Return path to leaf
5. Push real blocks on path into stash
6. Rebuild path from stash

new_path
Tree based ORAMs - Access

Client

Request ID : 7

Position Map

<table>
<thead>
<tr>
<th>id</th>
<th>leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>p</td>
</tr>
</tbody>
</table>

Stash

<id,leaf,data>  
<id,leaf,data>  
<id,leaf,data>  

Server

<id,leaf,data>  
<id,leaf,data>  
<id,leaf,data>  

1. Fetch leaf for id = 7 from Position Map
2. Request path to leaf $l$
3. Sample new leaf $p$
4. Return path to leaf
5. Push real blocks on path into stash
6. Rebuild path from stash
7. Return new path to leaf $l$ back to server
Tree based ORAMs - Access

Client

- Request ID: 7
- Position Map:
  - id | leaf
  - 7  | p

Stash:
- <id,leaf,>data
- <id,leaf,>data
- <id,leaf,>data
- <id,leaf,>data
- <id,leaf,>data
- <id,leaf,>data

Server

- Request path to leaf l
- Sample new leaf p
- Return path to leaf
- Push real blocks on path into stash
- Rebuild path from stash
- Return new path to leaf l back to server