ShieldStore: Shielded In-memory Key-value Storage with SGX

Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh

Dresden, Germany
March 25-28, 2019
Trusted Key-value Stores

• User data is exposed to malicious attackers in clouds

• Hardware-based security supports
  – Provide *trusted execution environment* for remote server
Trusted Key-value Stores

• User data is exposed to malicious attackers in clouds

• Hardware-based security supports
  – Provide *trusted execution environment* for remote server
Trusted Key-value Stores

• User data is exposed to malicious attackers in clouds

• Hardware-based security supports
  – Provide *trusted execution environment* for remote server

Malicious Users & Administrator
Trusted Key-value Stores

- User data is exposed to malicious attackers in clouds

- Hardware-based security supports
  - Provide *trusted execution environment* for remote server
Trusted Key-value Stores

- User data is exposed to malicious attackers in clouds

- Hardware-based security supports
  - Provide *trusted execution environment* for remote server
Trusted Key-value Stores

- User data is exposed to malicious attackers in clouds

- Hardware-based security supports
  - Provide *trusted execution environment* for remote server
Trusted Key-value Stores

• User data is exposed to malicious attackers in clouds

• Hardware-based security supports
  – Provide *trusted execution environment* for remote server
Trusted Key-value Stores with SGX

Throughput (Kop/s) vs Working set size (MB)

- Line 1: 계열1
- Line 2: 계열2
Trusted Key-value Stores with SGX

- Protected memory is limited to 128MB
  - Application can use about 92MB
Trusted Key-value Stores with SGX

- Protected memory is limited to 128MB
  - Application can use about 92MB
**Trusted Key-value Stores with SGX**

- Protected memory is limited to 128MB
  - Application can use about 92MB

Throughput (Kop/s)

- Working set size (MB)

- Protected Memory (EPC)

134X

128MB
Trusted Key-value Stores with SGX

- Protected memory is limited to 128MB
  - Application can use about 92MB
• Protected memory is limited to 128MB
  – Application can use about 92MB
Trusted Key-value Stores with SGX

- Protected memory is limited to 128MB
  - Application can use about 92MB
Trusted Key-value Stores with SGX

- Protected memory is limited to 128MB
  - Application can use about 92MB

---

Graph showing throughput (Kop/s) vs. working set size (MB) with two series labeled '계열1' and '계열2'. The graph indicates a significant drop in throughput as the working set size increases, with a notable 134X increase in throughput when using protected memory (EPC) compared to traditional paging.

Protected Memory (EPC)

Paging!
• Protected memory is limited to 128MB
  – Application can use about 92MB
Trusted Key-value Stores with SGX

- Protected memory is limited to 128MB
  - Application can use about 92MB
• Protected memory is limited to 128MB
  – Application can use about 92MB
Trusted Key-value Stores with SGX

• Protected memory is limited to 128MB
  – Application can use about 92MB
Throughput (Kop/s)

Working set size (MB)

- Protected memory is limited to 128MB
  - Application can use about 92MB
• Protected memory is limited to 128MB
  – Application can use about 92MB

How can we have efficient trusted key value stores?
Paging Mechanism of SGX

Access Page A

SGX enabled Processor

Core

TLB

Memory Encryption Engine

EPC

Encrypted Page A

DRAM

Enclave Context

Normal Context

Paging Mechanism of SGX

Enclave Context

Access Page A

SGX enabled Processor

Core

TLB

Memory Encryption Engine

Normal Context

Paging Mechanism of SGX

Access Page A

Enclave Context

Page table

SGX enabled Processor

Core

TLB

Memory Encryption Engine

Normal Context

Fault handler

EPC

Encrypted Page A

DRAM

Paging Mechanism of SGX

Enclave Context

Page table

Access Page A

SGX enabled Processor

Core

TLB

Memory Encryption Engine

Fault handler

Normal Context

Paging Mechanism of SGX

Enclave Context

Access Page A

Page table

SGX enabled Processor

Core

TLB

Memory Encryption Engine

Fault handler

Normal Context

Paging Mechanism of SGX

Paging Mechanism of SGX

Paging Mechanism of SGX

Paging Mechanism of SGX

Paging Mechanism of SGX

Paging Mechanism of SGX

Access Page A

Enclave Context

EEXIT

Normal Context

Fault handler

Page table

SGX enabled Processor

Core

TLB

Memory Encryption Engine

Encryption Integrity protection

EPC

Page A

Encrypted Page A

DRAM

Paging Mechanism of SGX

Paging Mechanism of SGX

Paging Mechanism of SGX

Access Page A

SGX enabled Processor

Core

TLB

Memory Encryption Engine

Encryption Integrity protection

Encrypted Page A

DRAM

EEXIT

EENTER

Fault handler

Page table

EPC

Page A

Normal Context

Enclave Context

Paging Mechanism of SGX

High overhead of crossing enclave boundary (~8000 cycles [1])

Paging Mechanism of SGX

Enclave Context

Access Page A

SGX enabled Processor

Core

TLB

Memory Encryption Engine

Encryption Integrity protection

High overhead of crossing enclave boundary (~8000 cycles [1])

EEXIT

EENTER

Observations

- Accessing untrusted memory incurs low overhead

![Graph showing latency per operation (ns) versus working set size (MB) for different contexts and data protection levels]

- Normal Context
- Enclave Context

Legend:
- NoSGX
- SGX_Enclave
- SGX_Unprotected
Accessing untrusted memory incurs low overhead
• Accessing untrusted memory incurs low overhead
• Accessing untrusted memory incurs low overhead
Observations

- Accessing untrusted memory incurs low overhead
Observations

- Accessing untrusted memory incurs low overhead
Observations

• Accessing untrusted memory incurs low overhead

Latency per operation (ns)
Working set size (MB)

- NoSGX
- SGX_Enclave
- SGX_Unprotected

Normal Context
Enclave Context

128MB
Processing buffer

Security Meta-data

Data protection
• Accessing untrusted memory incurs low overhead
Observations

- Accessing untrusted memory incurs low overhead

![Graph showing latency per operation vs. working set size]

- Observations

  - 128MB
  - Security Meta-data
  - Processing buffer
  - EPC
  - Data protection

![Diagram illustrating context and data protection]

- NoSGX
- SGX_Enclave
- SGX_Unprotected
Observations

- Accessing untrusted memory incurs low overhead

![Graph showing latency per operation vs working set size](image)

- No SGX Paging!
Observations

- Accessing untrusted memory incurs low overhead

![Graph showing latency per operation vs. working set size](image)

No SGX Paging!

![Diagram illustrating EPC, Data protection, and Enclave Context](image)
Observations

- Accessing untrusted memory incurs low overhead

![Graph showing latency per operation (ns) vs working set size (MB).]

- No SGX Paging!

![Diagram illustrating EPC, Security Meta-data, Data, and Enclave Context.]

- Data protection

![Thumb up emoji with text: No SGX Paging!]

Observations

• Accessing untrusted memory incurs low overhead

![Graph showing latency per operation (ns) vs. working set size (MB)]

- Accessing untrusted memory incurs low overhead.

![Diagram illustrating EPC, Security Meta-data, Data, and Processing buffer with No SGX Paging!]

- Observations:
  - 128MB EPC
  - Security Meta-data
  - Processing buffer
  - Data
  - No SGX Paging!
Observations

- Accessing untrusted memory incurs low overhead
Observations

- Accessing untrusted memory incurs low overhead

\[
\text{Latency per operation (ns)}
\]

Working set size (MB)

- **Reduce sgx-paging!**
  - Use protected memory as a secure processing buffer

No SGX Paging!
Observations

- Accessing untrusted memory incurs low overhead

- Reduce sgx-paging!
  - Use protected memory as a secure processing buffer

128MB Meta-data

No SGX Paging!
Observations

- Accessing untrusted memory incurs low overhead

![Latency per operation (ns) vs Working set size (MB)](chart.png)

- **Reduce sgx-paging!**
  - Use protected memory as a secure processing buffer

![Diagram of Protection Contexts](diagram.png)

- **No SGX Paging!**
Proposed Design: Semantic Aware Protection

Semantic aware protection

| Key | Value |

Access Object A

Enclave Context

Normal Context

EPC

Encrypted Object A

DRAM
Proposed Design: Semantic Aware Protection

Semantic aware protection

| Key | Value |

Enclave Context

Access Object A

Copy object

Normal Context

EPC

Encrypted Object A

DRAM
Proposed Design: Semantic Aware Protection

Semantic aware protection

| Key | Value |

Access Object A

Copy object

Encrypted Object A

DRAM

EPC

Enclave Context

Normal Context
Proposed Design: Semantic Aware Protection

Semantic aware protection

| Key | Value |

Enclave
Context

Access Object A

Copy object

Normal
Context

EPC

Object A

Encrypted Object A

DRAM
Proposed Design: Semantic Aware Protection

- **Enclave Context**
  - Access Object A
    - Copy object
      - Encryption
        - Integrity protection mechanism

- **Normal Context**
  - EPC
    - Encrypted Object A
      - DRAM

Semantic aware protection

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
</table>
Proposed Design: Semantic Aware Protection

Enclave Context

Access Object A

Copy object

Encryption Integrity protection mechanism

Semantic aware protection

Key

Value

EPC

Object A

Encrypted Object A

DRAM
Proposed Design: Semantic Aware Protection

Semantic aware protection

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
</table>

Enclave Context

Access Object A

Copy object

Encryption Integrity protection mechanism

Normal Context

EPC

Object A

Encrypted Object A

DRAM
Proposed Design: Semantic Aware Protection

Enclave Context

Access Object A

Copy object

Encryption Integrity protection mechanism

Semantic aware protection

Key

Value

EPC

Object A

Encrypted Object A

Normal Context

DRAM
Proposed Design: Semantic Aware Protection

Semantic aware protection

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
</table>

Access Object A

Copy object

Encryption

Integrity protection mechanism

No SGX Paging!
Proposed Design: Semantic Aware Protection

Semantic aware protection

| Key | Value |

Access Object A

Enclave Context

Copy object

Encryption Integrity protection mechanism

Fine-grained an efficient data protection!
Threat Model

• ShieldStore protects *confidentiality* and *integrity* of key/values

• Trusted Computing Base (TCB) of ShieldStore
  – SGX enabled Processor chip
  – Code & data in *enclave*

• Out of scope
  – Side channel attacks (ex. Foreshadow, controlled channel attacks)
  – Availability attacks
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region
- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries
Overall Design of ShieldStore

• Maintain small secure meta-data in trusted memory region

• Store main data structure on untrusted memory region
  – With encrypted and integrity-protected key-value entries

Client

Operations (Set/Get)

ShieldStore

Enclave

Data

Trusted

Untrusted
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries

**Challenge:** How to protect data efficiently?
Overall Design of ShieldStore

- Maintain small secure meta-data in trusted memory region

- Store main data structure on untrusted memory region
  - With encrypted and integrity-protected key-value entries

**Challenge:** How to protect data efficiently?
How To Protect Data?

Enclave

Non-Enclave

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC
How To Protect Data?

Enclave

Hash table pointer

Hash bucket

Non-Enclave

Encrypted key-value | Counter | MAC

Hash table pointer
How To Protect Data?

Hash table

pointer

Hash bucket

Encrypted
key-value

Counter

MAC

Enclave

Non-Enclave
How To Protect Data?

Enclave

Non-Enclave

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC
How To Protect Data?

Enclave

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC

Non-Enclave

Integrity protection
How To Protect Data?

- Enclave
  - Hash table
  - Hash bucket
  - Encrypted key-value
  - Counter
  - MAC

- Non-Enclave
  - Integrity protection
How To Protect Data?

Enclave

Non-Enclave

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC

Integrity protection
How To Protect Data?

Enclave

Non-Enclave

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC

Integrity protection
How To Protect Data?

Enclave

Non-Enclave

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC

Integrity protection
How To Protect Data?

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC

Integrity protection

Enclave

Non-Enclave
How To Protect Data?

Hash table pointer

Hash bucket

Encrypted key-value

Counter

MAC

Integrity protection

Enclave

Non-Enclave
How To Protect Data?

Hash table pointer → Hash bucket

Enclave

Non-Enclave

Integrity protection

Encrypted key-value → Counter → MAC
How To Protect Data?

- Enclave
- Non-Enclave

Hash table pointer

Merkle tree root nodes

Hash bucket

Encrypted key-value

Counter

MAC

Integrity protection
How To Protect Data?

Enclave

Non-Enclave

Integrity protection

Hash table pointer

Merkle tree root nodes

Hash bucket

Encrypted key-value

Counter

MAC
How To Protect Data?

- Enclave
  - Hash table pointer
  - Merkle tree root nodes

- Non-Enclave
  - Hash bucket
  - Encrypted key-value
  - Counter
  - MAC
  - CMAC function

Integrity protection
How To Protect Data?

Enclave

Hash table

pointer

Merkle tree

root nodes

Hash bucket

Encrypted

key-value

Counter

MAC

CMAC function

Non-Enclave

Integrity protection
How To Protect Data?

Enclave

Non-Enclave

Merkle tree
root nodes

Hash table
pointer

Hash bucket

Encrypted key-value

Counter

MAC

CMAC function

Integrity protection
How To Protect Data?

- Hash table pointer
- Merkle tree root nodes
- Hash bucket

- Encrypted key-value
- Counter
- MAC

- CMAC function

Enclave

Non-Enclave

Integrity protection
How To Protect Data?

Enclave

Non-Enclave

Integrity protection

Hash table
Hash bucket

Merkle tree root nodes

Encrypted key-value
Counter
MAC

CMAC function

114
How To Protect Data?

- Hash table pointer
- Merkle tree root nodes
- Hash bucket
- Encrypted key-value
- Counter
- MAC
- CMAC function

Enclave

Non-Enclave

Integrity protection
How To Protect Data?

- Enclave
- Non-Enclave

Hash table pointer
Merkle tree root nodes
Hash bucket
Encrypted key-value
Counter
MAC

AES encrypt function
CMAC function

Integrity protection
How To Protect Data?

- **Hash table pointer**
- **Merkle tree root nodes**
- **Hash bucket**
- **Encrypted key-value**
- **Counter**
- **MAC**

**Enclave**

**Non-Enclave**

**AES encrypt function**

**CMAC function**

**Integrity protection**
How To Protect Data?

- **Enclave**
  - Hash table pointer
  - Merkle tree root nodes
  - AES encrypt function
  - CMAC function

- **Non-Enclave**
  - Hash bucket
  - Encrypted key-value
  - Counter
  - MAC

- Integrity protection
How To Protect Data?

Enclave

Non-Enclave

Hash table pointer

Merkle tree root nodes

Hash bucket

Encrypted key-value

Counter

MAC

Data value

AES encrypt function

CMAC function

Integrity protection
How To Protect Data?

Enclave

Non-Enclave

Hash table

pointer

Merkle tree

root nodes

Hash bucket

Encrypted

key-value

Counter

MAC

Data value

AES encrypt

function

CMAC function

Integrity protection
How To Protect Data?

- Hash table pointer
- Merkle tree root nodes
- AES encrypt function
- CMAC function
- Enclave
- Non-Enclave
- Integrity protection
- Hash bucket
- Encrypted key-value
- Counter
- MAC
- Integrity protection
- Pointers

- AES encrypt function
- CMAC function
- Integ
How To Protect Data?

Enclave

Non-Enclave

Hash table pointer

Merkle tree root nodes

Hash bucket

Encrypted key-value

Counter

MAC

Data value

AES encrypt function

CMAC function

Integrity protection
How To Protect Data?

Hash table

Hash bucket

Merkle tree root nodes

Encrypted key-value

Counter

MAC

Data value

AES encrypt function

CMAC function

Enclave

Non-Enclave

Integrity protection
How To Protect Data?

Enclave
- Hash table pointer
- Merkle tree root nodes
  - Hash bucket
  - Hash value
  - MAC function
    - AES encrypt function
    - CMAC function

Non-Enclave
- Encrypted key-value
- Counter
- MAC
- Integrity protection
- Encryption/decryption
How To Protect Data?

- Hash table pointer
- Merkle tree root nodes
- Hash bucket
- Encrypted key-value
- Counter
- MAC
- Hash value
- MAC
- AES encrypt function
- CMAC function
- Data value
- Enclave
- Non-Enclave
- Integrity protection
- Encryption/decryption
How To Protect Data?

Enclave

- Hash table
- Merkle tree
  - root nodes
- Hash value

Non-Enclave

- Encrypted key-value
- Counter
- MAC

- AES encrypt function
- CMAC function

Integrity protection
Encryption/decryption
How To Protect Data?

Enclave

Non-Enclave

Hash table

Hash bucket

Merkle tree root nodes

Hash value

MAC

Encrypted key-value

Counter

Data value

AES encrypt function

CMAC function

Integrity protection

Encryption/decryption
How To Protect Data?

- Hash table pointer
- Merkle tree root nodes
- Hash bucket
- Encrypted key-value
- Counter
- MAC
- Hash value
- Data value
- AES encrypt function
- CMAC function

Enclave

Non-Enclave

Integrity protection

Encryption/decryption
How To Protect Data?

- Enclave
  - Hash table
  - Merkle tree root nodes
  - AES encrypt function
- Non-Enclave
  - Hash bucket
  - Encrypted key-value
  - Counter
  - MAC
  - Integrity protection
  - Encryption/decryption
How To Protect Data?

Enclave

Non-Enclave

Integrity protection
Encryption/decryption

Hash table pointer
Merkle tree root nodes
Hash bucket
Encrypted key-value
Counter
MAC

Hash value
MAC

Data value

AES encrypt function

CMAC function

Encrypted

Encryption/decryption
How To Protect Data?

- Hash table pointer
- Merkle tree root nodes
- Enclave
- Non-Enclave
- Hash bucket
- Encrypted key-value
- Counter
- MAC
- Hash value
- Data value
- AES encrypt function
- CMAC function

Integrity protection
Encryption/decryption
How To Protect Data?

Enclave

- Hash table
  - Hash table pointer
  - Merkle tree root nodes

- Hash bucket
  - Hash value
  - MAC

- Encrypted key-value
- Counter

Non-Enclave

- Data value
  - AES encrypt function
  - CMAC function

- Integrity protection
  - Encryption/decryption

- Enclave
  - Non-Enclave
How To Protect Data?

Enclave

- Hash table
- Merkle tree root nodes
- Hash bucket
- Encrypted key-value
- Counter
- MAC
- MAC function
- AES encrypt function
- Hash value

Non-Enclave

- Integrity protection
- Encryption/decryption

Data value
How To Protect Data?

Enclave

Non-Enclave

Hash table

pointer

Merkle tree

root nodes

Hash bucket

Hash value

Encrypted

key-value

Counter

MAC

Data value

AES encrypt

function

CMAC function

Integrity protection

Encryption/decryption
How To Protect Data?

Enclave

- Hash table pointer
- Merkle tree root nodes
- Hash bucket
- AES encrypt function
- CMAC function

Non-Enclave

- Encrypted key-value
- Counter
- Hash value
- MAC
- Integrity protection
- Encryption/decryption
Integrity Protection

• ShieldStore employs *Merkle Tree* mechanism
  – Exploits the hash-based index structure to verify integrity efficiently
Integrity Protection

- ShieldStore employs **Merkle Tree** mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

![Diagram of Traditional Merkle Tree and Data Structure Aware Merkle Tree](image)

**Traditional Merkle Tree**

- Hash tree nodes
- Data

**Data structure aware Merkle Tree**

- **Enclave**
  - Reduce the depth of tree
  - Keep subtree root node on *enclave*
- **Non-Enclave**
  - Traverse all the MAC entries
Integrity Protection

• ShieldStore employs *Merkle Tree* mechanism
  – Exploits the hash-based index structure to verify integrity efficiently

![Diagram of Traditional Merkle Tree and Data structure aware Merkle Tree]

- Reduce the depth of tree
- Keep subtree root node on *enclave*
- Traverse all the MAC entries
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

**Traditional Merkle Tree**

**Data structure aware Merkle Tree**

- Reduce the depth of tree
- Keep subtree root node on *enclave*
- Traverse all the MAC entries
Integrity Protection

- ShieldStore employs **Merkle Tree** mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

**Traditional Merkle Tree**

- Hash tree nodes

**Data structure aware Merkle Tree**

- CMAC key
- CMAC function
- Bucket

**Enclave**
- Reduce the depth of tree
- Keep subtree root node on *enclave*

**Non-Enclave**
- Traverse all the MAC entries
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

**Traditional Merkle Tree**

- Hash tree nodes
- Enclave vs. Non-Enclave

**Data structure aware Merkle Tree**

- CMAC key
- CMAC function
- Bucket

+ Reduce the depth of tree
+ Keep subtree root node on *enclave*
- Traverse all the MAC entries
Integrity Protection

- ShieldStore employs *Merkle Tree* mechanism
  - Exploits the hash-based index structure to verify integrity efficiently

**Traditional Merkle Tree**

- Hash tree nodes
- Data

**Data structure aware Merkle Tree**

- CMAC key
- CMAC function
- Bucket
- Data

**Enclave**

- Reduce the depth of tree
- Keep subtree root node on *enclave*

**Non-Enclave**

- Traverse all the MAC entries
Encryption

• ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
</table>

**Enclave**

- Concatenate

**Encryption key**

| Non-Enclave |  

Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

Key | Value | CTR

- Concatenate

- AES-CTR encryption

- Encryption key

- Enclave

- Non-Enclave
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket

Key | Value | CTR
--- | --- | ---
Concatenate

Enclave

AES-CTR encryption
Encryption key

Encrypted key-value

Non-Enclave
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket

![Diagram showing encryption process]
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information
+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
• ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket
ShieldStore encrypts both key and value of the entry

- Alleviate the leakage of information

+ Reduce information leaks

- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

- Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information
+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket

```
+-----------------+       +-------------------+
| Key             | Value | CTR               |
+-----------------+       +-------------------+
    |              |       |                   |
    |   Enclave    |       |                   |
    |              |       |                   |
| Encrypt key-value |      |                   |
|                  |      |                   |
|                  |      |                   |
|                  |      |                   |
|                  |      |                   |
+-----------------+       +-------------------+
    |              |       |                   |
    |   Non-Enclave|       |                   |
    |              |       |                   |
| Encrypt key-value |      |                   |
|                  |      |                   |
|                  |      |                   |
|                  |      |                   |
|                  |      |                   |
+-----------------+       +-------------------+

Concatenate

AES-CTR encryption

Encryption key

Matching entry

Enclave

Non-Enclave
```
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

  + Reduce information leaks
  - Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
ShieldStore encrypts both key and value of the entry
- Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

• ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks
  - Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket

![Diagram showing encryption process]

Key | Value | CTR
---|---|---

Enclave

Non-Enclave

Matching entry

Encrypted key-value

Encrypted key-value

AES-CTR encryption

Encryption key

Encrypt all the keys in a same bucket
Encryption

• ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
• ShieldStore encrypts both key and value of the entry
  – Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket

![Diagram of encryption process]
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
ShieldStore encrypts both key and value of the entry
- Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket

Key | Value | CTR
---|---|---
Concatenate

AES-CTR encryption
Encryption key

Matching entry
Decryption

Matching?

Encrypted key-value
CTR

Encrypted key-value
CTR

Matching entry

Encrypted key-value
CTR

Encrypted key-value
CTR

Non-Enclave
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket

\[
\begin{array}{|c|c|c|}
\hline
\text{Key} & \text{Value} & \text{CTR} \\
\hline
\end{array}
\]

Concatenate

AES-CTR encryption

Matching entry

Decryption

Matching?
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
- Decrypt all the keys in a same bucket
Encryption

- ShieldStore encrypts both key and value of the entry
  - Alleviate the leakage of information

+ Reduce information leaks
  - Decrypt all the keys in a same bucket

```
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>CTR</th>
</tr>
</thead>
</table>
```

- Concatenate

Enclave

```
Encrypt key-value
```

- AES-CTR encryption

```
Encryption key
```

Non-Enclave

```
Encrypted key-value
```

```
CTR
```

```
Matching entry
```

Enclave

```
Matching?
```

Decryption

```
Key
```

```
Encrypted key-value
```

```
CTR
```

Non-Enclave
Optimizations

• MAC bucketing
  – Maintain the MAC buffer per a hash bucket

• Searching encrypted key
• Custom heap allocator
• Optimization for multi-threading
Optimizations

- MAC bucketing
  - Maintain the MAC buffer per a hash bucket

- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading
Optimizations

- MAC bucketing
  - Maintain the MAC buffer per a hash bucket

- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading
Optimizations

- MAC bucketing
  - Maintain the MAC buffer per a hash bucket

- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading
Optimizations

• MAC bucketing
  – Maintain the MAC buffer per a hash bucket

• Searching encrypted key
• Custom heap allocator
• Optimization for multi-threading
Optimizations

- MAC bucketing
  - Maintain the MAC buffer per a hash bucket

- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading
Optimizations

- **MAC bucketing**
  - Maintain the MAC buffer per a hash bucket

- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading
Optimizations

- **MAC bucketing**
  - Maintain the MAC buffer per a hash bucket

- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading
Optimizations

• MAC bucketing
  – Maintain the MAC buffer per a hash bucket

• Searching encrypted key
• Custom heap allocator
• Optimization for multi-threading
Optimizations

- **MAC bucketing**
  - Maintain the MAC buffer per a hash bucket

- Searching encrypted key
- Custom heap allocator
- Optimization for multi-threading
Optimizations

• MAC bucketing
  – Maintain the MAC buffer per a hash bucket

• Searching encrypted key
• Custom heap allocator
• Optimization for multi-threading
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks

![Diagram showing non-volatile memory, monotonic counter, and storage]

- Non-volatile Memory
- Monotonic Counter
  - 00
- Enclave
  - Security meta-data
- Non-Enclave
  - Encrypted key-value entries

Storage: HDD/SSD
Persistent Support

• Intel SGX supports *sealing mechanism*
  – Using *monotonic counter* stored in non-volatile memory
  – Protect data from rollback attacks

![Diagram showing non-volatile memory, monotonic counter, enclave, non-enclave, security meta-data, encrypted key-value entries, and storage: HDD/SSD]
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks
Persistent Support

• Intel SGX supports *sealing mechanism*
  – Using *monotonic counter* stored in non-volatile memory
  – Protect data from rollback attacks
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks

![Diagram showing sealing mechanism and storage options]

- Non-volatile Memory
  - Monotonic Counter
  - SGX Sealing
  - Storage: HDD/SSD

- Enclave
  - Security meta-data

- Non-Enclave
  - Encrypted key-value entries
Persistent Support

- Intel SGX supports *sealing mechanism*
  - Using *monotonic counter* stored in non-volatile memory
  - Protect data from rollback attacks

![Diagram showing the relationship between non-volatile memory, monotonic counter, enclave, non-enclave, security meta-data, encrypted key-value entries, SGX sealing, and storage (HDD/SSD).]
Persistent Support

• Intel SGX supports *sealing mechanism*
  – Using *monotonic counter* stored in non-volatile memory
  – Protect data from rollback attacks

![Diagram showing the sealing mechanism and its components]

- Non-volatile Memory
  - Monotonic Counter
    - Current value: 08
  - Security meta-data
  - Enclave
  - Non-Enclave
    - Encrypted key-value entries

SGX Sealing

Storage: HDD/SSD

- Physical storage solution for secure data persistence
Experimental Setup

• Evaluation
  – Standalone: Focus on data store aspect without network
  – Network: Socket interface with a 10Gb NIC and 256 concurrent clients

• Metrics
  – *Secure Memcached*: memcached with grapheneSGX [2]
  – *ShieldBase*: ShieldStore without optimizations
  – *ShieldOPT*: ShieldStore with optimizations

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Key Size(B)</th>
<th>Value Size(B)</th>
<th>Working set(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>16</td>
<td>16</td>
<td>305</td>
</tr>
<tr>
<td>Medium</td>
<td>16</td>
<td>128</td>
<td>1,373</td>
</tr>
<tr>
<td>Large</td>
<td>16</td>
<td>512</td>
<td>5,035</td>
</tr>
</tbody>
</table>

Standalone Evaluation

• ShieldStore performs
  – 7 – 8 times better than *Secure Memcached* on 1 thread
  – 24 – 27 times better than *Secure Memcached* on 4 threads
• ShieldStore performs
  – 7 – 8 times better than Secure Memcached on 1 thread
  – 24 – 27 times better than Secure Memcached on 4 threads
• ShieldStore performs
  – 7 – 8 times better than *Secure Memcached* on 1 thread
  – 24 – 27 times better than *Secure Memcached* on 4 threads
• ShieldStore performs
  – 7 – 8 times better than *Secure Memcached* on 1 thread
  – 24 – 27 times better than *Secure Memcached* on 4 threads
• ShieldStore performs
  – 7 – 8 times better than Secure Memcached on 1 thread
  – 24 – 27 times better than Secure Memcached on 4 threads
Comparison to Key-value Store on Eleos [1/2]

• Eleos provides coarse-grained user space memory paging
  – Eleos provides 1KB/4KB page-grained protection
  – ShieldStore provides fine-grained data protection
Comparison to Key-value Store on Eleos [1/2]

- Eleos provides coarse-grained user space memory paging
  - Eleos provides 1KB/4KB page-grained protection
  - ShieldStore provides fine-grained data protection
Comparison to Key-value Store on Eleos [1/2]

- Eleos provides coarse-grained user space memory paging
  - Eleos provides 1KB/4KB page-grained protection
  - ShieldStore provides fine-grained data protection

![Exitless paging!](image)
Eleos provides coarse-grained user space memory paging
  – Eleos provides 1KB/4KB page-grained protection
  – ShieldStore provides fine-grained data protection
Comparison to Key-value Store on Eleos [2/2]

- ShieldStore performs better than Eleos even with 4KB value
  - Efficient data protection improves the performance of ShieldStore
• ShieldStore performs better than Eleos even with 4KB value
  – Efficient data protection improves the performance of ShieldStore
Network Evaluation

• ShieldStore with HotCalls [3] performs
  – 6 – 11 times better than *Secure Memcached* on 1 thread and 4 threads
  – 3 – 4 times slower than *Insecure Memcached* on 1 thread and 4 threads

Network Evaluation

- ShieldStore with HotCalls [3] performs
  - 6–11 times better than *Secure Memcached* on 1 thread and 4 threads
  - 3–4 times slower than *Insecure Memcached* on 1 thread and 4 threads

Summary of Paging Principles

- SGX paging (*Secure Memcached*)
- User space paging (*Eleos*)
- Semantic aware protection (*ShieldStore*)
Summary of Paging Principles

- **SGX paging** (*Secure Memcached*)
- **User space paging** (*Eleos*)
- **Semantic aware protection** (*ShieldStore*)

- Page
  - Fixed/coarse-grained
  - Variable/fine-grained

- More efficient
Summary of Paging Principles

- **SGX paging** *(Secure Memcached)*
- **User space paging** *(Eleos)*
- **Semantic aware protection** *(ShieldStore)*

- **Page**
  - Fixed/coarse-grained

- **Page/Sub page**
  - Variable/fine-grained

More efficient
Summary of Paging Principles

- SGX paging *(Secure Memcached)*
- User space paging *(Eleos)*
- Semantic aware protection *(ShieldStore)*

<table>
<thead>
<tr>
<th>Page</th>
<th>Page/Sub page</th>
<th>Key-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed/coarse-grained</td>
<td>Variable/fine-grained</td>
<td></td>
</tr>
</tbody>
</table>

More efficient
Summary of Paging Principles

- **Security Granularity**
  - SGX paging *(Secure Memcached)*
  - User space paging *(Eleos)*
  - Semantic aware protection *(ShieldStore)*

- **Page**
  - SGX paging: Fixed/coarse-grained
  - User space paging: Variable/fine-grained
  - Semantic aware protection: Key-value

- **More efficient**

- **Performance**
Summary of Paging Principles

- **SGX paging** *(Secure Memcached)*
  - Security: Page
  - Granularity: Fixed/coarse-grained

- **User space paging** *(Eleos)*
  - Security: Page/Sub page
  - Granularity: Variable/fine-grained

- **Semantic aware protection** *(ShieldStore)*
  - Security: Key-value

More efficient

Generality

Performance
Summary of Paging Principles

- SGX paging *(Secure Memcached)* - Fixed/coarse-grained
- User space paging *(Eleos)* - Variable/fine-grained
- Semantic aware protection *(ShieldStore)* - Key-value

Security

Granularity

Page
Page/Sub page
Key-value

Fixed/coarse-grained
Variable/fine-grained

More efficient

Generality
Performance
Summary of Paging Principles

- **SGX paging** *(Secure Memcached)*: Fixed/coarse-grained
- **User space paging** *(Eleos)*: Variable/fine-grained
- **Semantic aware protection** *(ShieldStore)*: Key-value

**Security**
- Page: SGX paging
- Page/Sub page: User space paging
- Key-value: Semantic aware protection

**Granularity**
- Fixed/coarse-grained: SGX paging
- Variable/fine-grained: User space paging

**Crossing Enclave Overhead**
- More efficient

**Generality vs. Performance**
- Generality: SGX paging
- Performance: User space paging

*User space paging* *(Eleos)*

*Semantic aware protection* *(ShieldStore)*

*SGX paging* *(Secure Memcached)*
Summary of Paging Principles

- SGX paging (*Secure Memcached*): Page/Secured
- User space paging (*Eleos*): Page/Sub page
- Semantic aware protection (*ShieldStore*): Key-value

- **Security**
  - SGX paging
  - User space paging
  - Semantic aware protection

- **Granularity**
  - Fixed/coarse-grained
  - Variable/fine-grained

- **Crossing Enclave Overhead**
  - High
  - More efficient

- **Generality vs. Performance**
  - Generality
  - Performance
Summary of Paging Principles

- **SGX paging** *(Secure Memcached)*
  - Security: Page
  - Granularity: Fixed/coarse-grained
  - Crossing Enclave: High
  - Enclave Overhead: Small
  - More efficient

- **User space paging** *(Eleos)*
  - Security: Page/Sub page
  - Granularity: Variable/fine-grained
  - Crossing Enclave: High
  - Enclave Overhead: Small
  - More efficient

- **Semantic aware protection** *(ShieldStore)*
  - Security: Key-value
  - Granularity: Variable/fine-grained
  - Crossing Enclave: High
  - Enclave Overhead: Small
  - More efficient
Summary of Paging Principles

- **SGX paging** *(Secure Memcached)*
- **User space paging** *(Eleos)*
- **Semantic aware protection** *(ShieldStore)*

<table>
<thead>
<tr>
<th>Security</th>
<th>Page</th>
<th>Page/Sub page</th>
<th>Key-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granularity</td>
<td>Fixed/coarse-grained</td>
<td>Variable/fine-grained</td>
<td></td>
</tr>
</tbody>
</table>

- Crossing Enclave Overhead:
  - High
  - Small

- More efficient

Generality → Performance
Summary of Paging Principles

- **SGX paging** (*Secure Memcached*): Page, High, Fixed/coarse-grained, More efficient, Performance
- **User space paging** (*Eleos*): Page/Sub page, Small, Variable/fine-grained, Generality
- **Semantic aware protection** (*ShieldStore*): Key-value, Small, More efficient

**Security**
- SGX paging, User space paging, Semantic aware protection

**Granularity**
- Fixed/coarse-grained, Variable/fine-grained

**Crossing Enclave Overhead**
- High, Small

**Performance vs. Generality**
- Performance
- Generality
Summary of Paging Principles

- **SGX paging** *(Secure Memcached)*
- **User space paging** *(Eleos)*
- **Semantic aware protection** *(ShieldStore)*

<table>
<thead>
<tr>
<th>Security</th>
<th>Granularity</th>
<th>Crossing</th>
<th>Enclave</th>
<th>Overhead</th>
<th>Page</th>
<th>Page/Sub page</th>
<th>Key-value</th>
<th>More efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX paging</td>
<td>Fixed/coarse-grained</td>
<td>High</td>
<td>Small</td>
<td>More efficient</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User space paging</td>
<td>Variable/fine-grained</td>
<td>Small</td>
<td>Small</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semantic aware protection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generality vs. Performance
Summary of Paging Principles

- **SGX paging** *(Secure Memcached)*
- **User space paging** *(Eleos)*
- **Semantic aware protection** *(ShieldStore)*

**Security**
- Page
- Page/Sub page
- Key-value

**Granularity**
- Fixed/coarse-grained
- Variable/fine-grained

**Crossing Enclave Overhead**
- High
- Small
- More efficient

**Generality vs. Performance**
- More efficient vs. Small

---

**Semantic aware protection** *(ShieldStore)*
Summary of Paging Principles

- **SGX paging** *(Secure Memcached)*
- **User space paging** *(Eleos)*
- **Semantic aware protection** *(ShieldStore)*

<table>
<thead>
<tr>
<th>Security</th>
<th>Granularity</th>
<th>Page</th>
<th>Page/Sub page</th>
<th>Key-value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed/coarse-grained</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Variable/fine-grained</td>
<td></td>
<td></td>
<td>Key-value</td>
</tr>
</tbody>
</table>

- High
- Small
- Small

More efficient

**ShieldStore code will be available at**
https://github.com/cocoppang/ShieldStore
ShieldStore: Shielded In-memory Key-value Storage with SGX

Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh

Dresden, Germany
March 25-27, 2019