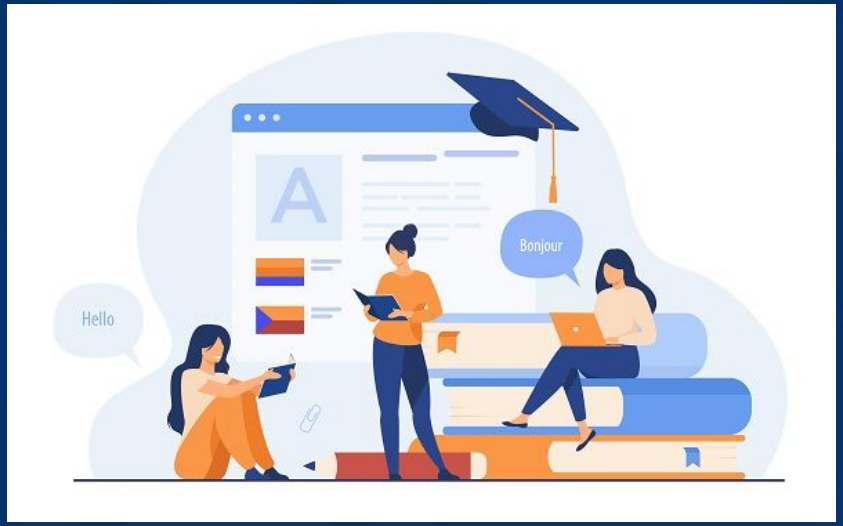
Research Paper Presentation

CS5322 Grp15

Name	Matric
Chun Hong Wei	A0167886E
Chew Zheng Xiong	A0167003R
Ng Jong Ray, Edward	A0216695U
Cheng Xianhao	A0167000X
Li YingHui	A0112832R



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

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

EuroSys '19, March 25–28, 2019, Dresden, Germany

1 Introduction

What is SGX?

Trusted Execution Environments & SGX

- **Intel's SGX & PRM:**
  - Provides a hardware-secured space using Processor Reserved Memory (PRM).
  - PRM prevent access to other software with CPU-blocked Direct Memory Access.
- **Enclave & EPC:**
  - Enclaves are encrypted, integrity-secured memory units called the Enclave Page Cache (EPC).
- **Protection & Application:**
  - Shields code from remote attackers, untrusted OS, and hardware threats.
  - Enables secure app execution on remote cloud servers.

What are Key Value Stores?

The Popularity of Key-Value Stores

- **Key-value stores** -> e.g. memcached and Redis -> widely used non-relational databases.
- Primarily retain key-value pairs **in main memory for fast data access**.
- **Logging to persistent storage** -> **durability**.
- Basic operations involve setting and getting key-value pairs.

2

Research Background

Challenges and Overheads of Intel SGX Memory Management

EPC's Memory Limitation

- Size Limit (128MB)
- Demand Paging in EPC

Performance degradation with exceeding EPC limit

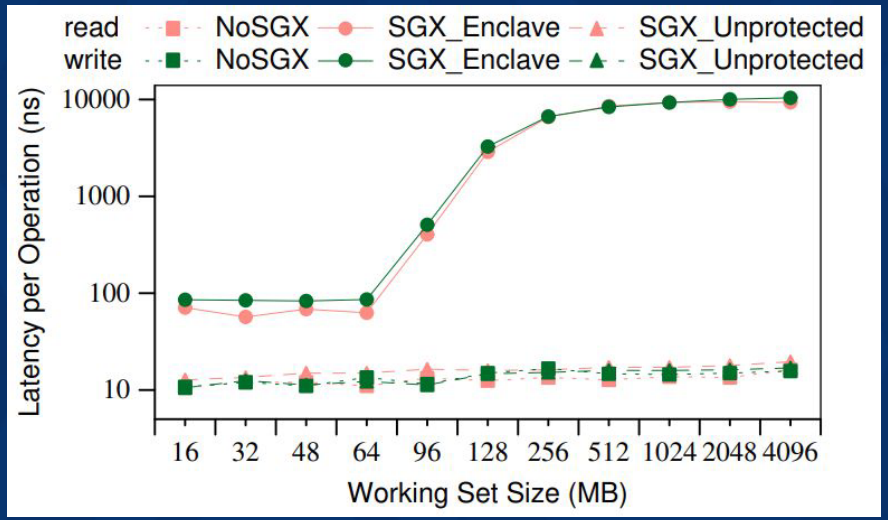
- Latency increases upon reaching effective EPC capacity

Enclave Boundary Crossing Overheads

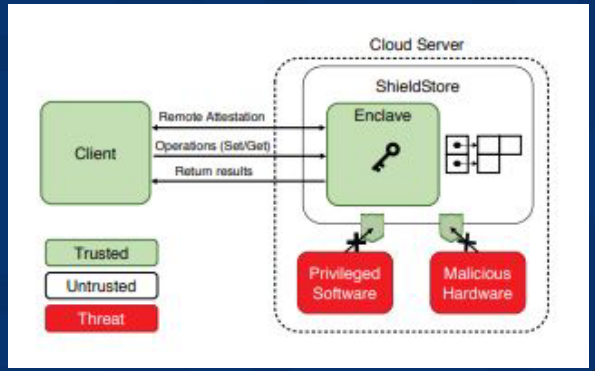
- Enclave Restrictions and Overhead
- Efforts to Reduce Overhead

The Caveat

- Accessing unprotected memory from enclave incurs insignificant overhead
  - Performance is comparable to memory access from normal context
- Potential solution to limited EPC memory and high overhead of demand paging



Introduction of ShieldStore



ShieldStore's Unique Approach

- Secure key-value store design without relying on the limited EPC memory.
- Predominantly uses non-enclave memory.
- More efficient memory usage and performance enhancements.

Benefits of ShieldStore

- Reduce unnecessary memory encryption overheads and improve execution performance.
- Efficient key-value store that operates on an enclave.

3 Creating a Baseline

ShieldStore: Design, Operation and Security in the SGX Ecosystem (1)

Baseline Key-Value Store on SGX

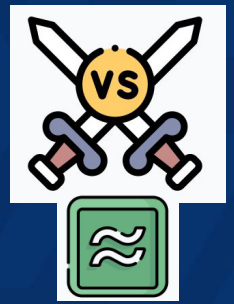
- The paper employs a hash-based index structure with chaining to manage hash collisions.
- Hash-based indices offer speed and simplicity but struggle with ranged searches. In contrast, tree-based indices allow ranged retrievals but at increased complexities.
- Benchmarking against memcached showed similar performance for the baseline design.
- A naive adaptation of key-value storage to SGX, especially with databases much larger than the EPC, demonstrates significantly reduced performance, motivating the need for optimized solutions like ShieldStore.

Paper Baseline
Key-Value Store on SGX

- Create naive adaptation of key-value storage to SGX employing a hash-based index structure with chaining.
- Hash-based indices
  - speed and simplicity
  - struggle with ranged searches.
- Tree-based indices
  - allow ranged retrievals
  - increased complexities.

validate maturity

BASELINE DESIGN (Benchmark)



Memcached

ShieldStore: Design, Operation and Security in the SGX Ecosystem (2)

Server-side vs. Client-side Computation for Key Value

- The paper proposes a server-side encryption model with SGX for key-value storage, contrasting it with the client-side encryption alternative.
- Server-side encryption offers more intricate operations on stored data, e.g., increment or append functions.
- The client-side model is restrictive, allowing only one fixed client to use data. It also demands multiple clients coordinate to exchange security keys and metadata.
- In the server-side model, post remote attestation and session key exchange, clients send requests via a secure channel. The server processes these requests and responds securely without clients needing to access ciphertexts or be aware of secret server keys.

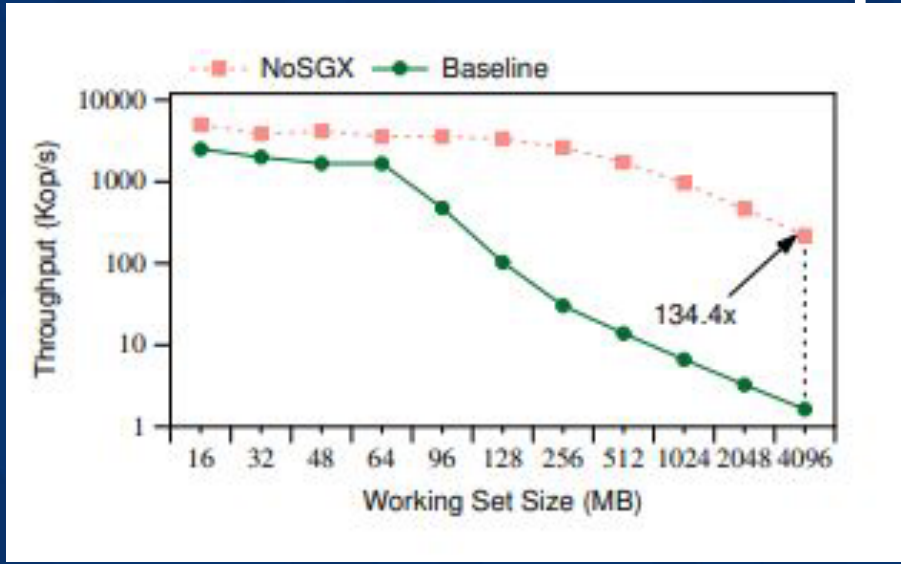
Paper Proposal:

Server-side Computation with SGX for key-value storage

- Server-side encryption
  - Offers more intricate operations on stored data, e.g., increment or append functions.
- Client-side model is restrictive
  - Allows only one fixed client to use data
  - Demands multiple clients coordinate to exchange security keys and metadata.
- Server-side model (post remote attestation and session key exchange)
  - Clients send requests -> secure channel
  - Server responses securely without clients needing to access ciphertexts or be aware of secret server keys.

Performance Analysis of Baseline Adaptation

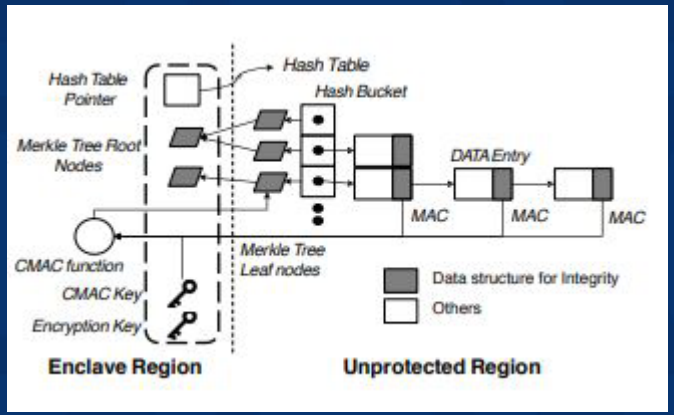
- Minor impact for database size < 64MB.
- Dramatic throughput drop as size increases:
  - Beyond 96MB sees significant decline.
  - 4GB database: 134 times slower.
- Databases much larger than the EPC will have significantly reduced performance.



ShieldStore: Design, Operation and Security in the SGX Ecosystem (3)

Threat Model for ShieldStore

- The Trusted Computing Base (TCB) of ShieldStore comprises the processor chip and the ShieldStore code in an enclave, making it resilient against certain malicious privileged and physical attacks.
- ShieldStore's design can counter threats from compromised OS or physical attacks by malicious actors.
- However, current SGX is susceptible to side-channel attacks, with demonstrated vulnerabilities like Foreshadow. ShieldStore's susceptibility to these SGX vulnerabilities is explored further in another section.
- ShieldStore does not account for availability attacks and assumes encrypted communication between clients and itself.



Threat Model

- Hardware - DRAM / Page Tables
  - Hardware attackers
  - Attack vectors - OS attack / Physical Attack / Side-Channel

- The Trusted Computing Base (TCB) of ShieldStore comprises the processor chip and the ShieldStore code in an enclave, making it resilient against certain malicious privileged and physical attacks.
- ShieldStore's design can counter threats from compromised OS.
- ShieldStore's design can counter threats from physical attacks.
- However, current SGX is susceptible to side-channel attacks, with demonstrated vulnerabilities like Foreshadow. ShieldStore's susceptibility to these SGX vulnerabilities is explored further in another section.
- ShieldStore does not account for availability attacks and assumes encrypted communication between clients and itself.

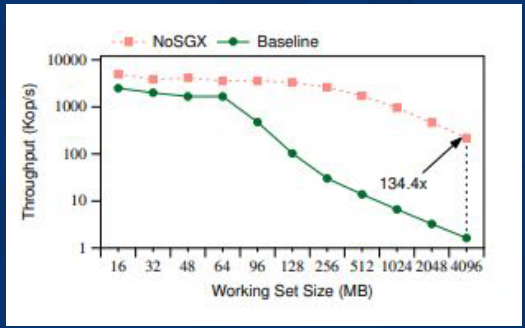
Shieldstore: Addressing SGX Challenges in Key-Value Stores

Performance Impact of SGX on Key-Value Stores

Naive key-value store adaptation to SGX shows degradation.
Minor impact for database size < 64MB.
Dramatic throughput drop as size increases:
Beyond 96MB sees significant decline.
4GB database: 134 times slower than insecure version.

Performance Improvements with ShieldStore

- Addresses SGX performance bottlenecks.
- Fine-grained SGX-based data protection.
- Efficient and secure key-value store design.



Data Organization in ShieldStore

- Dual hash tables: Enclave region & Unprotected region.
- Integrity ensured with Merkle Tree in enclave:
- Leaf nodes point to data entries.
- Each hash bucket data entry has associated MAC.
- Uses CMAC function with keys for added security.

4 Design

ShieldStore Design: Architecture, Encryption, and Persistency in SGX (1)

Overall Architecture

- Utilizes a hash-based index structure.
- Main hash table is placed in unprotected memory.
- Set operations encrypt each data entry within an enclave.
- Get operations retrieve and decrypt data from the main hash table.
- Avoids reliance on EPC paging, introduces fine-grained key-value protection.

Fine-grained Key-value Encryption

- AES-CTR counter mode encryption used for key-value pairs.
- Each entry includes key-index, encrypted key-value, IV/counter, MAC, and key/value sizes.
- Keyed-hash function hides actual key distributions.
- IV/Counter managed in combination, incremented per update.
- MAC hashing ensures data integrity in unprotected regions.

ShieldStore Design: Architecture, Encryption, and Persistency in SGX (2)

Integrity Verification with ShieldStore

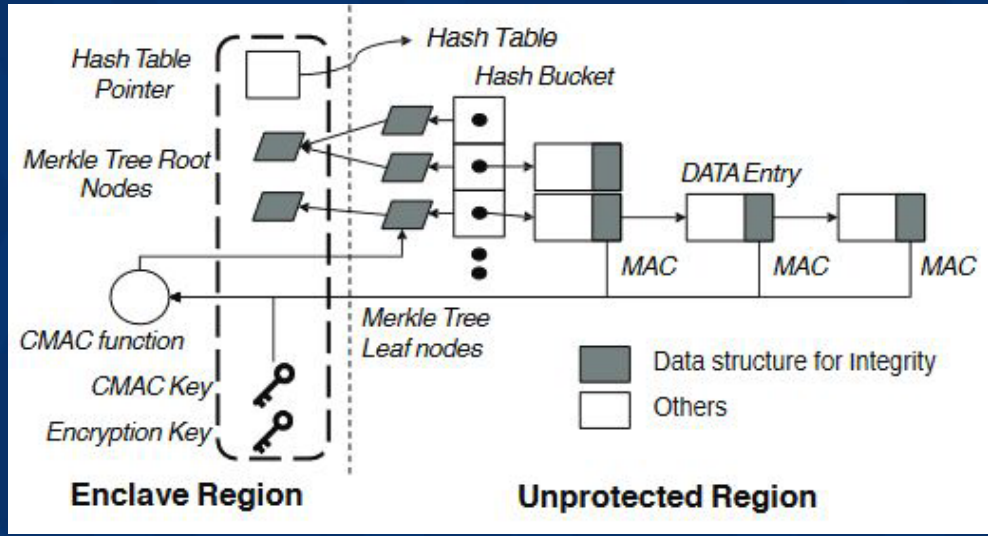
- Integrity derived from Merkle tree design.
- In-enclave hashes created, stored in enclave memory.
- Each MAC hash covers one or more hash table buckets.
- Flattened hashes reduce tall Merkle tree overheads.
- Decision on number of MAC hashes and buckets crucial for performance optimization.

Persistency Support

- Uses Intel SGX sealing mechanism for persistence.
- Snapshot mechanism, similar to Redis, is employed.
- Snapshotting initiated every 60 seconds.
- Parallelizes processing requests and storing key-value entries.
- Uses monotonic counters from SGX to counter rollback attacks.

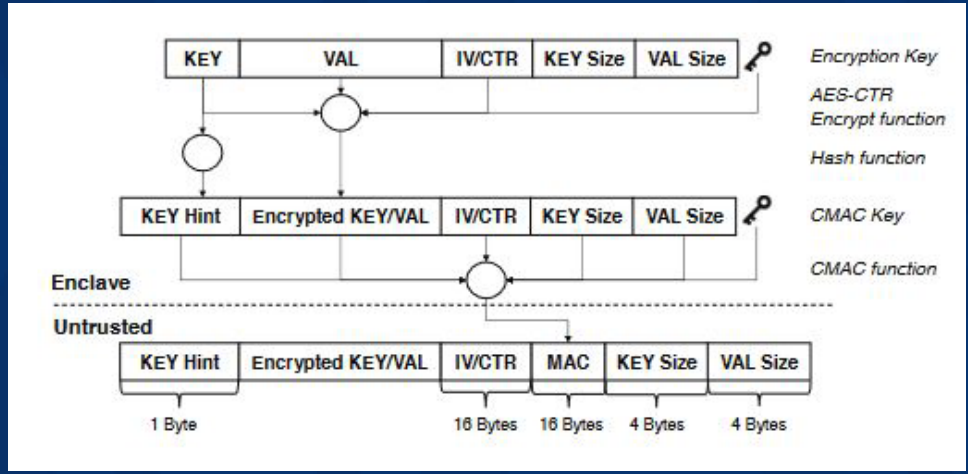
ShieldStore Design (1)
Overall Architecture

- Utilizes a hash-based index structure.
- ShieldStore Code running in the enclave.
  - Set operations encrypt and tag each data entry.
  - Get operations retrieve, decrypt and verify integrity of data from main hash table.
- Avoids reliance on EPC paging, introduces fine-grained key-value protection.



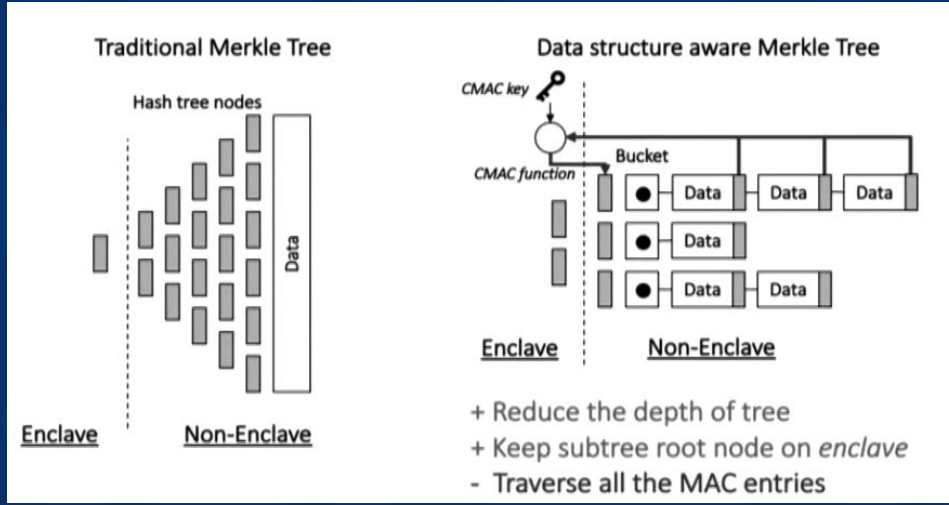
ShieldStore Design (2)
Fine-grained Key-value Encryption

- AES-CTR counter mode encryption used for key-value pairs.
- Each entry includes key-index, encrypted key-value, IV/counter, MAC, and key/value sizes.
- Keyed-hash function hides actual key distributions.
- IV/Counter managed in combination, incremented per update.



ShieldStore Design (3)
Integrity Verification with ShieldStore

- Integrity derived from Merkle tree design.
- Key Consideration: Number of MAC hashes and buckets crucial for performance optimization.
- Advantage: Merkle tree height can be adjusted depending on the number of key-value pairs.
    - Each MAC hash covers one or more hash table buckets.



ShieldStore Design (4)
Persistency Support

- Uses Intel SGX sealing mechanism for persistence with monotonic counters from SGX to counter rollback attacks.
- Parallelise processing requests and storing key-value entries.
- Advantage: Majority of the key-value data are already encrypted and integrity-protected, improving efficiency.

Algorithm 1 Optimized persistency support

```
1:  procedure PERSISTENT
2:      stop process of incoming requests
3:      seal meta-data on EPC
4:      fork the key-value store process
5:      if isParent? then
6:          write the sealed meta-data to storage
7:          store incoming requests in a temporary table
8:      else
9:          write encrypted key-value entries to storage
10:     if isChildDone? then
11:         update the main table with the temporary table
```

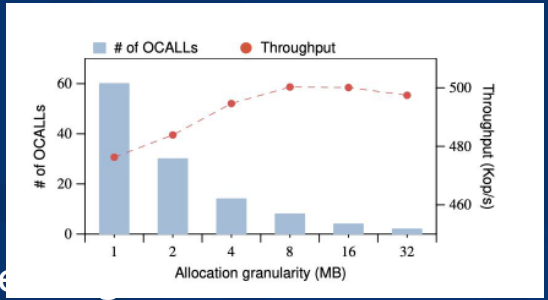
5 Optimisations In ShieldStore

Proposed Optimisation

- Extra Heap Allocator
- MAC Bucketing
- Multi-Threading
- Searching Encrypted Key

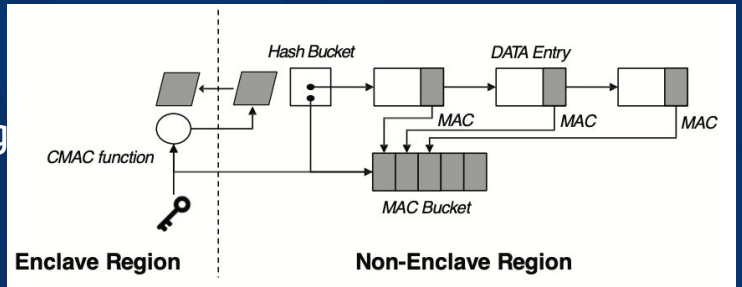
Extra Heap Allocator

- **Background**: Two Heap Allocators
  - Inside enclave for enclave memory.
  - Outside enclave for untrusted memory
- **Issue**
  - Costly exits from enclave for outside heap allocator calls.
- **Solution**
  - Custom memory allocator inside enclave for unprotected memory allocation.
- **Performance**
  - Drastic reduction in OCALL operations with incre
    chunk size for sbrk call - 16MB chunk size



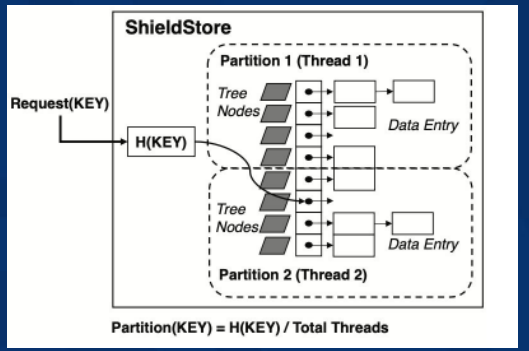
MAC Bucketing

- **Background:**
  - Pointer traversal of data entries to retrieve MACs value
- **Issue**
  - Overhead in accessing MAC values for integrity checking.
- **Solution**:
  - Dedicated MAC bucket for each hash bucket
  - MAC bucket contains only MAC fields of data entries.
  - If chain length exceeds > 30 MACs, chain of MAC buckets
- **Performance**
  - Reduce time taken for MACs reading



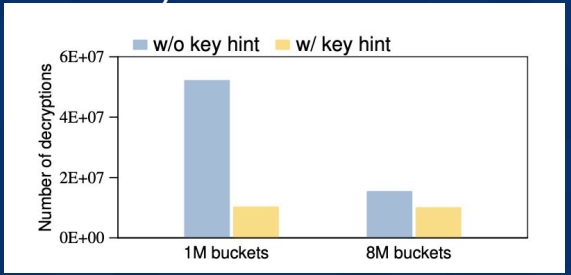
Multi-Threading

- **Issue**
  - Frequent synchronization across multiple threads
- **Solution**
  - An exclusive partition of hash key for each thread
- **Performance**
  - As each thread covers a disjoint part of the hash table, synchronization across threads for the table update is avoided
  - Exploiting parallelism without synchronization overhead
- **Limitation**
  - No dynamic changes in enclave threads



Searching for Encrypted Key

- **Background**
  - A naive search requires decrypting all the data entries in the hash bucket
- **Issue**
  - Cost of searching encrypted key increases when the length of a hash chain increases
- **Solution**
  - 1-byte key index as a hint to reduce decryption operations
- **Performance**
  - Significant reduction in number of decryption operations with key hint.
- **Limitation**
  - Key hint attack disrupt data availability of operations
  - Two-step search as a remedy



6 Performance

Setup

Hardware Setup

- Intel i7-7700 (4 physical cores with 8 logical threads)
- Ubuntu 16.4.5
- SGX driver/SDK 1.8

Evaluation Types

- Standalone Evaluation
  - Requests generated within the same machine
  - Focus on performance without network overhead
- Network Evaluation
  - Requests are generated by a separate client machine
  - Simulates 256 concurrent user requests

Setup

Test Controls and Variables

- Baseline
  - Stores entire hash table in enclave
  - Performance with demand paging
- Memcached
  - Common key-value store for comparison
  - Unmodified application running in enclave
- ShieldBase
  - Proposed key-value store without optimizations
- ShieldOPT
  - Proposed key-value store with optimizations

Evaluation Metric

Working Set

- Small (320 MB)
  - 16 byte key + 16 byte value
- Medium (1.6 GB)
  - 16 byte key + 128 byte value
- Large (5.2 GB)
  - 16 byte key + 512 byte value

Evaluation Metric

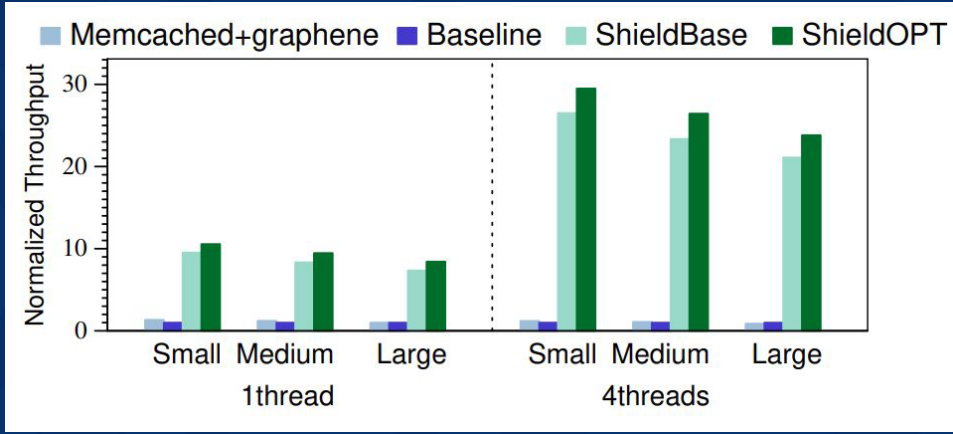
Workload

- Keys generated in both uniform and skewed (zipf) distributions
- Three configurations of read/write ratio
  - 50 read : 50 write
  - 95 read : 5 write
  - 100 read : 0 write

Workload	Description (R:W ratio)	Distribution
RD50_U	Update heavy workload (50:50)	Uniform
RD95_U	Read mostly workload (95:5)	Uniform
RD100_U	Read only workload (100:0)	Uniform
RD50_Z	Update heavy workload (50:50)	Zipfian
RD95_Z	Read mostly workload (95:5)	Zipfian
RD100_Z	Read only workload (100:0)	Zipfian
RD95_L	Read latest workload (95:5)	Latest
RMW50_Z	Read modify write workload (50:50)	Zipfian

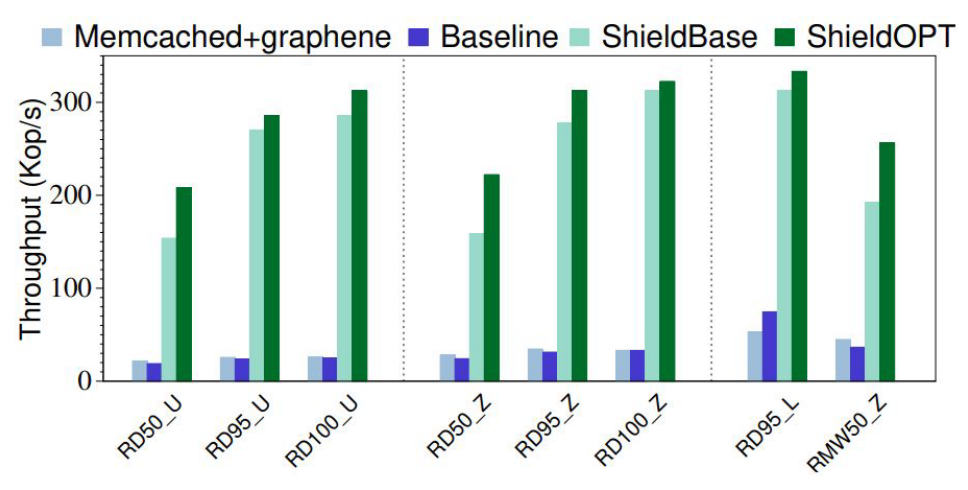
Standalone Working Set Result

- ShieldBase improves throughput by at least 7x on single thread
  - At least 21x on 4 threads
- ShieldOPT improves throughput by at least 8x on single thread
  - At least 24x on 4 threads



Standalone Workload Result

- ShieldBase improves throughput by 7x for write intensive workload
- Up to 11x improvement in throughput for read intensive workloads

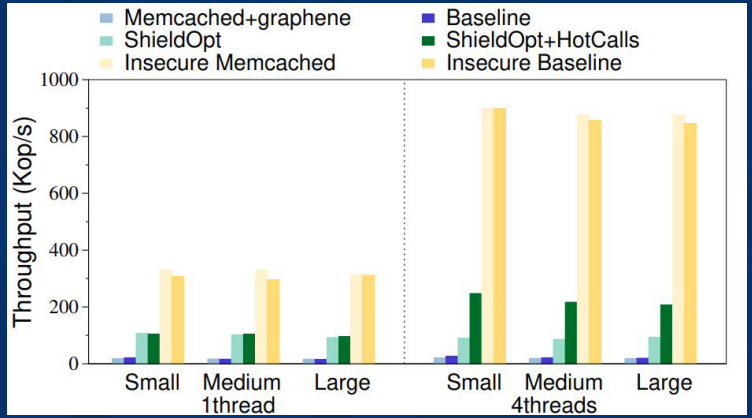


Network Evaluation Overhead

- Communication between client and server incurs additional overhead
  - Network communication must be done with system calls
  - Entails exiting enclave mode
- *HotCalls* reduces the overhead through shared memory
  - Facilitate enclave and non-enclave communication
  - Eliminates the need to exit enclave mode
- *HotCalls* is applied to the design for network evaluation
  - Reduce overhead from network communication
- Additional overhead from secure communication
  - Encryption/Decryption of requests/responses from server and client

Network Working Set Result

- ShieldOPT with *HotCalls* improves throughput by at least 4.9x on single thread
  - At least 9.2x on 4 threads
- Performance degrades by at most 3.9x compared to baseline without SGX
  - Overhead from protecting data and copying data to/from enclave
  - Still better than baseline design (at most 39.8x slower)

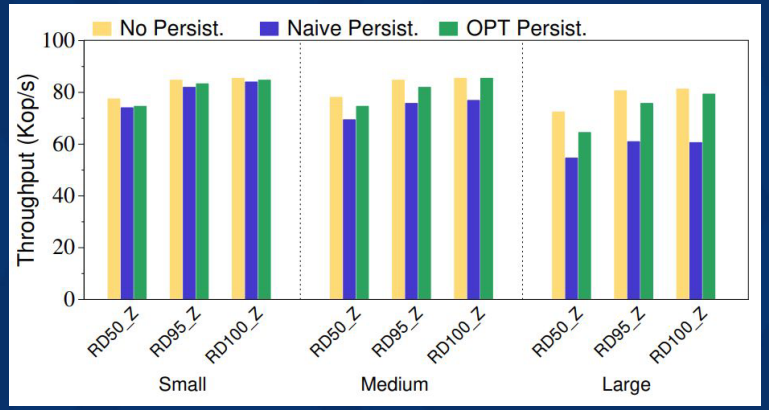


Network Working Set Result

- Overhead from secure communication is minimal in ShieldOPT with *HotCalls*
  - Impact performance by 5% on single thread
  - Impact performance by 7% on 4 threads
- Overhead from secure communication is more pronounced for baseline design
  - Impact performance by 9% on single thread
  - Impact performance by 27% on 4 threads

Persistency Performance

- Naive persistency as an additional experiment control
  - Execution is blocked during snapshot generation
- Naive persistency degrades performance up to 25%
- ShieldStore's optimized persistency degrades performance only up to 6%
  - Negligible impact on performance for high read operations



7 Limitations

Untrusted meta-data

- Changing a hash chain pointer to point to an enclave address
  - Critical data in the enclave region can be overwritten when writing new entries or data fields.

Mitigation: Before dereferencing pointers, check the value of untrusted pointers.

: Checking mechanism is simple and would add minimum overhead to ShieldStore.

Vulnerabilities

- Side Channel Attacks
  - Extract secrets through speculative execution even if stored in the EPC.

Mitigation: Currently none but expect future hardware to remedy.

- Custom heap allocator requires strict partitioning of metadata and data.

Mitigation: Currently none.

Weak persistency support

- If a crash occurs, ShieldStore loses all changes after the last snapshot.
- Current hardware monotonic counters for frequent sealing creates performance issues.

Mitigation: ShieldStore must be integrated with approaches that mitigate the limitation of monotonic counters.

ShieldStore Structure

- The hash-based index structure does not support range query operations.
- Incorporating the new index structures requires substantial changes.
  - Re-designing of integrity verification.

Mitigation: Modify index structure of ShieldStore (Future work)

Difficult Integration

- Popular Key-Value Cache/Stores such as Memcached or Redis use different data structures
  - Current implementation of integrity protection, makes it difficult to integrate/customise them to include ShieldStore.
  - Expected to change a substantial amount of code

8 Enhancements

Analysis

Cost Analysis

- Implementation Cost
- Maintenance Cost

Security Analysis

- Confidentiality and Integrity Guarantees

Analysis

Benchmarking

- Compare against other Industry Standards

Compliance

- Detail if ShieldStore will be compliant with current industry regulations

Thank you!

End of research paper
presentation by CS5322 Group 15

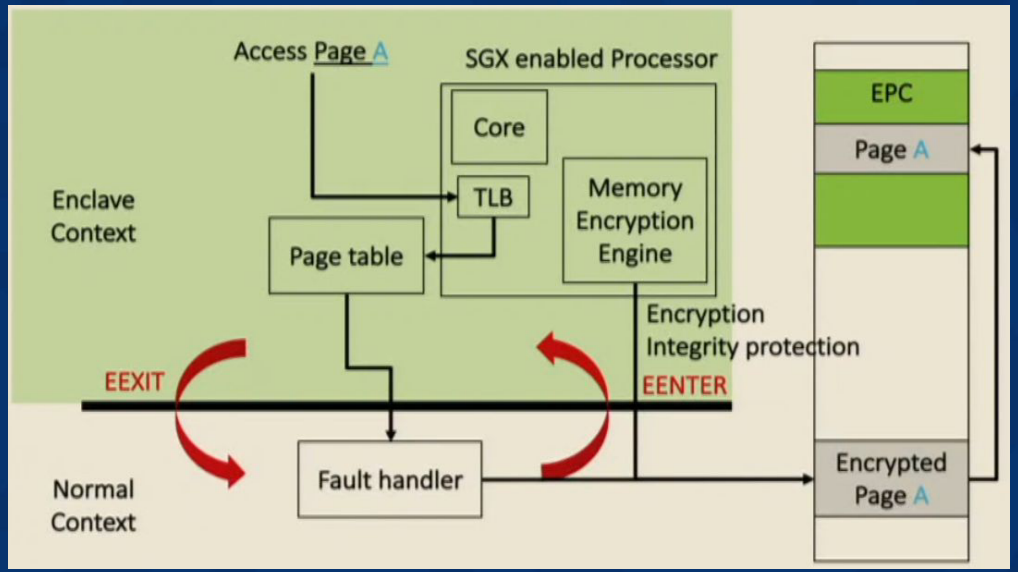
QnA

Intel SGX Memory Limitation

- Limited memory in EPC
  - 128 MB in total
  - Only 96 MB is usable by application due to security metadata
- Memory size of EPC is unlikely to increase
  - Space-time overhead of storing and processing fine-grained security metadata
  - Larger EPC memory will be unusable due to high latency of integrity verification

Intel SGX Memory Management Overhead

- Demand paging has high overhead as crossing enclave boundary is expensive
  - System calls must be done in normal context
  - 8000 CPU cycles required to enter/exit enclave context



Intel SGX Memory Management Overhead

- Performance degrades significantly when memory working set exceeds 96 MB
  - Demand paging incurs high overhead