Technische
Universität
Braunschweig

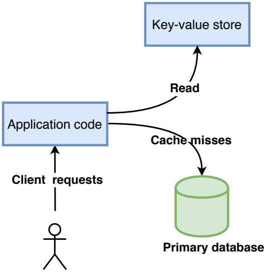
Precursor: A Fast Client-Centric Trusted Key-Value Store Using Intel SGX and RDMA

Ines Messadi, Rüdiger Kapitza, 2019-11-22

messadi@ibr.cs.tu-bs.de
Technische Universität Braunschweig, Germany

In-Memory Key-Value Stores

- Key-value stores are core of large-scale services
- Optimized systems can process millions of requests/second



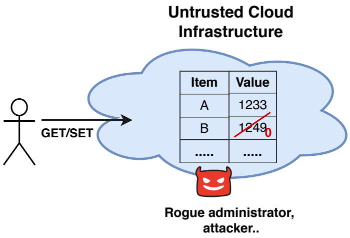
- Industry: Redis, Memcached,..

  → **Lack of basic security guarantees, e.g plaintext key-value items**
- Research: Concerto [SIGMOD'17]

  → **Secure but intensive computations and no support of fast networking technologies**

Security in the Cloud

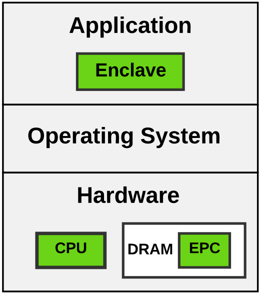
- Outsourced to the cloud
- Limited trusted in the cloud provider
- User data is exposed to malicious attacks
- Concerns about privacy & integrity



⇒ Improvements with trusted execution environments such as **Intel Software Guard Extensions (Intel SGX)**

Intel SGX Model

- Extension of the x86 instruction set
- Applications have secure compartments → **Enclave**
- Code & data reside in **Enclave Page Cache (EPC)**
- Confidentiality and integrity protected
- Restriction of **systems calls and I/O operations**

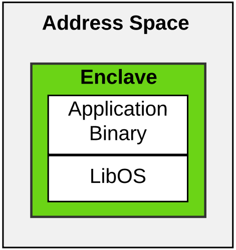


Approaches for Securing Applications

- **SGX SDK:** Porting the application → Tedious to port
- **Shielded execution:** Run unmodified applications with Graphene (ATC'17), SCONE (OSDI'16)..

→ Secure but large trusted computing base (TCB)

⇒ SGX is best suited for programs with small TCB

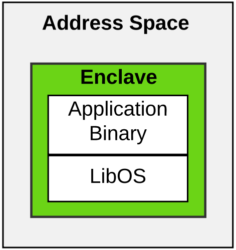


Approaches for Securing Applications

- **SGX SDK:** Porting the application → Tedious to port
- **Shielded execution:** Run unmodified applications with Graphene (ATC'17), SCONE (OSDI'16)..

→ Secure but large trusted computing base (TCB)

⇒ SGX is best suited for programs with small TCB



Can we use shielded execution runtime for key-value stores?

Intel SGX Architectural Limitations

1. Limited EPC memory
   - Limited to 128 Mibs, only $\sim$93 Mibs are usable
   - Secure paging mechanism $\rightarrow$ Overhead up to $\times$1000 [SCONE, OSDI]

   $\rightarrow$ **Cannot protect the full state using the EPC memory!**

2. System call restriction & enclave transitions
   - Enclave exiting, security checks and TLB flushing

   $\rightarrow$ **Performance loss**

3. New: DMA directly into the enclave are not allowed
   - Copying data in/out of enclaves

   $\rightarrow$ **Large copy overhead**

Intel SGX Architectural Limitations

1. Limited EPC memory
   - Limited to 128 Mibs, only ∼93 Mibs are usable
   - Secure paging mechanism → Overhead up to ×1000 [SCONE, OSDI]

   → **Cannot protect the full state using the EPC memory!**

2. System call restriction & enclave transitions
   - Enclave exiting, security checks and TLB flushing

   → **Performance loss**

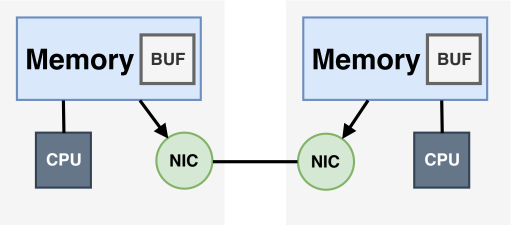
3. New: DMA directly into the enclave are not allowed
   - Copying data in/out of enclaves

   → **Large copy overhead**

How to secure applications that utilize
Remote Direct Memory Access (RDMA)?

Data Center Technology: RDMA

- Often employed in **data centers**
- **Zero-copy** & kernel bypassing communication
- Applications **register memory** with RDMA NIC
- Queue-based and asynchronous operations



Our Contribution

Precursor: A Fast and Secure Key-Value Store

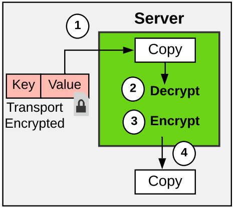
→ **Intel SGX** to Protect security-sensitive data
→ **RDMA** to achieve high-performance with low-latency

Security Properties

- Confidentiality: unauthorized entities cannot read the data
- Integrity: unauthorized changes to the data can be detected

Related Work: SGX-Based Key-Value Stores

- **SPEICHER** [FAST '19]
  - Tailored RocksDB implementation
  - Direct I/O library based on SPDK
- **ShieldStore** [Eurosys'19]
  - Store main data structure in untrusted memory
  - Relies on Merkle Tree for integrity verification

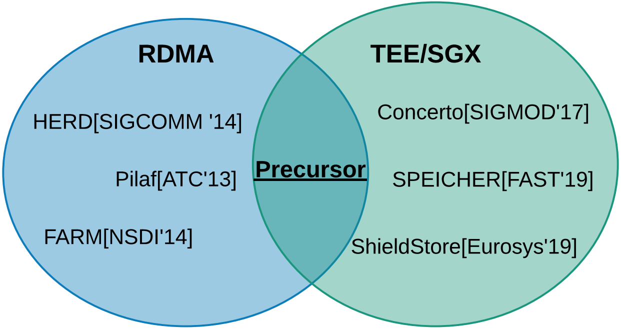


→ **Potential problems**

1. Additional data copy and encryption inside the enclave
2. Extensive server-side computation → CPU bottlenecks

Our approach: Client-side encryption to alleviate CPU bottlenecks

Contribution

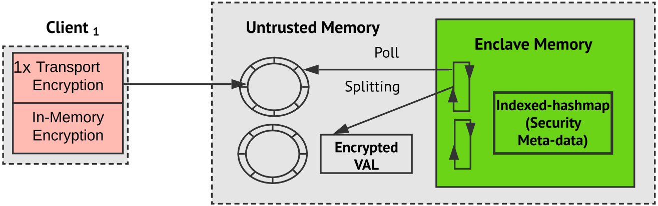


What do we gain from combining both technologies?
How to combine them efficiently?

Threat Model

- **What an adversary can do?**
  - Tamper with the OS and hardware
  - Tamper with key-value data
  - Tamper with key-value server code
- **An adversary cannot**
  - Modify the state within the enclave
- Clients environments are secure

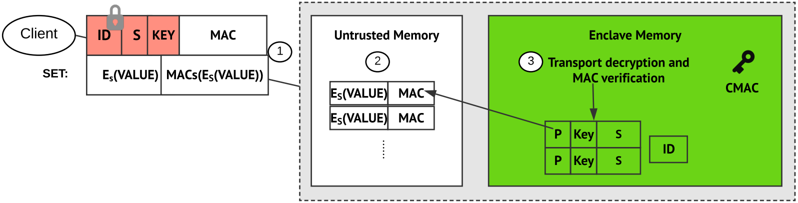
Overall Architecture



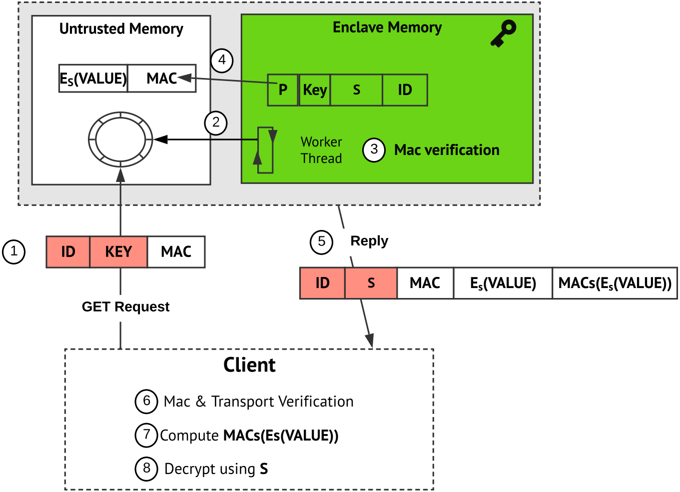
- Offloading cryptographic operations to the client-side
  $\rightarrow$ **Additional scalability**
- Splitting approach
  - No copy of the full payload in the enclave
- Flow control scheme
  - Server shares a memory window and regularly updates client

SET Request

- A unique per-operation client encryption key
- Data is placed in the untrusted memory
- Clients **pre-compute** cryptographic operations



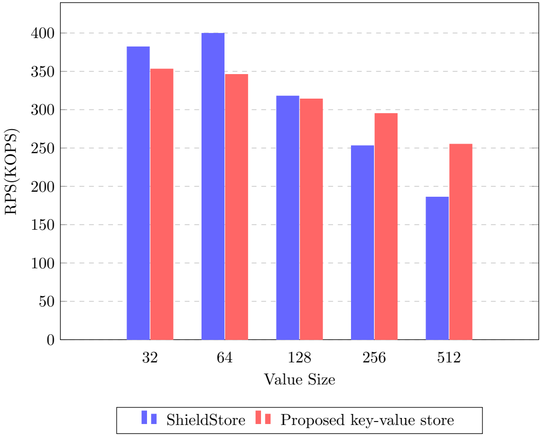
GET Request


Untrusted Memory
Enclave Memory
Es(VALUE)
MAC
4
P
Key
S
ID
2
Worker
Thread
3
Mac verification
1
ID
KEY
MAC
GET Request
5
Reply
ID
S
MAC
Es(VALUE)
MACs(Es(VALUE))
Client
6 Mac & Transport Verification
7 Compute MACs(Es(VALUE))
8 Decrypt using S

Evaluation Setup

- Workload: Yahoo! Cloud Serving Benchmark (YCSB) [SoCC 2010]
- Update-heavy workload
- Two machines with Intel Xeon E3-1230 v5
- Mellanox RoCE RDMA controller **10 Gbit/s**
- Comparison with Shieldstore [Eurosys'19]

Preliminary Results: Throughput



→ Reasonable performance and outperforms ShieldStore for large data sizes

Future Work

- Multi-core scalability
  - Efficient support of Multithreading with fewer synchronization
- Caches of the most popular accessed entries
  - Use of one-sided fetches
- Design of a distributed solution with multiple key-value stores

Conclusion

- A key-value store with strong confidentiality & integrity guarantees
- Contributions
  - Combination of RDMA and Intel SGX
  - Client-side computation

→ Leveraging RDMA improves the performance

→ Optimizing for **CPU utilization is key**