Treaty

Secure Distributed Transactions

Dimitra Giantsidi, Maurice Bailleu,
Natacha Crooks, Pramod Bhatotia

IEEE/IFIP DSN 2022
Distributed transactions

- A powerful programming abstraction
  - atomic processing of massive datasets
  - serializability
  - fault tolerance

- Properties (ACID)
  - Atomicity, Consistency, Isolation, Durability
Two-phase-commit (2PC) protocol

Txs require to exchange messages and log persistently their state
Transactions in the cloud

Attacker's can compromise the security properties
Threats for distributed Txs in the cloud

#1: Secure execution

#2: Secure persistency
Threat #1: Secure execution

How to guarantee secure execution for Txs?
Threat #2: Secure persistency

How to ensure secure persistency (crash-consistency + rollback protection) for Txs?
Problem statement

To design a distributed KV store with secure Tx execution and secure persistency
Our proposal

**Properties:**

- Distributed serializable Txs
- Confidentiality, integrity and secure persistency
- Performance

**Treaty**

A secure distributed transactional KV store
Treaty overview

Client

Untrusted storage

Coordinator

Participant #1

Participant #k

TEE

TEE

TEE

Secure protocol for Txs

Trusted services

Untrusted storage

Untrusted storage

Untrusted storage

Untrusted storage

Untrusted cloud infrastructure
Outline

- Motivation
  - Background and challenges
- Design
- Implementation
- Evaluation
Trusted Execution Environment

- HW extensions for trusted computing
  - Intel SGX, Arm TrustZone, etc.

- Trusted area (enclave)
  - Integrity + confidentiality

Treaty builds on TEEs to guarantee security for distributed Txs
Challenge #1: Distributed systems

- TEEs do not protect the network operations

- Adversaries can tamper with Txs messages
  - integrity, confidentiality
  - replay-attacks

**EEs cannot guarantee secure execution for distributed Txs**
Challenge #2: Stateful systems

- TEEs do **not** protect the persistent data and logs
- Adversaries can violate system correctness
  - delete or replace logs
  - compromise persistent data

**TEEs cannot guarantee secure persistency for committed Txs**
Outline

- Motivation
- Background and challenges
  - Design
  - Implementation
  - Evaluation
Treaty

Secure Tx protocol + Stabilization protocol

Secure execution

Secure persistency
Treaty shields (a) the 2PC protocol and (b) the network messages for secure execution
Treaty builds on (a) trusted services and (b) secure log files for secure persistency
Outline

- Motivation
- Background and challenges
- Design
  - Implementation
- Evaluation
A Treaty node: System stack

Other nodes

Untrusted network

TEE

- Trusted services
- Secure Tx protocol
- Network layer

Tx KV engine

Storage layer

Untrusted storage
Network layer

- Low-latency shielded communication
- Direct I/O within the TEE
- Metadata to prevent replay-attacks
- Implemented on top of RDMA/DPDK

Our network layer (a) optimises and (b) shields the network operations
Storage layer

- In-memory (hybrid) KV data structure
- Persistent data in authenticated files
- Pessimistic + optimistic single-node Txs
- Implemented on top of RocksDB

Our storage layer (a) secures the persistent data and (b) optimises the TEE usage
Outline

- Motivation
- Background and challenges
- Design
- Implementation
- Evaluation
Evaluation

Questions:

- What are the overheads of Treaty's 2PC (stand-alone)?
- What are the performance overheads for Treaty?

More results in the paper!

Hardware setup:

- TEE: Intel SGX
- 3x Intel i9-9900K (@3.60GHz, 8 cores, 16 HT)
- Intel NIC XL710 (40Gb/s, QSFP+)
Q1: 2PC's overheads

Treaty's 2PC overheads mainly derive from the TEE

Lower is better
Q2: Overall overheads

Treaty offers strong security w/ reasonable overheads w.r.t. the state-of-the-art
Summary

- Distributed Txs are an integral part of the third-party cloud infrastructure

- Secure transaction processing is challenging
  - TEEs are not designed for distributed systems with Txs and untrusted storage

- Treaty: A secure distributed Tx KV store with strong security guarantees
  - Secure 2PC protocol
  - Stabilization protocol
  - TEEs + direct I/O

Source code: https://github.com/TUM-DSE/Treaty
Backup slides
Is Treaty a viable solution?

<table>
<thead>
<tr>
<th>Secure Tx systems</th>
<th>Obladi [OSDI'18] (single-node)</th>
<th>Fabric [EuroSys'18] (blockchain)</th>
<th>Treaty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (ms)</td>
<td>~340</td>
<td>370-550</td>
<td>80-320</td>
</tr>
<tr>
<td>Secure storage systems</td>
<td>Speicher [FAST'22] (single-node)</td>
<td>TWEEZER [FAST'22] (single-node)</td>
<td>Treaty</td>
</tr>
<tr>
<td>Tps overheads</td>
<td>15x-17x</td>
<td>4x-9x</td>
<td>4x-15x</td>
</tr>
</tbody>
</table>

Treaty incurs similar overheads with the state-of-art secure systems.
<table>
<thead>
<tr>
<th>Threat model</th>
<th>Treaty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compromised system stack (OS/hypervisor)</td>
<td>Yes</td>
</tr>
<tr>
<td>Network adversaries, (e.g., delay, drop, replay</td>
<td>Yes</td>
</tr>
<tr>
<td>and manipulate network traffic)</td>
<td></td>
</tr>
<tr>
<td>Host memory memory manipulation</td>
<td>Yes</td>
</tr>
<tr>
<td>Unauthorized modifications to persistent storage</td>
<td>Yes</td>
</tr>
<tr>
<td>DoS</td>
<td>No</td>
</tr>
<tr>
<td>Cache-timing attacks (e.g., speculative execution,</td>
<td>No</td>
</tr>
<tr>
<td>access pattern leakage, memory safety vulnerabilities)</td>
<td></td>
</tr>
</tbody>
</table>
### EnclaveDB: A secure KVs with Txs

<table>
<thead>
<tr>
<th>TEEs</th>
<th>EnclaveDB [SP'18]</th>
<th>Treaty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data model</td>
<td>Emulated h/w</td>
<td>Real h/w</td>
</tr>
<tr>
<td>Data distribution</td>
<td>In-memory KVs</td>
<td>Persistent KVs</td>
</tr>
<tr>
<td>Overheads</td>
<td>No (single-node KVs)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>1.4x</td>
<td>4x-15x</td>
</tr>
</tbody>
</table>

EnclaveDB does not show the real TEEs' overheads.
Speicher: A secure LSM-based storage system

<table>
<thead>
<tr>
<th></th>
<th>Speicher [FAST'19]</th>
<th>Treaty</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEEs</td>
<td>Real h/w</td>
<td>Real h/w</td>
</tr>
<tr>
<td>Data model</td>
<td>Persistent KVs</td>
<td>Persistent KVs</td>
</tr>
<tr>
<td>Data distribution</td>
<td>No (single-node KVs)</td>
<td>Yes</td>
</tr>
<tr>
<td>Txs</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Overheads</td>
<td>~15x</td>
<td>4x-15x</td>
</tr>
</tbody>
</table>

Treaty shares similar overheads with state-of-the-art secure storage systems
Trusted substrate for Txs

- Configuration and attestation service (CAS)
  - low-latency attestation

- Userland scheduler
  - low-latency operations

- Memory management
  - TEE memory usage
Userland scheduler

- Low-latency operations for multiple clients
- A userspace thread (fiber) for each client
- Lightweight context switches
  - Round-robin scheduling
  - No context-switches or interrupts
Authenticated LSM data structure

- TEE
  - KV engine
  - In-memory KV
  - Host memory
    - SST#0
    - SST#1
    - SST#2
    - SST#3
    - SST#4
    - SST#k
  - Untrusted storage
    - WAL
    - Clog
    - MAN.

- Sorted KV pairs (encrypted)
  - KV1
  - KV2
  - KVn

- SSTable file
  - Footer
  - Hashes of blocks
## Log file and message format

<table>
<thead>
<tr>
<th>Trusted id</th>
<th>Hash</th>
<th>Encrypted entry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Secure Log file format

<table>
<thead>
<tr>
<th>IV</th>
<th>metadata</th>
<th>TX data</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Secure message format