LSKV: Democratising Confidential Computing from the Core

Andrew Jeffery – University of Cambridge
andrew.jeffery@cst.cam.ac.uk
“A **distributed**, reliable key-value store for the most **critical data** of a distributed system” — etcd.io (emphasis mine)
The core etcd API

- Put(key, value)
- Range(key, range_end, revision)
- DeleteRange(key, range_end)
- Txn(^)

- LeaseGrant(ttl)
- LeaseKeepAlive(id)
- LeaseRevoke(id)

- Watch(key, range_end, revision)

<table>
<thead>
<tr>
<th>Put</th>
<th>foo1 = bar @ revision 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Txn</td>
<td>foo2 = baz @ revision 6</td>
</tr>
<tr>
<td></td>
<td>foo3 = bat @ revision 6</td>
</tr>
</tbody>
</table>

Range(foo1, foo4) = [foo1, foo2, foo3]

Range(foo1, foo4, 5) = [foo1]
Datastores in the trusted cloud
Problems in the untrusted cloud
Problems in the untrusted cloud

Problem 1

Problem 2

Client

Proxy

etcd

etcd

etcd

Client
Problem 1 - Trusted Cloud?
Problem 1 - Trusted Cloud?
Solution 1 - **Untrusted Cloud**

- **TLS Key**
- **Memory FS Key**
- **LSKV**
- **SGX**
- **encrypted Storage**

SSL/TLS key

- **aTLS** connections to other LSKV instances.
LSKV: The Ledger-backed Secure Key-Value store

- LSKV
- CCF
- SGX Enclave
- Hypervisor
- Memory
- Storage
CCF: the Confidential Consortium Framework

github.com/microsoft/CCF

or

ccf.dev
CCF: the Confidential Consortium Framework

github.com/microsoft/CCF

or

ccf.dev
CCF: the Confidential Consortium Framework

github.com/microsoft/CCF

or

ccf.dev
CCF: the Confidential Consortium Framework

github.com/microsoft/CCF

or

ccf.dev

Operator
Untrusted

Governor
Partly trusted

User
Trusted, subject to access
CCF: the Confidential Consortium Framework

github.com/microsoft/CCF

or

ccf.dev

Operator
Untrusted

Governor
Partly trusted

User
Trusted, subject to access

Replicated Encrypted Ledger

Governance Transactions
Business Transactions

Start
Join
Node
LSKV has an etcd-compatible API

- Put(key, value)
- Range(key, range_end, ?revision)
- DeleteRange(key, range_end)
- Txn(^)
- LeaseGrant(ttl)
- LeaseKeepAlive(id)
- LeaseRevoke(id)
- Watch(key, range_end, ?revision)
Solution 1 – Confidentiality with compatible API

Problem 2

Solution 1

Client

Proxy

LSKV

LSKV

LSKV
## Trade offs

<table>
<thead>
<tr>
<th></th>
<th>etcd</th>
<th>LSKV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consistency</strong></td>
<td>Strong</td>
<td>Optimistic</td>
</tr>
<tr>
<td><strong>Confidential</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Management transparency</strong></td>
<td>Missing</td>
<td>Available on the ledger</td>
</tr>
<tr>
<td><strong>API</strong></td>
<td>etcd API</td>
<td>etcd API + extras</td>
</tr>
</tbody>
</table>
Technical interlude – Optimistic consistency
Onto problem 2! Mean proxies
Problem 2 - Proxies can be mean

Client

Put(alice, £500)

alice = £500

Proxy

Put(bob, £500)

bob = £500

LSKV
Solution 2: Don’t trust the proxy - get a receipt

Put(alice, £500) & GetReceipt()

alice = £500

Signed receipt for bob = £500

Put(bob, £500) & GetReceipt()

bob = £500

Signed receipt for bob = £500

INVALID RECEIPT!!
Solution 2 - Get receipts
Sorry, I missed that

- Current datastores aren’t suited for confidential operation

- LSKV is a new **confidential datastore**, built on CCF with an etcd-compatible API

- LSKV can highlight **untrustworthy proxies** using receipts

- Oh, and it is fast: **3.5x throughput, 50% latency** vs etcd
Thank you

github.com/microsoft/LSKV

Andrew Jeffery – andrew.jeffery@cst.cam.ac.uk
Oh, and it's fast – YCSB workloads (3 nodes)
Ledger

- Operations are either public or private

- Private operations cannot be decrypted by the operator, governors have to combine key shards

- Responsibility of the operator to synchronise the ledger files to other nodes when joining new ones

- Ultimately used for disaster recovery
Optimistic checking

- Since all operations are optimistically acknowledged you may need to follow up if you want to check commit status

- Can also get this from responses to other requests

- Plans to have a watch channel for commit status
Historical staleness

- Specifying a revision acts on a historical copy of the store

- This can lead to observing stale data

- Watches are served from this
Durability

- Operations are persisted to disk lazily

- They also may not be available later, try to keep things in memory

- Stems from not trusting the host
Tackling untrusted proxies – read receipts

- Similar to write receipts but for read operations

- Processed in-application, at any node (not just the leader)

- May need to add a nonce-like field or minimum revision to range requests
  - Maybe use min revision fields in etcd range requests already