HQLA\textsuperscript{x} and Corda: The Road to Production

Raoul Bhoedjang, Cees van Wijk
Outline

- HQLA\(^x\): The business-case
- Solution Architecture
- Experiences with Corda
- Next steps
HQLA\textsuperscript{x}: The Business Case
A new collateral exchange to manage liquidity more effectively and more cost-efficiently.
Current situation

- **Fragmented** Securities settlement system
- **Slow** Settlement / operational bottlenecks due to physical delivery of securities
- **Inefficient** Reconciliation
- **Expensive** Management of key regulatory ratios (e.g. Liquidity Coverage Ratio)
HQLA$^x$ solution with Corda DLT

- **Real-time** effective and cost-effective Marketplace without securities movement

- **Reduced** capital costs via enhanced LCR ratio management

- **Enhanced** regulatory **transparency** (understanding of collateral chains)

- **Mitigated** systemic **risks**

- Supported by **legal framework**
How does it work?

- Use **distributed ledger technology to digitize baskets** of assets under DCRs (Digital Collateral Receipts)
- The **legal title of DCRs** (and its underlying assets) is **transferred in real-time**
- The **securities** are maintained at a trusted third party.
HQLA® User Interface
Why did we choose Corda?

Why use a DLT in HQLA?*

• Create a standardized marketplace.
• Give regulators a “DCR Tracking” view.
• Consistent and transparent data across all market participants.
• Low volume high value use case.

Why use Corda in HQLA?*

• Selective multicasting of transactions (privacy).
• Pluggable consensus (performance, security).
• Intel Security Guard Extensions (privacy).
• Designed with input from major financial institutions.
Aiming for production in Q4 2018

- **IDEA PHASE**
  - From Oct to Dec 2016

- **INCUBATOR PHASE**
  - From January to May 2017

- **ACCELERATION PHASE**
  - Live Pilot HQLAX for Q4 2017

- **PRODUCTION PHASE**
  - Full production Q4 2018

**Timeline Events**

- **ENTERING R3'S INCUBATOR**
  - 31 October 2016

- **BANKS PARTICIPATE PROVIDING BUSINESS AND TECH KNOWLEDGE**
  - 16 January 2017

- **ENTERING R3'S ACCELERATOR WITH DEDICATED TECH TEAM**
  - 10 June 2017

- **KICK OF PHASE 2**
  - 10 July 2017

- **END OF POC**
  - 06 March 2017

- **LIVE ON CORDA TESTNET**
  - 05 July 2017

**Live Trade conducted Jan 24th**

**HQLAX**

**RAISING THE POWER OF LIQUIDITY**

**ING**
HQLA$^x$ Platform

HQLA$^x$ Minimum Viable Product (MVP) is under development, runs on Corda V2.0.

Core settlement functions for DCR swaps have been implemented:
- Management of DCR and trade lifecycles
- Scheduling and maturing of DCR swaps, including haircuts

HQLA$^x$ is deployed on R3’s Corda TestNet. Multiple partner banks run an HQLA$^x$ node.

HQLA$^x$ runs a separate private DevNet for development purposes.
Live Trade performed

- **Jan 24th 2018** ING another large European bank swapped two High Quality Liquid Asset (HQLA) Digital Collateral Receipts (DCRs) of value **EUR 25 million**.

- The DCRs were collateralized by a large European custodian.

- During the transaction, **legal ownership of the DCRs was transferred** on Corda while the underlying baskets of securities remained stationary at the custodian.

- The trade was activated on January 24th and matured on January 25th.

- The trade was performed on a dedicated Corda network (PilotNet).
Solution architecture
**HQLA<sup>x</sup> Architecture**

**Banks trade DCRs**

**Marketplace (Trading Front-end)**

- **Bank Node**
- **Bank Node**
- **Marketplace Node**

**Distributed Ledger**

**Records DCR Ownership**

- **Custodian Node**
- **Custodian Node**

**Ordering**

**Discovery**

**I&AM**

**Corda**

**Regulator Node**

**Regulator Node**

**Regulators observe transactions**

- **Custodians record DCR inventory**
Core Transactions

Financial transactions:
- DCR swap: atomic delivery-versus-delivery (implemented)
- DCR pledge (in progress)
Key Design considerations

Confidentiality
- Who can see what (e.g., in forward lending scenarios)?

What (not) to store on the ledger?
- DCR and Trade related states and transactions are stored.
- Storing inventory (ISIN etc) has pros and cons.

Who signs what?
- Custodian (e.g. do we need a signature per trade?).
- Notary (e.g. do we need a notary signature if both traders have signed).
Current Setup on R3’s Testnet

R3 Testnet + HQLA-x Software

Ordering
Discovery
I&AM

Pilot Bank UI
Bank Node
Bank Node
Custodian Node
Pilot Custodian UI

Regulator Node
Key roles of DLT in HQLA:
- Record and track DCR ownership.
- Post-trade settlement: atomic delivery-versus-delivery of DCRs.
- Enforce business rules (e.g. only DCR with confirmed collateral can swap).
- Create consistency and transparency among all participants.
- Enforcing digital signatures by all parties involved in a transaction.
- Give regulators a real-time tracking view.
Current Per-Node Deployment Structure

- Same deployment structure is used for traders and custodians.
- Web service provides REST API.
- REST APIs can be used as an integration point for custodian and frontend systems.
- At present, REST API are used only by the web UIs that function as stubs for these systems.
- Web service and Corda service can run on separate nodes.
- Deployable artifacts:
  - HQLA-x CorDapp (Corda plugin)
  - Custodian web server
  - Trader web server
Corda Notary in HQLA

- Current pilot set-up
- Non validating, high available notary is run by R3.
- For privacy reasons, we now use a non-validating notary.
- With SGX, we might switch to a validating notary.
Performance

Requirements:
- Throughput: a few hundred transactions per day
- Latency: seconds

Throughput is easily attainable with Corda today, Corda performance depends on configuration:
- Number of peer nodes: Currently 5
- Geographic spread of nodes: Currently all in Europe, later global.
- Consensus model: Currently non-validating crash-failure tolerant notary cluster.
- Data store: Currently H2, later Oracle
- Peer communication protocol: AMQP over TLS

The latency of HQLA transactions appears relatively high (seconds).
  - serialization costs of flow framework (single-threaded H2 database access)

More on Corda performance:
https://www.corda.net/2017/12/dlt-performance-considerations/
https://youtu.be/RekUU_keH7c
Privacy and confidentiality in HQLA

Current mechanisms:
- Transactions are only shared with parties directly involved in the transaction (including Notary, Custodian and Regulator).
- No forward lending chains.
- General problem: using an asset in multiple financial transactions can create privacy issues.

Mitigation options:
- Break multitransaction chains.
- Use Corda’s confidential identities (partial solution, no full confidentiality).
- Secure computing based on Intel SGX.
**Problem**: Corda verifies transaction chains. This potentially conflicts with privacy.

**Solution**: Run verification in SGX enclave.

**Trade-off**: rely on SGX or take extra privacy measures (e.g., reissuance)

**SGX concerns**: special hardware, vendor lock-in, key management, side channels
Experiences with Corda
Experiences with Corda

Benefits over competitors:
- Compared to **Ethereum & Quorum**:
  - Standard development tooling
  - Any JVM-based language can be used (Kotlin, Java etc.)
  - No EVM (gas, gas-limit, slowness)
  - Single support organization
  - No Proof of Work (Ethereum only)
  - Quorum private transactions do not achieve consensus (Corda privacy features do)
- Compared to **Hyperledger Fabric**:
  - Clear roadmap
  - More mature technology
  - BFT-SMaRt consensus

No free lunch:
- Requires fairly high level of DLT/Corda-specific expertise from developers.
- Explicit coordination (flow framework).
- Mapping Business logic to UTXO requires rethinking.
- Transaction verification can become complex.
BitCoin UTXO rules (subset)
- All inputs and outputs are positive.
- The sum of all inputs equals the sum of all outputs, except for coin-issuance (mining rewards).
- Only unspent outputs can serve as input.
- ...

HQLA\(^{x}\) UTXO rules (subset)
- Only new DCRs can be confirmed
- Only Custodian can confirm the inventory
- Only DCRs with confirmed inventory can be swapped
- ...

(Simplified version of actual system functions)
Transaction Verification

Some transactions are complex and consist of multiple subtransactions.

Verification can become, well, messy:

- Many conditions to check.
- Duplicate verification code for similar subtransactions.
- Correct selection of commands/states to check.

Solution patterns that we use (not specific to HQLA\(^x\)):

- State machines
- Transaction parsers
- Hierarchical checking
**State Machines**

Typical lifecycle transaction: one command, one input, one output

Implement a ContractState as a state machine.

State machine implements most checks.

State machine is used twice:
- Phase 1: **Build** transaction
- Phase 2: **Verify** transaction

```kotlin
fun verify(tx: LedgerTransaction) {
  ...
  tx.inputs.first().state.data
  .grantAccess()
  .assertEquals(tx.outputs.first())
  ...
}
```
Physical Transaction Layout

How to recover the logical transaction structure?
Transaction Parsing

*Transaction parsers* group commands/inputs/outputs into logical units.

Parsers operate from left to right (in a transaction).

Parsers can be combined into new parsers.

```scala
val dcrMoveParser = concat(input(), output())
val dcrSwapParser = concat(dcrMove, dcrMove)
transaction.parse(dcrSwapParser)
```

*Parsers recover the logical transaction tree structure*
Recovered Structure

```
SwapWithHaircuts
  └── TransferWithHaircuts
      ├── TitleTransfer
      │    └── StateUpdate
      │         └── Input
      │          └── Output
      └── Pledge
            └── StateUpdate
                  └── Input
                  └── Output

TransferWithHaircuts
  └── TitleTransfer
      └── StateUpdate
          └── Input
          └── Output

TransferWithHaircuts
  └── Pledge
      └── StateUpdate
          └── Input
          └── Output
```
## Hierarchical Checking

Each parser produces a logical, verifiable object. We verify each object as it is constructed.

<table>
<thead>
<tr>
<th><strong>Verifiable object</strong></th>
<th><strong>Checks applied to object</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>AssetSwap, AssetTransfer</td>
<td>Matching parties, matching amount</td>
</tr>
<tr>
<td>TransferWithHaircuts</td>
<td>Matching parties</td>
</tr>
<tr>
<td>TitleTransfer</td>
<td>New owner</td>
</tr>
<tr>
<td>LinearStateUpdate</td>
<td>Same linear id</td>
</tr>
</tbody>
</table>
Development tooling

Industry standard tooling is used (partly because Corda is JVM-based)
Live Transaction Experiences

Positive:
- Local deployment is easy.
- Deployment of Corda/Cordapps is relatively easy:
  - Ansible install
  - Get signed certificate from Corda doorman

        java -jar corda.jar --initial-registration

  - Drop HQLA² jar in Corda plugin directory
  - No reconciliation due to a shared ledger

Learnings:
- Peer-to-peer firewalling is painful and error-prone.
  - Firewall float (under development) will address this.
- Network map issues: code is being revised.

Missing:
- Smoke test: connectivity check after certificate signing
Next steps
Integration

The HQLAx system identifies 4 integrating parties: Banks, Trading front-end, Custodians, Regulators. All parties have 2 integration options:

1. **Easy connect**: (Pilot) Run a Corda-node in the cloud, interact with the User Interface.
2. **Full integration**: (Production) Run a Corda-node on Bank infrastructure and connect it to back-office systems

After successfully finishing the live trade, the participating banks are now working on full integration which requires compliance with all nonfunctional requirements:

- Risk, user access management, pen-tests etc.
- Operations: Monitoring, Capacity management, failover, backup/restore, failover etc.
- Alignment with existing organization: standardization, policies etc.
Bank integration of HQLA\textsuperscript{x}

**Easy Connect**

- Bank A User Interface
- Bank Node
- Pilot Custodian UI
- Custodian Node

**Full Integration**

- Bank B Backoffice
- Bank Node

**R3 Testnet + HQLA-x Software**

- Ordering
- Discovery
- I&AM

**corda**

**Regulator Node**

**Easy Connect**

**HQLA\textsuperscript{x}**

**Bank integration of HQLA\textsuperscript{x}**
Questions?

www.hqla-x.com