Token management in Fabric
Assets can be conveniently represented with digital tokens

Use cases:
- Securities trading
- Asset transfer
- Digital currency
- Supply chain
- Provenance
- …
Unspent Transaction Output (UTXO) token ownership model

**Standard case:**
- In a transaction, the sum of the values of all the inputs must be greater or equal to the sum of the values of all outputs
- Only *unspent* outputs of previous transactions can be used as inputs to a new transaction
- With a new transaction, inputs are deleted and new outputs are created that may be consumed in future transactions

**Privacy-preserving case:**
- Inputs of a valid transaction make respective outputs in the UTXO pool cryptographically unspendable
- Correctness of payments cryptographically enforced
Account model for token ownership

- Single account per system user
- Transactions carry transfer requests, and validation updates corresponding user-accounts
- To some extend and on the client side, can be simulated on top of UTXO model
- Do not support privacy-preserving transactions -> conversion to UTXO is needed
- Support a variety of transfer extensions (e.g, `transferFrom/approve`
Privacy is a key requirement in token management.

- **Participants**
  - Bank A: BNK_A
  - Bank B: BNK_B
  - Bank C: BNK_C

- **Token units**
  - LYYL
  - WTC
  - MFG

**Account Simulation**
- BNK_A: LYYL 2
- BNK_B: LYYL3
- BNK_B: WTC 5
- BNK_C: MFG 8

- Alice: LYYL 2
  - BNK_B: LYYL 4
  - BNK_B: WTC 5
  - BNK_C: MFG 8
  - BNK_C: LYYL 1

- BNK_B: LYYL 3
  - BNK_B: WTC 5
  - BNK_C: MFG 8
  - BNK_C: LYYL 2

**UTXO Transactions**
- Import from Issuer
  - 2 LYYL to BNK_A
  - 3 LYYL to BNK_B
  - 5 WTC to BNK_B
  - 8 MFG to BNK_C

- Transfer BNK_A’s 1 LYYL to BNK_B & 1 LYYL to self
- Transfer BNK_B’s 1 LYYL to BNK_C
- Transfer BNK_B’s 1 LYYL to BNK_C

**Standard model**
(No privacy)
Privacy is a key requirement in token management.

- **Participants**
  - Bank A: \( \text{BNK}_A \)
  - Bank B: \( \text{BNK}_B \)
  - Bank C: \( \text{BNK}_C \)

- **Token units**
  - LYYL
  - WTC
  - MFG

**Standard model** (No privacy)
- Import from Issuer
  - 2 LYYL to \( \text{BNK}_A \)
  - 3 LYYL to \( \text{BNK}_B \)
  - 5 WTC to \( \text{BNK}_B \)
  - 8 MFG to \( \text{BNK}_C \)

**Hyperledger Fabric** (Identities, assets concealed)
- Import from Issuer
  - 2 LYYL to \( \text{BNK}_A \)
  - 3 LYYL to \( \text{BNK}_B \)
  - 5 WTC to \( \text{BNK}_B \)
  - 8 MFG to \( \text{BNK}_C \)

**Transactions**
- Transfer \( \text{BNK}_A \)'s 1 LYYL to \( \text{BNK}_B \)
- Transfer \( \text{BNK}_B \)'s 1 LYYL to \( \text{BNK}_C \)
- Transfer Alice's 1 LYYL to \( \text{BNK}_C \)
- Transfer Bob's 1 LYYL to \( \text{BNK}_B \)
- Transfer Charlie's 1 LYYL to \( \text{BNK}_C \)
- Transfer Alice's 1 LYYL to \( \text{BNK}_B \)
- Transfer Alice's 1 LYYL to \( \text{BNK}_B \)
- Transfer Alice's 1 LYYL to self
Privacy is a key requirement in token management.

**Account Simulation**

<table>
<thead>
<tr>
<th>Account</th>
<th>UTXO Transactions</th>
<th>Account</th>
<th>UTXO Transactions</th>
<th>Account</th>
<th>UTXO Transactions</th>
<th>Account</th>
<th>UTXO Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNK_A</td>
<td>LYYL 2</td>
<td>BNK_B</td>
<td>LYYL 4</td>
<td>BNK_C</td>
<td>LYYL 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNK_B</td>
<td>LYYL 3</td>
<td>BNK_B</td>
<td>WTC 5</td>
<td>BNK_C</td>
<td>LYYL 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNK_C</td>
<td>MFG 8</td>
<td>BNK_B</td>
<td>LYYL 3</td>
<td>BNK_B</td>
<td>WTC 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNK_A</td>
<td>LYYL 2</td>
<td>BNK_B</td>
<td>WTC 5</td>
<td>BNK_C</td>
<td>MFG 8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Standard model**  
(No privacy)

**Hyperledger Fabric**  
(Identities, assets concealed)

**Hyperledger Fabric**  
(e.g., view of auditor of BNK_B)
FabToken in a nutshell

- Fabric enablement for *direct* or *as-a-service* token management using *UTXO*

- *Modular* architecture to accommodate a variety of implementations addressing different privacy, performance requirements & regulatory restrictions

- Compatible and *integrate-able* with other UTXO based token systems

- Easily *extensible* to support a variety of financial services operations
Zero-Knowledge Asset Transfer is a leading technology to privacy-preserving asset management on permissioned Blockchains

- **Strong identity management**
  - Users associated to long term identities that they cannot deny use of; provided by the Identity Mixer Technology

- **Accountability Non-repudiation**
  - User anonymity
  - Transferred token confidentiality (type, value)

- **Privacy**
  - On a per user-level: auditors bound to a user are guaranteed unconditional access to that user’s transaction details

- **Audit support**
  - Lightweight (trusted) setup, easily decentralized
  - Lightweight transfer request computation

- **Performance**
  - Standard cryptographic assumptions
How to combine public verifiability with privacy? Using Zero-Knowledge (ZK) proofs!

“I can prove to you that I know a secret”

- Age threshold (e.g., above 18 years)
- Funds (e.g., enough money on account)
- Asset ownership (e.g., private key)
- Membership (e.g., business network)
Token information flow in Fabric

**Ledger (K-V)**

- Block n-1
- Block n
- Block n+1

**Channel trust domain**

Transaction validation that takes place upon a transaction being added to the channel’s ledger

**Client domain**

Application that constructs & submits FabToken transactions to the system

**Ordering Service**

- FabToken transaction *broadcast*

- FabToken transaction *deliver*
FabToken exhibits a modular architecture

**Client domain**
*Application that constructs & submits FabToken transactions to the system*

**Channel trust domain**
*Transaction validation that takes place upon a transaction being added to the channel’s ledger*

**Ledger (K-V)**
- Block n-1
- Block n
- Block n+1

**TokLib [Token Management System Factory]**
- Construct Issue/Transfer/Redeem requests

**Read/write requests**
- Broadcast
- Deliver

**Ordering Service**
- FabToken transaction

**Read requests**
- Fabric transactor
FabToken exhibits a modular architecture

**Ledger (K-V)**
- Block n-1
- Block n
- Block n+1

**TokLib [Token Management System Factory]**
- Construct Issue/Transfer/Redeem requests

**Application**
- (End-user transactor/issuer)
  - Client Wallet Library [cwLib] (constructs & submits token transactions)
  - Prover component (computation of fabtoken input on trusted peer)
  - Client-SDK

**Verifier Component**
- (custom validation - VSCC - & commitment - transaction processor)
  - Channel committing peers

**Ordering Service**
- FabToken transaction **broadcast**
- FabToken transaction **deliver**

**Client Wallet Library**
- [cwLib]

**Prover component**
- (computation of fabtoken input on trusted peer)

**Application**
- (End-user transactor/issuer)
FabToken exhibits a modular architecture to accommodate various privacy levels.
Token information flow by example

Token issuer

Token user

Prover peer:
Trusted by the client;
Client proof computation

Prover peer:
Trusted by the client;
Client proof computation

Application
(End-user transactor/issuer)

Prover component
(computation of fabtoken input on trusted peer)

Committing Peers:
Trusted by the network
Transaction validation

Fabric Ordering Service

Client Wallet Library [cwLib]
(constructs & submits token transactions)

Client-SDK

Prover component
(computation of fabtoken input on trusted peer)
Token information flow by example

1. Issue request

2. Issue request proof

Prover peer: Trusted by the client; Client proof computation

Prover component (computation of fabtoken input on trusted peer)

Committing Peers: Trusted by the network
Transaction validation

Application (End-user transactor/issuer)
Token information flow by example

Token issuer

1. Issue request
2. Issue request proof
3. Submit itx with proof

Prover peer: Trusted by the client; Client proof computation

Prover peer: Trusted by the client; Client proof computation

Token user

Application (End-user transactor/issuer)

Client Wallet Library [cwLib]
(con structs & submits token transactions)

Prover component
(computation of fabtoken input on trusted peer)

Client-SDK

Committing Peers: Trusted by the network
Transaction validation

Fabric Ordering Service
Token information flow by example

1. Issue request
2. Issue request proof
3. Submit itx with proof

Prover peer:
- Trusted by the client;
- Client proof computation

Token user

Token issuer

Prover peer:
- Trusted by the client;
- Client proof computation

Verifier Component
- (custom validation - VSCC - & commitment - transaction processor)

Fabric Ordering Service

Committing Peers:
- Trusted by the network
- Transaction validation

Channel committing peers

1. Issue request
2. Issue request proof
3. Submit itx with proof
4. Deliver ordered transaction including itx
Token information flow by example

5. List tokens request

6. List of BNKA tokens

Prover peer:
Trusted by the client;
Client proof computation

Committing Peers:
Trusted by the network
Transaction validation

Application
(End-user transactor/issuer)

Client Wallet Library [cwLib]
(constructs & submits token transactions)

Prover component
(computation of fabtoken input on trusted peer)

Client-SDK

5. List tokens request

6. List of BNKA tokens

Prover peer:
Trusted by the client;
Client proof computation

Committing Peers:
Trusted by the network
Transaction validation

Application
(End-user transactor/issuer)

Client Wallet Library [cwLib]
(constructs & submits token transactions)

Prover component
(computation of fabtoken input on trusted peer)

Client-SDK
Token information flow by example

Token issuer

Token user

Token user

Prover peer:

Trusted by the client;
Client proof computation

Application
(End-user transactor/issuer)

Client Wallet
Library [cwLib]
(con structs & submits token transactions)

Prover component
(computation of fabtoken input on trusted peer)

Client-SDK

Commiting Peers:
Trusted by the network
Transaction validation
Token information flow by example

Token issuer

Token user

Prover peer: 
*Trusted by the client;*
Client proof computation

7. Transfer request

8. Transfer request proof

9. Submit tx with proof

Application
(End-user transactor/issuer)

Client Wallet Library [cwLib]
(constructs & submits token transactions)

Prover component
(computation of fabtoken input on trusted peer)

Client-SDK

Committing Peers: 
*Trusted by the network*
Transaction validation

Fabric Ordering Service

Token information flow by example

Token issuer

Token user

Prover peer: Trusted by the client; Client proof computation

7. Transfer request

8. Transfer request proof

9. Submit ttx with proof

Verifier Component (custom validation - VSCC - & commitment - transaction processor)

Committing Peers: Trusted by the network
Transaction validation

10. Deliver ordered transaction including ttx
FabToken exhibits a modular architecture to accommodate various privacy levels.
Client wallet library

- A library to expose user-friendly token functionalities to end user/application developer
- https://jira.hyperledger.org/browse/FAB-11153
FabToken exhibits a modular architecture to accommodate various privacy levels.
Prover peer

- A peer **trusted** by the client to
  - Perform computation on the client’s behalf
  - Maintain confidential information on the client’s behalf
  - Respond properly to client’s ledger queries (status of transactions, list of tokens)

- Implemented as a GRPC service of a peer

- Why do we need it?
  - Client needs ledger access to compute issue, transfer proofs
  - Proof computation (esp. in the privacy-preserving case) often requires heavy computation that we want to offload to a common code base

- Currently in [https://jira.hyperledger.org/browse/FAB-11149](https://jira.hyperledger.org/browse/FAB-11149)
FabToken exhibits a modular architecture to accommodate various privacy levels.
Token Management System

• An abstraction to represent token management low-level operations (i.e., proof computation & verification)

• Currently as parts of two epics:
  • https://jira.hyperledger.org/browse/FAB-11149
  • https://jira.hyperledger.org/browse/FAB-11144
FabToken exhibits a modular architecture to accommodate various privacy levels

Currently in https://jira.hyperledger.org/browse/FAB-11144
Transaction processing flow @Committing peer

**Validation phase** is served via validation system chaincodes & can take place *in parallel* for different transactions; transaction which successfully pass the validation checks we call **valid**.

**Commit phase** is served via transaction processors & takes place *sequentially* for **valid** transactions in a block after the validation phase completes for all block’s transactions.
More Diagrams
Token system bootstrapping on a given channel

• Token system stakeholders agree on the configuration of the token system & compile this into a config file, config ⇒ tools can be used to convert config into protobuf messages

• config (or protobuf equivalent) is passed to the channel stakeholders that deploy the token system using chaincode lifecycle operations, i.e.,
  • A namespace would be reserved for the token system & activated
  • config would serve as the validation parameter for validation of transactions that aim to modify state with the token system’s namespace (stored in the LSCC table)

• The peer retrieves config from the ledger to:
  • serve queries to the client (prover peer) for that channel
  • setup validator/committer components for transaction validation/commit (committing peer)

• Trust assumptions:
  • Channel stakeholders are trusted to propagate config for the system’s deployment
  • Token stakeholders are responsible for choosing properly parameters in config
  • Clients trust their prover peers for i) setup, ii) transaction construction, iii) queries on ledger state
Token system bootstrapping on a given channel

• Related JIRAs for peer setup:
  • https://jira.hyperledger.org/browse/FAB-11285
  • https://jira.hyperledger.org/browse/FAB-11169

• Related JIRAs for client setup:
  • https://jira.hyperledger.org/browse/FAB-11286
Token system setup

Token Stakeholder

ClientLib

Ordering Service

Committing Peer

Configuration file with FabToken config (e.g., yml/json)

LSCC Tx to "deploy" FabToken including FabToken config

LSCC Tx to "deploy" FabToken including FabToken config

LSCC Tx to "deploy" FabToken including FabToken config

- Validate FabToken "deploy" transaction
- Update LSCC table with <token, config> entry
Client setup flow

EndUser

ClientLib

Configuration file (yaml/json) with:
- peer PR
- channel identifier CH
- client credential info CRD

Peer PR

Ledger CH

Request FabToken config info for channel CH

Check permissions of requestor

Request FabToken config

config

config

Setup using config & CRD
Token issue

**Issuer**

Issue 20 USD tokens & assign to Bob

**ClientLib**

(GRPC) Request issue of 20 USD tokens to Bob

(GRPC) issProof

Create & Submit FabToken tx with arguments "issue issProof"

**Peer**

Construct Token transaction TX

**ClientSDK**

Get validation/commit parameters from LSCC

config

**Ordering Service**

Validate TX with config

**Ledger**

Setup token-validator with config, run validation on "issue TX, issProof"

**Validator**

Ok

Wait till transaction validations of the block complete

Commit TX with config

**Committer**

Setup token-commmitter with config, commit TX, issProof

Ok

Update Ledger
**Abstraction/dependency diagram**

### Token Setup
- Define configuration
- Protobuf messages
- Converter from JSON/YAML to protobuf
- Construct config tx
- Validation of fbt config tx: Setup custom validation component
- Commitment of fbt config tx: Setup custom tx processor
- Setup underlying TMS

### Token Issue Support (similar for Token Transfer/Redeem)
- Define fabtoken tx protobuf message
- Allow client fbt lib to produce fabtoken tx for issue tokens
- Allow client fbt lib to acquire tms issue request
- Allow client fbt lib to submit a tms issue req in a fbt tx
- Allow peer to validate fabtoken issue transactions
- Enhance tms with tms issue request validation & commitment
- Allow peer to commit fabtoken issue transactions
- Build a grpc peer service to provide tms issue request upon client request
- Enhance tms with issue request construction
- Extend client sdk to make fabtoken txs

### List Tokens Query Support
- Define fabtoken tx protobuf message
- Allow client fbt lib to produce fabtoken query to list tokens
- Enhance the peer grpc service to respond to list token queries
- Enhance tms with list tokens command
- Allow client fbt lib to acquire tms issue
- Allow client fbt lib to acquire tms issue request
- Enhance tms with issue request construction
- Allow peer to submit a tms issue req in a fbt tx
- Allow peer to validate fabtoken issue transactions
- Enhance tms with tms issue request validation & commitment
- Allow peer to commit fabtoken issue transactions
- Build a grpc peer service to provide tms issue request upon client request
- Enhance tms with issue request construction
- Extend client sdk to make fabtoken txs

### Legend
- **Client Domain**
  - Fbt Client Lib
  - Client sdk
  - GRPC service
- **Committing peer**
- **TMS**