FabZK: Supporting Privacy-Preserving, Auditable Smart Contracts in Hyperledger Fabric

Hui Kang (IBM), Ting Dai (NCSU), Nerla Jean-Louis (IBM), Shu Tao (IBM), Xiaohui Gu (NCSU)

NC STATE UNIVERSITY

IBM Research

Blockchain

- An immutable ledger for recording transactions, maintained within a distributed network
  - Each node has a copy of the ledger
  - Consensus protocol to order transactions
  - Transactions are grouped into blocks and chained together
- Benefits: transparency, security, traceability
- Existing platforms can be categorized into two types
  - Permission-less, e.g., bitcoin, Ripple, Stellar
  - Permissioned, e.g., Zcash, Ethereum, Hyperledger Fabric

Blockchain

- An immutable ledger for recording transactions, maintained within a distributed network
  - Each node has a copy of the ledger
  - Consensus protocol to order transactions
  - Transactions are grouped into blocks and chained together
- Benefits: transparency, security, traceability
- Existing platforms can be categorized into two types
  - Permission-less, e.g., bitcoin, Ripple, Stellar
  - Permissioned, e.g., Zcash, Ethereum, Hyperledger Fabric

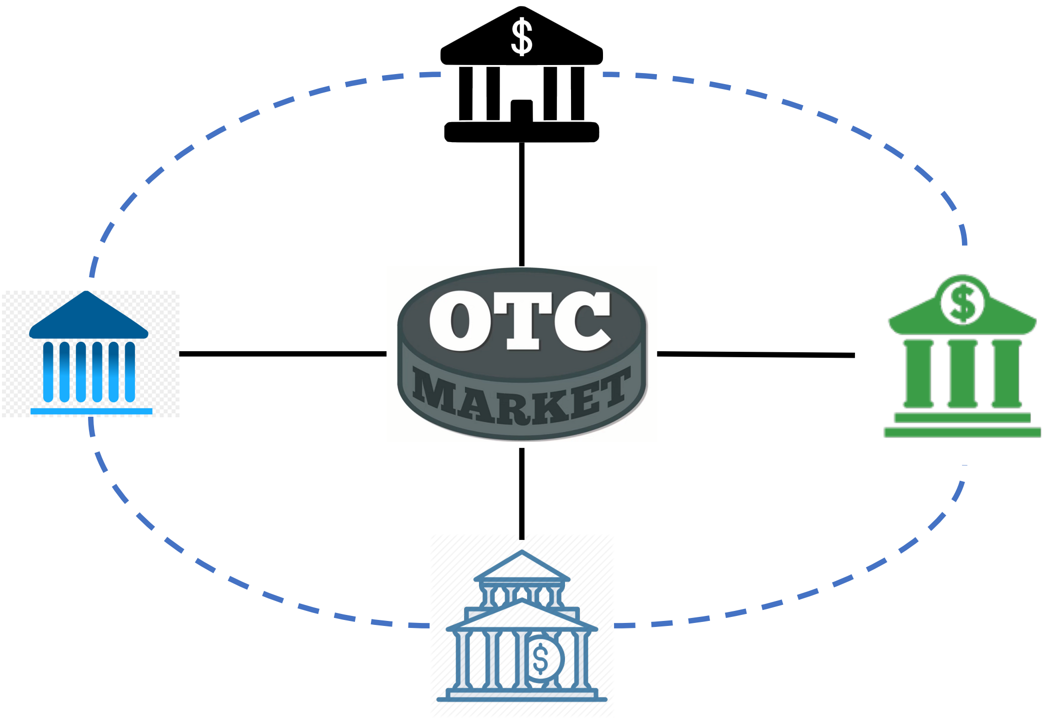
Lack of auditable privacy-preserving transactions

Hyperledger Fabric

- **Open source enterprise-grade distributed ledger platform**
- **Hosted by Linux Foundation**
- **170+ contributors world wide**
- **IBM Blockchain platform on IBM Cloud, AWS, and Azure**

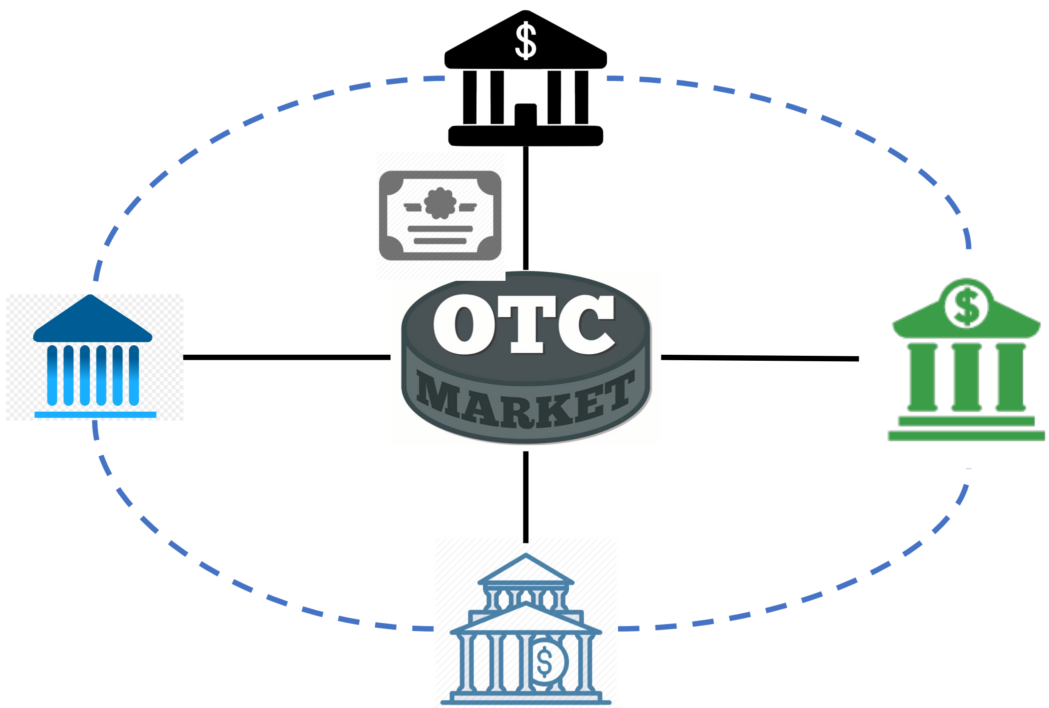
Motivating Example

- Running example: over-the-counter (OTC) platform



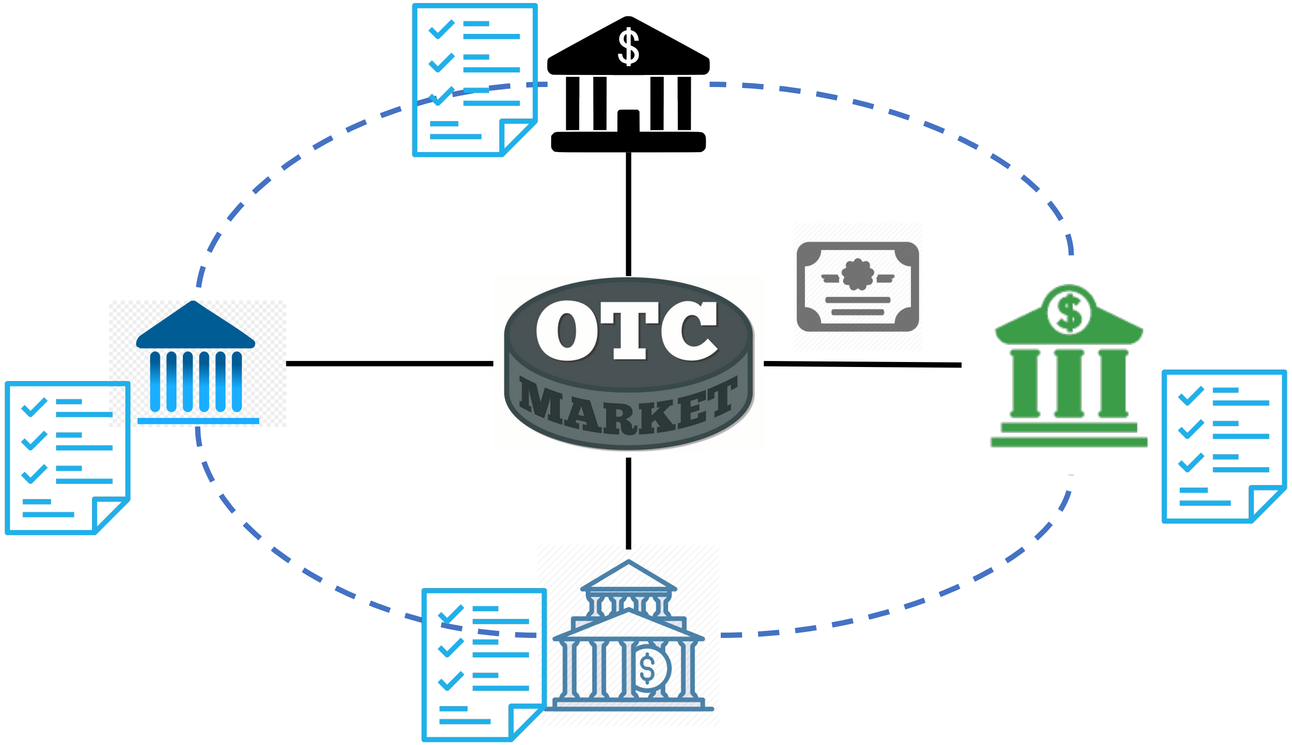
Motivating Example

- Running example: over-the-counter (OTC) platform

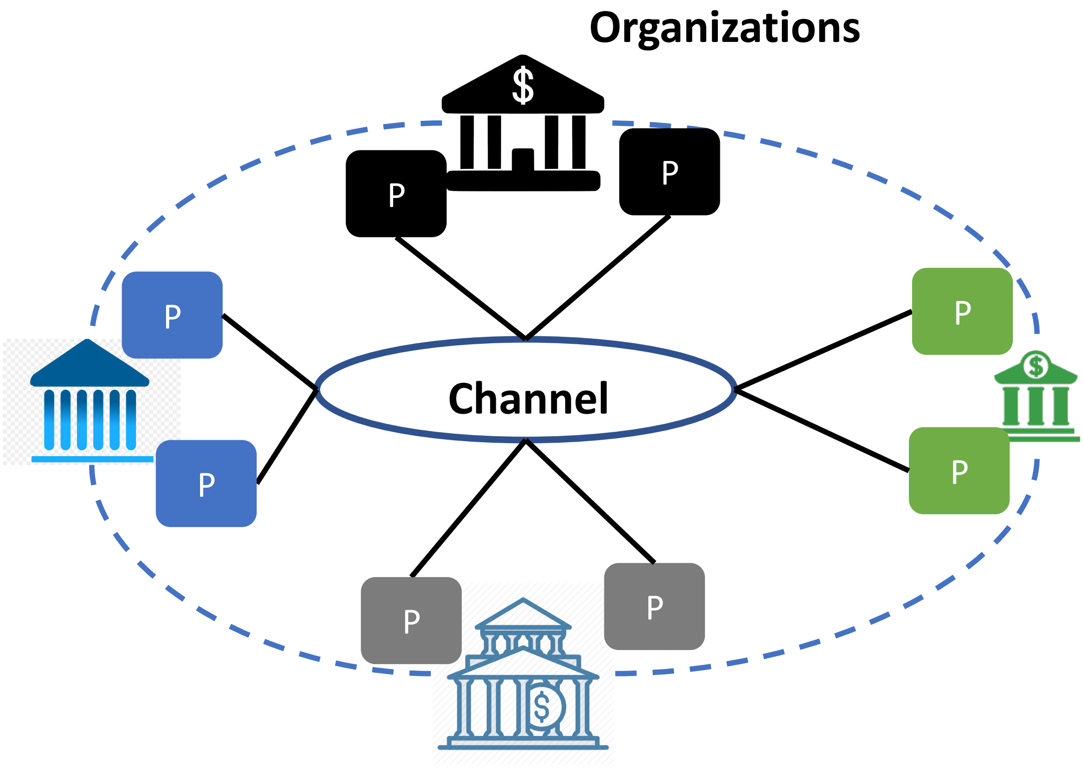


Motivating Example

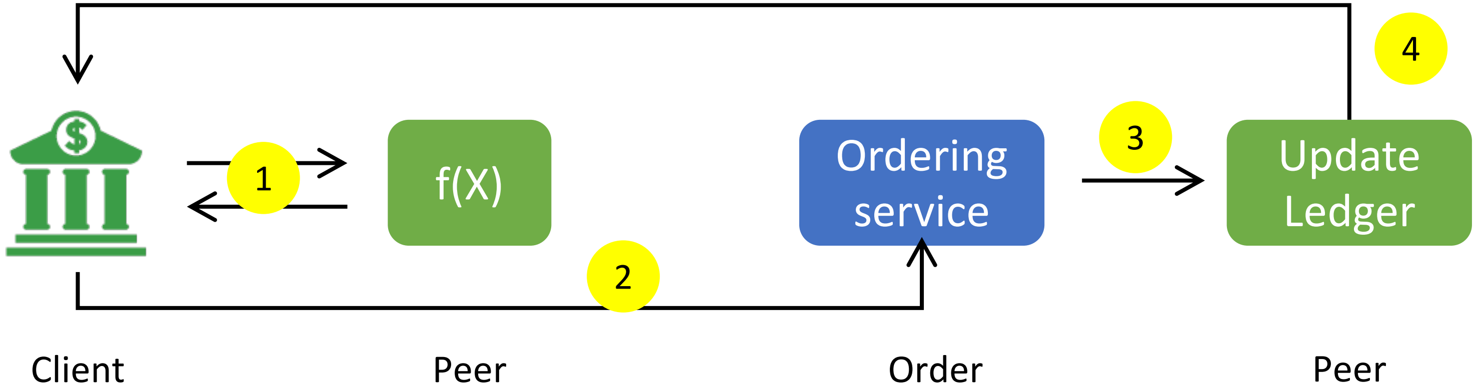
- Running example: over-the-counter (OTC) platform



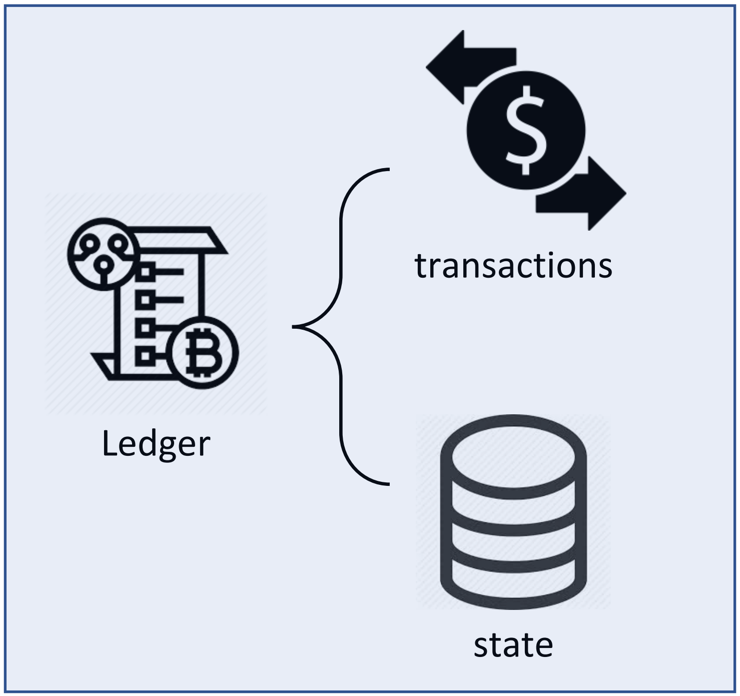
Implementation in Fabric



Transaction Flow in Fabric

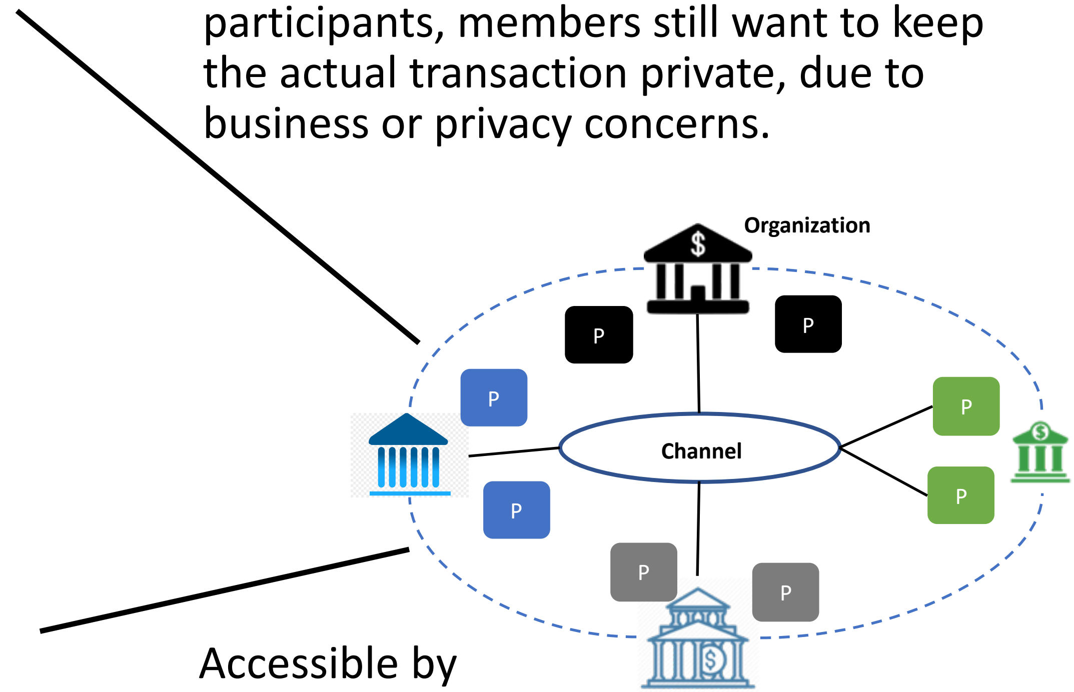


Privacy in Hyperledger Fabric (Motivation)



Most Important Business Resources

- Although consortium contains a certain degree of knowledge about the channel participants, members still want to keep the actual transaction private, due to business or privacy concerns.



Accessible by
all admitted organizations

Transfer transaction

Spending org: **A**
Receiving org: **B**
Transfer amount: **100**


Auditor

- 100 + (-100) = 0
- Transaction graph revealed

Standard Fabric
(No privacy, auditable)

Transfer transaction

Spending org: **A**
Receiving org: **B**
Transfer amount: **100**


Auditor

- 100 + (-100) = 0
- Transaction graph revealed

Standard Fabric
(No privacy, auditable)

Transfer transaction

Spending org: **A**
Receiving org: **B**
Transfer amount: **H(100)**

- H(100), H(-100) are non-auditable
- Transaction graph revealed

amount concealed
(Privacy, non-auditable)

Transfer transaction

Spending org: **A**
Receiving org: **B**
Transfer amount: **100**


Auditor

- 100 + (-100) = 0
- Transaction graph revealed

Standard Fabric
(No privacy, auditable)

Transfer transaction

Spending org: **A**
Receiving org: **B**
Transfer amount: **H(100)**

- H(100), H(-100) are non-auditable
- Transaction graph revealed

Identity and amount concealed
(Privacy, non-auditable)

Transfer transaction

Spending org: **F(A)**
Receiving org: **F(B)**
Transfer amount: **F(100)**

- F (100) + F(-100) + F(0) + … = 0
- Transaction graph concealed

Identity and amount concealed
(**Privacy, Auditable**)

Transfer transaction

Spending org: **A**
Receiving org: **B**
Transfer amount: **100**

Auditor

- 100 + (-100) = 0
- Transaction graph revealed

Standard Fabric
(No privacy, auditable)

Transfer transaction

Spending org: **A**
Receiving org: **B**
Transfer amount: **H(100)**

- H(100), H(-100) are non-auditable
- Transaction graph revealed

Identity and amount concealed
(Privacy, non-auditable)

Transfer transaction

Spending org: **F(A)**
Receiving org: **F(B)**
Transfer amount: **F(100)**

- F (100) + F(-100) + F(0) + ... = 0
- Transaction graph concealed

Identity and amount concealed
(**Privacy, Auditable**)

Q: How to combine public auditability with privacy?

A: Using Zero-knowledge asset transfer

This Talk

- **FabZK**: Auditable, zero-knowledge asset transfer in Hyperledger Fabric
  - Theoretical model via proven cryptographic primitives
  - FabZK design and architecture
  - Computation Parallelism
  - Performance evaluation

Auditable, Zero-Knowledge Transfer

Auditable, Zero-Knowledge Transfer

- TX_m: organization A sends u=100 shares of asset to organization B

Ledger on Fabric

Transaction ID	Organization A	Organization B
1		
m	-100	+100

- ***Pedersen commitment:*** a commitment scheme that encrypts a value, with the ability to reveal it later

$$Com(u,r) = g^u h^r$$

Auditable, Zero-Knowledge Transfer

- TX_m: organization A sends u=100 shares of asset to organization B

Ledger on Fabric

Transaction ID	Organization A	Organization B
1		
m	🔒 **Com(-100, r1)**	🔒 **Com(+100, r2)**

Auditable, Zero-Knowledge Transfer

- TX_m: organization A sends u=100 shares of asset to organization B

Ledger on Fabric

Transaction ID	Organization A	Organization B
1		
m	**Com(-100, r1)**	**Com(+100, r2)**

- ***Homomorphism of Pedersen commitment:***

$$\because \quad \prod_{i=1}^{n} \mathrm{Com}_i = Com(u1, r1)(Com(u2, r2) \cdots = Com(u1 + u2 \cdots, r1 + r2 \ldots) = g^{\sum u} h^{\sum r}$$

$$\therefore \quad \sum_{i=1}^{n} u_i = 0 \qquad \sum_{i=1}^{n} r_i = 0 \qquad \textbf{prove} \qquad \prod_{i=1}^{n} \mathrm{Com}_i = g^0 h^0 = 1$$

Auditable, Zero-Knowledge Transfer

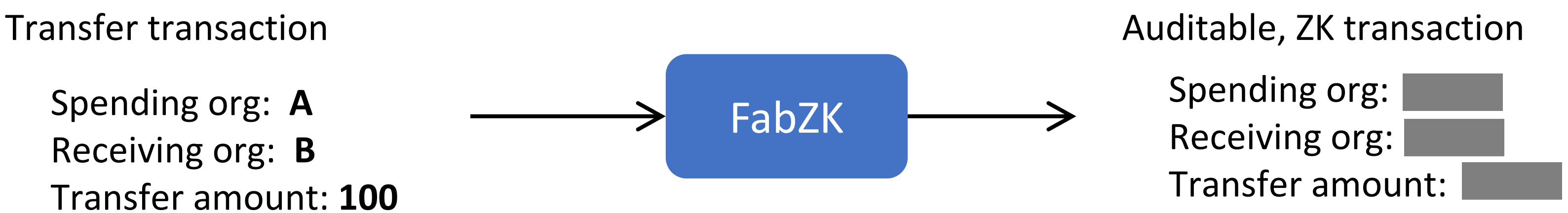
- TX$_m$: organization A sends *u*=100 shares of asset to organization B

Ledger on Fabric

Transaction ID	Organization A	Organization B
1		
m	Com(-100, r1)	Com(+100, r2)

- ***Proof of Balance***: the auditor verifies the balance of individual transactions, $\prod_{i=1}^{n} Com = 1$
- Privacy is preserved as the actual transaction amount is not exposed to the auditor

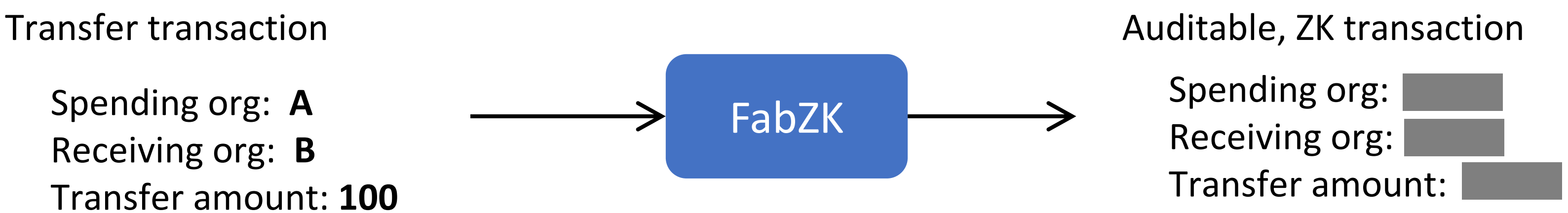
Overview



Plaintext transaction

Privacy-preserving, auditable transaction on ledger

Overview



Plaintext transaction

Privacy-preserving, auditable transaction on ledger

- Privacy-preserving
  - Pedersen commitment
  - Anonymize the identities of the spending and the receiving organization
- Auditable
  - Non-interactive zero-knowledge (NIZK) proof

Anonymity

- The identity of organization A and B (aka., transaction graph) is exposed

Transaction ID	Organization A	Organization B
1		
m	**Com(-100, r1)**	**Com(+100, r2)**

Anonymity

- The identity of organization A and B (aka., transaction graph) is exposed

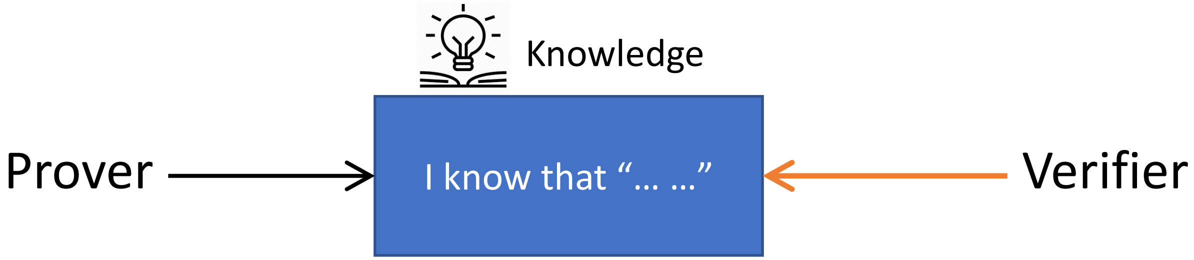
Transaction ID	Organization A	Organization B
1		
m	**Com(-100, r1)**	**Com(+100, r2)**

Include the commitments of all organizations in the transaction record

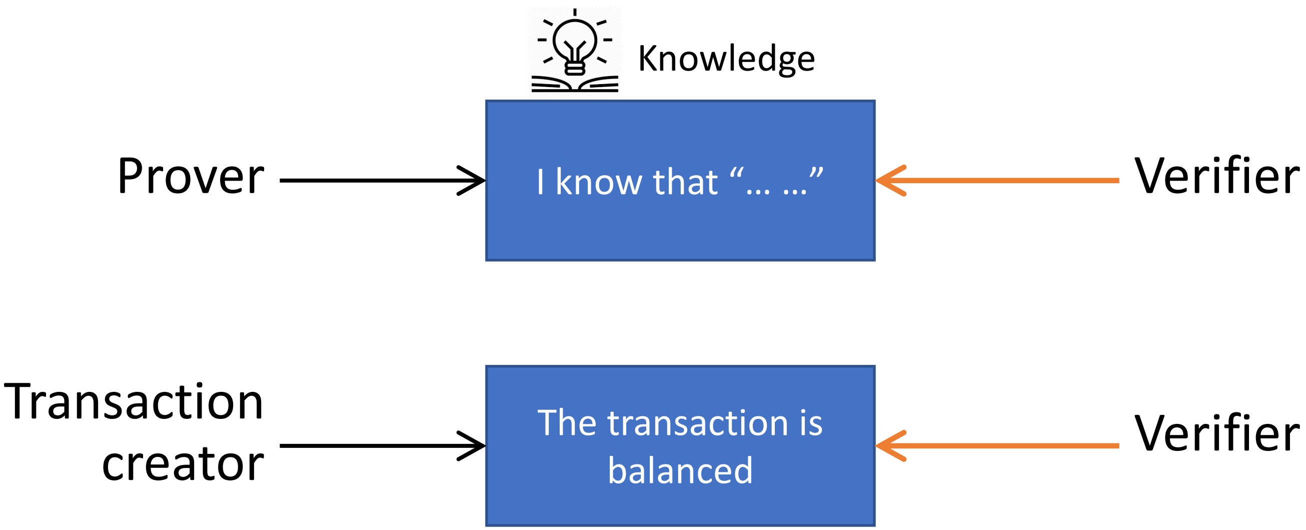
Transaction ID	Organization A	Organization B	Organization C	Organization D
1				
m	**Com(-100, r1)**	**Com(+100, r2)**	**Com(0, r3)**	**Com(0, r4)**

Commitments are indistinguishable to outsiders, so the transaction graph is concealed

Non-interactive Zero-Knowledge Proofs



Non-interactive Zero-Knowledge Proofs



m	**Com(-100, r1)**	**Com(+100, r2)**	**Com(0, r3)**	**Com(0, r4)**

Proof of Balance $\sum_{i=1}^{n} u_i = 0$ $\sum_{i=1}^{n} r_i = 0$ **prove** $\prod_{i=1}^{n} \mathrm{Com}_i = g^0 h^0 = 1$

- A transaction row is created by the spending organization

Transaction ID	Organization A	Organization B	Organization C	Organization D
1				
m	**Com(-100, r1)**	**Com(+100, r2)**	**Com(0, r3)**	**Com(0, r4)**

- A transaction row is created by the spending organization

Transaction ID	Organization A	Organization B	Organization C	Organization D
1				
m	**Com(-100, r1)**	**Com(+100, r2)**	**Com(0, r3)**	**Com(0, r4)**

A malicious organization may steal assets from non-transactional organization

Transaction ID	Organization A	Organization B	Organization C	Organization D
1				
m	***Com(-50, r1)***	**Com(+100, r2)**	***Com(-50, r3)***	**Com(0, r4)**

$$Com(-50, r1) * Com(100, r2) * Com(-50, r3) * Com(0, r3) \ = 1$$

- A transaction row is created by the spending organization

Transaction ID	Organization A	Organization B	Organization C	Organization D
1				
m	**Com(-100, r1)**	**Com(+100, r2)**	**Com(0, r3)**	**Com(0, r4)**

A malicious organization may steal assets from non-transactional organization

Transaction ID	Organization A	Organization B	Organization C	Organization D
1				
m	***Com(-50, r1)***	**Com(+100, r2)**	***Com(-50, r3)***	**Com(0, r4)**

$$Com(-50, r1) * Com(100, r2) * Com(-50, r3) * Com(0, r3) \ = 1$$

Proof of Balance is insufficient

Proof of Correctness

- Prove the legitimacy of commitment written by the spending organization
  - Each commitment has an token generated from an organization's public key (*pk*) and private key (*sk*)

$$\text{Token} = \text{pk}^r \qquad \text{pk} = h^{\text{sk}}$$

If $\text{Token}_m \cdot g^{\text{sk} \cdot u_m} = (\text{Com}_m)^{\text{sk}}$ holds, it proves Com_m matches u_m

Proof of Correctness

- Prove the legitimacy of commitment written by the spending organization
  - Each commitment has an token generated from an organization's public key (*pk*) and private key (*sk*)

$$\text{Token} = \text{pk}^r \qquad \text{pk} = h^{\text{sk}}$$

If $\text{Token}_m \cdot g^{\text{sk} \cdot u_m} = (\text{Com}_m)^{\text{sk}}$ holds, it proves Com_m matches u_m

Transaction ID	Organization A	Organization B	Organization C	Organization D
1				
m	***Com(-50, r1)***	**Com(+100, r2)**	***Com(-50, r3)***	**Com(0, r4)**

- Organization C knows its actual transfer amount is 0

$$\text{Token}_m \cdot g^{\text{sk} \cdot u_m} \neq (\text{Com}_m)^{\text{sk}}$$

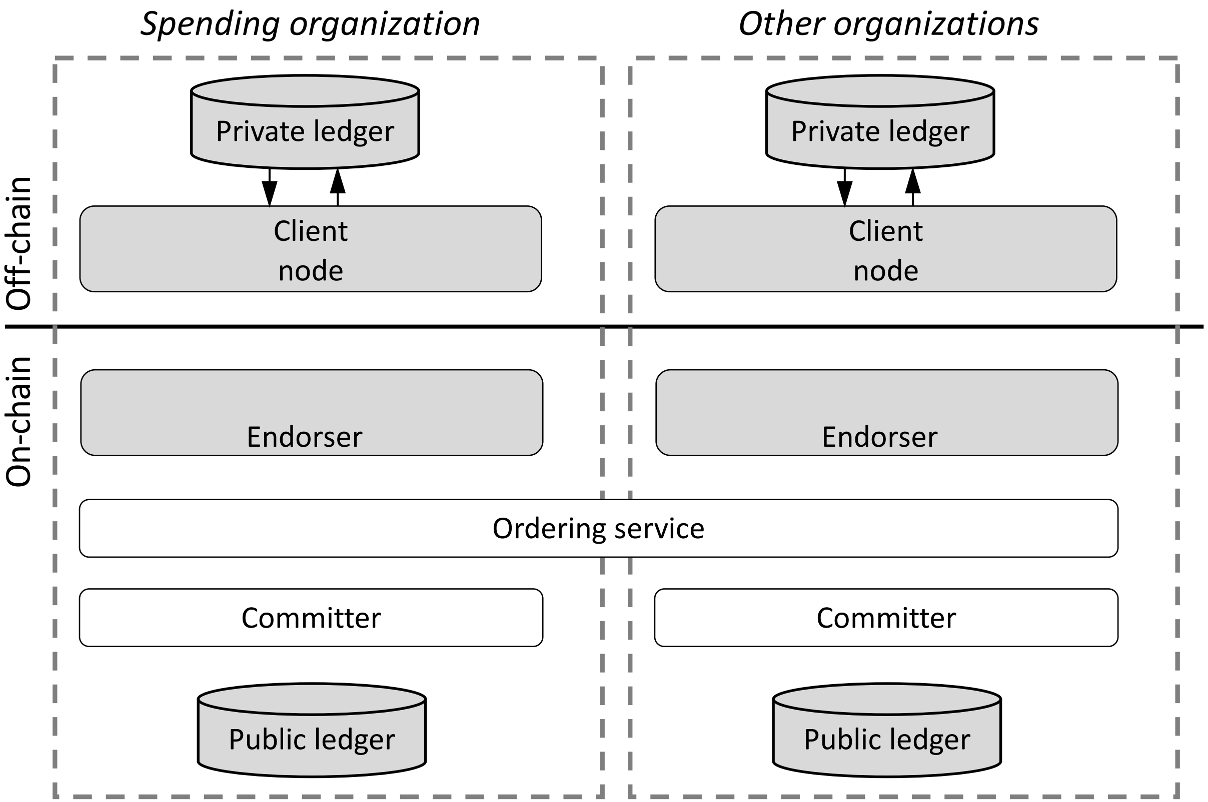
➡

- The transaction row is **invalid** due to ***Com(-50, r3)***
- Privacy is preserved; each organization verifies by itself

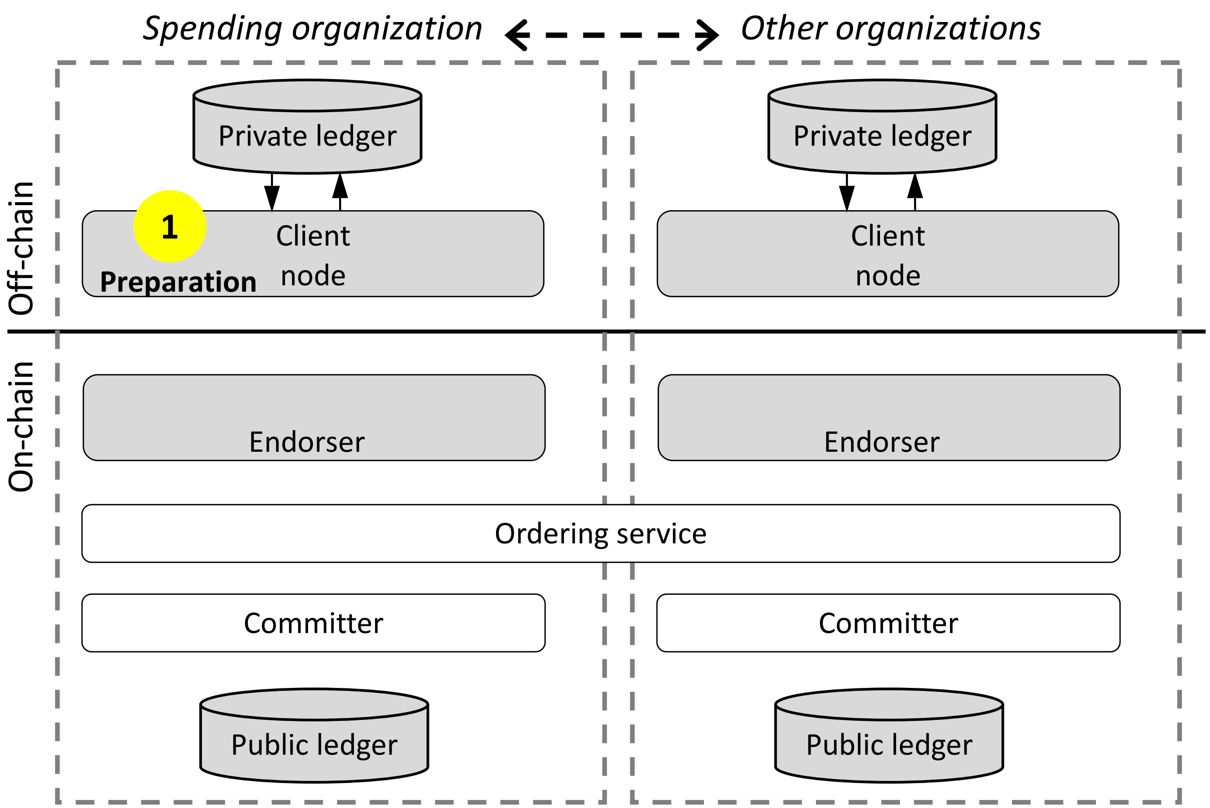
- ***Proof of Assets*** ensures the spending organization has enough assets
- ***Proof of Amount*** ensures the transaction amount is within certain range
- ***Proof of consistency*** ensures that expressions and parameters are consistent across the different proofs

- **Data dependency** in computing the five proofs
    - ***Proof of balance and proof of correctness*** does not reply on prior data, while
    - The other three proofs have to be computed based on historical data
    - An important feature to be leveraged in FabZK's implementation

FabZK Architecture

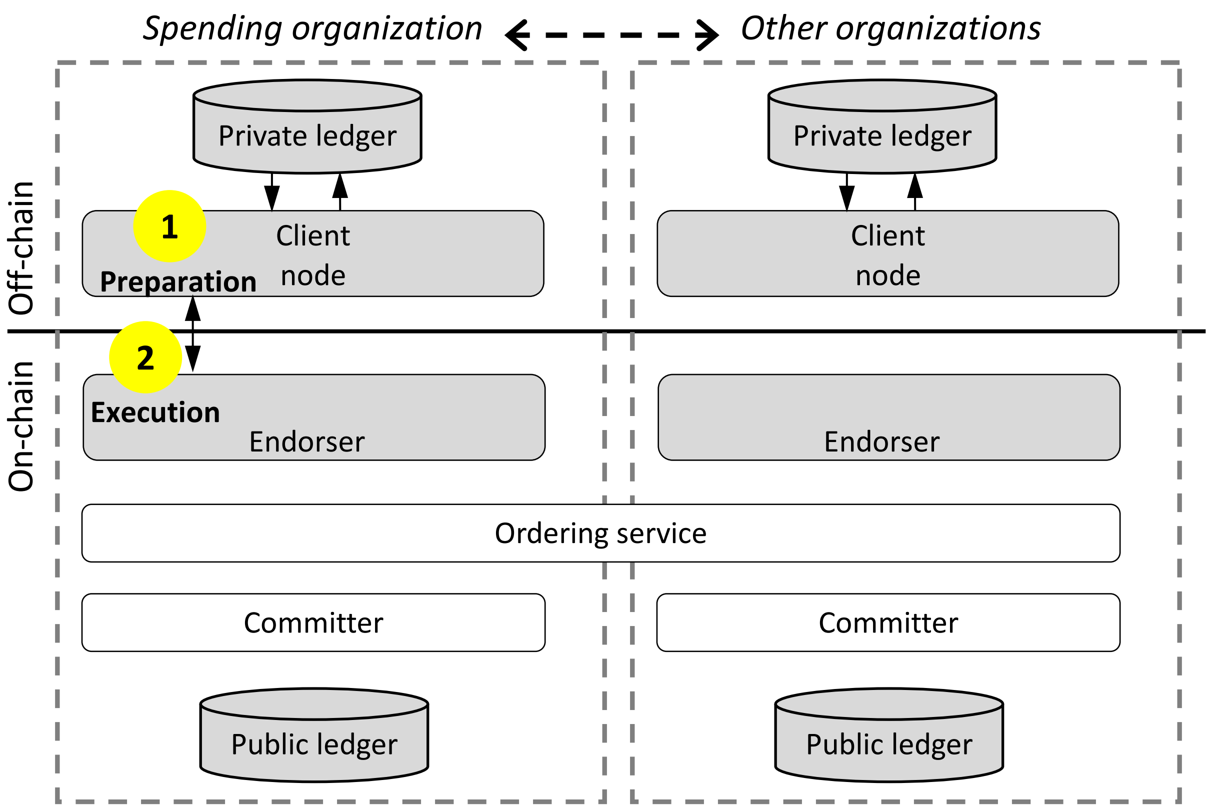


FabZK Transaction Flow by Example



1. ***Preparation*** – Prepare the transaction request in the form of *N tx amount*, and submit to the Blockchain network

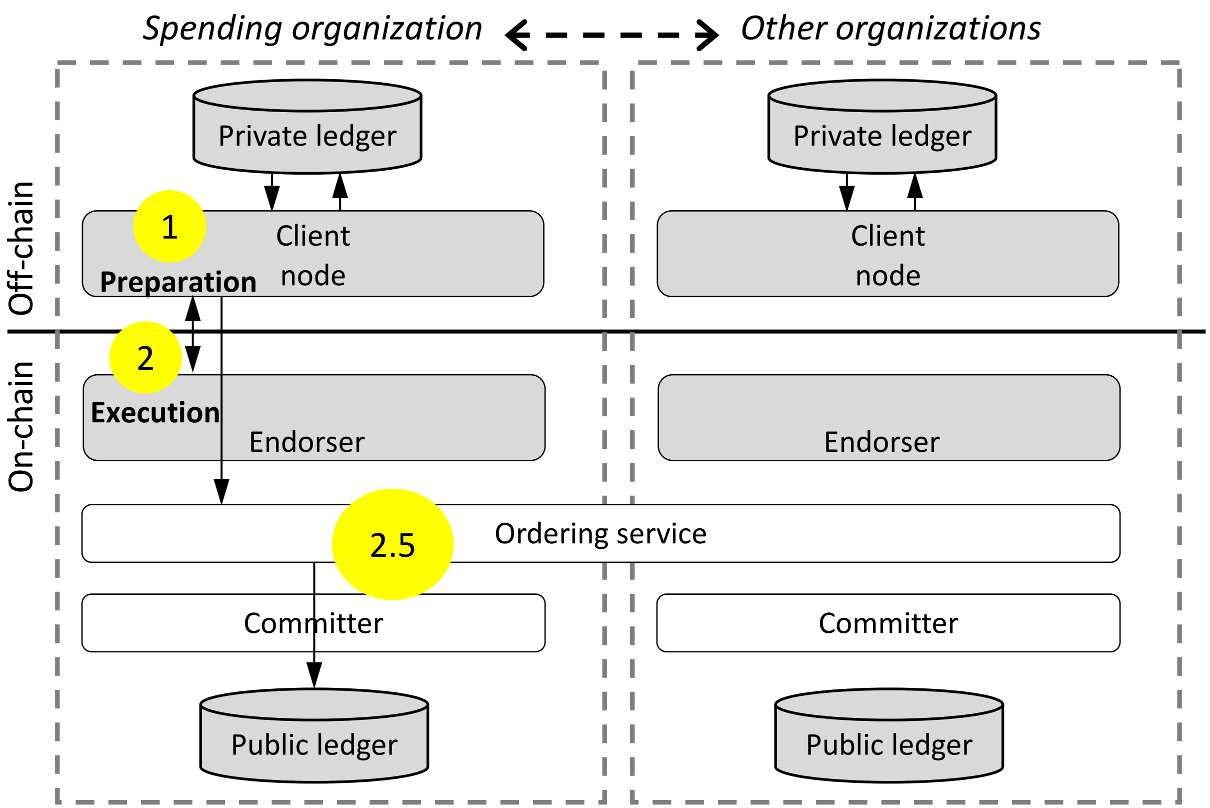
FabZK Transaction Flow by Example



1. ***Preparation*** – Prepare the transaction request in the form of *N tx amount*, and submit to the Blockchain network

2. ***Execution*** – Execute chaincode to compute *N* <Com, token> of the tx, return to client code

FabZK Transaction Flow by Example

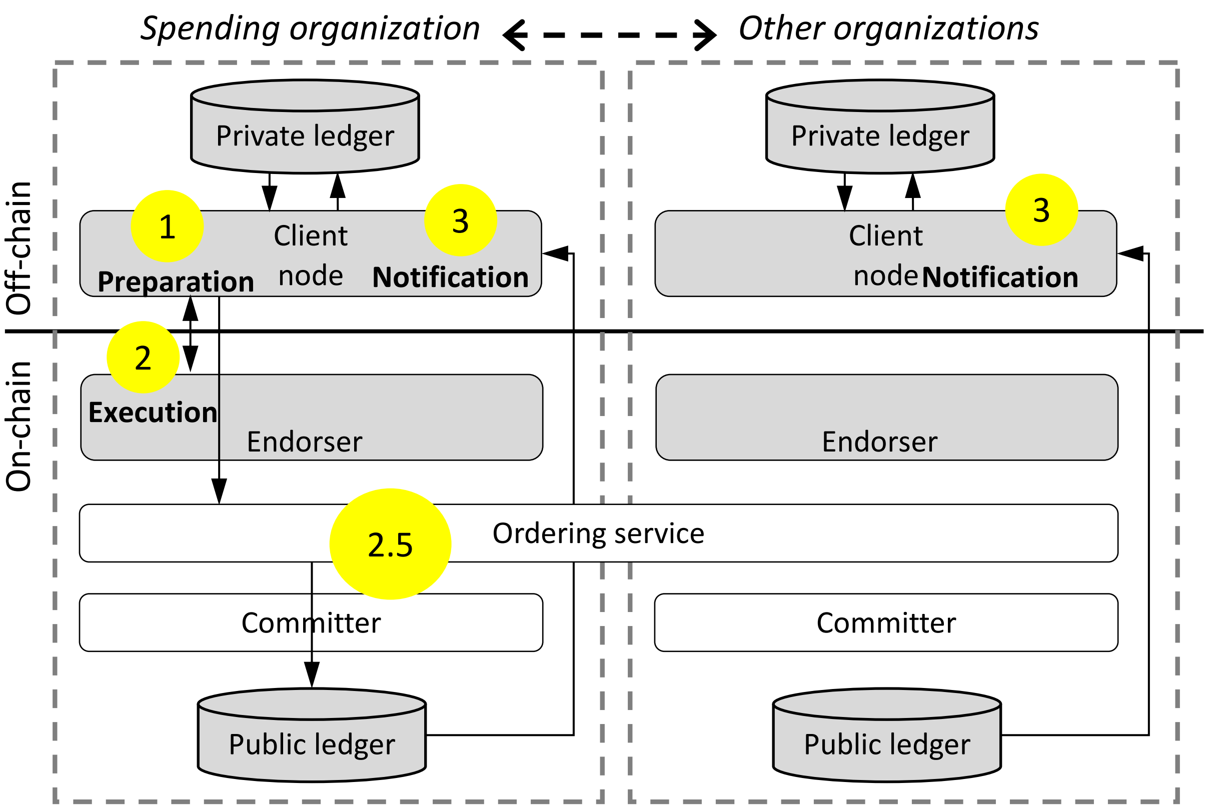


1. ***Preparation*** – Prepare the transaction request in the form of *N tx amount*, and submit to the Blockchain network

2. ***Execution*** – Execute chaincode to compute *N* <Com, token> of the tx, return to client code

2.5 Ordering and committing the *N* <Com, token> of the tx

FabZK Transaction Flow by Example



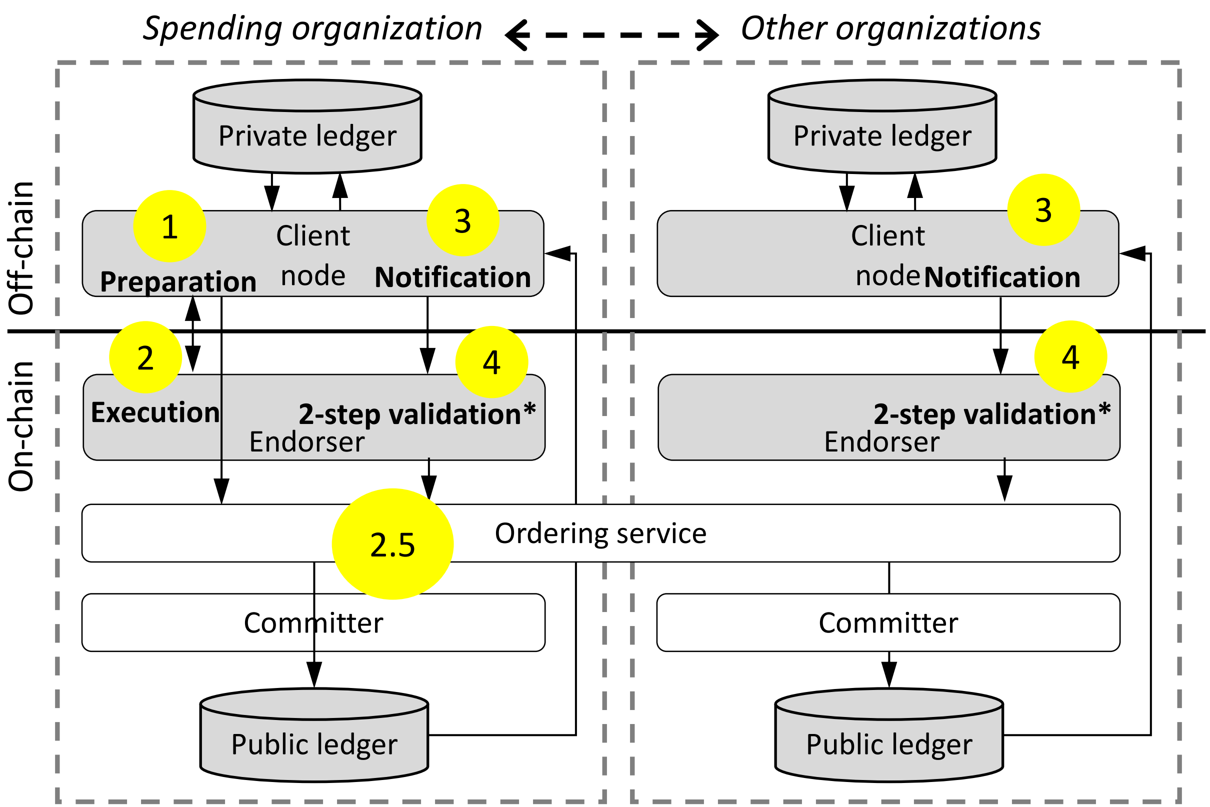
1. ***Preparation*** – Prepare the transaction request in the form of *N tx amount*, and submit to the Blockchain network

2. ***Execution*** – Execute chaincode to compute *N* <Com, token> of the tx, return to client code

2.5 Ordering and committing the *N* <Com, token> of the tx

3. ***Notification*** – client code of all organizations informed of the new committed tx

FabZK Transaction Flow by Example



1. ***Preparation*** – Prepare the transaction request in the form of *N tx amount*, and submit to the Blockchain network
2. ***Execution*** – Execute chaincode to compute *N* <Com, token> of the tx, return to client code

2.5 Ordering and committing the *N* <Com, token> of the tx

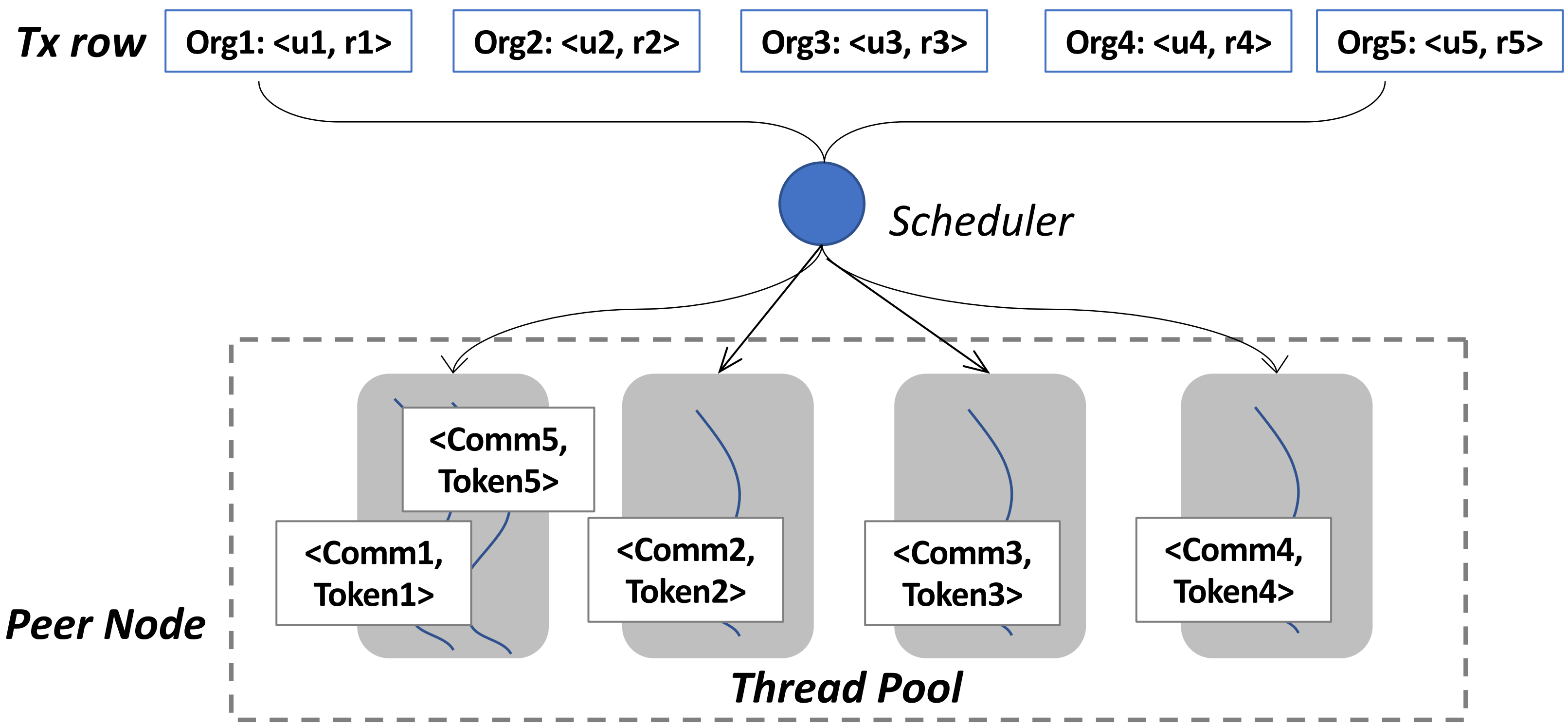
3. ***Notification*** – client code of all organizations informed of the new committed tx
4. ***2-step validation***
   4.1 Proof of balance and correctness concurrently and parallelly by all organizations
   4.2 The other 3 proofs are computed sequentially

Implementation: Computation Parallelism

- Cryptographic algorithms are compute-intensive
- To improve performance, we explore parallelizing the computation during the ***execution*** and ***two-step validation*** phases

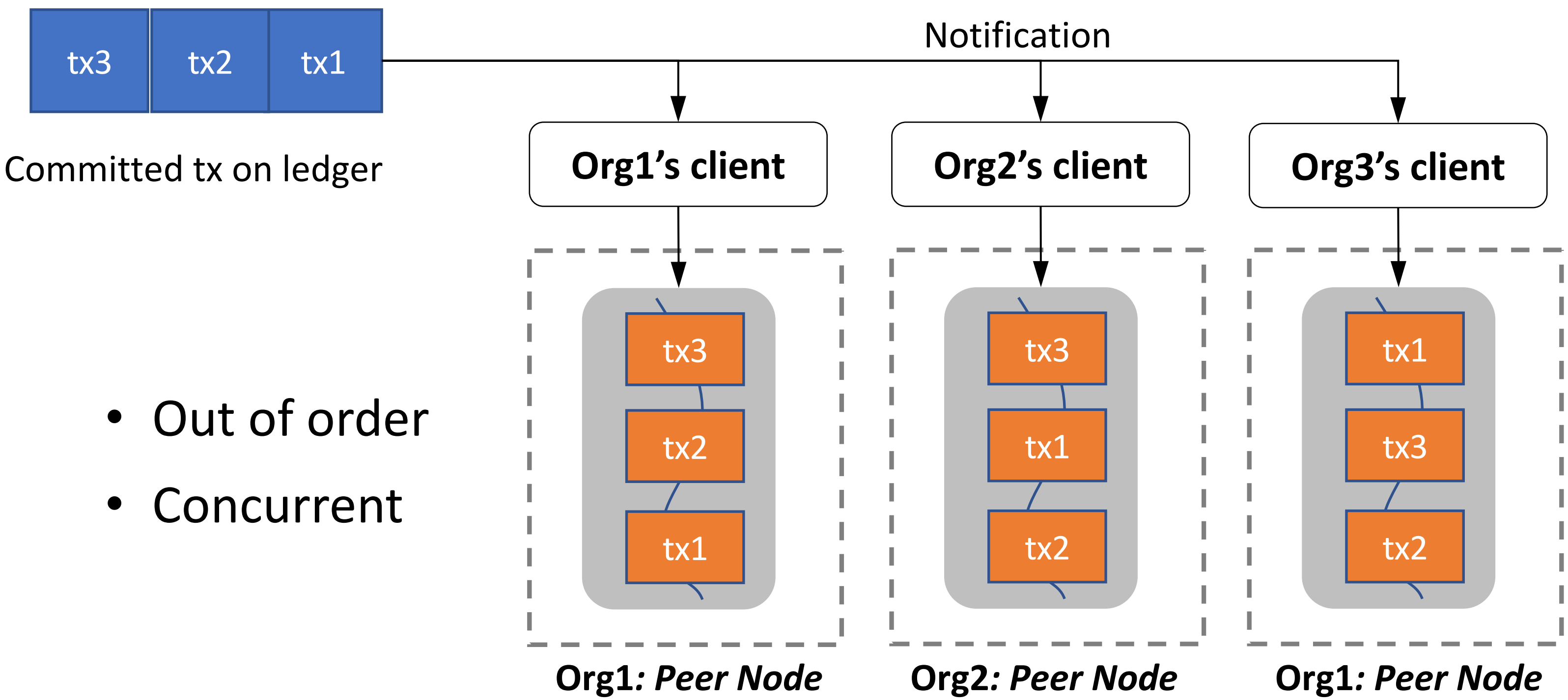
Parallelism in *Execution Phase*

- The spending organization's chaincode computes commitments and tokens for each organization



Parallelism in *Two-step Validation*

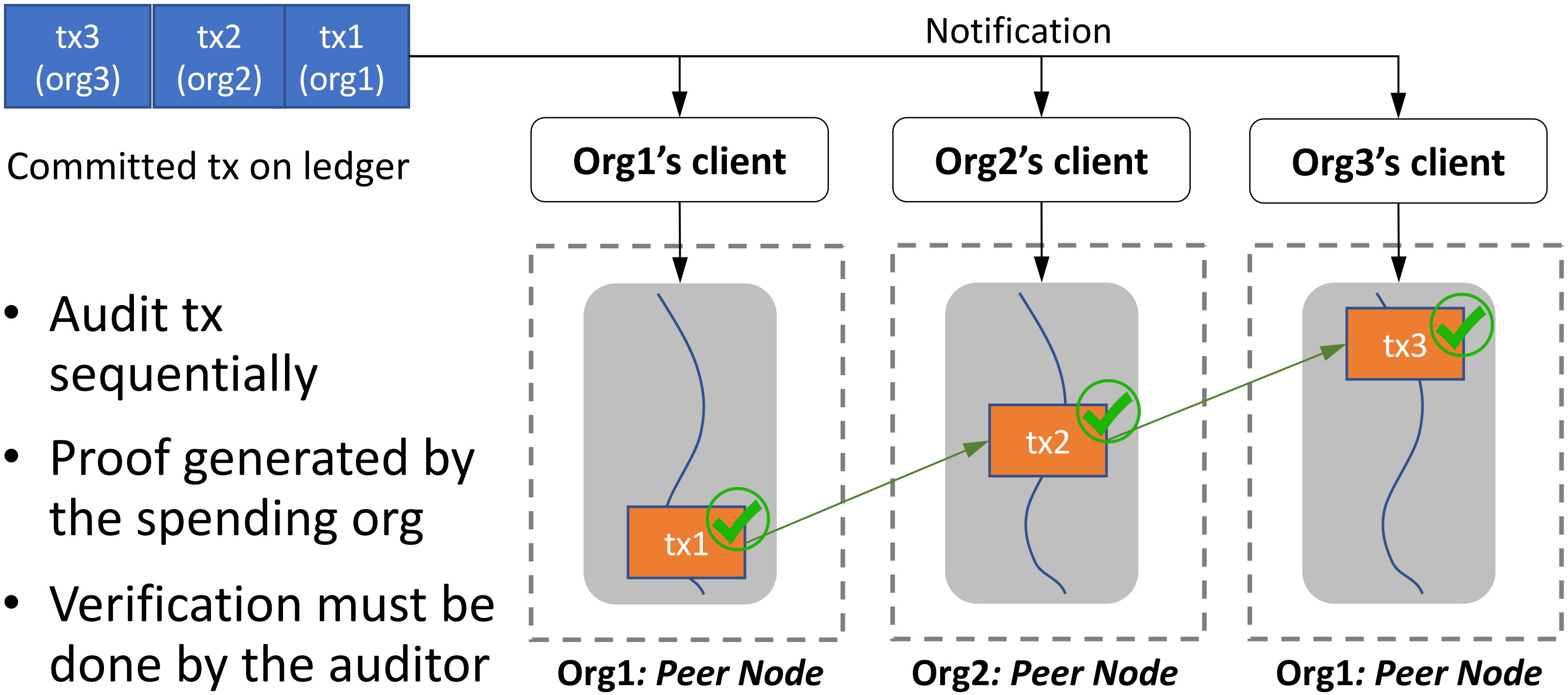
- **Step-1**: Verifying proof of balance and proof of correctness has no dependency on prior transactions



- Out of order
- Concurrent

Parallelism in *Two-step Validation* (cont'd)

- **Step-2**: computing range proof and disjunctive proof depends on prior transactions



- Audit tx sequentially
- Proof generated by the spending org
- Verification must be done by the auditor

Writing Chaincode in FabZK

- Similar to Fabric, except for using **FabZK’s API**

Writing Chaincode in FabZK

- Similar to Fabric, except for using **FabZK's API**
- A bare-minimum application in FabZK supports the following chaincode methods:
  - **Transfer**: exchange asset between organizations and write the transaction to the public ledger (`zkPutState`)
  - **Audit**: Compute the range proof and disjunctive proof for the transactions and write to the public ledger (`zkAudit`)
  - **Validation**: Invoke the 2-step validation to verify the transaction (`zkVerify` will be called twice)

Performance of Cryptographic Algorithm

- Time to ***encrypt*** the tx amount, ***generate proofs***, and ***verify proofs***
  - **Number of organizations** ranges from 1 to 20
- FabZK outperforms in encryption and proof verification
  - Further improvement by exploring scheduling schemes

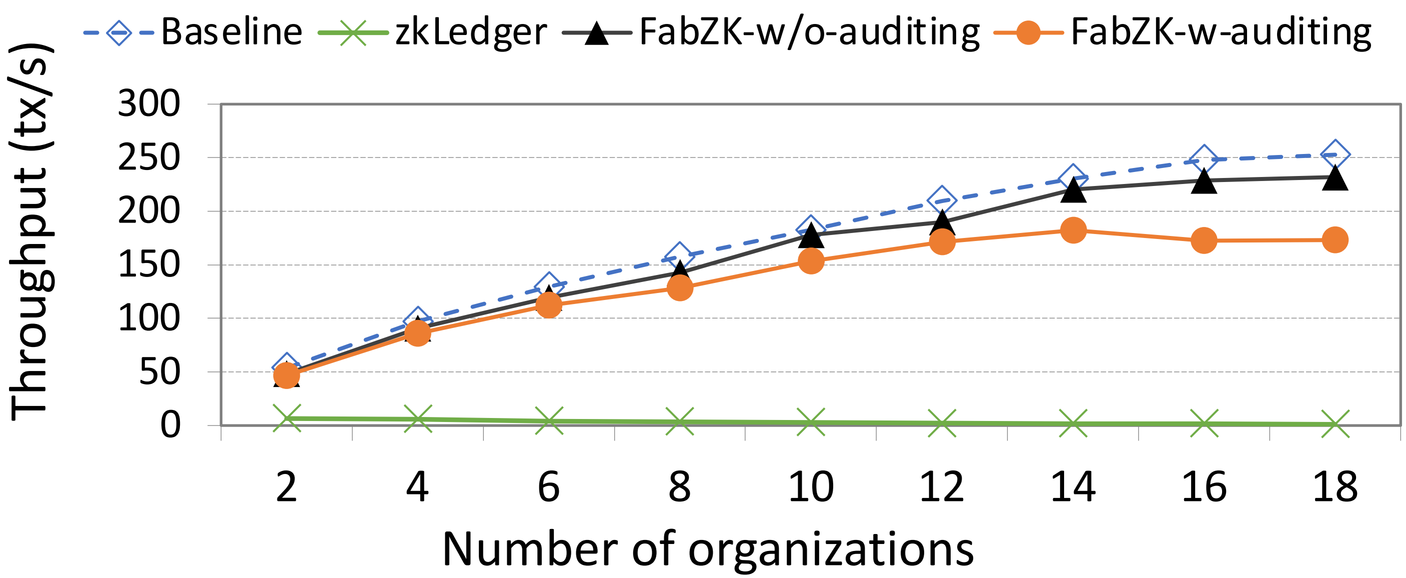
# of orgs	Data encryption		Proof generation		Proof verification	
	libsnark	FabZK	libsnark	FabZK	libsnark	FabZK
1	185.6	0.2	193.3	150.1	5.1	2.0
4	186.4	0.6	195.5	158.8	5.7	2.6
8	188.4	0.8	196.4	169.0	6.6	3.9
12	195.2	1.4	195.6	224.9	5.7	4.3
16	194.9	1.8	199.1	313.1	7.2	7.7
20	195.5	2.0	196.4	448.7	9.8	9.2

Performance of OTC Application

- **Throughput comparison:** Fabric, FabZK w/wo auditing, and zkLedger

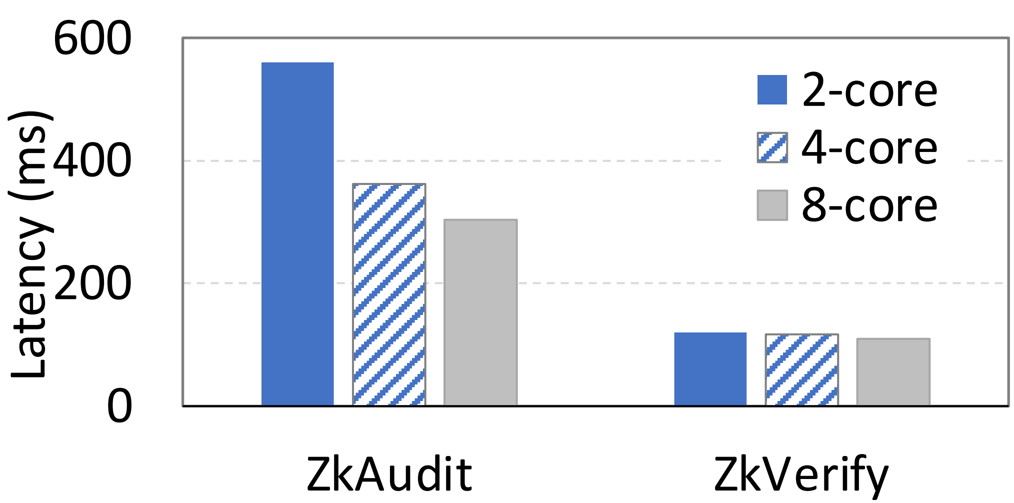
Performance of OTC Application

- **Throughput comparison:** Fabric, FabZK w/wo auditing, and zkLedger
- The overhead of FabZK from 3% to 10% w/o auditing
- Parallelized 2-step validation avoids sequential commits as in zkLedger



Performance of OTC Application (cont'd)

- Latency of auditing: time to run 2rd step of the two-step validation
  - `ZkAudit` and `ZkVerify`: compute and verify range proofs and disjunctive proofs
  - # of CPU cores from 2-core to 8-core; 4-organization network
  - Performance improved by ~50% for `ZkAudit;` minimal impact on `ZkVerify`



Conclusion

- Data privacy and auditability are critical in blockchain
- FabZK is an extension to Fabric to enable auditable privacy-preserving smart contracts
- FabZK enables auditable privacy-preserving transactions with reasonable performance cost

Thanks You!

Questions?

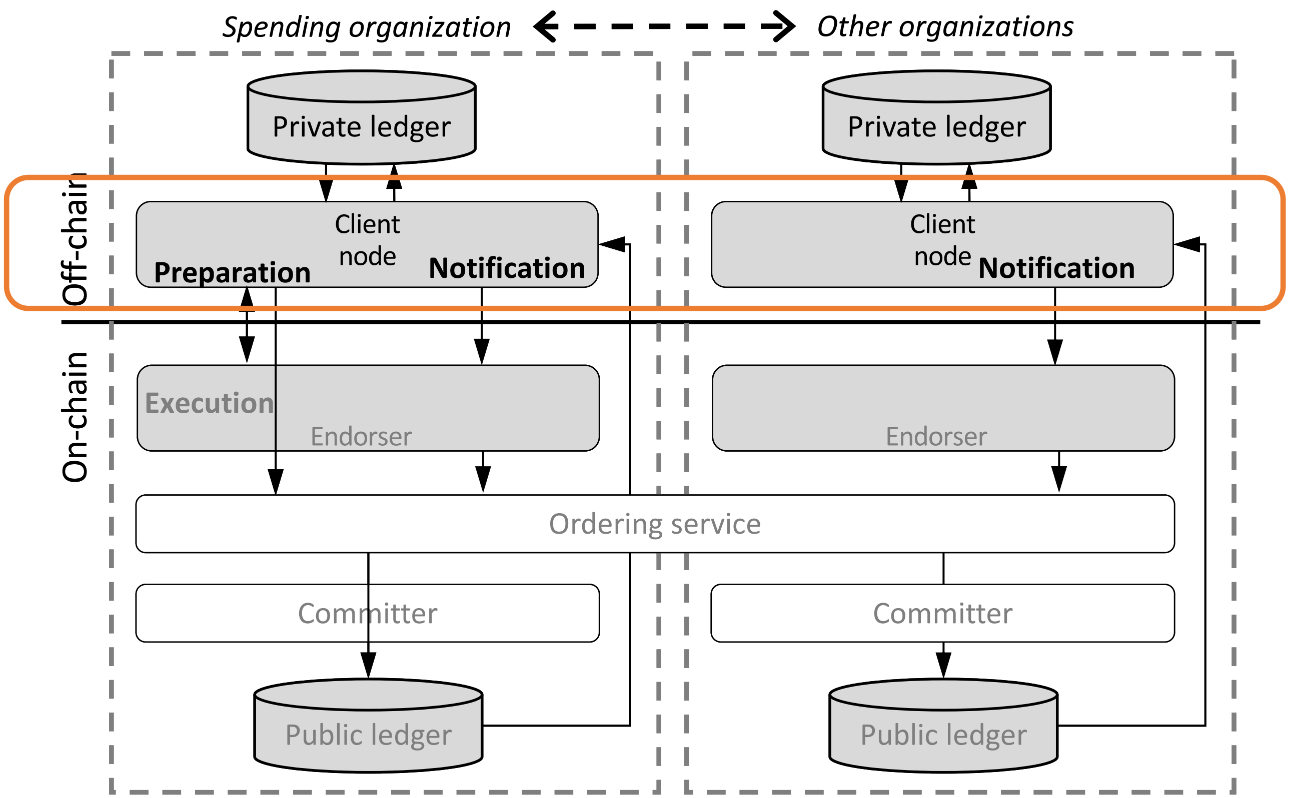
Backup

Ledger of FabZK

Tx ID	Organization A	Organization B	Organization C	Organization D	V_r	V_c
1						
m	**Com(-100, r1), token, proofs**	**Com(+100, r2), token, proofs**	**Com(0, r3), token, proofs**	**Com(0, r4), token, proofs**	**Bitmap**	**Bitmap**

- ***Row***: represents one transaction indexed by its ID
- ***Columns***: all organizations in the blockchain network
  - Hides the transaction details in commitment
  - Proves the legitimacy through the zero-knowledge Proofs
- Two validation ***bitmap***s
  - *Vr*: proof of balance, proof of correctness
  - *Vc*: proof of assets, proof of amount, and proof of consistency

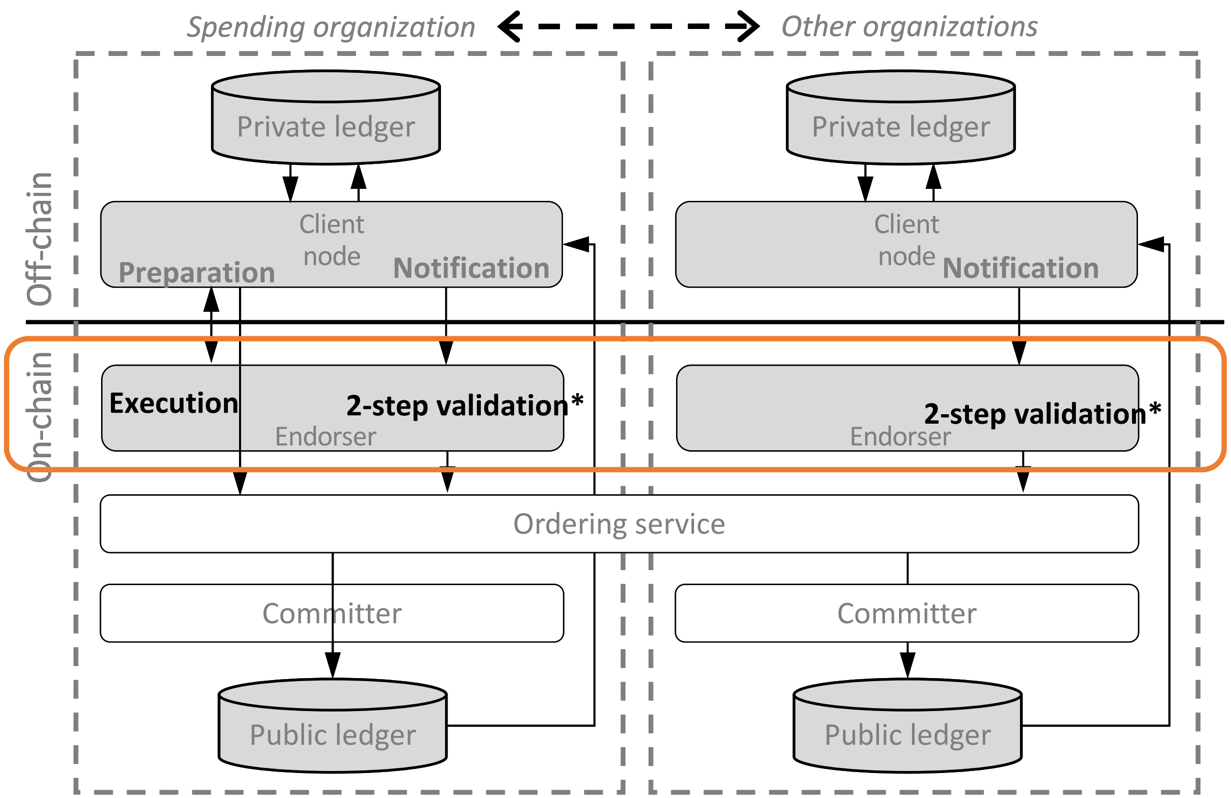
API Interface to FabZK App Developer



Client code API

- Access private and public ledgers
- Constructs and submit transactions
- Trigger the validation process

API Interface to FabZK App Developer



Client code API

- Access private and public ledgers
- Constructs and submit transactions
- Trigger the validation process

Chaincode API

- Write transactions on the public ledger (commitment, token)
- Compute proofs in 2-step validation phase
- Verify proofs

Implementation: Public Ledger

Ledger on Fabric

Transaction ID	Organization A	Organization B	V_r	V_c
1				
m	**Com(-100, r1), token, proofs**	**Com(+100, r2), token, proofs**	**Bitmap**	**Bitmap**

```
message zkrow {
    map<string , OrgColumn> columns = 1;
    bool isValidBalCor = 2;
    bool isValidAsset = 3;
}
```

Implementation: Public Ledger

Ledger on Fabric

Transaction ID	Organization A	Organization B	V_r	V_c
1				
m	**Com(-100, r1), token, proofs**	**Com(+100, r2), token, proofs**	**Bitmap**	**Bitmap**

```
message zkrow {
    map<string, OrgColumn> columns = 1;
    bool isValidBalCor = 2;
    bool isValidAsset = 3;
}
```

```
message OrgColumn {
    // transaction content
    bytes commitment = 1;
    bytes auditToken = 2;
    // two step validation state
    bool isValidBalCor = 3;
    bool isValidAsset = 4;
    // auxiliary data for proofs
    bytes TokenPrime = 5;
    bytes TokenDoublePrime = 6;
    RangeProof rp = 7;
    DisjunctiveProof dzkp = 8;
}
```

- Chaincode API
  - `zkPutState`: <comm, token>
  - `zkAudit`: range proofs, disjunctive proofs, etc
  - `zkVerify`: Set the valid status for both columns and row

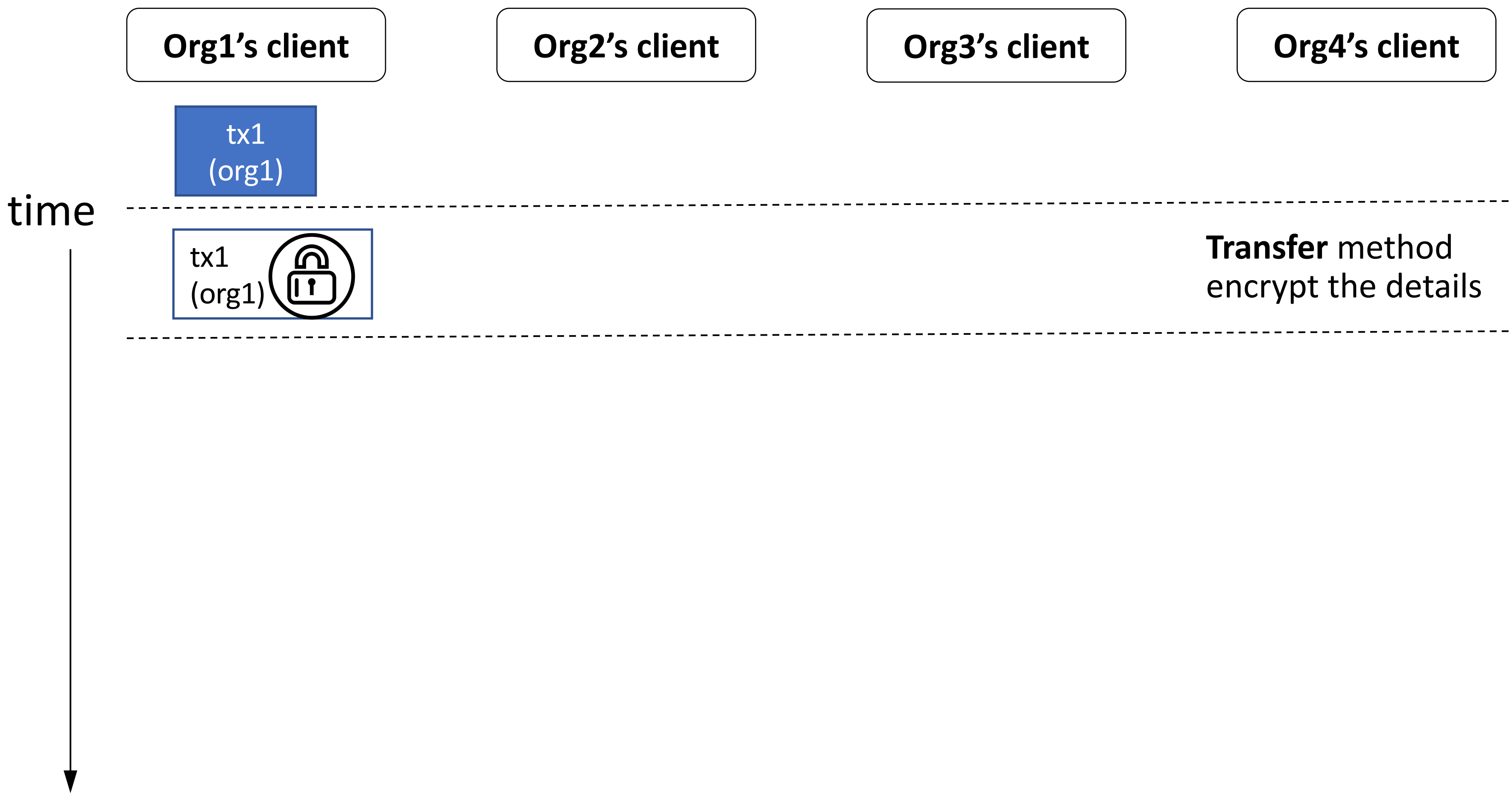
Writing Chaincode in FabZK

- Similar to Fabric, except for using **FabZK's API**

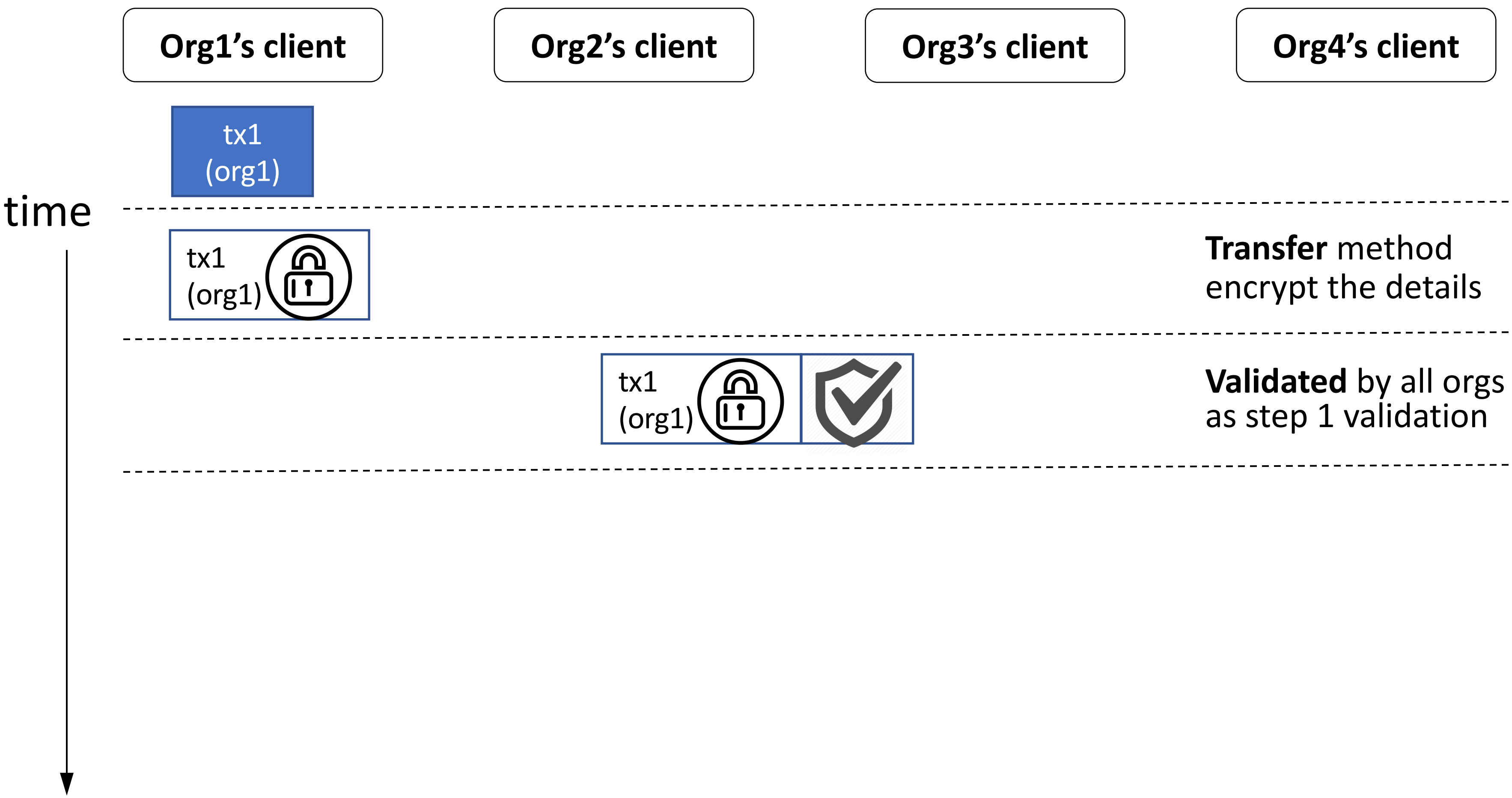
Writing Chaincode in FabZK

- Similar to Fabric, except for using **FabZK's API**
- A bare-minimum application in FabZK supports the following chaincode methods:
  - **Transfer**: exchange asset between organizations and write the transaction to the public ledger (`zkPutState`)
  - **Audit**: Compute the range proof and disjunctive proof for the transactions and write to the public ledger (`zkAudit`)
  - **Validation**: Invoke the 2-step validation to verify the transaction (`zkVerify` will be called twice)

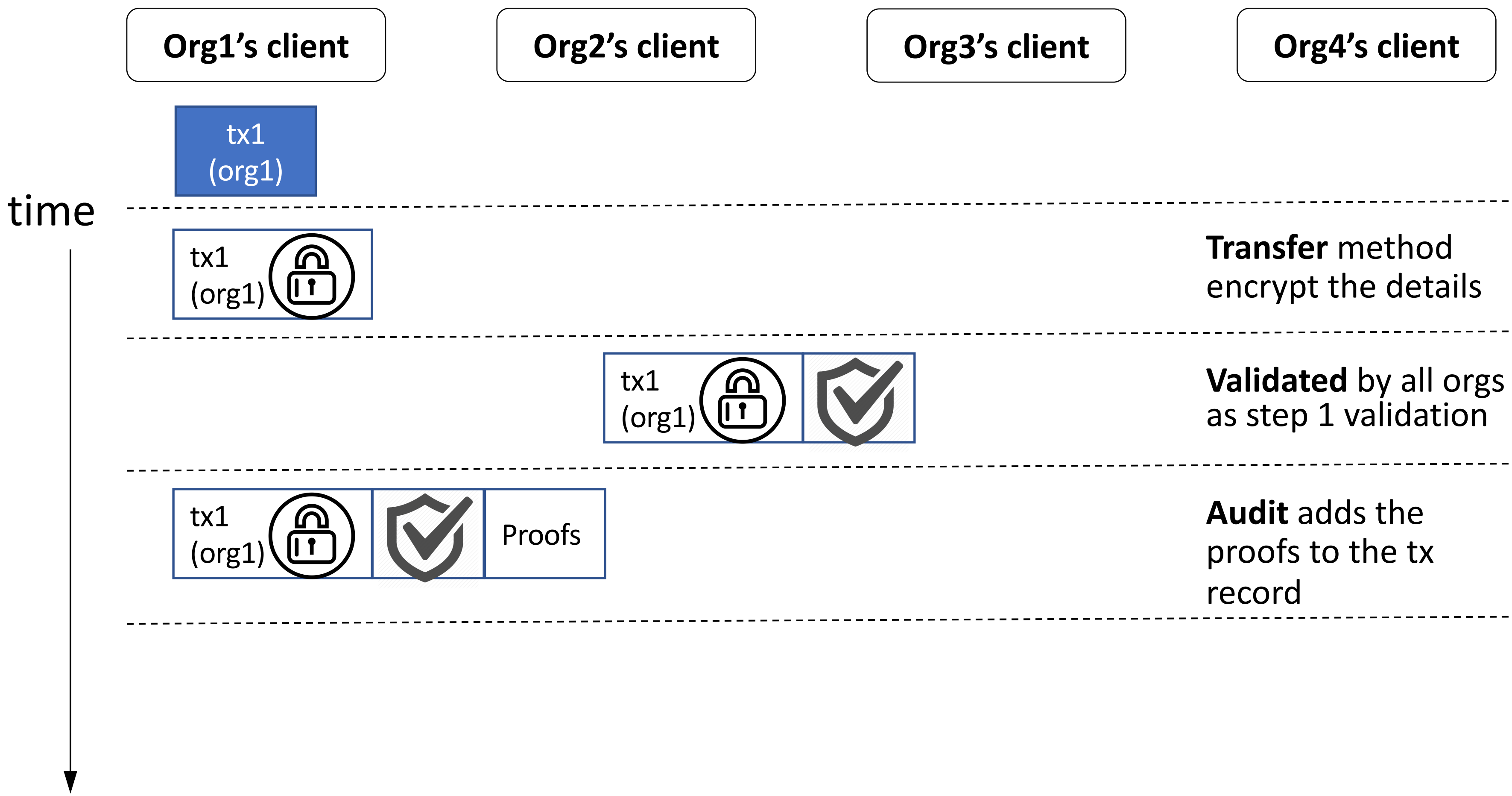
OTC Application written in FabZK



OTC Application written in FabZK



OTC Application written in FabZK



OTC Application written in FabZK