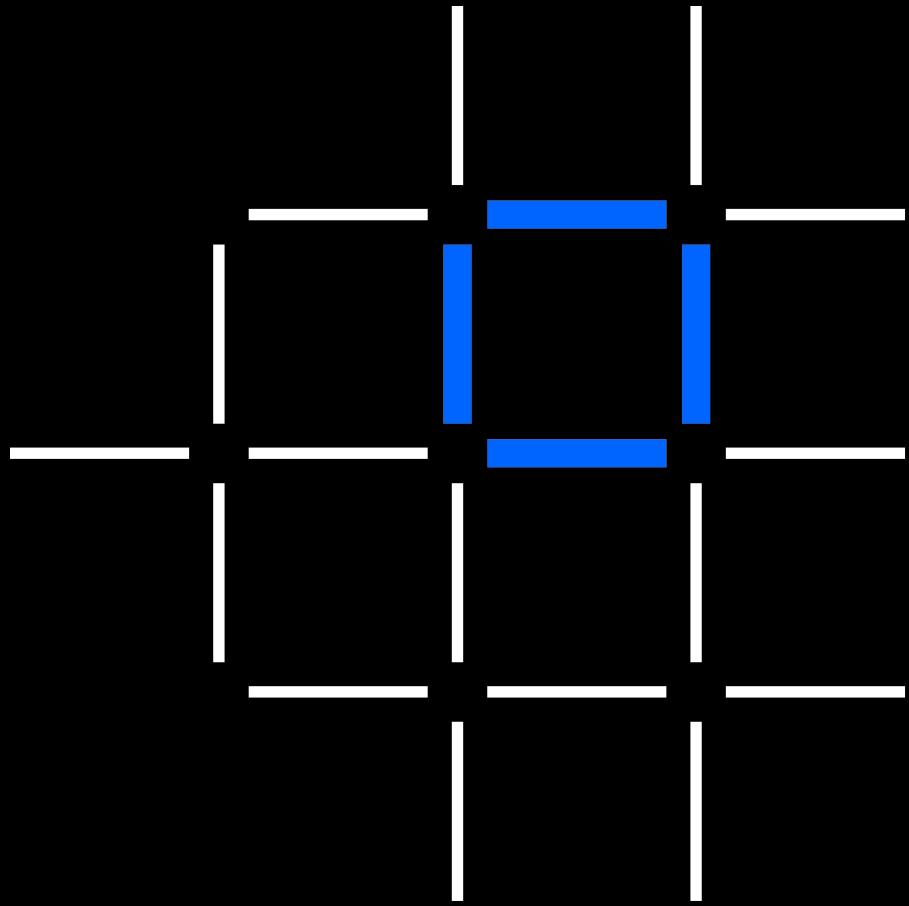
Blockchain Secured

A high-level view of Hyperledger Fabric v1 identity and privacy features



IBM **Blockchain**

Olivier VALLOD
IT Architect – Blockchain Center of Competency
IBM Client Center Montpellier, France
olivier.vallod@fr.ibm.com

V5.0
Mai 2018

IBM

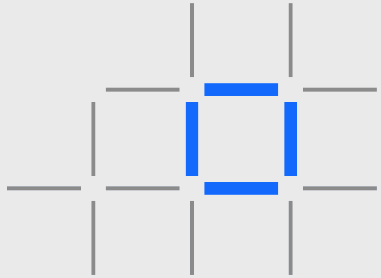
Contents

Overview

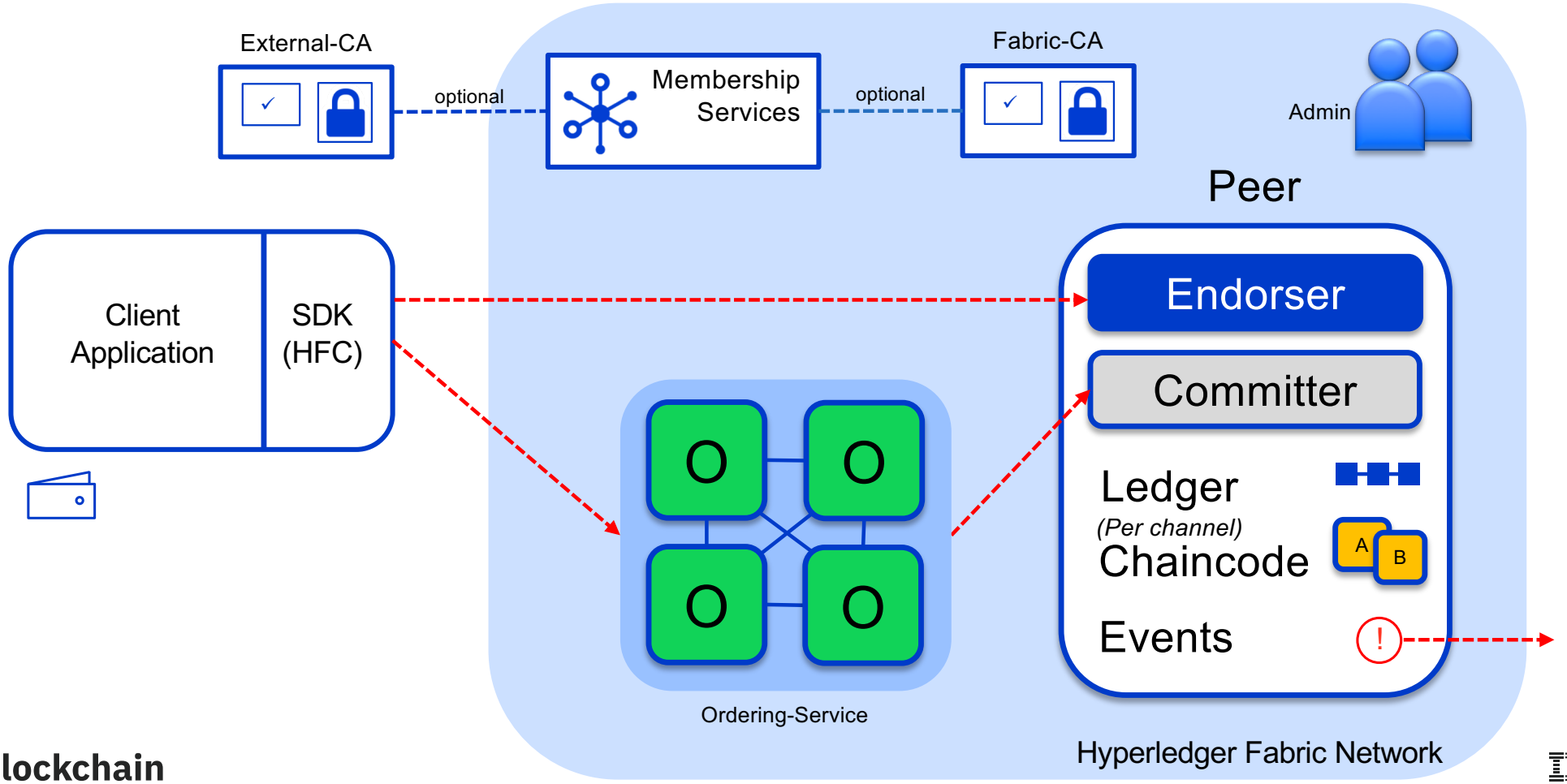
Identity

Privacy &
Confidentiality

Endorsement

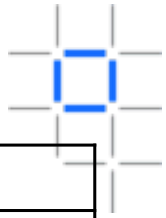


Hyperledger Fabric V1 architecture recap



Security

Enterprise blockchain networks require a high degree of security to protect confidential data and proprietary information



	Feature	Hyperledger Fabric
Security	Permission model	• **Identity management** is implemented through a **membership service** which provides identities in the form of X.509 certificates. • The **Certificate Authority (CA)** provides the necessary certificates to **enroll** and **issue transactions** in the network, and is designed to be **pluggable** in order to leverage existing, trusted, certificate authorities.
	Privacy and confidentiality	• isolation and confidentiality through **channels.** • Fine grained privacy through **the smartcontract (chaincode) and the attributes handled in the X509 certificates** • Additional isolation through **Private data** (selected Peers) • **Encryption** to reinforce the confidentiality
	Runtime isolation	• Each contract runs in its own **Docker** container, providing complete isolation between contracts.
	Resistance to attacks	• Specification of **endorsement policies** define the rules for establishing consensus for a specific smart contract.

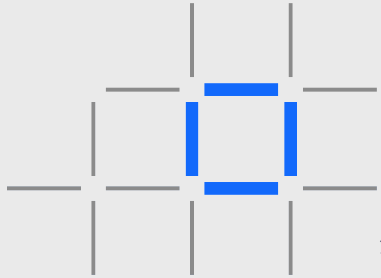
Contents

Overview

Identity

Privacy &
Confidentiality

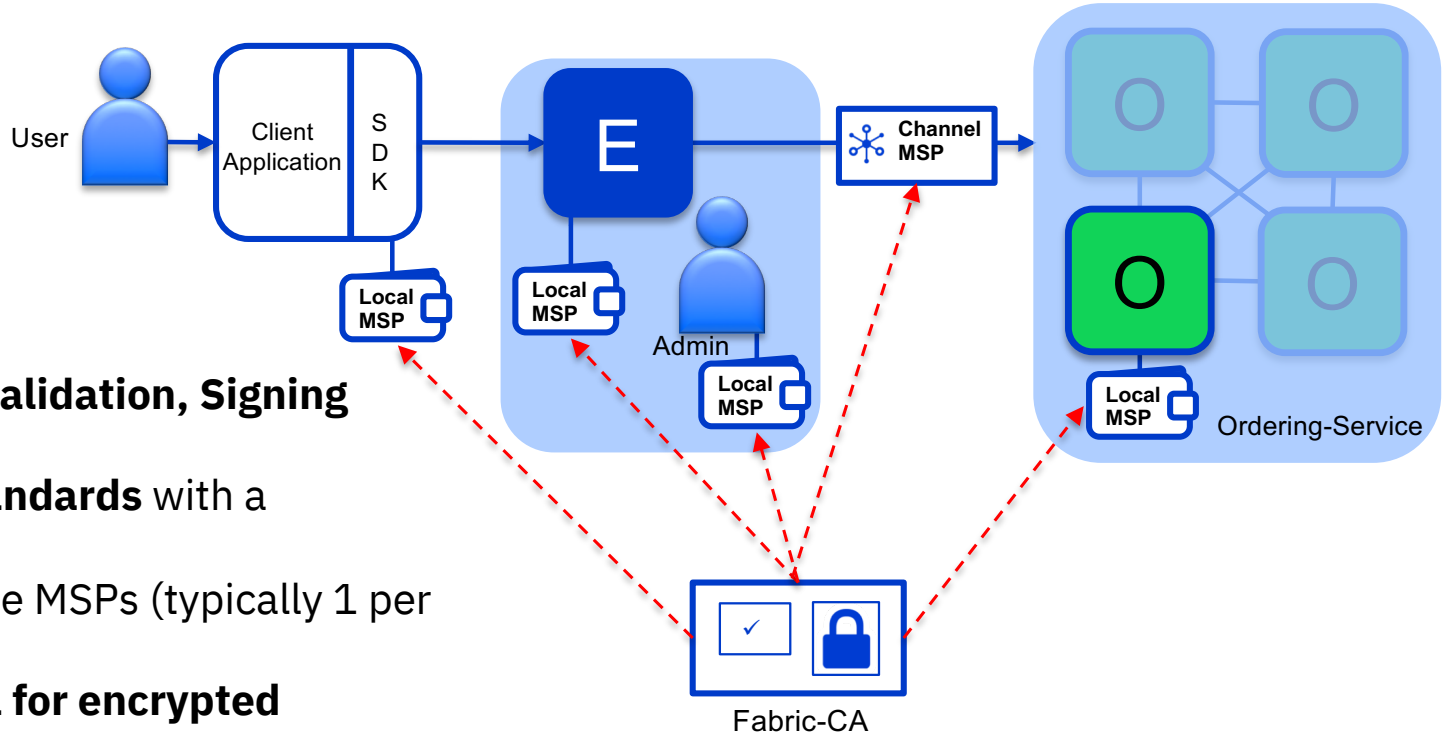
Endorsement



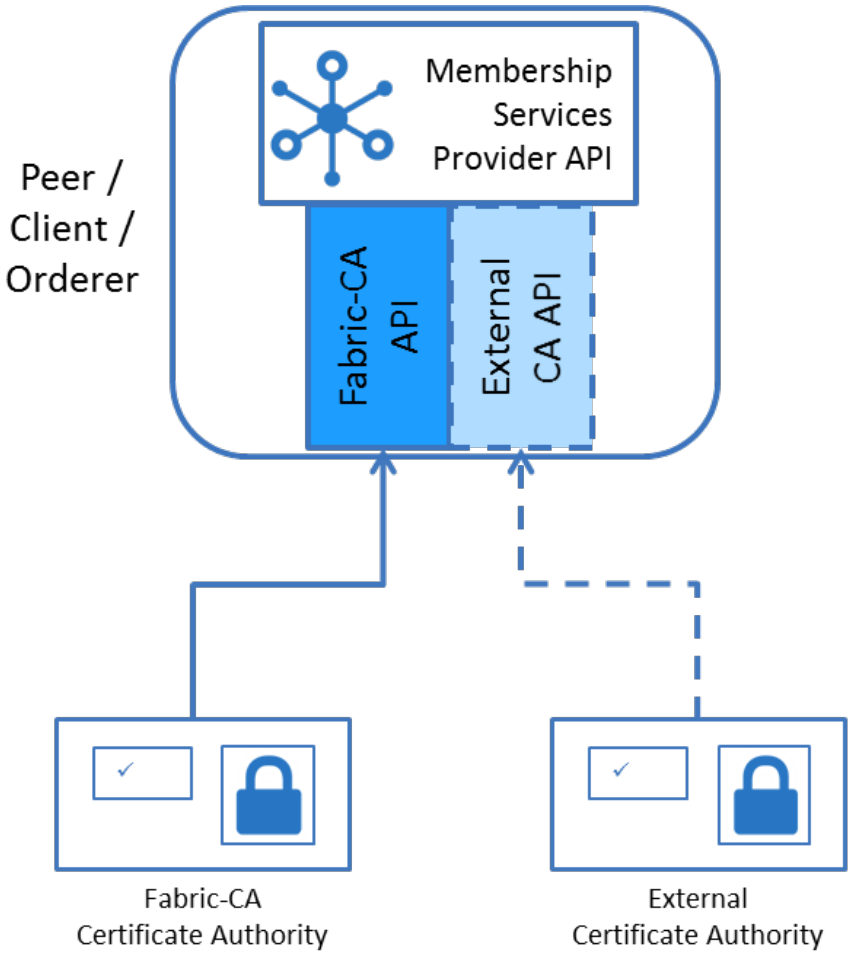
Membership Services Provider - Overview

A MSP manages a set of identities within a distributed Fabric network

- Provides **identity** for:
  - Peers and Orderers
  - Client Applications
  - Administrators
- Identities can be issued by:
  - Fabric-CA
  - An external CA
- Provides: **Authentication, Validation, Signing and Issuance**
- Supports **different crypto standards** with a pluggable interface
- A network can include multiple MSPs (typically 1 per org)
- Includes **TLS crypto material for encrypted communications**



Standard, PKI-based MSP for Fabric

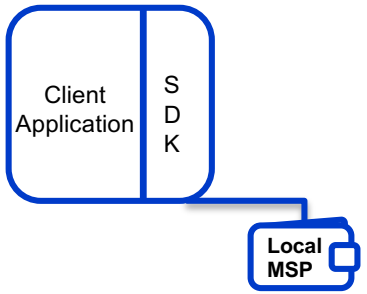


- Identity = standard X.509 certificate
- Support for ECDSA keys. Limited support for RSA keys.
- Governed by standard PKI hierarchies (root CA / intermediate Cas, CRLs)
- For certificate issuing, one can use existing CA, or Hyperledger default provided Fabric-CA.

User Identities

Each client application has a local MSP to store user identities

- Each local MSP includes:
  - **Keystore**
    - **Private key** for signing transactions
  - **Signcert**
    - **Public x.509 certificate**



- May also include TLS credentials
- Can be backed by a **Hardware Security Module (HSM)**

user@org1.example.com	
keystore	<private key>
signcert	user@org1.example.com-cert.pem

Admin Identities

Each Administrator has a local MSP to store their identity

- Each local MSP includes:
  - **Keystore**
    - **Private key** for signing transactions
  - **Signcert**
    - **Public x.509 certificate**

- May also include TLS credentials
- Can be backed by a Hardware Security Module (HSM)

admin@org1.example.com	
keystore	<private key>
signcert	admin@org1.example.com-cert.pem

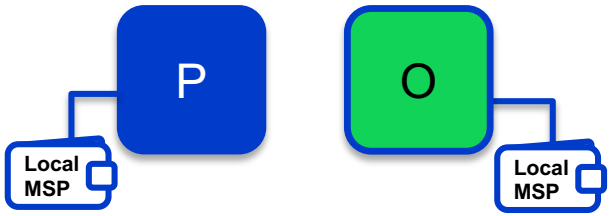
Peer and Orderer Identities

Each peer and orderer has a local MSP

- Each local MSP includes:
  - **keystore**
    - **Private key** for signing transactions
  - **signcert**
    - **Public x.509 certificate**

- In addition Peer/Orderer MSPs identify authorized administrators:
  - **admincerts**
    - List of **administrator certificates**
  - **cacerts**
    - The **CA public cert** for verification
  - **crls**
    - List of **revoked certificates**

- Peers and Orderers also receive channel MSP info
- Can be backed by a Hardware Security Module (HSM)

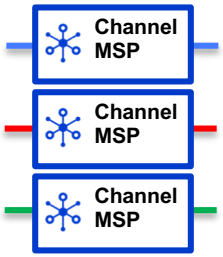


peer@org1.example.com	
admincerts	admin@org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
keystore	<private key>
signcert	peer@org1.example.com-cert.pem
crls	<list of revoked admin certificates>

Channel MSP information

Channels include additional organisational MSP information

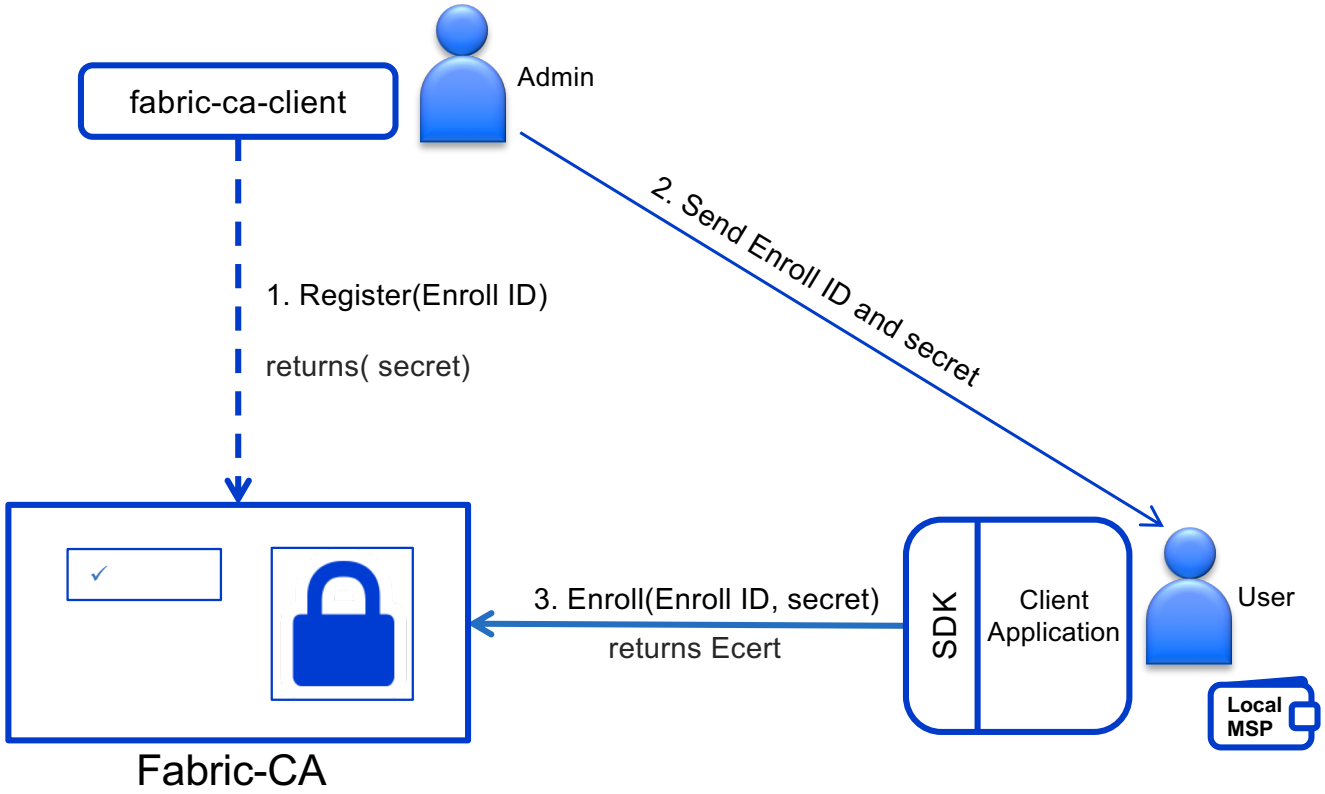
- Determines which orderers or peers can join the channel
- Determines client applications read or write access to the channel
- Stored in configuration blocks in the ledger
- Each channel MSP includes:
  - **admincerts**
    - Any public certificates for administrators
  - **cacerts**
    - The CA public certificate for this MSP
  - **crls**
    - List of revoked certificates
- Does not include any private keys for identity



Channels

ID = MSP1	
admincerts	admin.org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
crls	<list of revoked admin certificates>

New User Registration and Enrollment



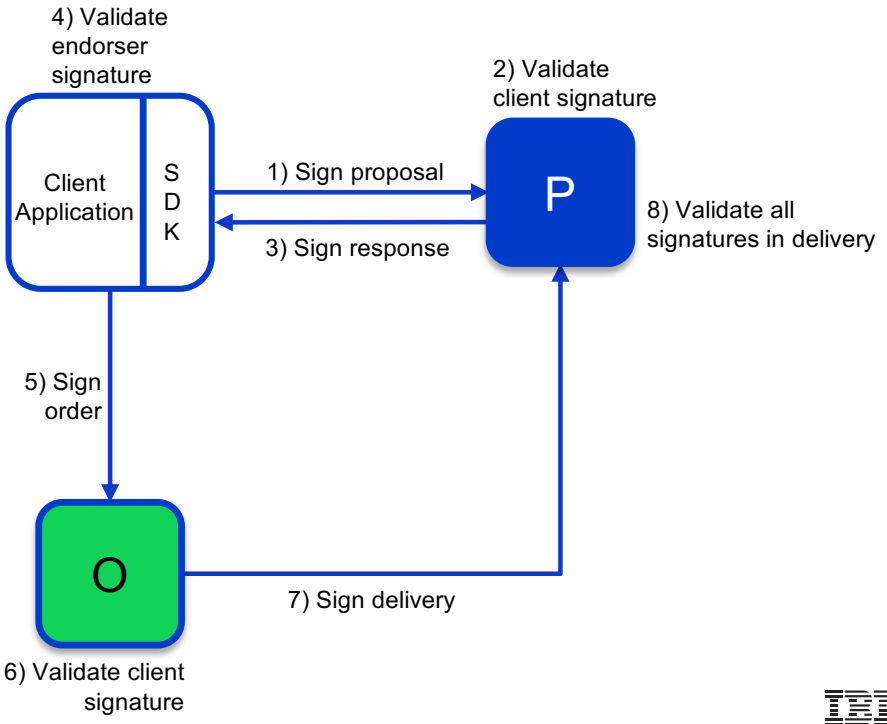
Registration and Enrollment

- Admin registers new user with Enroll ID
- User enrolls and receives credentials
- Additional offline registration and enrollment options available

Transaction Signing

All transactions within a Hyperledger Fabric network are signed by permissioned actors, and those signatures validated

- Actors sign transactions with their enrollment private key
  - Stored in their local MSP
- Components validate transactions and certificates
  - Root CA certificates and CRLs stored in local MSP
  - Root CA certificates and CRLs stored in Org MSP in channel



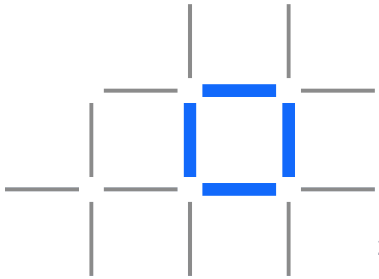
Identities and policies are required at every stage

IBM Blockchain

- Identities and policies are enforced at every stage of the lifecycle of a blockchain:
  - **Creating channels**
  - **Installing and instantiating chaincodes**
  - **Endorsing transactions**

Here is, for example, what happens if you try to create a channel while not being an admin:

2017-11-24 14:04:52.074 UTC [msp] SatisfiesPrincipal -> DEBU 16b Checking if **identity** satisfies **ADMIN** role for **BlockChainCoCMSP**
2017-11-24 14:04:52.074 UTC [cauthdsl] func2 -> DEBU 16c 0xc420156750 identity 0 does **not** satisfy principal: This identity is **not** an admin
2017-11-24 14:04:52.074 UTC [cauthdsl] func2 -> DEBU 16d 0xc420156750 principal evaluation **fails**
2017-11-24 14:04:52.074 UTC [cauthdsl] func1 -> DEBU 16e 0xc420156750 gate 1511532292069900528 evaluation fails
2017-11-24 14:04:52.074 UTC [orderer/common/broadcast] Handle -> WARN 16f Rejecting CONFIG_UPDATE because: Error authorizing update: Error validating DeltaSet: Policy for [Groups] /Channel/Application not satisfied: Failed to reach implicit threshold of 1 sub-policies, required 1 remaining
2017-11-24 14:04:52.074 UTC [orderer/main] func1 -> DEBU 170 Closing Broadcast stream

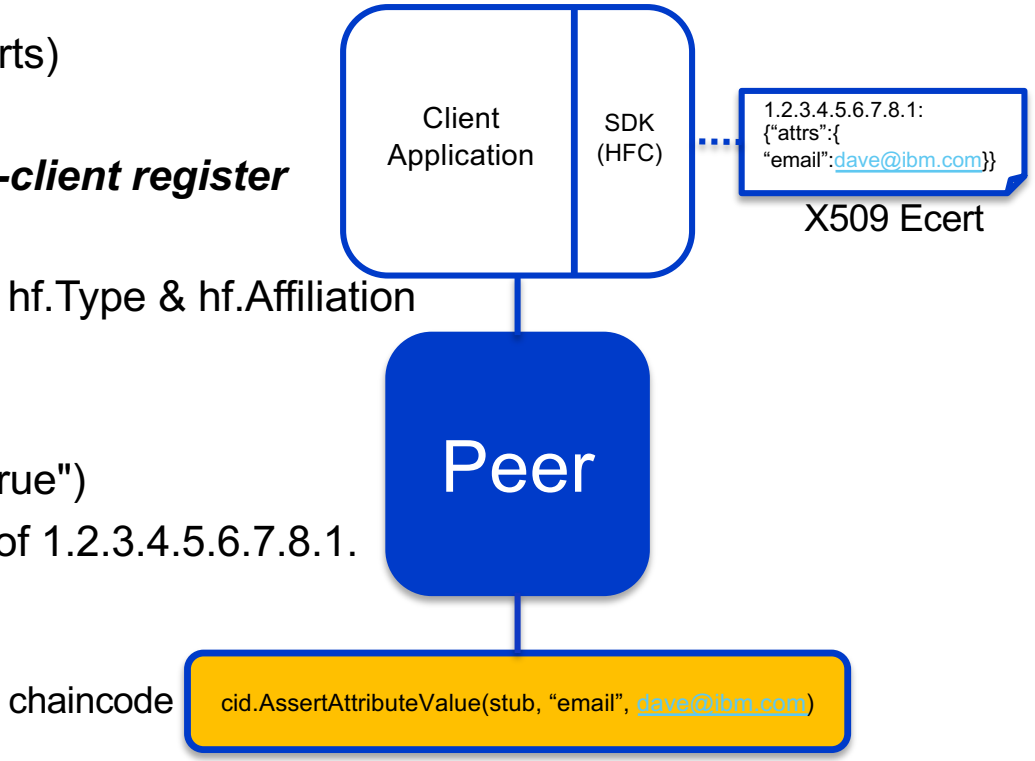


Attribute Based Access Control

Include identity attributes in enrollment certificates for chaincode

- Include attributes in **X509 enrollment certificates** (Ecerts)
- Defined as name/value pairs: email=dave@ibm.com
- Define mandatory and optional attributes with ***fabric-ca-client register***
- Specify attribute values with ***fabric-ca-client enroll***
- Ecerts automatically include attributes: hf.EnrollmentID, hf.Type & hf.Affiliation
- API provided by Client Identity chaincode Library:
  - cid.GetAttributeValue(stub, "attr1")
  - cid.AssertAttributeValue(stub, "myapp.admin", "true")
- Stored as an extension in the Ecert with an ASN.1 OID of 1.2.3.4.5.6.7.8.1.

```
1.2.3.4.5.6.7.8.1:
    {"attrs":{"attr1":"val1"}}
```



https://jira.hyperledger.org/browse/FAB-5346 + https://jira.hyperledger.org/browse/FAB-6089

Contents

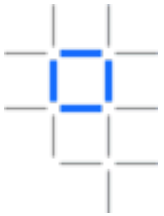
Overview

Identity

Privacy & Confidentiality

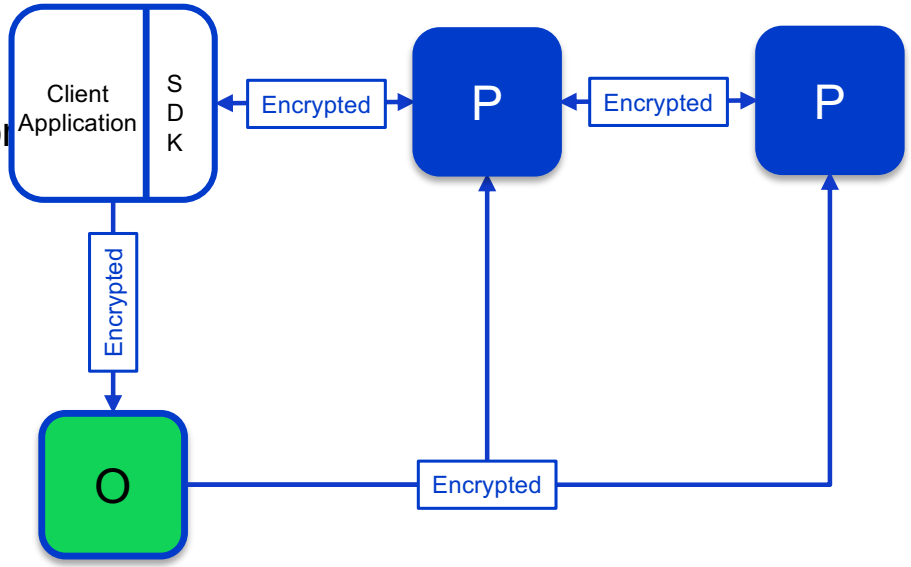
Endorsement

Transport Layer Security



- All communications within a Hyperledger Fabric network can be secured using TLS

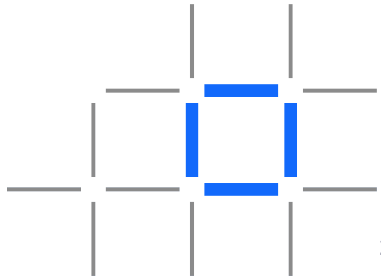
- Peers and Orderers are both TLS Servers and TLS Clients
- Applications and commands are TLS Clients
- Fabric 1.1 supports mutual TLS (client & server authentication



https://jira.hyperledger.org/browse/FAB-6715

Insuring privacy in a blockchain

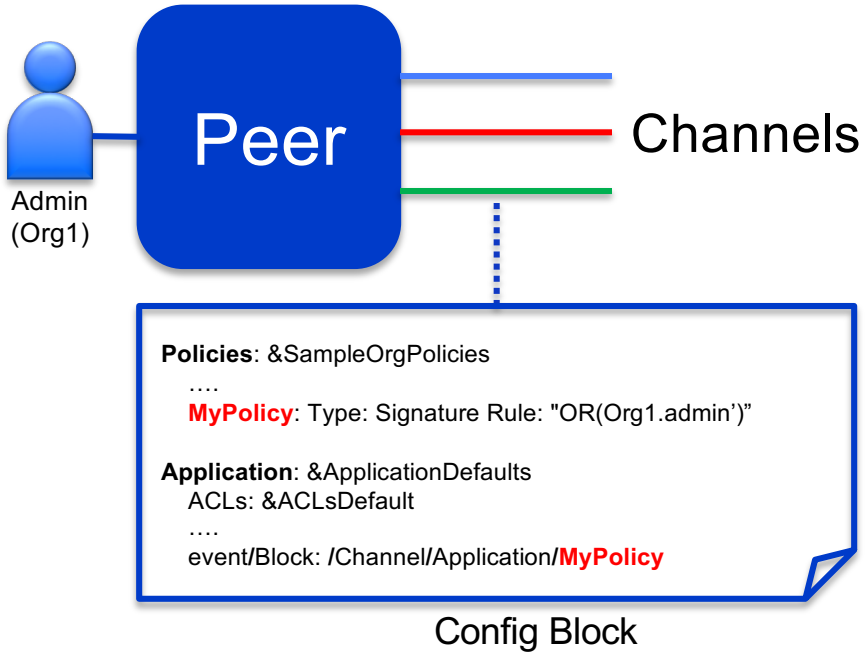
- Privacy in a blockchain is insured through different mechanisms.
  - **Channel support**
    - Data partition mechanism implementing its own total order broadcast mechanism
    - Channel transaction ordering takes place independently of other channels, and by the members of the channel.
    - Channel creation upon properly authenticated and authorized request
    - Channel creation request submitted and evaluated by the ordering service
      - Participation is restricted to a subset of organizations / participants (MSPs)
      - Participation is defined by means of **policies** for **Readers**, **Writers** and **Admins**.
  - **Chaincode availability mechanism**
    - Protects business logic incorporated on a chaincode logic
  - **Encryption library**
    - Used on client side, or at chaincode execution time
    - Protects chaincode data



ACL mechanism per channel

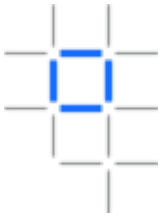
Support policy based access control for peer functions per channel

- Access control defined for channel and peer resources:
  - User / System chaincode
  - Events stream
- Policies specify identities and include defaults for:
  - Readers
  - Writers
  - Admins
- Policies can be either:
  - Signature : Specific user type in org
  - ImplicitMeta : “All/Any/Majority” signature types
- Custom policies can be configured for ACLs



https://jira.hyperledger.org/browse/FAB-8727

Private Data Collections - Explained

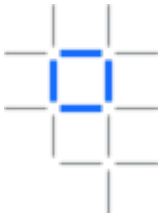


- Data privacy within a channel
- Transaction proposal and worldstate read/write sets available to only **permissioned peers**
- Ordering service has only evidence of transactions (hashes)
- Policy defines which peers have private data

1. **Private data:**
    1. Excluded from transactions by being sent as ‘transient data’ to endorsing peers.
    2. Shared peer-to-peer with only peers defined in the collection policy.
2. **Hashes of private data** included in transaction proposal for evidence and validation.
    1. Peers not in the collection policy and the Orderer only have hashes.
3. **Peers maintain** both a public worldstate and a private worldstate.
4. **Private data** held in a transient store between endorsement and validation.

https://jira.hyperledger.org/browse/FAB-1151

Private Data Collections – Marble Scenario



Privacy Requirements:

- No marble data should go through ordering service as part of a transaction
- All peers have access to general marble information
  - *Name, Size, Color, Owner*
- Only a subset of peers have access to marble *pricing* information

Transaction
- Primary read/write set (if exists)
- Hashed private read/write set (hashed keys/values)

Transaction
- Public channel data
- Goes to all orderers/peers

Collection: Marbles
- Private Write Set
- Name, Size, Color, Owner

Policy: Org1, Org2
"requiredPeerCount": 1,
"maxPeerCount":2,
"blockToLive":1000000

Collection: Marbles
- Private data for channel peers
- Goes to all peers but not orderers

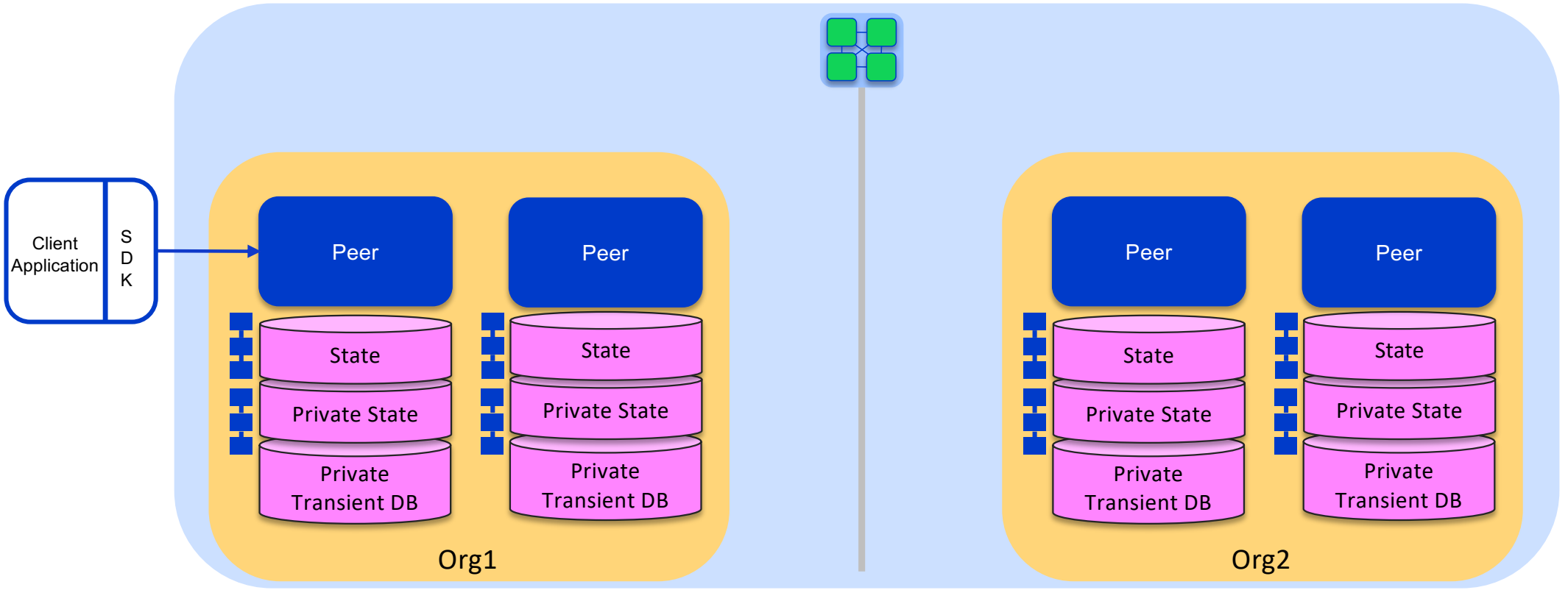
Collection: Marble Private Details
- Private Write Set
- Price

Policy: Org1
"requiredPeerCount": 1,
"maxPeerCount": 1,
"blockToLive":3

Collection: Marbles Private Details
- Private data for subset of channel peers
- Goes to subset of peers only

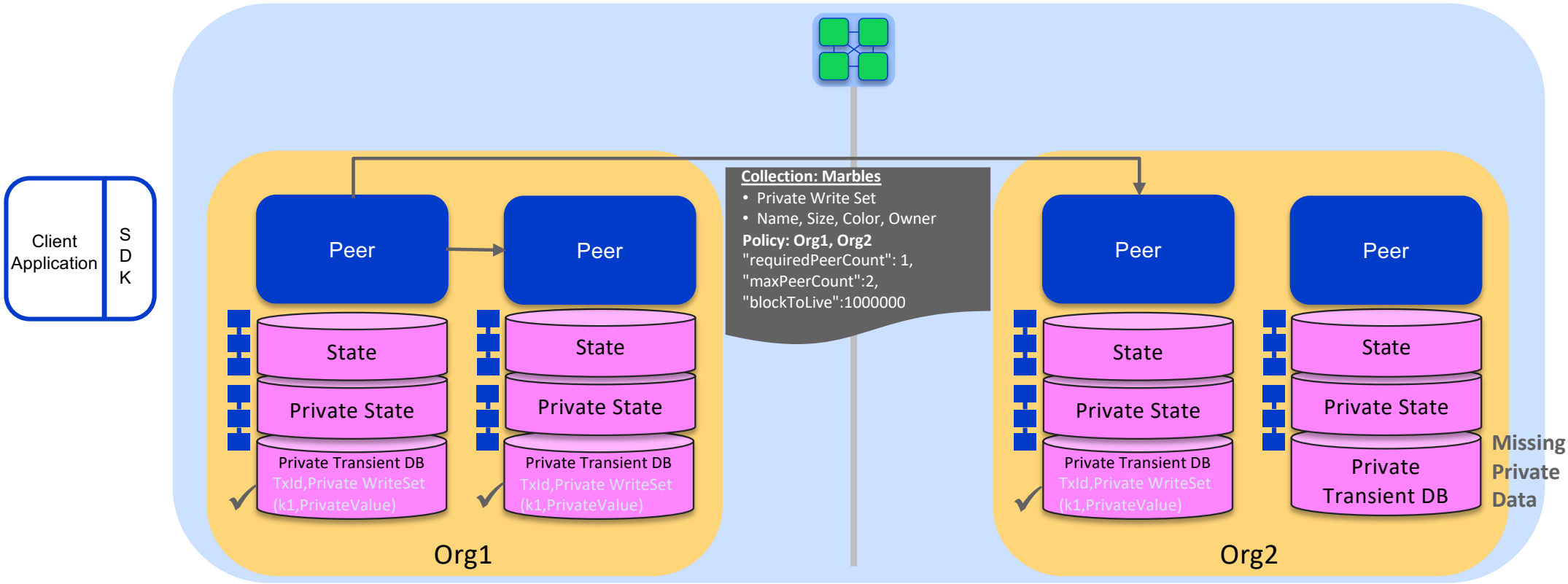
Step 1: Propose Transaction

Client sends proposal to endorsing peer(s)



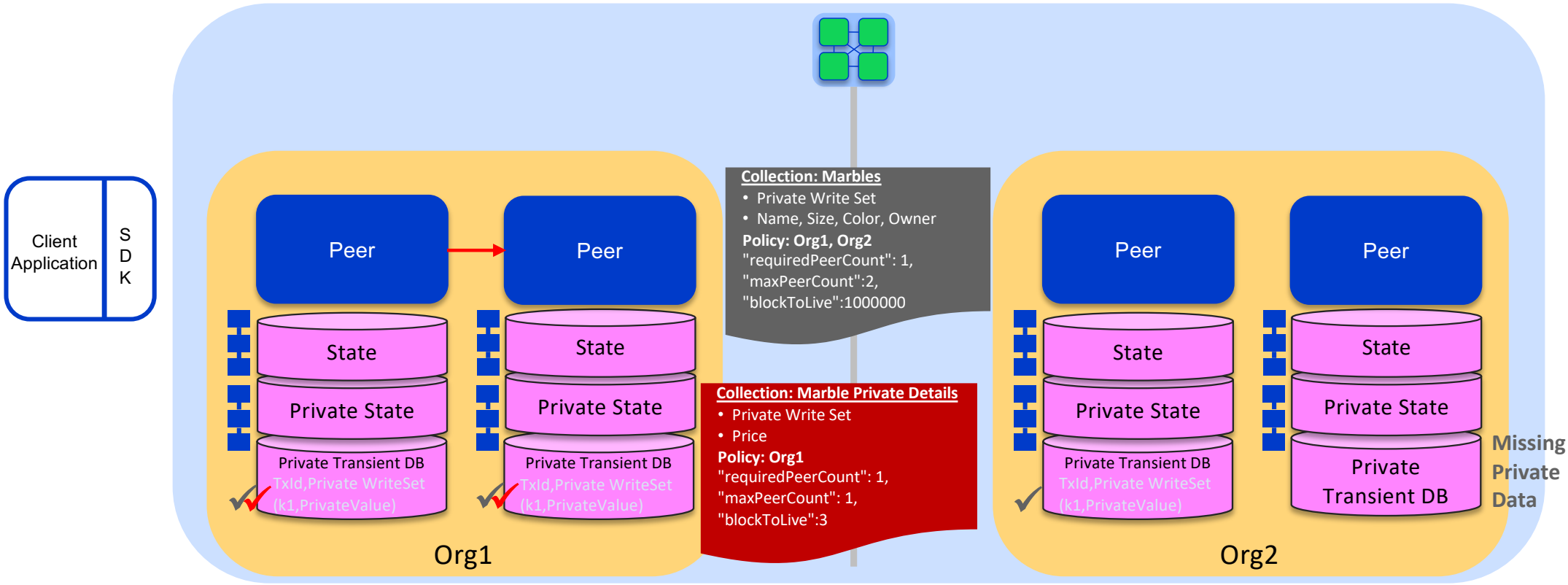
Step 2a: Execute Proposal and Distribute 1st Collection

Endorsing peer simulates transaction and distributes **marbles collection** data based on policy



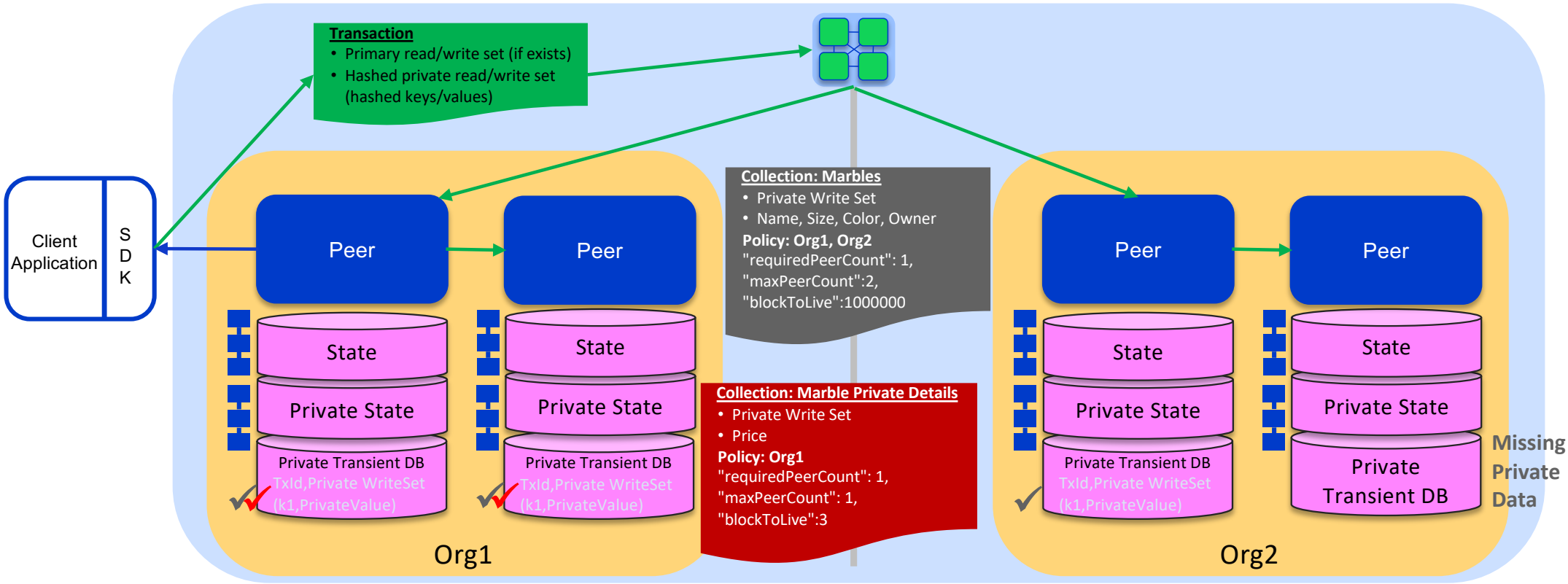
Step 2b: Distribute 2nd Collection

Endorsing peer distributes **marbles private details collection** data based on policy



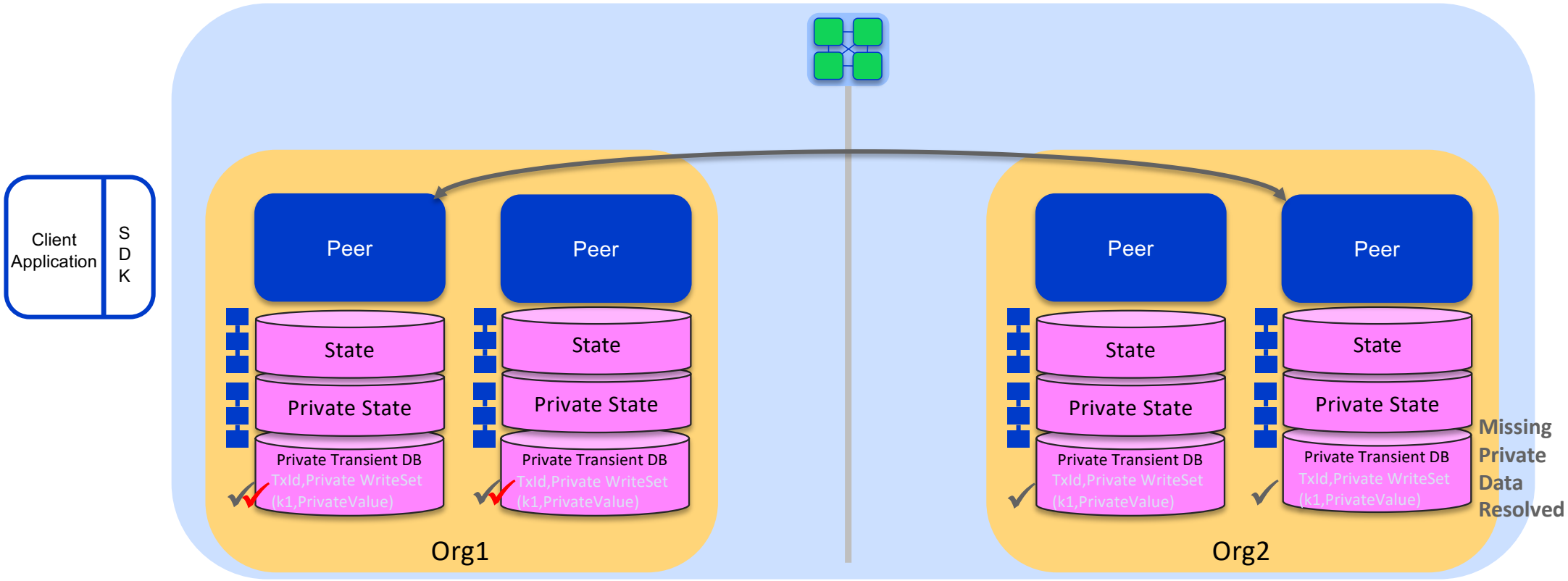
Step 3: Proposal Response / Order / Deliver

Proposal response sent back to client, which then sends the proposal to the ordering service for delivery to all peers



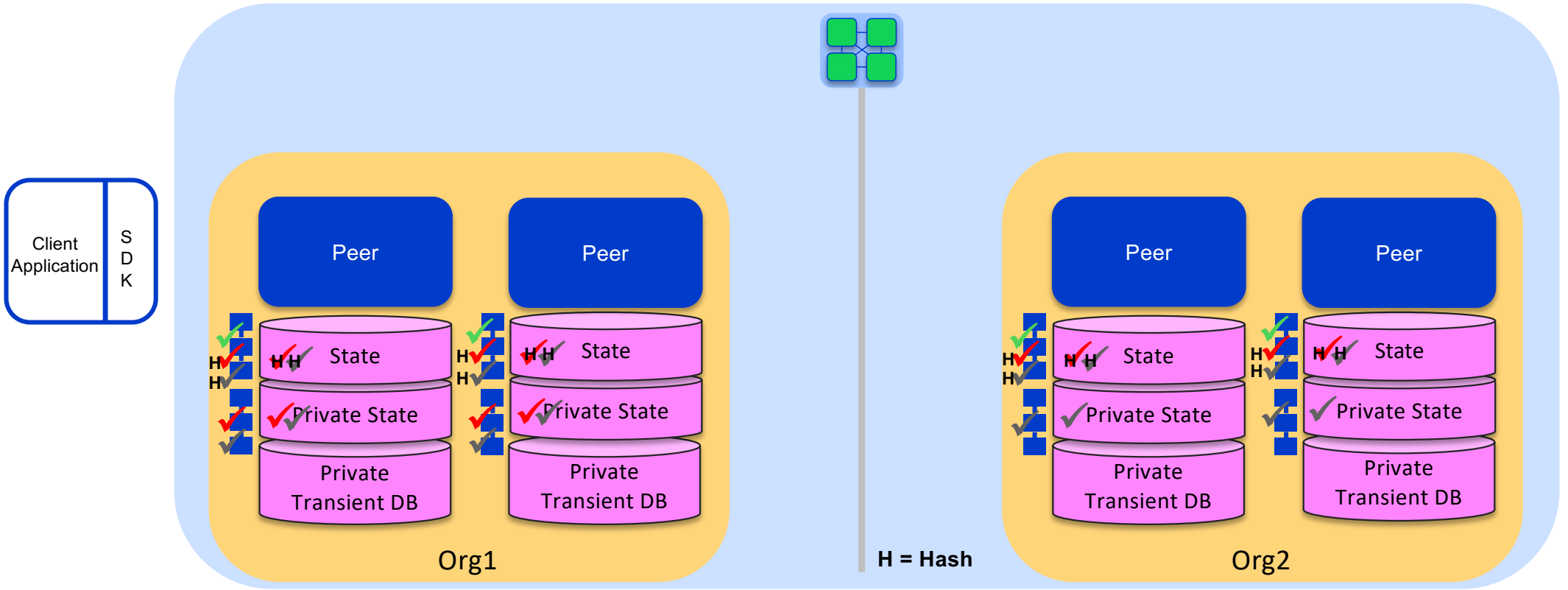
Step 4: Validate Transaction

Peers validate transactions. Private data validated against hashes. Missing private data resolved with pull requests from other peers.



Step 5: Commit

1) Commit private data to private state db. 2) Commit hashes to public state db.
3) Commit public block and private write set storage. 4) Delete transient data.

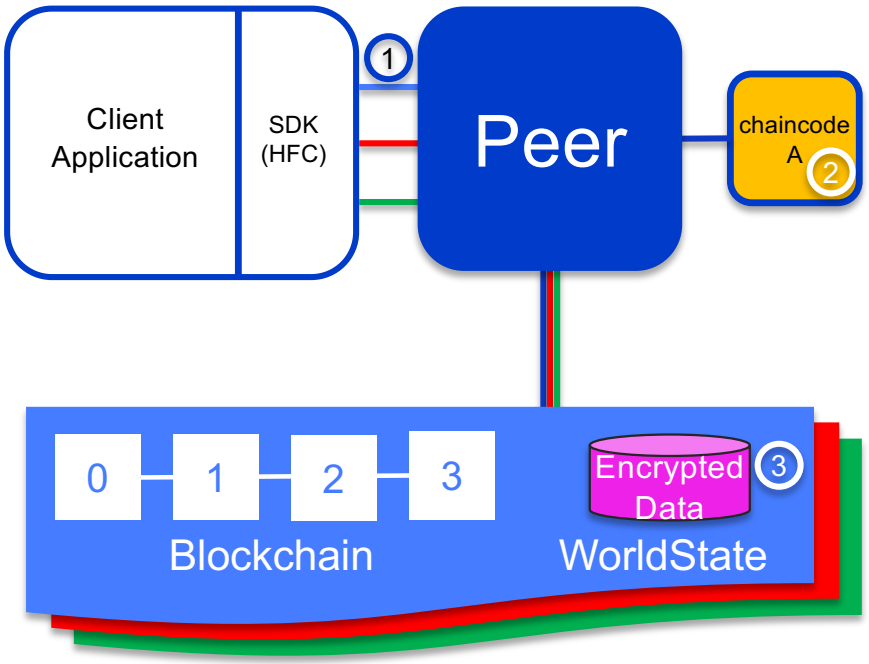


Data confidentiality using encryption at the application level

Encrypt/Decrypt and sign data in chaincode

- Fabric includes an encryption library for use by chaincode
- New chaincode “entities” API provides interface to:
  - Encrypt (AES 256 symmetric or asymmetric)
  - Decrypt
  - Sign (ECDSA)
  - Verify
- Pass cryptographic key(s) to chaincode via **transient-data** field
- Support for Initialisation Vector (IV) allowing multiple endorsers to calculate same cypher text

1. Pass unencrypted data and keys to endorser
2. Chaincode encrypts data to put in worldstate
3. Encrypted data stored in worldstate



https://jira.hyperledger.org/browse/FAB-830

Bring your own cryptography

Blockchain Cryptographic Services Provider - BCCSP

	Abstraction	An abstraction of cryptographic operations used in Hyperledger Fabric
	Pluggability	Alternate implementations of cryptographic interfaces can be used within the Hyperledger Fabric code, without modifying the core
	Multiple BCCSP	Easy addition of more types of CPs, for example to support different HSM types
	International Standard support	Pluggable crypto service provide. Potential to support more fine grained confidentiality features

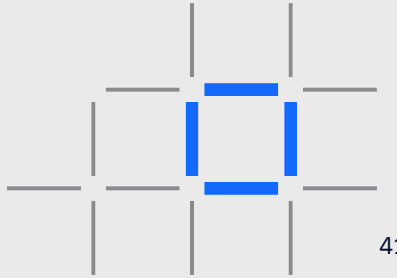
Contents

Overview

Identity

Privacy &
Confidentiality

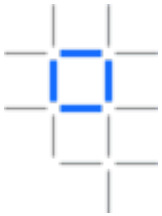
Endorsement



Consensus is achieved using the following transaction flow:

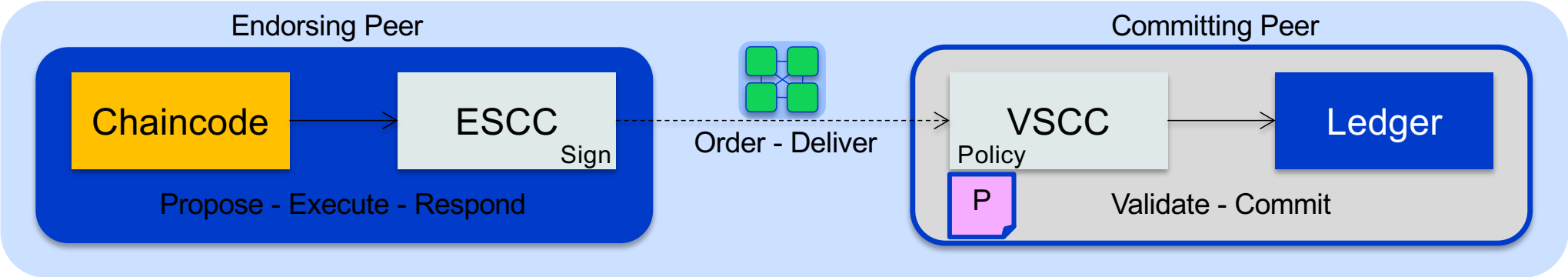
Hyperledger Fabric Consensus

Endorsement Policies



An endorsement policy describes the conditions by which a transaction can be endorsed. A transaction can only be considered valid if it has been endorsed according to its policy.

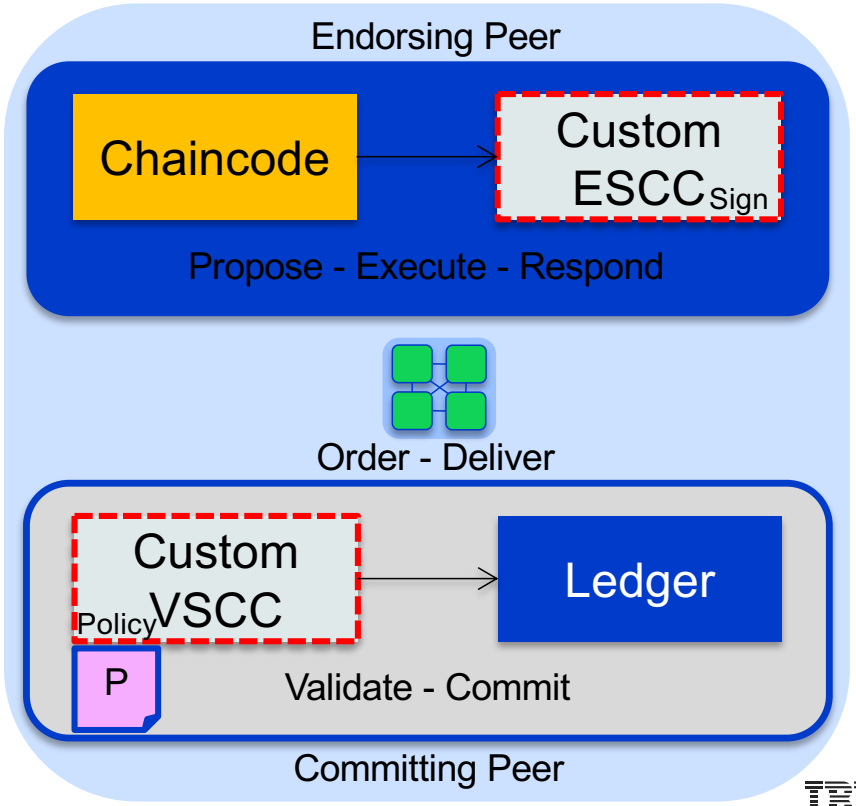
- Each chaincode is deployed with an Endorsement Policy
- **ESCC** (Endorsement System ChainCode) signs the proposal response on the endorsing peer
- **VSCC** (Validation System ChainCode) validates the endorsements



Pluggable endorsement and validation

Support for custom transaction endorsement and validation plugins

- Supports alternative transaction models for: State based endorsement, UTXO etc
- No need to recompile peer, **core.yaml** specifies additional golang plugins
- Support for custom:
  - **ESCC** : Endorsement System Chaincode
  - **VSCC** : Validation System Chaincode
  - **QSCC** : Query System Chaincode
  - **CSCC** : Configuration System Chaincode
  - **LSCC** : Lifecycle System Chaincode
- Chaincode associated with custom ESCC and VSCC at instantiation



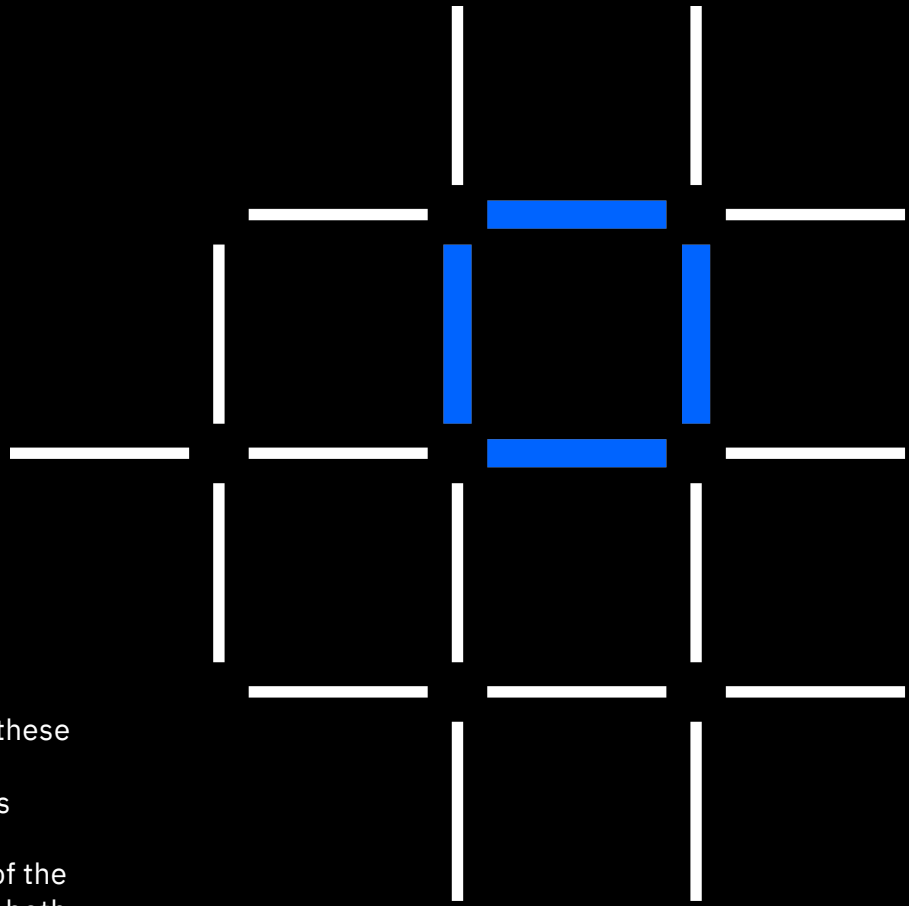
IBM **Blockchain**

https://jira.hyperledger.org/browse/FAB-8779

Thank you

Olivier VALLOD
olivier.vallod@fr.ibm.com

IBM **Blockchain**



www.ibm.com/blockchain

developer.ibm.com/blockchain

www.hyperledger.org

IBM