State-based Endorsement

(formerly known as “ownable state”, also known as “key-level validation”)

M. Neugschwandtner, A. Sorniotti

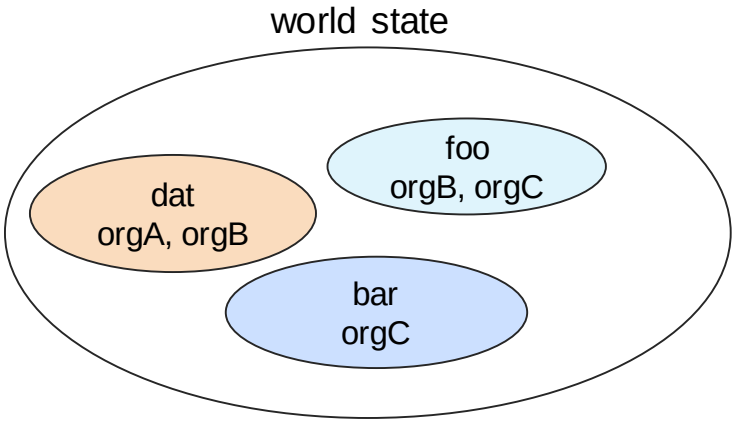
Motivation

- Endorsement policies are currently
  - coarse-grained, on a per-chaincode level
  - change requires chaincode upgrade

- Proposal: state-based endorsement
  - endorsement policies on a per-KVS-key-level
  - changes require only chaincode logic

State-based endorsement

- KVS key with designated endorsement policy
  - EP encoded as key-level metadata
  - EP used to validate changes to KVS value



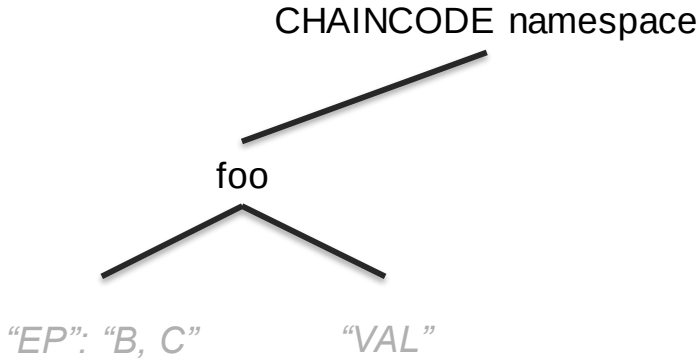
Comparison

	Chaincode Endorsement Policy	State-based Endorsement Policy
Modification process	tied to the the chaincode lifecycle: chaincode upgrade required to set a new endorsement policy	part of a regular fabric transaction that updates key metadata
Granularity	one endorsement policy per chaincode: different endorsement policies require multiple chaincodes	one endorsement policy per KVS key

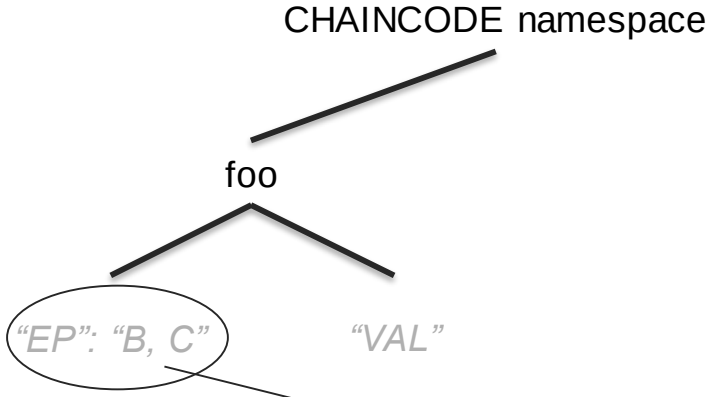
Chaincode Shim Interface

```
// SetStateEndorsementPolicy sets the endorsement policy for the specified
// `key`. The endorsement policy can not be set on a key that does not
// exist. To remove the endorsement policy for the given key, set `ep` to nil.
SetStateEndorsementPolicy(key string, ep []byte) error

// GetStateEndorsementPolicy retrieves the endorsement policy for `key`.
// Note that this will introduce a read dependency on `key`.
GetStateEndorsementPolicy(key string) ([]byte, error)

// SetPrivateDataEndorsementPolicy sets the endorsement policy for the private
// data specified by `key`.
SetPrivateDataEndorsementPolicy(collection, key string, ep []byte) error

// GetPrivateDataEndorsementPolicy retrieves the endorsement policy for the
// private data specified by `key`. Not that this introduces a read dependency
// on `key`.
GetPrivateDataEndorsementPolicy(collection, key string) ([]byte, error)
```

On Ledger Representation

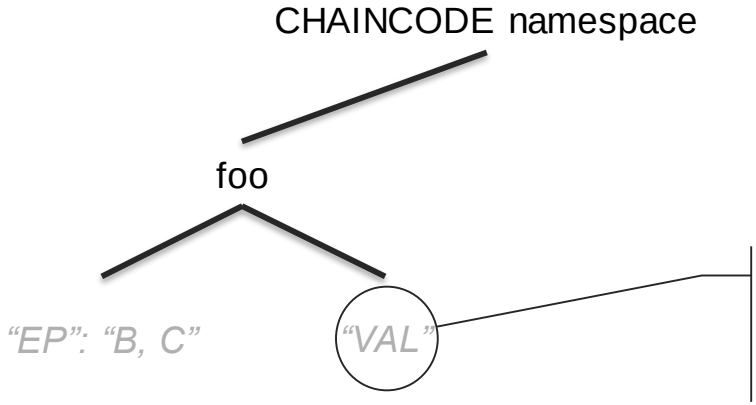


On Ledger Representation



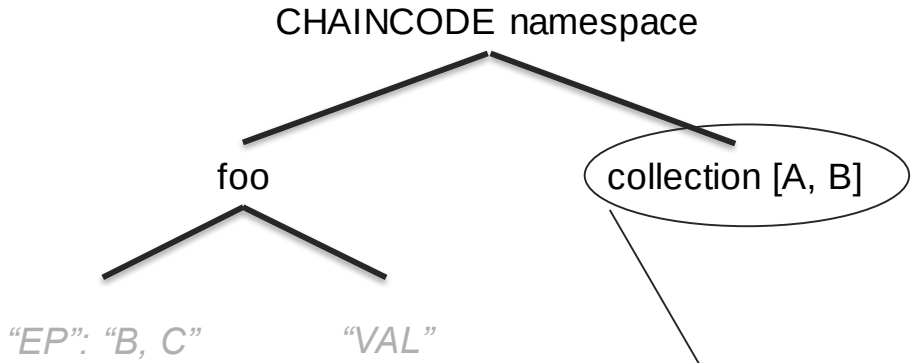
ENDORSEMENT POLICY
- Stored in key-level metadata
- Allows for public verifiability
- Reuses cauthdsl

On Ledger Representation



VALUE
• Changes endorsed according to EP

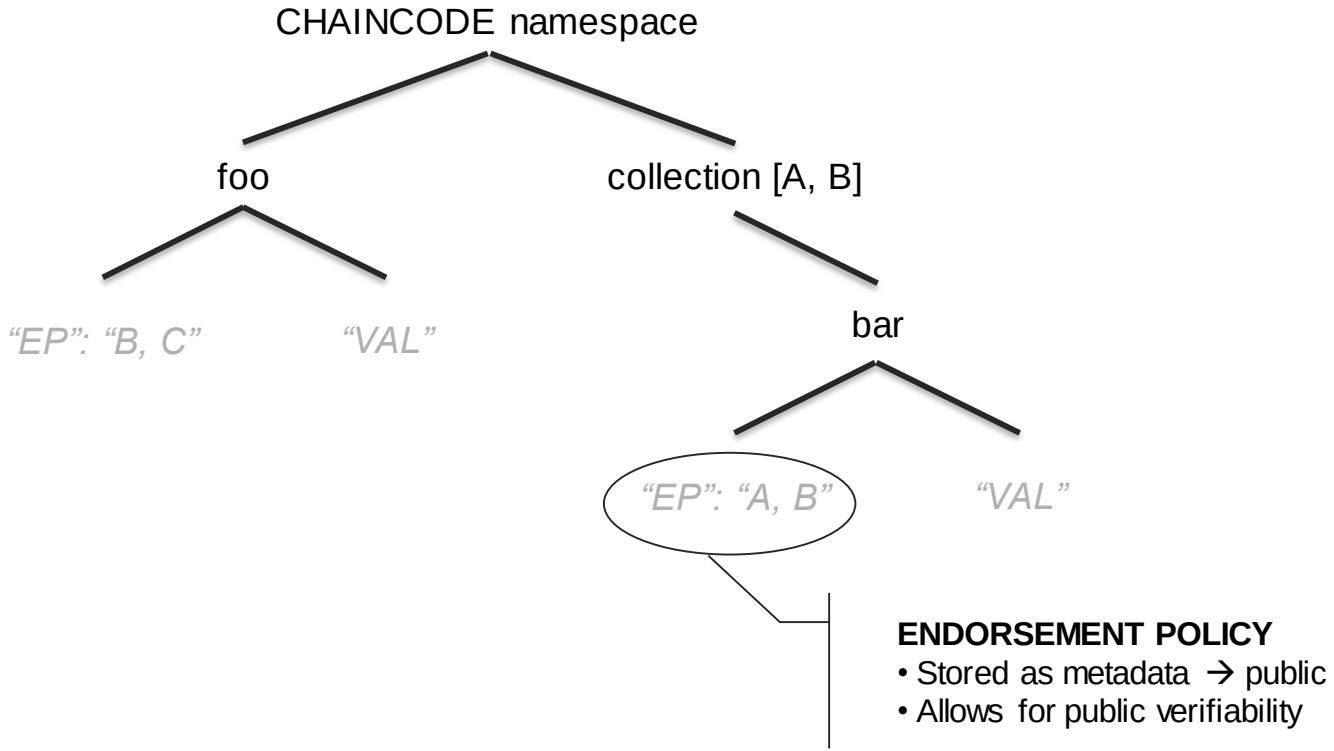
On Ledger Representation



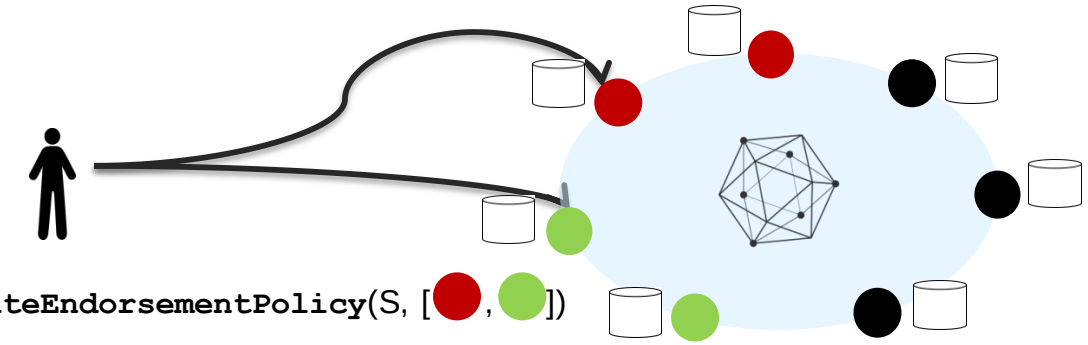
COLLECTION

- Can be used together with key-level metadata
- Way to control both dissemination and endorsement
- Dissemination policy must be superset of EP

On Ledger Representation

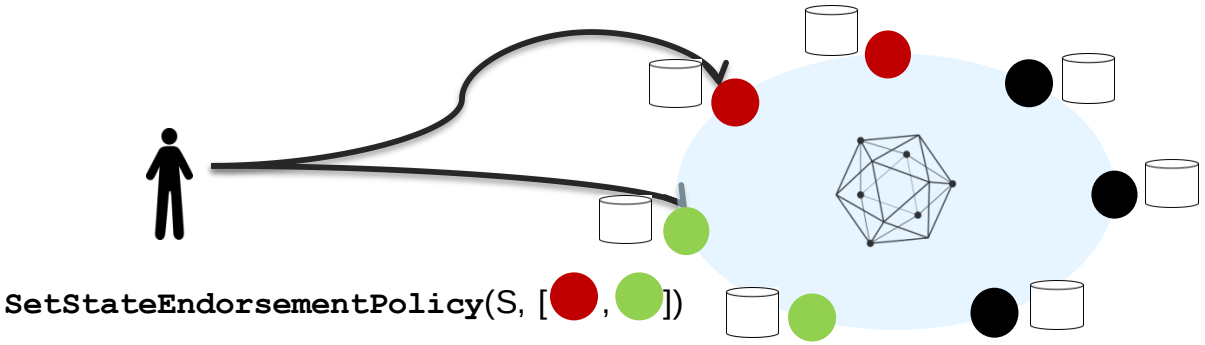


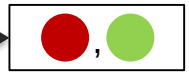
Workflow



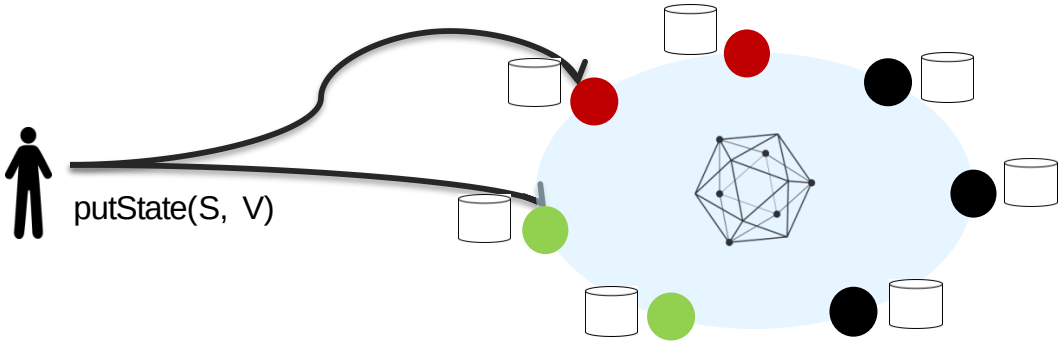
SetStateEndorsementPolicy(S, [,])

Workflow



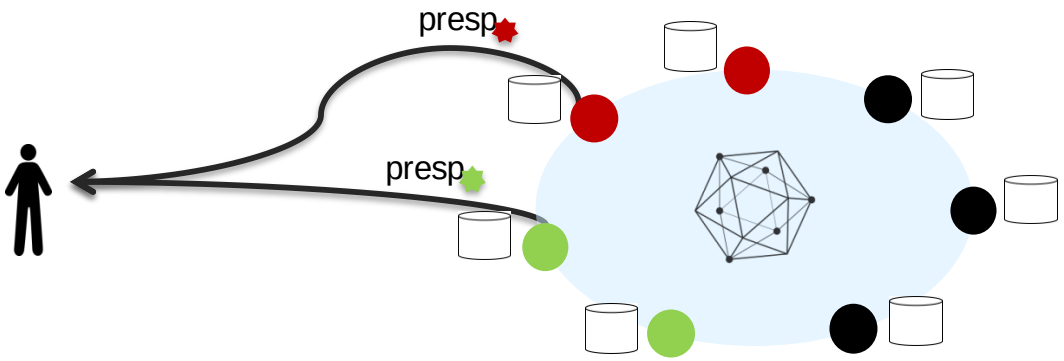
S.METADATA[EP] → 

Workflow



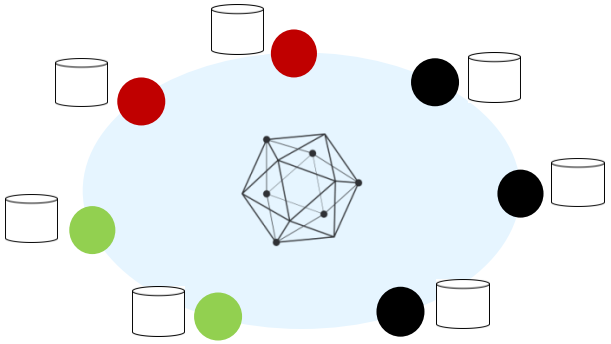
S.METADATA[EP] → 

Workflow

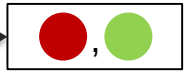


S.METADATA[EP] → 

Workflow

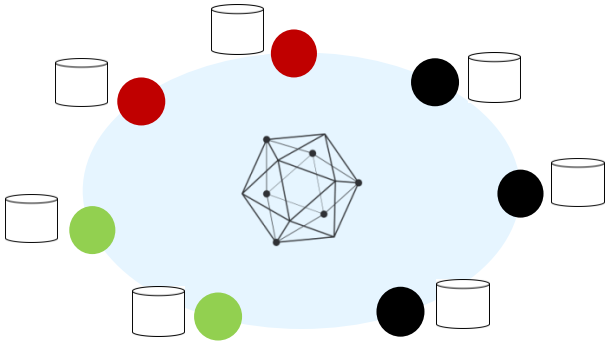


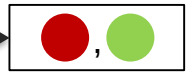
S.METADATA[EP] →



Workflow

1. Lookup S.METADATA[EP]
2. Verify that tx complies with policy therein
3. If so, commit; else, abort



S.METADATA[EP] → 

Integration with plugins

- State-based endorsement will be delivered as a default fabric feature
- Default validation plugin will be extended to support key-level endorsements
- No change required to endorsement plugin

- Validation plugin must perform key-level endorsement policy lookup for every key
  - Even for keys that never had an endorsement policy set
  - (Possible) ledger optimization: avoid trip to DB and return nil endorsement policy for keys that never had a key-level EP set

Validation Rules

Validation	Key-level validation parameter on the ledger	Key-level validation parameter in the rwset
Standard case: check against chaincode EP	no	no
Creation of key-level EP: check against chaincode EP	no	yes
Value update for key with key-level EP: check against key-level EP in the ledger	yes	no
State-level EP update: check against key-level EP in the ledger	yes	yes

NOTE: it is now possible that a tx is committed without a check against the cc EP!

Leder metadata interface

```
type QueryExecutor interface {
      ...
      // GetStateMetadata returns the metadata for given namespace and key
      GetStateMetadata(namespace, key string) (map[string][]byte, error)
      // GetPrivateDataMetadata gets the metadata of a private data item
      GetPrivateDataMetadata(namespace, collection, key string) (map[string][]byte, error)
      ...
}

type TxSimulator interface {
      QueryExecutor
      ...
      // SetStateMetadata sets the metadata associated with an existing key-tuple
      SetStateMetadata(namespace, key string, metadata map[string][]byte) error
      // DeleteStateMetadata deletes the metadata (if any) associated with an existing key-tuple
      DeleteStateMetadata(namespace, key string) error
      // SetPrivateDataMetadata sets the metadata associated with an existing key-tuple
      SetPrivateDataMetadata(namespace, collection, key string, metadata map[string][]byte) error
      // DeletePrivateDataMetadata deletes the metadata associated with an existing key-tuple
      DeletePrivateDataMetadata(namespace, collection, key string) error
      ...
}
```

Metadata behaviour

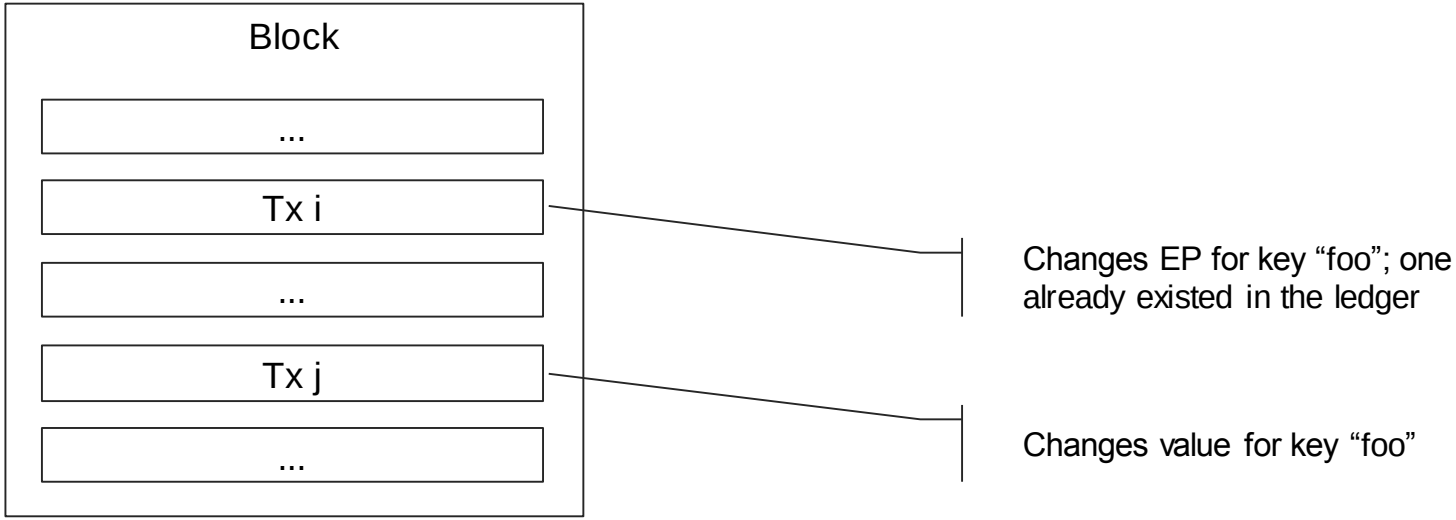
- Endorsement policy is stored as ledger key metadata
  - Endorsement policy is currently only user of this facility
- Metadata is a map[mkey, mvalue] per KVS key
  - Current interface only permits full map (over)write
- Metadata can't be set for non-existing KVS pair
- Tx can
  - Modify data only
    - Current metadata value will be kept
  - Modify metadata only
    - Current data value will be kept
  - Modify data and metadata
  - Delete metadata
    - Current data value will be kept
  - Delete KVS key
    - Both data and metadata deleted
- Only one read dependency exists, irrespective of whether only data, only metadata or both were read

Capabilities and fork-resistance

- In a network with some 1.3 peers and some <1.3 peers, a chaincode is executed on a 1.3 peer and it performs a SetStateEndorsementPolicy call for key “foo”. Tx is ordered and committed
  - 1.3 peer
    - Sets EP on the ledger
    - Checks future changes of “foo” against key-level EP
  - <1.3 peer
    - Ignores metadata rwset
    - Doesn’t perform any special validation

- New capability introduced and used
  - In the peer-side of the shim to ensure metadata can be set/retrieved only in a 1.3 network
  - In the standard validation plugin to perform key-level validation only in a 1.3 network

Special Case in Validation



- When Tx j is validated by the validation plugin, the status of Tx i is undetermined
  - Due to parallelism, validation of Tx i might be scheduled after that of Tx j
  - MVCC validation takes place sequentially afterwards

To ensure correctness..

- if Tx i is valid according to the validation plugin, invalidate Tx j with a new error code (like we do today for cc upgrade)
- Create lock structure to ensure that
  - Parallel validation occurs to the extent possible
  - Create lock structure to ensure validation of Tx j only starts when validity status of Tx i is known

Thank you