Collections with a local scope

Elli Androulaki, Mathias Bjoerkqvist, Angelo De Caro, Matthias Neugschwandtner, Alessandro Sorniotti

# Motivation

In many blockchain use-cases it is required to store hashed data on the ledger to satisfy privacy requirements such as GDPR. At the same time, this means the raw data needs to be distributed to a subset of organisations, which is done in Hyperledger Fabric by sharing preimages.

This approach is currently implemented by introducing so-called *collections*, essentially data structures mapping a string to a list of orgs. The chaincode developer can then invoke methods to put private data into a pre-defined collection, with the guarantee that hashed data will be in the ledger and preimages are disseminated to collection members.

Dissemination is supported by the gossip component: at endorsement time, gossip pushes preimage data out to other peers. At commit time, gossip pulls missing preimage data from other peers.

In this document we will illustrate a new approach that supports stronger confidentiality properties at the cost of moving to the client additional data replication related responsibilities. This approach will get Hyperledger Fabric up to speed with respect to other DLT platforms such as Corda and Quorum that **consider the same confidentiality model**.

## Stronger confidentiality: concealing collection membership

The current approach to private data leaks information about who’s doing business with whom. Indeed, collection configuration is stored on the ledger (detailing which groups of entities are planning to conduct private business together). Furthermore, every private write reveals the collection name, and hence, the set of orgs that are party to that transaction. This effectively voids the benefits of privacy-preserving authentication mechanism such as tcerts and idemix.

## Confidentiality offered by other DLT platforms

Other DLT platforms such as Corda and Quorum support this approach.

### Corda

In Corda, flows can be used to generate and distribute data across parties. If an entity wants to send private state to another entity, they can simply use the send capability of flows to deliver the private data to the other entity (akin to a fabric proposal being sent from the SDK of an org to the peer of another).

In contrast to Hyperledger Fabric, Corda does not have the concept of a broadcast. All transactions are scoped to the participants of the transaction and are explicitly sent to named participants.

### Quorum

Quorum supports private transactions and smart contracts by splitting public and private state. The payload of private transactions is only visible to participants whose public keys are specified in the *privateFor* parameter. A private transaction is sent to the rest of the network if and only if the sender receives nack from all the participants in that private transaction. Once all the nacks are received, the sender replace the payload with a hash of the encrypted payload and propagate the transaction. A node that is not a participant of the private transaction will just skip it. On the other hand, a node participating in the transaction will be able to replace the hashes with the actual data and execute the smart contract. The complete flow can be found at: <https://github.com/jpmorganchase/quorum/wiki/Transaction-Processing>

# Approach

Fabric should introduce support for collections with a *local* scope. A collection with *local* scope is one where private data is disseminated only within the organization of the peer receiving the chaincode invocation calling PutPrivateState. A collection configured with local scope can then be used to disseminate private preimage data to a set of organizations that is determined at runtime by the application, without the need of statically defining collection members at configuration time.

Let us explain with an example:

* Collection “MyCollection” is defined with *local* scope for chaincode “MyCC”; note that no dissemination policy is present in the collection definition;
* A client sends a transaction proposal to peers P1 of orgA and P2 of orgB to invoke “MyCC”;
* As a result of the cc invocation, the chaincode calls PutPrivateState with target collection ”MyCollection” for key “MyKey”;
* The preimage data of “MyKey” is disseminated by P1 to other peers of orgA, and by P2 to other peers of orgB;

In the case of GetPrivateState, the mechanism would not differ from how it works today: ~~if the peer finds it does not have the preimage, it can ask other peers within its org for it (gossip pull)~~gossip performs appropriate pulls to fill missing private data prior to the commit into the state, so at the time GetPrivateState is called, preimage data should be available. If the peer does not have it, the key at the current version is not part of its local collection and the peer cannot endorse.

### Members of a local collection

Members of a local collection are implicitly determined on an organization per key-version granularity. They can change on a per transaction basis, depending on whom the client sends the transaction proposal to as only these organizations will have the preimage.

If the application logic requires that an org needs to know the value of a key for which it did not endorse the proposal that “created” it (e.g., in an asset transfer scenario), the client can simply make the key available to this org by letting it endorse a proposal that sets a new version of the same key (with the same or a new value).

The minimum amount of functionality required is storing preimages *locally* and gossiping them *within* an organization, both of which are already supported by fabric today. The peer will receive proposal arguments/transient and its chaincode may call PutPrivateState on the local collection, instructing the peer to

1. Produce a RWSet with hashed data
2. Store preimage data solely in its own sideDB and disseminate to peers of its org.

# Pros

1. **Security**: this approach ensures no information is leaked other than the content of the KVS key and the (salted) hash of the KVS value. Removing the need for collection configuration and limiting gossip to preimage dissemination within an organization reduces the potential for attacks on preimage integrity;
2. **Preimage integrity**: this approach guarantees by default that the preimage is the same across orgs for the same private data write, because the orgs that received the proposal have independently generated a RWSet with hashed data. This advantage is strengthened by an endorsement policy that requires multiple orgs to endorse and thus guarantees that more than one org has the preimage.
3. **Flexibility**: cc developers don’t have to pre-install all required permutations of org members as collection definitions; they don’t have to upgrade this list in case of unforeseen combinations;
4. **Competition**: corda and quorum support this kind of flexibility and security, which is **NOT** supported by fabric with the current static collection definitions.

# Impact on the various components

Collections with a *local* scope is implemented as an extension to the current collection components and so the impact on each, and the additional code changes required to implement them, is minimal.

## Gossip

With this approach, gossip only disseminates preimages of private data within an organization. Preimages would be pushed to sufficiently many peers before a transaction calling PutPrivateState is endorsed.

The gossip pull of preimages needs to be changed. Currently, gossip will try to pull missing preimages from other peers across organizations based on two properties:

1. The peer in question matches a filter as provided by the collection configuration.
2. The peer in question endorsed the transaction, if it is known (which is not the case if Idemix is used).

While (1) is taken care of by the local collection-specific collection store and configuration, (2) is an issue: since all organizations are technically members of a local collection, a peer from org A will try to pull data from the local collection of org B - which is going to fail because preimages of the local collection are not shared across organizations. This behavior needs to be changed to honor local collections.

### Collection store & configuration

Gossip uses the concept of collection store to route private data pushes and pulls. The collection definition should be enhanced with an option to enable *local* scope, ensuring that private data written to that collection is only disseminated within an organization. As stated above, this behavior needs to be honored by the default gossip pull behavior as well.

## Committer

Between validating and committing a transaction that contains writes into a collection with a *local* scope, the committer can perform gossip pull from other peers of its own organization to retrieve preimages. If the preimage was received, the committer commits it to its sideDB. If the preimage is not available, the tx is committed without private data.

## Salt

As we know, salt has to be used to make sure that the hashed data on the ledger does not leak information about the preimage. This can be achieved simply by requiring the invoker of a transaction that calls PutPrivateState into a collections with a *local* scope to provide a salt through the transient field. A convenience layer of the shim will prepend it to the value parameter of corresponding PutPrivateData invocations. This ensures that the same chaincode invocation at multiple peers will produce equal results that can be endorsed by all peers.

## Security considerations and Endorsement Policies

There is no change in how endorsement policies are honored when endorsing changes to local collections. That means that endorsement policies to some extent act as dissemination policies: if the EP requires a set of organizations to endorse a proposal, collections with local scope also guarantee that at least those organization will have received preimages of private data.

With collections with local scope, it is assumed that the client of an Org is trusted to disseminate the preimages to the right set of orgs; with existing collections, this trust requirement is moved to the peer of the Org. Also, readers and writers of a certain key-value pair in the KVS are the same set.

## Endorsers and information leakage

While this approach ensures that collection configuration and collection names do not leak information about owners of private data, that same information may still be leaked implicitly by the certificate of the endorsers endorsing private state changes.

Crucially however, this leakage is attributable to the ESCC/VSCC/endorsement policy chosen by the chaincode, and **not** to fabric itself. As such, this leakage can be addressed outside of fabric proper, for instance by developing custom ESCC/VSCC (e.g. one where changes to private state need to be endorsed by ephemeral keypairs that do not leak fabric organization membership information). A complete description of such approach will be the subject of a follow-up design document.

# Discussion / Outlook

* The local collection could be used to disseminate organisation private system information; for example an org can leverage a static collection with its own org in the access policy to disseminate signing identities to its peers associated to a chaincode or other secret information tagging a certain chaincode.
* Adding more fine-grained, privacy-preserving endorsement policies, potentially using multiple signing identities per peer; signing identities distribution
* Read-set vs write-set: a scenario where a CC modifies two different pairs, pairA and pairB at the same time leads to a problem if pairA and pairB are supposed to have different scope, i.e. pairA should only be seen by org1 and org2 whereas pairB should only be seen by org1 and org3. In this case, sending the proposal to all three orgs would not honor the scope (only org1 should endorse).

In the case pairA and pairB depend on one another, one option in such a case would be that the client provides the missing (preimage) data to endorse the CC on either org2 or org3 via the transient field, alas making an exception to the scope assumption.