In a discussion with @Michael yesterday it became clear to me that we should explicate what is meant with “observer dependence” and things being “contingent on data availability”.

It feels to me that a useful description of the possible case distinctions would be downstream of a (partial) specification of the information flow control system as we roughly envision it.

On the one hand, we might want to enumerate the potential case distinctions the design or implementation currently enables “close to the metal”.

On the other hand, we should revisit the model ofIFC drafted here to check if it can map to all (or at least the desired) settings.

Somewhere in the middle we probably want to describe some patterns in respect to how they will appear “in the wild” and be used for actual application implementation, ideally linking their explanations to expressions in the abstract model, as well as the concrete primitives needed to implement them.

An incomplete enumeration draft could look like this:

- Fully transparent TX: All plaintext for all resources available.
- Fully transparent TX: Some plaintext available, rest needs to be queried?
- Mixed transparent/shielded TX: Plaintext for transparent resources available, proofs attached to nullifiers.
- Fully shielded TX: All plaintext for all resources available, so proofs can be computed.
- …

(Note: I’m not too familiar with the workings of shielded TXs and the ZK components, so the above might make absolutely no sense.)

Depending on the application, the developers will need to implement code paths for each case, even if we provide more developer friendly abstractions for these patterns, since they differ not only in implementation details, but semantics.

We should also make explicit in the interfaces and documentation of juvix that local DA can and should be used to populate, e.g. evaluation contexts for predicates over TXs, and likely provide utility functions to improve the quality of life for developers.

Some examples that arose yesterday:

nullifier ↔ resource correspondence

A TX struct contains nullifiers for consumed resources, as well as commitments for created resources.

Since commitments are the hash of the resource the correspondence of commitment → resource is clear, and the data can be retrieved, should it be available to the user making the request.

For nullifiers, that is less clear: If the resource and the nullifier key are available to a user, they can compute the nullifier.

Are there any cases where users are supposed to have access to nullifiers and resources, but not their keys? (@vveiln)

If that is the case, do we want to provide a map that ties nullifiers to resources?

mixed shielded / transparent TXs

In case only the nullifiers are available to an entity, it will not be able to compute any proofs, but if the proofs are shipped with them, they could be plugged into the validation of the TX.

For example, I get plaintext for some resources, but none for others. Then I can compute proofs for all the resource logics of the plaintext available ones, and fill in some gaps using nullifiers and (succinct and/or zk) proofs provided with them.

How are proofs and nullifiers tied together? What information can be disclosed about the nullifiers and proofs, i.e. could they contain hints about the resource kind, s.t. tranparent resource logics can use them during evaluation?

I assume that some people thought about this already, but I am unaware of the current state of implementation or documentation of these concepts, so if thi, or parts of it are closed problems, I’m happy to just be linked to the artifacts and help unify the pieces into a coherent and accessible document.

Feedback by @mariari @Michael @vveiln @xuyang @paulcadman @cwgoes would be welcome