TSS Ephemeral Ledger State Proposal

Current working proposal for accommodating concurrent dependant single ledger TSS requests

Context

State of the Turing Signing Server

Requirements

1. In question accounts must be fully controlled by the TSS network.
    - Otherwise operations external to the ephemeral ledger state could be in play which we cannot know about or react to which could alter the flow.
2. All incoming mutation operations must be batched into single or sequential transactions. i.e. requests to the Turrets wouldn't respond with a unique XDR to a request but rather a pooled XDR(s) for all requests for that ledger.
    - Otherwise early operation failures could affect later operations breaking the flow. The system would be cake to lock up with intentionally failing operations.

Proposal

When a contract is uploaded to a Turret rather than responding with a signing key to be attached to an external account the Turret responds with both the account and the signing key. This account would be a joint account entirely under the control of the Turret network selected when the contract was uploaded. The contract upload request would need to specify which Turrets it intended to upload to and each Turret would respond then with the same signed XDR for creating and multisigning a brand new account over which the request initiator never has any signing control over whatsoever. The goal again being to ensure the account for which the Turrets are signing is fully and only controlled by the Turret network itself. This can be done currently but it cannot be ensured. You can see the account is multisigned but you cannot be sure those multisigns are actually Turrets. This could be accomplished via a reverse lookup on the signers and just pinging each known Turret to see if any of them own the signing key in question but that is likely more cumbersome than simply having the Turrets generate both the account and signing keys for the account in response to a contract upload.

Once we’re assured that the controlled account is fully and only owned by the TSS network we can progress to the next step.

Currently contract requests respond individually with a singular XDR to each incoming request. Often this is fine but for high throughput and concurrent requests this can cause a problem of replication and inaccurate XDR generation. (See Context link) The goal then is to batch or hold incoming concurrent requests until the current ledger has closed and then to deliver a single batched transaction composed of all valid incoming operations during the time period in question. Incoming requests to a Turret would slot themselves for a specific time period and assuming all Turrets arrived at the same aggregate conclusion the “super” XDR would be valid and successful when submitted. Since the controlled account is fully under Turret control no outside influences can affect the account such to cause the “super” XDR to fail. The contract simply filters out invalid operations, stores them for a short period of time, and then reassembles and signs for a single “super” XDR once the time is right.

Timing and failure handling become the biggest issues here where all requests must be identical to all Turrets and the timing must be within the same range. Time is relatively easy to handle by specifying a ledger or sequence slot and then just waiting for all Turrets to respond after {x} amount of time. Redundancy becomes the way to fight failure. If one Turret is too slow or fails to deliver the hope would be that another would succeed and your signing threshold for the “super” XDR would be met.