Shielded Sync

Before performing MASP operations such as querying shielded balances, you will need to sync the local shielded context with the chain (this is equivalent to running the namadac shielded-sync command when using the CLI).

Syncing the shielded context directly from the chain is performance-intensive, so it's recommended to sync using a running instance of the namada-masp-indexer (opens in a new tab).

Example code

This example demonstrates how to perform a shielded-sync from a masp-indexer endpoint, and save the synced context to disk.

Add the following dependencies to yourCargo.toml :

reqwest

"0.11.4" kdam =

"0.5.2" Then, the following code will sync the shielded context and save it to the directory provided when creating the context .

use std :: time :: Duration ; use namada_sdk :: masp :: ExtendedSpendingKey ; use namada_sdk :: control_flow :: install_shutdown_signal; use namada_sdk :: error :: Error ; use namada_sdk :: wallet :: DatedSpendingKey ; use namada_sdk :: masp :: { MaspLocalTaskEnv , ShieldedSyncConfig };

// the spend key with which to check note ownership let spend_key =

ExtendedSpendingKey :: from_str ("zsknam1q0medj45qqqqpq9wh90qd9c7d9f7n5xxn89h6dl54k0jfmucwn4yk7nykxwcrjmk4ylkdnlnn3wkkd9f3ul3nyw8hv5wlsfgklzr5ghzk2spzzwm05csvl2s3rn0aq7f9w4z7guul682yrw4hsmren2k2lgdp003uuj) . expect ("Invalid spending key");

let dated_key =

DatedSpendingKey :: try_from (spend_key) . expect ("Error reading spend key");

// create a thread pool for the shielded sync let env =

MaspLocalTaskEnv :: new (500) . expect ("could not create masp env");

// progress bars for displaying sync progress let fetched = kdam :: tqdm! (total =

0 , desc =

"fetched " , animation =

"fillup" , position =

0 , force_refresh =

true , dynamic_ncols =

true , miniters =

0 , mininterval =

0 . 05);

let scanned = kdam :: tqdm! (total =

0 , desc =

"scanned " , animation =

"fillup" , position =

1 , force_refresh =

true , dynamic_ncols =

true , miniters =

0 , mininterval =

0 . 05);

let applied = kdam :: tqdm! (total =

0 , desc =

"applied " , animation =

"fillup" , position =

2 , force_refresh =

true , dynamic_ncols =

true , miniters =

0 , mininterval =

0 . 05);

// create a masp client to sync from the masp-indexer let client = reqwest :: Client :: builder () . connect_timeout (Duration :: from_secs (60)) . build () . map_err (| err | { Error :: Other (format! ("Failed to build http client: {err}")) }) . expect ("could not create client");

let endpoint =

"http://localhost:5000/api/v1" . to_string (); let url = endpoint . as_str () . try_into () . map_err (| err | { Error :: Other (format! ("Failed to parse API endpoint {endpoint:?}: {err}")) }) . expect ("failed to parse url");

let shielded_client =

IndexerMaspClient :: new (client, url, true , 100 ,);

let config =

ShieldedSyncConfig :: builder () . client (shielded_client) . fetched_tracker (fetched) . scanned_tracker (scanned) . applied_tracker (applied) . shutdown_signal (install_shutdown_signal (false)) . build ();

// shielded sync and save the results println! ("Syncing shielded context"); sdk . shielded_mut () .await. sync (env, config, None , & [dated_key], & []) .await. expect ("Could not sync shielded context"); println! ("Shielded context synced"); In the above example, one would replace the URL in theendpoint with the proper URL for the desired MASP indexer.

Constructing transfers Shielded transfers