Setting up an SDK wallet

Relevant imports

General imports

// namada_sdk::Namada is a high level interface in order to interact with the Namada SDK use namada_sdk :: Namada ; // The NamadaImpl provides convenient implementations to frequently used Namada SDK interactons use namada_sdk :: NamadaImpl ;

Wallet specific imports

// SecretKey, common and SchemeType give access to Namada cryptographic keys and their relevant implementations. Namada supports ED25519 and SECP256K1 keys. use namada_sdk :: types :: key :: common :: SecretKey ; use namada_sdk :: types :: key :: {common, SchemeType }; // Filesystem wallet utilities (stores the path of the wallet on the filesystem) use namada_sdk :: wallet :: fs :: FsWalletUtils ;

Creating a wallet from a mnemonic

The SDK can create a wallet from a mnemonic phrase. The mnemonic phrase is a 24 word phrase that can be used to restore a wallet.

let mnemonic =

Mnemonic :: from_phrase (MNEMONIC_CODE, namada_sdk :: bip39 :: Language :: English) // Assuming a cometbft node is running on localhost:26657 let http_client =

HttpClient :: new ("http://localhost:26657") . unwrap (); // Assuming wallet.toml exists in the current directory let

mut wallet =

FsWalletUtils :: new (PathBuf :: from ("wallet.toml")); // The key can be generated from the wallet by passing in the mnemonic phrase let (_key_alias, sk) =

NamadaImpl :: new (& http_client, &mut wallet, &mut shielded_ctx, & NullIo) . wallet_mut () .await let (_key_alias, sk) = namada . wallet_mut () .await . derive_store_key_from_mnemonic_code (scheme :

SchemeType :: Ed25519 , alias :

Some (alias), alias_force :

false , derivation_path : derivation_path, mnemonic_passphrase :

Some ((mnemonic . clone (), Zeroizing :: new ("" . to_owned ()))), prompt_bip39_passphrase :

false , password :

None ,) . expect ("unable to derive key from mnemonic code"); In the second part of the above function, the key is derived from the mnemonic phrase. Thealias is the name of the key that will be stored in the wallet. Thederivation_path is the path to the key in the HD wallet. Themnemonic is the mnemonic phrase that was generated earlier. Theshielded_ctx is the context for the shielded transactions. TheNullIo is the IO context for the wallet.

Generating a new wallet and saving it to the filesystem

It is also possible to create the sdk wallet from scratch. This is more involved because it requires generating a new store for the wallet to exist in.

use std :: path :: PathBuf ;

use namada :: { sdk :: wallet :: { alias :: Alias , ConfirmationResponse , GenRestoreKeyError , Store , StoredKeypair , Wallet , WalletUtils , }, types :: { address :: Address , key :: {common :: SecretKey , PublicKeyHash }, }, }; use rand :: rngs :: OsRng ;

pub

struct

SdkWallet { pub wallet :

```
Wallet < SdkWalletUtils

        , }

impl

SdkWallet { pub

fn

new (sk :

SecretKey , nam_address :

Address ) -> Self { let store =

Store :: default (); let

mut wallet =

Wallet :: new ( PathBuf :: new (), store); let stored_keypair =

StoredKeypair :: Raw (sk . clone ()); let pk_hash =

PublicKeyHash :: from ( & sk . to_public ()); let alias =

"alice" . to_string (); wallet . insert_keypair (alias, stored_keypair, pk_hash, true ); wallet . add_address ( "nam" ,
nam_address, true ); Self { wallet } } }

pub

struct

SdkWalletUtils {}

impl

WalletUtils

for

SdkWalletUtils { type

Storage

=

PathBuf ;

type

Rng

=

OsRng ;

fn

read_decryption_password () -> zeroize :: Zeroizing <std :: string :: String

        { panic! ( "attempted to prompt for password in non-interactive mode" ); }

fn

read_encryption_password () -> zeroize :: Zeroizing <std :: string :: String

        { panic! ( "attempted to prompt for password in non-interactive mode" ); }

fn

read_alias (_prompt_msg :

& str ) -> std :: string :: String { panic! ( "attempted to prompt for alias in non-interactive mode" ); }
```

fn

read_mnemonic_code () -> std :: result :: Result <namada :: bip39 :: Mnemonic , GenRestoreKeyError

{ panic! ("attempted to prompt for mnemonic in non-interactive mode"); }

fn

read_mnemonic_passphrase (_confirm :

bool) -> zeroize :: Zeroizing <std :: string :: String

{ panic! ("attempted to prompt for mnemonic in non-interactive mode"); }

fn

show_overwrite_confirmation (_alias :

& Alias , _alias_for :

& str ,) -> namada :: sdk :: wallet :: store :: ConfirmationResponse { // Automatically replace aliases in non-interactive mode ConfirmationResponse :: Replace } } The above code allows us now to construct any instance of theSdkWallet by simply passing in a secret key and the address for theNAM token. If we wish to make transfers with other tokens, we would need to add those addresses as well.

Setting up a client Generating accounts