Wallet usage

In this document a short overview of cardano-wallet and cardano-cli is given for wallet and address manipulation. Procedure of restoring wallet address and signing keys from mnemonics is given.

Mnemonics can be generated (and imported) via icarus as well, although procedure here is cli specific.

SUGGESTION: For an introduction to wallets, see here (link to intro section about wallets)

Prerequisites

Tools required for the process are:

- cardano-wallet
- cardano-cli
- cardano-address
- bech32

If you started local tools from provided doker compose file, they are all available in **`wallet-api`** container

```
NOTE: --testnet magic should be set to the appropriate number. Please

$ export TESTNET_MAGIC="<CORRECT VALUE HERE>"
```

Generate mnemonic

Create a wallet with mnemonic phrase

```
cardano-wallet recovery-phrase generate --size 24 > mnemonic-phrase.d
```

Check the generated mnemonic it should contain 24 words space separated fro
shelley wallet.

```
cat mnemonic-phrase.dat
```

Create extended root private key

Create an extended root private key from the mnemonic

```
cat mnemonic-phrase.dat | cardano-wallet key from-recovery-phrase She

OR

cat mnemonic-phrase.dat | cardano-address key from-recovery-phrase Sh
```

Generate extended payment/stake/change keys (with or without account keys)

Generate extended stake and payment private keys directly from the root private key:

```
cat root.xprv | cardano-address key child 1852H/1815H/0H/2/0 > stake.
cat root.xprv | IDX=0 cardano-address key child 1852H/1815H/0H/$(echc
```

Or

Create two keys:

- extended account private key (will be used to further derive your payment/staking private/public keys)
- extended account public key (this is the same as the "wallet public key" in Daedalus for example)

```
cat root.xprv | cardano-wallet key child 1852H/1815H/0H \
| tee acct.xprv | cardano-wallet key public --with-chain-code > acct.
```

Use the extended account private key to further derive the address-specific payment/stake keypairs. Following will create an extended signing key

```
cat acct.xprv | cardano-wallet key child <DERIVATION_PATH>
```

Where `` `<DERIVATION_PATH>` `` is one of the following (`` `n` `` is address index):

- `` `0/n` `` for payment keypairs
- `` `1/n` `` for change-address payment keypairs
- `` `2/0` `` for the account-level staking keypair

For example:

```
cat acct.xprv | cardano-wallet key child 2/0 > stake.xprv
cat acct.xprv | cardano-wallet key child 0/$(echo 0) > payment.xprv
cat acct.xprv | cardano-wallet key child 1/$(echo 0) > change.xprv
```

Create public extended keys from private

```
cat payment.xprv | cardano-address key public --with-chain-code > pay
cat stake.xprv | cardano-address key public --with-chain-code > stake
cat change.xprv | cardano-address key public --with-chain-code > chan
```

Conversion to regular private and public keys

Generate base address candidate

```
cat payment.xpub \
| cardano-address address payment --network-tag $(echo "testnet") \
| cardano-address address delegation $(cat stake.xpub) > base.addr_ca
```

Inspect the base address candidate

```
cat base.addr_candidate | cardano-address address inspect
```

Fix line end on base address candidate

```
cat base.addr_candidate | bech32 | bech32 addr_test > base.addr_candi
mv base.addr_candidate_test base.addr_candidate
```

```
cat base.addr_candidate
```

Convert cardano-addresses extended signing keys to corresponding Shelley-format keys.

```
cardano-cli key convert-cardano-address-key --shelley-payment-key --s
cardano-cli key convert-cardano-address-key --shelley-stake-key --sig
```

Get verification keys from signing keys.

```
cardano-cli key verification-key --signing-key-file stake.skey --veri
cardano-cli key verification-key --signing-key-file payment.skey --ve
```

Get non-extended verification keys from extended verification keys.

```
cardano-cli key non-extended-key --extended-verification-key-file sta
cardano-cli key non-extended-key --extended-verification-key-file pay
```

Build stake and payment addresses

```
cardano-cli stake-address build --stake-verification-key-file stake.v
cardano-cli address build --payment-verification-key-file payment.vke
```

```
cardano-cli address build \
    --payment-verification-key-file payment.vkey \
    --stake-verification-key-file stake.vkey \
    --testnet-magic $TESTNET_MAGIC \
    --out-file base.addr
```

Compare `base.addr` and `base.addr_candidate` they must match, then rename it to payment address

```
mv base.addr payment.addr
```

Move final keys and addresses to safe secret folder:

```
cp stake.vkey stake.skey stake.addr payment.vkey payment.skey payment
```

Or inspect them for further usage

```
# cat stake.vkey
{
    "type": "StakeVerificationKeyShelley_ed25519",
    "description": "",
    "cborHex": "5820...REDACTED..."
}

# cat stake.skey
{
    "type": "StakeExtendedSigningKeyShelley_ed25519_bip32",
    "description": "",
    "cborHex": "5880...REDACTED..."
}

# cat stake.addr
stake_test1...REDACTED...
```

```
# cat payment.vkey
{
    "type": "PaymentVerificationKeyShelley_ed25519",
    "description": "",
    "cborHex": "5820...REDACTED..."
}

# cat payment.skey
{
    "type": "PaymentExtendedSigningKeyShelley_ed25519_bip32",
    "description": "",
    "cborHex": "5880...REDACTED..."
}

# cat payment.addr
addr_test1...REDACTED...

# cat payment-only.addr
addr_test1...REDACTED...
```

Create a wallet from mnemonics and passphrase

```
cardano-wallet wallet create from-recovery-phrase <WALLET_NAME>
cardano-wallet wallet list
cardano-wallet wallet delete <WALLET_ID>
```