Local Wallet

Overview

To transact with fet on the Fetch.ai ledger, you will need to manage your wallet with you agent. We use the Cosmpy library for ledger actions, coupled with in library support for Cosmpy in uAgents.

Prerequisites

Make sure you have read the following resources before going on with this guide:

- Quick Start Guide for uAgents Framework
- Creating your first agent
- Agents address
- Almanac contract
- Register in Almanac
- CosmPy installation
- CosmPy: Wallets and private keys

Imports needed

- CosmPy(opens in a new tab)
- uAgents(opens in a new tab)

Understanding your wallet

Wallets allow individuals to manage, store and transact across blockchain networks. Wallets are composed of a public and private key pair. The private key is a unique and secret cryptographic code that provides ownership and control over the wallet and allows individuals to authorize transactions signing. The public address is a cryptographic identifier that allows individuals to receive and view transactions.

Changes to the state of a network are achieved by broadcasting transactions. To accomplish this, we can use CosmPy in a straightforward way to create and manage private keys and addresses. In this guide, you will learn how to both generate a new private key as well as recover a previously generated one.

The examples below outlines how to achieve both of these operations.

Generating a new private key

First of all, import the needed modules and then use the PrivateKey class to create yourprivate_key :

Import necessary classes

from cosmpy . aerial . wallet import LocalWallet from cosmpy . crypto . keypairs import PrivateKey

Creating a random private key

private_key

PrivateKey ()

Recover an existing private key

Let's start by extracting the private key and convert it into a base64 encoded string. You can do this on macOS or Linux for the Fetch.ai network using the FetchD CL.

An example is provided in the code snippet below:

Extract the private key and convert it into a base64 encoded string

fetchd keys export mykeyname --unsafe --unarmored

hex

| xxd - r - p | base64 Once you have extracted and converted the key into a base64 encoded string you can recover to an existing private key by following along the provided example below:

Import necessary classes

from cosmpy . aerial . wallet import LocalWallet from cosmpy . crypto . keypairs import PrivateKey

Recovering an existing private key

private_key

PrivateKey (")

Here is where you provide the base64 encoded private key string

ThePrivateKey object is one of CosmPy's low level primitives. This is why it is generally paired with aWallet object. You can now create a local wallet using theLocalWallet class from the CosmPy library and initialize it with theprivate_key you generated or recovered earlier. TheLocalWallet class is used to manage a local wallet associated with a private key for interacting with the Cosmos blockchain.

With this wallet object, you can perform various operations, such as generating the corresponding public address associated with the private key and signing transactions. You can also query useful information such as the address from the wallet directly. The example below showcases how to achieve both of these outlined operations:

wallet

LocalWallet (private_key) print (wallet. address ())

will print the address for the wallet

From mnemonic

You can also use an account's mnemonic phrase to get the associated private key. The example provided below showcases how to achieve that:

from cosmpy . aerial . wallet import LocalWallet

mnemonic

"person knife december tail tortoise jewel warm when worry limit reward memory piece cool sphere kitchen knee embody soft own victory sauce silly page"

wallet

LocalWallet . from_mnemonic (mnemonic)

An Agent that checks for a transaction, and sends funds

A simple agent that creates, or loads in a wallet, then validates a transaction has been received.

Self hosted local_wallet_agent.py from cosmpy . aerial . wallet import LocalWallet from cosmpy . aerial . client import LedgerClient , NetworkConfig from cosmpy . crypto . keypairs import PrivateKey from uagents import Agent , Context , Model from uagents . network import get_faucet , wait_for_tx_to_complete

mainnet_ledger

LedgerClient (NetworkConfig. fetchai_mainnet ())

class

RequestWithTX (Model): message :

str tx_hash :

str

class

DataResponse (Model): message :

str

class

PaymentError (Model): message :

str tx_hash :

str

DataSellingAgent

Agent (name = "DataSellingAgent" , seed = "dwefwegferdwdwedgko4o430490349jf340-jffjweiopfnw" , port = 8001 , endpoint = ["http://localhost:8001/submit"],)

print (DataSellingAgent.address)

AMOUNT

1 DENOM =

"afet" DATA_TO_SELL =

"..."

at first you may want to generate a wallet

my_wallet

LocalWallet . generate ()

or open one from a seed you've set

my_wallet = LocalWallet.from_unsafe_seed("registration test wallet")

pk = my_wallet._private_key

or from a pk you already have

wallet =

LocalWallet(PrivateKey("T7w1yHq1QIcQiSqV27YSwk+i1i+Y4JMKhkpawCQIh6s="))

...

@DataSellingAgent . on_message (model = RequestWithTX) async

def

message_handler (ctx : Context ,

sender :

str ,

msg : RequestWithTX): ctx . logger . info (f "Received message from { sender } : { msg.message } ")

mainnet_ledger . query_tx (msg.tx_hash) tx_resp =

await

wait_for_tx_to_complete (msg.tx_hash, mainnet_ledger)

coin_received

tx_resp . events ["coin_received"] if (coin_received ["receiver"]

==

str (my_wallet.address) and coin_received ["amount"]

==

f " { AMOUNT }{ DENOM } ") : ctx . logger . info (f "Transaction was successful: { coin_received } ") await ctx . send (sender, DataResponse (message = DATA_TO_SELL))

else : await ctx . send (sender, PaymentError (message = "Incorrect tx" , tx_hash = msg.tx_hash))

if

name

==

"**main**" : DataSellingAgent . run () Validating a transaction with Cosmpy and uAgents is really easy, and to test the above agent you just need to replicate theModels and send aRequestWithTX in any function in another agent:

@DataBuyingAgent.on_event("startup") async def introduce_agent(ctx: Context): ctx.logger.info("Starting DataBuyingAgent...") await ctx.send(agent_address, RequestWithTX(message="buy", tx_hash="D165966A6B9269EHHHHH7C8F659F1CDA871C8CD83F9102DD35A437211F3DE0CF")) Last updated on October 17, 2024

Was this page helpful?

You can also leave detailed feedback[on Github]

Multi-file agent pipeline for AI Engine: Network of Primary and Secondary functions in AgentverseAgents Name Service

On This Page