Transaction Structure

A transaction on the Sei blockchain consists of the following main components:

•Body : Contains the list of messages to be executed, memo, timeout height, and extension options.

•Auth Info : Includes signer information and fee details.

•Signatures : Holds the signatures of the signers authorizing the transaction.

Here’s a general structure of a transaction:

```
{ "body" : { "messages" : [ // List of messages ] , "memo" :

"" , "timeout_height" :

"0" , "extension_options" : [] , "non_critical_extension_options" : [] } , "auth_info" : { "signer_infos" : [] , "fee" : { "amount" : [ { "denom" :

"usei" , "amount" :

"100000" } ] , "gas_limit" :

"200000" , "payer" :

"" , "granter" :

"" } } , "signatures" : [] }
```

Adding Multiple Messages

You can add multiple messages of different types in the messages array. Here’s how you can do this using CosmJS and seid.

Using CosmJS

```
const { DirectSecp256k1HdWallet } =

require ( "@cosmjs/proto-signing" ); const { SigningStargateClient ,

assertIsBroadcastTxSuccess ,

coins } =

require ( "@cosmjs/stargate" );

async

function

sendTransaction () { const

rpcEndpoint

=

"https://rpc-endpoint" ; const

mnemonic

=

"your mnemonic" ; const

wallet

=

await

DirectSecp256k1HdWallet .fromMnemonic (mnemonic , { prefix :
```

```
"sei" }); const [ account ] =
await
wallet .getAccounts ();
const
client
=
await
SigningStargateClient .connectWithSigner (rpcEndpoint , wallet);
const
msgSend
= { typeUrl :
"/cosmos.bank.v1beta1.MsgSend" , value : { fromAddress :
account .address , toAddress :
"sei1destinationaddress" , amount :
coins ( 1000 ,
"usei" ) , } , };
const
msgDelegate
= { typeUrl :
"/cosmos.staking.v1beta1.MsgDelegate" , value : { delegatorAddress :
account .address , validatorAddress :
"sei1validatoraddress" , amount :
coins ( 500 ,
"usei" ) , } , };
const
fee
= { amount :
coins ( 2000 ,
"usei" ) , gas :
"200000" , };
const
memo
=
"Transaction with multiple messages" ;
const
result
=
```

await

client .signAndBroadcast (account .address , [msgSend , msgDelegate] , fee , memo); assertIsBroadcastTxSuccess (result); console .log (result); }

sendTransaction () .catch (console .error);

Using seid

To create and broadcast a transaction with multiple messages using seid, follow these steps:

1. Define the Unsigned Transaction
2. :

Create an unsigned-tx.json file with multiple messages:

{ "body" : { "messages" : [{ "@type" :

"/cosmos.bank.v1beta1.MsgSend" , "from_address" :

"sei1sourceaddress" , "to_address" :

"sei1destinationaddress" , "amount" : [{ "denom" :

"usei" , "amount" :

"1000" }] } , { "@type" :

"/cosmos.staking.v1beta1.MsgDelegate" , "delegator_address" :

"sei1sourceaddress" , "validator_address" :

"sei1validatoraddress" , "amount" : { "denom" :

"usei" , "amount" :

"500" } }] , "memo" :

"Transaction with multiple messages" , "timeout_height" :

"0" , "extension_options" : [] , "non_critical_extension_options" : [] } , "auth_info" : { "signer_infos" : [] , "fee" : { "amount" : [{ "denom" :

"usei" , "amount" :

"2000" }] , "gas_limit" :

"200000" , "payer" :

"" , "granter" :

"" } } , "signatures" : [] } 1. Sign the Transaction 2. :

seid tx sign unsigned

tx .json -- from < your_account

-- output - document = signed - tx .json -- node < YOUR_RPC_URL

1. Broadcast the Signed Transaction
2. :

seid tx broadcast signed

tx .json -- node < YOUR_RPC_URL

Last updated onMay 23, 2024 Oracles HD Path & Coin Types