Permissioned Keys

Overview

Permissioned Keys are a dYdX specific extension to the Cosmos authentication system that allows an account to add custom logic for verifying and confirming transactions placed on that account. For example, an account could enable other accounts to sign and place transactions on their behalf, limit those transactions to certain message types or clob pairs etc, all in a composable way.

To enable this there are currently six types of "authenticator" that can used, four that enable specific authentication methods and two that allow for composability:

Sub-Authenticator Types

- SignatureVerification
- – Enables authentication via a specific key
- MessageFilter
- – Restricts authentication to certain message types
- SubaccountFilter
- – Restricts authentication to certain subaccount constraints
- ClobPairIdFilter
- – Restricts transactions to specific CLOB pair IDs

Composable Authenticators

- AnyOf
-
    - Succeeds ifany
- of its sub-authenticators succeeds
- AllOf
-
    - Succeeds only ifall
- sub-authenticators succeed

The following example demonstrates how “Bob” can add an “AllOf” authenticator that grants “Alice” permission to submit only “MsgPlaceOrder” transactions on Bob’s account.

Example: Adding an AllOf Authenticator With SignatureVerification + MessageFilter

1. Building Sub-Authenticators

To allow Alice’s key to sign transactions, and to restrict transactions to the “MsgPlaceOrder” message type, construct two sub-authenticators:

- SignatureVerification: Uses Alice’s public key
- MessageFilter: Only allows"/dydxprotocol.clob.MsgPlaceOrder"
- messages.

In Go (pseudocode):

subAuths := []aptypes.SubAuthenticatorInitData{ { Type: "SignatureVerification" , Config: AlicePrivateKey. PubKey (). Bytes (), }, { Type: "MessageFilter" , Config: [] byte ("/dydxprotocol.clob.MsgPlaceOrder"), }, }

// Marshal sub-authenticators into JSON data for the AllOf authenticator. allOfData, err := json. Marshal (subAuths) require. NoError (t, err) 2. Creating the AllOf Authenticator

We want to associate this new authenticator with Bob’s account. Therefore, Bob must be the “Sender” of a “MsgAddAuthenticator.”

The “AuthenticatorType” is "AllOf," and the “Data” is the marshaled sub-authenticators from step 1.

addAllOfMsg :=

& aptypes.MsgAddAuthenticator{ Sender: constants.BobAccAddress. String (), AuthenticatorType: "AllOf" , Data: allOfData, } This tells the chain to store a new “AllOf” authenticator on Bob’s account. The chain will return an authenticator ID (for example, 0).

1. Submitting the Add Authenticator Transaction

Because Bob owns the account, Bob’s signature is required to add this authenticator:

tx, err := testtx. GenTx (ctx, txConfig, []sdk.Msg{addAllOfMsg}, someFeeCoins, // transaction fee someGasAmount, // gas chainID, [] uint64 {bobAccNum}, // Bob’s account number [] uint64 {bobSeqNum}, // Bob’s sequence []cryptotypes.PrivKey{bobPrivKey}, // signature by Bob []cryptotypes.PrivKey{bobPrivKey}, // fee payer is also Bob nil , // no additional authenticators referenced here) Broadcast the transaction to the network. If successful, Bob’s account now has an AllOf authenticator. The chain references it by an ID (e.g. 0).

Example: Submitting an Order With the New Authenticator

After adding this AllOf authenticator, Bob implicitly allows transactions on his account, but only if they match both sub-authenticators (Alice must sign, and the message must be “MsgPlaceOrder”).

1. Creating a PlaceOrder Message

Construct a “MsgPlaceOrder.” In Go:

placeOrderMsg := clobtypes. NewMsgPlaceOrder (// Place order using Bob's account details),) 5. Building and Signing the Transaction

Even though the account belongs to Bob, the AllOf authenticator requires Alice’s signature. Hence, we sign the PlaceOrder transaction with Alice’s private key:

orderTx, err := testtx. GenTx (ctx, txConfig, []sdk.Msg{placeOrderMsg}, someFeeCoins, someGasAmount, chainID, [] uint64 {bobAccNum}, // Bob's account number [] uint64 {bobSeqNum}, // Bob's sequence []cryptotypes.PrivKey{alicePrivKey}, // sign with Alice’s key []cryptotypes.PrivKey{alicePrivKey}, // Alice is also paying fees [] uint64 { 0 }, // reference the AllOf authenticator by ID = 0) Broadcast this transaction. During verification:

- SignatureVerification sub-authenticator checks that Alice signed the transaction.
- MessageFilter sub-authenticator checks that the message is/dydxprotocol.clob.MsgPlaceOrder
- AllOf requires that both sub-authenticators pass, which they do if Alice is indeed signing and the message is of the correct type.

Once successful, Bob’s account effectively “permits” the order to be placed, but only under the conditions enforced by the AllOf authenticator.

Typescript Client Example

Our Typescript client has helper functions for:

- Adding an authenticator (link(opens in a new tab)
-)
- Removing an authenticator (link(opens in a new tab)
-)
- Viewing all authenticators for an given address (link(opens in a new tab)
-)

As well as an end to end example, adding an authenticator and placing a short term order with the authenticated account.

Link to e2e example(opens in a new tab)