Aiken-lang Escrow New features and functions development

1. Aiken escrow

- Secure exchange of assets between two parties
- The escrow smart contract allows two parties to exchange assets securely. The contract holds the assets until both parties agree and sign off on the transaction.

There are 4 actions available to interact with this smart contract:

- initiate escrow and deposit assets
- deposit assets
- complete escrow
- cancel escrow

Install package

- First you can to install the @meshsdk/contracts package:

npm install @meshsdk/contract

2. Initialise the contracts

To initialize the escrow, we need to initialize a provider, MeshTxBuilder and MeshEscrowContract.

```
import { MeshEscrowContract } from "@meshsdk/contract";
import { MeshTxBuilder } from "@meshsdk/core";

const blockchainProvider = new BlockfrostProvider('<Your-API-Key>');

const meshTxBuilder = new MeshTxBuilder({
fetcher: blockchainProvider,
submitter: blockchainProvider,
});

const contract = new MeshEscrowContract({
mesh: meshTxBuilder, escrow
fetcher: blockchainProvider,
wallet: wallet,
networkId: 0,
});
```

3. Initiate escrow

Initiate Escrow

- An escrow is initiated by one of the party, user A, by locking assets to the escrow contract.
- initiateEscrow() initiate an escrow. The function accepts the following parameters:
- escrowAmount (Asset[]) - a list of assets user A is trading
- The function returns a transaction hex if the escrow is successfully initiated.

4. Recipient deposits

-
Recipient Deposit

- User B can deposit assets into the escrow after initiation step (initiateEscrow()).

> recipientDeposit() deposit assets into the escrow. The function accepts the following parameters:

- escrowUtxo (UTxO) - the utxo of the transaction on the contract
- depositAmount (Asset[]) - a list of assets user B is trading
- We have provided a very handle function, getUtxoByTxHash, which will return the UTxO object for a given transaction hash.

5. Complete escrow

Complete Escrow
→
A user can complete an escrow if the terms of the agreement are met. The completion can be initiated by any recipient of the escrow.

completeEscrow() complete an escrow. The function accepts the following parameters:

escrowUtxo (UTxO) - the utxo of the transaction in the script to be completed
Important: This is a multi-signature transaction. Both users must sign the transaction to complete the escrow.

6. Cancel escrow

Person A signs the transaction
User A completes the escrow by calling the `completeEscrow()` function and partial sign the transaction.

Tx hash
Tx hash

const utxo = await contract.getUtxoByTxHash('');
const tx = await contract.completeEscrow(utxo);
const signedTxUserA = await wallet.signTx(tx, true);
Connect wallet to run this demo

Connect Wallet
Person B signs and submits the transaction
The signed transaction will be handled to User B to sign the transaction and submits it to the blockchain to complete the escrow.

Tx hash
Tx hash
Transaction CBOR
Transaction CBOR

const signedTxUserB = await wallet.signTx(signedTxUserA, true);
const txHash = await wallet.submitTx(signedTxUserB);
Connect wallet to run this demo

Connect Wallet
Cancel Escrow

A user can cancel an escrow if the other party fails to fulfill the terms of the agreement. Cancel can be initiated by any users who have partcipated in the escrow and can be done at any time before complete. Canceling the escrow will return the assets to the respective users.

cancelEscrow() cancel an escrow. The function accepts the following parameters:

escrowUtxo (UTxO) - the utxo of the transaction to be canceled
We have provided a very handle function, getUtxoByTxHash, which will return the UTxO object for a given transaction hash.

Test result

```
┌── tests/escrow ──────────────────────────────────────────────────────────────
|  PASS [mem: 453434, cpu: 178460465] success_cancel
|  PASS [mem: 431138, cpu: 170363078] success_cancel_at_initiation
|  PASS [mem: 652201, cpu: 252644029] success_cancel_at_active_with_initiator_signed
|  PASS [mem: 661392, cpu: 255826526] success_cancel_at_active_with_recipient_signed
```

| PASS [mem: 433171, cpu: 171078988] fail_cancel_at_initiation_without_signature
| PASS [mem: 638130, cpu: 247284265] fail_cancel_without_signature
| PASS [mem: 662990, cpu: 256306118] fail_cancel_without_initiator_value_returned
| PASS [mem: 663820, cpu: 256368081] fail_cancel_without_recipient_value_returned
| PASS [mem: 678543, cpu: 263661318] success_deposit
| PASS [mem: 628210, cpu: 246485477] fail_deposit_without_updating_datum
| PASS [mem: 681725, cpu: 264623369] fail_deposit_without_depositing_value
| PASS [mem: 740458, cpu: 288554997] success_complete
| PASS [mem: 710203, cpu: 277726467] fail_complete_without_initiator_signed
| PASS [mem: 683677, cpu: 264261397] fail_complete_without_value_sent_to_initiator
| PASS [mem: 717962, cpu: 280411610] fail_complete_without_recipeint_signed
| PASS [mem: 683677, cpu: 264261397] fail_complete_without_value_sent_to_recipient
└──

──────────── 16 tests | 16 passed | 0 failed

```
Summary 16 checks, 0 errors, 0 warnings
```

Cypress integration

{
"name": "Using fixtures to represent data",
"email": "hello@cypress.io",
"body": "Fixtures are a great way to mock data for responses to routes"
}