Alice Smart Contracts

Last Updated: 15 Nov 2021

daniel@, jack@

1 Motivation

1.1 Goals

- Non-custodial wallet
  - The user should own their funds.
- Earn high yield passively, backed by Anchor
  - Separate "checkings" and "savings" accounts should not be needed.
- We can subsidize transaction costs
  - Peer-to-peer payments, for example, should be free.

1.2 Problems

The simplest app would be a plain non-custodial wallet, but this would not fulfill our goals:

- Terra transactions have gas and tax costs, which is poor UX for peer-to-peer payments.
- Using Anchor requires explicit deposit and redeem transactions, not passive.

1.3 Solution: aliceUST

aliceUST is a wrapper token for aUST that:

- Supports meta-transactions for gas subsidization
- Supports authorized deposits for a seamless onramp
- Can be upgraded by Alice Software, with a timelock

2 aliceUST Contract Specification

Currently, the following functions in this section are all in one smart contract. It may make sense to split them into different contracts in the future.

2.1 aliceUST Token

The aliceUST token implements the CW20 spec without any extensions:

- `Transfer { recipient, amount }`: transfer token to another Terra account
- `Burn { amount }`: burn token
- `Send { contract, amount, msg }`: send token to a contract

2.2 Exchange rate & interest

aliceUST is a direct wrapper for aUST, and so earns interest (relative to UST) at a rate equal to aUST.

2.3 Deposit and withdraw UST

- `DepositStable { recipient }`: accept UST & mint equivalent aliceUST for the `recipient` (optional, default is TX sender)
  - Terra tax for transferring UST to Anchor is deducted
- `RedeemStable { burn_amount, recipient }`: burn aliceUST & send equivalent UST to the `recipient` (optional, default is TX sender)
  - Terra tax for transferring UST to the recipient is deducted

2.4 Anchor

The contract will hold all user funds in Anchor as aUST. On every deposit or withdraw, the contract will deposit or withdraw the appropriate amount of aUST.

2.5 Meta-transactions

Alice mobile app Terra accounts have no native coins for gas. We implement meta-transactions, where an Alice-owned Terra account acts as a relay or "gas station."

To prevent replay attacks, the contract keeps an incrementing integer nonce for each account.

User wants to execute JSON-serialized message `msg` on the contract without gas:

1. User signs `meta_tx = MetaTx { nonce, msg }` using their private key to get secp256k1 `signature`.
2. User sends `meta_tx`, `signature`, and `public_key` to the Alice server (using usual app authentication & authorization).
3. The server checks the relay transaction (e.g. verifying signature, per-account limits).
4. The server executes `Relay { meta_tx, signature, public_key }` on the token contract from an Alice-owned account that has funds for gas.
5. The token contract verifies `signature` for `meta_tx` and `pub_key`.
6. The token contract de-serializes `meta_tx = { nonce, msg }`, then verifies and increments the nonce.
7. The token contract de-serializes `msg` and executes it as the user.

2.5.1 Paying for gas in aliceUST

As a fallback option, Alice app users can pay for gas in aliceUST.

Modifying the protocol above slightly:

- Add `tip` field to `MetaTx`: aliceUST amount the relay will charge the user

The tip will be collected regardless of whether the enclosed transaction succeeds.

2.6 Pre-authorizing deposits

Issue: The Alice onramp can only send UST to the user's wallet (after some delay), not aliceUST.

To convert the UST into aliceUST seamlessly, we propose the following procedure:

1. User initiates onramp
2. User authorizes the aliceUST contract to spend the onramped UST (SendAuthorization)
3. Once onramp completes:

- Alice server executes `DepositStableAuthorized { sender, recipient, amount }` on the aliceUST contract, which sends onramped UST from the user to the aliceUST contract (MsgExec) and credits the recipient with aliceUST.
- Terra tax for MsgExec and Anchor deposit is taken from the funds sent with the TX (i.e. Alice-owned account pays)

3 Overseer Contract Specification

The overseer contract is its own admin and also the aliceUST contract's admin.

The main purpose is to manage a set of contracts. Specifically, it implements a timelock for contract migrations (upgrades).

- `InitiateMigrate { contract_addr, new_code_id, msg }`: Starts the timelock for a contract migration
- `Migrate {}`: Execute migration after timelock (`MsgMigrateContract`)

Both the overseer and the aliceUST contract allow changing contract config through migrate msg.