AIP[1]-012: Escrow Service

platfowner, liayoo 2021-02-08

Goals

- Provide a design of escrow service to reserve tokens for multilateral transactions in other services (e.g. payment)

Problem Definition

Usually, a payment transaction for a service is handled in the following steps:

1) A customer pays money for the service
2) The service provider serves the requested service
3) The money is transferred to the service provider (i.e., target account) or refunded to the customer (i.e., source account)

Requirements

For safe service transactions, we need a utility service with the following requirements:

- Customer's money can be reserved to guarantee payment when the service is provided successfully
- If the service was provided successfully, the money is safely transferred to the service provider
- Otherwise, the money is safely refunded to the customer
- The decision on the success can be made by either the customer, the service admin, or the service provider, and it's configurable.

There are basically three accounts are involved in:

- Source account (customer account)
- Target account (service provider account)
- Admin account

And accounts here means either individual accounts (e.g. **0x_abcd**) or service accounts (e.g. **payments|collaborative_ai|0x_asdf**).

[1] AI Network Improvement Proposal. Visit https://docs.ainetwork.ai for the full list.

Proposed Design

Key Ideas

- Use Service Account as value-holding accounts for escrow instances and _transfer function for value transfer
- Provide the following methods by native functions:
    - _openEscrow
        - Opens an escrow account by writing parameters:
            - source account (the customer account)
            - target account (the service provider account)
            - admin account who makes the disbursement
        - Note that the given admin account is set as the admin account of the service account. This can be either the source account or the target account or a third party account. Usually, the service owner account (e.g. the owner account of ‘collaborative_ai’ service) is used for this value.
    - _hold
        - Holds (i.e., reserves) money by transferring money to the escrow account using _transfer
    - _release
        - Disburses (i.e., transfers or refunds) money to transacting parties (the source account and the target account) in the given ratio using _transfer
        - Ratio 1 means all the money in escrow goes to the target account while ratio 0 means all the money is refunded to the source account.

Design Details

Use Cases

Payment (AS-IS):

- _claim
    - Triggered by:
      /payments/collaborative_ai/<user_addr>/<account_id>/<account_id>/claims/<record_id>/{
        amount: <claim_amount>,
        target: <target_addr>
      }
    - _transfer

        - Triggered by:
          /transfer/ payments | <service_name> | <user_addr> | <account_id> / <target_addr> / <key> / value / <claim_amount>
          e.g. /transfer/ payments | collaborative_ai | 0x_Collaborative_AI | 0 / 0x_Teachable_NLP / 12345 / value / 100
        - Value transfer:
          /service_accounts/payments/<service_name>/<user_addr>|<account_id>|<account_id>/balance ->
          /accounts/<target_addr>/balance
          e.g.
          /service_accounts/payments/collaborative_ai/0x_Collaborative_AI|0|0/balance ->
          /accounts/0x_Teachable_NLP/balance

Service Account Key Mapping

To avoid conflicts, each escrow instance is mapped to a service account instance in the following way:

- (source_account, target_account, escrow_key) -> account_key
  e.g. (0x_A_User, 0x_Teachable_NLP, 12345) -> 0x_A_User:0x_Teachable_NLP:12345

where a service account path for escrow service is defined as /service_accounts/escrow/escrow/<account_key>.

Database Paths & Native Functions

_openEscrow

- Triggered by:
  - /escrow/<source_account>/<target_account>/<escrow_key>/open_escrow/{
    admin: {
    <admin_account>: true
    }
    }
  - e.g. /escrow / **payments|collaborative_ai|0x_A_User|0** / **0x_Teachable_NLP** / **112233** / open_escrow / {
    admin: {
    0x_Collaborative_AI: true
    }
    }
- Action:
  - Validity check:
    - Check whether there is such an escrow account yet

        - Check whether the given parameters are valid ones
    - Opens an escrow account (service account) by setting the account admin:
        - /service_accounts/escrow/escrow/<account_key>/{
            admin: {
                <admin_account>: true
            }
        }
        where <account_key> is
        “<source_account>:<target_account>:<escrow_key>”.
        - e.g. /service_accounts / escrow / **escrow** / **payments|collaborative_ai|0x_A_User|0 : 0x_Teachable_NLP : 112233** / {
            admin: {
                0x_Collaborative_AI: true
            }
        }

_hold

- Triggered by:
    - /escrow/<source_account>/<target_account>/<escrow_key>/hold/<record_id>/{
        amount: <hold_amount>,
    }

e.g. /escrow / **payments|collaborative_ai|0x_A_User|0** / **0x_Teachable_NLP** / **112233** / hold / 12345 / {
    amount: 100,
}

- Action:
    - Permission check:
        - If source_account is an individual account, it’s compared with the sign address
        - If source_account is a service account, the sign address is compared with the admin addresses of the service account
    - Transfer money to escrow account from the source account by calling transfer **internally**:
        - /service_accounts/escrow/escrow/<account_key>/{
            balance: <existing_balance> + <hold_amount>,
            }
            where <account_key> is
            “<source_account>:<target_account>:<escrow_key>”.
        - e.g. /service_accounts / escrow / **escrow** / **payments|collaborative_ai|0x_A_User|0 : 0x_Teachable_NLP :**

112233 {
balance: 100 + 50,
}

- Set the action result to:
  - /escrow/<source_account>/<target_account>/<escrow_key>/hold/<record_id>/result
  - e.g. /escrow / **payments|collaborative_ai|0x_A_User** / **0x_Teachable_NLP** / **112233** / hold / 12345 / result

_release

- Triggered by:
  - /escrow/<source_account>/<target_account>/<escrow_key>/release/<record_id>
    {
    ratio: <disburse_ratio>
    }
  - e.g. /escrow / **payments|collaborative_ai|0x_A_User** / **0x_Teachable_NLP** / **112233** / release / 98765 / {
    ratio: 1
    }
- Action:
  - Validity check:
    - Check whether the given parameters are valid ones
  - Permission check:
    - The sign address is compared with the admin address of the escrow account
  - Transfer the money in escrow to <target_account> and <source_account> in the given ratio (e.g. ratio 0.2 means 0.2:0.8) using _transfer
  - Record the action result to:
    - /escrow/<source_account>/<target_account>/<escrow_key>/release/<record_id>/result
    - e.g. /escrow / **payments|collaborative_ai|0x_A_User** / **0x_Teachable_NLP** / **112233** / release / 98765 / result

How To Support Multiple Hold Operations To Escrow Account?

This issue was resolved by separating _openEscrow from _hold, i.e., between _openEscrow call and _release call, _hold can be triggered multiple times.

How To Integrate with Payment Service?

Native function _claim of the payment service is triggered by:

- /payments/collaborative_ai/<user_addr>/claims/<record_id>/{
  amount: <claim_amount>,
  target: <target_addr>,
  session_id: <session_id>,
  }
- e.g. /payments/collaborative_ai/0x_A_User/claims/12345/{
  amount: 100,
  target: 0x_Teachable_NLP,
  session_id: 112233,
  }

Then _claim triggers the following escrow functions by writing appropriate to the database:

- _openEscrow (if necessary) which opens an escrow account:
  - /service_accounts/escrow/escrow/<account_key>/{
    admin: {
    <admin_account>: true
    }
    }
    where <account_key> = “<source_account>:<target_account>:<session_id>”.
    Note that <session_id> is used as <escrow_key> in <account_key>.
  - e.g. /service_accounts / escrow / escrow /
    payments|collaborative_ai|0x_A_User|0 : 0x_Teachable_NLP : 112233 / {
    admin: {
    0x_Collaborative_AI: true
    }
    }
- _hold which transfers the money to the escrow account:
  - /service_accounts/escrow/escrow/<account_key>/{
    balance: <claim_amount>
    }
    where <account_key> = “<source_account>:<target_account>:<session_id>”
  - e.g. /service_accounts / escrow / escrow / payments|collaborative_ai|0x_A_User
    : 0x_Teachable_NLP : 112233 / {
    balance: 100
    }

Finally, _release is triggered to release the money in the escrow account.

In short,

- Succeeding native functions are called using function chaining, and
- <session_id> is mapped to <escrow_key> in <account_key>

How To Avoid Direct Transfer?

We need to avoid direct transfer from/to escrow accounts. This can be done by function permission setting, i.e., allowing only the source account, the target account, or the service owner account (using `auth.addr`) to perform money transfers. See AIM-004 and AIM-008 for more information.

Conclusion

A design is provided for escrow service with the following requirements:

- Value hold
- Value release, and
- Permission control

Further Extension

We can consider further extensions in the following directions:

- Agreement between transacting parties e.g.
    - Confirmation by the source account
- Deadline and default behavior
    - If the escrow is not released by the given deadline, the default behavior can be activated by any of the transacting parties

Links

- Escrow (wiki)
- AIP-011 Service Account & Transfer (link)
- AIM-004: Explicit Write Permissions of Native Functions (link)
- AIM-008: Chained Native Function Calls & Write Permissions (link)

Document History

Date	Who	Change	Notes
2021-02-08	seo@	Initial draft	
2021-02-26	seo@, lia@	Review & revise: - Multiple holds - How to avoid direct transfer	

2021-03-02	platfowner, liayoo, shyun-comcom	Review & revise: - _openEscrow - How to integrate with payment service	
2021-03-03	platfowner, liayoo	Internal function calls -> external function calls	
2021-05-07	platfowner	Published	
2021-05-12	platfowner	Github IDs Link to full list	
2021-05-12	platfowner	Republished	