Lab 3: Escrow Services and Applications

Introduction

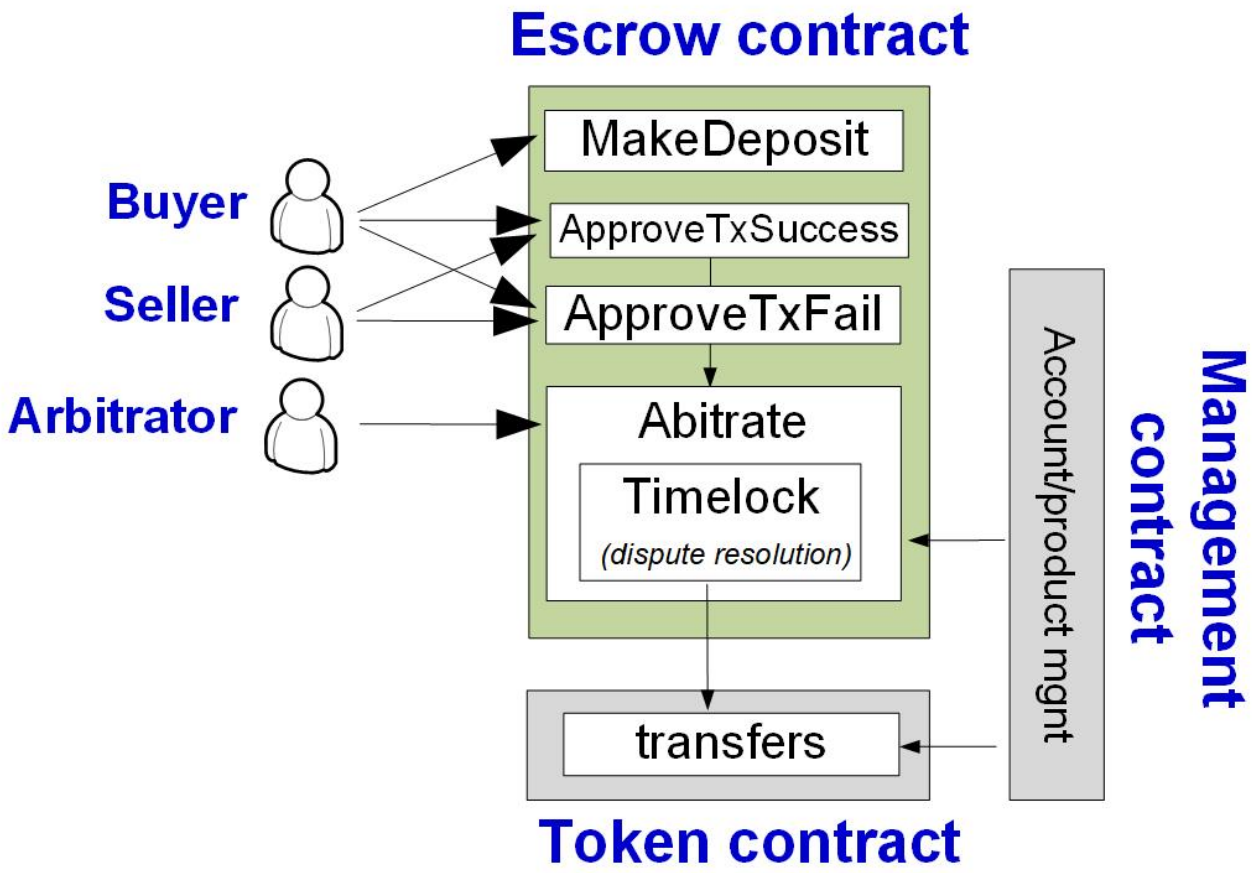
Consider a buyer and a seller, who do not necessarily trust each other, get engaged in a transaction. A problem with a circular dependency is who should make the first move? Should the buyer sends her payment to the seller before the seller sends the product? or the reverse? Traditionally, if the seller is large retailers with significant reputation, like Walmart, the buyer may feel comfortable to make the first move and send the payment, with the implicit trust assumption that Walmart will ship the good. For smaller sellers, buyers typically pay to a trusted third-party, such as eBay or Amazon. If the buyer does not receive the item, this third-party can mediate the dispute and refund the buyer. The functionality of this third-party is escrow.

System design

In this lab, we develop a blockchain-based escrow service (with physical goods). In real life, the escrow model that will be described next is used in the now-ceased Silk Road. The system model consists of four parties: a buyer, a seller, an escrow contract and a mediator (or arbitrator). In the beginning, Bob the Buyer sends the payment (or security deposit) to a contract address called escrow address from which neither Bob or Alice can withdraw unilaterally. After the transaction occurs in the physical world, if both the seller and buyer agree on the transaction state (success or failure), the escrow contract is notified to transfer the payment according (to the seller or to the buyer). In the case of dispute, an off-chain mediator (or arbitrator as in the figure) can investigate what happened and decide which party can withdraw funds from the escrow address. The workflow of escrow service is listed below:

1. The buyer sends a security deposit to the escrow service.
2. (Case 1): The transaction is successful and is agreed upon between the buyer and the seller. Signaled by both parties, the escrow service proceeds to send the deposit to the seller.
3. (Case 2): The transaction fails and the failed state is agreed upon between the buyer and seller. Signaled by both parties, the escrow service proceeds to refund the buyer.
4. (Case 3): There is a dispute about the state of transaction. For instance, the buyer may think the transaction finishes successfully but the seller may think the opposite. In this case, an off-chain trusted party is used to arbitrate the transaction state and will tell the escrow service of her decision. Depending on the result, the escrow service may refund the buyer or pay the seller.

- Note that in all cases, the escrow service should collect certain amount of security deposit for the service fee (e.g., 1% of the security deposit).



A system design to implement the above workflow is to write a smart contract that plays the role of escrow service. Here, your escrow smart contract relies on, in addition to its own contract address, three externally owned addresses (**EOA**): a seller, a buyer and an offline mediator. Each account possesses certain **tokens**. Minimally, the system runs two smart contracts, an escrow contract and a token contract. And the addresses (EOAs) are hard-coded.

In the following, you will design and implement a Ethereum-based escrow service, **iteratively**. First (in Task 1 and 2), you will implement the basic two-contract design described as above. Then (in Task 3), you will be asked to base the escrow service on Ether, instead of escrow-specific token. After that (in Task 4), you will extend the design by supporting account and product managements/registration.

Task 1: Token contract

In this task, you are required to implement a simple token and use it to support the smart escrow contract.

- Implement a simple token contract `SimpleToken` that supports function `transfer(address sender, address recipient, uint256 amount)`

- Use the token to back the accounts of buyer and seller in the smart escrow contract.

Task 2: Escrow contract supporting buyer-seller agreement

Implement the smart escrow contract that supports the following functions:

- `MakeDeposit()` which allows the buyer to send the payment/security deposit to the smart contract. The payment should be the price of product plus service fee (1%).
- `ApproveTxSuccess()` which allows only the buyer or seller to send their approval and to signal the success of transaction. Only both parties approve, the contract sends the payment to the seller.
- `ApproveTxFail()` which allows only the buyer or seller to send their approval and to signal the failure of transaction. Only both parties approve, the contract refunds the payment back to the buyer.

Task 3: Escrow contract supporting dispute resolution

Dispute occurs when the seller sends `ApproveTxFail()` and the buyer sends `ApproveTxSuccess()` (or vice versa). When this happens, the escrow contract enters the following time-lock logic: It will wait for the input of an off-chain mediator via function `Arbitrate()`. If such an input is not received for the 2 minutes, the contract will time-out and refund the deposit to the buyer.

- Implement Function `Timelock()`, such that the smart contract waits for 2 minutes (or 12 Ethereum blocks) for the event of `Arbitrate()` invocation. If the timeout is reached, it refunds. The function `Timelock()` can be called by `ApproveTxSuccess()` / `ApproveTxFail()` after a dispute state occurs and should do two things:
  - Record current time/block height (depends on how you're measuring the time)
  - Change the transaction state to dispute (Transaction is marked as dispute)
- Implement Function `Arbitrate()` which is supposed to be called by account mediator. The mediator decides the transaction state and the function takes action to refund the buyer (upon the transaction failure) and to pay for the seller (upon the transaction success).
  - `Arbitrate()` method should check the difference between time/block heights in addition to the transaction state

Task 4: Revised escrow contract supporting Ether

Revise your escrow contract so that it supports the escrow between Ether owners.

Task 5: Account management

Implement a management contract that supports account/product registration, destroy, etc. Your smart contracts should eventually support multiple buyers/sellers on multiple products.

Bonus Task (30%)

Deploy and run the code of all previous tasks on our on-campus Blockchain. Include screenshots of the results in your report. You can use [this tutorial] as a reference of how to deploy smart contracts on the on-campus Blockchain.

Deliverable

- For all tasks, you should 1) submit your smart-contract code, and 2) show the screenshot of the program execution.

Q/A

- How to implement timeout on smart contract?
  - Hint: You can implement the timeout differently: one design is to rely on an off-chain party to send probe request periodically, which after timeout, triggers the escrow contract to refund/withdraw the deposit.
  - Alternatively, you can make `refund` and `withdraw` as two functions callable by the buyer and seller, respectively. These functions, before refund and withdraw, will need to check if timeout holds.
- How can the "escrow"account send ether to either the seller account, or refund ether back to the buyer account?
  - Hint: You can use functions `address.transfer()` or `address.send()`. Such a function transfers ether from the escrow-contract account to the `address`, which can be either seller or buyer account.
  - Hint: To send ether to the escrow contract, you can send a regular transaction from off-chain.