COMP6451 T1 2022

Assignment 2

Haozhe Chen

z5142012

**Part 1a: Contract structure overview**

This assignment can be divided into two parts of contract:

1. Share contract used ERC20 standard interface
2. 图示

   描述已自动生成Fundraising contract which is used to bid and revel.

**[Round 0] Initial**

In this round, beneficiary (aka NeverPay company), will deploy the fundraising contract (NeverPayFundraising.sol) into the blockchain. In the fundraising contract’s constructor, the share contract (ERC20NeverPayToken.sol) will be initiated.

Before this round end, the fundraising contract should keep 10000 tokens.

**[Round 1] Bid**

In this round, investors are able to call bid function to make a blinded bid. The parameter should be a hash value = keccak256(share, value, nonce).

**[Round 2] Reveal**

In this round, investor will call reveal function to open their bid in round1 and pay to the contract. The parameters are shares, value and nonce.

If hash value of these three parameters is equal to bid record in round1 and amount paid is larger than price \* shares, this reveal is successful. Otherwise, this reveal is failed.

**[After Round 2] Issue**

This round is used to let every investor to get their shares and refunds. Beneficiary can also call this function to get ETH collected during the last round (only successful bid).

The issue function will firstly sort every bid (reveal successfully in round2), then the successful bid’s owner will get the shares, failed bid’s owner will get the refunds.

**Part 1b (stretch): CA Contract structure overview**

CA contract contains two main parts:

1. Off-chain: CA use node.js script to sign investor’s address
2. 图示

   描述已自动生成Fundraising contract uses CA contract to verify signature.

图示

描述已自动生成

**[off-chain script] Signature generator**

CA (ASIC) will use this node.js script to create a key pair, the private key will be used to sign investor’s address by script. After that, signature will be sent to investor and public key will be added into public key set of CA contract (SophisticatedInvestorCertificateAuthorityRegistry.sol).

**[Round1] Signature verification**

When investors bid, their provide not only hash value but also signature (get from CA) to bid function. Bid function will then pass signature and investor’s address (by msg.sender) to **recoverSigner** function of CA contract.

In recoverSigner function, it will call **ecrecover** to get public key corresponding to private key which signed signature before.

CA contract should then use **checkPK** function to check whether this public key is in public key set of CA contract.

*If publicKey in public\_key\_set:*

*Verify pass*

*Else:*

*Verify failed*

**Part 2a: Data model (Token contract)**

|  |  |  |
| --- | --- | --- |
| **Token contract: ERC20NeverPayToken.sol** | | |
| **Property** | **Type** | **Description** |
| name | uint256 | Token’s name |
| decimals | uint8 | How many decimals to show |
| symbol | string | Token’s identifier |
| balances | mapping  address => uint256 | user\_adress => balances  User’s balances |
| totalSupply | uint | Token’s total supply |

The token contract is followed by ERC20 token interface. Every users’ balance is stored by a mapping structure explicitly.

|  |  |  |
| --- | --- | --- |
| **Function: transfer** | | |
| **Parameters** | **Type** | **Description** |
| \_to | address | Receiver’s address |
| \_value | uint256 | Transfer amount |
| **Returntype** | **Description** | |
| bool | Transfer successfully or failed | |

This function is used to let sender to transfer money to certain address.

|  |  |  |
| --- | --- | --- |
| **Function: transferFrom** | | |
| **Parameters** | **Type** | **Description** |
| \_from | address | Sender’s address |
| \_to | address | Receiver’s address |
| \_value | uint256 | Transfer amount |
| **Returntype** | **Description** | |
| bool | Transfer successfully or failed | |

This function is also used to transfer money. However, this function should be called by receiver to initiate a transaction from a certain address. Which also need sender’s to approve this transaction with approve function.

|  |  |  |
| --- | --- | --- |
| **Function: approve** | | |
| **Parameters** | **Type** | **Description** |
| \_spender | address | Receiver’s address |
| \_value | uint256 | Approve transfer amount |
| **Returntype** | **Description** | |
| bool | Approve successfully or failed | |

This function should be called by sender to approve amount of money transferred by certain address. In case of receiver ‘stole’ money from sender (can initiate transfer with any amount from any address).

|  |  |  |
| --- | --- | --- |
| **Function: balanceOf** | | |
| **Parameters** | **Type** | **Description** |
| \_owner | address | Owner’s address |
| **Returntype** | **Description** | |
| uint256 | Owner’s balance | |

This function is used to get the balance of certain address.

|  |  |  |
| --- | --- | --- |
| **Function: allowance** | | |
| **Parameters** | **Type** | **Description** |
| \_owner | address | Sender’s address |
| \_spender | address | Receiver’s address |
| **Returntype** | **Description** | |
| uint256 | Value approved by sender to transfer. | |

This function is used to check the amount of value approved by sender to transferred to receiver.

**Part 2b: Data model (Fundraising contract)**

|  |  |  |
| --- | --- | --- |
| **Token contract: NeverPayFundraising.sol** | | |
| **Property** | **Type** | **Description** |
| token | ERC20NeverPayToken | Token’s contract address. |
| beneficiary | address payable | Beneficiary’s address. |
| bidEnd | uint | timestamp of round1 end:  22/04/2022 |
| revealEnd | uint | timestamp of round2 end:  27/04/2022 |
| bids | mapping(address => mapping(bytes32 => bool)) | addr => (bid\_hash\_value => valid)  Whether this bid is able to reveal. |
| refunds | mapping(address => uint) | addr => refund\_amount  Record refund amount of every investor. |
| validBid | struct:  address addr  uint shares  uint price  uint bid\_order | Structure of a valid bid.  Contain bidder’s address, bid’s share and price, and order of this bid. |
| validBids | validBid[] | List contain every valid bid. |
| bidOrder | mapping(bytes32 => uint) | bid\_hash\_value => order  Mapping to keep order of each bid. |
| order | uint32 | Record global order of bid. |
| issued | mapping(address => bool) | addr => issued  Mapping to record whether this issue has been issued before. |

This contract is used to fundraise the ETH from investors. Several mapping structures are used to keep bid’s information:

**[bids]** is used to record whether this bid is able to reveal, it locate different bids by their investors’ address and hash\_value. The value is true when bid is been created. When investors withdraw this bid or reveal this bid, it will be marked as false.

**[refunds]** is used to record refund value of every investor at round1 & 2.

E.g., bid withdraw, reveal failed (mismatched bid info, insufficient payment)

**[bidOrder]** is used to keep every bid’s order at round1, so that when investors reveal their bid, contract can get bid’s order at round1 by bid’s hash\_value.

**[issued]** is used to check whether certain investor has been issued before, to prevent the investors issuing multiple times to get extra shares and refund.

After investors revealing their bid, contract will use **validBid** structure to store the bids, it contains every detail information of this bid to help contract initiate shares transfer of ETH refund after round2. The **order** property of **validBid** structure will be used to keep bids’ order when sort if there are multiple bids with same price.

|  |  |  |
| --- | --- | --- |
| **Function: bid** | | |
| **Parameters** | **Type** | **Description** |
| \_bindedBid | bytes32 | h(shares, price, nonce) |

Investors will use this function to create a bid in round1. Since this fundraising contract is a blinded bid contract. User should use a keccak256 hash function to encrypt their bids information, so that other investors cannot get bids’ detail. And in case of bids with same shares and price, hash function also contains a different nonce given by different investors.

|  |  |  |
| --- | --- | --- |
| **Function: withdrawBid** | | |
| **Parameters** | **Type** | **Description** |
| \_bindedBid | bytes32 | h(shares, price, nonce) |

Investors will use function to withdraw a bid. It will modify **bids[msg.sender][\_bindedBid]** to false, so that this bid cannot be revealed in round2.

|  |  |  |
| --- | --- | --- |
| **Function: reveal** | | |
| **Parameters** | **Type** | **Description** |
| \_shares | uint | Shares contained in bid in round1. |
| \_price | uint | Price contained in bid in round1. |
| \_nonce | bytes32 | Nonce used to calculate hash with price and shares. |

During the round2, investors will give the shares, price, and nonce to reveal function. If hash value calculated by three parameters can be founded in **bids** mapping (from round1), this reveal will be counted as a valid reveal and recorded by **validBids** list.

|  |  |  |
| --- | --- | --- |
| **Function: issue** | | |
| **Parameters** | **Type** | **Description** |
|  |  |  |

This function should be called by investors or beneficiary after round2. It will firstly sort the validBids list, so that some potentially successful bids will become invalid bids because total shares may greater than 10000. At the end of this function, investor can get their shares or refunds depends on their bids’ status. Beneficiary can call this function to calculate how much ETH collected by fundraising and get ETH transfer from the contract.

**Part 2c: Data model (CA contract)**

|  |  |  |
| --- | --- | --- |
| **Token contract: SophisticatedInvestorCertificateAuthorityRegistry.sol** | | |
| **Property** | **Type** | **Description** |
| publicKeys | Mapping(address => bool) | Mapping to store public key. |
| ASIC | address | ASIC’s address |

This contract should be initiate by ASIC, it can use this contract to manage the public key list. Fundraising contract can use **recoverSigner** function inside CA contract to verify public key of signature.

|  |  |  |
| --- | --- | --- |
| **Function: addPK** | | |
| **Parameters** | **Type** | **Description** |
| pk | address | Public key |

This function is strictly called only by ASIC, it can use this function to add public key into public key set.

|  |  |  |
| --- | --- | --- |
| **Function: removePK** | | |
| **Parameters** | **Type** | **Description** |
| pk | address | Public key |

This function is strictly called only by ASIC, it can use this function to remove public key from public key set.

|  |  |  |
| --- | --- | --- |
| **Function: checkPK** | | |
| **Parameters** | **Type** | **Description** |
| pk | address | Public key |

This function can be used to check whether certain public key is contained in public key list. Fundraising contract can use it during verification.

|  |  |  |
| --- | --- | --- |
| **Function: recoverSigner** | | |
| **Parameters** | **Type** | **Description** |
| sig | btyes | Signature |
| addr | address | Investor’s address |

This function is used to recover public key of signature. Function will use **addr** to restore hash value of investor’s address, so that **ecrecover** can use **hash(investor.address)** to recover investor’s address.

**Part 3a: Off-chain computation (bid encrypt)**

文本

描述已自动生成As discussed in previous part, in round1, investors need to submit a hash value calculated by **keccak256(shares, value, nonce)** to make a blinded bid.

This node.js script will help investors to get the hash value by inputting these three parameters. The script **bidEncryptScript.js** is already contained in submission file.

*Notice：Investors should run* **npm install web3***to install web3 package before running the script.*

文本

描述已自动生成

**Part 3b: Off-chain computation (Signature)**

文本

描述已自动生成This script is used to create key pair and sign the investor’s address.

This script will create a new Ethereum account every time been called. New account’s address will used as public key and its private key will be used as private key.

It uses private key to sign investor’s address which need CA to manually typed in terminal.

**Why sign investor’s address?**

I design this strategy because contract can easily get investor’s address when investor make a bid (by msg.sender). So that when contract do verification, it can use msg.sender to re-calculate the hash value. This can prevent attacker to fabricate other’s address to pass the verification because there is no way to call function by other’s address (unless steal total account include private key).

**Part 4: Requirements discussion**

**Requirement 1: Total shares = 10000**

Token contract (ERC20NeverPayToken.sol) will be initiated with totalSupply = 10000 in fundraising contract’s (NeverPayFundraising.sol) constructor. So that this 10000 tokens will be owned by fundraising contract before round1.

文本

描述已自动生成

**Requirement 2: Minimum price per share is 1 Ether**

Since this contract is design to settle account with Ether, so when investors use node.js script to encrypt their bids, decimal less than 1 will not be acceptable. Also, reveal function make a condition to double check the price of bid .

*Notice: actually uint type in solidity will not even accept decimal number, so no need to worry about price < 1.*

文本

描述已自动生成

文本

描述已自动生成

**Requirement 3: Investors anonymity**

Contract will only store investors’ address but not other information. Contract function can identify different investors by their Ethereum address (through **msg.sender**).

**Requirement 4: Blind auction of two rounds**

Fundraising contract has two main functions which are **bid** and **reveal. bid** is helping investors to make blinded bid and **reveal** is let user to open (reveal) their bids in round1.

**Requirement 5: Deadlines setting**

Deadline of each round will be hard code in contract constructor. Contract used modifier to check whether current time before every function runs

手机屏幕的截图

描述已自动生成

**Requirement 6: Round1 bid information invisible**

Investors will make bid in first round by submit a hash value calculated by shares, price and nonce to **bid** function. Which means nobody else even contract itself shall get no information about each bid. Investors can use node.js script mentioned in Part3 to get their bids’ hash value.

文本

描述已自动生成

**Requirement 7: Multiple bid and bid withdraw**

There is no limitation of bid times in first round. Investors shall make several bids, contract can track bids by bids’ hash value and investors’ address.

手机的屏幕截图

描述已自动生成

Investors can also withdraw their bids by submit the hash value to **withdrawBid** function. Once bid has been withdrawn, it cannot be revealed in the next round.

文本

描述已自动生成

**Requirement 8: Round2 reveal**

In this round, investors should reveal their bids by submit the shares, prices, and nonce to **reveal** function. If the hash value calculated by these three parameters cannot be found in **bids** mapping (used to record bid in round1), this reveal will be treated as failed.

文本

中度可信度描述已自动生成

**Requirement 9: Round2 payment**

Investors should pay exactly full price (shares \* price) of their bids. If the payments are greater than full price or less than full price they should pay, contract will firstly record the refund amount, then investors can call **issue** function to get their refunds back after round2 finished.

文本

描述已自动生成

**Requirement 10: After Round2**

After the round2, investors and beneficiary should call the **issue** function to check whether their bids are successful or fail. With successful bids, function will transfer shares (tokens) to investors. With failed bids, investors will get their refunds.

Function will firstly sort every valid bid (verify when reveal at round2).

文本

描述已自动生成

Then if the function is called by beneficiary, it will calculate the Ethers collected by successful bids within range of first 10000 shares. After that, contract will transfer the Ethers amount to beneficiary’s address

文本

描述已自动生成

If function is called by investors, it will find caller’s bids from sorted valid bids list. If the bids are failed of partial successful, function will transfer refunds Ether back to investors. With successful bids, function will transfer corresponding shares (token) by calling **ERC20NeverPayToken.transfer** function.

文本

描述已自动生成

**Requirement 11: Bid sort with same price**

Contract will use **Insertion sort** to sort the valid bids in **issue** function. By fetching the bid order from **bid\_order** property of **validBid** structure, insertion sort can maintain the order of bids if they have same price.

文本

描述已自动生成

**Requirement 12: Payment refund**

In round2, if bids revealed failed, refund will be temporarily record by a mapping (address -> refund amount).

When investors call **issue** function, if his/her partial successful bid becomes fail due to overflow shares, investors will get refund\_amount = refund\_round1 + refund\_round2 paid back to their address.

文本

描述已自动生成

**Requirement 13: ERC-20 standard**

Since the shares (token) contract is strictly following the ERC20 interface, investors owned the shares can call function of **ERC20NeverPayToken.sol** to do the shares trade-off.

**Requirement 14: Cost effective for NeverPay**

As discussed above, the three main functions: **bid, reveal, issue** are all called by bidders or investors. And shares transfer function and refund pay back function are both contained in **issue** function. So that, every investor will pay for his/her own transaction, NeverPay aka beneficiary will not be charged by transferring Ether or shares to investors.

**Requirement 15: Treat the investors uniformly**

With this contract structure, investors only need to pay gas fee for their personal bids ‘s transaction cost and calculation because each main function is on the foundation of caller’s bids.

However, in issue function, every caller need to pay for sorting the valid bids. This issue may be resolved by using priority queue if there is more time space for further development.

**Part 7a: Token test [TestNPT.js]**

**Test1: Initial test**

This test case is used to test total token supply is correct. Also check accounts[0] which is Token contract creator hold all initial tokens.

**Test2: Approved transaction**

Account0 approve transaction: account0 -> transfer 500 -> account1

Check transaction can be proceed successful if it has been approved before.

**Test3: Unapproved transaction**

Account0 un-approves transaction: account0 -> transfer 500 failed -> account1

Check transaction cannot be proceed successful if it has not been approved before.

**Test4: Transfer successful**

Check transfer can proceed successfully if there is enough balance. Check balance before and after transaction to make sure transaction is successful.

**Test5: Transfer failed**

Check transfer cannot proceed if there is insufficient balance.

**Part 7b: Fundraising test [TestNPFR.js]**

**Test1: Initial test**

This test case is used to test total token supply is correct. Also check accounts[0] which is Token contract creator hold all initial tokens.

**Test2: Bid with Signature test**

This test contains following testcase:

**[Testcase1]:** Investors can bid successfully with valid Signature.

**[Testcase2]:** Investors cannot bid successful with invalid Signature.

**Test3: Bid test**

This test is used to check bid function can make bid to contract successfully. Test also checks bid withdraw function can successfully modify bid’s status from true to false.

**[Testcase1]:** Account 1 make bid successfully

**[Testcase2]:** Account 2 make bid successfully

**[Testcase3]:** Account 1 withdraw bid successfully, bid’s status => false

**[Testcase4]:** Account 2 withdraw bid successfully, bid’s status => false

**Test4: Reveal test**

This test is used to check reveal function in round2. It contains these testcases.

**[Testcase1]:** Account 1 reveal bid successfully

**[Testcase2]:** Account 2 reveal bid failed, because of mismatch nonce

**[Testcase3]:** Account 3 reveal bid failed, because payment is less than amount need to pay (price \* shares).

**Test5: Transfer failed**

This test use ten accounts to simulate different edge cases which may happen after round2. Contains:

**[Testcase1]:** Investor’s bid is valid and get deserved shares transfer

**[Testcase2]:** Investor’s bid is partial valid because this bid makes total share greater than 10000 shares. This investor can get partial shares and partial refund.

**[Testcase3]:** Investor’s paid more than he should pay can get refunds (actually paid – should paid)

**[Testcase4]:** Investor’s bid is invalid because total share is greater than 10000, so he can get full refund.

**[Testcase5]:** Investor’s bid is invalid because he paid less Ether than he should pay for. He can get full refund even his money is not enough.

**[Testcase6]:** Investor make multiple bids, one is valid, one is invalid. He can get right shares transfer and right refund amount.

**[Testcase7]:** Investor cannot call issue function because he has never bid before.

**[Testcase8]:** Investor cannot call issue function multiple times.

**[Testcase9]:** Beneficiary can get Ether collected during fundraising successfully.

**[Testcase10]:** Bids can be sort rightfully, bids with same price will maintain order in round1.

图形用户界面, 文本

描述已自动生成