CMPE 483 Blockchain Programming
Homework 1

Ahmet Melih Aydoğdu
2019400291

Spring 2022

1 Introduction

An autonomous decentralized lottery is implemented. Each lottery round lasts one week and every lottery has two stages. First stage is purchase stage where users can buy tickets by submitting their hashed random number. Second stage is reveal stage where users can reveal their random numbers by providing their random number and commitment key. After every lottery, a new lottery will start automatically.

TL tokens are used as currency unit and ERC20 token contract is used to implement TL tokens. Users can buy lottery tickets by using TL tokens. Lottery tickets are implemented by using ERC721 token contract. Every ticket is treated as NFT and they are transferable to other users.

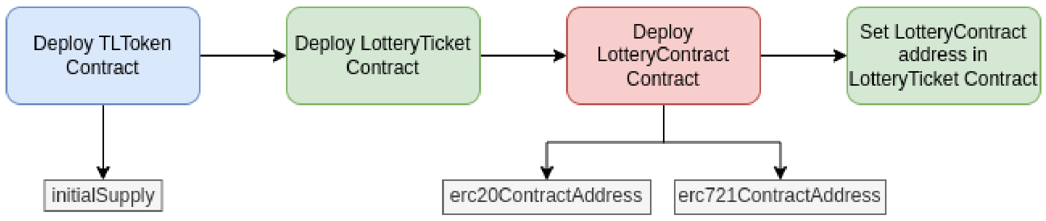
I determined a difficulty constant for the lottery. This constant specifies the possibility of winning the lottery. The default difficulty is 2^{20} which means the probability of winning the lottery is $\frac{1}{2^{20}}$. This is implemented by taking the modulus of every revealed number by difficulty.

Winner tickets are determined by using the random numbers that were provided by users. Each random number's modulus is taken by difficulty and they are XOR'd. In this way, a secure random number is generated by the lottery. Other winning tickets are obtained by taking hash of the winning number.

The tickets can be refunded if the lottery is not ended and the random number is not revealed. Half of the money will be added to user's balance if refunded.

2 Flow

2.1 Deployment



Deploy TLToken Contract
Deploy LotteryTicket Contract
Deploy LotteryContract Contract
Set LotteryContract address in LotteryTicket Contract
initialSupply
erc20ContractAddress
erc721ContractAddress

Figure 1: Deploying Contracts

TLToken and LotteryTicket contracts must be deployed first and *initialSupply* must be passed to the TLToken contract. LotteryContract must be deployed after TLToken and LotteryTicket contract. *erc20ContractAddress* and *erc721ContractAddress* must be passed to the LotteryContract. ERC20 contract is TLToken and ERC721 contract is LotteryTicket. After deploying the contracts, LotteryContract address must be set in LotteryTicket contract so that only LotteryContract can create tickets.

2.2 Stages

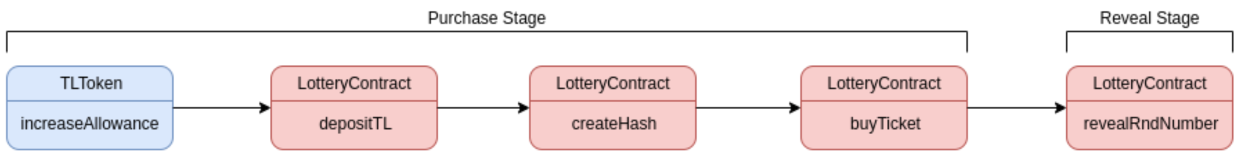


Figure 2: Purchase and Reveal Stages

Users can buy ticket if they have enough balance in LotteryContract. To increase their balance, the allowance must be increased by using *increaseAllowance* function in TLToken. After increasing allowance, users must deposit TL to LotteryContract. If the allowance is enough, LotteryContract transfers TL to itself from user and increase balance.

When buying ticket, every user must provide a random hash to the *buyTicket* function. Users can create this random hash by using *createHash* function in LotteryContract. Users must provide a random number and a commitment key that they determine. The random number must be a positive number or there is no chance of winning. After creating random hash and having enough balance in the account, a user can buy a ticket. The owner of the ticket will be the user that bought it and it is represented as an NFT. It is transferable to other users. Users must also give the random number and commitment key to other user when transferring.

After purchase stage, reveal stage begins and users can reveal their tickets by using their random number and commitment key. The user that reveals the ticket must be the owner of the ticket.

When reveal stage ends, the lottery ends and users can collect their prizes if their tickets won.

2.3 Transfer

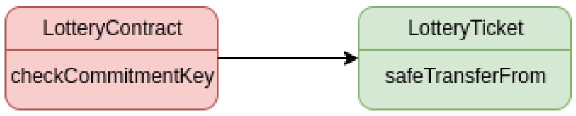


Figure 3: Transferring Ticket

Users can transfer their tickets to other users. Before transferring owner should give the random number and commitment key to the user to which the ticket will be transferred. To confirm those values, *checkCommitmentKey* function can be used. After that, the ownership of the ticket can be transferred by using *safeTransferFrom* function in LotteryTicket contract.

3 Documentation

3.1 Variables

- **balance**: A map that stores balance information for addresses
- **erc20**: TLToken contract
- **erc721**: LotteryTicket contract
- **beginTime**: Unix time in seconds that current lottery's start time
- **createTimeInWeek**: Unix time in weeks that LotteryContract created
- **lotteryNo**: The number of current lottery
- **ticketNo**: The next ticket ID
- **lotteries**: Array that stores all lotteries' information
- **tickets**: Array that stores all tickets' information
- **difficulty**: It determines probability of winning a ticket

3.2 Structs

- **Ticket**
  - **revealHash**: Random hash that user provides
  - **revealedNumber**: Random number that user reveals
  - **revealed**: Whether ticket is revealed or not
  - **redeemed**: Whether ticket is redeemed or not
  - **refunded**: Whether ticket is refunded or not
  - **lotteryno**: The lottery that ticket belongs
- **Lottery**:
  - **winner**: The winning number of lottery
  - **startTime**: Unix time in seconds that lottery starts
  - **firstTicketNo**: The ticket ID of first ticket in the lottery
  - **lastTicketNo**: The ticket ID of last ticket in the lottery
  - **totalMoney**: Total money collected in the lottery

3.3 Modifiers

Modifiers are used to add control to the functions. They are very useful when different functions need the same control statement. Thanks to modifiers, these control checks can be gathered in one place. The name of the modifiers and corresponding error messages explain the purpose of modifiers very well.

3.4 Functions

3.4.1 setDifficulty(uint _difficulty)

The probability of winning can be changed by changing the difficulty of the contract. The default value is 2^{20}. Only the owner who is the account that deploys the contract can change this value.

3.4.2 buyTicket(bytes32 hash_rnd_number)

Users can buy tickets. The lottery should be in purchase stage and user should have enough money in his/her balance in the contract. The price of a ticket 10 TL. Thus, the user's balance will be reduced by 10 and a new ticket will be minted in LotteryTicket. This function will return ticket ID which the number of ticket in the LotteryContract.

3.4.3 depositTL(uint amnt)

Users can transfer TL to LotteryContract. Before using this, users should increase allowance of LotteryContract by using TLToken contract. TL will be transferred to the LotteryContract and the balance of the user will increase.

3.4.4 withdrawTL(uint amnt)

Users can withdraw their money in LotteryContract. The amount they try to withdraw should be higher than their balance. TL will be transferred to the sender and the balance of the user will decrease.

3.4.5 collectTicketRefund(uint ticket_no)

Ticket owner can refund his/her ticket if lottery is not ended. Also, the ticket should not be refunded or revealed before. Half of the money will be added to user's balance.

3.4.6 revealRndNumber(uint ticketno, uint rnd_number, bytes32 commitment_key)

Ticket owner can reveal his/her ticket's random number in reveal stage by using random number and commitment key. Ticket must belong to current lottery. It must not be revealed and refunded previously. The winning number in the lottery is determined by using reveal numbers of tickets. The modulus of random number is taken by difficulty and all random numbers XOR'd to determine the winning number.

3.4.7 getLastOwnedTicketNo(uint lottery_no)

Users can find their last ticket in a lottery. This function will return the ticket ID and the status of the ticket. If the ticket is refunded, the status is 0. If the ticket is redeemed the status is 1. If the ticket is revealed, the status is 2. If neither of those are true, the status is 3.

3.4.8 getIthOwnedTicketNo(uint i, uint lottery_no)

Users can find their ith ticket in a lottery. i must be positive. This function will return the ticket ID and the status of the ticket. If the ticket is refunded, the status is 0. If the ticket is redeemed the status is 1. If the ticket is revealed, the status is 2. If neither of those are true, the status is 3.

3.4.9 checkIfTicketWon(uint ticket_no)

Users can find the amount of prize a ticket won. The lottery must be ended. Also, the ticket must be revealed and it must not be refunded. The modulus of ticket's random number is taken by difficulty so that the number can be in range of the numbers that have probability of winning. There can be i tickets that can won. i is determined by the following equation:

$$i = 1, ..., \lceil log_2(M) \rceil + 1$$

The prize that a ticket won is determined by the following equation:

$$P_i = \lfloor M/2^i \rfloor + (\lfloor M/2^{i-1} \rfloor mod 2)$$

The consecutive winning number is found by hashing the previous winning number, casting to *uint* and taking the modulus of the resulting number by difficulty. This function will return the amount of money the ticket won. A ticket can win more than one prize.

3.4.10 collectTicketPrize(uint ticket_no)

The owner of the ticket can collect the prize. The lottery must be ended. The ticket must be revelead. Also, it must not be refunded or redeemed previously. The amount of prize will be transferred to owner's balance.

3.4.11 getIthWinningTicket(uint i, uint lottery_no)

Users can find the ith winning ticket in a lottery. The lottery must be ended and i must be positive. This function will return the ticket ID and the amount of money that it has won.

3.4.12 getLotteryNo(uint unixtimeinweek)

Users can learn the number of the lottery. Contract creation unix time in week is subtracted from given unix time in week and 1 added. Current unix time in week can be found by dividing unix time in seconds by 604800. The lottery no is found. This function will return this lottery no.

3.4.13 getTotalLotteryMoneyCollected(uint lottery_no)

Users can find the total money that lottery collected.

3.4.14 createHash(uint random_number, bytes32 commitment_key)

Users can create hash with their random number and commitment key. Commitment key can be generated by using following command in linux terminal:

```
$ echo hello | sha256sum
5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03
```

When passing the commitment key **0x** should be added to beginning of the resulting hash value. Example:

- **random_number**: 15
- **commitment_key**: 0x5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03

This function will return the hash value that can be passed to *buyTicket* function.

3.4.15 checkCommitmentKey(uint ticket_no, uint random_number, bytes32 commitment_key)

Ticket owner can check the commitment key and random number. This can be used to confirm these values before transferring. This function returns true if those values correct and returns false if otherwise.

4 Testing

4.1 How to Run

Truffle is used for testing. Node 14 is required to run tests. To run tests, use following commands:

```
$ npm install
$ npm install -g truffle
$ npm install -g ganache
$ ganache -a <account-amount> # run in a separate terminal
$ truffle test
```

These commands should be executed on the homework folder path. The account number in testing is parametric. Thus, by changing the parameter that passed to *ganache* command, the account amount can be increased or decreased.

Before every test, *ganache* should be restarted because time is advanced in the tests.

4.2 Results

- Contracts are deployed and initialized properly.
- Users can increase their allowance by using TLToken and deposit TL to LotteryContract.
- Users can withdraw their money.
- Users can transfer their tickets to other users by using LotteryTicket.
- Users can buy tickets only in purchase stage by using their balance. Tickets are minted in LotteryTicket.
- Users can find their ticket's ID and their status by using LotteryContract.
- Users can reveal their random number only in reveal stage.
- Users can refund their tickets and gets half of the money.

Table 1: Achievements

Task Achievement Table	**Yes**	**Partially**	**No**
I have prepared documentation with at least 6 pages.	x		
I have provided average gas usages for the interface functions.			x
I have provided comments in my code.	x		
I have developed test scripts, performed tests and submitted test scripts as well documented test results.	x		
I have developed smart contract Solidity code and submitted it.	x		
Function depositTL is implemented and works.	x		
Function withdrawTL is implemented and works.	x		
Function buyTicket is implemented and works.	x		
Function collectTicketRefund is implemented and works.	x		
Function revealRndNumber is implemented and works.	x		
Function getLastOwnedTicketNo(uint lottery_no) is implemented and works.	x		
Function getIthOwnedTicketNo is implemented and works.	x		
Function checkIfTicketWon is implemented and works.	x		
Function collectTicketPrize is implemented and works.	x		
Function getIthWinningTicket is implemented and works.	x		
Function getLotteryNo is implemented and works.	x		
Function getTotalLotteryMoneyCollected(uint lottery_no) is implemented and works.	x		
Function depositTL is implemented and works.	x		
Function withdrawTL is implemented and works.	x		
Function buyTicket is implemented and works.	x		
Function collectTicketRefund is implemented and works.	x		
Function revealRndNumber is implemented and works.	x		
Function getLastOwnedTicketNo is implemented and works.	x		
Function getIthOwnedTicketNo is implemented and works.	x		
Function checkIfTicketWon is implemented and works.	x		
I have tested my smart contract with 5 addresses and documented the results of these tests.	x		
I have tested my smart contract with 10 addresses and documented the results of these tests.	x		
I have tested my smart contract with 100 addresses and documented the results of these tests.	x		
I have tested my smart contract with 200 addresses and documented the results of these tests.	x		
I have tested my smart contract with more than 200 addresses and documented the results of these tests.	x		