NEVER Phase 2 (Auctions)

Developed by Pruvendo at 05/06/22

Introduction

The present set of conracts describes EVER/NEVER auction implementation with the full support of D'Auctions. The solution follows the winner of DeFi contest #14.

Overview

The entities described below were implemented throughout the present contest.

Auctions

Auctions are intended to define the fare EVER/USD rate. The initial estimations are done by Oracle handling (implemented during the first stage of NEVER contests (DeFi contest #16) while the goal of the auction is to define the final price, to sell the required amount to the winner as well as to open an exchange center for all the participants allowing then to buy/sell NEVER's with some surplus fee comparing to the Auction price.

The auction is desinged as a Vickrey second price auction.

D'Auctions

While the minimal stake allowed for the auction is supposed to be pretty high (up to millions of dollars) very few people can participate there.

To make the auction more acceptible for a wider audithorium the paradigm of D'Auctions has been implemented. The participants can combine their capabilities and create a joining stake trying to win as a single entity.

Later the winning stake is fairly distributed between the participants (with some extra award for the D'Auction owner) and the lost stakes are returned back (with the possibility to be automatically reinvested).

The important feature is to let D'Action owner to explicitly declare his investment strategy, has no ability to hiddenly change it, thus letting a participant to select a D'Auction that is best in terms of satisfying the participant's requirements and desires.

Location

The implementation provided is located at GitHub. The product is under MIT license.

Architecture

The following key contracts were implemented.

BidLocker

This contract keeps the bid of a participant until the auction is completed. Upon completion the bid either sent to *NeverBank* or returned back to the participant.

BlindAuction

It's a core contract that represents auction itself, below the process of its creation described.

- The creation of *BlindAuction* is initiated by *OracleProxy* mentioned below.
- At the same time, *NeverBank* is deployed that acts as an entrance point for the "winning" coins.
- *registerBank* method of *OracleProxy* is called to keep the bank address. Later this address will be sent to each *BlindAuction* created by the present *OracleProxy*.
- Also *setAuctionCode* method of *OracleProxy* is called as it's needed for auction deployment.
- In the present implementation the *BlindAuction* itself is deployed by *initiateAuctions* method, but during the Phase 3 (final stage) it's planned to improve the robustness. At the same time the bank is called with *updateAuction* to let it be prepared to pay to the winners.

The auctions consists on two parts: *NEVER/EVER* and *EVER_NEVER*. It has as much as three static variables:

- *OracleProxy* address
- *NeverBank* address
- Ordinary number of the auction (monitored by *OracleProxy* that deploys the auctions to keep them unique).

The auction bid looks like a triple: nanoevers, nanoevers and direction, where the first two numbers indicate the amounts the participant plans to pay or receive, where the last one represents the direction (*true* means conversion from nanonevers to nanoevevers and *false* the opposite).

The constructor parameteres define the floor rate (all the lower bids are dropped) as well as minimal exchange amounts (too small deals are forbidden). The rest of parameters describe the duration of *OPEN* and *REVEAL* stages.

The creation of bids is not controlled by the auction. The first interaction of bids with the auction happens at *REVEAL* stage that allows to keep all the bids secret for other participants (all the data is hashed and salted, the exception here is D'Auction as it can not make a hidden bid).

To create a bid (*BidLocker*) it's necessary to calculate the following hashes:

- Auction address
- Owner address (where coins are to be transferred later, important for D'Auctions)
- hashes of the bid itself (a pair of values)

All these hashes are kept as static variables of the *BidLocker* contract.

Upon deployment of *BidLocker* it's necessary to transfer there enough coins and call the *lock* method BEFORE starting the *REVEAL* stage of the auction. During the *REVEAL* stage the *verify* method should be called to check if the bid corresponds to the one created at the time of *lock* method invocation. The same parameteres (that were hashed before deployment and salting) must be used.

The winning bid is the one with the best *myCurrency/buyCurrency* ratio, however, *buyCurrency* can be purchased by the rate of the second-best bid.

Upon the auction completion the auction calls *updateWinners* method of *NeverBank* and informs it about winners and necessary payments. The owner of the winning *BidLocker* can call *receiveWin* method that transfers the required amount to the bank and receives coins in the desired currency back.

Calculator

Calculator is an auxiliary contract that performs hash calculations with salting.

DeAuction

The present implementation of D'Auctions works as follows:

- *DeAuction* contract is created using the owner's public key where the parent *BlindAuction* is kept as a static variable. *DeAuction* as well as a *BlindAuction* works in both directions. It will create two stakes - *NEVER/EVER* and *EVER_NEVER*, certainly, only in case the corresponding *WeightedAggregate* contracts will exist.
- The corresponding *WeightedAggregate* contracts are created keeping the *DeAuction* address as a static variable.
- The owner calls *setAggregators* methods to keep the addresses of aggregators and trust them. Additionally, it calls *setLockerCode* to be able to create bids as well as *setStakeCode* to trust to the *Stake* contracts to be appeared.
- Upon the completion of the auction (in case of any result) the *DeAuction* will be notified by *BidLocker* (described above) using *notifyWin* method and thus receives coins that can be withdrawn by *Stake* owners using *withdraw* method.

NeverBank

NeverBank is an auxiliary contract that is responsible for payments to winners of auctions. Basically, it's an entrance point for the "winning" coins. It keeps an *OracleProxy* address as a static variable.

OracleProxy

This contract acts a bridge between oracles and auctions, creating the laters. The reference to the oracle (that provides a preliminary exchange rate) is kept as a static variable. Also it's needed for a validation of *BlindAuction* updates for the auctions created by the present *OracleProxy*.

Stake

The *Stake* contract works in the following way:

- *Stake* contract is created using the owner's public key where the parent *DeAuction* is kept as a static variable.
- Upon creation, it's necessary to put there a required amount and call *lock* with two numbers - (nano)evers and (nano)nevers as well as with direction flag (*true* means that the coversion from NEVER to EVER is expected).
- Upon the successful lock the contract send all the coins to the *DeAuction* to take them back upon completion using *withdraw* method (*DeAuction* is trustful as the contract address is checked).

Usage

All the scripts use *everdev* utility. To set the required *Solidity* compiler version run:

```
everdev sol set -c 0.59.4
```

The user must have standard *signers* and *givers* to make the scripts runnable. The corresponding instruction can be found here.

- In the file *.project_config* change *TVM_LINKER* to the own linker. The recompile can be done by using:

```
./clean.sh && ./compile.sh
```

- At first run `deploy_proxy_and_bank.sh`. It will deploy contracts and register them for the mutial work. Remember the proxy address from the last output line.

- After this the user can freely deploy the auctions by the script

```
./deploy_auction.sh <address>
```

where the *<address>* is a proxy address received above. Again, remember the address from the last output line.

Now the auction has started.The user can do bids and create D'Auctions.

- Create D'Auction by

```
./deploy_deauction.sh <address>
```

where the *<address>* is taken from the previous bullet point. Remember the address.

- Now the user can create stakes by

```
./deploy_stake.sh <address>
```

where the *<address>* is an D'Auction address. Remember the address.

- To lock the stake and withdraw use:

```
./lock_stake.sh <address>
```

- Just to withdraw:

```
 ./withdraw_stake.sh <address>
```

Also the user can act without D'Auctions:

- To deploy locker with the given hashes and flags. Remember the address:

```
./deploy_locker.sh <auctionAddrHash> <ownerAddrHash> <bidHash> <isNever>
```

- To lock the bid:

```
./lock_locker.sh <address>
```

- To reveal:

```
./reveal_locker.sh <address> <owner> <auction> <nanonevers> <nanoevers> <salt>
```

Note: Addresses should be without 0:

Links

OracleProxy 0:76c91eebfcf6cb8a12bcc1a6c77ada5d9488821716faa91fec512feb3a6e39bd *NeverBank* 0:424e2d4611d45e3daf148a49c5ae6b1ca8358bcaae2df6f2924c9a82efa88585 *DeAuction* 0:4cfab22670b64b71206d5a71031453223b8b002e3caa4538674b665c535d05bf *Stake* 0:e09d6bcaad311d3a5dfa30b21c5c81fb05fbd70cb68dcbbe0cba9f3044e92604 *BidLocker* 0:ae0291e79bd4fb550bb10cd8d6c95f2df1674e02d63993d8bb4a3989352d8cc1

Quality

The present system was both covered by TS4 tests as well as integration testing in a special dev net.

More information

Feel free to ask any question Sergey Egorov, he will either answer or address them to the proper developer.