

# Free TON On-line Auctions

## *Contest submission by OCamlPro*

Steven DE OLIVEIRA & Fabrice LE FESSANT  
Contact: Fabrice LE FESSANT @fabrice\_dune

## 1 Introduction

This document will describe the submission proposed by the OCamlPro team for solving the DevEx contest on on-chain auctions. The goal of this contest was to provide smart-contracts (SC) for deploying and managing different type of auctions (English & Dutch) with multiple variations (Forward, Reversed & Blind). These SCs must be compatible with TON's Crystals, TIP3 & DePools. Also, they must be written in Solidity or C++, the FreeTON SC's languages.

The code of our submission is available on Github:

[https://github.com/OCamlPro/freeton\\_auctions/](https://github.com/OCamlPro/freeton_auctions/)

## 2 Contributions of this submission

Our submission is in two complementary parts, `first-version/` and `second-version/`

Note that a “classical error” is to implement reverse auctions as a simple change in the direction of auctions, i.e. English auctions are moving up, while Reverse English auctions are moving down. Instead, reverse auctions also reverse the position of buyers and sellers. We made this error in the `first-version/` (so we do not claim correctness of the reverse auction implementation in that part), but correctly implemented them in the `second-version/` (but we didn't have enough time to finish that version with debot and testing).

### 2.1 Part `first-version/`

This part contains:

- A set of smart contracts in Solidity implementing various flavors of on-line auctions:
  - An AuctionRoot contract used to easily deploy the different auction flavors
  - An EnglishAuction contract

- A DutchAuction contract
- Bid, ReverseBid and BlindBid contracts to bid in these auctions, either in direct mode, reverse mode, and blind direct mode

All these auctions work on a generic interface for TIP-3 tokens described in the IVault interface.

- Deployment scripts using `ft`, our FreeTON wallet for developers, to test the contracts
- A formal specification of each auction in TLA+ (tla+)
- A high-level specification in an UML fashion (spec/)

with the following limitations:

- Limited checks on accesses to public methods of contracts
- No debot yet
- No deployment yet

## 2.2 Part second-version/

This part complements the first-version/ part with a focus on:

- Using direct TON transfers and standard TIP-3 tokens (we used Broxus TIP-3 token contracts)
- In the “direct” flavor, the winner (buyer) “wins” control over the auction smart contract, allowing him to control the “merchandise” for the auctioneer (seller), usually, another TIP-3 token wallet. No specific method is used to bid, a simple transfer is used to bid both for TONs and TIP-3 tokens.
- In the “reverse” flavor, the winner (seller) receives the tokens (TONs or TIP-3 token) from the auctioneer (buyer), while the “merchandise” (another TIP-3 token) is transferred from his bid contract vault to the auctioneer vault. Note that the bidder can only start bidding after transferring his merchandise to the bid contract vault, so that the merchandise is in an “escrow”.
- Particular attention has been paid on access control

The following flavors have been implemented in this model:

- English auctions (with min-increment percentage)
- Dutch auctions in automatic mode : the price decreases automatically with time
- Dutch auctions in manual mode : the price is decreased manually by the “auction house”
- Reverse English auctions as well as Reverse Automatic Dutch auctions

Limitations:

- No debot yet
- No deployment yet

## 3 Details on the first-version/ part

### 3.1 Auction specification

We divided the auctions into two categories, English, Dutch, which can either be Forward or Reverse and Clear (by default) or Blind. The logic behind each auction is described in the TLA+ files.

English Forward: the auction is initialized with a *starting price*, a *max tick* (time unit) and a *max time* (time unit). The auction always end when *max time* is reached. As long as no bid is done, nothing happens. When a bid is done, the auction ends after *max tick* without any other bid **higher** than the last one (or if *max time* is reached).

English Reverse: same than English Forward, except a new bid must be **lower** than the last one.

Dutch Forward: the auction is initialized with a *starting price*, a *limit price*, a *price delta* and a *time delta* (time unit). The *bidding price* is a function of the time spend since the beginning of the auction, the *price delta* and the *time delta*. After each *time delta*, the bidding price is **decremented** of *price delta* until it reaches *limit price*. If the price reaches *limit price*, the auction ends after *time delta*. If anyone bids a value **higher** than the *bidding price*, he automatically wins the auction.

Dutch Reverse: same than Dutch Reverse, except the *bidding price* is **incremented** and the auction ends when anyone bids a value **lower** than the *bidding price*.

**Blind** auctions work the same way as a **Clear** auctions, except the commitment is hidden until validation of the bidder.

These auction definitions hare the pleasant property that they **eventually ends**, either with or without a winner. Also note that the difference between a Forward & Reverse auction is quite small. The SC implementation takes into account this similarities to limit their size.

## 3.2 Infrastructure

Here is the list of the main contracts (including their interface) with a short description.

*AuctionRoot*: the main contract of the infrastructure ; this contract will deploy the auctions and the bid managers.

*BidBuilder*: the bid manager ; an instance of this contract is built when an auction is created and will be in charge of creating Bid contracts and validating bids.

*Bid*: the bid contract ; when a user wants to bid to an auction, the auction sends a request to its BidBuilder to create a new Bid – this Bid is in charge of checking the status of a generic “Vault” contract. If the status of the “Vault” is correct, then the Bid is valid and tells the BidBuilder that it is correct.

ReverseBid : the bid for reverse auctions

BlindBid : the bid for blind auctions

*VEnglishAuction*: the logic behind any English auction ; this abstract contract encodes the bidding process of an English Auction with a virtual function *newBidsBetterThan*.

*VDutchAuction*: the logic behind any Dutch auction ; this abstract contract encodes the bidding process of an Dutch Auction with a virtual function *betterPriceThanCurrent*.

*Constants* : defines miscellaneous constants, modifiers and events used through the project.

The actual Auction contracts are defined in *EnglishAuction.sol*, *EnglishReverseAuction.sol*, *DutchAuction.sol* and *DutchReverseAuction.sol*. They inherit their respective abstract contract (*VEnglishAuction* or *VDutchAuction*) and only define the virtual function (*newBidsBetterThan* or *betterPriceThanCurrent*).

Few more interfaces are defined. These contracts have no static implementation and must be defined by the Auctioneer.

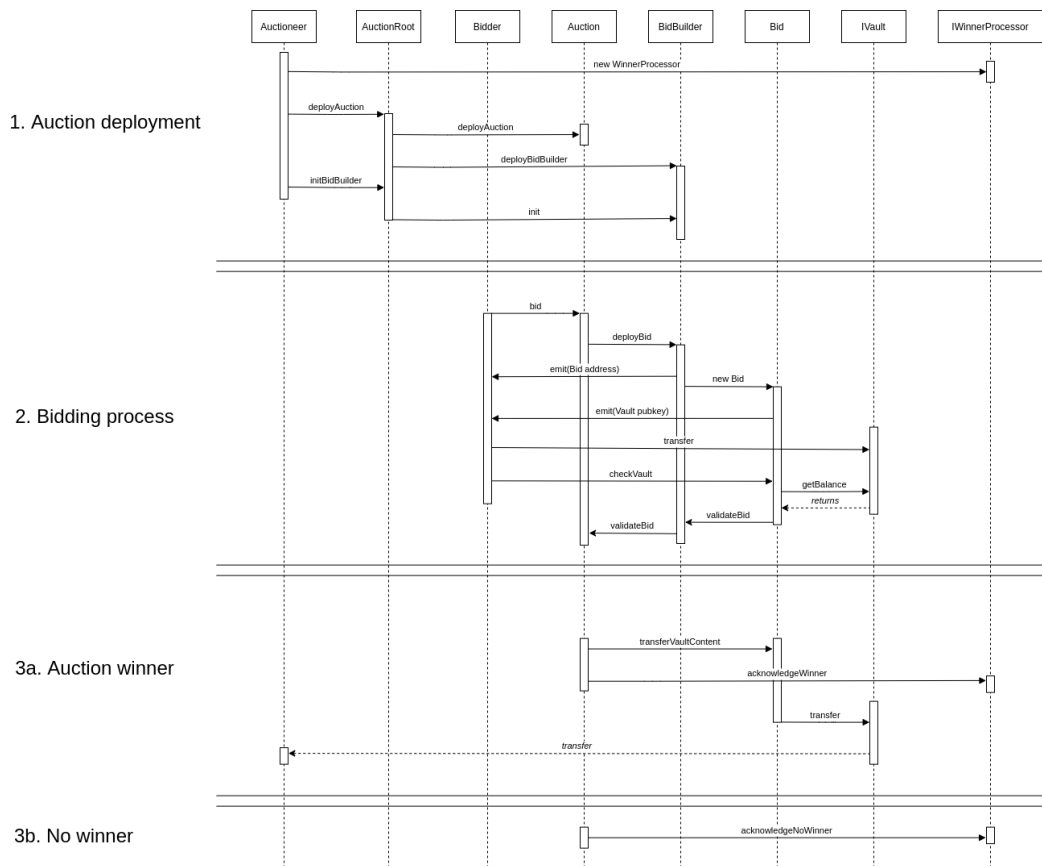
*IVault*: a contract holding the bidding funds. When a bid is done, the user must satisfy the "Vault" constraint (for example, transferring a given amount of crystals on its balance). The funds on the "Vault" are locked and managed by the Bid contract ; this last contract will either send it back to the Bidder if he lost the auction or to the Auctioneer if he won.

*IRootWallet*: the manager of Vaults. They define a function *getWalletAddress* that returns a Vault address (mostly used for TIP3 compatibility)

*IProcessWinner*. a contract that should only be called by an auction. It has two functions: *acknowledgeWinner* and *acknowledgeNoWinner* that are respectively called when there is a winner / there is no winner.

### 3.3 Description of the Auction process

The file *spec/bid\_diagram.pdf* describes the different steps of an auction.



### 3.3.1 Auction deployment

The auctioneer defines a Winner Processor. In Forward auctions, the auctioneer will sell the contract ownership while in Reverse auctions, he simply provides the contract code he wants to take control of. Then, he deploys an auction through the AuctionRoot contract. AuctionRoot deploys 2 contracts : the Auction & its associated BidBuilder. The Auctioneer then must initialize the BidBuilder with the address of the Auction (this step could be avoided by calculating in advance the auction address). The Auction can then be started.

### 3.3.2 Bidding process

Any user can commit a bid by calling the *bid* function of an Auction. The Auction requests the BidBuilder to create a Bid contract. This Bid contracts emits the details of its associated Vault (in Forward, a vault is created for every contract while in Reverse, there is a single vault containing the auctioneer funds). The user then transfers funds to the Vault (Forward) or deploys a winner processor contract (Reverse); when this is done, the bidder asks to check the correctness of the bid. If it is correct with respect to the Bidder's commitment, Bid validates its status to BidBuilder, who checks the origin of the validation. If the validation comes from a correct Bid contract, it transfers the validation to the Auction.

The Auction has now a new bid, the potential old bid transfers back its vault to the previous bidder.

### 3.3.3 Ending an auction

#### 3.3.3.1 Auction winner

The function *acknowledgeWinner* is called with the best bidder information (address, amount, Bid address & Vault address) by the auction. The Vault content is transferred to the Auctioneer (Forward) or the buyer (Reverse).

#### 3.3.3.2 No winner

The function *acknowledgeNoWinner* is called.

## 4 Details on the second-version/ part

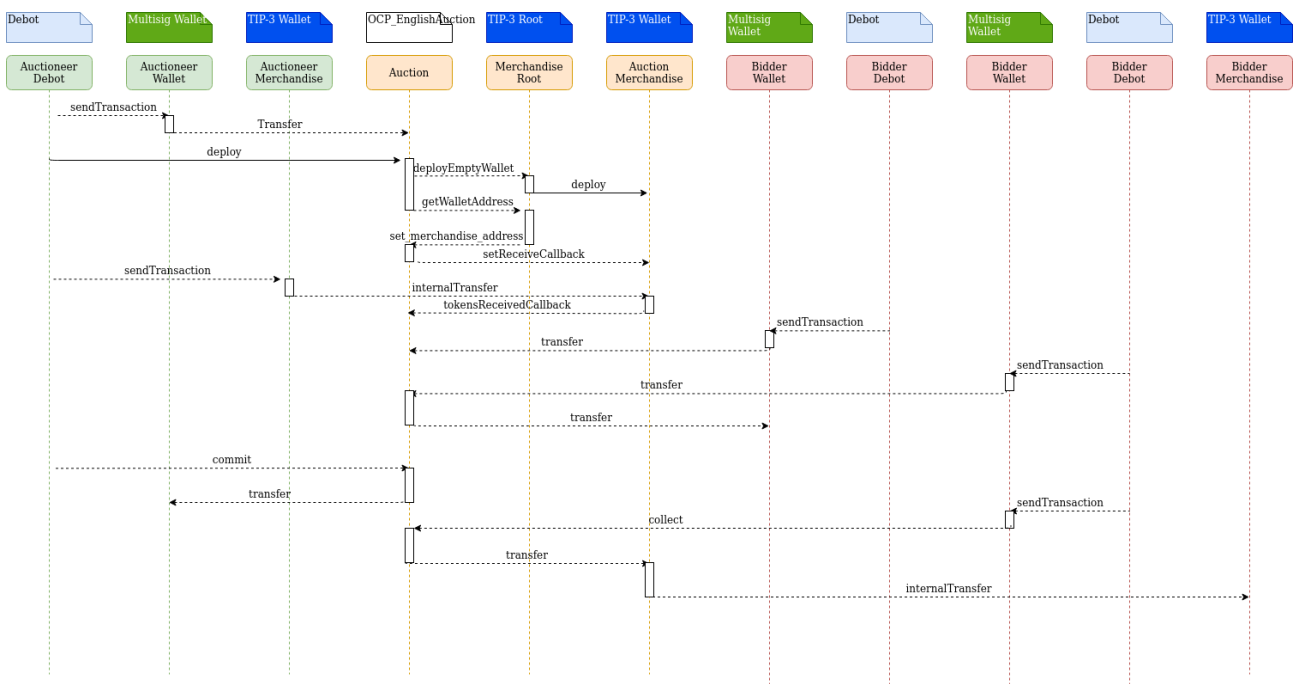
The second-part/ implements on-line auctions with the following properties:

- The merchandise being sold is represented by a quantity of TIP-3 tokens, stored in vaults during the bidding process. In the “direct” flavors, the ownership of the auction contract is transferred to the winner, so that other merchandises may be sold, as long as ownership of the contract grants full access to such merchandises
- The “money” used is either TON crystal, or any TIP-3 token. We used Broxus contracts in our implementation. During “direct” auctions, a bid is done by simply transferring the money to the auction contract (or its wallet for TIP-3 tokens).
- For “reverse” auctions, the auctioneer (buyer) transfers his money to the auction wallet at the beginning of the auction. Bidders create AuctionBidder contracts, a vault (TIP-3 wallet) is attached to every such AuctionBidder where the bidder must store his merchandise before being able to bid. In case of win, the merchandise is transferred to the auctioneer vault, while the auctioneer’s money is transferred to the winner.

### 4.1 Example: The Direct English Auction

The next picture represents all the messages sent during a Direct English Auction done with fiat (TONs):

- In the first part, the auctioneer deploys the auction contract (we plan to simplify this process by either providing a root contract for auctions, or simply using a debot).
- In the picture, the contract would create a vault where the auctioneer would transfer the merchandise, however, in the current implementation, we didn't implement that step, and just transfer ownership of the auction contract to the winner. Thus, the contract implements a sendTransaction function, à la Multisig, so that the winner can forge and send any message originating from that contract, for example to ask for a transfer of the merchandise. We think this approach is more generic, but will still implement the vault for merchandise also.
- In the next step, a first bidder transfers TONs to the auction from his multisig. The bid is accepted (if greater than the minimal price set by the auctioneer)
- In the second step, another bidder transfers TONs to the auction. Since the bid is bigger (if wanted, the contract can check for a minimal bid increment, like 10%), it is accepted, and the former bidder is immediately refunded
- Finally, when the auction is finished (the auctioneer can set a maximal time, or a delay after the last bid), a commit() message can be send by the owner to transfer the bid money to his account; The winner is now owning the contract and can use it to send messages to other contracts. In the picture, we propose to implement a collect() message in the specific case of a TIP-3 merchandise that has been stored in the auction vault.



Note that, although the picture is for an English Auction, the difference with a Dutch Auction is minimal: in the case of a Dutch auction, the first bid is the winning bid if it is greater than the current price, so there would be only one bidder in the scenario, and no need to refund a previous bidder or wait for the end of the auction.

In the second picture, we present the same Direct English Auction, but using a TIP-3 token as money. The interactions are a little more complex, because every entity must own a TIP-3 wallet and use it to transfer/receive the bids.





## 4.2 Example: The Reverse English Auction

The following pictures depicts the exchange of messages in the Reverse English Auction using TONs as money:

The following actions happen at the beginning:

- The auctioneer deploys the `OCP_ReverseEnglishAuction` contract, providing the address of his merchandise vault (TIP-3)
- The auction contract recovers the root wallet of the merchandise vault
- The auctioneer transfers his money to the auction contract (the starting price of the auction)
- The auction is now ready

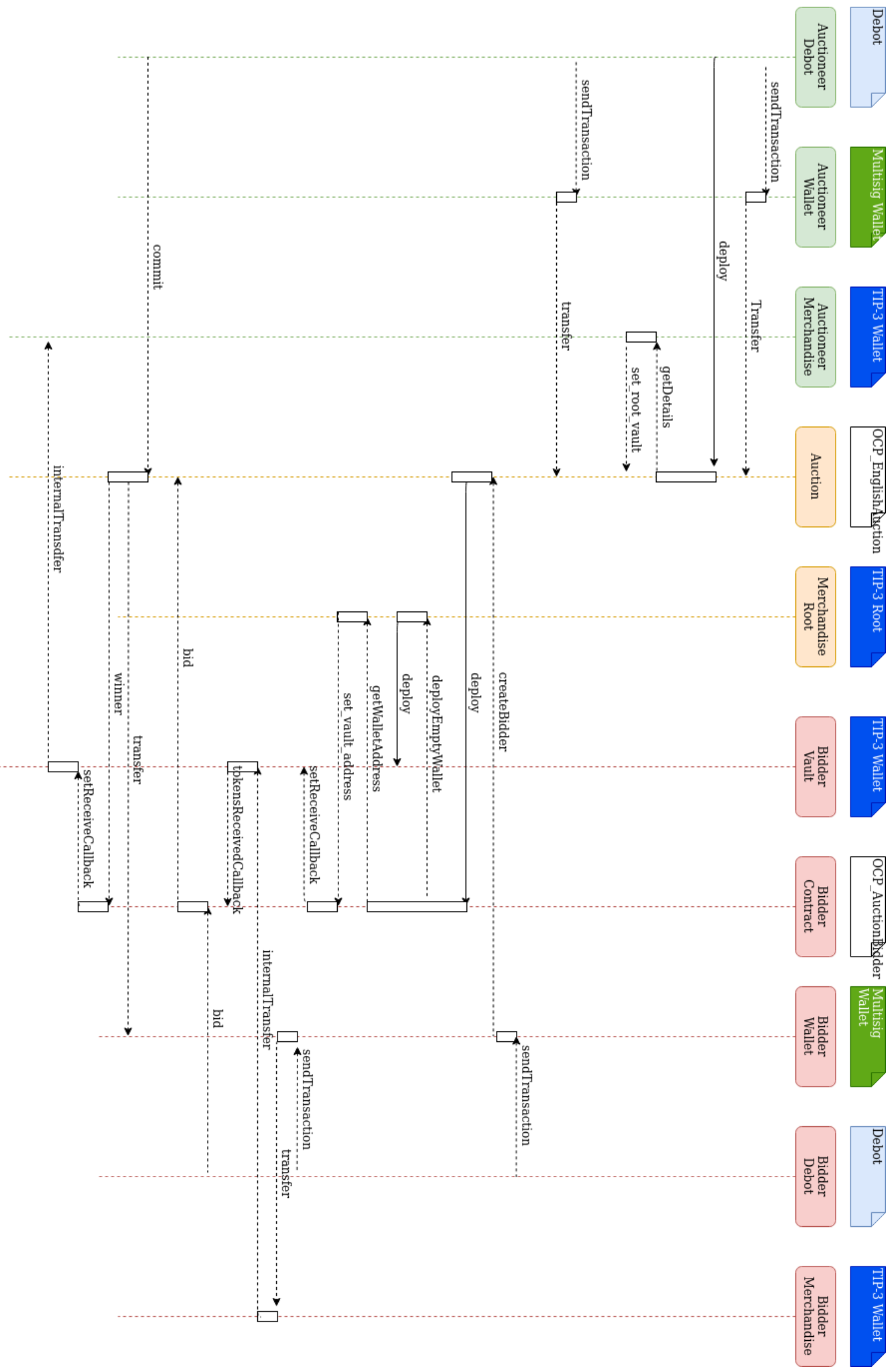
When a bidder wants to participate:

- The bidder calls the `createBidder` method of the auction contract. The auction contract deploys a vault to serve as an escrow for the merchandise to be sold.
- The bidder transfers his merchandise to the bidder contract vault. The bidder contract receives the notification
- The bidder contract is now ready
- The bidder calls the `bid()` method, and his bid is sent to the auction contract. At the same time, the bidder contract “locks” the vault, until either an unbid is received from the auction (if another bidder sends a better bid), or the bidder wins the bid. Note that, if another bidder sends a better bid, the vault is unlocked and the bidder can decide to recover his merchandise (using the `empty_vault` message), or decide to bid again.

At the end of the auction (when either the limit time is reached, or a particular delay after the last bid without further bids):

- The owner sends the `commit` message. The winner is notified, and his bidder contract immediately sends the merchandise to the auctioneer vault. In parallel, the money is sent from the auction contract to the bidder multisig.

Note that it is quite easy to generalize from fiat (TONs) to TIP-3 token as money, exactly in the same way as done for the Direct English Auction.



## 4.3 Blind Auctions

Our implementation of blind auctions satisfies the following properties:

- Another user than the bidder cannot guess :
  - That the bidder is going to bid
  - The amount that the bidder is going to bid
- The bidder must choose the bidding amount when creating the bid contract
- The bidder must at least send the auction start price as money (TONs or TIP-3 tokens) to the bid contract BEFORE the end of the blind period
- The bidder can transfer additional money to the bid contract after the end of the blind period to match the bidding amount
- If the bidder does not reveal the bid contract by calling the 'bid' method after the blind period and before the end of the auction, all the money stored in the contract is lost, i.e. at least the auction start price.

The process of a blind auction is the following:

- The auctioneer creates an auction contract (English or Dutch), setting up to dates: the ending date of the *blind period*, and the ending date of the *revelation period*. It also defines a *starting price*, all bids must be above this price
- During the blind period, nothing happens for the auctioneer, as bidders actions are not visible
- During the revelation period, bids are received from OCP\_BlindAuctionBid contracts. The auction contract verifies that the address of the bid contract is indeed an OCP\_BlindAuctionBid, and that the blind date in the bid contract is before the end of the blind period (to verify that the money was sent before) and that the starting price of the bid contract is above the auction starting price (to verify that the bidder committed enough money on time). The verification of the address of the bid contract is also used to verify that the bid parameters did not change since the creation of the bid: the static hash stored in the contract contains the amount of the bid, the auction address, the bidder information (address and pubkey), and a nonce. The nonce is a random number that prevents other users from recovering these parameters from the hash stored in the contract (otherwise, they would be able to know the amount that the bidder is going to bid).
- If a winning bid is found at the end of the revelation period, a winner() message is sent to the bid contract, that should transfer the money to the auctioneer's wallet. For non-winning bids, an unbid() message is sent to the bid contract, that should transfer the money back to the bidder's wallet.

For bidders, the actions are:

- The bidder notices the start of the blind auction, with a blind period, a revelation period and a starting price.
- The bidder chooses the amount he wants to bid, and generates a random nonce to hide the amount in his bid contract
- The bidder creates an OCP\_BlindAuctionBid with:
  - a hash computed from the amount of the bid, the nonce, the auction address and his own information (address and pubkey)
  - a blind period date that should be BEFORE the one of the auction
  - a start price that should be ABOVE the one of the auction
- The bidder sends at least start price money to the bid contract (or its wallet for TIP-3 tokens) before the end of the blind period. The bidder can still send more money to the bid contract after the end of the blind period, as long as he had

already sent money equivalent to the starting price BEFORE the end of the blind period.

- When the revelation period of the auction starts, the bidder sends a bid() message to the bid contract with the actual amount he wanted to bid, and the auction address. Before sending the bid() message to the auction, the contract verifies that (1) the hash of the contract matches the amount and the auction address, (2) the money stored in the bid contract (or associated wallet) matches the amount sent.
- When receiving a winner() message, it transfers the amount to the auctioneer's wallet (and the remaining to the bidder's wallet)
- When receiving an unbid() message, it transfers all the balance to the bidder's wallet
- There is no way to recover the money if no bid() message was sent

We implemented the OCP\_BlindAuctionBid contract, but we didn't have time to implement the auction counterpart yet.

## 4.4 Auction Smart contracts

The following pictures describes the inheritance relationship between the different smart contracts that we implemented:

