Distributed Anonymous Sealed Bid Auction

Nafisa Shazia, Denis Trailin, Matheus Stolet, John Jang
d3w9a, b3c0b, b3l0b, k0b9
Repository

Introduction

For this project, we implemented a distributed anonymous auction, following the protocol described in the paper titled *Multi-round Anonymous Auction Protocols* by Kikuchi, Harkavy, and Tygar [KHT98].

Traditionally, an auction is an arrangement among three parties: a seller, a single auctioneer, and some number of bidders. The seller approaches the auctioneer with the item they want to sell. The auctioneer agrees to hold an auction to identify the highest price the item can be sold at. Once the auction begins, interested bidders submit their bids to the auctioneer. Bidders may not withdraw bids once made. After some iterations, the auction ends with the auctioneer resolving the winner, i.e the single highest bidder. The winner is publicly announced and the sale is made.

Although there is an abundance of different auction models, they can be broadly divided into two categories: public bid and sealed bid auctions. In a public bid auction, all bids made are known by all parties, including other bidders. Examples include English and Japanese-style auctions. A sealed bid auction, by contrast, hides bids among bidders but does not hide bids from the auctioneer. This works similarly to voting in that a bidder submits their bid in a sealed envelope to the auctioneer, who unseals all the envelopes and searches for the highest offer.

We wanted to take the concept of a sealed bid auction further by enabling multiple distributed auctioneers and providing much stronger secrecy: an auctioneer should not be able to unseal bids. Only the seller should be able to unseal *potentially winning* bids. This implies that auctioneers, who are effectively blind, must be able to coordinate with each other to deduce potentially winning bids and propose them to the seller.

We will achieve this by enforcing a maximum and minimum price, or more specifically, by having a discrete range of acceptable bids. The blind auctioneers will first attempt to aggregate the IDs of bidders at the maximum price[1]. This aggregate can be thought of as the "sum of IDs of bidders for this price". This will be forwarded to the seller who will be able to decompose the aggregate into unsealed individual bidder IDs. From there, the seller will ascertain one of three possible outcomes:

[1] If a bidder is not interested in that price, their ID will be 0 (but the auctioneers will not be able to tell). All of this is described mathematically in the section *Secret Sharing Protocol*

1) No bidders are interested in this price (sum of IDs is 0) -- obtain the aggregate interested bidders for the next lower price. If there is no next lower price, auction ends inconclusively.
2) A single bidder is interested in this price -- winner identified, contacted by seller, auction ends, losing bidders infer loss.
3) Multiple bidders are interested in this price -- trigger next round of auction to break ties.

Provided there were no ties, the auction must also end with the winner's identity kept secret. These two guarantees combine to protect the privacy of all bidders and is the basis of applying the term "anonymous" to this auction.

As mentioned, we will do this in a distributed setting with multiple auctioneers, the majority of whom we will assume are honest.

Guarantees Given

1. Bids remain secret to auctioneers and other bidders at all times
2. Bidders cannot learn who the other bidders are
3. Only seller and winner learn identity of winner
4. A passive attack orchestrated by a minority of auctioneers will not succeed

Regarding the fourth point, a passive attack is one where a group of auctioneers collude to obtain information needed to unseal bids. More detail is given in the section *Threat Model.*

Design

Roles

Seller: The user who wishes to sell an item and publicizes information about the auction to be held. Only a single seller is supported. The information publicized must include:

- ❏ Item to be sold
- ❏ Discrete price range of acceptable bids in some currency
- ❏ List of IPs for auctioneer servers it approves
- ❏ The value t representing the maximum number of (suspected) colluding auctioneers, $t < N/2$ where N is the number of auctioneers in the approved list
- ❏ Auction round number
- ❏ Start time of the auction round in Unix epoch time
- ❏ Interval between phases in an auction round
- ❏ Public key

Bidder: Users who bid on the auction item at prices set by the seller. There is no upper bound on the number of bidders who can participate during the early phases of an auction round. TODO.

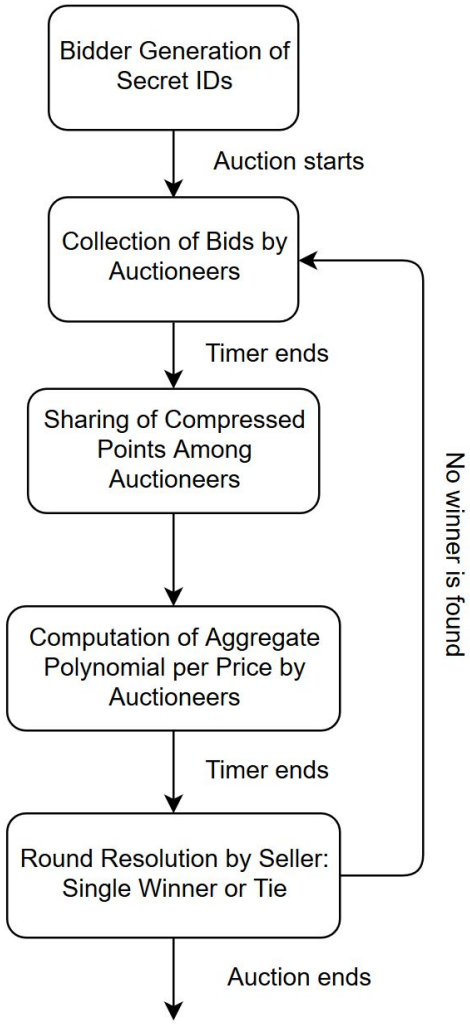
Auctioneer: The server that collects bids from interested bidders, does computations, and makes the result of its computations available to the seller towards the end of the round for resolution.

Phases of Auction

An auction round is divided into several phases. A timer marks the end of some phases, where the precise time interval is specified originally by the seller.

<u>1. Bidder Generation of Secret IDs</u>

Prior to the start of the auction, bidders retrieve the auction round information publicized by the seller. Each bidder decides the maximum bid they are willing to make for the item and generates secret IDs for each price in the seller's price range that is below or equal to their maximum bid. The secret ID for a price is `<<bidderIP:port price>>` encrypted with the seller's public key; the bidder client will choose a port it can listen to for seller notifications until the round is over.



Each bidder then generates random polynomials for each price in the seller's price range such that for polynomial *f*, *f(0)* = secret ID if the bidder is willing to bid that price, *f(0)* = 0 otherwise. For each auctioneer *i* in the seller's list (the auctioneers are indexed from 1), a bidder samples the points (i, f(i)) from each polynomial. Bidders prepare to send custom maps of `{price: point}` to the auctioneers.

<u>2. Collection of Bids by Auctioneers</u>

Once the auction starts, auctioneers begin the first timer and accept points from bidders, storing an internal mapping of `{price: []points}` for each price in the seller's range and the sampled points amassed from all bidders for that price. Auctioneer *i* should reject all points (x,y) where x != *i*.

<u>3. Sharing of Compressed Points Among Auctioneers</u>

After the first timer ends, the second timer begins. Auctioneers use the points they received from the bidders to compute compressed points, summing up every *y* value. Their maps will be updated to something resembling `{price: (x, sumY([]points))}`. Each auctioneer then requests others' compressed points by querying the corresponding REST endpoint defined on every auctioneer server for the purpose of sharing this information.

<u>4. Computation of Aggregate Polynomial per Price by Auctioneers</u>

After an auctioneer receives compressed points from other auctioneers, it will compute Lagrange interpolating polynomials for each price using all possible *t*+1 size subsets of these points. The auctioneer will choose the polynomial the majority of these subsets produce for each price and then extract the value of the polynomial at *f(0)*. It will make these values available

to the seller by querying.

5. Round Resolution by Seller: Single Winner or Tie

When the second timer ends, the seller queries the auctioneers for their encrypted "sum of IDs" for the highest price. It selects the value the majority of auctioneers agree on. The seller will use its private key to try and decrypt the IDs. If it decrypts a zero, it means there was no bid for that price, so then the seller repeats this process for the next highest price. This descending search continues until either there is no lower price (auction ends inconclusively) or one of the following two cases occur:

Case 1: Seller gets an error when trying to decrypt the value for a price. This means that there are multiple winning bids at that price. The seller will calculate prices for a tie-breaking round and make the information for the new round available in its REST server. Bidders are expected to poll the REST server to learn the prices for the tie-breaking round. A new auction round starts with the new prices for the same item.

Case 2: Seller is able to decrypt the IDs without error and the result is non-zero. This means that there is a single winning bid. The seller uses the IP and port contained in the ID to notify the winning bidder that they won the auction, then sets the current auction round to -1 (signifying that the auction is over). The losing bidders will be able to infer loss since they are polling the seller's REST server and will see the change to -1.

Tie Breaking Round

The seller has to calculate the new prices for the tie-breaking round. Suppose oldK is the number of possible prices that the seller advertises and oldS is the stride between the prices. For example, if oldK=5, oldS=100, and the initial price is 100, the seller advertises [100, 200, 300, 400, 500] as the prices. If there is more than one bid for 500, the seller sets up a new round with newK=5, newS=100 and the initial price as 500. The seller will then advertise [500, 600, 700, 800, 900] as the prices for the tie-breaking round.

If the highest bids are not for the highest possible price, the seller sets up a tie-breaking round where newK=oldK, newS=Ceiling(oldS / oldK) and the initial price is the price of the highest bid. For example, if the two highest bids are for price 300, the seller will set up a tie breaking round with prices [300, 320, 340, 360, 380].

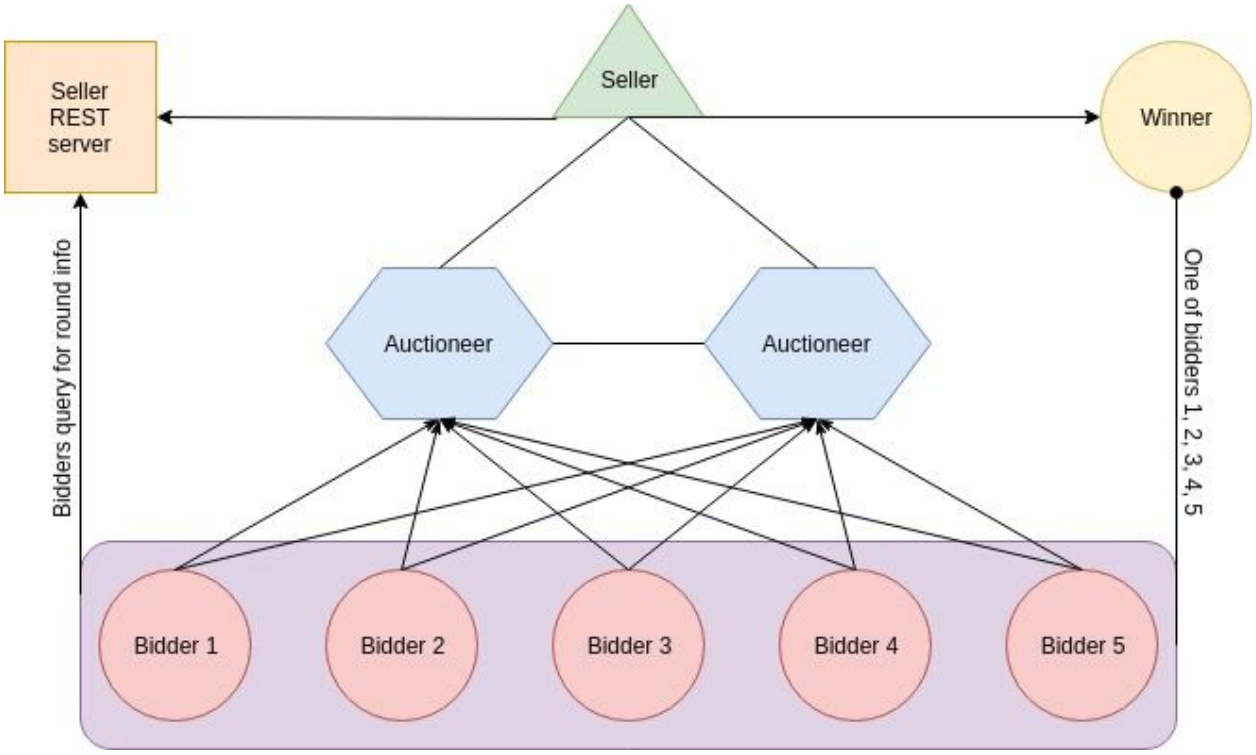
Joins

A bidder can join and participate in an auction round that is in phases 1 or 2. A bidder who attempts to join in later phases will be blocked until the auction round is over and presented with the opportunity to participate in the next round.

Auctioneers can never join an auction if they are not in the seller's approved list. Approved auctioneers may fail and rejoin, however.

Network Topology

- The seller is connected to every auctioneer and additionally needs to notify the winner at the end of the auction
- Every auctioneer is connected to every other auctioneer
- Every bidder is connected to every auctioneer and additionally needs to access the publicized auction information hosted by the seller (via the seller's REST API).



Assumptions and Out of Scope Considerations

- The list of approved auctioneers is predetermined by the seller before the auction. The auctioneer selection procedure is out of scope.
- Network partitions are not tolerated. Identifying when such partitions occur and handling them is out of scope.
- We assume that clocks are synchronized between all actors in the system.
- We assume that the majority of auctioneers are honest.
- We assume that if a bidder cannot reach an auctioneer, the auctioneer is down.
- If $>t$ auctioneers are down, we cancel the auction. Working around the failure of a majority is out of scope.

Threat Model

Auctioneers who attempt to undermine the anonymity of bids are the threat in this system. A colluding network of auctioneers may seek to pool their resources together (individual bids intended exclusively for them, computational power) to unseal bids and discreetly act on the information.

Even if a group of auctioneers is only able to identify the price attracting most bids, they can subvert the auction by having accomplice bidders continue to make bids for that price, forcing unnecessary tie-breaking rounds until real bidders drop out of the auction.

As mentioned earlier, we assume that the majority of auctioneers are honest. Therefore the threat we will guard against is a minority colluding network dealing a passive attack. We do not address active attacks, where auctioneers actively lie about the bids they receive to other auctioneers and the seller.

Secret-Sharing Protocol
We are using a cryptographic scheme called *Shamir's Secret Sharing* [Sh79] to guard against passive attacks by a minority colluding auctioneer network. At a high level, this technique prevents a secret from falling into the wrong hands by breaking the secret up into pieces and distributing unique parts to unique participants. To reconstruct the original secret, a certain threshold of participants must pool their pieces together. A group that doesn't meet the threshold size will not be able to reconstruct the secret.

In our auction, we want to prevent a bidder's secret IDs from being captured by the minority colluding network, as possession of these secrets will allow the auctioneers to open the bids (provided they have the computational power to crack RSA).

We make bidders generate and distribute unique points from secret random polynomials of degree t, the maximum number of colluding auctioneers defined by the seller (a value that cannot exceed half of the total number of auctioneers) because such polynomials can only be reconstructed by t+1 points supplied by auctioneers collaborating together. A minority colluding network with t points will be incapable of computing the bidder's secret ID.

Failures and Recovery
Seller Failure: Our auction does not support seller failure. If the seller is down, the auction will end inconclusively.
Bidder Failure: A bidder who fails before sending bids is never considered a participant. A bidder who fails after having sent bids and remains down past Round Resolution will neither win nor lose the auction. If they come back alive before the round is over, they will be informed of a win or loss.
Auctioneer Failure: Our system is able to withstand failures of N/2 - 1 auctioneers. As long as we have a majority of auctioneers running, the auction is able to proceed and guarantee the privacy of the bidders. This is made possible by the Shamir secret sharing scheme, which only requires the seller to have a threshold of points (the threshold here is t+1 and $t < N/2$). If an auctioneer goes down, it is not able to rejoin in the same round but will be able to rejoin the auction in subsequent tie-breaking rounds.

Work in Progress

What Currently Works

- Bidder generates secret IDs and random polynomials, samples points from polynomials and sends it to all auctioneers
- Auctioneers process the bids they receive, compress the points, share compressed points with other auctioneers and perform Lagrange interpolation on various subsets of the compressed points for each price. Auctioneers make the results of their computations available to the seller via REST endpoints.
- Seller starts an auction, waits for the round to reach the Round Resolution phase, queries the auctioneers for their results, determines whether there is a single winner, tied winner, or no bids. In event of single winner, notifies winner and updates round information on REST server. In event of tied winners, announces tie-breaking round. In event of no bids, ends auction inconclusively.

What Still Needs to be Done

- Handle the failure of all actors appropriately
- Make bidder block sending of bids until auction has officially started
- Deploy and test our system on Azure
- Better error messages and filter console output so that there is less noise

Evaluation of Prototype

Major Changes from Proposal

In our proposal we indicated that only tied winners should be able to participate in subsequent tie-breaking rounds. This seemed reasonable because we wanted to iteratively reduce the number of competing bidders until there was a single winner. However, we realized that there was no way for us to identify the tied winners and add them to a whitelist since the seller will fail to decrypt the sum of IDs for a price if there are multiple interested bidders for that price. Thus our prototype lets any bidder -- old or new, tied winner or loser -- participate in tiebreaking rounds. This is not optimal because auctions can last much longer than desirable due to ineffective tie-breaking.

External Libraries Used

The following are all the non-standard Go libraries we used:

github.com/jongukim/polynomial	To generate random polynomials, sample points from those polynomials, and compute the Lagrange interpolating polynomial for some given points.
github.com/phayes/freeport	To identify an available port for the bidder to use.
github.com/gorilla/mux	To route HTTP requests to the corresponding handlers.

Approximate Demo Script

Demo Normal Operation (Auction with 1 round):

1. Start auction with 7 auctioneers and 1 seller on Azure.
2. Have 3 bidders running on our computers send bids to the auctioneers for different prices.
3. Wait for the auction to end and show that the winning bidder is successfully identified and notified by the seller.

Demo Normal Operation (Auction with tie-breaking round):

1. Start auction with 7 auctioneers and 1 seller on Azure.
2. Have three bidders running on our computers send bids to the auctioneers. Show that two bidders sent a bid for the highest price.
3. Wait for the auction round to end and show that the seller detected there were multiple bids for the highest price.
4. Show that the bidders are notified of a tie-breaking round with new prices.
5. Have the three bidders submit bids for different prices
6. Show that the winning bidder is notified by the seller.

Demo Node Failure:

1. Start auction with 7 auctioneers and 1 seller on Azure
2. Have two bidders running on our computers send bids to the auctioneers.
3. Kill three auctioneers
4. Have a third bidder send a bid for a price that will tie him for the highest bid.
5. Show that a tie-breaking round is properly started even though three of the auctioneers failed.
6. Send bids for the tie-breaking round and show that the winning bidder is contacted, despite node failures.

Demo Bidders Joining

1. Start auction with 7 auctioneers and 1 seller on Azure.
2. Have three bidders running on our computers send bids to the auctioneers. Make sure that there is a tie for the highest bid.
3. Seller and auctioneers will start a tie-breaking round.
4. Have the three bidders from step 3 and three new bidders join and send bids for the new round (have one of the newly joined bidders send the highest bid).
5. Show that the winner, one of the newer participants, is notified.

References

[Sh79] A. Shamir, How to share a secret, CACM, 22, 1979, pp.612-613

[KHT98] H. Kikuchi, Michael Harkavy, and J.D. Tygar, Multi-round Anonymous Auction, In Proceedings of the First IEEE Workshop on Dependable and Real-Time E-Commerce Systems, pp. 62-69, June 1998