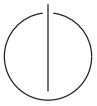
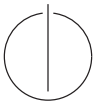



Technical University of Munich
Department of Informatics

Master's Thesis in Informatics

Implementing Privacy Preserving Auction Protocols

Markus Teich



Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

Implementing Privacy Preserving Auction Protocols

Implementierung von privatsphäreerhaltenden Auktionsprotokollen

Author	Markus Teich
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisors	Sree Harsha Totakura, M. Sc.
	Dr. Christian Grothoff
	Prof. Dr. Felix Brandt
Date	February 15, 2017

Informatik VIII
Chair of Network Architectures and Services

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Garching b. München, February 15, 2017

Signature

Abstract

In this thesis we translate Brandt's privacy preserving sealed-bid online auction protocol from RSA to elliptic curve arithmetic and analyze the theoretical and practical benefits. With Brandt's protocol, the auction outcome is completely resolved by the bidders and the seller without the need for a trusted third party. Loosing bids are not revealed to anyone. We present `libbrandt`, our implementation of four algorithms with different outcome and pricing properties, and describe how they can be incorporated in a real-world online auction system. Our performance measurements show a reduction of computation time and prospective bandwidth cost of over 90% compared to an implementation of the RSA version of the same algorithms. We also evaluate how `libbrandt` scales in different dimensions and conclude that the system we have presented is promising with respect to an adoption in the real world.

Zusammenfassung

In dieser Arbeit übersetzen wir Brandt's datenschutzfreundliches Online-Auktions-Protokoll mit verschlossenen Geboten von RSA zu Elliptische-Kurven Arithmetik und analysieren die theoretischen und praktischen Vorteile. Bei Brandt wird das Auktionsergebnis ausschließlich von den Bietern und dem Verkäufer berechnet ohne die Mithilfe von vertrauenswürdigen Dritten. Verlierende Gebote bleiben stets geheim. Wir präsentieren `libbrandt`, unsere Implementierung von vier Algorithmen mit verschiedenen Eigenschaften für Ergebnis-Öffentlichkeit und Gewinnpreis-Bestimmung, und beschreiben, wie diese im Rahmen eines echten Online-Auktionssystemes eingebunden werden können. Unsere Auswertung zeigt eine Rechenzeit- und voraussichtliche Nachrichtenlängen-Reduktion von mehr als 90% im Vergleich zu einer Implementierung derselben Algorithmen in RSA Arithmetik. Zusätzlich evaluieren wir die Skalierbarkeit von `libbrandt` in Bezug auf unterschiedliche Dimensionen und kommen zu dem Schluss, dass die Anwendung des präsentierten Systems in echten Anwendungen erfolgversprechend ist.

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I thank my advisors Sree Harsha Totakura, Christian Grothoff and Felix Brandt for their guidance and support during the making of this thesis. Christian discussed the cryptographic details with me and was quick to spot and explain any wrong turns I was thinking about. Sree Harsha helped me structure and focus my intermediary talk and was available for questions together with Christian in a weekly telephone conference. Felix came up with the whole concept of the protocols I was implementing and also referred me to two other implementations of his work which I did not know about in the beginning.

My thanks additionally go to the Google Summer of Code program and all people involved with my project for giving this thesis the initial momentum it needed.

For still being available and able to discuss openly his own and my new results nine years after his thesis I appreciate Philip Wassenberg.

I like to acknowledge my proofreaders Christian Grothoff, Sree Harsha Totakura, my father Werner Teich and Sven Hertle for their corrections and also insights.

Last but not least I thank all of my family and friends for their tremendous support and patience.

Chapter 1

Motivation

Since the Internet has become widely available to the public, auctions have migrated from the real to the virtual world [1]. From the early platforms to the ones used today, these platforms require bidders and sellers to trust the platform operator in two ways: First, clients have to assume that the auction service operator is determining auction's outcomes correctly. One can easily imagine how any collusion between operator and sellers or bidders would allow the operator to modify the outcome for the benefit of the colluding parties. Additionally, users need to trust the platform operator to handle information gained from the auctions confidentially (e.g., bids, winning price, winner identity). The resulting "big data" attacks are more subtle than direct collusion with a buyer or seller. With detailed market profiles, a platform operator can selectively compete in areas where sellers are having high profit margins, or target price-insensitive buyers with advertisements for overpriced products.

Due to these trust issues and increased interest in solving them, research emerged trying to reduce the required trust in platform operators by creating new cryptographic auction resolution algorithms. In Chapter 2 we describe Brandt's foundational algorithms [2] in this domain, and contribute a few minor improvements of our own. We discuss some approaches to online auctions with different focus in more detail in Chapter 5.

The main focus of our work is to provide a practical implementation of a secure auction protocol that eliminates the need for a trusted third party. This includes achieving reasonable performance in terms of runtime and bandwidth, and a well-documented API that application developers can directly use. We have chosen Brandt's work [2] as the base for our `libbrandt` implementation (Chapter 4), because it provides the complete independence of a trusted third party and shifts the trust issue from the platform to the seller and bidders themselves.

The architecture in which `libbrandt` should be used to best incorporate it's privacy properties is described in Section 3.

Results from an experimental evaluation comparing different auction protocols can be found in Chapter 6. We discuss issues relating to the system's usability in Chapter 7.

Contemporary popular auction platforms not only determine the outcome of the auction, but they also provide the infrastructure where auctions are published and users can search for goods they intend to buy. For this thesis, we ignore this market making operation and leave the problem that market making platforms are able to analyze the market volume for future work.

1.1 Auction Formats

There are several common auction formats used in real-world and online auctions with various sets of optional features. The **English auction** or **first price auction** is the most widespread scheme and belongs to the family of iterating auctions. The seller sets a publicly known starting price and then bidders are allowed to place publicly announced bids higher than the previous bid until either no bidder wants to place a higher bid or an optional timer runs out. On termination the good is sold to the highest bidder, who has to pay his own bid. There is also the possibility of a *reserve price*, hidden at first. If the bid which won the auction is lower than the reserve price, the good is not sold.

Another common format is the **Dutch auction** where the seller sets a high starting price and then iteratively decreases the price. The bidder who first announces he wants to buy the good wins the auction. Dutch auctions are known for selling tulips. In a typical tulip auction, there is not just one tulip to be sold but many. The first bidder who accepts the announced price is allowed to choose how many tulips he wants to buy for that price. Afterwards the auction continues until there are no more tulips left or the optional reserve price is hit. From the bidder's point of view the Dutch auction is strategically equivalent to a **sealed bid auction** [3], where each bidder submits a hidden bid, so no bidder knows any of the other bids before placing his own. After all bids have been collected, the winner is determined.

In some auctions the winners do not have to pay their own bid, but just the next lower bid or even the lowest winning or highest loosing bid in case of multiple similar items being sold. For the special case of only one good and a sealed bid format this scheme is called a **Vickrey auction**. In contrast to English auctions, in Vickrey auctions there is a game-theoretic weakly dominant strategy for bidders: The bidder chooses his bid equal to his real valuation of the item independently of all other bids [2] [4, Chapter 3.1.2].

In this thesis we will discuss and implement sealed bid auction formats using one of the following two formats:

- **First price (English auction).** One indivisible item is sold and the winner has to pay the price of his own bid.

- $M + 1$**st price.** One or more items are sold and the winners have to pay the price of the highest loosing bid. For $M = 1$ this is a Vickrey auction. For a multi-unit auction we need to choose $M > 1$ equal to the number of units to sell. A single bidder can only bid for one unit at the same time so if a bidder wants to purchase more than one unit from a single auction he has to create that many virtual bidders and can also choose to use different bids for each of them.

The second dimension the seller has to choose, is the outcome privacy. This leads to the four possible formats shown in Table 1.1. The outcome is defined as the set of winners and the price that they have to pay per unit.

- **Private outcome.** No information is leaked to the loosing bidders or other third parties. Only the seller and the winners learn the outcome of the auction. In case of multi-unit auctions only the seller learns all the winners. The winners only learn that they have won (and the price), but not who else has won a unit.

- **Public outcome.** The outcome is also visible for loosing bidders, but the loosing bids are still not revealed to anyone.

Table 1.1: Auction schemes available in `libbrandt`.

first price private outcome	first price public outcome
M + 1st price private outcome	M + 1st price public outcome

All of our auction format algorithms also have the following features and restrictions:

- **No third party.** We do not require a third party for resolving the auction outcome. Only the bidders and the seller are involved in the protocol.

- **Limited price pool.** Due to restrictions in the algorithms we will only handle auctions with an apriori fixed and finite number of possible bids. Bidders need to choose their bid from this set which must be defined by the seller before the auction starts.

- **No hidden reserve price.** If the seller wants an ensured minimum amount of money for his goods, he has to set the range of possible prices accordingly or join his own auction with enough virtual bidders of the reserve price so that he basically sells every unit to himself which would have been sold for a price lower than his reserve price otherwise.

1.2 Main Contributions

The main theoretical contribution of this thesis is the translation from Brandt's RSA (Rivest-Shamir-Adleman cryptosystem)-based algorithms to elliptic curve-based ones. We also addressed a few documented [5] and undocumented issues in the algorithms. In practical terms, the transition to elliptic curve cryptography reduces computation time and network usage while maintaining an equivalent security level.

We provide extensive empirical results demonstrating the viability of the resulting design and implementation for auctions at scales that are relevant in practice. Our implementation is available online at `git://gnunet.org/libbrandt` and can be cloned with `git`. This code is released under the GPLv3+ license[1].

Additionally we designed the system architecture in which `libbrandt` should be used.

[1] https://www.gnu.org/licenses/gpl-3.0.en.html

Chapter 2

Background

In this chapter, we describe Brandt's algorithm for secure private auctions. However, we chose a formulation that is already using our adaptations of the original algorithm to elliptic curves. If you are interested in understanding how exactly the algorithm ensures correctness and privacy, please read the original paper [2] from Brandt. We only provide a brief introduction in the following Section 2.2.

2.1 Introduction to RSA and Elliptic Curve Similarities

Since both, RSA and elliptic curve cryptography, are based on finite groups of the same structure, many algorithms in one of those systems can be translated to the other. For example, in RSA computing the public key p from the secret key s is done by computing $p = g^s$, where g is a publicly known group element with the property of being a generator of this cyclic group. Here s is a simple scalar, while p and g have to be considered as group elements, although they also just contain scalar values. In elliptic curves the same procedure is done with multiplication: $P = sG$. Out of convention we write group elements of elliptic curves in upper case, and we need to clearly distinguish them from scalars, for which we will use lower case. P and G are *points* on the curve. Cryptography then needs a one-way function which can be computed easily while the inverse function needs to be hard. For RSA this is the before mentioned exponentiation, which relates to the discrete logarithm problem (or in the case of the RSA trapdoor also to factoring large numbers). In elliptic curves the equivalent one-way function is the product of a point and a scalar. Given only $P = sG$ and G, it is hard to compute the elliptic curve discrete logarithm (ECDL) $s = P/G$. Another operation occurring often in Brandt's algorithms is the multiplication of two group elements. This corresponds to point addition in elliptic curves. From these two examples a simple explanation for the translation would be to just replace multiplication with addition, division with subtraction, and exponentiation with multiplication. However, care needs to be taken since these simple rules do not

apply for scalar-only operations like computing $2M + 2$ or the powers of 2 in the public outcome schemes. Here we need exactly that power of 2 to index the winner during outcome determination. Still, as we will see in the next Section, RSA and elliptic curves are similar enough to translate Brandt's algorithms from one crypto system to the other.

2.2 Overview of Brandt's Algorithms

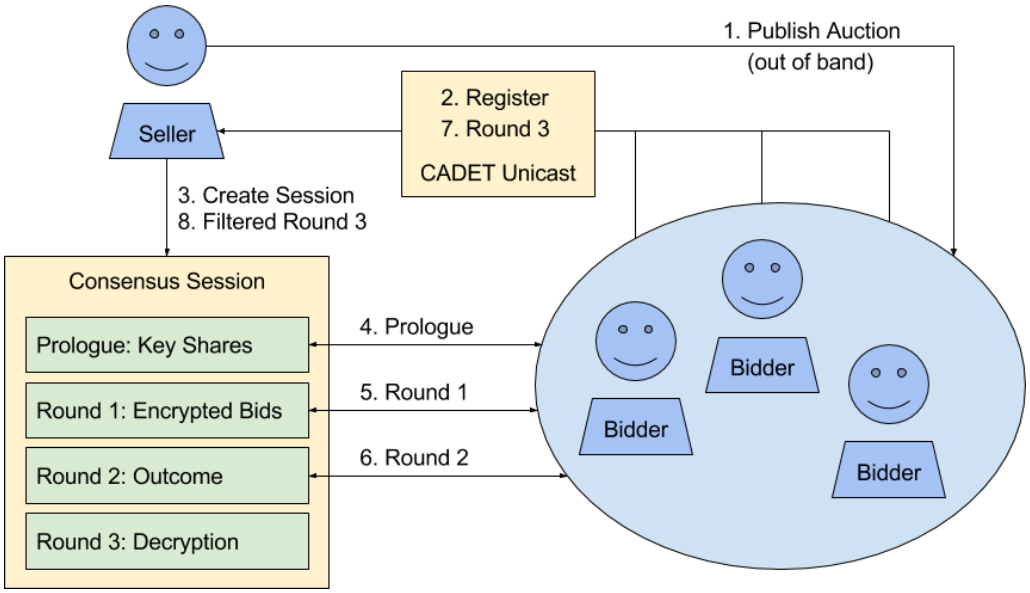
The algorithms we will be using are based on a few key concepts.

- **Bids.** For each auction the seller defines a price pool with strictly monotonic descending order, e.g. $pool = (\$100, \$80, \$60, \$40, \$20)$. The cardinality of this tuple is assigned to $k = |pool|$. Each bid has to be selected from this pool by choosing the 1-based index of the price the bidder wants to pay, e.g. $\text{bid}_{Alice} = 2$ for a bid of \$80. From this a *bid vector* is constructed by taking a vector with k elements all being 0 and setting the one with the index of the bid to 1, e.g. $b_{Alice} = (0, 1, 0, 0, 0)^\top$.
- **Outcome Determination.** To compute the winner(s) and selling price of the unit(s) the bid vectors of all n bidders are combined in a few matrix-vector products to one resulting outcome vector. This result vector only has a single component set to 0 and its index denotes the selling price. The Winner-determination depends on the auction format. In the private outcome variants this is done by computing one outcome vector for every of the n bidders. These outcome vectors differ slightly in that the 0 component can only be found in the winner's outcome vector. For public outcome schemes only one additional outcome vector is computed. The winning price is represented by the 0 component in the first outcome vector and the winner can be computed from the component with the same index in the second outcome vector.
- **El Gamal Encryption.** [6] To prevent bidders from learning each other's bids during the outcome vector computations, bid vectors are first encrypted with El Gamal. El Gamal is a public key cryptography system working with RSA as well as elliptic curves [7] and has the special property of homomorphism. This means we can encrypt a plaintext, make some computations with the cyphertext, decrypt the result and get the computations done directly to the plaintext, i.e. $\text{Enc}(m_1 \cdot m_2) = \text{Enc}(m_1) \cdot \text{Enc}(m_2)$ with $\cdot$ being the group operation. This property is also used to create a shared key pair. While the private key shares always stay with the bidders who generated them, the resulting public key is used for encryption. Since no party knows the combined private key, decryption is also done in shares where each bidder decrypts part of the outcome only. This ensures that the encrypted bids can not be read by anyone, only the computed outcome is revealed after the shared decryption.

- **Zero Knowledge Proofs.** To ensure correctness of all computations exchanged between the participants, zero knowledge proofs (ZKP) are used to certify every step without revealing the secret parts to other parties. These proofs can certify the knowledge or property of some input without revealing the input itself. The simplest example is to proof the knowledge of the private key to a presented public key without revealing the private key to the verifying party. We use three different such proofs described in Section 2.5.

The concepts are put together in a protocol with several rounds depicted in Figure 2.1. First the bidders compute a single shared public El Gamal key, where no single party can derive the private key. Then in the first round this key is used by each bidder to encrypt his bid and share the ciphertext of this encryption with all other participants. In the second round the encrypted bid vectors are used in the matrix multiplication to compute the encrypted outcome parts which are also shared between all participants. In the third and last round each bidder decrypts his share of the outcome and depending on the format publishes his whole share directly or just a part of it by letting the seller filter out the curve points, which allow to compute this bidder's personal outcome vector. Here the seller needs to do this filtering, because he must be able to compute all bidder outcome vectors to learn each winner. Afterwards every participant can combine the parts he received and derive an outcome from it. In private outcome formats each participant has access to different parts in the end so that only the winner(s) can derive the price and that they won. The seller can always derive all winners and the selling price. The losing bidders either learn only that they lost in private outcome schemes, or the winner(s) and the selling price in a public outcome scheme.

Figure 2.1: Round Based Protocol Overview.



2.3 Switching to the Ed25519 Elliptic Curve

The algorithms proposed by Brandt [2] are all based on the RSA arithmetic. Because of the increasing attack efficiency against RSA through index calculus [8], we translated them to elliptic curve arithmetic. Elliptic curve crypto systems are suspected to not be vulnerable against index calculus based attacks [9] and therefore the difference between RSA and elliptic curves is expected to extend even further in the future when better index calculus attacks are found which do not apply to elliptic curves. Our implementation uses the Ed25519 curve [10]. Clear benefits are the increased CPU performance and significantly smaller bandwidth requirements for a group element of similar security level, leading to lower bandwidth requirements (see Chapter 6). A benefit of Ed25519 over other elliptic curves is that we do not have to check if the points received over the network are actually points on the curve. For other elliptic curves this can be necessary to prevent weakening attacks [10]. The key size inflexibility of Ed25519 is also not a huge problem, since it is considered secure for the next few years and it is easy to change the curve in the future.

For the remainder of this chapter let G be the base point of the Ed25519 elliptic curve and $q = ord(G)$ the order of it. 0 is the neutral point for addition on the Ed25519 curve. Each curve point and scalar is serialized into a chunk of 32 bytes when sent over the network.

2.4 Privacy and Security Properties

Before we describe the detailed protocol schemes in Sections 2.6 to 2.10, we state the security properties of the proposed system. First, we look at what information can be gained by different kinds of passive adversaries.

- A loosing bid stays private to the respective bidder under the assumption of one or a collusion of more honest but curious other participants. This means the colluding participants can not gain more information by sharing their own data with each other. For example if the seller colludes with all but a single bidder and this bidder looses the auction, his bid is still not computable from all the information of the colluding participants. The protection of loosing bids depends on the secrecy of the private keys chosen by the respective bidders and the intractability of the ECDL problem.
- A collusion of honest but curious *bidders* can not derive information about winners outside of their own group in private outcome schemes. This not only depends on the intractability of the ECDL problem, but on the honesty of the seller as well. If the seller colludes with anyone, he can obviously reveal the outcome to them.

- A passive external adversary with control over the network could collect metadata and derive the auction parameters n, k and the auction format. If a bidder misbehaves and is excluded, that fact and the bidder's host can be observed as well. Since messages need to be encrypted, nothing about the content will be revealed if the encryption keys are kept secret by participants.

An active adversary can still not gain any extra information if the communication channels are authenticated correctly and non-malleable ZKPs are used [5]. However, an active adversary with control over the network can launch denial of service (DoS) attacks to disrupt the protocol and even target specific bidders by dropping the respective messages. This would lead to the targeted bidder not providing his round computation in time and then he will be excluded from the auction and lose his escrow deposit.

Another open attack possibility for a malicious seller is when he falsely reports one or more bidders as not having finished their round computations in time. This is especially hard to prevent for the round 3 messages in private outcome schemes, which are unicasted directly to the seller and so no other party can certify the correct behaviour of the bidder. For all other round messages, bidders could certify each other's correct behaviour. If a reputation system for the sellers is used one could also be more confident about a seller's correct behaviour before joining an auction.

2.5 Zero Knowledge Proofs

As proposed by Brandt [2, Section 5.2] and shown by Dreier et al. [5] the zero knowledge proofs used by the protocol need to be non-interactive. We used the Fiat-Shamir heuristic [11] to translate the proofs given by Brandt to non-interactive ones. These non-interactive versions require a key derivation function HKDF to allow both parties to compute the same challenge c deterministically. We use the `GNUNET_CRYPTO_kdf_mod_mpi()` implementation[1] of HKDF [12] with a per proof constant string as salt to compute the challenges from the respective input values. Of course we also translated the proofs from the original RSA variant to Ed25519 to work with the rest of the protocol.

We assume that the Ed25519 curve parameters G, q and 0 are known to all participating entities. A proof is a tuple of some computed and/or generated values combined to a single blob of data. For each of the following three proofs all incorporated curve points and scalars as well as the output size in bytes are given.

[1] Available from `https://gnunet.org/git/`

2.5.1 Proof 1: Knowledge of an ECDL

Alice and Bob know V, but only Alice knows x, so that $V = xG$. With the following instructions she can prove the knowledge of x to Bob without revealing the value of x.

1. Alice chooses $z \bmod q$ at random and calculates $A := zG$.
2. Alice computes $c := \text{HKDF}(G, V, A) \bmod q$.
3. Alice sends A and $r := (z + cx) \bmod q$ to Bob.
4. Bob computes c as above.
5. Bob checks that $rG = A + cV$.

Prover only knowledge:	x, z
Common knowledge:	V
Proof:	r, A (64 bytes)

2.5.2 Proof 2: Equality of Two ECDL

Alice and Bob know V, W, G_1 and G_2, but only Alice knows x, so that $V = xG_1$ and $W = xG_2$. With the following instructions she can prove the knowledge of x which fulfills those two equations.

1. Alice chooses $z \bmod q$ at random and calculates $A := zG_1$ and $B := zG_2$.
2. Alice computes $c := \text{HKDF}(G_1, G_2, V, W, A, B) \bmod q$.
3. Alice sends A, B and $r := (z + cx) \bmod q$ to Bob.
4. Bob computes c as above.
5. Bob checks that $rG_1 = A + cV$ and $rG_2 = B + cW$.

Prover only knowledge:	x, z
Common knowledge:	V, W, G_1, G_2
Proof:	r, A, B (96 bytes)

2.5.3 Proof 3: An Encrypted Value is One out of Two Values

Alice proves that an El Gamal encrypted value $(\alpha, \beta) = (M + rY, rG)$ decrypts to one of the fixed values 0 or G without revealing which is the case, in other words, it is shown that $M \in \{0, G\}$.

If $M = 0$:

1. Alice chooses $r_1, d_1, w \bmod q$ at random and calculates $A_1 := r_1 G + d_1 \beta$, $A_2 := wG$, $B_1 := r_1 Y + d_1(\alpha - G)$ and $B_2 := wY$.
2. Alice computes $c := \text{HKDF}(G, \alpha, \beta, A_1, A_2, B_1, B_2) \bmod q$.
3. Alice chooses $d_2 \leftarrow c - d_1 \bmod q$ and $r_2 \leftarrow w - rd_2 \bmod q$.

If $M = G$:

1. Alice chooses $r_2, d_2, w \bmod q$ at random and calculates $A_1 := wG$, $A_2 := r_2 G + d_2 \beta$, $B_1 := wY$ and $B_2 := r_2 Y + d_2 \alpha$.
2. Alice computes $c := \text{HKDF}(G, \alpha, \beta, A_1, A_2, B_1, B_2) \bmod q$.
3. Alice chooses $d_1 \leftarrow c - d_2 \bmod q$ and $r_1 \leftarrow w - rd_1 \bmod q$.

Then regardless of the value of M:

4. Alice sends $A_1, A_2, B_1, B_2, d_1, d_2, r_1, r_2$ to Bob.
5. Bob computes c as above.
6. Bob checks that

$$c = d_1 + d_2 \bmod q \tag{2.1}$$
$$A_1 = r_1 G + d_1 \beta \tag{2.2}$$
$$A_2 = r_2 G + d_2 \beta \tag{2.3}$$
$$B_1 = r_1 Y + d_1(\alpha - G) \tag{2.4}$$
$$B_2 = r_2 Y + d_2 \alpha. \tag{2.5}$$

Prover only knowledge:	r, x, w
Common knowledge:	Y, α, β
Proof:	$A_1, A_2, B_1, B_2, d_1, d_2, r_1, r_2$ (256 bytes)

2.6 Prologue

These steps are the same for all protocols following in this Section.

Let n be the number of participating bidders/agents in the protocol and k be the number of possible valuations/prices for the sold good. $a \in \{1,2,\ldots,n\}$ is the index of the agent executing the protocol, while $i,h \in \{1,2,\ldots,n\}$ are other agent indices. Let $j,b_a \in \{1,2,\ldots,k\}$ with b_a denoting the price p_{b_a} bidder a is willing to pay. We assume that the prices are sorted such that $\forall j : p_j < p_{j+1}$.

All messages are signed by the sender. All zero knowledge proofs are checked immediately when they are received, and the protocol only continues if the proofs are accepted. If the proof is not acceptable, the receiving agent publishes the unacceptable proof to all other participants, causing the protocol to be restarted with the malicious participant excluded and possibly fined.[2]

2.6.1 Generate Public Key Y

All bidders:

1. Choose a private key share $x_{+a} \in \mathbb{Z}_q$ and
   $\forall i,j$: Blinding factors m_{ij}^{+a} mod q and
   $\forall j$: El Gamal encryption parameters r_{aj} mod q at random.
2. Publish $Y_{\times a} := x_{+a}G$ along with Proof 1 of $Y_{\times a}$'s ECDL (96 bytes).

The seller and all bidders compute:

$$Y := \sum_{i=1}^{n} Y_{\times i}. \tag{2.6}$$

2.6.2 Round 1: Encrypt Bid

All bidders:

1. $\forall j$: Set $B_{aj} := \begin{cases} G & \text{if} \quad j = b_a \\ 0 & \text{else} \end{cases}$ and publish $\alpha_{aj} := B_{aj} + r_{aj}Y$ and $\beta_{aj} := r_{aj}G$
2. $\forall j$: Using Proof 3 to show that (α_{aj},β_{aj}) decrypts to either 0 or G and

[2] For example, bidders may be expected to pay p_1 into an escrow account when joining the auction. That amount would then be forfeit given proof that they failed to properly execute the protocol. This might be necessary to discourage denial-of-service attacks.

3. Using Proof 2 to show that:

$$\mathrm{ECDL}_Y\left(\left(\sum_{j=1}^{k}\alpha_{aj}\right)-G\right)=\mathrm{ECDL}_G\left(\sum_{j=1}^{k}\beta_{aj}\right). \tag{2.7}$$

The message has k parts, each consisting of 10 points plus an additional 3 points for the last proof. Therefore the message is $k \cdot 10 \cdot 32 + 3 \cdot 32 = k \cdot 320 + 96$ bytes large.

2.7 First Price Auction Protocol with Private Outcome

2.7.1 Round 2: Compute Outcome

All bidders compute and publish $\forall i, j$:

$$\gamma_{ij}^{\times a} := m_{ij}^{+a}\left(\left(\sum_{h=1}^{n}\sum_{d=j+1}^{k}\alpha_{hd}\right)+\left(\sum_{d=1}^{j-1}\alpha_{id}\right)+\left(\sum_{h=1}^{i-1}\alpha_{hj}\right)\right) \text{ and} \tag{2.8}$$

$$\delta_{ij}^{\times a} := m_{ij}^{+a}\left(\left(\sum_{h=1}^{n}\sum_{d=j+1}^{k}\beta_{hd}\right)+\left(\sum_{d=1}^{j-1}\beta_{id}\right)+\left(\sum_{h=1}^{i-1}\beta_{hj}\right)\right) \tag{2.9}$$

with corresponding Proofs 2 for $\mathrm{ECDL}(\gamma_{ij}^{\times a}) = \mathrm{ECDL}(\delta_{ij}^{\times a})$.

The message has nk parts, each consisting of 5 points. Therefore the message is $n \cdot k \cdot 5 \cdot 32 = n \cdot k \cdot 160$ bytes large.

2.7.2 Round 3: Decrypt Outcome

All bidders unicast $\forall i, j$:

$$\varphi_{ij}^{\times a} := x_{+a}\left(\sum_{h=1}^{n}\delta_{ij}^{\times h}\right) \tag{2.10}$$

with a Proof 2 showing

$$\mathrm{ECDL}(\varphi_{ij}^{\times a}) = \mathrm{ECDL}(Y_{\times a}) \tag{2.11}$$

to the seller who broadcasts all $\varphi_{ij}^{\times h}$ and the corresponding proofs of correctness for each i, j and $h \neq i$ after having received all of them.

The unicast message has $n \cdot k$ parts, each consisting of 4 points. Therefore this message is $n \cdot k \cdot 128$ bytes large.

The broadcast message by the seller has $(n-1) \cdot n \cdot k$ parts, each consisting of 4 points. Therefore it is $(n-1) \cdot n \cdot k \cdot 128$ bytes large.

In the private outcome formats this last broadcast message from the seller is the barrier after which the outcome is revealed to the winners. A malicious seller could decide not to reveal the outcome after he has learned it himself. However, the only situation where this would be beneficial to the seller is when a specific bidder wins, which the seller does not want to sell his goods to. We argue that the net gain would be higher if the seller just did not accept this unwanted bidder's registration for the auction, since not

revealing the outcome within the round time will result in the seller loosing reputation and the optional auction creation fee.

2.7.3 Epilogue: Outcome Determination

All bidders compute:

$$\forall j : V_{aj} := \sum_{i=1}^{n} \gamma_{aj}^{\times i} - \sum_{i=1}^{n} \varphi_{aj}^{\times i}. \tag{2.12}$$

The seller is able to compute V_{hj} for all bidders h, since he has all γ and φ. If $\exists w : V_{aw} = 0$, then bidder a is the winner of the auction. p_w is the selling price.

2.8 First Price Auction Protocol with Public Outcome

2.8.1 Round 2: Compute Outcome

All bidders compute and publish $\forall j$:

$$\gamma_j^{\times a} := m_j^{+a}\left(\sum_{h=1}^{n}\sum_{d=j+1}^{k}\alpha_{hd}\right) + \sum_{h=1}^{n} 2^{h-1}\alpha_{hj} \text{ and} \tag{2.13}$$

$$\delta_j^{\times a} := m_j^{+a}\left(\sum_{h=1}^{n}\sum_{d=j+1}^{k}\beta_{hd}\right) + \sum_{h=1}^{n} 2^{h-1}\beta_{hj} \tag{2.14}$$

with corresponding Proofs 2 for:

$$\text{ECDL}\left(m_j^{+a}\left(\sum_{h=1}^{n}\sum_{d=j+1}^{k}\alpha_{hd}\right)\right) = \text{ECDL}\left(m_j^{+a}\left(\sum_{h=1}^{n}\sum_{d=j+1}^{k}\beta_{hd}\right)\right). \tag{2.15}$$

The message has k parts, each consisting of 5 points. Therefore the message is $k \cdot 5 \cdot 32 = k \cdot 160$ bytes large. Note, that compared to auctions with private outcome the message size is reduced by a factor of n because we do not need to compute different outcome functions for each bidder. Therefore we also do not need nk blinding factors m_{ij}^{+a} in this scheme, but only k different ones m_j^{+a}.

2.8.2 Round 3: Decrypt Outcome

All bidders compute and publish $\forall j$:

$$\varphi_j^{\times a} := x_{+a}\left(\sum_{h=1}^{n}\delta_j^{\times h}\right) \tag{2.16}$$

with a Proof 2 showing

$$\text{ECDL}(\varphi_j^{\times a}) = \text{ECDL}(Y_{\times a}). \tag{2.17}$$

This message has k parts, each consisting of 4 points. Therefore the message is $k \cdot 4 \cdot 32 = k \cdot 128$ bytes large, reducing message size by a factor of n compared to the first price auction format with private outcome.

Note, that in the public outcome case this message can be directly broadcasted and does not have to be unicasted to the seller who then broadcasts part of all the received messages back to the bidders. This optimization allows the last bidder, after having

received all other messages from this round, to not send his own part of the decryption after he learns the outcome. To prevent this DoS attack the malicious bidder can be detected by not broadcasting his message within the maximum round duration and then the auction can be restarted with the same parameters, bids and bidders except the malicious bidder being blocked. He would loose his registration fee for the auction in exchange for learning the outcome a little bit earlier than the remaining bidders who will compute the same outcome eventually. The only possible beneficiary scenario would be if the malicious bidder changed his mind after learning that he himself is a winner and does not want to purchase the unit anymore. He then would "pay" for his withdrawal from the auction with the registration fee. In this case the malicious bidder still has to gamble for all other bidders' messages to arrive early enough so he can still send his own message before the round timer runs out in case he decides to actually purchase the item. Therefore this strategy will not work if there are two bidders waiting for each others' decryption message revealing the outcome.

2.8.3 Epilogue: Outcome Determination

The seller and all bidders compute $\forall j$:

$$V_j := \sum_{h=1}^{n} \gamma_j^{\times h} - \sum_{h=1}^{n} \varphi_j^{\times h}. \tag{2.18}$$

The V_j with the biggest index p where $V_p \neq 0$ denotes that p is the selling price. The seller and all bidders then compute $d := \mathrm{ECDL}(V_p)/n$ which is doable since it has only small factors. The lowest w where the bit w is set in d denotes the winner.

2.9 M + 1st Price Auction Protocol with Private Outcome

This auction format allows the seller to offer more than one item of the same type in a single auction. Bidders can also bid on as many of them as they desire by creating that many separate bidding agent processes also with different bids. For example in an auction with three flowers being sold and two bidders, Alice and Bob with Alice wanting to pay \$5 for the first, \$4 for the second and \$2 for the third flower and Bob wanting to buy only two flowers both for the price of \$3 the outcome would be that Alice receives two flowers and Bob receives just one. The M + 1st highest bid would be the 4th bid of the sorted bid list (\$5, \$4, \$3, \$3, \$2) and therefore both bidders would have to pay \$3 per flower they receive. Restricting each bidding agent process to only one bid keeps the protocol simple and prevents leaking statistics about how many items bidders desire.

In a M + 1st price auction there are two types of ties possible. First there could be more than one M + 1st highest bid. For example with $M = 2$ and bids (\$4, \$3, \$2, \$2, \$1) the two bids of \$2 would cause such a tie. The second possible tie involves the M + 1st highest bid and at least one other winning bid. If we modify the example by removing the \$3 bid, there would be one \$2 bid which should be a winner and the other \$2 bid denotes the M + 1st highest bid and therefore the selling price.

The tie breaking for the first type is not only computationally intensive, but also adds significant complexity to the protocol if done in an optimized way [4]. This would lead to a huge amount of additional code (which would likely introduce more bugs [13]). Thus, we decided to keep it simple and take another approach for tie breaking the M + 1st price format. We took the simplest one [14, Chapter 5.2], interlacing the bids, so that no two bidders are allowed to bid the same price. On the application level we will still handle k_{app} different prices, but within `libbrandt` we will multiply that by a factor of n to get $k_{\text{lib}} := nk_{\text{app}}$ "prices" to be used internally.

The bids are scaled up as well by the mapping $\forall i \in [1, n] : b_{i,\text{lib}} = b_{i,\text{app}} n - i + 1$. Therefore the set of allowed bids for bidder i is defined as $\{j | k_{\text{lib}} - j + 1 \equiv i \pmod{n}\}$.

This method causes bidders with a lower index to win in case of ties. To verify that bidders obey the restriction, we introduce an additional zero knowledge proof to the "Encrypt bid" message. The expansion will be done right at the beginning of an auction by `libbrandt` and the reverse mapping is applied before reporting the auction outcome to the application, so this expansion is transparent to the application. In the remaining part about the M + 1st price auction protocols we will use k instead of k_{lib}, so k will be divisible by n without remainder.

Unfortunately, this tie breaking simplification has the disadvantage of revealing the identity of the bidder who had the highest bid amongst the losing bidders. If there are multiple bidders fulfilling this criteria (having a tie on the M+1st bid), then only the one

with the lowest index will be revealed. This problem only affects M + 1st price auctions with private outcome and can be prevented using anonymized bidder identities, so the winners do not learn who placed the M + 1st highest bid.

An advantage of this price pool expansion is that we do not need to care about the second kind of ties anymore. Since every bidder has a distinct set of prices which he can choose from, the bids of any two bidders can not be the same. This is an improvement over the Wassenberg implementation which does not support tie breaking winners in M + 1st price auctions.

2.9.1 Addition to Round 1: Encrypt Bid

The bidders also have to use Proof 2 to show that:

$$\mathrm{ECDL}_Y\left(\left(\sum_{j=1}^{k/n} \alpha_{a,jn+a}\right) - G\right) = \mathrm{ECDL}_G\left(\sum_{j=1}^{k/n} \beta_{a,jn+a}\right). \tag{2.19}$$

Together with the other proofs in this message we know:

1. All Proofs 3: $\forall j : B_{aj} = 0$ or $B_{aj} = G$
2. First Proof 2: $\exists_{=1}\, j : B_{aj} = G$
3. Second Proof 2: $\exists_{=1}\, j \in \{j | k - j + 1 \equiv a \pmod{n}\} : B_{aj} = G$.

From this we can infer that only one component of the bid vector B_a is set to G and it is one of the components exclusive to bidder a which we need due to the multiplication of possible prices by n for M + 1st price auctions. This additional Proof 2 increases the message size by 96 bytes to a total of $k \cdot 320 + 192$ bytes.

2.9.2 Fixes for Minor Issues in M + 1st Price Auctions

In Step 5 of the protocol specification in [14, Section 5.1] we found two minor issues.

First, the nested product in the γ and δ formulas contains an index-out-of-bounds problem. The value of d will range up to k, but there is no $\alpha_{h,k+1}$, since $\alpha_{h,k}$ is the last element in that array. Even if it is clear from a mathematical point, that this last element is to be ignored, the direct implementation would lead to out of bounds errors. Therefore we split the inner product into two separate ones.

The second issue is the denominator of γ and probably just a typo. Here we need to compute the power of Y to the scalar $2M + 1$, not the product with it.

The updated formulas we used for our translation to elliptic curve arithmetic follow:

$$\gamma_{ij} := \frac{\prod_{h=1}^{n} \prod_{d=j}^{k} (\alpha_{hd}\alpha_{h,d+1}) \left(\prod_{d=1}^{j} \alpha_{id}\right)^{2M+2}}{(2M+1)Y} \tag{2.20}$$

$$\text{changed to} \qquad \frac{\prod_{h=1}^{n} \left(\prod_{d=j}^{k} \alpha_{hd} \cdot \prod_{d=j+1}^{k} \alpha_{hd}\right) \left(\prod_{d=1}^{j} \alpha_{id}\right)^{2M+2}}{Y^{2M+1}} \tag{2.21}$$

$$\delta_{ij} := \prod_{h=1}^{n} \prod_{d=j}^{k} (\beta_{hd}\beta_{h,d+1}) \left(\prod_{d=1}^{j} \beta_{id}\right)^{2M+2} \tag{2.22}$$

$$\text{changed to} \qquad \prod_{h=1}^{n} \left(\prod_{d=j}^{k} \beta_{hd} \prod_{d=j+1}^{k} \beta_{hd}\right) \left(\prod_{d=1}^{j} \beta_{id}\right)^{2M+2} \tag{2.23}$$

2.9.3 Round 2: Compute Outcome

All bidders compute and publish $\forall i, j$:

$$\gamma_{ij}^{\times a} := m_{ij}^{+a} \left(\sum_{h=1}^{n} \left(\sum_{d=j}^{k} \alpha_{hd} + \sum_{d=j+1}^{k} \alpha_{hd} \right) + (2M+2) \sum_{d=1}^{j} \alpha_{id} - (2M+1)\, G \right) \text{ and} \tag{2.24}$$

$$\delta_{ij}^{\times a} := m_{ij}^{+a} \left(\sum_{h=1}^{n} \left(\sum_{d=j}^{k} \beta_{hd} + \sum_{d=j+1}^{k} \beta_{hd} \right) + (2M+2) \sum_{d=1}^{j} \beta_{id} \right) \tag{2.25}$$

with corresponding Proofs 2 for $\text{ECDL}(\gamma_{ij}^{\times a}) = \text{ECDL}(\delta_{ij}^{\times a})$.

The message has $n \cdot k$ parts, each consisting of 5 points. Therefore the message is $n \cdot k \cdot 5 \cdot 32 = n \cdot k \cdot 160$ bytes large.

2.9.4 Round 3: Decrypt Outcome

This protocol step is exactly the same as Round 3 (Section 2.7.2) from the first price private outcome protocol.

2.9.5 Epilogue: Outcome Determination

All bidders compute:

$$\forall j : V_{aj} := \sum_{i=1}^{n} \gamma_{aj}^{\times i} - \sum_{i=1}^{n} \varphi_{aj}^{\times i}. \tag{2.26}$$

The seller is able to compute V_{hj} for all bidders h, since he has all γ and φ. If $\exists w : V_{aw} = 0$, then bidder a is a winner of the auction. p_w is the selling price.

2.10 M + 1st Price Auction Protocol with Public Outcome

The tie prevention from the M + 1st price auction protocol with private outcome apply here as well including the addition to Round 1.

2.10.1 Round 2: Compute Outcome

All bidders compute and publish $\forall j$:

$$\gamma^{\times a}_{\text{price},j} := m_j^{+a}\left(\sum_{h=1}^{n}\left(\sum_{d=j}^{k}\alpha_{hd} + \sum_{d=j+1}^{k}\alpha_{hd}\right) - (2M+1)\,G\right) \text{ and} \tag{2.27}$$

$$\delta^{\times a}_{\text{price},j} := m_j^{+a}\left(\sum_{h=1}^{n}\left(\sum_{d=j}^{k}\beta_{hd} + \sum_{d=j+1}^{k}\beta_{hd}\right)\right) \text{ and} \tag{2.28}$$

$$\gamma^{\times a}_{\text{winner},j} := \gamma^{\times a}_{\text{price},j} + \left(\sum_{h=1}^{n}\sum_{d=j+1}^{k} 2^{h-1}\alpha_{hd}\right) \text{ and} \tag{2.29}$$

$$\delta^{\times a}_{\text{winner},j} := \delta^{\times a}_{\text{price},j} + \left(\sum_{h=1}^{n}\sum_{d=j+1}^{k} 2^{h-1}\beta_{hd}\right) \tag{2.30}$$

with corresponding Proofs 2 for $\text{ECDL}(\gamma^{\times a}_{uj}) = \text{ECDL}(\delta^{\times a}_{uj})$.

Since the second summands of $\gamma^{\times a}_{\text{winner},j}$ and $\delta^{\times a}_{\text{winner},j}$ do not depend on the index of the participant computing them, we do not have to send the $\gamma^{\times a}_{\text{winner},j}$ and $\delta^{\times a}_{\text{winner},j}$ points. The receiving participants can just compute this second summand once and add it to all received $\gamma^{\times a}_{\text{price},j}$ and $\delta^{\times a}_{\text{price},j}$ points to compute the respective $\gamma^{\times a}_{\text{winner},j}$ and $\delta^{\times a}_{\text{winner},j}$ points themselves. The message has k parts, each consisting of 5 points. Therefore the message is $k \cdot 5 \cdot 32 = k \cdot 160$ bytes large.

2.10.2 Round 3: Decrypt Outcome

All bidders compute and publish $\forall j$:

$$\varphi^{\times a}_{\text{price},j} := x_{+a}\left(\sum_{h=1}^{n}\delta^{\times h}_{\text{price},j}\right) \text{ and} \tag{2.31}$$

$$\varphi^{\times a}_{\text{winner},j} := x_{+a}\left(\sum_{h=1}^{n}\delta^{\times h}_{\text{winner},j}\right) \text{ and} \tag{2.32}$$

$$\tag{2.33}$$

with two Proofs 2 for $u \in \{\text{price}, \text{winner}\}$ showing

$$\text{ECDL}(\varphi_{uj}^{\times a}) = \text{ECDL}(Y_{\times a}). \tag{2.34}$$

This message has k parts, each consisting of $2 \cdot 4 = 8$ points. Therefore the message is $k \cdot 8 \cdot 32 = k \cdot 256$ bytes large.

The effects discussed in Section 2.8.2 also hold in this format.

2.10.3 Epilogue: Outcome Determination

The seller and all bidders compute $\forall j$:

$$V_j := \sum_{h=1}^{n} \gamma_{\text{price},j}^{\times h} - \sum_{h=1}^{n} \varphi_{\text{price},j}^{\times h}. \tag{2.35}$$

$$W_j := \sum_{h=1}^{n} \gamma_{\text{winner},j}^{\times h} - \sum_{h=1}^{n} \varphi_{\text{winner},j}^{\times h}. \tag{2.36}$$

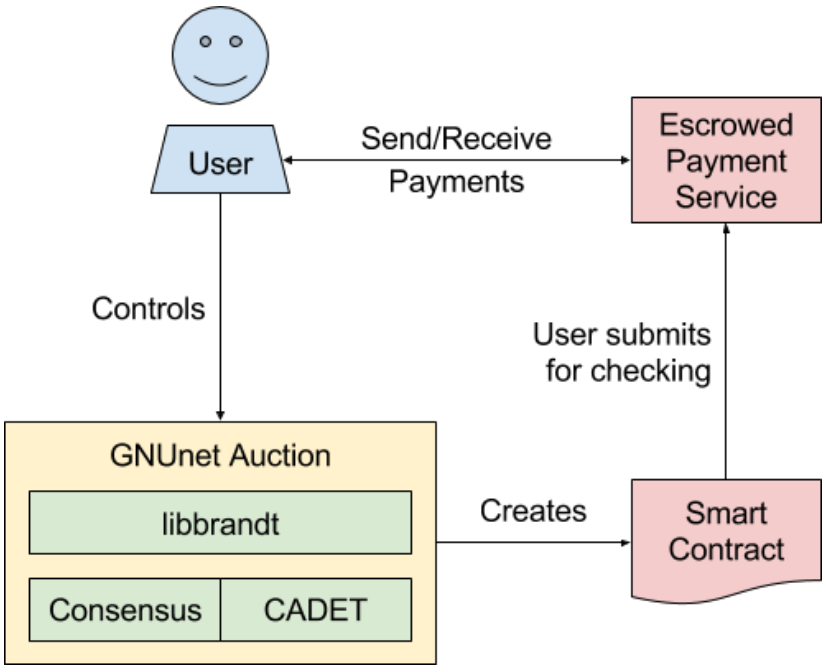
The selling price is the p where $V_p = 0$. The seller and all bidders then compute $d := \text{ECDL}(W_p)/n$ which is doable since it only has small factors. Every w, where bit w is set in the binary representation of d, denotes a winner.

Chapter 3

Architecture

In this chapter we describe how the auction protocols from Brandt could be used for real world auctions. Multiple components are involved and we describe each one and how they interact with each other. The user's perspective of the system is depicted in Figure 3.1.

Figure 3.1: System Architecture.



3.1 Sellers and Bidders

For a simple example, suppose seller Sally decides she wants to sell her A-Team DVD collection. First she runs the GNUnet auction program to create an auction description file which contains the auction format to be used, the price mapping, the maximum number of bidders, the time when the auction starts, how long the rounds may take, her payment system information, a textual description of the item and possibly some images of the DVDs. The price mapping could for example be defined by a minimum reserve price, a maximum price and a function which interpolates between those. Suppose she selects an $M + 1$st price auction with private outcome and $M = 1$ (Sally only has one collection), a price pool of \$20, \$21, \$22, ... to \$99. The auction shall start five days later and each round may take up to five minutes. She also enters her bank account information, a nice description and some photos so everybody can see that there are no scratches on the DVDs. Now Sally got her auction description file and can go ahead and publish her auction by uploading the file to the platform.

Suppose five people find this listing and are interested: Alice, Bob, Carol, Dave and Eve. Unfortunately, Eve notices that her computer is probably not fast enough for her to complete the auction rounds within the five minutes time frame and decides not to participate. This estimation can of course be done automatically by a script, so users do not have to calculate it themselves. Alice, Bob, Carol and Dave go ahead and join the auction by downloading the description file and giving it to the GNUnet auction program with their respective bids. Suppose Alice is willing to pay \$30, Bob does not really know the A-Team yet, but has read a positive review so he just bids \$25, Carol is a real fan and misses one of the offered DVDs in her own collection, so she bids \$42, and Dave wants to pay \$35. When joining the auction, all of them commit to a shipping address and place the \$20 minimum bid into an escrow account provided by the payment service.

After the five days have passed and the auction is started, the GNUnet auction service computes the outcome and returns to Carol that she has won. She pays the additional \$15 to Sally and provides her with her shipping address. The other bidders learn that they did not win the auction. Because they participated honestly their escrow funds are released and they can use them again to participate in other auctions.

3.2 Platform

In contrast to most existing online auction systems the platform in our architecture is only responsible for publishing offers and letting users search through the active offers. The platform is in no way part of the outcome determination and does neither learn loosing nor winning bids (except some participating bidder or seller reveals that

information). To make it worthwhile to run such a platform, the platform provider can demand fees for his services. For example publishing an auction offer on the platform costs the seller some amount of money either fixed or depending on the auction parameters.

The auction parameters most important to users are displayed publicly on the platform listing and if a bidder wants to join the auction, he can download the auction description file from this Web site. From the file the bidder's program can then find how to connect to the seller and register for the auction.

If the seller already knows a set of potential bidders he does not even need to use the platforms service at all. He could just send the auction description file to the prospective bidders directly. Thus, the platform Web site is optional and just used for publishing and finding auction offers.

3.3 GNUnet Auction

The GNUnet auction suite is the main component missing from making this architecture usable. We planned and started implementation of the GNUnet auction subsystem, but due to time limitations could not complete a working demonstrator. Our current progress is committed to the GNUnet repository[1] in the `src/auction/` directory.

The subsystem follows the usual GNUnet scheme of a service, a library, and multiple user interfaces. The service is a long-running process which can be started and stopped with the `gnunet-arm` program. The library just provides an interface to control the service via the GNUnet Inter-Process-Communication (IPC) channels. The user interfaces are several small programs responsible for several tasks described in Sections 3.3.2 to 3.3.5.

This whole subsystem could be used in two ways. Either the user installs GNUnet and all the dependencies for the auction programs locally on his computer, or they are compiled to JavaScript with, e.g., emscripten[2], so sellers and bidders can use the whole system from their Web browser without the need to install other software on their own computer. This would dramatically increase usability in exchange for the need to trust the browser environment. Also the underlying protocols would not change, so a bidder using the Web browser version can join auctions created by sellers using the local installation approach and the other way around. Bidders in the same auction also do not have to use the same approach. If the JavaScript version the user is planning to employ is conveniently served from the platform Web site, the user would have to trust, that the platform did not add code which breaks the privacy properties by secretly sending outcome information to the platform.

[1] https://gnunet.org/git/gnunet.git/

[2] https://github.com/kripken/emscripten

3.3.1 GNUnet Auction Service

The main part of the GNUnet auction subsystem is the auction service. It incorporates `libbrandt` to resolve auctions and a few other GNUnet components as well:

- **GNUnet CADET.** This component is used for unicast messages and especially for joining an auction before it starts. When the seller creates a new auction with the `gnunet-auction-create` program, the service opens a CADET port listening for joining bidders. The peer-id and port are stored in the auction description file created by `gnunet-auction-create`. The CADET channels provide all required features like message authentication and reliability; however, given that CADET provides off-the-record messaging the auction service still has to explicitly sign messages to ensure that cryptographic proofs can be exported and shown to other parties.
- **GNUnet Consensus.** This subsystem serves as the blackboard required by Brandt's protocols. A consensus session is opened by the seller and shared with all bidders when the auction starts. It will ensure that the broadcast messages of the protocol are exchanged between all participants. The consensus service is still responsible for ensuring that the messages are authenticated.
- **GNUnet Identity.** This subsystem could be used to build a reputation system for bidders. In this case after an auction the winning bidders could certify good behaviour, quick shipping time or other stuff for the seller. Before joining an auction, bidders could check the reputation of the seller before making a decision.

The auction service is used by sellers and bidders alike and has some responsibilities:

- **Active Auctions.** The service needs to manage a list of active auctions and which identities are participating in which role in these. After an auction is finished, the results of the auction need to be stored to disk so that the `gnunet-auction-info` program can later retrieve it.
- **Instance Mapping.** When a message is received, the service needs to map that message to the respective auction instance and forward it to `libbrandt` for handling. Also, in the reverse direction when sending messages, it needs to find the correct participant's CADET port or consensus session where the message needs to be forwarded to.
- **Registration Confirmation.** After bidders sent their joining message to the seller, the seller's service responds with an acknowledgement and his own local time. This is done so bidders can get a better estimate of when the auction will start exactly and when each round ends, as the seller's clock is considered authoritative for the auction.

- **Restarting Auctions.** When one participant does not adhere to the protocol, which can be detected by checking all the ZKPs, the seller's service needs to exclude him and then restart the auction with the same bids and all but the misbehaving bidder. A bidder is also excluded if he does not manage to finish a round within the specified time for the round.
- **Smart Contracts.** In case the auction finishes correctly or incorrectly in case of misbehavior, the service collects the transcript and produces a smart contract stating one of the following:
  - A specific other participant failed to provide a correct proof.
  - The bidder has not won the auction.
  - The bidder has won the auction and needs to pay a stated price.
  - Other bidders have won the auction and pay the stated price. This is used for public outcome auctions and for the seller who always learns the whole outcome.

  These smart contracts can then be used to provide cryptographic proof of the outcome. The winner(s) can show the proof to the escrow service and transfer the winning price minus the already deposited escrow. A wrong proof of a participating bidder can also be presented to the escrow service by the seller to block this bidder's deposit. In case of a misunderstanding between seller and winner(s), the smart contracts could also be handed to a judge, who will be able to apply the local laws to resolve the conflict, or to a payment processor to settle any remaining obligations.

3.3.2 The `gnunet-auction-create` Command

The `gnunet-auction-create` program takes all required parameters for creating an auction and forwards them to the auction service. Those parameters contain at least the following items:

- The **auction format** to be used, i.e. first price or M + 1st price and in the later case a value for M.
- A flag denoting if the auction should be of the **public outcome** type. If the flag is not set the private outcome type will be used.
- The **price mapping** and a currency with all prices sorted in a strictly descending order.
- The **starting time** of the auction. This is the limit until which new bidders may register.

- The maximum **round duration** after which an unresponsive bidder will be excluded from the auction and the protocol will restart.
- The **maximum number of bidders** so each bidder can check if they are able to compute each round within the maximum round duration.
- A **description** of the items for sale. We propose to use json with optional embedded serialized images since it is easy to parse and allows storing additional meta-data which is not immediately important to users.
- Some kind of **payment system information** so bidders can check if they actually can use the required payment system before trying to join the auction.

The service then opens the CADET port listening for joining bidders, schedules the start of the auction. It composes the auction description file from the given parameters and the open CADET port, signs the file and returns it to the program for publishing.

In interactive mode the program waits for the auction to finish and reports the outcome. In non-interactive mode the seller can check the outcome with the `gnunet-auction-info` command.

3.3.3 The `gnunet-auction-info` Command

The `gnunet-auction-info` program takes an auction description file as input and reports the auction parameters back to the user. If the user is already participating in that particular auction, the status and possibly the outcome is also reported. Another option should allow the program to query the service for all active auctions used by the peer or just a specific identity.

3.3.4 The `gnunet-auction-join` Command

The `gnunet-auction-join` program also takes an auction description file as input and forwards it to the service. The service then extracts the seller's peer identity and CADET port and sends a join request message to the seller. If the seller has not reached the maximum number of bidders for this auction yet, he reserves a spot for a limited amount of time and acknowledges that to the bidder. The bidder then needs to make his deposit to the escrow service and present proof of that as well as a salted hash of his bid value to the seller. If the proof of the escrow deposit is correct, the seller adds the bidder to the auction and stores the bid hash for checking later if this bidder does win the auction.

In interactive mode the program waits for the auction to finish and reports the outcome. In non-interactive mode the bidder can check the outcome with the `gnunet-auction-info` command.

3.3.5 A Runtime Estimation Script

To enable bidders to estimate if their computer hardware is capable of computing all the rounds within the desired time frame, some tool is needed which measures the computer's performance on a few simple cases and extrapolates the result to given input parameters n, k and the selected auction format. Depending on the estimate the user can decide if he is willing to participate in an auction with such parameters.

3.3.6 `libbrandt`

The core component of the GNUnet auction service is our `libbrandt` library. It is responsible for determining the outcome of an auction and is used by the seller and all bidders. After an auction is completed or a misbehaving bidder provided a wrong ZKP, it also compiles the cryptographic material into a smart contract.

Only the GNUnet auction service links against `libbrandt`, providing a unified interface for all programs.

3.4 GNU Taler as an Escrowed Payment Service

As payment system any system could be used, but we envision to use GNU Taler[3] as its properties fit the requirements [15]. Specifically, Taler allows the buyers to stay anonymous, supports transparency of the sellers (which should improve government approval), and most importantly allows for refunding escrow deposits without breaking bidder anonymity.

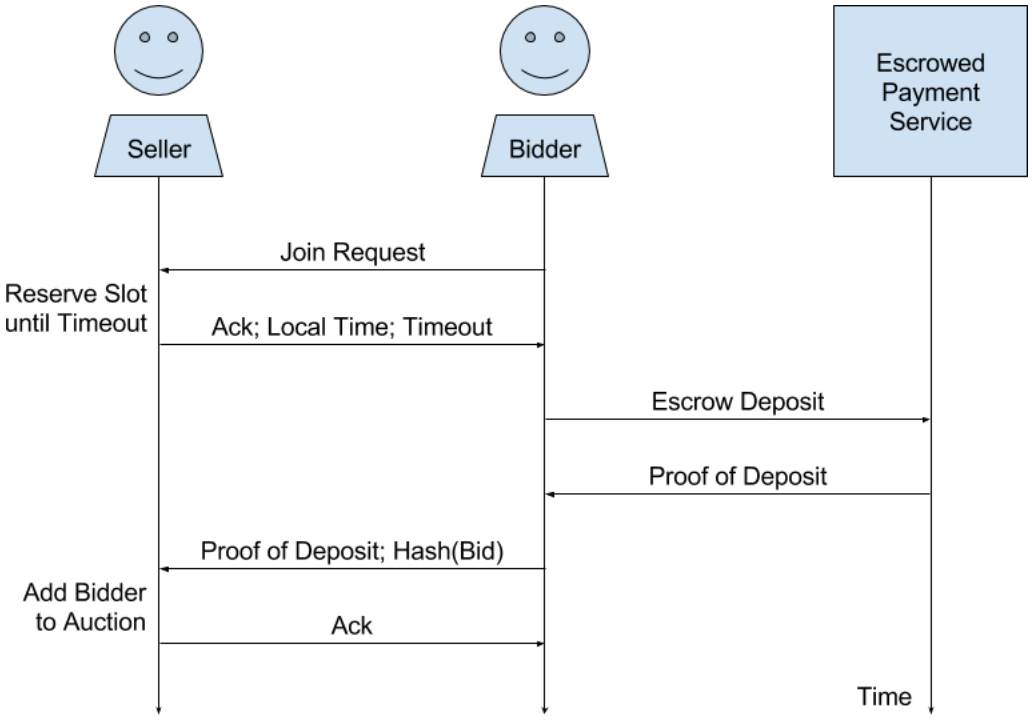
Requiring bidders to place a certain amount of money, e.g., the minimum bid, into an escrow service before the auction starts can deter DoS attacks by malicious bidders. If the seller proves misbehaviour of a bidder to the escrow service, the deposit should not be refunded and instead transfered somewhere else. To avoid any conflict of interest,[4] we suggest that involved parties (seller, bidder, escrow service, platform service) do not benefit in this case. Instead, escrow funds that are confiscated due to bad behavior should be donated to some independent charitable or non-profit organization. The deposit would only be refunded to the bidder if the seller did not report the bidder as misbehaving, and the bidder provides proof that he did not win the auction. If the bidder wins he can not provide a proof that he lost and the deposit will be transferred to the seller automatically. The winner would subsequently transfer the difference between the escrow deposit and the selling price. An anonymous malicious bidder can still disrupt an auction by placing the maximum bid, but refusing to pay for the item

[3] https://taler.net/

[4] Note that the seller is responsible for keeping time and thus also could misbehave.

after the auction is concluded in a timely fashion. In this case, the seller gets to keep the escrow deposit of the malicious bidder and can keep his item and offer it in another auction.

Figure 3.2: A Bidder Successfully Registers for an Auction.



A possible auction registration handshake between bidder and seller might look like depicted in Figure 3.2. The timeout is used to prevent bidders from blocking a space in the limited set of prospective bidders by just sending the join request but never placing the escrow deposit. The hash of the bid is used to prevent bidders from adjusting their bid to the number of bidders n after they learn that number when the auction starts. This is described in more detail in Section 4.2.3.

Chapter 4

`libbrandt`

`libbrandt` is a C-library and the core component of our auction system, responsible for determining the outcome of an auction. It is based on the algorithms and protocols by Felix Brandt [2], with some adjustments and optimizations described in the previous chapter. `libbrandt` is capable of computing the auction outcome for four different auctioning schemes (see Table 1.1). All of the auction schemes are sealed bid auctions.

In a first price auction only one item can be sold and only one bidder wins. This winner is the bidder with the highest bid amongst all bidders and he has to pay the amount of his own bid. In M + 1st price auctions a total number of M items is sold (so if $M = 1$ only one item like in the first price auctions) and the M bidders with the highest bids win. Unlike in the previous scheme, each winner only has to pay the price of the $M+1$st highest bid.

In the private outcome auctions only the winner(s) and the seller learn the winning price and the identity of the winner(s). In the public outcome auctions this information is revealed to all participants. The other bids and bid-sorted order of the bidders is never revealed to any party.

4.1 Requirements

The `libbrandt` library has several requirements for additional primitives which must be provided by the applications using the library.

1. **Unicast Communication Channel.** The bidders need to exchange messages with the seller for registration and during the protocol. This channel has to be reliable and should not delay messages for too long as there are time limits for the registration and maximum round duration.

2. **Broadcast/Multicast Communication Channel.** All participants need to publish messages between each other during the protocol. In the original paper this is called the "blackboard". All broadcast/multicast schemes which ensure that messages are delivered to all participants can be used in theory, but the choice will influence the overall communication cost greatly. See Section 6.4 for more information.
3. **Business Logic.** The application needs some kind of identity system so it is able to execute business logic based on the simple indices identifying participants in `libbrandt`. The mapping is established when the bidders register for an auction and needs to be preserved so the business logic can resolve the correct winners after the auction is completed. Ultimately, the indicies need to be mapped to sufficient data such that the business logic can process payments and ensure the winners receive the goods from the seller.
4. **Message Authentication.** As shown in [5], all messages need to be authenticated by the sender. This ties the message origin to the identity from the underlying identity system used by the application. It is also needed to prove the auction outcome to the others in the end.
5. **Availability.** The application needs to stay connected to the other participants throughout the whole auction. Alternatively, there needs to be a mechanism for reliable asynchronous message delivery.

4.2 Handling Corner Cases

There are some corner cases involved in handling the auction protocols. In this section we describe how we handle them.

4.2.1 No Bidders

When no bidder registers for the auction before the auction start timer triggers, the application of the seller will be notified with a `NULL` outcome.

4.2.2 M + 1st Price Auctions with fewer Bidders than Items to Sell

If the number of bidders $n \leq M$, then the algorithm can not compute the outcome. Since sellers choose their reserve price by setting the lowest possible price in the price map to the desired minimum amount they are willing to sell their goods for, we can immediately return the outcome without any computations. There is no restriction to our privacy goal by this shortcut, because all registered bidders are winners anyway, so

all of them must know the selling price. The identity of the winning bidders can not be protected by an algorithmic approach, since all of the bidders are winners.

4.2.3 First Price Auctions with only one Bidder

If only one bidder registers for a first price auction, then he would be able to cheat the seller by choosing his bid to be the lowest possible bid after learning that he is the only bidder. To prevent this we added the requirement for the bidder to commit to his bid already when registering for an auction and does not yet know the total number of bidders. This can be done by computing a cryptographic hash function of the bid with a random number used once (nonce) used for salting. After the auction ends, the winners have to reveal their nonce to the seller so he is able to verify that those bidders did not choose a different bid after learning the number of other participants.

4.3 On the Synchronous Protocol Structure

An issue is the fact that all participants need to be online during outcome resolution. One could adapt the round time to several hours or even a day and modify `libbrandt` to be able to store the state of auctions on disk and pause and resume computations at will which would ease the restriction so that participants only need to be online once each day to compute the current round. However, this might also increase the chance of some participants missing for one round leading to exclusion and restarting the auction with the remaining bidders which means even longer resolution times.

4.4 Application Programming Interface

Throughout the application programming interface (API) closure pointers are used. These are pointers of any type given from the application to `libbrandt` to reference a specific context. They are handed back to the application in the callback functions. Typically one would use pointers to the structs referring to the respective auction or participant from the applications point of view. Here we will provide the documentation of the most important functions from the `libbrandt` API but first will be the declarations of the function types used for callbacks.

4.4.1 BRANDT_CbResult

```
struct BRANDT_Result {
        /** Id of the bidder this instance refers to */
        uint16_t bidder;

        /** The price the bidder has to pay.
         * Only set if #status indicates the bidder has won. */
        uint16_t price;

        /** Status of the bidder */
        enum BRANDT_BidderStatus status;
};

typedef void
(*BRANDT_CbResult)(void                 *auction_closure,
                   struct BRANDT_Result results[],
                   uint16_t             results_len);
```

Functions of this type are called by libbrandt to report the auction outcome or incorrectly behaving participants.

auction_closure Closure pointer representing the respective auction. This is the Pointer given to `BRANDT_join()` or `BRANDT_new()`.

results An array of results for one or more bidders. Each bidder will only be listed once. Misbehaving bidder results and auction completion results are not mixed.

results_len Amount of items in `results`.

4.4.2 BRANDT_CbDeliver

```
typedef int
(*BRANDT_CbDeliver)(void       *auction_closure,
                    const void *msg,
                    size_t     msg_len);
```

Functions of this type are called by libbrandt to deliver messages to other participants of an auction. There are two variants how this Callback needs to be implemented. The first is delivering messages as unicast directly to the seller, the second is delivering messages as broadcast to all participants (bidders and seller). All messages need to be authenticated and encrypted before sending and the signature needs to be checked immediately by the recipients.

auction_closure Closure pointer representing the respective auction. This is the Pointer given to `BRANDT_join()` or `BRANDT_new()`.

msg The message to be delivered

msg_len The length of the message `msg` in byte.

return value 0 on success, −1 on failure.

4.4.3 BRANDT_CbStart

```
typedef uint16_t
(*BRANDT_CbStart)(void *auction_closure);
```

Functions of this type are called by libbrandt when the auction should be started as a seller. The application has to broadcast the ordered list of all bidders to the bidders and must return the amount of bidders to libbrandt. After this function is called no more new bidders may be accepted by the application.

auction_closure Closure pointer representing the respective auction. This is the Pointer given to `BRANDT_new()`.

return value The number of bidders participating in the auction.

4.4.4 BRANDT_new

```
struct BRANDT_Auction *
BRANDT_new (BRANDT_CbResult                    result,
            BRANDT_CbDeliver                   broadcast,
            BRANDT_CbStart                     start,
            void                               *auction_closure,
            void                               **auction_desc,
            size_t                             *auction_desc_len,
            struct GNUNET_TIME_Absolute        time_start,
            struct GNUNET_TIME_Relative        time_round,
            uint16_t                           num_prices,
            uint16_t                           m,
            int                                outcome_public,
            struct GNUNET_CRYPTO_EccDlogContext *dlogctx);
```

When called, this function creates a new auction as the seller. It takes one callback function pointer used to broadcast messages generated by libbrandt, one which is called when all bidders should be notified about the auction starting, and one which is called to report the outcome to the application. Apart from the closure pointer which will be used to refer to this auction instance in callbacks the function takes the auction parameters. The `dlogctx` parameter is used for public outcome auctions where we need to compute a simple ECDL in the end.The auction description blob is created and returned in the `auction_desc` pointer.

result Pointer to the result callback function

broadcast Pointer to the broadcast callback function

start Pointer to the seller start callback function

auction_closure Closure pointer representing the auction. This will not be touched by libbrandt. It is only passed to the callbacks.

auction_desc The auction information data as an opaque data structure. It is generated by this function and should be distributed to all possibly interested bidders. The application must sign this data block before publishing it!

auction_desc_len The length in byte of the `auction_desc` structure. Will be filled by `BRANDT_new()`.

time_start The time when the auction will start. Bidders have until then to register.

time_round The maximum duration of each round in the protocol.

num_prices The amount of possible valuations for the sold item(s). Must be > 0.

m The mode of the auction. If 0, it will be a first price auction where the winner has to pay the price of his bid. If > 0 it will be a $M + 1$st price auction selling exactly that amount of items and each winner has to pay the price of the highest loosing bid.

outcome_public If 1, the auction winner and price will be public to all participants, if 0, this information will only be revealed to the winner and the seller.

dlogctx The discrete log context pointer obtained from a call to `GNUNET_CRYPTO_ecc_dlog_prepare()`. Only needed for public outcome auctions.

return value if invalid parameters are passed, `NULL` is returned. else the return value is a pointer, which should only be remembered and passed to libbrandt functions when the client needs to refer to this auction. this is a black-box pointer, do not dereference/change it or the data it points to!

4.4.5 BRANDT_join

```
struct BRANDT_Auction *
BRANDT_join (BRANDT_CbResult                    result,
             BRANDT_CbDeliver                   broadcast,
             BRANDT_CbDeliver                   unicast,
             void                               *auction_closure,
             const void                         *auction_desc,
             size_t                             auction_desc_len,
             uint16_t                           bid,
             struct GNUNET_CRYPTO_EccDlogContext *dlogctx);
```

Call this function to join an auction described by the `auction_desc` parameter. It takes two callback function pointers which are used to send messages generated by libbrandt and one which is called to report the outcome to the application. Apart from the closure pointer which will be used to refer to this auction instance in callbacks the function takes the description blob generated by the `BRANDT_create()` call from the seller and the bid the user wants to place. The `dlogctx` parameter is used for public outcome auctions where we need to compute a simple ECDL in the end.

result Pointer to the result callback function

broadcast Pointer to the broadcast callback function

unicast Pointer to the unicast callback function

auction_closure Closure pointer representing the auction. This will not be modified by libbrandt itself, but is passed to the callbacks.

auction_desc The auction information data published by the seller. This is opaque to the application. Its content will be parsed. The application must check the signature on this data block before passing it to libbrandt!

auction_desc_len The length in byte of the `auction_desc` structure.

bid How much to bid on this auction.

dlogctx The discrete log context pointer obtained from a call to `GNUNET_CRYPTO_ecc_dlog_prepare()`. Only needed for public outcome auctions.

return value A pointer, which should only be remembered and passed to libbrandt functions when the client needs to refer to this auction. This is a black-box pointer, do not dereference/change it or the data it points to from the application!

4.4.6 BRANDT_parse_desc

```
int
BRANDT_parse_desc (const void                  *auction_desc,
                   size_t                      auction_desc_len,
                   struct GNUNET_TIME_Absolute *time_start,
                   struct GNUNET_TIME_Relative *time_round,
                   uint16_t                    *num_prices,
                   uint16_t                    *m,
                   uint16_t                    *outcome_public);
```

With this function an auction description data blob received from the seller can be checked before deciding to join the auction. See 4.4.4 for an explanation of the different auction description fields. All but the first two parameters are used as output pointers.

auction_desc The auction description blob published by the seller.

auction_desc_len Length of `auction_desc` in byte.

time_start Starting time of the auction. May be `NULL`.

time_round Maximum round time of the auction. May be `NULL`.

num_prices Number of possible prices. May be `NULL`.

m Auction mode. May be `NULL`.

outcome_public Outcome setting. May be `NULL`.

return value 0 on success, −1 on failure.

4.4.7 BRANDT_got_message

```
void
BRANDT_got_message (struct BRANDT_Auction *auction,
                    uint16_t              sender,
                    const unsigned char   *msg,
                    size_t                msg_len);
```

This function hands a received message related to a specific auction to libbrandt. It takes the sender's index and the message itself as parameters.

auction The pointer returned by BRANDT_join() or BRANDT_new() from which message msg was received.

sender The id of the sender.

msg The message that was received.

msg_len The length in byte of msg.

4.5 Implementation Details and Status

Internally the auction description blob created by the seller has the following structure. All fields are stored in network byte order.

```
struct BRANDT_DescrP {
        /** Starting time of the auction. Bidders have to join
         * the auction via BRANDT_join until this time */
        struct GNUNET_TIME_AbsoluteNBO time_start;

        /** The maximum duration in which the participants
         * have to complete each round. */
        struct GNUNET_TIME_RelativeNBO time_round;

        /** The number of possible prices */
        uint16_t k GNUNET_PACKED;

        /** Auction type. 0 means first price Auction,
         * >= 0 means M+1st price auction with
         * a number of m items being sold. */
        uint16_t m GNUNET_PACKED;

        /** Outcome type. 0 means private outcome,
         * everything else means public outcome. */
        uint16_t outcome_public GNUNET_PACKED;

        /** reserved for future use. Must be zeroed out. */
        uint16_t reserved GNUNET_PACKED;
};
```

The following details are still missing from our implementation:

- The bid commitment during registering for an auction.
- Compiling a smart contract from the cryptographic material when an auction ends or aborts due to a wrong ZKP.
- Checking and enforcing the maximum round time by the seller.
- Checking some of the intermediary results to not be 0. This is necesary to keep the protocol verifiable according to Dreier et al. [5].
- Caching round messages which are received out of order.

Chapter 5

Related Work

5.1 Brandt's Work

We are not the first to implement or evaluate auction protocols based on Brandt's design. Here we have a look at other research on his protocols.

5.1.1 Wassenberg Diploma Thesis and Implementation

A first implementation and evaluation of Brandt's algorithms was done by Wassenberg in his diploma thesis [4]. The work included implementations for first and second price auctions, both with private and public outcome versions. For the second price formats the M + 1st price auction algorithms were used, but due to missing tie-breaking the usage for multi-unit auctions was excluded. For multi-unit auctions three other implementations using Brandt's order statistic sub-protocol are provided. The uniform price module has the same pricing properties as our M + 1st price implementation, while the discriminatory price and the generalized Vickrey auction formats have slightly different pricing models.

The implementation does not provide a seller process returning the auction outcome. The blackboard process is only used for message exchange and SSL encryption and signing. For the implementation to be usable in real-word scenarios, the blackboard process needs to be enhanced to store all the exchanged messages, check the proofs and compute the outcome in the end.

5.1.1.1 The (t,u) Finding Heuristic

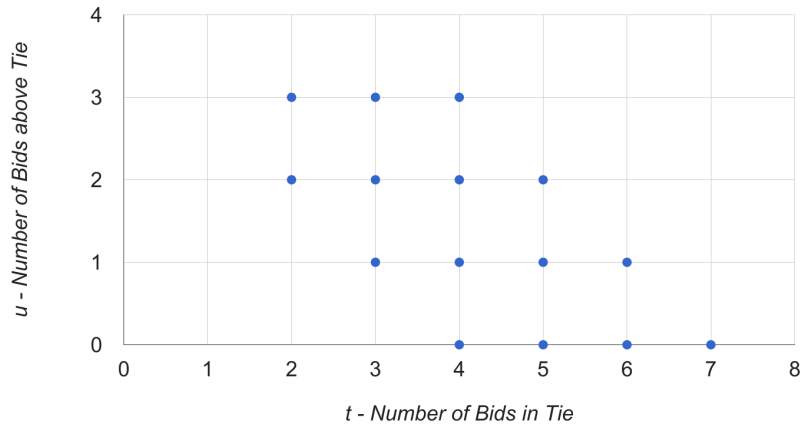
In Brandt's original M + 1st price auction handling tie breaking is needed if the tie involves the M + 1st highest bid [2, Section 4.2]. A tie can be described by two values,

t — the number of bids which have the same value and thus form the bid, and u — the number of bids which are higher than the tie. The $M+1$st price can only be involved in one or no tie at all. If there is no tie with the $M+1$st price, we do not need tie breaking and the protocol finishes immediately. If there is a tie, the t and u value needs to be found to compute the outcome.

Wassenberg developed a special heuristic [4, Section 4.1] to check the most probable t and u values first and the more uncommon ones last. For this heuristic the best case is if there is no tie at all (no (t,u) search is needed) and the worst case comes up if every bid is the same (the correct (t,u) pair is the last one checked). The badness of the worst case depends on the number of possible (t,u) pairs that need to be checked, which itself depends on a few inequalities limiting the area of possibilities as seen in Figure 5.1. We list those inequalities describing the borders here in clockwise order starting at the bottom of the shape.

- $u \geq 0$. There can not be a negative number of bids higher than the tie.
- $t + u \geq M + 1$. If the tie is only affecting the winner's bids, but not the $M+1$st price, it is not a relevant tie since the algorithm will still find the correct winning price (albeit possibly wrong winners, but that case is not handled by the (t,u) heuristic).
- $t \geq 2$. A tie obviously needs more than one bidder sharing the same bid.
- $u \leq M$. If there are more bids above the tie than M, the $M+1$st price is not part of the tie and therefore irrelevant.
- $t + u \leq n$. There can not be more bids than bidders.

Figure 5.1: Possible (t,u) Pairs for $M = 3$, $n = 7$ and any $k \geq n$.



The order in which the pairs are checked exactly is described by the following recursion from Wassenberg's thesis:

$$\text{Start with} \quad t := 2 \text{ and } u := 0. \tag{5.1}$$

$$\text{next}(t,u) := \begin{cases} (t+1, u-1) & \text{if} \quad u > 0 \\ (2, t-1) & \text{else} \end{cases} \tag{5.2}$$

When the next possible (t,u) pair is needed, this function is called until the output is a valid (t,u) pair according to the inequalities.

In graphic terms each diagonal $u = a - t$ with a being the offset from the origin is checked starting at the point with the lowest u value. The diagonals themselves are then checked with the one with the lowest a value first, so we can see that $(2,0)$ is checked first and $(n,0)$ is checked last, representing the worst case.

5.1.1.2 Bugs

We encountered an index-out-of-range crash in the uniform price algorithm preventing it from running correctly. We notified the author and he found the bug, but due to time limitations we could not re-run the tests against a fixed version of the code.

In the $M+1$st price **private outcome** implementation we noticed two other bugs. The first one was the (t,u) chain going too far in some cases, where the algorithm should have found the outcome earlier. Our smallest example for that is using the `englishmp` protocol with $M = 1$, $n = k = 3$ and the bids $\{1,1,0\}$ (zero based index, lower index represents higher bid). In that case the expected (t,u) chain would be $(2,0)$ into $(2,1)$ which should have already lead to the correct outcome. However, according to the output of the agents the chain continued one step further to $(3,0)$, which unnecessarily increases the runtime. The second bug was returning a wrong outcome and also occurred with the same setup. The case we presented above should return the third bidder as winner because he bid the lowest bid index which represents the highest bid, but the outcome reported by the agent is that the second bidder did win. We reported both bugs to the author and he acknowledged them to be probably related to bid ordering since the same test case works if only the bids are reshuffled to $\{0,1,1\}$. We saw some more occurrences of both bugs with different parameters, but never with the best and worst case scenarios of Wassenberg's heuristic (See Section 6.3.3).

We also noticed a bug where Wassenberg's $M+1$st price **public outcome** implementation runs into an endless loop in the heuristic. The `englishmpp` version with $M = 1$, $n = 3$, $k = 2$ and the bids $\{1,0,1\}$ lead to a (t,u) chain of $(2,0)$, $(2,1)$, $(3,0)$, $(3,1)$, $(3,1)$

and so on. This chain should have stopped at $(2, 1)$ and the value $(3, 1)$ is even impossible in this specific auction setup since we only have three bidders. We notified the author about this bug as well and he confirms it, noting that it did not occur in the specific cases he was measuring in his thesis. After his evaluation Wassenberg concludes that the algorithms are usable since the runtime is less than ten minutes for the base test case with six bidders, a price pool of size 40 and 2048 bit RSA keys.

5.1.2 Security Analysis

Dreier et al. evaluated the security properties of Brandt's protocols [5] and found several issues. They also provided efficient implementations for two possible attacks and measured it against their own parallelized version of Brandt's first price private outcome protocol. We did not measure against their code since it was not easily available and we already have two other RSA implementations. Following, you find proposals by the authors for how to address the four issues they found.

- The ZKPs must be non-interactive or non-malleable to prevent active attackers from breaking bid privacy. Our ZKPs are non-interactive not just to avoid that issue, but also to simplify the protocol.
- All messages need to be authenticated. We explicitly require this from applications using `libbrandt`.
- To ensure the protocol is verifiable, some intermediary results need to be checked to not be 1. In our elliptic curve version this translates to a check against 0.
- Also for verifiability, bidders need to prove that they actually used the same private key share for decryption and for generating the public key share previously. This additional check is implemented in `libbrandt`.

5.1.3 Stanford Implementation

Four students from Stanford wrote another implementation [16] of Brandt's scheme. It is using Golangs big integer package[1] for cryptographic computations, which are also based on the original RSA arithmetic. The authors implemented the first price auction with a private outcome. There are a few issues with their code. First they are using a non cryptographically secure pseudo random number generator (PRNG) seeded by the current system time. This leads to deterministic and therefore easily guessable private keys. Secondly, there is a syntax error in the code preventing compilation. Thirdly, paths to the auction file and certificates are hard coded which complicates running the code on only one host because the auction file has to be changed manually between

[1] https://golang.org/pkg/math/big/

starting of the clients. Parameters like the RSA primes and the number of possible prices are hard coded as well. Lastly, even after going through this setup, the actual execution of the protocol failed before the prologue due to a certificate validity error.

They also implemented a simple Python server responsible for auction participant address exchange and a simple certificate infrastructure. It supplies each participant with his own x509 [17] private key and certificate to use for authenticating the protocol messages. As the authors mention in their paper this server has to be a trusted party, since it is vulnerable to man-in-the-middle (MitM) attacks. There are a few other problems with the server. First, it does not provide encrypted transmission of the auction parameters and cryptographic material leading to further MitM attack possibilities. Secondly, it is hard coded to listen on port 80 which needs special privileges. The path to the certificate authority (CA) is hard coded as well. Thirdly, the x509 key material is supplied inside a zip file, but the client implementation does not automatically extract it which leads to unnecessary effort by the user of the program.

We did not include this code in our evaluation due to the required extensive bug fixes and hard coded values complicating automation. We reported all of the issues mentioned above to the authors on their GitHub page[2].

5.2 Other Auction Systems

Brandt is not the only researcher designing cryptographic auction protocols. Here we describe alternative designs and point out the differences in their privacy properties. Most of these schemes have more relaxed privacy goals and instead focus on computational efficiency.

5.2.1 Secure Vickrey Auctions without Threshold Trust

In their paper [18] Lipmaa et al. propose an auction scheme which incorporates a semi-trusted party A, called "auction authority", apart from the seller S and the bidders. A and S are supposed to verify each other's computations, but if they collude, they can map all bids to the bidders and even change the outcome of the auction. Like Brandt the authors also use homomorphic encryption to compute a product of all the encrypted bids which can then be analyzed by A to find the second-highest bid. Zero knowledge proofs are also used to certify the correct behavior of A.

Another important contribution of this paper, which we will use in our experimental evaluation, is the argument that a price pool of size 500 should be sufficient for most auctions.

[2] https://github.com/ashwinsr/auctions

5.2.2 t-Private and t-Secure Auctions

Hinkelmann et al. presented another cryptographic sealed-bid auction protocol [19] using garbled circuits, an evaluating auctioneer, and an "auction issuer" party. If the issuer and the auctioneer collude, bid privacy is broken.

5.2.3 A Sealed-Bid Knapsack Auction

Ibrahim used the Knapsack problem to create another cryptographic auction protocol [20]. It has similar properties to our approach, but differs in the following points:

- In Ibrahims protocol the seller learns which bids were placed, but not by whom. This allows the seller to estimate what people are willing to pay for this item and he might adopt his price range accordingly on future sales of the same item.
- The seller has to publish the winning price at the end. In our private outcome formats this is not required.
- The winner has to reveal himself to the seller in the end. This allows the winner to reconsider if he really wants to make the purchase. This is similar to our public outcome protocols, but only if the winner waits to receive all other Round 3 messages before broadcasting his own.
- There is no support for multi-unit auctions.

This last paper also contains a good overview of the pre-2011 cryptographic auction protocol research.

Chapter 6

Experimental Results

6.1 Algorithm Execution Time Test Setup

We compared (1) the RSA based algorithms of Wassenberg's [4] Java implementation and (2) our own prototype using the PARI/GP[1] scripting language (for private outcome first price auctions only[2]) against (3) our new Curve25519 based C library `libbrandt`. The authors of Ed25519 [10] argue that breaking their elliptic curve has similar difficulty to breaking RSA with ≈ 3000 bit keys. Therefore we used `ssh-keygen` [21] with the `-G` and `-T` options to generate a 3072 bit RSA safe prime p and derived the missing RSA parameters $q = (p-1)/2$ and $g = 3$.

All performance evaluations were done on a Lenovo X240 laptop with an Intel® Core™i7-4600U CPU at 2.1GHz. Since this CPU throttles automatically depending on the available power supply, all tests were executed while using the same power supply.

For all tests except the comparison between the Wassenberg heuristic and our own price expansion approach the bids for each bidder were chosen uniformly at random from the pool of possible prices. Therefore tied bids can and in some setups even have to occur.

To normalize the results, each test setup was run ten times (only with different randomized bids) unless noted otherwise, and then the median of those execution times is reported.

Since our tests did only run on one machine, we measured the execution time of the whole algorithm (i.e. sum of all single bidder measurements where available). For the charts in this chapter we divided the resulting median by the number of bidders in the respective test run to get an estimate on the computation time needed by a single

[1] http://pari.math.u-bordeaux.fr/

[2] Can be found in the `gp-scripts` directory of the `libbrandt` source.

bidder. This is slightly over-estimating the real cost of our own two implementations, as the seller's computation time is also included in the total. Since in Wassenberg's implementation there is no such seller process following the protocol, checking all the proofs and computing the outcome in the end, those results are closer to real-world per-bidder computation times. The original measurement data for the whole auction computation time can be found in Section A.2.

For all measurements we used the GNU `time` program [22] to measure the CPU time the process spent in user mode and in kernel mode (which tends to be below 1% of the user mode time) and added those two values together to get the total CPU time consumption for the process.

Where the measured data allowed for a useful trend line we also added one. All of those are polynomial trend lines with the maximum degree expected from the algorithm complexity, e.g. when using the price pool size k as the x-axis for measuring first price auctions, linear trend lines are used; while, when using the number of bidders n as the x-axis for measuring first price auctions with private outcome, quadratic trend lines are used.

Note, that Brandt describes the computation cost of the multiplication (RSA) as "typically negligible" compared to exponentiation (RSA). The actual complexity might be higher, but in our measurements the scalar point multiplication (Ed25519) dominated the measured complexity.

6.1.1 Notes on Measuring the Wassenberg Implementation

For Wassenberg's implementation we installed a `cgroup` [23] to restrict the processes of all agents to the same CPU core. Since the other two implementations also run the whole auction in a single thread, this should enable a fair comparison. We again summed up all the total time consumption values for the bidders to get a total execution time of the auction on a core.

We did not include Wassenberg's blackboard process measurement since it only serves as a message exchange tool with SSL tunneling and the other implementations did exchange messages directly. The blackboard computation time was consistently less than 2% of the total computation time of an auction.

We did not install the interface for Java to use the fast GMP[3] computations, so our tests only used the slower default `BigInteger` implementation. Switching this out for the GMP implementation should improve the speed to levels similar to the PARI/GP implementation which is using GMP internally.

The Java environment used for the benchmarks is Oracle JDK Version 1.8.0.112.

[3] https://gmplib.org/

6.2 First Price Auctions Results

First, we have a look at the single-unit auctions of the first price format. While one provides outcome privacy, the other one is less complex and therefore finishes faster. We fixed one of n or k to the value five although this is a rather small value to assume for number of prices and in some cases also for the number of bidders. This was done to improve comparability with the M + 1st price public outcome auction results. At the end of this chapter there is also a measurement of just `libbrandt` using the more reasonable price pool size of 512.

6.2.1 Private Outcome

We compare all three implementations on first price auctions with private outcome in Figures 6.1 and 6.2. The expected complexity of the algorithm is $O(n^2k)$. For these comparison we fixed the number of possible prices or number of bidders respectively to a value of five and varied the other value within the range $[2, 8]$. From the graphs we can see that `libbrandt` only needs around 7% of the computation time of Wassenberg's implementation. If we assume the GMP interface would be installed the runtimes of the Wassenberg and PARI/GP implementation should be similar, changing the runtime of `libbrandt` to roughly 10% of the runtime of either RSA version.

6.2.2 Public Outcome

In Figures 6.3 and 6.4 we can see the comparison between `libbrandt` and the Wassenberg implementation on first price auctions with public outcome. Since the algorithm does not have a separate result for each bidder, the complexity is just $O(nk)$ in this case.

Figure 6.1: First Price Private Outcome Auction with five Prices.

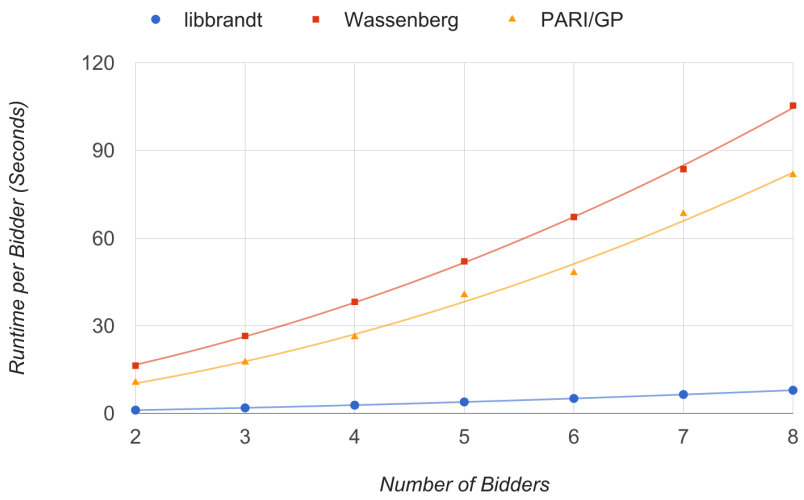


Figure 6.2: First Price Private Outcome Auction with five Bidders.

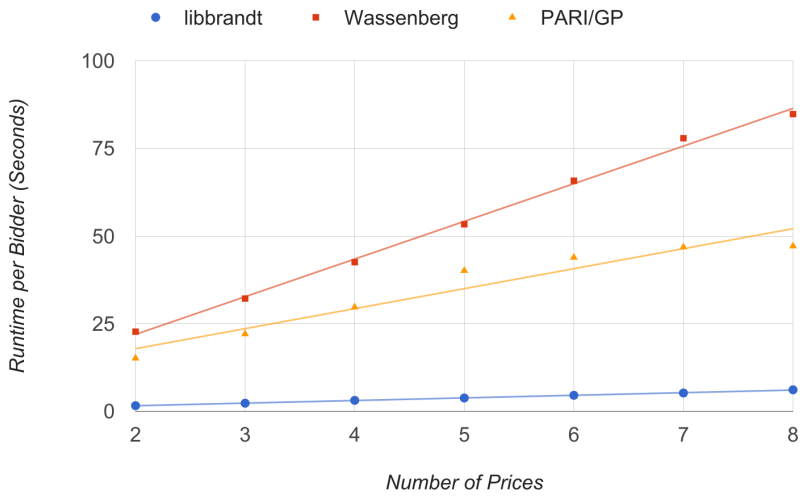


Figure 6.3: First Price Public Outcome Auction with five Prices.

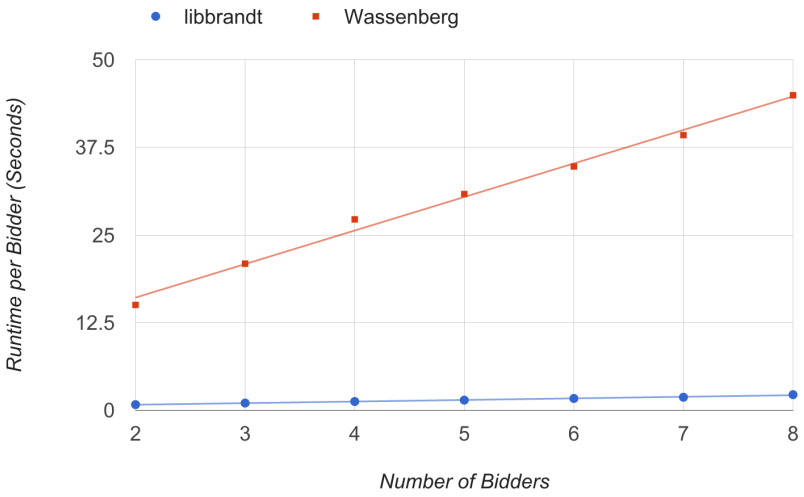
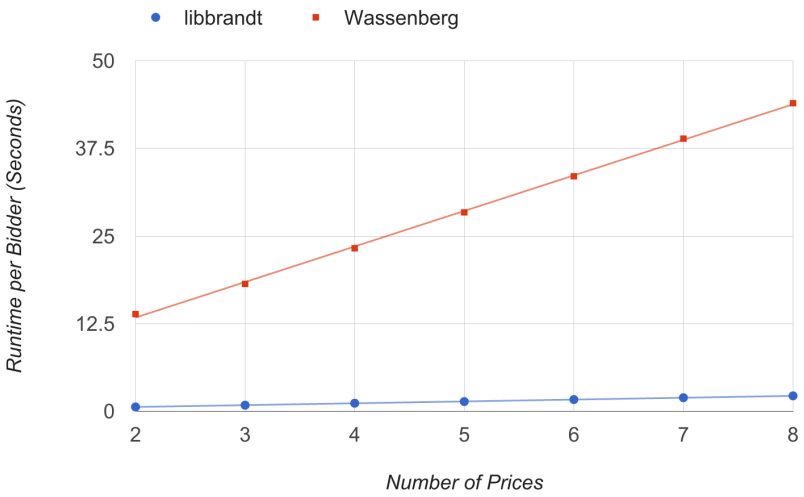


Figure 6.4: First Price Public Outcome Auction with five Bidders.



6.3 Multi-Unit Formats

We did not compare against the discriminatory, and generalized Vickrey auction implementations from Wassenberg since we have no implementation with equal properties. We could not use the uniform price algorithm which shares the same pricing properties as our $M+1$st price implementation due to the bug described in Section 5.1.1. Instead we used Wassenberg's M + 1st price implementation which is based on the same proposal by Brandt [14] for multi-unit auctions even though the author correctly points out its incorrect results in case of ties of the second type (M + 1st price is equal to one of the winning bids). However, the incorrect outcome should not lead to wrong computation times since it is only discovered after the very last round during winner determination and the implementation does not seem to complain if it detects more winning bids than M. Thus, this should lead to a fair comparison between our $M+1$st price implementation using the price pool expansion strategy to prevent ties and Wassenberg's M + 1st price implementation using his own (t,u) finding heuristic.

6.3.1 Private Outcome

One of the two bugs we noticed in Wassenberg's M + 1st price private outcome implementation (See Section 5.1.1) affects the runtime if the (t,u) chain is longer than it needs to be. Hence the measurements of the average case in Figures 6.5, 6.6 and 6.9 should only be considered carefully.

In the two M + 1st price private outcome comparisons we had to limit the fixed value to three and the x-axis value to a range of $[2,6]$ to limit the computation time as this algorithm has a complexity of $O(n^3k)$ in the `libbrandt` version and $\Omega(n^2k)$ in the Wassenberg version. We can already see Wassenberg's heuristic at work here producing runtimes with a high variance depending on where ties are located. Therefore also the median has a higher variance than for the other algorithms and we did not add trend lines for the Wassenberg series in those two graphs. However, we can see that `libbrandt` takes between 10% and 20% of the time of Wassenberg's implementation within the limited range of our input parameters. Furthermore, `libbrandt` could probably be improved by a reasonable factor by adopting Wassenberg's heuristic.

Figure 6.5: M + 1st Price Private Outcome Auction ($M = 1$) with three Prices.

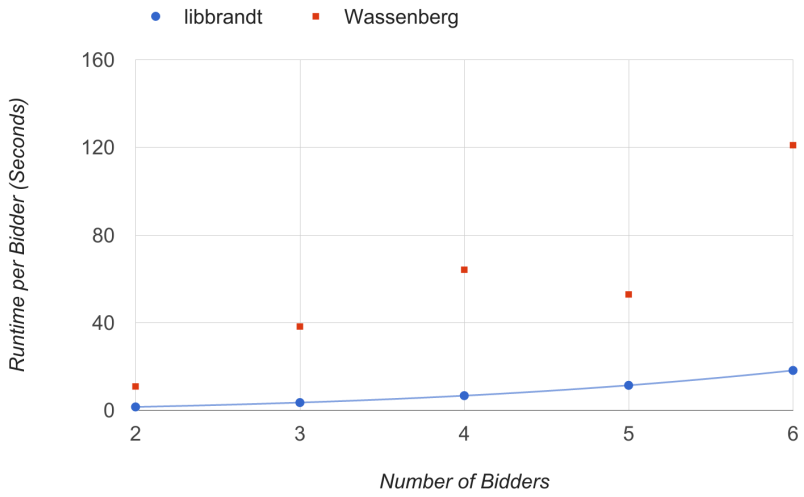
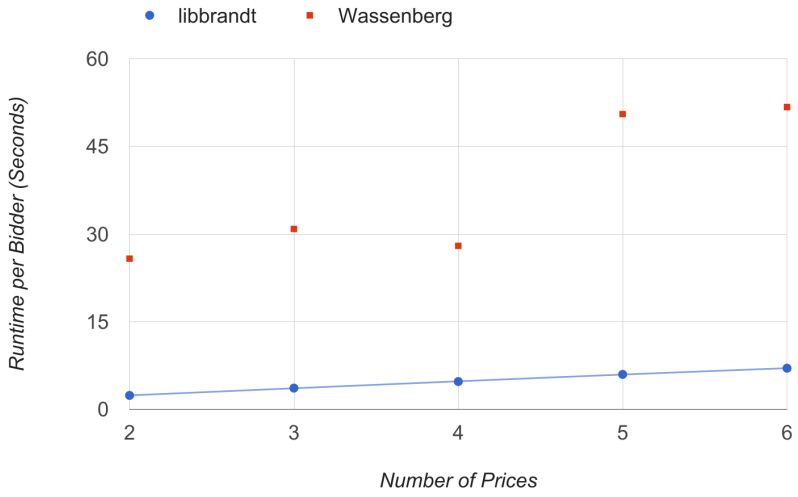


Figure 6.6: M + 1st Price Private Outcome Auction ($M = 1$) with three Bidders.



6.3.2 Public Outcome

Unfortunately the bug in the `englishmpp` implementation occurred too often with our desired parameters, which would lead to impaired results if we just removed the cases with the parameters where the bug occurs as the bids from the valid runs would not represent a uniformly random distribution anymore. Therefore we did not measure this algorithm and can only present the runtimes of our own M + 1st price public outcome algorithm in Figures 6.7 and 6.8. The results were pretty much as expected with a complexity of $O(n^2k)$.

6.3.3 Wassenberg's Heuristic for Tie Breaking

Since the heuristic strongly depends on the input parameter M we choose that as our x-axis values. To get a two-dimensional graph and still cover best and worst cases we decided to set $n = k = M + 2$. To generate a test series for the best case of Wassenberg's implementation we set each bidder's bid to be equal to his index, e.g. the first bidder will use bid 1, the second bidder will use bid 2, and so on. For the worst case test series we set every bid to the same value. For the average case we use uniform random sampling from the available price pool as in all other tests; additionally, we increase the number of test iterations to 20 to further reduce the chance of outliers affecting the result too much.

Our `libbrandt` implementation trades computation complexity against protocol complexity and the runtime complexity is not depending on the input, so we do not have such explicit best or worst cases for `libbrandt` and just measured one test series with the usual uniform random bid sampling for comparison against the three cases of the Wassenberg heuristic.

Figure 6.7: M + 1st Price Public Outcome Auction ($M = 1$) with five Prices.

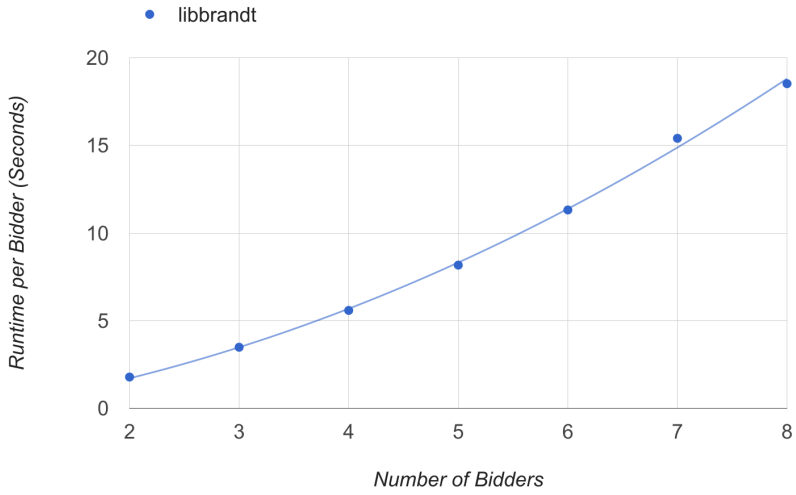
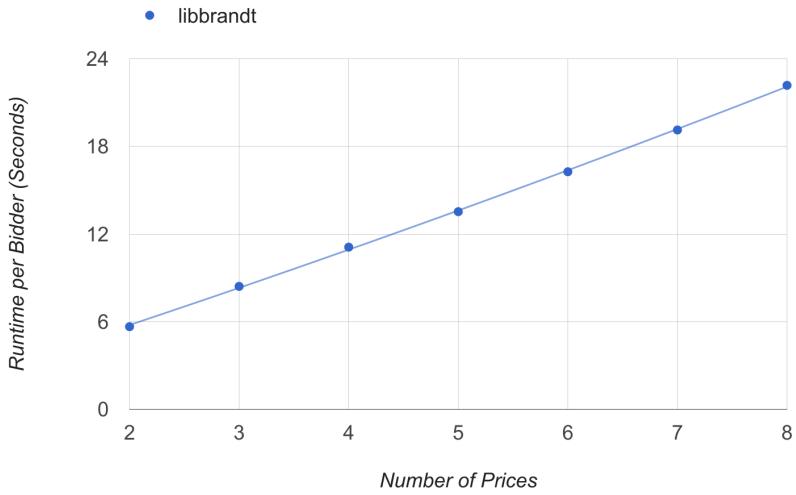
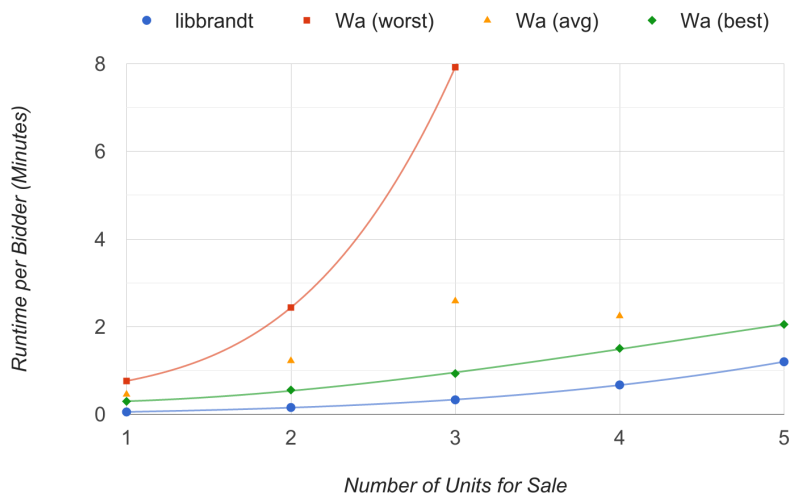


Figure 6.8: M + 1st Price Public Outcome Auction ($M = 1$) with five Bidders.



The results in Figure 6.9 show trend lines for the Wassenberg heuristics best and worst cases as well as the `libbrandt` implementation. Measurements for the worst and average case were cut off to prevent too long runtimes, especially since we increased the number of test run repetitions for the average case to 20 to try to get a more stable result. However, the data points of the average series still are not close enough to add a meaningful trend line. Also note that we changed the y-axis unit from seconds to minutes due to the long runtimes.

Figure 6.9: M + 1st Price Private Outcome Auction with $n = k = M + 2$.



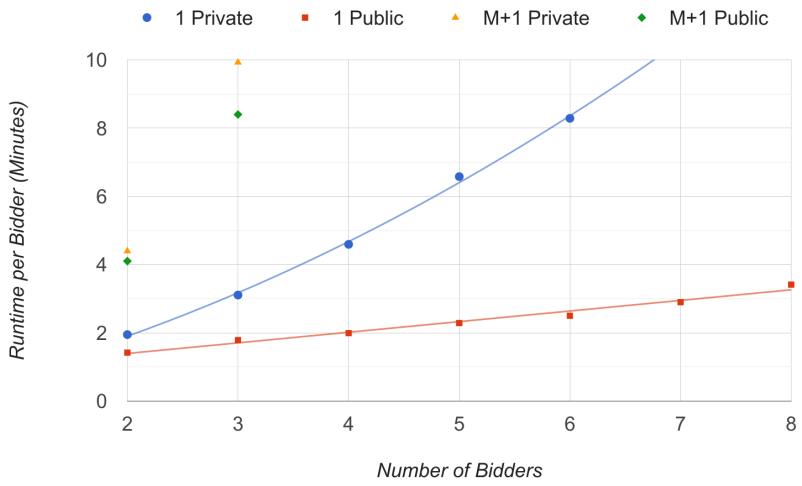
Because the two bugs in Wassenberg's M + 1st price private outcome implementation did not occur in the best and worst case setups, those give good upper and lower bounds for the (t, u) guessing heuristic while the average case series should still be considered carefully.

We can see that `libbrandt` still manages to stay faster than even the best case of the Wassenberg heuristic within our limited range of tests, but that will change for higher values due to the additional factor n in the complexity of `libbrandt`. For our test the average case managed to stay quite close to the best case. There are two arguments why selecting bids uniformly at random is not representative for real world auctions, but they even each other out. On the one hand we can argue that in real-world auctions it is very probable that the bids cluster around the estimated value of the item, so there is a higher probability of more and/or larger ties. On the other hand, if we take the reasonable price pool size of 500, we can assume that n is at least ten times smaller for most auctions leaving the few bidders plenty of choices in the price pool and therefore limiting the possibility of ties. In any case we can conclude from this graph that for performance it would be beneficial to adopt the Wassenberg heuristic in addition to the improvements done by switching to Ed25519.

6.3.4 `libbrandt` with a Reasonable Price Pool Size

For this last set of measurements we took the argument about price pool size by Lipmaa et al. [18] and set $k = 512$. Since this would lead to very long runtimes in the Wassenberg implementations we only measured libbrandt with the number of bidders ranging from two to five. In Figure 6.10 we abbreviated the first price auction with just a "1" and the $M + 1$st price auction with "M+1". The $M + 1$st price algorithm measurements were aborted when they started to take too long. For the $M + 1$st price public outcome we have three data points of roughly 15 minutes per bidder which were recorded over night. This data point is not in the graph to keep the scale more detailed but it can be found in Table A.13 in the Appendix.

Figure 6.10: All `libbrandt` Algorithms with 512 Prices.



We can see that to stay under the ten minute mark per bidder which Wassenberg described as "reasonable", `libbrandt` still needs to restrict the number of bidders to quite a low value (six bidders were used as a base case by Wassenberg), especially for the $M + 1$st price auctions. However, Wassenberg assumed a lower security setting (2048 bit RSA keys instead of Ed25519 which is similar to 3072 bit RSA keys) and fewer prices (40 instead of 512), so we could improve security and the price pool drastically and still maintain similar runtimes.

6.4 Bandwidth Usage

Since the size of our elliptic curve group elements is constant we can state exact byte sizes for our protocols, depending only on n and k as described in Table 6.1. For comparison we use the RSA key size of 3072 bits as stated above. Consequently, an RSA implementation of the same protocols would use roughly $3072/256 = 12$ times more bandwidth compared to the Ed25519-based `libbrandt` implementation.

Note, that the table only includes transmitted data relevant to the algorithm. Each message will also have a few additional headers depending on the used network backends. In fact the seller's transmission cost is not zero for public outcome auctions, since he also has to announce the auction somehow and introduce the start of the auction to all bidders. We also only calculated for an optimal broadcasting backend, where the sender only has to send the message once and each participant receives this message only once. In a simple broadcast emulation the sender would have to create separate unicast messages for each receiving host increasing the sending cost of broadcast messages by a factor of $n - 1$.

We also assumed that broadcast messages will be received by the sender as well. This assumption does not change the overall bandwidth complexity of the protocols.

To give a rough notion of where these numbers end up in realistic scenarios, we computed it for an example with 8 bidders, 512 prices and the first price private outcome auction format. Here, each bidder needs to send 1.28MiB, the seller needs to send 3.5MiB, each bidder needs to receive a total of 9.13MiB and the seller needs to receive a total of 10.25MiB. Since receiving data usually is not as limited by the network providers as sending is, the sent number of bytes is probably the bottleneck. We think this amount of bandwidth cost is appropriate, at least for computers. For generally more restricted — in terms of bandwidth and in terms of maximum transfers per month — mobile networks this might still be too much to be attractive to users.

Table 6.1: Bandwidth Cost for `libbrandt` in bytes.

Format	**Round**	**seller rx**	**seller tx**	**bidder rx**	**bidder tx**
First Price Private Outcome	Prologue	$96n$	0	$96n$	96
	Round 1	$320nk + 96n$	0	$320nk + 96n$	$320k + 96$
	Round 2	$160n^2k$	0	$160n^2k$	$160nk$
	Round 3	$128n^2k$	$128(n-1)nk$	$128(n-1)nk$	$128nk$
	Result	$\mathbf{\in O(n^2k)}$	$\mathbf{\in O(n^2k)}$	$\mathbf{\in O(n^2k)}$	$\mathbf{\in O(nk)}$
First Price Public Outcome	Prologue	$96n$	0	$96n$	96
	Round 1	$320nk + 96n$	0	$320nk + 96n$	$320k + 96$
	Round 2	$160nk$	0	$160nk$	$160k$
	Round 3	$128nk$	0	$128nk$	$128k$
	Result	$\mathbf{\in O(nk)}$	**0**	$\mathbf{\in O(nk)}$	$\mathbf{\in O(k)}$
M + 1st Price Private Outcome	Prologue	$96n$	0	$96n$	96
	Round 1	$320n^2k + 192n$	0	$320n^2k + 192n$	$320nk + 192$
	Round 2	$160n^3k$	0	$160n^3k$	$160n^2k$
	Round 3	$128n^3k$	$128(n-1)n^2k$	$128(n-1)n^2k$	$128n^2k$
	Result	$\mathbf{\in O(n^3k)}$	$\mathbf{\in O(n^3k)}$	$\mathbf{\in O(n^3k)}$	$\mathbf{\in O(n^2k)}$
M + 1st Price Public Outcome	Prologue	$96n$	0	$96n$	96
	Round 1	$320nk + 192n$	0	$320nk + 192n$	$320k + 192$
	Round 2	$160n^2k$	0	$160n^2k$	$160nk$
	Round 3	$256n^2k$	0	$256n^2k$	$256nk$
	Result	$\mathbf{\in O(n^2k)}$	**0**	$\mathbf{\in O(n^2k)}$	$\mathbf{\in O(nk)}$

Note, that $k = k_{\text{app}}$ = the real number of prices, not “kilo” or “kibi”

Chapter 7

Discussion and Conclusion

7.1 Improvements

Through translating the algorithm from an RSA to an Ed25519 elliptic curve based crypto system and measuring the performance we have shown that we can reduce computation time to around 7% and message size to around 8% of the RSA version at similar security levels. This is a huge improvement and especially relevant for Brandt's algorithms which are one of the computationally most intensive auction resolution algorithms due to the strong security and privacy goals. Furthermore, this is not only applicable to Brandt's work, but can probably be used on a wide variety of other RSA based algorithms as well to reduce runtimes, power consumption, bandwidth requirements and, depending on the network provider plan, also cost. This improvement is also important for embedded devices with low computation power and mobile devices due to the common monthly traffic limits in mobile data plans. The only disadvantage of Ed25519 over RSA we could find is the slightly longer signature verification time [10] [24], but this only needs consideration in signature heavy protocols, which ours are not.

7.2 Usability

When it comes down to the usability question on the low level of algorithm runtime it is a highly subjective matter. How much time are people generally okay with their CPU computing the outcome of an auction? We have no empirical data on that yet, but argue that the ten minute mark from Wassenberg is a reasonable assumption. Single core CPUs are arguably rare nowadays, so auctions will probably not block the whole system for most users, but merely one of several cores.

Since the computation time remains the major bottleneck, the `libbrandt` implementation would have to be parallelized before it becomes suitable for high-frequency auctions,

as they are being envisioned for smart grids negotiating the price of electricity based on regional supply and demand. We also note that the seller's computations are limited to checking all the ZKPs and outcome determination from all received messages. As a result, the seller's computation is significantly more lightweight.

In conclusion our evaluation results show that the system is usable in real world applications. We could not find another online auction system with the same or even stricter security and privacy properties faster than our implementation. Still, a few additions described in Section 7.4 need to be made before the system should be used for the first real auction and beyond that more performance improvements are needed for auctions with more bidders.

7.3 Open Questions

To optimize runtimes depending on the input parameters of an auction it would be helpful to know what minimum and maximum prices should be used, how many prices are needed, and how to interpolate between the minimum and maximum. A first attempt for the interpolation would be to use an exponential function so prices have a finer granularity on the low end and still reach a reasonably big price pool due to the bigger gaps between two prices near the maximum. However, further studies are required to verify or negate that assumption.

To assess the actual real-world usability, empirical studies are needed to evaluate what computation times are okay for users.

While we have established that an escrow service helps to prevent malicious bidders and DoS attacks, the exact parameters of how this service should operate are not yet fixed. Open questions are who benefits from misbehaving bidders, how the smart contracts should be structured so the escrow service can verify them, and how payment and shipping information is exchanged between the seller and the winners.

An important point for the reputation system is who can actually influence the reputation of the seller. One might argue that losing or misbehaving bidders should not be allowed to do this due to conflict of interest, but on the other hand it might be required to flag a seller who is falsely reporting correct bidders as misbehaving to the escrow service.

In his paper Brandt mentions that the protocol could also be used to emulate incremental auction styles. This might be interesting since those types of auctions seem to be more popular due to their expected higher revenue for sellers.

7.4 Future Work

First of all, as noted in Section 3.3, the GNUnet auction subsystem needs to be completed.

In Section 4.5 we listed a few details which are still missing from `libbrandt`. For real-world usage these need to be addressed and implemented.

To reduce latency, the cryptographic operations in `libbrandt` should be run in parallel.

To reduce the attack surface even further one might want to ensure the implementation is resistant against side-channel attacks. For example constant time computations could be incorporated to achieve this goal.

One can think of a new distributed system which replaces the platform in our proposed architecture and ensures innumerability of auction offers.

To improve fault-tolerance, it would help if `libbrandt` supported having checkpoints where its state would be serialized for backups, and where the process could be resumed after system recovery.

Appendix A

Appendix

A.1 Measurement RSA Parameters

These are the exact RSA parameters we used in our measurements to emulate a similar security level like Ed25519:

p = 449854698218374180604204687492523084136775261010521576894643825547012074019552284920185699717986681512631333975691555816742339833407263977802640190403184401686168296088147345012026525632764131070943783358088625044116465255103165540530132941388525058740857331962113830467809461159843611985403588155547207988936430770198327542749579608223939042630659023963007129330447699318811214529540618550440077037925044823675938805114985619157219947595827496389254903658633237355556162437838532401856364178107372212128292404819407333288538658385328683538489628646848059448985198863513714630405074311940603015045721470311542841502834544543908082490596734776741006509612469115543410609078854149130197151076707267864128631738838288497900835194163473840702042110917641699818136591169734014884729213611401595138283604534231490958695735199141953824592097342969762501656994779480311455139652741493362410339178831303875105158998076241369840028120 3

$q = (p-1)/2$ = 224927349109187090302102343746261542068387630505260788447321912773506037009776142460092849858993340756315666987845777908371169916703631988901320095201592200843084148044073672506013262816382065535471891679044312522058232627551582770265066470694262529370428665981056915233904730579921805992701794077773603994468215385099163771374789804111969521315329511981503564665223849659405607264770309275220038518962522411837969402557492809578609973797913748194627451829316618677778081218919266200928182089053686106064146202409703666644269329192664341769244814323424029724492599431756857315202537155970301507522860735155771420751417272271954041245298367388370503254806234557771705304539427074565098575538353633932064315869419144248950417597081736920351021055458

820849909068295584867007442364606805700797569141802267115745479347867599570976912296048671484881250828497389740155727569826370746681205169589415651937552579499038120684920014060 1

$g = 3$

A.2 Raw Measurement Data

In Tables A.1 to A.13 we present the raw measured runtimes. Each data cell contains either a set of measured values in the 10 or 20 iterations, or the median of the set in the left adjacent cell. This median was then divided by the number of bidders and possibly by 60 to get the result in minutes instead of seconds per bidder. In Table A.9 we left out the computed median so the table fits on one page.

Table A.1: First Price Private Outcome Auction with five Prices (Measured all Bidders).

	libbrandt		Wassenberg		PARI/GP	
n	Execution Times (Seconds)	Median	Execution Times (Seconds)	Median	Execution Times (Seconds)	Median
2	{2.10, 2.10, 2.11, 2.11, 2.12, 2.13, 2.16, 2.18, 2.21, 2.22}	2.13	{30.71, 31.46, 31.76, 32.43, 32.54, 32.63, 32.68, 32.82, 33.26, 34.27}	32.58	{20.55, 20.83, 20.99, 21.00, 21.01, 21.05, 21.06, 21.06, 21.09, 21.12}	21.03
3	{5.25, 5.29, 5.30, 5.42, 5.48, 5.50, 5.65, 5.69, 6.00, 6.12}	5.49	{75.97, 77.85, 78.95, 78.98, 79.00, 79.81, 80.01, 81.04, 87.59, 88.93}	79.40	{51.57, 52.18, 52.19, 52.31, 52.35, 52.38, 52.39, 52.48, 52.74, 52.75}	52.37
4	{10.77, 10.84, 10.85, 10.91, 10.96, 11.13, 11.16, 11.20, 11.30, 11.32}	11.04	{147.95, 151.50, 152.07, 152.43, 152.59, 152.63, 152.71, 153.12, 154.06, 165.82}	152.61	{103.45, 103.65, 104.27, 104.27, 104.27, 104.60, 104.89, 105.46, 107.09, 107.42}	104.44
5	{18.63, 18.77, 19.10, 19.18, 19.28, 19.34, 19.37, 19.37, 19.51, 19.95}	19.31	{252.63, 254.69, 256.33, 259.04, 259.13, 260.82, 262.58, 263.85, 277.20, 279.90}	259.97	{189.52, 190.66, 190.84, 193.83, 198.39, 206.61, 210.01, 214.65, 217.29, 223.71}	202.50
6	{29.69, 29.79, 29.90, 29.97, 30.11, 30.60, 30.70, 30.81, 30.95, 31.00}	30.36	{390.59, 391.69, 396.04, 396.57, 401.98, 404.44, 422.42, 558.15, 562.65, 579.10}	403.21	{287.03, 287.72, 288.12, 288.20, 288.36, 288.56, 289.32, 289.48, 292.29, 301.07}	288.46
7	{44.52, 44.60, 44.74, 44.82, 44.88, 44.94, 45.36, 45.56, 46.01, 46.29}	44.91	{572.61, 577.18, 577.84, 579.36, 579.99, 590.22, 592.71, 613.12, 623.11, 629.85}	585.10	{443.37, 444.83, 446.09, 450.44, 475.10, 480.98, 495.48, 497.33, 511.14, 512.62}	478.04
8	{62.50, 62.73, 62.73, 62.97, 63.00, 63.00, 64.32, 64.66, 64.71, 64.96}	63.00	{794.13, 799.67, 805.89, 816.69, 828.39, 857.08, 863.91, 874.86, 1040.53, 1493.24}	842.73	{632.29, 633.00, 640.71, 640.92, 642.06, 663.77, 669.36, 685.63, 724.19, 730.28}	652.91

Table A.2: First Price Private Outcome Auction with five Bidders (Measured all Bidders).

	libbrandt		Wassenberg		PARI/GP	
k	Execution Times (Seconds)	Median	Execution Times (Seconds)	Median	Execution Times (Seconds)	Median
2	{7.62, 7.71, 7.71, 7.73, 7.86, 7.98, 8.01, 8.01, 8.03, 8.06}	7.92	{108.19, 110.65, 111.37, 112.59, 112.83, 114.30, 114.99, 120.88, 121.13, 164.78}	113.56	{75.39, 75.42, 75.43, 75.62, 75.64, 75.65, 75.78, 75.99, 76.10, 76.14}	75.64
3	{11.30, 11.35, 11.37, 11.41, 11.41, 11.42, 11.77, 11.81, 11.84, 11.85}	11.42	{158.00, 158.99, 159.27, 160.47, 160.71, 160.94, 161.00, 161.21, 170.08, 172.68}	160.82	{109.15, 109.30, 109.80, 110.20, 110.20, 110.23, 110.43, 110.81, 111.45, 111.64}	110.21
4	{14.99, 14.99, 15.02, 15.11, 15.46, 15.53, 15.56, 15.60, 15.62, 15.65}	15.49	{197.76, 205.54, 207.11, 209.08, 212.44, 212.76, 223.01, 226.87, 229.74, 229.90}	212.60	{144.62, 145.63, 146.44, 147.04, 147.92, 148.92, 160.05, 181.50, 210.64, 218.98}	148.42
5	{18.59, 18.71, 18.73, 18.79, 18.80, 18.97, 19.26, 19.36, 19.38, 19.38}	18.89	{260.42, 262.81, 264.05, 266.09, 266.43, 266.98, 279.89, 280.51, 283.54, 285.16}	266.70	{186.29, 186.91, 188.08, 188.20, 188.82, 212.49, 213.75, 219.94, 221.07, 229.61}	200.66
6	{22.20, 22.20, 22.24, 22.32, 22.35, 23.11, 23.11, 23.11, 23.26, 23.28}	22.73	{308.24, 315.83, 317.02, 317.89, 323.16, 334.48, 334.83, 338.96, 346.24, 355.58}	328.82	{180.52, 187.49, 202.52, 206.88, 219.37, 219.38, 219.67, 221.74, 222.86, 222.90}	219.38
7	{25.86, 25.87, 25.90, 25.93, 26.01, 26.05, 26.88, 26.89, 26.92, 27.02}	26.03	{353.31, 366.68, 372.70, 375.01, 382.68, 396.19, 404.69, 410.28, 440.53, 527.18}	389.43	{210.34, 211.61, 211.98, 213.21, 216.15, 252.10, 253.22, 257.31, 258.52, 273.70}	234.12
8	{29.48, 29.59, 29.67, 29.76, 30.47, 30.59, 30.63, 30.67, 30.67, 30.71}	30.53	{413.71, 415.07, 418.83, 420.75, 423.56, 424.32, 426.56, 426.65, 428.27, 474.66}	423.94	{232.25, 233.10, 234.82, 234.87, 235.44, 235.78, 235.83, 239.05, 239.55, 242.09}	235.61

Table A.3: First Price Public Outcome Auction with five Prices (Measured all Bidders).

	libbrandt		Wassenberg	
n	Execution Times (Seconds)	Median	Execution Times (Seconds)	Median
2	{1.56, 1.58, 1.58, 1.58, 1.59, 1.60, 1.60, 1.66, 1.67, 1.68}	1.60	{28.55, 29.43, 29.57, 29.77, 30.03, 30.05, 30.50, 30.54, 30.79, 30.81}	30.04
3	{2.94, 2.99, 3.02, 3.04, 3.04, 3.07, 3.11, 3.17, 3.18, 3.24}	3.06	{57.89, 58.16, 60.54, 62.37, 62.50, 62.97, 63.27, 63.30, 63.53, 64.29}	62.73
4	{4.89, 4.90, 4.92, 4.93, 4.97, 4.98, 5.00, 5.02, 5.07, 5.09}	4.98	{102.65, 107.07, 108.14, 108.41, 108.89, 108.94, 109.50, 109.96, 110.95, 136.54}	108.91
5	{7.09, 7.10, 7.12, 7.15, 7.16, 7.22, 7.33, 7.34, 7.39, 7.44}	7.19	{146.19, 147.08, 147.12, 148.10, 152.54, 155.91, 156.57, 164.71, 166.75, 228.34}	154.22
6	{9.67, 9.75, 9.80, 10.06, 10.06, 10.08, 10.18, 10.33, 10.38, 10.70}	10.07	{200.58, 201.74, 202.46, 204.15, 207.52, 209.70, 214.88, 215.35, 215.36, 217.51}	208.61
7	{12.61, 12.62, 12.75, 12.77, 12.86, 13.08, 13.34, 13.90, 13.99, 14.03}	12.97	{264.91, 270.64, 272.25, 272.38, 274.46, 275.03, 278.36, 281.07, 284.71, 286.38}	274.74
8	{16.98, 17.58, 17.65, 17.66, 17.67, 17.83, 17.89, 17.94, 18.01, 18.36}	17.75	{339.72, 349.62, 350.90, 354.15, 355.48, 363.47, 369.52, 449.05, 457.27, 555.10}	359.47

Table A.4: First Price Public Outcome Auction with five Bidders (Measured all Bidders).

	libbrandt		Wassenberg	
k	Execution Times (Seconds)	Median	Execution Times (Seconds)	Median
2	{3.00, 3.00, 3.01, 3.02, 3.03, 3.05, 3.09, 3.12, 3.14, 3.14}	3.04	{64.33, 64.52, 65.15, 65.98, 67.92, 70.67, 70.69, 92.45, 94.85, 122.73}	69.29
3	{4.30, 4.31, 4.32, 4.32, 4.32, 4.32, 4.33, 4.48, 4.51, 4.51}	4.32	{87.68, 87.90, 88.10, 88.93, 89.72, 91.99, 92.48, 95.45, 96.94, 134.93}	90.85
4	{5.57, 5.58, 5.59, 5.59, 5.61, 5.83, 5.83, 5.84, 5.84, 5.85}	5.72	{113.46, 113.48, 115.40, 115.72, 116.10, 116.50, 117.32, 117.55, 121.15, 123.67}	116.30
5	{6.84, 6.85, 6.85, 6.87, 6.88, 6.90, 7.17, 7.17, 7.18, 7.20}	6.89	{139.12, 139.23, 139.40, 140.43, 141.75, 142.03, 142.09, 142.11, 142.86, 155.06}	141.89
6	{8.13, 8.15, 8.15, 8.16, 8.17, 8.47, 8.49, 8.52, 8.52, 8.52}	8.32	{163.93, 165.50, 165.96, 166.07, 167.03, 168.15, 168.35, 171.12, 180.39, 238.07}	167.59
7	{9.42, 9.43, 9.45, 9.45, 9.48, 9.81, 9.85, 9.87, 9.88, 9.98}	9.65	{185.81, 187.16, 190.35, 190.37, 192.26, 196.63, 201.14, 201.95, 202.30, 204.39}	194.44
8	{10.70, 10.70, 10.71, 10.71, 10.72, 11.19, 11.19, 11.19, 11.19, 11.22}	10.96	{212.90, 215.96, 217.50, 218.27, 219.30, 220.29, 225.43, 228.79, 231.25, 232.28}	219.79

Table A.5: M + 1st Price Private Outcome Auction ($M = 1$) with three Prices (Measured all Bidders).

	libbrandt		Wassenberg	
n	Execution Times (Seconds)	Median	Execution Times (Seconds)	Median
2	{2.88, 2.89, 2.90, 2.95, 3.00, 3.02, 3.03, 3.05, 3.06, 3.06}	3.01	{20.92, 21.14, 21.18, 21.60, 21.72, 21.80, 29.71, 30.02, 30.28, 30.67}	21.76
3	{10.02, 10.07, 10.08, 10.09, 10.45, 10.52, 10.54, 10.54, 10.58, 10.76}	10.48	{51.23, 52.12, 52.29, 77.48, 100.97, 128.47, 128.58, 129.45, 139.26, 141.86}	114.72
4	{25.91, 25.95, 25.99, 26.01, 26.12, 27.08, 27.11, 27.23, 27.30, 27.30}	26.60	{151.98, 154.87, 164.59, 203.87, 250.24, 262.77, 263.27, 280.30, 426.76, 619.02}	256.50
5	{55.87, 56.02, 56.12, 56.25, 56.34, 57.36, 57.38, 57.39, 57.66, 57.90}	56.85	{179.09, 261.64, 263.78, 263.81, 263.95, 264.51, 280.80, 285.37, 488.36, 1050.20}	264.23
6	{105.90, 106.00, 106.34, 106.45, 108.77, 109.28, 109.36, 109.47, 109.63, 109.64}	109.02	{402.25, 422.71, 428.50, 428.64, 723.11, 729.18, 731.02, 754.95, 1111.03, 1160.96}	726.14

Table A.6: M + 1st Price Private Outcome Auction ($M = 1$) with three Bidders (Measured all Bidders).

	libbrandt		Wassenberg	
k	Execution Times (Seconds)	Median	Execution Times (Seconds)	Median
2	{6.72, 6.72, 6.91, 7.00, 7.05, 7.13, 7.14, 7.19, 7.20, 7.52}	7.09	{53.84, 53.96, 54.11, 56.44, 65.35, 89.32, 91.95, 92.92, 93.06, 99.93}	77.33
3	{10.50, 10.55, 10.55, 10.78, 10.84, 10.85, 10.88, 10.92, 10.94, 10.96}	10.85	{50.46, 52.74, 79.12, 79.75, 80.42, 104.77, 127.38, 128.86, 132.47, 132.68}	92.59
4	{13.94, 14.00, 14.01, 14.17, 14.19, 14.20, 14.42, 14.44, 14.46, 14.47}	14.20	{64.94, 66.53, 66.93, 67.05, 67.78, 100.08, 100.43, 133.95, 168.40, 171.51}	83.93
5	{17.33, 17.33, 17.41, 17.62, 17.86, 17.94, 18.02, 18.04, 18.17, 18.23}	17.90	{81.16, 82.44, 123.15, 127.77, 134.49, 168.73, 201.46, 210.19, 210.52, 213.52}	151.61
6	{20.63, 20.72, 20.74, 20.85, 20.93, 21.19, 21.48, 21.61, 21.63, 21.69}	21.06	{92.02, 92.89, 94.41, 145.30, 151.06, 159.35, 191.18, 245.95, 247.37, 255.63}	155.20

Table A.7: M + 1st Price Public Outcome Auction ($M = 1$) with five Prices (Measured all Bidders).

	libbrandt	
n	Execution Times (Seconds)	Median
2	{3.53, 3.54, 3.54, 3.54, 3.56, 3.56, 3.56, 3.56, 3.67, 3.73}	3.56
3	{9.80, 10.25, 10.27, 10.32, 10.43, 10.44, 10.71, 10.76, 10.93, 12.05}	10.44
4	{21.58, 21.82, 21.90, 22.28, 22.33, 22.33, 22.41, 22.51, 22.84, 23.98}	22.33
5	{40.20, 40.22, 40.32, 40.40, 40.55, 41.13, 41.18, 41.32, 41.49, 41.60}	40.84
6	{66.52, 67.04, 67.27, 67.48, 67.80, 67.98, 68.32, 68.84, 68.96, 69.32}	67.89
7	{102.17, 104.50, 104.59, 105.82, 107.03, 108.60, 109.42, 116.51, 116.67, 121.62}	107.82
8	{145.73, 145.81, 147.24, 147.94, 148.16, 148.18, 149.19, 149.23, 150.52, 151.03}	148.17

Table A.8: M + 1st Price Public Outcome Auction ($M = 1$) with five Bidders (Measured all Bidders).

	libbrandt	
k	Execution Times (Seconds)	Median
2	{16.44, 16.45, 16.48, 16.64, 16.76, 17.17, 17.24, 17.27, 17.31, 17.40}	16.96
3	{24.33, 24.53, 24.53, 25.17, 25.23, 25.24, 25.32, 25.37, 25.39, 25.63}	25.24
4	{32.24, 32.32, 32.40, 32.74, 33.19, 33.40, 33.52, 33.61, 33.64, 33.84}	33.30
5	{40.29, 40.30, 40.47, 40.54, 40.57, 40.62, 41.68, 41.87, 41.91, 41.99}	40.59
6	{48.25, 48.38, 48.45, 48.45, 48.66, 48.92, 49.87, 50.13, 50.15, 50.26}	48.79
7	{56.31, 56.38, 56.42, 56.94, 56.96, 57.77, 58.21, 58.32, 58.44, 58.48}	57.37
8	{65.25, 65.39, 65.52, 65.55, 65.79, 67.29, 68.21, 71.67, 73.36, 75.73}	66.54

Table A.9: M + 1st Price Private Outcome Auction with $n = k = M + 2$ (Measured all Bidders).

M	libbrandt Execution Times (Seconds)	Wa (best) Execution Times (Seconds)	Wa (avg) Execution Times (Seconds)	Wa (worst) Execution Times (Seconds)
1	{8.74, 8.85, 8.87, 9.15, 9.31, 9.33, 9.60, 9.60, 9.67, 9.75}	{50.82, 51.60, 52.22, 52.40, 52.55, 52.71, 53.43, 53.43, 54.25, 59.99}	{52.57, 56.43, 56.64, 65.01, 78.79, 78.96, 79.32, 80.58, 80.63, 80.66, 82.25, 83.09, 83.91, 84.17, 85.19, 85.27, 105.38, 111.72, 112.33, 132.48}	{126.17, 127.55, 130.92, 134.27, 134.57, 138.51, 143.73, 148.04, 150.08, 150.26}
2	{33.54, 33.89, 34.44, 36.16, 36.69, 37.13, 38.33, 38.48, 38.70, 38.82}	{128.72, 129.35, 130.29, 130.73, 131.23, 133.54, 140.45, 140.86, 141.71, 143.12}	{127.41, 129.30, 129.32, 130.99, 136.21, 137.66, 139.32, 153.56, 215.34, 287.40, 298.05, 302.67, 335.21, 435.22, 451.33, 452.77, 566.53, 571.09, 606.47, 679.06}	{564.21, 570.94, 575.74, 576.26, 578.03, 590.69, 607.10, 686.55, 717.59, 730.02}
3	{95.86, 96.05, 97.44, 97.48, 98.24, 98.82, 102.29, 111.31, 116.50, 125.91}	{269.24, 269.90, 271.38, 273.84, 276.56, 280.62, 289.58, 297.59, 356.80, 371.58}	{245.14, 254.40, 259.06, 268.74, 280.31, 286.06, 444.96, 624.46, 709.94, 744.45, 805.73, 834.07, 980.73, 987.08, 1220.87, 1329.32, 1353.27, 1388.78, 1428.26, 1553.45}	{2140.38, 2176.76, 2290.58, 2314.00, 2356.31, 2396.44, 2897.05, 2898.48, 2945.08, 3025.08}
4	{217.94, 219.82, 233.82, 239.43, 239.54, 242.02, 242.58, 243.84, 245.18, 248.05}	{498.56, 504.58, 506.17, 533.73, 537.20, 544.51, 545.36, 546.41, 573.17, 721.63}	{465.96, 475.45, 478.51, 481.70, 485.08, 515.86, 525.75, 651.01, 782.46, 807.27, 810.27, 819.16, 1175.43, 1707.49, 1756.28, 1765.35, 2918.75, 3027.46, 3110.23}	no data
5	{451.05, 485.16, 485.25, 498.33, 501.63, 504.30, 514.36, 542.86, 555.84, 599.79}	{839.05, 839.45, 851.15, 856.92, 860.68, 861.14, 871.46, 892.49, 910.98, 1205.77}	no data	no data

Table A.10: First Price Private Outcome Auction with 512 Prices (Measured all Bidders).

	libbrandt	
n	Execution Times (Seconds)	Median
2	{215.94, 228.17, 233.03, 233.17, 233.48, 234.29, 237.61, 239.33, 240.06, 240.12}	233.88
3	{557.91, 558.04, 558.34, 558.80, 559.01, 560.10, 560.50, 567.42, 570.46, 570.58}	559.55
4	{1087.66, 1089.54, 1090.29, 1091.60, 1102.57, 1102.60, 1104.84, 1105.92, 1105.99, 1106.66}	1102.59
5	{1870.28, 1880.51, 1902.73, 1927.67, 1968.16, 1979.24, 2024.20, 2029.40, 2030.64, 2073.22}	1973.70
6	{2953.78, 2956.16, 2963.98, 2973.45, 2983.09, 2983.62, 2984.81, 2985.89, 2991.00, 3015.86}	2983.36

Table A.11: First Price Public Outcome Auction with 512 Prices (Measured all Bidders).

	libbrandt	
n	Execution Times (Seconds)	Median
2	{157.44, 158.71, 168.74, 169.45, 170.06, 171.08, 172.14, 172.42, 173.12, 176.74}	170.57
3	{305.24, 308.42, 312.78, 313.24, 320.20, 323.68, 323.73, 325.36, 326.74, 354.93}	321.94
4	{465.77, 466.98, 469.21, 477.17, 477.64, 478.00, 481.61, 482.31, 487.74, 549.96}	477.82
5	{675.01, 680.92, 681.24, 683.77, 684.08, 687.85, 691.59, 710.97, 734.82, 846.41}	685.96
6	{889.24, 891.89, 892.07, 899.78, 899.80, 902.63, 904.92, 920.09, 930.63, 973.04}	901.21
7	{1176.07, 1179.57, 1183.43, 1188.31, 1206.44, 1229.15, 1251.01, 1261.05, 1264.63, 1279.35}	1217.80
8	{1508.46, 1525.78, 1564.39, 1578.53, 1629.55, 1647.05, 1686.93, 1694.76, 1696.90, 1697.83}	1638.30

Table A.12: M + 1st Price Private Outcome Auction with 512 Prices (Measured all Bidders).

	libbrandt	
n	Execution Times (Seconds)	Median
2	{476.71, 509.74, 517.95, 518.56, 527.16, 527.89, 530.13, 534.13, 536.84, 537.20}	527.52
3	{1701.07, 1717.45, 1746.21, 1761.33, 1777.29, 1796.31, 1857.54, 1870.85, 1896.55, 1911.35}	1786.80

Table A.13: M + 1st Price Public Outcome Auction with 512 Prices (Measured all Bidders).

	libbrandt	
n	Execution Times (Seconds)	Median
2	{478.35, 484.77, 486.30, 487.47, 490.08, 494.80, 495.79, 503.96, 513.45, 802.28}	492.44
3	{1491.13, 1498.61, 1499.66, 1504.81, 1509.12, 1515.15, 1519.28, 1520.02, 1521.94, 1526.82}	1512.13
4	{3644.23, 3649.50, 3693.75, no more data }	3649.50

Bibliography

[1] R. Bapna, P. Goes, and A. Gupta, "Insights and Analyses of Online Auctions," *Commun. ACM*, vol. 44, no. 11, pp. 42–50, Nov. 2001.

[2] F. Brandt, "How to Obtain Full Privacy in Auctions," in *International Journal of Information Security 5(4)*, 2006, pp. 201–216.

[3] V. Krishna, *Auction Theory*. Academic Press, 2002.

[4] P. Wassenberg, "Implementation and Evaluation of Cryptographic Auction Protocols," Diploma Thesis, LMU, October 2008.

[5] J. Dreier, J.-G. Dumas, and P. Lafourcade, "Brandt's Fully Private Auction Protocol Revisited," in *Progress in Cryptology – AFRICACRYPT 2013: 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 88–106.

[6] T. ElGamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 10–18.

[7] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[8] R. Barbulescu, P. Gaudry, and T. Kleinjung, *The Tower Number Field Sieve*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 31–55.

[9] V. S. Miller, *Use of Elliptic Curves in Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426.

[10] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[11] A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Proceedings on Advances in cryptology—CRYPTO '86*. London, UK, UK: Springer-Verlag, 1987, pp. 186–194.

[12] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869 (Informational), Internet Engineering Task Force, May 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5869.txt

[13] M. Lipow, "Number of Faults per Line of Code," *IEEE Transactions on Software Engineering*, vol. SE-8, no. 4, pp. 437–439, July 1982.

[14] F. Brandt, "Fully Private Auctions in a Constant Number of Rounds," in *Financial Cryptography: 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003. Revised Papers.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 223–238.

[15] J. Burdges, F. Dold, C. Grothoff, and M. Stanisci, "Enabling Secure Web Payments with GNU Taler," in *International Conference on Security, Privacy, and Applied Cryptography Engineering.* Springer, 2016, pp. 251–270.

[16] A. Rampuria, A. Kulshrestha, A. Sreenivas, and M. Denton, "Cryptographically Secure Multiparty Computation and Distributed Auctions using Homomorphic Encryption," 2016.

[17] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, and R. Nicholas, "Internet X.509 Public Key Infrastructure: Certification Path Building," RFC 4158 (Informational), Internet Engineering Task Force, Sep. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4158.txt

[18] H. Lipmaa, N. Asokan, and V. Niemi, "Secure Vickrey Auctions without Threshold Trust," Cryptology ePrint Archive, Report 2001/095, 2001. [Online]. Available: http://eprint.iacr.org/2001/095

[19] M. Hinkelmann, A. Jakoby, and P. Stechert, "t-Private and t-Secure Auctions," *Journal of Computer Science and Technology*, vol. 23, no. 5, pp. 694–710, 2008.

[20] M. Ibrahim, "A Novel Approach to Fully Private and Secure Auction: A Sealed-Bid Knapsack Auction," *International Journal of Research and Reviews in Applied Science*, vol. 9, no. 2, pp. 260–269, 2011. [Online]. Available: http://works.bepress.com/maged-hamada-ibrahim/13/

[21] "ssh-keygen(1) Linux User's Manual," openssh version 7.3, February 2016.

[22] "time(1) Linux User's Manual," GNU time version 1.7, February 2016.

[23] "cgroups(7) Linux Programmer's Manual," Linux kernel version 4.9.9, February 2016.

[24] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "Making the Case for Elliptic Curves in DNSSEC," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 13–19, Sep. 2015.