Privacy Preserving Auctions and Mechanism Design*

Moni Naor Benny Pinkas Reuben Sumner

Abstract

We suggest an architecture for executing protocols for auctions and, more generally, mechanism design. Our goal is to preserve the privacy of the inputs of the participants (so that no nonessential information about them is divulged, even a posteriori) while maintaining communication and computational efficiency. We achieve this goal by adding another party - the auction issuer - that generates the programs for computing the auctions but does not take an active part in the protocol. The auction issuer is not a trusted party, but is assumed not to collude with the auctioneer. In the case of auctions, barring collusion between the auctioneer and the auction issuer, neither party gains any information about the bids, even after the auction is over. Moreover, bidders can verify that the auction was performed correctly. The protocols do not require any communication between the bidders and the auction issuer and the computational efficiency is very reasonable. This architecture can be used to implement any mechanism design where the important factor is the complexity of the decision procedure.

1 Introduction

Imagine participating in an online auction run by the auctioneer eSleaze.com. The auction is a sealed bid second price auction (also known as Vickrey auction [36]). In this type of auction the highest bidder wins, and the clearing price, the price that the winner has to pay, is equal to the second highest bid. An important property of second price auctions is that the optimal strategy of bidders is simply to bid their true valuation of the goods for sale. That being the case, assume that you value the goods at $1000, and this is the bid you submit. At the end of the auction, eSleaze.com congratulates you on winning, and announces that the second highest bid was $999... Would you be convinced that eSleaze.com did not manipulate the second highest bid to maximize profits?

It would seem that the above problem could be solved if the value of the bids could be hidden *until bidding closes*, thus preventing a corrupt auctioneer from manipulating auction results. However, consider the following scenario: you bid $1000 and the second highest bid is only $600. As the auctioneer does not place fake bids, you win and are required to pay $600. The following day eSleaze.com put a second unit of the same product up for auction, and you're again interested. This time, however, it sets a reservation price of $999, meaning that the clearing price would be the maximum of the second highest price and $999. (Alternatively you might find that this time the second highest bid is $999.) You might suspect that eSleaze.com learned from your previous bid that you highly value the product, and set the reservation price accordingly.

Your concern might be justified. It is common to use sealed-bid second-price auctions as a replacement for the open-cry English auction. In English auctions bidding is interactive and the winner is the bidder who outbids all other bidders. Winners are required to pay their last and highest bid, which is essentially only slightly higher than the final bid of the second highest bidder. The sealed-bid second-price auction requires less interaction and is therefore easier to run, but its main disadvantage is that unlike the English auction, the bid (i.e. the valuation) of the highest bidder is revealed to the auctioneer [11]. A corrupt auctioneer may take advantage of this information, either in future auctions or by reneging on the sale. (In fact, a corrupt auctioneer can also take advantage of learning the bids of the other (non-winning) bidders. The scheme that we introduce hides even that information from the auctioneer).

This problem was stated by Varian [35] as follows: *"Even if current information can be safeguarded, records of past behavior can be extremely valuable, since historical data can be used to estimate the willingness to pay. What should be the appropriate technological and social safeguards to deal with this problem?"*

This paper aims to provide a solution to this problem. We present results which are of interest from different aspects:

Game Theory: The analysis of auctions and of mechanism design almost always assumes the trustworthiness of the auctioneer (or of the center which computes the output of the mechanism); this assumption, unfortunately, might not be justified in real life. Most auctions currently run on the Internet are open cry auctions [6]. One reason for this phenomenon is the mistrust of auctioneers that handle sealed bid auctions. We introduce a simple architecture which ensures that the auctioneer never gains access to more information than a legitimate and honest auctioneer. This architecture justifies the assumption of a trustworthy auctioneer.

Cryptography: Although how to securely compute any function is known in principle [18, 40], it was believed that these results were theoretical and inefficient in practice. We present an efficient example of a secure evaluation protocol for very useful applications, namely for auctions and mechanism design. The protocol requires very little interaction. We are currently experimenting with the implementation of this protocol for auctions with hundreds of bidders.

*The authors are with the Dept. of Computer Science and Applied Math, Weizmann Institute of Science, Rehovot 76100, Israel. Email: {naor,bennyp,rasumner}@wisdom.weizmann.ac.il.

Business: The market for online auctions and procurement is large. There are many security and privacy issues that make users suspicious of online auctions, in particular for business-to-business applications involving large sums of money. Our protocols use an established auction issuer which generates 'programs' that compute the auctions but is not required to take an active part in the auction itself. Bidders are assured that as long as the auction issuer and the auctioneer do not collude then the auction is computed correctly. There seems to be a very promising business opportunity for implementing and running the auction issuer. Auctions run by an auctioneer and backed by the auction issuer can be trusted even if the auctioneer is of a somewhat dubious character (say, from an unfamiliar country). The use of new protocols might, therefore, develop new markets for online auctions.

The rest of this section discusses trust issues, presents the suggested architecture, and reviews related work. Section 2 discusses the protocol for computing auctions, while Section 3 suggests its use in general mechanism design. Further work and open questions are discussed in Section 4.

1.1 Managing Trust

We aim to minimize the level of trust that bidders must place in the auctioneer both in sealed bid auctions and in more general mechanisms. This goal is especially important in online auctions where long-term relationships between bidders and auctioneers often do not exist, and where auctions may be run by many small scale parties.

Mechanism design deals with the design of protocols for selfish parties. The goal of a protocol is to aggregate the preferences of the parties to reach some "social choice" (e.g. decide whether a community should build a bridge, how to route packets in a network, or who wins an auction). Each party has a utility function expressing its valuation of each possible outcome of the protocol. The party sends information about its utility function to a center (of course, it might choose to report according to an untrue utility function, if it believes benefit will be derived). The center determines the outcome of the protocol based on the reports received. The goal is to develop mechanism designs in which parties have no incentive to report a false utility function. The *Revelation Principle* states that for any mechanism there is a direct, incentive-compatible mechanism with the same result (see [31] Chapter 10). That is to say, there is an equivalent mechanism in which the optimal strategy for each party is to *report its true utility function.* It is often assumed that the parties can trust the center , however this may not always be the case, particularly in an Internet setting. The revelation principle might not be applicable if the center is corrupt and misuses the truthful bids it receives. Privacy is therefore essential to ensure the credibility of the center.

The method we present controls the amount of information revealed to the auctioneer, and, in fact, offers bidders *more privacy than in the physical realm*, such as in sealed bid auctions using (physical) envelopes, paper, etc. With our scheme the outcome of the auction will be the only information that the auctioneer gains. For example, in the case of second price auctions, the auctioneer should learn the identity of the highest bidder (but not the bid!) and the clearing price which is the second highest bid. The auctioneer (and all other parties) should not learn the identity of the second highest bidder or any information about the other bids. Still, all parties should be able to verify that the auction was run properly, that the highest bidder won, and that the clearing price is correct.

Auctioneers can use the bids to gather useful and legitimate information. There might be tension between bidders who prefer that auctioneers not gain *any* information other than the final outcome of the auction, and auctioneers who strive to gather as much information as possible. The architecture we present enables precise control of the information the auctioneer receives. For example, the auctioneer might be allowed to learn some aggregate statistics of the bids (e.g., the average bid, the number of bids in a certain range), but not the identity of the bidders associated with a given bid. All these variants can be easily incorporated into our schemes.

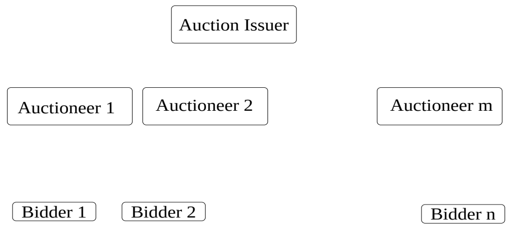

Figure 1: The different entities: A single auction issuer, multiple auctioneers, and numerous bidders.

1.2 Architecture and Entities

The architecture we present introduces a new entity – the *Auction Issuer* (AI). This entity runs in the background and ensures that the auctions are executed properly. The architecture contains the following types of entities, which are depicted in Figure 1:

Bidders In the simplest case one or several bidders wish to sell items, and the remaining bidders are interested in buying the items. In the general case the bidders are parties who should allocate some resources using a predefined mechanism. The bidders send a message describing (in an "encrypted" manner) their utility function, to the auctioneer, and at the end of the protocol they receive an allocation and can verify that it was computed correctly.

Auctioneer The auctioneer runs the show: it advertises the auction, receives the bids from the bidders, communicates with the auction issuer and computes the output of the protocol. The auctioneer might be a party that merely organizes the auction or the mechanism. It can also be one of the bidders (for example selling an item which all other bidders are interested in buying). The protocol ensures that the auctioneer cannot uncover any information about the bids that it receives, except for computing the desired outcome of the protocol (unless it colludes with the auction issuer).

Auction issuer The auction issuer (AI) is responsible for "coding the program" that computes the output of the protocol so as to preserve privacy, and supplying this program to the auctioneer. Preparation of the program can be completed ahead of time and is not dependent on the identities of the auctioneer or the bidders. The AI is not required to interact with bidders, but only performs a single, one-round interaction with the auctioneer after the auctioneer receives the bids. The AI is, therefore, a service provider that can provide programs for many auctions carried out by many auctioneers.

The scheme is very efficient. The communication pattern used is identical to that of an insecure auction – each bidder is only required to send a single message to the auctioneer. The running time of the procedure which determines the outcome of the auction is just a few seconds on a modern computer.

In the remainder of the paper we concentrate on a solution for second price sealed bid auctions. This solution, however, can be used to privately implement other mechanisms such as first price or kth price auctions, auctions with reservation prices, double auctions, Generalized Vickrey auctions, Groves-Clarke mechanisms, etc. This is demonstrated in Section 3. Furthermore, it is possible to use this architecture for tasks such as stable matching (e.g. for residents and hospitals [20, 33]), or decision making.

1.3 Trust

In our architecture the auction is secure providing the auctioneer and the auction issuer do not collude. Note that the AI is not required to be a trusted third party, but rather security is guaranteed as long as the auctioneer and the AI do not collude. That is, the architecture ensures that neither the auctioneer nor the AI can uncover alone any information about the bids.

More precisely, consider an *ideal model* where there is a special party which is fully trusted by all other parties. An auction in this model can be conducted in a trivial manner : all parties submit bids to the trusted party who then computes and outputs the results. Note that even in this model some information is leaked about the bids[1], but this is inevitable.

The protocol we suggest ensures that *no party gains more information than in the ideal model.* Likewise the auctioneer and auction issuer are also blocked from learning more than in the ideal model[2].

Only an act of collusion by both the auctioneer and auction issuer enables the privacy of the bidders to be breached. Therefore, bidders need only trust that the AI and the auctioneer are not in collusion, and are assured that neither the AI nor the auctioneer alone learn anything more than in the ideal model. In a sense, the bids are locked in a double lock vault, where one key is in the hands of the auctioneer and the other in the hands of the AI.

More generally, a coalition of at most one of the the auctioneer or auction issuer with several bidders is at most as powerful as in the ideal model, for the bids submitted by other bidders cannot be learned. Consequently, the extent to which the bidders should trust the auctioneer or the auction issuer is less than that they usually put in their banks, credit card companies, or software vendors. The auctioneer can be any party wanting to organize an auction, while the auction issuer is typically an established party such as a financial institution or large company, which supplies services to numerous auctioneers. The auction issuer does not need to communicate directly with bidders and does not even need to know their identities (thus eliminating the danger of it "stealing" customers from the auctioneers). Bidders are assured that their privacy is preserved provided they participate in an auction in which the auctioneer uses the AI's "programs".

[1]There are many possible variations regarding the information learned by different parties. For example, the highest bidder and the clearing price might be publicly announced. On the other extreme this information might be revealed only to the highest bidder and to the seller. Even here, winner learn that they met the winning criteria, which itself reveals limited information about the other bids.

[2]The protocol differs from one in the ideal model in that it enables the auction issuer to learn the *number* of bidders. By placing an upper bound on the number of bidders, it is simple to prevent this information from being revealed to the auction issuer.

Another appealing property of our schemes is that they prevent disputes regarding the operation of the auctioneer. At the end of the protocol, all parties can verify that the auctioneer computed the desired auction or mechanism correctly.

We assume throughout the paper that the Public Key of the auction issuer is known to the bidders. Beyond this, we do not require any Public Key Infrastructure (PKI).

1.4 Related work

Internet auctions are the topic of major commercial and research efforts. Kumar and Feldman [25] describe several issues concerning Internet auctions as well as an application for auctioning goods over the Internet. Chui and Zwick [6] present a thorough survey of commercial Internet auctions. There are also several academic auction servers on the Internet, which enable experimentation with more complex auctions than those offered by commercial sites. For example, the AuctioBot server of the University of Michigan [1, 38] supports Mth and $(M+1)$th price double auctions with multiple sellers. A recent design of an academic auction server, eAuctionHouse [22], supports combinatorial auctions and bidding by automated agents.

An exciting topic of cryptographic research is *secure function evaluation* (see e.g. [18, 40] and [16] for an up-to-date and erudite discussion). For any function $f(x_1, x_2, \ldots, x_n)$ it is possible *in principle* to construct a protocol that allows a group of n parties, where party i has as its private input α_i, to jointly evaluate $f(\alpha_1, \alpha_2, \ldots, \alpha_n)$. Following the protocol, the parties learn $f(\alpha_1, \alpha_2, \ldots, \alpha_n)$ but no party i can learn about the other inputs $\{\alpha_j\}_{j\neq i}$ more than can be computed from α_i and $f(\alpha_1, \alpha_2, \ldots \alpha_n)$. Since an auction can be considered an evaluation of a function of the bids, it is tempting to try to use such a protocol to conduct an auction. The drawback, however, is that these protocols are rather complex and require significant interaction between the parties. They are secure as long as less than a certain number of the parties collude maliciously. (Such protocols are the basis for the auction protocols of [21], see below.)

There are suggestions for distributing the operation of an auctioneer among multiple servers in a manner that is secure as long as not too many of these servers operate maliciously. Franklin and Reiter [14] developed a distributed system for sealed-bid auctions which ensures the confidentiality of the bids until end of the bidding period. Their system further enables the bids to be backed by escrowing financial commitments of the bidders. Harkavy, Tygar, and Kikuchi [21] present systems for secure first price and second price sealed bid auctions that preserve the privacy of the bids even after the winning bid is chosen (this variant was also briefly mentioned in [14], Section 5.2.5). Both systems distribute the operation of the auctioneer among several servers, and privacy is guaranteed as long as not too many of the servers collude (most of the protocols require that less than a third of the servers collude, and therefore need a minimum of four servers). A different auction scheme was suggested very recently by Cachin [5], involving two auction servers, but requiring users to contact just a single server. After receiving the bids, the auction servers engage in several rounds of communication, at the end of which they have a list of the bidders sorted by their bids, but not the bids themselves.

The systems of [14, 21] require bidders to communicate directly with all the servers. Furthermore, the systems of [21, 5] require high interactivity between the servers which exchange numerous *rounds* of interaction. These requirements impose bandwidth and latency problems on all the auction servers. There is no motivation for a global party such as the auction issuer to participate as a server in many auctions, as a considerable amount of resources must be invested in each auction. The implication might be that all auction servers would essentially be managed by the same organization – the auctioneer. Note the distinction between this architecture and the architecture that we suggest. In the former, privacy is guaranteed only if the auctioneer is trusted not to combine the information held by the different servers it controls. This assumption is not enforceable and cannot be verified by an outsider. Consequently, the only protection is against *external* break-ins to the auctioneer's servers (under the presumption that threshold of servers break-ins is not exceeded). This architecture therefore requires complete trust in the auctioneer, which, in the case of small Internet auctioneers, might not justified (see [25]).

Compared to these solutions our architecture provides bid privacy after the auction is over and does not require distribution of the auctioneer between several noncollusive servers. It uses a single round of communication between the auctioneer and auction issuer and can, therefore, use a *separate organization* as the auction issuer. In addition, our scheme can be used to implement general mechanisms.

The work of [34] considers online English auctions with a single auctioneer (which might be corrupt) and secures them against selective blocking of bids based on their amount, and selective early termination of the auction. Our schemes obtain similar properties for sealed bid auctions.

Our work employs several techniques developed for two-party secure function evaluation and obtains communication efficient *multi-party* protocols for computing auctions. In particular, we employ the garbled circuit technique attributed to Yao [40, 18]. We are able to obtain rather efficient protocols, as we assign different roles to the players, i.e., the bidders, the auctioneer, and the auction issuer. (See a similar phenomenon in [13].)

2 The Protocol

In Section 2.1 we provide a high-level description of the protocol for running secure auctions. The protocol rests on a number of cryptographic tools described in Sections 2.2 and 2.3. Section 2.4 describes the complete protocol, Section 2.5 discusses its overhead and possible optimizations, and Section 2.6 describes a prototype implementation.

2.1 High-level Description of the Protocol

The protocol comprises the following steps (as depicted in Figure 2):

1. The auctioneer publishes the details of the auction it is organizing. These should include the rules for selection of winning bids are chosen, closing time and auction issuer (AI) supporting the auction.

2. Bidders submit encrypted bids to the auctioneer. (The AI can decrypt part of the encryption, but even it cannot discover the actual bids).

3. The AI generates a program to compute the output of the auction. More precisely, it generates a circuit (composed of Boolean gates such as AND, OR and NOT) that performs this task and then "garbles" the circuit (this step can be performed in advance, prior to the submission of the bids). The auctioneer forwards portions of the bids to the AI, which decrypts the bids and uses them to compute "garbled inputs" to the circuit. It sends the circuit and the inputs to the auctioneer, along with a signed translation table that "decrypts" the output of the circuit (alternatively, the AI can send the garbled program, which is the bulk of the communication, in advance).

4. The auctioneer uses the garbled inputs and the encrypted circuit to compute the output of the circuit. It publishes the result and the signed translation table received from the AI.

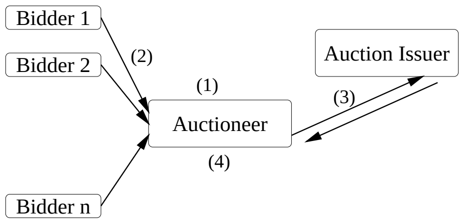

Figure 2: High-level description of the protocol with a single auction issuer: (1) The auctioneer publishes the auction. (2) Bidders place their bids. (3) The auctioneer sends a message to the auction issuer, and receives garbled circuit inputs. (4) The auctioneer computes the result of the auction.

Most previous designs of multi-party protocols (e.g. [18, 16]) require all parties to interact with each other, and furthermore to exchange many rounds of communication. The novelty of our approach is the design of a secure multi-party protocol which preserves the communication pattern of an auction protocol, in which a bidder is only required to exchange a limited number of messages with a single auctioneer. Notice that the only new communication channel required by our protocol (compared to a protocol with no security at all) is a single back and forth communication round between the auctioneer and AI after bids are received.

We examine the overhead of the protocol in Section 2.5. Circuits which compute the output of auctions are of reasonable size. For example, if there are N bidders and bids are in a range of $L = 2^\ell$ possible values, then the number of gates in a circuit which computes a second price auction is $O(N \log L) = O(N\ell)$ with a small constant (see details in Section 2.5). The AI and auctioneer should perform several applications of a pseudo-random function per gate and several applications of a public key operation per input wire. On a modern processor it is possible to perform hundreds of thousands of applications of a pseudo-random function and dozens of public-key operations in a single second. Thus, the computational overhead of the protocol is of the order of less than one second per bidder.

2.2 Cryptographic Tools

The protocol uses two types of cryptographic tools: pseudo-random functions and oblivious transfer.

2.2.1 Pseudo-random functions

A pseudo-random function is a function that cannot be distinguished from a truly random one by an observer granted access to the function in a black-box manner. Assume, for

OT	Sender	Chooser
Input	m_0, m_1	$\sigma \in \{0,1\}$
Output	—	m_σ

Table 1: 1-out-of-2 Oblivious Transfer

example, a function F_K, specified by a short key K which can only be accessed by the observer by adaptively specifying inputs and obtaining the value of the function on these inputs. (See [17, 26] for precise definition and various constructions). Our working assumption is that block ciphers (such as DES, or triple DES) or *keyed* one-way hash functions (such as HMAC), can be modeled as a pseudo-random function. Therefore, the function $F_K(x)$ can be implemented by keying a block cipher with the key K and encrypting x, or keying a hash function with K and applying it to x. The evaluation of a pseudo-random function is therefore considerably cheaper than a typical public-key operation.

2.2.2 Oblivious Transfer and Proxy-Oblivious Transfer

The following two-party protocol is known as *1-out-of-2 oblivious transfer* (1-out-of-2 OT). The protocol involves two parties, a *sender* that knows two secret values $\langle m_0, m_1\rangle$, and a *chooser* whose input is $\sigma \in \{0,1\}$. At the end of the protocol, the chooser learns m_σ, while learning nothing about $m_{1-\sigma}$, and the sender learns nothing about σ. This is summarized in Table 1.

The notion of 1-out-2 oblivious transfer was suggested by Even, Goldreich and Lempel [12] as a generalization of Rabin's "oblivious transfer" [32]. For an up-to-date discussion of OT, see Goldreich [16].

Oblivious transfer protocols are rather efficient, and the noninteractive OT protocols of Bellare and Micali [4] are particularly attractive. The combination of their protocols with the proof techniques of [8] yields an efficient 1-out-of-2 OT protocol that is based on the Decision Diffie-Hellman assumption. We describe this protocol in Appendix A. The main computational overhead of the protocol is two public key encryptions conducted by the sender, and one public key decryption conducted by the chooser[3].

Proxy Oblivious Transfer: We extend the notion of OT to *1-out-of-2 proxy oblivious transfer*. This protocol involves *three* parties: A *sender* (that knows two inputs m_0 and m_1), and a *chooser* (with an input $\sigma \in \{0,1\}$), as well as a a third party, the *proxy*, which has no input and serves as the chooser's proxy for learning the output. At the end of the protocol, the proxy learns m_σ, while the two other parties learn nothing. Note that the proxy does not learn σ. The definition is summarized in Table 2. In Appendix A we describe a protocol that implements 1-out-of-2 proxy OT, with an overhead identical to the protocol for 1-out-of-2 OT.

In our protocol for computing auctions, the auction issuer is the sender, the bidders are the choosers, and the auctioneer is the proxy.

[3]This overhead is achieved if the protocol uses a *random oracle* function H, i.e., a concrete function assumed to behave as a random function. If it is not assumed that such functions exist, then the protocol should use the proof techniques of [8] which require several additional exponentiations.

PROXY-OT	Sender	Chooser	Proxy
Input	m_0, m_1	$\sigma \in \{0,1\}$	—
Output	—	—	m_σ

Table 2: 1-out-of-2 Proxy-Oblivious Transfer

2.3 Secure Function Evaluation for Two Parties

We describe a secure function evaluation protocol for *two* parties, which is a variant of the protocol of Yao [40] (see also [18]). The protocol is run between two parties, the *Input Owner* A and the *Program Owner* B. The input of A is a value x, and the input of B is a description of a function f. At the end of the protocol, A should learn $f(x)$ (and no other information about f), and B should learn nothing about x. We will apply this protocol as a key component in our auction protocol, where the Program owner B is the auction issuer, and the input owner A corresponds to the auctioneer. The program computes the result of an auction.

The protocol is based on expressing f as a combinatorial circuit with gates defined over some fixed base $\mathcal{B}$ (e.g. $\mathcal{B}$ can include all the functions $g : \{0,1\} \times \{0,1\} \mapsto \{0,1\}$). The bits of the input are entered into input wires and are propagated through the gates.

Protocol for two-party secure function evaluation

Input: **A**'s input is a value x, **B**'s input is a combinatorial circuit which computes f.

Output: **A**'s output should be $f(x)$.

The Protocol:

- **Encrypting the circuit:** B assigns to each wire i of the circuit two random values (W_i^0, W_i^1) corresponding to 0 and 1 values of the wire (the random values should be long enough to be used as keys to a pseudo-random function, say 80 bits long). Denote the value of the wire by $b_i \in \{0,1\}$, B also assigns to the wire a random permutation over $\{0,1\}$, $\pi_i : b_i \mapsto c_i$. Denote $\langle W_i^{b_i}, c_i\rangle$ as the 'garbled value' of wire i.

  Consider a gate g which computes the value of the wire k as a function of wires i and j, $b_k = g(b_i, b_j)$. B prepares a table T_g which enables computation of the *garbled* output of g, $\langle W_k^{b_k}, c_k\rangle$, from the garbled inputs to g, namely the values $\langle W_i^{b_i}, c_i\rangle, \langle W_j^{b_j}, c_j\rangle$. Given the two garbled inputs to g, the table does not disclose information about the output of g for any other inputs, nor does it not reveal the values of the bits b_i, b_j, b_k of the inputs and output of g.

  The construction of T_g uses a pseudo-random function F whose output length is $|W_k^{b_k}| + 1$. Assume first that the fan out of each gate is one. The table contains four entries of the form

  $$c_i, c_j : \langle (W_k^{g(b_i,b_j)}, c_k) \oplus F_{W_i^{b_i}}(c_j) \oplus F_{W_j^{b_j}}(c_i)\rangle$$

  for $0 \le i, j \le 1$, where $c_i = \pi(b_i), c_j = \pi(b_j)$, and $c_k = \pi_k(b_k) = \pi_k(g(b_i, b_j))$. (The entry does not include its index c_i, c_j explicitly, as it can be deduced from the location.) The table masks the garbled value of the output wire using the output of the pseudo-random function F keyed by the garbled values of the input wires.

To verify that the table enables computation of the garbled output value given the garbled input values, assume that A knows $\langle W_i^{b_i}, c_i\rangle, \langle W_j^{b_j}, c_j\rangle$. A should find the entry (c_i, c_j) in the table T_k, and compute its exclusive-or with $(F_{W_i^{b_i}}(c_j) \oplus F_{W_j^{b_j}}(c_i))$. The result is $\langle W_k^{b_k} = W_k^{g(b_i,b_j)}, c_k\rangle$.

- **Coding the input:** The tables described above enable to compute the garbled output of every gate from its garbled inputs. Therefore given these tables and the garbled values $\langle W_i^{b_i}, c_i\rangle$ of the input wires of the circuit, it is possible to compute the garbled values of its output wires. Party A should therefore obtain the garbled values of the input wires.

  For each input wire, B and A engage in a 1-out-of-2 oblivious transfer protocol in which B is the sender whose inputs are the two garbled values of this wire, and A is the chooser whose input is the input bit. As a result of the oblivious transfer protocol A learns the garbled value of its input bit (and nothing about the garbled value of the other bit), and B learns nothing.

  B sends to A the tables that encode the circuit gates and a translation table from the garbled values of the output wires to output bits.

- **Computing the circuit:** At the end of the oblivious transfer stages party A has sufficient information to compute the output of the circuit for the input x by its own.

To show that the protocol is secure it should be proved that no party can gain more information than in the ideal model, in which there is a trusted third party which receives x from A and f from B, and sends $f(x)$ to A.

The main observation regarding the security of each gate, is that every masking value (e.g. $F_{W_i^{b_i}}(c_j)$) is used only once, and that the pseudo-randomness of F ensures that without knowledge of the correct key these values look random. Therefore knowledge of one garbled value of each of the input wires discloses only a single garbled output value of the gate; the other output values are indistinguishable from random to A.

As for the security of the complete circuit, the oblivious transfer protocol ensures that the A learns just a single garbled value for each input wire, and B does not learn which value it was. Inductively, A can compute just a single garbled output value of each gate, and in particular of the circuit. The use of permuted bit values c_k, hides the values of intermediate results (i.e. of gates inside the circuit).

Observe that the tables must use the output of a pseudo-random function F to mask the garbled output values of the gate. If this masking were accomplished by simply xoring the garbled input values to the corresponding garbled output values, then one could just xor the table entries, cancel out the masking elements, and discover the relations between garbled values. The pseudo-random function is therefore essential to hide relations between the different masking elements. If the fan out of a gate is greater than 1 then a different input to the pseudo-random function must be used at each gate where the wire is used. A simple method for achieving this is to assign to each gate a unique identifier I_g, and use $F_{W_i^{b_i}}(c_j, I_g) \oplus F_{W_j^{b_j}}(c_i, I_g)$ for masking.

It is also possible to adapt the protocol for circuits in which gates have more than two inputs, and even for wires with more than two possible values. The size of the table for a gate with ℓ inputs which each can have d values is d^ℓ.

Overhead

Note that the communication between the two parties can be done in a single back and forth round, and B can prepare the circuit in advance, before the input is known to A.

Consider a circuit with n inputs and m gates. The protocol requires B to prepare m tables and send them to A. This is the major communication overhead of the protocol and can be performed offline, prior to disclosure of the input to A. In the case of binary gates, the communication overhead is $4m$ times the length of the output of the pseudo-random function (typically 8 to 16 bytes long).

The main computational overhead of the protocol is the computation of the n oblivious transfers. They require each of the two parties to perform a total of $O(n)$ exponentiations. Afterwards party A computes the output of the circuit, and this stage involves m applications of a pseudo-random function. The overhead of this stage is typically negligible compared to the oblivious transfer stage.

2.4 Secure Function Evaluation for Auctions

The computation of auctions involves three types of parties: *Bidders* who know the inputs, an *auction issuer* that prepares the circuit and and *auctioneer* that learns the output of the circuit. The secure protocol we present preserves the original communication pattern of auctions in which a bidder only interacts with the auctioneer. It uses a single additional round of communication between the auctioneer and the auction issuer.

The protocol evaluates a function f which computes the result of the auction. For a second price auction, where the bids are $(x_1, \ldots, x_n)$, this function is $f(x_1, \ldots, x_n) = \langle i, p\rangle$, where i is the identity of the highest bidder ($x_i = max(x_1, \ldots, x_n)$) and $p = max(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n)$ is the second highest bid. (The specification of the function should also define how ties are broken.)

Consider the protocol of Section 2.3 for two-party secure function evaluation. A key observation about this protocol is that if there are multiple inputs $x_1, \ldots, x_n$ (known to different parties) and somehow, party A who was given the garbled version of a circuit that evaluates f is given the garbled values for each of the bits of the x_i's, then A can evaluate $f(x_1, \ldots, x_n)$ without learning anything else about the x_i's themselves. This observation shows how to transform the two-party protocol into a multi-party protocol for computing auctions: the auctioneer is party A, the inputs are known to the bidders, and the auction issuer is party B. The only step missing is allowing the auctioneer to learn the garbled values of the bidders' inputs. This is done using *proxy oblivious transfer.*

Protocol for secure evaluation of an auction

Input: **Bidder** i'a input is a bid x_i. The **auction issuer** has a description of a function f that computes the auction.

Output: The **auctioneer** should compute $f(x_1, \ldots, x_n)$.

Protocol:

- **Encrypting the circuit:** The AI constructs a circuit that computes the auction. It garbles the circuit identically in the manner in which party B garbled the circuit in the two-party protocol.
- The auctioneer advertises the auction, its terms and the public-key of the AI and invites bids.
- **Coding the input:** Each bidder i engages in a a *1-out-of-2 proxy oblivious transfer* protocol for each of the bits of x_i. In this protocol, the AI is the sender, and its two inputs are the garbled values for the input bit. The bidder is the chooser, and its input is its input bit, and the auctioneer is the proxy. At the end of the protocol, the auctioneer learns the garbled value of the input bit (i.e. $W_i^{x_i}$), but not the value of x_i.
- **Computing the circuit:** At the end of the proxy oblivious transfer stage, the auctioneer possesses sufficient information to compute the circuit independently, exactly as in the two-party protocol.

To complete the specification of the protocol we should discuss several issues.

Communication pattern: Bidders communicate only with the auctioneer and not with the auction issuer or with other bidders. The naive communication pattern of the proxy OT protocol consists of a first stage in which each bidder sends a message to the auctioneer and a message to the AI, and a second stage in which the AI sends a message to the auctioneer. This structure requires direct communication between the bidder and the AI, which we are trying to avoid. However, the bidders can use the auctioneer as a communication channel to the AI. Each bidder takes the message that should be sent to the AI and encrypts it with the AI's public key (using a non-malleable encryption scheme, see below) and sends it to the auctioneer. When all bidders have completed this phase, the auctioneer can send the messages to the AI.

Encryption scheme security: It is crucial to encrypt the messages from the bidders using a non-malleable encryption scheme (defined and introduced in [10]). Such a scheme prevents the auctioneer from causing *meaningful* changes in the cleartext by changing the ciphertext (in fact auctions are the example given in [10] for the need for non-malleable encryption: Suppose, for example, that the encryption was simply xoring the bid with a one-time pad. Then it is easy to reverse some bits of the bid by reversing the corresponding bits of the ciphertext). If the public-key of the AI is to be used for several auctions that it should be secure against chosen ciphertext attacks in the post-processing mode (see [10, 9, 3] for an up-to-date discussion of the issues).

Care should also be taken to prevent a replay attack that repeats a bid from an old auction in a new one. This can simply be handled by adding names to the auctions (say auctioneer name and date). The bidders add this name to their plaintext messages and then encrypt. The non-malleability property assures that the name cannot be modified. The AI should make sure it does not engage in an auction with the same name twice, and that it uses a fresh garbled circuit for every auction.

Verifying the output: Bidders should verify that the auctioneer has computed the circuit constructed by the AI, and that the auctioneer indeed sent the bids to the AI (if not, the bids were not considered in computing the outcome).

A naive verification procedure is to require the auctioneer to publish the tables and garbled input values of the circuit (signed by the AI), and allow suspecting bidders to simulate its computation. However, a more efficient verification method is to use a signed 'translation' table that the AI generates for the output wires of the circuit. For each output wire, i, the table should contain the entries $\langle 0, G(W_i^0)\rangle$ and $\langle 1, G(W_i^1)\rangle$, where G is a one-way function[4]. The auctioneer displays the values $\langle b_i, W_i^{b_i}\rangle$ for each output wire. Each bidder can verify that $\langle b_i, G(W_i^{b_i})\rangle$ appears in the table. Since G is one-way, the only way that the auctioneer can generate $W_i^{b_i}$ is by computing the circuit.

A simple method for verifying that all bids were considered in the auction requires the AI to sign a list of hash values of each of the messages it received from the bidders. These hash values are displayed by the auctioneer. Bidders can check that the AI signed the hash of their messages.

A corrupt auction issuer: A different type of attack involves a corrupt auction issuer. If the AI colludes with a bidder (or subset of bidders), it can, for example, provide a program that always declares a certain bidder the winner. Alternatively, it can disrupt the computation by sending incorrect values to the input wires. These types of attacks can be detected using a "cut-and-choose" technique: the AI is required to provide m copies of the program (including commitments to the garbled input wires). The auctioneer then asks it to remove the garbling and provide the inputs for half of the copies, examines that they compute the desired function, and runs the protocol with the remaining copies, verifying that they all yield the same output. The probability that a corrupt AI is not caught is exponentially small in m. It seems that even using $m = 2$ copies and opening one of them is sufficient, as an AI caught cheating even once would (at least) lose its credibility, and so the risk-to-benefit ratio of such collusion is quite small (in fact, it might be sufficient for the auctioneer to decide at random whether to require more than a single copy of the program, setting the probability according to the risk that it is willing to take.).

Denial of service attack by bidder: A corrupt bidder might disrupt the computation of the auction by preventing the auctioneer from receiving correct garbled input values in the proxy-OT stage (see Appendix A). The solution to this attack should enable the auctioneer to prove to the AI that this event took place, and receive the garbled values corresponding to a '0' bid of that bidder. However, care must be taken to prevent a corrupt auctioneer from using this mechanism to complain against innocent bidders and learn their bids. See Appendix A for the details of this solution.

2.5 Computational and Communication Overhead

The different parties of the above protocol incur different computational and communication loads. We will survey them now. Each bidder engages as a chooser in a proxy oblivious transfer protocol a number of times proportional to the number of bits in its input. Under the implementation we propose this implies a number of exponentiations proportional to its input length. For an application like an auction this yields a modest load. Note that the encryption with the AI public-key can be done as one message for all the inputs bits of each bidder, and hence does not add significantly to the load.

[4] $G(x)$ can be defined, for example, as $F_x(0)$, where F is the pseudo-random function used for garbling the circuit.

The AI has to prepare the garbled circuit (which can be done offline) and send it to the auctioneer, and it has to engage as the sender in the proxy oblivious transfer protocol a number of times proportional to the *total* number of input bits. This last part may be significant, so we have attempted to reduce it (see below).

The auctioneer has to participate as proxy in the proxy oblivious transfer protocol a number of times proportional to the total number of input bits. It has to evaluate the garbled circuit, which means a number of pseudo-random function evaluations proportional to the circuit size. Since the circuit size affects both the communication between the AI and the auctioneer (in total table size) and the work the auctioneer must perform, it makes sense to put effort in optimizing it (see below).

Optimizing the proxy OT: The computation of the exponentiations in the proxy oblivious transfer stage the main computational overhead of the protocol (unless the circuit is extremely large, which is not the case for auctions). The proxy oblivious transfer protocol uses El Gamal encryptions which are of the form $(g^r, (g^x)^r \cdot m)$, where r is chosen by the AI and x by the bidder (see App. A). It is therefore tempting to minimize the work and the communication by reusing the same r for several encryptions. Care should be taken not to hamper the security by such an optimization. Given that the total length of these encryptions may be the bulk of the communication (and these messages cannot be sent offline) it makes sense perhaps to employ El Gamal encryptions over elliptic curves which have more succinct representation than $GF[P]$.

Optimizing the circuit size: Consider an auction with N bidders, where a bid is an integer in the range $[1, L]$ (i.e. it can be represented by $\ell = \log L$ bits). Our best circuit design for second price auctions uses approximately $30N\ell$ table entries. In order to optimize the size of the tables that describe this circuit, it uses some wires which are not binary but can rather have one of three values. Assuming that each entry is of 10 bytes, the table size is about 300 bytes per input bit. Note that even for $N = 1000$ users, and bids with one million possible values, the size of the circuit is moderate, $30 \cdot 1000 \cdot 20 = 600,000$ table entries, or 6Mbytes. A circuit for first price auctions is about half as large.

It is possible to further reduce the size of the tables by a factor of 25%. For the case of gates with two binary input wires the saving is achieved by setting one garbled output value to be a function of the corresponding garbled input values (and then the table does not have to contain an entry for this output value).

2.6 Implementation of the Protocol

We are developing a prototype of the architecture for the case of second price auctions with hundreds of bidders. The implementation is done in the Python scripting language. The entire implementation takes about 1500 lines of python code, together with about 800 lines of C code for the computation of the pseudo-random function and the encryption. It uses El Gamal public key encryption and DSA signatures, which use exponentiations over a 768-bit modulos. The exponentiations are coded in assembler.

The proxy oblivious transfer protocol incurs most of the communication and computation overhead. As for the communication overhead, the tables that code the circuit can be sent from the AI to the auctioneer in advance, before the auction begins ,possibly on a CD-ROM or DVD. (The structure of the circuit does not have to be sent, since the auctioneer can compute it by itself, given the number of bidders and the size of the bids.) Most of the online communication between the AI and the auctioneer is for the proxy OT protocol in which for each bit the AI receives one public key and sends two encryptions.

The proxy OT stage incurs also most of the computation overhead. A straightforward implementation requires the AI to compute, for every input bit, two online exponentiations (for encryption), and requires the auctioneer to compute one exponentiation (for decryption). It is possible to use a variant of oblivious transfer with low amortized overhead to reduce this overhead to one exponentiation per several input bits. The circuit itself contains about four gates per input bit, and its evaluation is less computation intensive. To evaluate each gate the auctioneer computes two values of a pseudo-random function. The throughput of the pseudo-random function is hundreds of thousands of operations per second and thus the total overhead is marginal.

3 Other Auctions and Mechanisms

The overhead of the protocol depends only on *the size of the combinatorial circuit* that evaluates the function deciding the outcome of the auction. It is, therefore, possible to use the same protocol for computing various other types of auctions and mechanisms which can be expressed as a circuit of moderate size.

3.1 Auctions

The difference between a protocol for a first-price auction and a protocol for a second-price auction is only that they use different circuits to compute the result of the auction (the circuit for first-price auctions being simpler). Similar circuits can be used to express **k-th price auctions**, where the winner is the highest bidder and the clearing price is the k-th highest bid. Note that unlike first or second price auctions, k-th price auctions do not have an interactive implementation (similar to the English or the Dutch auction) which guarantees that the auctioneer does not misuse the submitted sealed bids. The size of the circuit for a k-th price auction is *not* k times the size of a circuit for first price auction, but is rather just $O(N\ell+k\ell)$. (This can be done via a binary search on the value of the k largest bid) The number of inputs, determining the number of oblivious transfers and thus of public key operations, is the same as for first price auctions. k-th price auctions where proposed as suitable for risk seeking bidders [28, 29].

Wurman, Walsh, and Wellman [39] discuss the design of **double auctions** supported by the AuctionBot [1]. In particular they consider the case where there are M sell offers and N buy offers, and analyze the **Mth-price** and **$(M+1)$st-price** rules. These rules can be easily expressed as circuits similar to those for k-th price auctions, and can, consequently, be implemented by our protocol.

MacKie-Mason and Varian [27, 35] present an exposition of the **Generalized Vickrey Auction** (GVA) as a powerful mechanism capable of solving many complex problems. In this mechanism each party reports a utility function, and the center calculates the allocation that maximizes the sum of the reported utilities subject to the resource constraints. The payment of party i depends on the difference between the sum of the utilities of the *other parties* in the chosen allocation and the sum of their utilities in the allocation which does not take party i into account. This mechanism can be used to sell *multiple units* of the same goods to consumers who have a utility function which depends on the number of units they receive The GVA can be expressed as a small circuit, if the consumer utility functions are not overly complex. For example, in the problem of selling M units of the same goods, each consumer utility function can be a mere list of the values for the 1st, 2nd, and up to the Mth unit of the goods. Therefore, each consumer's input is composed of M values, and the circuit that decides the outcome of the mechanism is relatively simple.

Finally note that auctions are sometimes implemented by several rounds where in each round the bidders may get to add some inputs (e.g. the notable FCC spectrum auctions). Such auctions can also be handled in our framework, though it may be the case that in some instances the need for several rounds is to reduce leaking information, which is taken care of directly by our architecture.

3.2 Other Mechanisms

There are efforts to design algorithms based on mechanism design, which involve many selfish agents where the goal is to solve a 'global' problem. (e.g. routing, or some cooperation between the agents). See for example Walsh et al [37], or Nisan and Ronen [30]. Our protocols can be used to compute these algorithms without requiring trust in the center[5]. The plausibility of using our protocols for this task depends on the complexity of expressing the utility functions and the decision procedure as circuits.

A particular case of interest is **Groves-Clarke** mechanisms [19, 7], where a public good is produced if the sum of reported values is higher than a given threshold. The circuit that computes this function is quite simple, as is a circuit that computes the sum of the reported values for several options and decides on the option with the highest sum. It is, therefore, simple to provide a private protocol which computes these mechanisms.

Another relevant application is the design of mechanisms to **elicit opinions** of a group of independent experts. Glazer and Rubinstein [15] observe that in a "culture" in which experts care that their recommendations are accepted, there is a mechanism with a single equilibrium which achieves the public target (and such a mechanism does not exist if experts only care about the public good). The mechanism they suggest can be computed very efficiently by our architecture (essentially the mechanism requires that one expert choose a subset of the experts whose opinions are considered, learn their opinions, and then add his/her own opinion. The decision is the majority opinion. Note that this process requires privacy – the expert choosing the subset must do so *before* learning their opinions. A privacy preserving implementation of this application might be useful in organizations that want to poll the opinions of their members, but realize that individuals might bias their reported opinion if they believe it will be revealed.

Stable matching [20, 33] is yet another example of a global decision which depends on the private preferences of many parties. In many scenarios it is quite plausible that parties would be hesitant to reveal their matching preferences, even to the center computing the matching (consider for example matching couples for a prom...). Our architecture enables the parties to reveal their true preferences without fearing that the center will learn them. The overhead of implementing our architecture for this application depends on the complexity of expressing the matching algorithm as a combinatorial circuit. Finding a reasonably small circuit for this problem is an interesting problem, as the classical Gale-Shapley algorithm requires the power of indirect addressing of a RAM, and hence its translation into a circuit is rather cumbersome.

[5] However, this task might be computationally difficult regardless of privacy requirements. The work of [24] shows that occasionally the procedures that compute the outcome are NP-hard

4 Further Research

There are many issues for further research, both in investigating how to improve the properties of the architecture and in making the protocol more efficient.

It is possible to further reduce the trust that bidders should put in the architecture, by distributing the task of the auction issuer between several parties. Ideally, these could be different companies, with none of them knowing all the information currently known to the AI. A more modest goal (for which we have promising preliminary results) is to have a single party generate the garbled circuit (offline) and then distribute each garbled gate between several AI servers. The system should have a threshold k, such that the auctioneer should have to contact k servers in order to compute the auction, and collusion between the auctioneer and any $k-1$ servers reveals nothing about the bids (note that a naive use of secret sharing does *not* achieve this property). As for distributing the garbling part, one can use the techniques of [2], but they seem to incur a significant overhead.

The problem of backing the bids with financial commitments is somewhat orthogonal to the main issues we consider. The solutions of [14] for this problem should be adaptable to our architecture.

Our current techniques implement second price auctions for hundreds or even thousands of users with very reasonable overhead. A great deal of research remains to be done in improving the efficiency of the protocol. One direction is designing efficient circuits for auctions and other mechanisms, while another direction is improving the efficiency of the cryptographic tools we use, mostly of the extensive application of oblivious transfer.

5 Acknowledgments

We thank Noam Nisan and Uri Zwick for useful discussions and the anonymous referees for their comments.

References

[1] `http://auction.eecs.umich.edu`

[2] D. Beaver, S. Micali and P. Rogaway, "The round complexity of secure protocols", *Proc. 22nd ACM Symp. on Theory of Computing*, 1990, 503–513.

[3] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway, "Relations among notions of security for public-key encryption schemes", *Advances in Cryptology - Crypto '98*, Springer-Verlag LNCS 1462 (1998), 26–45.

[4] M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications", *Proc. Adv. in Cryptology - Crypto '89*, Springer-Verlag LNCS 435 (1990), 547-557.

[5] C. Cachin, "Efficient private bidding and auctions with an oblivious third party", to appear, *Proc. 6th ACM Conf. on Computer and Communications Security*, 1999.

[6] K. Chui and R. Zwick, "Auction on the Internet – A preliminary study", manuscript, 1999. Available at `http://home.ust.hk/~mkzwick/Internet_Auction.html`

[7] E. Clarke, "Multiparty pricing of public goods", *Public Choice*, 11:17-23, 1971.

[8] R. Cramer, I. Damgard and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols", *Proc. Advances in Cryptology – Crypto '94*, Springer-Verlag LNCS 839 (1994), 174–187.

[9] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attacks", *Proc. Advances in Cryptology - Crypto '98*, Springer-Verlag LNCS 1462 (1998), 13–25.

[10] D. Dolev, C. Dwork and M. Naor, "Non-malleable cryptography", *Proc. 23th ACM Symp. on Theory of Computing*, 1991. Full version: to appear Siam J. on Computing. Available at `http://www.wisdom.weizmann.ac.il/~naor/onpub.html`

[11] R. Engelbrecht-Wiggans and C. M. Kahn, "Protecting the winner: second price vs. ascending bid auctions", *Economic Letters*, Vol. 35, 1991, 243–248.

[12] S. Even, O. Goldreich and A. Lempel, "A Randomized Protocol for Signing Contracts", *Communications of the ACM* **28**, 1985, pp. 637–647.

[13] U. Feige, J. Kilian and M. Naor, "On minimal models for secure computation", *Proc. 26th ACM Symp. on Theory of Computing*, 1994, pp. 554–563.

[14] M. K. Franklin and M. K. Reiter, "The design and implementation of a secure auction server", *IEEE Tran. on Software Engineering*, 22(5), pp. 302–312, 1996.

[15] J. Glazer and A. Rubinstein, "Motives and implementation: on the design of mechanisms to elicit opinions", *J. of Economic Theory* 79, 157–173, 1998.

[16] O. Goldreich, *Secure Multi-Party Computation* (working draft) Version 1.1, 1998. Available at `http://philby.ucsd.edu/books.html`

[17] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions", *J. of the ACM.*, vol. 33, 1986, 792–807.

[18] O. Goldreich, M. Micali and A. Wigderson, "How to play any mental game", *Proc. 19th ACM Symp. on Theory of Computing*, 1987, pp. 218–229.

[19] T. Groves, "Incentives in teams", *Econometrica*, 41:617–631, 1973.

[20] D. Gusfield and R. Irving, **The Stable Marriage Problem : Structure and Algorithms**, MIT Press, 1989.

[21] M. Harkavy, J. D. Tygar and H. Kikuchi, "Electronic auctions with private bids", *3rd USENIX Workshop on Electronic Commerce*, pp. 61–73, 1999.

[22] Q. Huai and T. Sandholm, "Mobile Agents in an Electronic Auction House", *Mobile Agents in the Context of Competition and Cooperation (MAC3-workshop)*, 1999.

[23] R. Impagliazzo and S. Rudich, "Limits on the Provable Consequences of One-Way Permutations", *20th ACM Symp. on the Theory of Computing*, 1989, 44–61.

[24] N. Kfir-Dahav, D. Monderer and M. Tennenholtz, "Mechanism design for resource bounded agents", 1998.

[25] M. Kumar and S. I. Feldman, "Internet auctions", *3rd USENIX Workshop on Electronic Commerce*, 1999.

[26] Luby M., **Pseudorandomness and Cryptographic Applications**, Princeton University Press, 1996.

[27] J. K. MacKie-Mason and H. R. Varian, "Generalized Vickrey auctions", 1994.

[28] D. Monderer and M. Tennenholtz, "K-Price auctions", Working Paper, 1998.

[29] D. Monderer and M. Tennenholtz, "Internet auctions", Working Paper, 1998.

[30] N. Nisan and A. Ronen, "Algorithmic mechanism design", *Proc. 31st ACM Symp. on Theory of Computing*, 1999, 129-140.

[31] A. J. Osborne and A. Rubinstein, **A Course in Game Theory**, MIT Press, 1994.

[32] M. O. Rabin, "How to exchange secrets by oblivious transfer", Tech. Memo TR-81, Aiken Computation Laboratory, 1981.

[33] A. E. Roth and M. A. Sotomayor, **Two-Sided Matching : A Study in Game-Theoretic Modeling and Analysis**, Cambridge Univ Press, 1990.

[34] S. G. Stubblebine and P. F. Syverson, "Fair on-line auctions without special trusted parties", *Proc. of Financial Cryptography '99*, 1999.

[35] H. R. Varian, "Economic mechanism design for computerized agents", *First USENIX Workshop on Electronic Commerce*, 1995.

[36] D. Vickrey,"Counter speculation, auctions, and competitive sealed tenders", *Journal of Finance*, March 1961, pp. 9–37.

[37] W. Walsh, M. Wellman, P. Wurman and J.K. MacKie-Mason, "Auction protocols for decentralized scheduling", *18th Int. Conf. on Distributed Computing Sys.*, 1998.

[38] M. Wellman and P. Wurman, "Real time issues for Internet auctions", *First IEEE Workshop on Dependable and Real-Time E-commerce Systems*, 1998.

[39] P. Wurman, W. Walsh and M. Wellman, "Flexible double auctions for electronic commerce: theory and implementation", *Decision Support Systems* 24:17-27, 1998.

[40] A.C. Yao, "How to generate and exchange secrets", *Proc. of the 27th IEEE Symp. on Foundations of Computer Science*, 1986, pp. 162–167.

A Oblivious Transfer

Essentially every known suggestion of public-key cryptography also allows implementation of oblivious transfer, (*but there is no general theorem that implies this state of affairs*). OT can be based on the existence of trapdoor permutations, factoring, the Diffie-Hellman assumption (both the search and the decision problems, the latter yielding more efficient constructions) and the hardness of finding short vectors in a lattice (the Ajtai-Dwork cryptosystem). In contrast it seems to be highly unlikely that OT can be based on one-way functions (the reason being that given an OT protocol, it is simple to implement secret-key exchange using it. It therefore follows from the work of Impagliazzo and Rudich [23] that there is no black-box reduction of OT from one-way functions).

Following we describe a protocol which is based on the protocols of Bellare and Micali [4]. The inputs and the required outputs of the protocol were described in section 2.2.2.

Initialization: The sender S and the chooser C agree on a large cyclic group G_g generated by g, in which the discrete log problem is believed to be hard. In addition, they agree on a value $c \in G_g$ whose discrete log is unknown to C.

Preparing the query: C chooses a random value $0 < r < |G_g|$, calculates $PK_\sigma = g^r$ and $PK_{1-\sigma} = c/PK_\sigma$ and sends PK_0 to S. In the El Gamal public key encryption system these PK_i's are public keys, and r is the private key corresponding to PK_σ.

Sending the values: S calculates $PK_1 = c/PK_0$. Using El Gamal encryption, S then sends to C the encryptions $E_{PK_0}(m_0), E_{PK_1}(m_1)$.

Receiving the desired value: Knowing the private key corresponding to PK_σ, C decrypts $E_{PK_\sigma}(m_\sigma)$ to recover m_σ.

Assuming that C knows the private key for PK_σ it cannot possibly know the private key for $PK_{1-\sigma}$ since if it did, it could calculate the discrete log of $c = PK_\sigma \cdot PK_{1-\sigma}$. It would therefore appear that C can only decrypt one of the messages it receives from S. Note though that while it is impossible for C to know both private keys, the protocol does not rule out the possibility that C possesses partial information about each of the keys (although there seems to be no obvious way in which C can obtain such information). This problem can be solved if we require C to prove to S that it knows the discrete log of one of the public keys (without disclosing to S of which one). This proof can be efficiently done using the techniques of Cramer et al [8], at the cost of a few more exponentiations. The resulting protocol is, therefore, as secure as the Decision Diffie-Hellman assumption, which is believed to be secure.

Alternatively, if one postulates the existence of random oracles (i.e., of a function H which behaves as a random function), it is possible to run the protocol without the proofs of [8]. To send the values the sender should select a random value k, and send the message

$$\langle g^k, H(PK_0^k) \oplus m_0, H(PK_1^k) \oplus m_1 \rangle$$

to the chooser. The chooser can only decrypt m_σ by computing $PK_\sigma^k = (g^k)^r$.

1-out-of-2 Proxy OT

The protocol for proxy OT involves a third party P which is C's proxy. The roles of the parties were defined in Section 2.2.2. The protocol is as follows:

Initialization: The parties agree on a large cyclic group G_g and a generator g, as in the 1-out-of-2 OT protocol.

Preparing the query: C chooses a random $0 < r < |G_g|$, calculates $PK_\sigma = g^r$ and $PK_{1-\sigma} = c/PK_\sigma$. It sends PK_0 to S, and sends the private key r to P. S computes $PK_1 = c/PK_0$. Using El Gamal encryption, it then sends to P the following pair, in random order,

$$\langle E_{PK_0}(\mathcal{C}(m_0)), E_{PK_1}(\mathcal{C}(m_1)) \rangle,$$

where $\mathcal{C}$ is a good error detecting code.

Receiving the desired value: P knows the private key corresponding to PK_σ. It tries to decrypt both values and uses the error detecting code $\mathcal{C}$ to determine which one was decrypted correctly.

Note that P does not learn the value of σ, C's choice. In order to achieve security which depends only on the Decisional Diffie-Hellman assumption C must prove that it knows the discrete log of one of the keys, as in the protocol for 1-out-of-2 OT. Alternatively, one can use a "random oracle" H as in the 1-out-of-2 OT protocol described above. This is the method currently used by our prototype implementation.

Preventing a denial of service attack: The denial of service attack described in section 2.4 involves a bidder which prevents the auctioneer from retrieving correct garbled values in the proxy OT stage. In more details, the bidder (the chooser C) sends to the auctioneer (the proxy P) a private key which does not correspond to any of the public keys sent to the auction issuer (the sender S). The auctioneer, therefore, cannot decrypt any of the messages received from the auction issuer.

A straightforward solution is to require the bidder to prove to the auctioneer that it sent it a correct private key. Such a proof is, however, quite inefficient.

A different approach is to enable the auctioneer to prove to the AI that the auctioneer received an incorrect private key from the bidder. Care must be taken to prevent a corrupt auctioneer from accusing an innocent bidder. A concrete solution is as follows: (1) The AI chooses a key K_b for every bidder b, and uses it to encrypt all the garbled values corresponding to the input bits of this bidder. (2) The proxy OT protocol is used to transfer these encrypted values. (3) The auctioneer checks whether it received legitimate values (using the error detection code $\mathcal{C}$). (4) If all the values of input wires of bidder b were received correctly, the auctioneer asks to receive the key K_b from the AI. It uses the key to decrypt the received values and retrieve the garbled values of the input wires. Otherwise the auctioneer asks to receive the garbled inputs corresponding to a '0' input of this bidder. Note that this solution adds a communication round between the auctioneer and the AI, but it typically consists of a message containing a single key for every bidder, and its length is quite short.