An algorithm applicable to clearing combinatorial exchanges

Marius-Călin Silaghi
Florida Institute of Technology

September 17, 2002

Technical Report
CS-2002-14
Florida Institute of Technology
Melbourne, Florida 32901-6975

Abstract

It is important to approach negotiations in a way that ensures privacy. So far, research has focused on securely solving restricted classes of negotiation techniques, mainly the (M+1)-st-price auctions. Here we show how these results can be adapted to more general problems.

This paper extends our previous results on how distributed finite discrete problems can be solved securely. Such problems can model larger classes of negotiation problems, e.g. Combinatorial Exchanges [SF02]. In Finite Discrete Maximization, each tuple in the problem space is associated with an integer value in a predefined interval and we search for a maximizing input. Values from different subproblems are combined additively. We show that unconstrained distributed Finite Discrete Maximization problems can be solved securely using a scheme that we propose for translating shared secret values into shared differential bids. Differential bid vectors are already used in [AS02, Bra02b].

Constrained distributed Finite Discrete Maximization poses additional challenges, due to the loss of additivity of the maximized cost, when infeasibility is marked as the lowest finite value. We found two ways of solving this problem: a) by using an additional multiplication value; and b) by using larger variable domains. While the first alternative enforces a threshold to the privacy level in our current protocol, the second one increases much the complexity of the computation. The proposed algorithms are only (t/3)-private, where t is the number of participants.

1 Introduction

Cryptographic protocols can enforce certain privacy guarantees in distributed computation of functions [GB96]. [GMW87, CCD88b, CCD88a, BOGW88] show how cryptographic protocols can be compiled from protocols/functions for honest agents. For some combinations of problems, concepts of security and types of attacks, cryptographic protocols obtained this way can be safe.

Example 1.1 *Without intractability assumptions, when at most a minority of the participants (<1/3) can make coalition, there exist cryptographic protocols that are safe even against active attacks [CCD88a].*

Example 1.2 *"Fully private (M+1)-st-price auctions", secure against any collusion, have been described in [Bra02a]*

Many existing cryptographic protocols are targeted to the computation of functions based on addition and multiplication. Optimization problems are efficiently approached by branching algorithms (e.g. branch and bound, ...), but these show to be difficult to implement securely [Sil02c].

In this article we show how general finite discrete maximization can be solved securely, being guaranteed to resist attacks from any colluding set of less than a third of the participants. Our algorithm is based on the recent secure auction solving, proposed in [Bra02b]. Finite Discrete Maximization problems are problems where each tuple in the problem space is associated with an integer value in a predefined interval and a maximizing input is looked for. Values from different parts combine additively. We show that unconstrained distributed Finite Discrete Maximization problems can be solved securely using a scheme that we propose for translating shared secret values into shared differential bids [AS02].

Constrained distributed Finite Discrete Maximization poses additional challenges, due to the loss of additivity of the maximized cost, when infeasibility is marked as the lowest finite value. We propose two ways of solving this problem: a) by using an additional multiplication value; b) and by using larger variable domains. While the first alternative enforces a threshold to the privacy level, the second one increases strongly the complexity. Unfortunately, but expectedly, the obtained algorithms are slower than known optimization algorithms.

2 Background

After a long series of important results in cryptographic protocols [GMW87, CCD88b, CCD88a, BOGW88, GB96], recent results report more and more applications. Several reports are enthusiastic about the impressing results that can be achieved using semi-trusted parties [SM]. However, the final success of this kind of techniques is not yet clear [LAN02]. [Bra02b] show how simple auctions can be solved privately without any servers and auctioneers.

Here we address a more general class of maximization problems that allows us to approach problems such like combinatorial auctions with multiple buyers and sellers.

2.1 Maximization problems

An important class of optimization problems requires to find valuations of a set of variables that maximize the sum of a set of secret functions.

Definition 2.1 (FDM) *A Finite Discrete Maximization problem* (X, D, F) *is defined by a set of variables* $X = \{x_1, x_2, ..., x_n\}$ *and a set of functions* $f_i \in F$, $f_i : D_i^1 \times D_i^2 \times ... \rightarrow D_i'$, *where* D_i' *is a finite range in* $\mathbb{N}$, $D_i^k \in D$ *is the domain of a variable* $x_i^k \in X$.

The problem is to find a valuation t *of all variables in* X, *that maximizes* $\sum_i f_i(t)$.

Definition 2.2 (FDDM) *A Finite Discrete Distributed Maximization problem* (A, X, D, F) *is defined by a set of agents, each of them,* A_j, *owning a private FDM with functions* $f_{i,j}$ *and eventual shared variables. They are looking for a solution that maximizes the FDM* (X, D, F) *obtained by the union of the variables and functions in all the agents.*

The problem is to find a valuation t *of all variables in* X, *that maximizes* $\sum_{i,j} f_{i,j}(t)$ *for all* $f_{i,j}$ *in all agents* A_j.

In public settings, algorithms can exploit intricate branching procedures to ensure efficient cuts in the problems. However, there hasn't been much progress in securely applying this type of efficient algorithms.

It is important to note that FDDMs can be used to model Combinatorial Exchanges, a general type of negotiations of which auctions are a well-known instance [SF02].

2.2 Constrained problems with constant objective function

Imagine we want to solve $P = (X, D, F)$ where X is a set of variables $x_1, x_2, ...x_n$, F is a set of functions with results in the set $\{0, 1\}$, D a set of finite discrete domains for X, and we require that $\sum f_i = |F|$. The domain of x_i is $D_i \in D$, whose values are $v_1^i, v_2^i, ..., v_{|D_i|}^i$.

Definition 2.3 (first solution) *The first solution of* P *given a total order* $\prec_1$ *on its variables* X, *and a total order on its values* $\prec_2$ *is the first among solution tuples when these are ordered with the lexicographical order induced by* $\prec_1$ *and* $\prec_2$.

An algorithm for finding the first solution of such a problem based solely on additions and multiplication was proposed in [Sil02b], and can be straightforwardly compiled [GMW87, CCD88b, CCD88a, BOGW88] into a secure protocol. As this is the main predecessor of the techniques proposed here, we will detail it and its drawbacks.

Often, one has to restrict a problem by adding additional constraining functions. We define the union between a problem $P = (X, D, F)$ and a function f as the problem $P = (X, D, F \cup \{f\})$.

$$(X, D, F) \cup f = (X, D, F \cup \{f\})$$

Let us imagine that we have a function `satisfiable`*(P)* with the next property (an example is given later):

$$\texttt{satisfiable}(P) = \begin{cases} 1 & \text{if } P \text{ has a solution} \\ 0 & \text{if } P \text{ is infeasible} \end{cases}$$

We will design now a set of functions: $f_1, f_2, ..., f_n,\ f_i : \mathcal{P}^* \rightarrow \mathbb{N}$, such that each f_i will return the index of the value of x_i in the first solution, or 0 if no solution exists.

$$f_i(P) = \begin{cases} k & \text{if } P \text{ has the first solution for } x_i = v_k^i \\ 0 & \text{if } P \text{ has no solution} \end{cases}$$

For this purpose, we will first design the functions $g_{i,1}, g_{i,2}, ..., g_{i,|D_i|}.\ g_{i,j} : \mathcal{P}^* \rightarrow \{0, 1\}$.

$$g_{i,j}(P) = \begin{cases} 1 & \text{if } P \text{ has a first solution for } x_i = v_j^i \\ 0 & \text{if } P \text{ is infeasible for } x_i = v_j^i \end{cases}$$

Whenever a first solution exists, a simple implementation is:

$$g_{i,j}(P) = \texttt{satisfiable}(P \cup \{x_i = v_j^i\} \cup_{k<i} (x_k = v_{f_k(P)}^k)) \tag{1}$$

where $v_{f_k(P)}^k$ is the ($f_k(P)$-th) value of x_k, the value that x_k takes in the first solution.

Namely, $g_{i,j}(P)$ is 1 selecting v_j^i for x_i, if and only if the problem obtained by adding to P the function

$$\lambda_i = \begin{cases} 1 & \text{if } x_i = v_j^i \\ 0 & \text{if } x_i \neq v_j^i \end{cases}$$

that selects the current value, and the functions

$$\lambda_k^* = \begin{cases} 1 & \text{if } x_k = v_{f_k(P)}^k \\ 0 & \text{if } x_k \neq v_{f_k(P)}^k \end{cases}$$

instantiating previous variables to their values in the first solution, is satisfiable.

This recursion is possible by first computing the value of the first variable in the first solution, f_1, based on $g_{1,j}$ as described next. Based on the result one can compute $g_{2,j}$. Then, one can now compute f_2. The recursion continues with $g_{i,j}$ that helps in computing f_i.

We define functions $t_{j,1}, t_{j,2}, ..., t_{j,|D_j|}.\ t_{j,i} : \mathcal{P}^* \rightarrow \{0, 1\}$. These functions hold temporary results, whose semantic is that **no** satisfiable subtree exists under any node $x_j{=}v_k^j, k{\leq}i$, when previous variables are assigned according to the values in the first solution:

function value-to-differential-bid-vector($c(t)$, K)

1. Jointly, all agents build a vector a_t^0 for the secret value $c(t)$.
   $a_t^0 =< c_{t,0}^0, c_{t,1}^0, ..., c_{t,K}^0 >$. Actually,
   $a_t^0 =< c(t), c(t)-1, c(t)-2, ..., c(t)-K >$.

   To achieve this, each agent A_i computes
   $a_t^0(i) =< c_{t,0}^0(i), c_{t,1}^0(i), ..., c_{t,K}^0(i) >$. where $c_{t,0}^0(i) = c_i(t)$, and for $k > 0$, $c_{t,k}^0(i) = c_{t,k-1}^0(i) - s(i)$.

2. Jointly, all agents build a vector a_t^1 for each possible tuple t.
   $a_t^1 =< c_{t,0}^1, c_{t,1}^1, c_{t,2}^1, ..., c_{t,K}^1 >$.

   To achieve this, each agent A_i computes
   $c_{t,k}^1 = (c_{t,k}^0 + 1) * \prod_{0<\tau\leq K} (c_{t,k}^0 - \tau)(c_{t,k}^0 + \tau)$.

3. Return a_t^1.

Figure 1: Transforming a secret value $c(t) \in \{0, 1, 2, ..., K\}$ to a differential bid vector. The share of $c(t)$ to the agent A_i is $c_i(t)$.

$$t_{j,i}(P) = \prod_{0<k\leq i} (1 - g_{j,k}(P)) \tag{2}$$

Functions $t_{j,i}$ are obtained incrementally as follows:

$$t_{j,0}(P) = 1 \tag{3}$$
$$t_{j,i}(P) = t_{j,i-1} * (1 - g_{j,i}(P)) \tag{4}$$

Once $t_{j,i}$ have been computed for all i, one can compute the index of the value of x_j, namely f_j:

$$f_j(P) = \sum_{i=1}^{|D_j|} i * (g_{j,i}(P) * t_{j,i-1}(P)) \tag{5}$$

Lemma 2.1 *The functions g and f given by Equations 1 and 5 correspond to their definition.*

Proof. The properties can be checked recursively starting with g_1, k and f_1. □

It remains to find an efficient implementation of `satisfiable()`.

Remark 2.1 *It is improbable that one will ever find secure protocols more efficient than generate and test. This is due to the fact that any trimming of a branch reveals information when the "test" operator cannot be implemented securely (see [Sil02c]).*

procedure satisfiable(P)

1. i=first tuple; a=0; b=1; (they do not need to be securely shared, and can be distributed in plain)
2. loop: $a = a + p(t_i) * b$
3. if $i \geq |SS(P)|$ (problem space exhausted), then terminate and return a.
4. $b = b * (1 - p(t_i))$
5. i=next tuple;
6. goto loop

Figure 2: satisfiable(P) when P is already shared. Several such functions for different values of a variable can be computed in parallel in order to exploit common partial results in computing p(t).

Let $SS(P)$ be the ordered set of all tuples in the cross-product of the domains of $P =< X, D, F >$. Each function c in the set of constraints F is a function, $c : SS(P) \rightarrow \{0, 1\}$. The secret parameters of the distributions are the various values $c(t)$ where t is a tuple in D. Let us define the function p, $p : SS(P) \rightarrow \{0, 1\}$, defined as $p(t) = \prod_{c \in F} c(t)$.

$$\texttt{satisfiable}(P) = \sum_{t_i \in SS(P)} (p(t_i) \prod_{k<i} (1 - p(t_k)))$$

Proposition 2.1 *Given the previous definitions of the functions p,* `satisfiable`*(), $g_{i,j}$, and f_i, and a problem P, the vector $\langle v^i_{f_i(P)} \rangle$ defines a solution of P (the first one).*

Proof. See the definition of the functions f. □

To avoid storing all the tuples in memory, the function `satisfiable` will be computed similarly with the functions $g_{i,j}$, namely by using two temporary values (see Figure 2).

Remark 2.2 *The computation of the vector $\langle f_i(P) \rangle$ requires only additions and multiplications and can be easily compiled unto a secure protocol using any of the classic techniques mentioned in this chapter.*

The secret parameters of the computation are the values c(t).

Remark 2.3 *Actually whenever an element of the vector $\langle f_i(P) \rangle$ is 0, the computation can be stopped since P is infeasible.*

The secure algorithm obtained by compiling the computation of $\langle v^i_{f_i(P)} \rangle$ is referred to as Secure Problem Solver. To circumvent the exponential number of

intermediary results and to have an acceptable space requirement, the computation of `satisfiable` should be done tuple after tuple.

Remark 2.4 *Submitted shares of an input value, v, of a tuple in a constraint can be verified by checking that v belongs to $\{0,1\}$ (e.g. using the protocol given in [Bra02b], section 4.1.6). Alternatively, one could also verify that $v(v-1)=0$.*

The algorithm that any agent has to follow here is given in Figure 3.
The drawbacks of this approach are that:

- due to multiplications, this approach offers only n/3-privacy.
- the space complexity is exponential for the best efficiency (to avoid recomputation of all the tuples)
- the best efficiency is n^2 times higher than the efficiency of generate and test.

3 Secure Maximization Solver

Here I describe a technique applicable to maximization problems. Without loss of generality, we consider that each agent is interested in a single function, f. An important first idea is to map securely shared values into differential bid vectors. A technique for achieving this is proposed next. Additional techniques are then proposed for allowing for conditions in these problems.

Finite Discrete Distributed Maximization problems were defined at the beginning of the Background section.

3.1 Intuitive Description

Now I describe a protocol for securely solving a finite discrete distributed maximization problem (A, X, D, F) where all the functions in F return results in the set $\{0, 1, ..., h\}, h \in \mathbb{N}_+$. Many other situations, like negative values, can be mapped to this case. Let $m = h|F|$. The main steps of Secure Finite Distributed Discrete Maximization (SFDDM) are as follows.

1. Each A_k shares function tuples $c^k(i)$ of its problem by distributing shares $c^k_j(i)$ to agents A_j.
2. Verify that value, $c^k(i) = v$, of each submitted function tuple belongs to $\{0, 1, ..., h\}$ (It can be done by checking that $v(v-1)...(v-h)=0$, but this introduces multiplications, therefore other 0-knowledge techniques are preferred).
3. Securely compute the value $c(t)$ of each tuple t of the global problem, using the existing shares. This is achieved by summing up the shares that are projection of the tuple on the corresponding variables. Unfortunately this has exponential complexity.

procedure SecureSatisfaction

1. Securely distribute to each agent A_j encrypted Shamir shares of the feasibility of each local tuple t^i_k of A_i: (t^i_k, s^j_k).
2. Verify the shared values as described in Remark 2.4.
3. Compute `satisfiable(P)`. If P is not satisfiable (result 0), exit. If space complexity allows, store $p(t)$ for all t, or for as many t as possible.
4. $j = 1$
5. Compute in parallel all $g_{j,k}$. The parallel computation can reuse common partial results of the functions $p(t)$, namely for the initial functions in P. The space complexity is then $|D_i|$ and the tuples in $SS(P)$ are enumerated in lexicographical order.

   Compute $t_{j,k}$ for all k.
6. Compute $f_j(P)$.
7. Announce $f_j(P)$ to the owners of the x_j variable (agents that have functions involving x_j). This is done by sending them all the shares that they do not have.
8. The shared secret $f_j(P)$ will be transformed in a differential bid vector of size $|D_i|$ where the element $f_j(P)$ is 1 and all the other elements are 0. This is achieved by the call `value-to-differential-bid-vector(`$f_j(P)$`,`$|D_i|+1$`)`, followed by multiplying each element of the returned vector by $\frac{1}{((-1)^m(m!)^2)}$ where $m = (|D_i|+1)$. The function `value-to-differential-bid-vector` doing this is shown in Figure 1 (also see [Sil02a]). The obtained vector will be used as a function checking that $x_j = v^j_{f_j(P)}$ by multiplying the k-th element of the vector with any tuple for $x_j = k$. Eventual stored values of p(t) can be multiplied with the corresponding value of this new function.
9. if $j = |X|$, then terminate algorithm.
10. $j = j + 1$
11. goto step *4*

Figure 3: Algorithm performed by each agent A_i for finding a solution satisfying conditions where g functions are computed in parallel. It is possible to also compute them sequentially with lower space complexity.

4. Let m be the highest possible value of a tuple (e.g. $h|F|$). Make values into differential bids with the non-zero term y, $y = (-1)^m(m!)^2$. This can

be done by calling the algorithm we propose in Figure 1.

5. Apply any of the known standard secure protocols to determine the winner differential bid (see [Bra02b, Bra02a]).

3.2 Detailed Protocol

Assume a Distributed FDDM with n agents. By s(i) we denote i's share of the secret '1'. We say that a tuple t_i is in a larger tuple t when the projection of t on the variables of t_i yields t_i.

1. Each agent A_i generates secretly a value v^{t_i} for each tuple t_i in FDM(A_i).
2. According to Shamir's scheme, each agent A_i generates secretly a polynome for each secret value v^{t_i}, and generates n secret shares, $v^{t_i}_{(i,j)}$, $1 \leq j \leq n$.
3. Each agent A_i sends to A_j the share $v^{t_i}_{(i,j)}$.
4. Verify that v^{t_i} belongs to the the set $\{0..h\}$, for each submitted value v^{t_i}. Eliminate agents that lie. This verification could be verified by checking that $v^{t_i}(v^{t_i}-1)...(v^{t_i}-h)=0$.
5. All agents compute for each tuple t the global cost $c(t) = \sum_{\forall i, t_i; t_i \in t} v^{t_i}$.

   To achieve this, each agent A_i computes $c_i(t) = \sum_{\forall k, t_k \in t} v^{t_k}_{(k,i)}$.
6. Jointly the agents perform a_t=value-to-differential-bid-vector($c(t)$,m), where $m=|F|h$, and verify with one of the standard techniques [Bra02b, Bra02a] that the result is a correct differential bid with the non-zero element y, $y=(-1)^m(m!)^2$.
7. Run a standard secure protocol for deciding the winner among a_t for all t, considered as differential bids (see [Bra02b, Bra02a]).
8. The results for the agents are revealed and the winner tuple with its value are found (as in the other standard protocols).

Figure 4: SFDDM.

The SFDDM algorithm is detailed in Figure 4. The Algorithm consist of securely distributing shares of the values of different tuples for different agents using Shamir's scheme. The shared values are then summed for each tuple and the total value of each tuple is then securely transformed into a differential bid vector.

Remark 3.1 *Despite the fact that the differential bid vectors obtained for all tuples are then used in a 1-st price auction that can be implemented "fully secure"*

with any of the techniques proposed in [Bra02b, Bra02a], due to our implementation of `value-to-differential-bid-vector` *based on multiplications, a lower threshold has to be adopted for the offered privacy level.*

Remark 3.2 *The fact that several winning tuples can have the same value leads to ties that have to be broken like in [Bra02a]. This is a problem that can strongly increase the complexity of the protocol.*

4 Constrained FDDM problems

In typical optimization problems, besides maximizing some objective function, participants also want to enforce some conditions. In general, such conditions can be modeled by setting the objective function in the excluded regions to $-\infty$. Unfortunately, this technique cannot be applied like this in our case due to the finiteness of our domains.

4.1 Constrained FDDM problems with extended domains

Following the standard approach to Constrained FDDM problems we would like to use $-\infty$. We notice however that $-h(|F|-1)$ is sufficient due to the known upper bound of a value. Constrained FDDM problems can therefore be solved by modeling hard constraints with functions mapping forbidden regions to $-h|F|$ or $-h(|F|-1)$.

The previously described algorithm can be used by mapping the domains $[-h|F|, h]$ into $[0, (h+1)|F|]$.

Remark 4.1 *The complexity of the solved problem increases from* $O(h^{|X|})$ *to* $O((h(|F|+1)^{|X|})$.

Remark 4.2 *When the problem is over-constrained, it is required at the end of the auction protocol to destroy solutions with values less than* $h|F|^2$ *(the value to which 0's are mapped).*

Remark 4.3 *This solution would make sense especially if the multiplications currently used in the proposed SFDDM protocol will be successfully removed by further research, such that the technique could be implemented "fully privately" (without privacy thresholds).*

4.2 Constrained FDDM problems with additional multipliers

A more efficient solution is obtained by using specific feasibility values instead of domain extensions. For each tuple t in a function f, each agent A_i has to securely generate and distribute two shared secrets: $f_i(t)$ and $\phi_i(t)$. Constraining functions ϕ are defined as:

1. Each agent A_i generates secretly a value v^{t_i} for each tuple t_i in FDM(A_i), and a feasibility value $\phi_i(t_i)$.

2. According to Shamir's scheme, each agent A_i generates secretly a polynome for each secret value v^{t_i} respectively. $\phi_i(t_i)$, and generates n secret shares, $v^{t_i}_{(i,j)}$ respectively. $\phi_i(t_i)_{(i,j)}$, $1 \leq j \leq n$.

3. Each agent A_i sends to A_j the shares $v^{t_i}_{(i,j)}$ and $\phi_i(t_i)_{(i,j)}$.

4. Verify that $\phi_i(t_i)_{(i,j)}$ belongs to the set $\{0,1\}$ (e.g. as in the algorithm SecureSatisfaction).

5. Verify that v^{t_i} belongs to the set $\{0..h\}$, for each submitted value v^{t_i}. Eliminate agents that lie. This verification could be verified by checking that $v^{t_i}(v^{t_i}-1)...(v^{t_i}-h)=0$.

6. All agents compute for each tuple t the global cost $c(t) = \sum_{\forall i, t_i; t_i \in t} v^{t_i}$. To achieve this, each agent A_i computes $c_i(t) = \sum_{\forall k, t_k \in t} v^{t_k}_{(k,i)}$.

7. The secret value is than multiplied with the secret multipliers: $c(t) = c(t) \prod_{\forall i, t_i; t_i \in t} \phi_i(t_i)$.

8. Jointly the agents perform a_t=value-to-differential-bid-vector($c(t)$,m), where $m=|F|h$, and verify with one of the standard techniques [Bra02b, Bra02a] that the result is a correct differential bid with the non-zero element y, $y=(-1)^m(m!)^2$.

9. Run a standard secure protocol for deciding the winner among a_t for all t, considered as differential bids (see [Bra02b, Bra02a]).

10. The results for the agents are revealed and the winner tuple with its value are found (as in the other standard protocols).

Figure 5: SFDDM for constrained FDDM problems.

$$\phi_i(t) = \begin{cases} 1 & \text{if } t \text{ is feasible for } A_i \\ 0 & \text{if } t \text{ is infeasible for } A_i \end{cases}$$

The optimized functions f maintain their semantic and take values between 1 and h, taking any value (e.g. 0) when the tuple is infeasible. A detailed revision of the SFDDM protocol that includes multipliers is given in Figure 5.

Remark 4.4 *The main drawback of this technique is that multiplications seem to necessarily require a low threshold ($|A|/2$) on the achieved level of privacy.*

Remark 4.5 *The values 0 in the obtained global problem were reserved for*

infeasibility. Winners with value 0 can be disabled, but actually, no secret is lost when an infeasible tuple is revealed in a problem known to be over-constrained.

5 Summary

In this article we first describe finite distributed discrete maximization problems. We then show how classic secure protocols (for addition and multiplication) can be applied to finite distributed discrete maximization problems (FDDMs).

Besides the *SecureSatisfaction* algorithm for solving constrained problems with constant objective functions, we also propose algorithms for both unconstrained and constrained FDDMs. An important ingredient that we propose to enable these protocols is a functions that transforms a securely shared value into a securely shared differential bid vector.

While the space complexity of the *SecureSatisfaction* algorithm is acceptable, the algorithms for solving FDDMs have to simultaneously store in memory the whole problem space. Further research may be able to solve this problem.

The privacy level offered by our protocols is currently limited to $(|A|/3)$-privacy. Nevertheless, due to the fact that most parts of these protocols can be implemented in a "fully private" way (namely without a threshold), we consider it important to continue research for implementing the currently most problematic part (transforming shared secret values into shared secret differential bid vectors) in a secure way (e.g. without multiplications).

References

[AS02] M. Abe and K. Suzuki. M+1-st price auction using homomorphic encryption. In Springer, editor, *5th IC on PKC*, volume 2274 of *LNCS*, pages 115–224, 2002.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[Bra02a] Felix Brandt. Fully private auctions in a constant number of rounds. preliminary draft, August 2002.

[Bra02b] Felix Brandt. A verifiable, bidder-resolved auction protocol. In L.Korba R.Falcone, editor, *Deception, Fraud and Trust in Agent Societies (AAMAS-W5)*, pages 18,25, Bologna, July 15 2002.

[CCD88a] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 11–19, Chicago, 1988.

[CCD88b] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In Springer-Verlag, editor, *Proc. CRYPTO 87, LNCS 293*, page 462, 1988.

[GB96] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. MIT, July 1996.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.

[LAN02] Helger Lipmaa, N. Asokan, and Valtteri Niemi. Secure Vickrey Auctions without Threshold Trust. In *Financial Cryptography 2002*, Lecture Notes in Computer Science, Southhampton Beach, Bermuda, 11–14 March 2002. Springer-Verlag. To appear.

[SF02] Marius-Călin Silaghi and Boi Faltings. Self reordering for security in generalized english auctions. In *AAMAS*, July 2002.

[Sil02a] Marius-Călin Silaghi. On securely solving distributed CS(O)Ps. Technical Report IC-55-2002, EPFL, July 2002. http://icwww.epfl.ch.

[Sil02b] Marius-Călin Silaghi. *Privacy with Cryptographic Protocols. Asynchronously Solving Distributed Problems with Privacy Requirements, Chapter 15.* 2601, Swiss Federal Institute of Technology (EPFL), CH-1015 Ecublens, June 27, 2002. http://www.cs.fit.edu/~msilaghi/teza/chapter15.pdf.

[Sil02c] Marius-Călin Silaghi. *Problems with a secure test operator. Asynchronously Solving Distributed Problems with Privacy Requirements, Annex B.* 2601, Swiss Federal Institute of Technology (EPFL), CH-1015 Ecublens, June 27, 2002. http://www.cs.fit.edu/~msilaghi/teza/chapter23.pdf.

[SM] D.X. Song and J. Millen. Secure auctions in a publish/subscribe system. http://www.csl.sri.com/users/millen/.