Using Secure DisCSP Solvers for Generalized Vickrey Auctions*

Complete and Stochastic Secure Techniques

Marius C. Silaghi
Florida Institute of Technology
msilaghi@fit.edu

Abstract

Within incentive auctions, several bidders cooperate for clearing a set of offers or requests formalized by an auctioneer, ensuring that each participant cannot do better than by inputing his true utility.

We use a distributed weighted constraint satisfaction (DisWCSP) framework where the actual constraints are secrets that are not known by any agent. They are defined by a set of functions on the secret inputs from all agents. The solution is also kept secret and each agent learns just the result of applying an agreed function on the solution. We show how to apply this framework for modeling and solving General Vickrey Auctions (GVAs). Solutions based on secure complete algorithms as well as solutions using faster secure stochastic algorithms are proposed.

1 Introduction

Within incentive auctions, several bidders cooperate for clearing a set of offers or requests formalized by an auctioneer. The allocation mechanism is designed to ensure that each participant cannot do better than by inputing his true utility. This increases the social welfare by efficient allocations, and for one item auctions it is proven to have similar outcomes as the traditional English Auctions. A previous attempt to solve auctions using distributed constraint frameworks is described in [SF02], but it did not provide incentiveness.

A constraint satisfaction problem (CSP) is modeled as a set of variables and a set of constraints on the possible values of those variables. The CSP problem consists in finding assignments for those variables with values from their domains such that all constraints are satisfied. The CSP techniques require every participant to reveal its

*This article is an adapted and improved extract from FIT Technical Report CS-2004-11 [Sil04a], [SZB04] and 2002,2003, and 2004 patents.

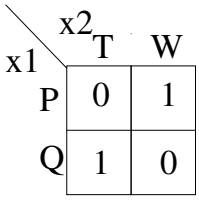

Figure 1: A constraint between two variables, place (x_1) $x_1 \in \{Paris(P), Quebec(Q)\}$, and time ($x_2$) $x_2 \in \{Tuesday(T), Wednesday(W)\}$. The 0s mark rejected tuples. I.e. this constraint allows only the pairs (P,W) and (Q,T), and can be written $\{(P,W),(Q,T)\}$.

preferences (e.g. to a trusted server), to compute the solution. Therefore, they apply only when the participants accept to reveal their preferences to the trusted party.

There exist frameworks and techniques to model and solve distributed CSPs (DisCSPs) with privacy requirements, namely when the domains of the variables are private to agents [YDIK98, MJ00], or when the constraints are private to agents [SSHF00, Sil03, SR04].

In this article we present a recent framework [SAZB05] for the distributed constraint satisfaction problems. That framework can model naturally existing distributed constraint satisfaction problems, and also all necessary steps for incentive auctions [YS03, YS04, Fal04]. The new framework assumes that the constraints are not necessarily known to absolutely any agent but they are computable from secret inputs, with known functions. These functions use secret inputs provided securely by the different participants. Similarly, the final assignments are secret and each agent can retrieve just the result of applying some agreed function on the secret solution.

We start introducing formally the CSP problem.

CSP. A *constraint satisfaction problem* (CSP) is defined by three sets: (X, D, C). $X = \{x_1, ..., x_m\}$ is a set of variables and $D = \{D_1, ..., D_m\}$ is a set of finite domains such that x_i can take values only from $D_i = \{v^i_1, ..., v^i_{d_i}\}$. $C = \{\phi_1, ..., \phi_c\}$ is a set of constraints. A constraint ϕ_i limits the legality of each combination of assignments to the variables of an ordered subset X_i of the variables in X, $X_i \subseteq X$. An assignment is a pair $\langle x_i, v^i_k \rangle$ meaning that the variable x_i is assigned the value v^i_k.

A tuple is an ordered set. The projection of a tuple ϵ of assignments over a tuple of variables X_i is denoted $\epsilon_{|_{X_i}}$. A solution of a CSP (X,D,C) is a tuple of assignments, $\epsilon*$, with one assignment for each variable in X such that each $\phi_i \in C$ is satisfied by $\epsilon*_{|_{X_i}}$. The search space of a CSP is the Cartesian product of the domains of its variables.

Example 1 *In a CSP, one has to find a place (x_1) and time (x_2) for meeting. x_1 is either Paris (P) or Quebec (Q), i.e. D_1=$\{P, Q\}$. x_2 is either Tuesday (T) or Wednesday (W), i.e. D_2=$\{T, W\}$. There are two constraints: ϕ_1=$\{(P,W),(Q,T)\}$, and ϕ_2=$\{(P,W),(Q,T),(Q,W)\}$. ϕ_1 is depicted in Figure 1. The problem is to find values for x_1 and x_2 satisfying both ϕ_1 and ϕ_2.*

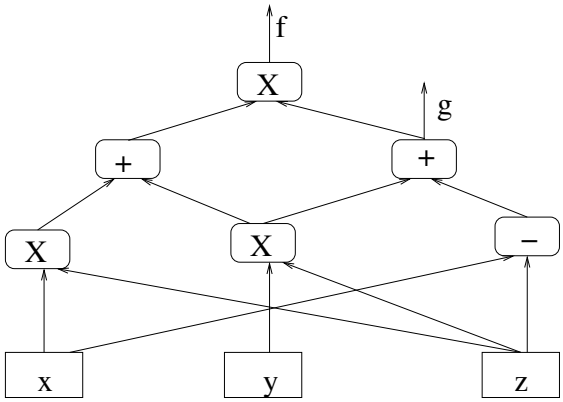

Figure 2: An arithmetic circuit, $g = yz + (x - z)$ and $f{=}(xz + yz)g$. Each input can be the secret of some participant. The output may not be revealed to all participants. All intermediary values remain secret to everybody.

We consider that a set of participants are the source of such CSPs and one has to find agreements for a solution, from the set of possible alternatives, that satisfies a set of (secret) requirements of the participants. This view suggests a concept of a distributed CSP. Several frameworks were proposed so far for Distributed Constraint Satisfaction [ZM91, CDK91, YSH02a, MJ00, SAZB05]. Some versions consider that each agent owns a constraint of the CSP [ZM91, SGM96]. This constraint could model the private information of the agent [SSHF00]. Other versions consider that each agent owns the domain of a variable while the constraints are shared [YDIK98]. The secret domains can also model some private constraints of the agent.

We show how to use complete secure distributed weighted CSP solvers for addressing GVA. In [SF05] we propose secure stochastic algorithms for solving Distributed CSPs. We show that (taking some precautions to always explore the same subset of the search space) such stochastic algorithms can be used to provide an incomplete but secure solver for large GVA problems.

2 Background

Our techniques here apply only to problems whose constraints and outputs can be represented as first order logic expressions, or as arithmetic circuits on inputs. Actually, we proposed a procedure to translate first order logic definitions of constraints/outputs into arithmetic circuits [SZB04]. In the following we introduce arithmetic circuits and a short overview of the literature and techniques that made them relevant.

2.1 Secure Arithmetic Circuit Evaluation

Secure multi-party computations can simulate any arithmetic circuit [BOGW88] or boolean circuit [Kil88, Gol04] evaluation. An *arithmetic circuit* can be intuitively imagined as a directed graph without cycles where each node is described either by an addition/subtraction or by a multiplication operator (see Figure 2). Each leaf is a constant. In a secure arithmetic circuit evaluation, a set of participants perform the operations of an arithmetic circuit over some inputs, each input being either public or an (encrypted/shared) secret of one of them. The result of the arithmetic circuit are the values of some predefined nodes. The protocol can be designed to reveal the result to only a subset of the agents, while none of them learns anything about intermediary values. One says that the multi-party computation *simulates* the evaluation of the arithmetic circuit. A *boolean circuit* is similar, just that the leafs are boolean truth values, false or true, often represented as 0 and 1. The rest of the nodes are boolean operators like AND or XOR. A function does not have to be represented in this form to be solvable using general secure arithmetic circuit evaluation. It only needs to have such an equivalent representation. For example, the operation $\sum_{i=B}^{E} f(i)$ is an arithmetic circuit if B and E are public constants and $f(i)$ is an arithmetic circuit. The same is true about $\prod_{i=B}^{E} f(i)$. Such constructs are useful when designing arithmetic circuits. Secure protocols for evaluating such circuits use [BOGW88] or oblivious transfer.

Using the techniques for secure addition and multiplication one can build secure techniques for comparison ($cmp(x, y)$ returning 1 if $x < y$ and 0 otherwise), and for Kronecker's delta ($\delta_K(x, y)$ returning 1 if $x = y$ and 0 otherwise).

3 Privacy concepts

Definition 1 ([BOGW88]) *A multi-party computation is t-private if an attacker controlling any at most t participants cannot learn anything from the computation, except from what can be inferred from its outputs and prior knowledge.*

Given secret constraints σ the prior knowledge Γ of the t colluders and a multi-party computation process Π with answer α, the technique is t-private if the probability distribution of the secrets is conditionally independent on Π given answer α and knowledge Γ.

$$P(\sigma|\alpha, \Gamma, \Pi) = P(\sigma|\alpha, \Gamma)$$

However, many algorithms provide answers α that contain more information than what is actually needed. We typically decompose α in a desired data α^* and an algorithmic dependent unrequested data $\overline{\alpha}$. For DisCSPs the desired data is an assignment of some variables satisfying constraints, and the unrequested data consists of peculiarities of the used algorithm $\mathcal{A}$ (e.g., the solution is the first/last in some known order on alternatives).

function *(S)MPC-DisCSP1(T,(X,D,C))*
SHUFFLE(X,D,C) //using the mixnet;
for *i=1 to T* **do**
S[i]=$\prod_{\phi \in C} \phi(\epsilon_i)$;
F=DisCSP1(T,(X,D,C));
UNSHUFFLE(F); // Unshuffle each vector in F separately;
set solution F to 0 with probability p; //optional (see [Sil05a, SF05]);
return F;

Algorithm 1: Stochastic MPC-DisCSP1 for solving a CSP (X, D, C) with k alternatives and exploring T alternatives. When $T = |D_1 \times ... \times D_m|$ then the algorithm is complete. The function DisCSP1 is the arithmetic circuit in Figure 3.

We say that an algorithm $\mathcal{A}$ achieves *maximal t-privacy* if the probability distribution of the secrets is conditionally independent on $\Pi, \mathcal{A}$ and $\overline{\alpha}$ given requested data α^* and prior knowledge Γ.

$$P(\sigma|\alpha, \Gamma, \Pi, \mathcal{A}) = P(\sigma|\alpha^*, \Gamma)$$

For distributed CSPs, maximal t-privacy typically implies the return of uniformly random selected solutions whenever the problem may have more than one solution [SR04].

3.1 Overview of MPC-DisCSP1

In [Sil03] we have proposed a polynomial space multi-party computation technique, called MPC-DisCSP1, that extracts a (non-uniformly) random solution of a distributed CSP [SR04]. MPC-DisCSP1 is t-private (and can be t-resilient when using the zero-knowledge proofs in [Sil05b]) and uses general multi-party computation building blocks.

MPC-DisCSP1 implements DisCSP() in five phases (also see Algorithm 1):

1. Share the secret parameters of the input DisCSP using secret sharing.
2. The shared DisCSP problem is shuffled in a cooperative way, reordering values (and eventually variables), with a permutation that is not known to anybody [Sil03, Sil05b].
3. A computation returning the first solution of the DisCSP where the operations performed by agents are independent of the input secrets (to avoid leaking the secrets), is executed by simulating arithmetic circuits evaluation. This is shown in Figure 3.
4. The solution (represented as unary constraints on domains) returned at Step 3 is translated into the initial problem formulation using a transformation that is inverse of the shuffling at Step 2 [Sil05b].

$$
\begin{aligned}
p(\epsilon, P) &= \prod_{\phi \in C} \phi(\epsilon) \\
\texttt{satisfiable}(P) &= 1 - \delta_K(0, \sum_{\epsilon_i \in [\epsilon_1 ... \epsilon_T]} p(\epsilon_i, P)) \\
g_{i,j}(P) &= \texttt{satisfiable}(P \cup \{x_i = j\} \cup_{k<i} (x_k = f_k(P))) \\
f_j(P) &= \sum_{i=1}^{|D_j|} i * (g_{j,i}(P) * \delta_K(0, \sum_{k<i} g_{j,k}(P)))
\end{aligned}
$$

Figure 3: Arithmetic circuit DisCSP1 for a CSP $P = (X, D, C)$. The result is the vector of vectors $\{\{\delta_K(f_i, j)\}_{j \in [1..|D_i|]}\}_{i \in [1..m]}$. Versions with other primitives appear in [Sil03, Sil04b]

5. Construct the solution from its secret shares.

It is also possible and very simple to find all solutions [HCN+01]. However, when only a single solution is needed, this leaks a lot of information. At Step 3, MPC-DisCSP1 requires a computation whose cost is independent of the input, since otherwise the users can learn things like: *The returned solution is the only one, being found after unsuccessfully checking all other tuples, all other tuples being infeasible.* Since the computation has to be independent of the problem details, its cost is exponential (at least as long as nobody proves P=NP).

Note that other alternative techniques are available, notably MPC-DisCSP1 [Sil03], MPC-DisCSP2 [SM04], MPC-DisCSP3 [Sil04b], and MPC-DisCSP4 [Sil05a]. We will call them generically MPC-DisCSPx.

In [SM04] we show how to transform any of these techniques in a solver for distributed weighted constraint satisfaction, denoted accordingly MPC-DisWCSPx. The modification consists only in a change to Step 3, namely computing W times (where W is the number of possible total weights) an arithmetic circuit similar to the one in the Step 3 of the corresponding MPC-DisCSPx, and performing an additional secure computation integrating their result. Also, for the computation corresponding to a possible total weight B, the function $p(\epsilon_i, P)$ is replaced with $p(\epsilon_i, P) = \delta_K(B, \sum_{\phi \in C} \phi(\epsilon))$.

3.1.1 Complete versus Stochastic Techniques

The algorithm MPC-DisCSP1 is complete when the parameter T is set to the size of the search space. If T is smaller than $|D_1 \times ... \times D_m|$ then the algorithm is incomplete. Namely it will return a solution picked randomly from T unknown randomly selected

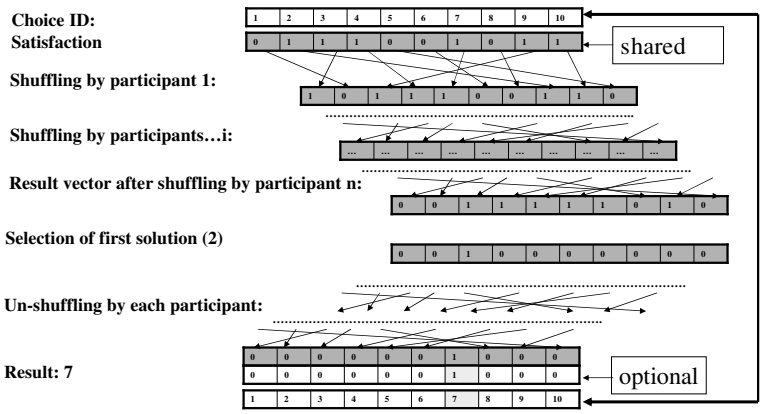

Figure 4: MPC-DisCSP4 using mix-nets

elements of the search space. If no solution exists among those T elements then a *don't know* answer is returned. Note that the time cost of the computation is linear in this parameter T (assuming that the constraints have a bounded arity, limiting the cost of the shuffle).

As follows, a stochastic algorithm is obtained wherein one can address a problem of any size in any predefined available amount of time (larger than the time for the shuffle) by choosing the T parameter accordingly. If no solution is found after exploring the T elements then the returned failure has the meaning *don't know*. The obtained algorithm is called secure Stochastic Multi-party Computation for solving DisCSPs (SMPC-DisCSP1). Similar stochastic algorithms can be obtained from the other versions of secure DisCSP solvers, based both on mix-nets and arithmetic circuits [SF05].

3.2 Overview of MPC-DisCSP4

In [Sil05a] we have proposed a multi-party computation technique, called MPC-DisCSP4, that extracts a uniformly random solution of a distributed CSP.

MPC-DisCSP4 implements DisCSP() in five phases:

1. Share the secret parameters of the input DisCSP using secret sharing. The value of each publicly possible assignment (allocation in GVA) is securely evaluated into a vector of shared secrets S.

2. The shared vector S is shuffled in a cooperative way with a permutation that is not known to anybody [Sil05b].

3. An arithmetic circuit is used to set all shared values in the shuffled S to 0, except for the first non-zero one.

4. The shared vector obtained at Step 3 is translated into the initial problem formulation using a transformation that is inverse of the shuffling at Step 2 [Sil05b].

5. Construct the solution from its secret shares.

In this article we only address multi-party computations without trusted servers. A family of secure solvers based on trusted servers is proposed in [YSH02a].

4 Distributed CSPs with constraints secret to everybody

Here we show how a distributed weighted CSP framework can model some famous WCSP problems, namely optimizations occurring in clearing incentive auctions.

4.1 DisCSPs with constraints secret to anybody

Here we remind the DisCSP framework in [SAZB05]. It proposes a way to model distributed CSPs, where a constraint is not (necessarily) a secret known to an agent, or public, but can also be a secret unknown to all agents.

A Distributed CSP (DisCSP) is defined by six sets (A, X, D, C, I, O) and an arithmetic structure F. A=$\{A_1, ..., A_n\}$ is a set of agents. X, D, and the solution are defined like for CSPs.

I=$\{I_1, ..., I_n\}$ is a set of secret inputs. I_i is a tuple of α_i secret inputs (defined on a set F) from the agent A_i. Each input I_i belongs to F^{α_i}.

Like for CSPs, C is a set of constraints. There may exist a public constraint in C, ϕ_0, defined by a predicate $\phi_0(\epsilon)$ on tuples of assignments ϵ, known to everybody. However, each constraint ϕ_i, i>0, in C is defined as a set of known predicates $\phi_i(\epsilon, I)$ over the secret inputs I, and the tuples ϵ of assignments to all the variables in a set of variables X_i, $X_i \subseteq X$.

O=$\{o_1, ..., o_n\}$ is the set of outputs to the different agents. $o_i : D_1 \times ... \times D_m \rightarrow F^{\omega_i}$ is a function receiving as parameter a solution and returning ω_i secret outputs (from F) that will be revealed only to the agent A_i.

This framework has been experimented successfully for modeling and solving stable desk-mates problems with 7 participants [SAZB05] (requiring two minutes) and stable marriages problems with 4 participants [Sil04c] (taking a few seconds).

4.2 Distributed Weighted Constraint Satisfaction Problems

Now let us describe the extension of that framework to Distributed Weighted Satisfaction Problems.

Definition 2 *A distributed constraint satisfaction problem (DisWCSP) is defined by six sets (A, X, D, C, I, O), an arithmetic structure F, and a set of acceptable solution qualities B, that can be often represented as an interval $[B_1, B_2]$. A=$\{A_1, ..., A_n\}$ is a set of agents. $X = \{x_1, ..., x_m\}$ is a set of variables and $D = \{D_1, ..., D_m\}$ is a set of finite domains such that x_i can take values only from $D_i = \{v^i_1, ..., v^i_{d_i}\}$. An assignment is a pair $\langle x_i, v^i_k \rangle$ meaning that the variable x_i is assigned the value v^i_k. A tuple is an ordered set. I=$\{I_1, ..., I_n\}$ is a set of secret inputs. I_i is a tuple of α_i secret inputs*

(defined on a set F) from the agent A_i. Each input I_i belongs to F^{α_i}. $C = \{\phi_0, ..., \phi_c\}$ is a set of constraints. A constraint ϕ_i weights the legality of each combination of assignments to the variables of an ordered subset X_i of the variables in X, $X_i \subseteq X$. ϕ_0 is a public constraint defined by a function $\phi_0(\epsilon)$ on tuples of assignments ϵ, known to everybody. Each constraint ϕ_i, i>0, in C is defined as a known function $\phi_i(\epsilon, I)$ over the secret inputs I, and the tuples ϵ of assignments to all the variables in a set of variables X_i, $X_i \subseteq X$. The projection of a tuple ϵ of assignments over a tuple of variables X_i is denoted $\epsilon_{|_{X_i}}$. A solution is $\epsilon* = \underset{\epsilon \in D_1 \times ... \times D_n}{\operatorname{argmin}} \sum_{i=1}^{c} \phi_i(\epsilon_{|_{X_i}})$, if $\sum_{i=1}^{c} \phi_i(\epsilon*_{|_{X_i}}) \in [B_1...B_2]$. O=$\{o_1, ..., o_n\}$ is the set of outputs to the different agents. $o_i : I \times D_1 \times ... \times D_m \rightarrow F^{\omega_i}$ is a function receiving as parameter the inputs and a solution, and returning ω_i secret outputs (from F) that will be revealed only to the agent A_i.

Solvers developed in our previous work require that the functions in sets O and C are input either in first order logic form, or in the form of arithmetic circuits.

The public constraint ϕ_0 can be input into the system using a set of constraints $\{\phi_0^1, \phi_0^2, ...\}$, and the tuples of assignments accepted by ϕ_0 can be obtained separately by each agent, when needed, using any systematic search technique that finds all solutions of a CSP, e.g. backtracking or lookahead algorithms (BT, BM, CBJ, FC, MAC, EMAC, etc.).

5 Incentive Auctions[1]

To clear a combinatorial auction according to the (non-incentive) 1^{st} price criteria when several allocations may be optimal:

1. The participants select as public parameters of the problem a set of variables X where there is a distinct variable for each item to be sold, and the domain of each variable is the set of participants that may own the item at the end of the auction (by winning it or by not selling it). There is a function $\phi_k(\epsilon, I)$ for each participant A_k, which associates to each possible tuple ϵ of assignments of the variables in X, an element of I_k (the bid of A_k for ϵ). The maximum and minimum value of the sum of the bids B_1, B_2, are also enforced by allocating ranges of possible bids to each participant.

   (Alternatively, variables can be associated to agents and their domain is the bid for each possible bundle. Public constraints limmit the possible allocations of bundles to agents.)

2. Each participant decides its secret inputs I_k (bids) for each tuple defining an allocation, by taking into account both the items she acquires and the reservation price of the items she cedes in that allocation.

[1] Patent Pending.

3. The secret inputs are shared with a secret sharing scheme. The weight of each publicly permitted constraint ϕ_k (bid of each agent) is computed securely (typically equal to the input).

4. A solution of the DisWCSP is computed with a secure protocol (e.g. MPC-DisWCSPx) [SM04]. As a result, a shared secret w_i will specify the value of each variable x_i in the selected optimal solution, and w_0 specifies the total weight of the selected solution.

5. The chosen allocation and its total price can be revealed by reconstructing the secrets w_i from shares.

   To reveal the winner allocation only to the participants involved in it, the participants must agree on the functions o_k specifying that each participant learns the allocation of the items that she receives. Also, each participant receives the shares of the variables for items that she is selling, to learn whom to give them. Namely, o_k returns an array such that with m items and n participants, $\forall i, 0 < i \leq m$, if x_i models an item of A_k then $o_k[i] = x_i$, otherwise $o_k[i] = (x_i = k)$. These first order logic predicates are translated in arithmetic circuits as shown in [SZB04]: $(x_i = k)$ (aka $\delta_K(x_i, k)$–Kronecker's delta) becomes $\frac{1}{(k-1)!(n-k)!} \prod_{i=1}^{k-1}(x_i - i) \prod_{i=k+1}^{n}(i - x_i)$. Other ways of computing $\delta_K(x_i, k)$ are given in [Kil05, DFNT05].

6. The exact price p_u to be paid by each agent A_u in this case is the bid of the agent A_u for the solution, and can be made known to a participant A_i with an output $o'_i[u] = \phi_u(\epsilon*, I)$, by the arithmetic circuit: $\sum_k \lambda(\sum_{i=1}^{n}(w_i \prod_{j=1}^{i-1} d_i), \prod_{i=1}^{n} d_i)[k]\phi_u(\epsilon_k, I)$,
   where w_i is the shared secret specifying the index of x_i in the solution and $\lambda(s, d)$ is a function translating a shared secret s into a vector of shared secrets with dimension $d+1$ having a one at index s and 0 elsewhere (see function `value2unaryconstraintX` in [Sil03]). Calling the λ function is not needed when using DisWCSP2, DisWCSP3, or DisWCSP4, since they already have the result as the vector that is the outcome of unshuffling.

   The result is the value of the point product between the vector returned by λ and a vector with the weights (bids) for each tuple, ordered lexicographically.

   The previous arithmetic circuit is usable if each agent provided a separate bid for each possible allocation [Sil02, YS03, NSY04]. Otherwise an appropriate arithmetic circuit has to be designed that is adapted to select the bid of the winning solution from the format of the input of the bidder.

For incentive auctions (using Clarke tax, like GVA), one uses the algorithm for 1^{st} price auctions as a component where the result is not revealed:

function *GVA(T,(X,D,C))*

SHUFFLE(X,D,C) //using the mixnet or arithmetic circuits;
for $b = B_2$ *to* B_1 **do**
  for *i=1 to T* **do**
    for *j=1 to n* **do**
      $c_j[i] = \delta_K(b, \sum_{\phi \in C \setminus \{\phi_j\}} \phi(\epsilon_i))$;
  F[b]=DisCSP1(T,(X,D,C));
  $W_j[b] = 1 - \delta(\sum_{i=1}^{T} c_j[i])$;
W=integrate $F[b]$ to get w_i using technique in [SM04];
for *j=1 to n* **do**
  $w_0[j] = \sum_{b=B_2 to B_1} bW_j[b]\delta_K(0, \sum_{k=b+1}^{B_2} W_j[k])$;
UNSHUFFLE(W); // Unshuffle each vector in W separately;
for $k = 1$ *to* n **do**
  p_k is computed as in Step 6 of 1^{st} price action;
  $w'_0[k] = w_0 - p_k$;
  $VCG_k = w_0[k] - w'_0[k]$;
reveal allocations w_i to interested agents (as in Step 5 of 1^{st} price action);
reveal each A_k its Vickrey-Clarke-Grooves tax VCG_k;

Algorithm 2: GVA for an auction represented as a shared CSP (X,D,C).

GVA

- The winning allocation is computed with the algorithm for 1^{st} price auctions without revealing the results. Namely, at the end of the last step, the price p_k to be paid by each agent A_k is not revealed but it is independantly subtracted from w_0 (the shared secret representing the total weight of the solution to the DisWCSP, as returned by the used MPC-DisWCSPx) obtaining a shared secret $w'_0[k] = w_0 - p_k$.
- Additional n more maximization processes are run solving the DisWCSP, each k-th maximization by not considering the bids of the agent A_k, and recording the obtained w_0 as $w_0[k]$ (skipping the computation of the corresponding allocations, w_i for $i > 0$).

  Separate shuffling/unshuffling is not needed in used MPC-DisWCSPx processing for these additional maximizations, and these computations are preferably done on the shuffled encoding of the problem used at Step 4 in the algorithm computing the winning allocation with 1^{st} price criteria.
- The price (Clarke tax) to be paid by each bidder A_k is given by $w_0[k] - w'_0[k]$.

6 Analysis

We have shown here how to apply secure solvers for Distributed Weighted CSPs to securely clear incentive auctions (in particular GVA). The computational complexity for securely clearing a GVA auction with this technique is given by the cost of solving the DisWCSP, i.e., b times higher than to solve a DisCSP of the same size (where b is the number of possible total weights – aggregated utilities – for an allocation).

For the n rounds of optimisation for finding $w_0[k]$, the cost if nb calls to the *satisfiable* function for $c-1$ constraints.

Remark 1 *The n runs of the solver for computing the elements of $w_0[k]$ will add only $O(nb)$ calls to the* `satisfiable(P)` *function, each of them excluding an agent's constraint (bids).*

Secure Stochastic Techniques One typical way to improve the speed of GVA solvers is to reduce the number of possible bundles to a small predefined set. When possible such a technique speeds our secure algorithm since its cost directly depends on this number (specially when based on MPC-DisCSP4).

Another way of speeding up computations is by using secure stochastic solvers (that explore only a subset of the search space), as SMPC-DisCSP1. However, to ensure individual rationality one has to guarantee that all instances of the optimization (for the computation of each $w_0[k]$) have to be run using only allocations that were considered at the computation of the winning allocation. This can be achieved if in the aforementioned method for solving GVA, the same T is used for all the n computations of the elements of $w_0[k]$.

7 Comparison with related work

There exist many recent algorithms for securely clearing incentive auctions (e.g., [Sil02, YS03, NSY04, YS04]). The technique in [YS04] is based on dynamic programming and therefore can be faster. Our technique differentiates itself by the guarantee that the solution is selected randomly among the optimal allocations. Other techniques did not use distributed CSPs (except indirectly by our previous technique in [Sil02, NSY04], but where the problem of several solutions was not handled).

8 Conclusions

Privacy has been recently stressed in [MJ00, FMW01, WF02, FMG02, YSH02b] as an important goal in designing algorithms for solving DisCSPs. In this article we have investigated how versions of old and famous problems, incentive auctions, can be solved such that the privacy of the participants is guaranteed minimizing even what is leaked by the selected solution. Incentive auctions are a very intense field of research as application of agents and economic theories. Our technique uses secure simulations of

arithmetic circuit evaluations and is therefore information theoretically secure whenever no majority of the participants colludes to find the secret of the others, and when all agents follow the protocol. Since we are restricted to arithmetic circuits, full privacy is achievable with computational security. We also show how faster but incomplete secure solvers can be provided using the secure stochastic techniques proposed in [SF05], namely by making sure that the same subset of the search space is explored by the different optimization instances performed by the GVA solver.

References

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Widgerson. Completeness theorems for non-cryptographic fault-tolerant distributed computating. In *STOC*, pages 1–10, 1988.

[CDK91] Z. Collin, R. Dechter, and S. Katz. On the feasibility of distributed constraint satisfaction. In *Proceedings of IJCAI 1991*, pages 318–324, 1991.

[DFNT05] I. Damgård, M. Fitzi, J. B. Nielsen, and T. Toft. How to split a shared number into bits in constant round and unconditionally secure. Cryptology ePrint Archive, Report 2005/140, 2005.

[Fal04] B. Faltings. Incentive-compatible social choice. In *IAT'04*, 2004.

[FMG02] B. Faltings and S. Macho-Gonzalez. Open constraint satisfaction. In *CP*, 2002.

[FMW01] E.C. Freuder, M. Minca, and R.J. Wallace. Privacy/efficiency tradeoffs in distributed meeting scheduling by constraint-based agents. In *Proc. IJCAI DCR*, pages 63–72, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography*, volume 2. Cambridge, 2004.

[HCN+01] T Herlea, J. Claessens, G. Neven, F. Piessens, B. Preneel, and B. Decker. On securely scheduling a meeting. In *Proc. of IFIP SEC*, pages 183–198, 2001.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. of ACM Symposium on Theory of Computing*, pages 20–31, 1988.

[Kil05] Eike Kiltz. Unconditionally secure constant round multi-party computation for equality, comparison, bits and exponentiation. Cryptology ePrint Archive, Report 2005/066, 2005. `http://eprint.iacr.org`.

[MJ00] P. Meseguer and M. Jiménez. Distributed forward checking. In *CP'2000 Distributed Constraint Satisfaction Workshop*, 2000.

[NSY04] J. Nzouonta, M.-C. Silaghi, and M. Yokoo. Secure computation for combinatorial auctions and market exchanges. In *AAMAS*, pages 1398–1399, 2004.

[SAZB05] M.-C. Silaghi, A. Abhyankar, M. Zanker, and R. Bartak. Desk-mates (stable matching) with privacy of preferences, and a new distributed csp framework. In *FLAIRS'05*, 2005.

[SF02] M.-C. Silaghi and B. Faltings. Self reordering for security in generalized english auctions. In *AAMAS*, July 2002.

[SF05] M.-C. Silaghi and G. Friedrich. Secure stochastic multi-party computation for combinatorial problems. Technical Report CS-2005-14, FIT, 2005.

[SGM96] G. Solotorevsky, E. Gudes, and A. Meisels. Algorithms for solving distributed constraint satisfaction problems (DCSPs). In *AIPS96*, 1996.

[Sil02] M.-C. Silaghi. An algorithm applicable to clearing combinatorial exchanges. Technical Report TR-CS-2002-14, FIT, September 2002.

[Sil03] M.-C. Silaghi. Solving a distributed CSP with cryptographic multi-party computations, without revealing constraints and without involving trusted servers. In *IJCAI-DCR*, 2003.

[Sil04a] M.-C. Silaghi. Incentive auctions and stable marriages problems solved with n/2-privacy of human preferences. Technical Report CS-2004-11, FIT, 2004.

[Sil04b] M.-C. Silaghi. Meeting scheduling system guaranteeing n/2-privacy and resistant to statistical analysis (applicable to any DisCSP). In *3rd IC on Web Intelligence*, pages 711–715, 2004.

[Sil04c] M.-C. Silaghi. Secure multi-party computation (SMC) programming language. `http://www.cs.fit.edu/~msilaghi/SMC/`, 2004.

[Sil05a] M.-C. Silaghi. Hiding absence of solution for a discsp. In *FLAIRS'05*, 2005.

[Sil05b] M.-C. Silaghi. Zero-knowledge proofs for mix-nets of secret shares and a version of elgamal with modular homomorphism. Cryptology ePrint Archive, Report 2005/079, 2005. `http://eprint.iacr.org/`.

[SM04] M.-C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *3rd IC on Intelligent Agent Technology*, pages 531–535, 2004.

[SR04] M.-C. Silaghi and V. Rajeshirke. The effect of policies for selecting the solution of a DisCSP on privacy loss. In *AAMAS*, pages 1396–1397, 2004.

[SSHF00] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proc. of AAAI2000*, pages 917–922, Austin, August 2000.

[SZB04] M.-C. Silaghi, M. Zanker, and R. Bartak. Desk-mates (stable matching) with privacy of preferences, and a new distributed csp framework. In *Proc. of CP'2004 Immediate Applications of Constraint Programming Workshop*, 2004.

[WF02] R.J. Wallace and E.C. Freuder. Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss. In *DCR*, pages 176–182, 2002.

[YDIK98] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE TKDE*, 10(5):673–685, 1998.

[YS03] M. Yokoo and K. Suzuki. Secure generalized vickrey auction using homomorphic encryption. In *FC*, 2003.

[YS04] M. Yokoo and K. Suzuki. Generalized Vickrey Auctions without Third-Party Servers. In *FC04*, 2004.

[YSH02a] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *Proc. of the AAMAS-02 DCR Workshop*, Bologna, July 2002.

[YSH02b] M. Yokoo, K. Suzuki, and K. Hirayama. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*, 2002.

[ZM91] Y. Zhang and A. K. Mackworth. Parallel and distributed algorithms for finite constraint satisfaction problems. In *Proc. of Third IEEE Symposium on Parallel and Distributed Processing*, pages 394–397, 1991.