Desk-mates (Stable Matching) with Privacy of Preferences, and a new Distributed CSP Framework

Marius C. Silaghi and **Amit Abhyankar**
Florida Institute of Technology, USA
{msilaghi,aabhyank}@fit.edu

Markus Zanker
Universität Klagenfurt, Austria
markus@ifit.uni-klu.ac.at

Roman Barták
Charles University, Czech Republic
bartak@kti.mff.cuni.cz

Abstract

The desk-mates matcher application places students in pairs of two for working in projects (similar to the well known problems of stable matchings or stable roommates). Each of the students has a (hopefully stable) secret preference between every two colleagues. The participants want to find an allocation satisfying their secret preferences and without leaking any of these secret preferences, except for what a participant can infer from the identity of the partner that was recommended to her.

The peculiarities of the above problem require solvers based on old distributed CSP frameworks to use models whose search spaces are higher than those in centralized solvers, with bad effects on efficiency. Therefore we introduce a new distributed constraint satisfaction (DisCSP) framework where the actual constraints are secrets that are not known by any agent. They are defined by a set of functions on some 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. The expressiveness of the new framework is shown to improve the efficiency ($O(2^{m^3-\log(m)})$ times) in modeling and solving the aforementioned problem with m participants. We show how to extend our previous techniques to solve securely problems modeled with the new formalism, and exemplify with the problem in the title. An experimental implementation in the form of an applet-based solver is available.

Introduction

The desk-mates matcher application groups a set of students in stable working teams of two, such that whenever one student wants to change her partner for a third one, the third one prefers her current partner to the change (similar to stable matchings or stable roommates (Irving & Manlove 2002)). It is desirable for these teams to be stable for the duration of the project. Otherwise discontinuities and changes may reduce the efficiency of the learning process. The students have a preference between any pair of potential partners, and between working with any given partner or working alone. Some students insist to work alone, and it is typically difficult for students to refuse other's offers of partnership. In fact students sometimes prefer to keep private their preferences between colleagues, to avoid hurting others. We decided that it is needed to provide students with a support in solving these situations. We therefore built a web-application that insures their privacy using cryptographic solvers of distributed constraint satisfaction problems, as proposed in this paper. Note that our approach can also be applied to combinatorial auctions or to the problem of distributed configuration of products based on components from several providers, with secret configuration requirements, etc.

Versions of these problems, without privacy requirements, have been long known and studied. It is an example of constraint satisfaction problem (CSP) (Gent & Prosser 2002).[1] A CSP is described by 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 centralized CSP techniques require every eventual participant to reveal its 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 (Yokoo *et al.* 1998; Meseguer & Jiménez 2000), or when the constraints are private to agents (Silaghi, Sam-Haroud, & Faltings 2000). However, the desk-mates problem seems not to be modeled efficiently (i.e. with a reduced search space) with any of the two known types of distributed CSP frameworks. This is because the private data of these problems (the preferences) do not *directly* constrain the allocation of the natural shared resources (the matching). An indirect relation exist with such a constraint. Aditional (redundant) variables would need to be introduced in the system, modeling the secret preferences, but reducing efficiency.

In this article we propose a new framework for the distributed constraint satisfaction problems that avoids the aforementioned redundant variables. It can model naturally existing distributed constraint satisfaction problems, and also the desk-mates (stable matchings problems) with

[1] Operations research has also provided very efficient solutions to some instances without privacy.

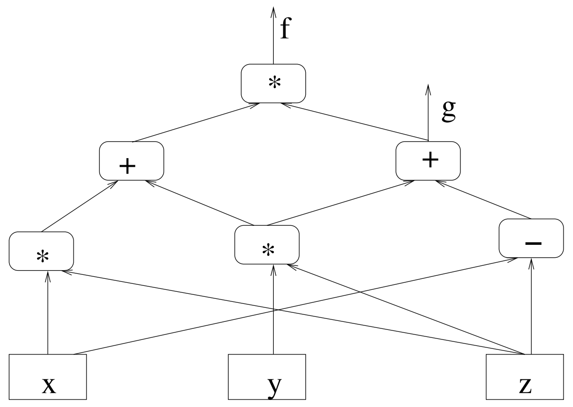

Figure 1: An arithmetic circuit, $g = yz + (x - z)$ and $f{=}(xz+yz)g$. Each input can be the secret of some participant. The output may not be revealed to all participants. All intermediary values remain secret to everybody.

private preferences. The new framework assumes that the constraints are not known to absolutely any agent but they are computable from secret inputs provided securely by the different participants, by applying public functions on them. Similarly, the final assignments are secret and each agent can retrieve just the result of applying some agreed function on the secret solution.

We also show how secure multi-party computation techniques that we have recently developed for solving DisCSPs with private constraints can be extended to solve problems described in the new framework.

Background

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

Secure Arithmetic Circuit Evaluation

Secure multi-party computations can simulate any arithmetic circuit or boolean circuit evaluation (Ben-Or, Goldwasser, & Widgerson 1988). 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 1). 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.

There must be some machinery to compute the result of the circuit from the inputs. However, existing techniques allow for the secret inputs not to be revealed to this machinery. Namely the machinery works only with encrypted secrets that it cannot decrypt.

Let us give a simple example of a secure computation. Three faculty members, A_0, A_1, A_2, want to compute the average of their wages x_0, x_1, x_2 without revealing any one of them. Each professor A_i generates two random numbers, $r_{i,-1}, r_{i,1}$. A_i sends each $r_{i,j}$ to $A_{(i+j) \mod 3}$ through a secure channel. Each professor A_i computes now

$$r_i = x_i + r_{((i+1) \mod 3),-1} + r_{((i-1) \mod 3),1} - r_{i,-1} - r_{i,1}$$

and publishes r_i. Their average wage is $(r_0{+}r_1{+}r_2)/3$ and no particular wage is revealed except if communication is intercepted, or if two professors collude. It was shown in (Silaghi 2003) how such computations can be extended to solve general CSPs. r_i is what we call secret input in the paper and $(r_0{+}r_1{+}r_2)/3$ was here the equivalent to the machinery proposed for computing the constraints.

In the simple example above, the secrets were shared among participants in the computation by using random numbers. A slightly more careful secret sharing technique is famous due to its properties that allow easily for some more general computations. This is Shamir's secret sharing that we will present later.

Distributed CSPs with constraints secret to everybody

In this article we redefine the distributed CSP framework, aiming to model efficiently (i.e. with a reduced search space) the distribution of some famous CSP problems, namely the stable matching problems (e.g. the desk-mates problem). Let us now describe how the stable matching problem can be modelled using a traditional CSP and a traditional DisCSP with secure constraints.

Modelling the desk-mates problem with a traditional CSP The desk-mates problem consists in placing a set of students $A = \{A_1, ..., A_m\}$ in teams of two (or two-seats desks), such that if any student A_i prefers a colleague A_j to the desk-mate selected for her, then A_j prefers her current desk-mate to A_i.

A way of modeling the desk-mates problem as a CSP is to have one variable x_i for each student A_i specifying the index of the desk-mate assigned to her by the solution, or specifying i, the index of A_i itself, if she remains alone. The constraints are obtained by preprocessing the input from participants about their preferences. The fact that a student A_i prefers A_u to A_v is specified by the first order logic predicate $P_{A_i}(u,v)$. There is a constraint ϕ^{ij} between every pair of distinct variables x_i and x_j. In first order logic notation, the constraint between each two variables x_i and x_j is:

$$\begin{aligned}\forall x_i, x_j : \phi^{ij}(x_i,x_j) \overset{\text{def}}{=} & (P_{A_i}(x_j,x_i) \Rightarrow P_{A_{x_j}}(j,i)) \\ & \wedge(P_{A_j}(x_i,x_j) \Rightarrow P_{A_{x_i}}(i,j)) \wedge \\ & ((x_i = j) \Leftrightarrow (x_j = i)) \qquad (1)\end{aligned}$$

Read: *For each pair of participants A_i, A_j, (and corresponding variables x_i and x_j) there is a constraint ϕ^{ij} that allows a pair of assignments to these variables only if:*

- *the fact that A_i prefers the participant assigned to A_j (A_{x_j}) to her own match A_{x_i} implies that:*

  the agent assigned by these assignments to A_j (A_{x_j}), prefers the agent A_j to the agent A_i.
- *the fact that A_j prefers the participant assigned to A_i (A_{x_i}) to her own match A_{x_j} implies that*

  the agent assigned by these assignments to A_i (A_{x_i}), prefers the agent A_i to the agent A_j.
- *A_j is the match of A_i only if A_i is the match of A_j.*

Note that this model subsumes the constraints: $\forall i,j : x_i \neq x_j$. The main complication with this kind of CSPs is that the constraints are functions of secrets that cannot be easily elicited from the participants. Distributed CSP frameworks are meant to address such problems.

Modeling the desk-mates problem with DisCSPs with secret constraints that are known to some agents. One can model the desk-mates problem with secret constraints known to some agents (Zhang & Mackworth 1991; Silaghi, Sam-Haroud, & Faltings 2000) by choosing as variables, $x_1, ..., x_m$, the index of the partner associated to each agent (that has to be computed) and using one additional boolean variable for each secret preference, $P_{A_i}(u,v)$. The values of variables $P_{A_i}(u,v)$ are the secret unary constraints of A_i. The total number of boolean variables is m^3, m^2 of them being actually fixed by public constraints (e.g. $P_{A_i}(u,u) = 0$). However, also taking into account the variables $x_1, ..., x_m$, the total search space becomes $O(m^m 2^{m^3})$. This is $O(2^{m^3})$ times worse than the centralized CSP formalization whose search space is only $O(m^m)$.

Example of a technique for DisCSPs with constraints known to somebody One of the simplest and fastest techniques for DisCSPs with constraints known to somebody is the version for retrieving all solutions (Herlea *et al.* 2001). Let us introduce one of its versions.

The agents share the secret values of their preferences. Each tuple of assignments candidating as solution to the input problem is refered by an index. They compute for each such tuple ϵ_k, a shared secret whose unknown value is 1 when the tuple satisfies all agents and 0 otherwise. This is done by securely simulating the evaluation of the arithmetic circuit (Ben-Or, Goldwasser, & Widgerson 1988):

$$p(\epsilon_k) = \prod_{i=1}^{|C|} \phi_i(\epsilon_k)$$

Simply reconstructing the obtained shared secrets reveals all solutions, similarly to (Herlea *et al.* 2001). To be noted that (Herlea *et al.* 2001) uses an equivalent boolean circuit evaluation instead of the arithmetic circuit evaluation described here.

We propose now a distributed constraint satisfaction framework that allows to model these problems with the same search space size as the CSP framework, $O(m^m)$.

Redefining the Distributed Constraint Satisfaction Framework

In the previous part of this section we have exemplified CSP models for the stable matchings problem. We have seen that it is difficult to model efficiently these problems using existing private variable-, or private constraint- oriented distributed constraint satisfaction frameworks.

Let us propose a framework for modeling distributed CSPs, where a constraint is not (necessarily) a secret known to an agent, or public, but can also be a secret unknown to anybody. Any distributed problem is essentially (in our view) described by a set of inputs and expected outputs from/to each participant. A distributed CSP is a specialization in the sense that the inputs are used to specify constraints/domains of a CSP, and the outputs are derived from the solution of that CSP. Therefore, we propose that problem descriptions must specify how to obtain the constraints from the inputs of the agents (using functions) rather than attributing constraints to agents.

Definition 1 *A Distributed CSP (DisCSP) is defined by six sets (A, X, D, C, I, O) and an algebraic 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 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 $\phi_i, i{>}0$, 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. Let m be the number of variables. $o_i : D_1 \times ... \times D_m \rightarrow F^{\omega_i}$ 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.

Theorem 1 *The framework in the Definition 1 can model any distributed constraint satisfaction problems with private constraints.*

Proof. The new DisCSP framework can be used to model any of the DisCSP problems with constraints private to agents, by defining I_i as the extensional representation of the private constraint of A_i (assuming the simple but sufficient case of one constraint per agent). $\phi_i(\epsilon, I)$ is then given by the corresponding value for ϵ in I_i (true/1 or false/0). The outputs are going to be $o_i(\epsilon) = \epsilon$ for all i. □

Theorem 2 *The framework in the Definition 1 can model distributed constraint satisfaction problems with private domains (Yokoo* et al. *1998).*

Proof. A private domain of an agent can also be modeled as a private unary constraint, in a DisCSP where each domain is the maximum possible domain for the variable. Then, Theorem 1 applies. □

We do not claim that the new framework is more general than the existing frameworks. It enables us to model naturally and efficiently the desk-mate (stable matchings) problems. One can also model these problems with the old frameworks, but they seem to yield much larger search spaces, and therefore less efficient solutions. Let us now exemplify how this framework can model the new problems.

Modeling the desk-mates problem as a DisCSP. A way of modeling the desk-mates problem as a DisCSP is to have one agent, A_i, and one variable, x_i, for each participant in the problem description. x_i specifies the index of the desk-mate assigned to A_i by the solution, or specifies i if she remains alone. The inputs I_i of each agent are given by the set of preferences $P_{A_i}(u, v)$, specifying whether A_i prefers A_u to A_v, for each u and v. The set F, to which belong the inputs and the outputs, is $\{true, false\}$.

There is a constraint ϕ^{ij} between every pair of variables x_i and x_j, defined as in Equation 1. The output functions are defined as: $o_i(\epsilon) \overset{\text{def}}{=} \epsilon_{|\{x_i\}}$. Namely, each agent learns only the name of her desk-mate. There is a public constraint:

$$\phi_0 \overset{\text{def}}{=} \forall i, j, ((x_i = j) \Leftrightarrow (x_j = i)) \wedge (x_i \neq x_j) \qquad (2)$$

Adapting existing secure solvers to the new DisCSP framework

There exist several algorithms addressing distributed CSPs with privacy of constraints (Herlea *et al.* 2001; Freuder, Minca, & Wallace 2001; Wallace & Silaghi 2004; Yokoo, Suzuki, & Hirayama 2002). Note that none of the recent secure techniques involves propagation. The ones that we succeed to extend to the new framework are:

- Finding the set of all solutions of a distributed constraint problem with secret constraints (Herlea *et al.* 2001).
- Finding the first solution in a lexicographic order for a distributed constraint satisfaction problem with secret constraints that are known to some agents (Silaghi 2003).
- Finding a random solution for a DisCSP with secret constraints that are known to some agents (Silaghi 2003).

When a solution is returned to the desk-mates problem, each agent A_i can infer that: *any agent A_k preferred by A_i to her current desk-mate A_j, prefers her current partner to A_i*. If only one solution is returned (picked randomly among the existing solutions), then no other secret preference can be inferred with certainty.

Theorem 3 *The desk-mates problem can have several solutions.*

Proof. Consider a case with three agents, A_1, A_2, A_3 where $P_{A_1}(2, 3)$, $P_{A_2}(3, 1)$, $P_{A_3}(1, 2)$. This is a loop of preferences, and has three stable solutions, the sets of teams $\{(A_1, A_2), (A_3)\}$, $\{(A_2, A_3), (A_1)\}$, $\{(A_3, A_1), (A_2)\}$. Such an example can be constructed out of any similar loop of preferences, of any size. □

If there exist several solutions, the agents will prefer not to reveal more then one of them. The remaining solutions would only reveal more secret preferences:

- Typically there is no other fair way, except randomness, to break the tie between several solutions.
- If the single solution that is returned is selected as the first one in some given lexicographic order on the variables and domains of the problem, then additional information is leaked concerning the fact that tuples placed lexicographically before the suggested solution do not satisfy the constraints (Silaghi 2003).

General Scheme

We will note that the main difference between the new DisCSP framework, and the one with secret constraints that are known to some agents, is that now the constraints need to be computed dynamically from secrets inputs. All the techniques we extend to the new framework contain a component based on Shamir's secret sharing, that we introduce now. It is the achievement of this sharing which is most affected by the change in framework. We will start by describing Shamir's secret sharing, its importance in distributed multi-party computations, and them we will introduce our changes.

The secure multi-party simulation of arithmetic circuit evaluation proposed in (Ben-Or, Goldwasser, & Widgerson 1988) exploits Shamir's secret sharing. This sharing is based on the fact that a polynomial $f(x)$ of degree $t-1$ with unknown parameters can be reconstructed given the evaluation of f in at least t distinct values of x, using Lagrange interpolation. Absolutely no information is given about the value of $f(0)$ by revealing the valuation of f in any at most $t-1$ non-zero values of x. Therefore, in order to share a secret number s to n participants $A_1, ..., A_n$, one first selects $t-1$ random numbers $a_1, ..., a_{t-1}$ that will define the polynomial $f(x) = s + \sum_{i=1}^{t-1}(a_i x^i)$. A distinct non-zero number τ_i is assigned to each participant A_i. The value of the pair $(\tau_i, f(\tau_i))$ is sent over a secure channel (e.g. encrypted) to each participant A_i. This is called a (t, n)-threshold scheme. Once secret numbers are shared with a (t, n)-threshold scheme, evaluation of an arbitrary arithmetic circuit can be performed over the shared secrets, in such a way that all results remain shared secrets with the same security properties (the number of supported colluders, $t-1$) (Ben-Or, Goldwasser, & Widgerson 1988). For Shamir's technique, one knows to perform additions and multiplications when $t \leq (n+1)/2$. Since any $\lfloor (n-1)/2 \rfloor$ participants cannot find anything secret by colluding, such a technique is called $\lfloor (n-1)/2 \rfloor$-private (Ben-Or, Goldwasser, & Widgerson 1988).

We do not try to encode functions, but only their inputs. All functions (more exactly, arithmetic circuits) that will be computed are public and known by all participants. Their inputs, intermediary values, and outputs are shared secrets. The techniques computing these functions do not reveal any information to anybody and work by letting agents to process the Shamir shares that they know, and by sharing additional secret values.

The techniques solving DisCSPs with private constraints can be used as a black box, except for the secret constraint sharing. Namely, instead of simply sending encrypted Shamir shares of one's constraint, those shares of the constraints have to be computed from the secret inputs of the agents. We therefore propose to replace the secret sharing/reconstruction steps with simulations of arithmetic circuit evaluation which will compute each $\phi_k(\epsilon, I)$ for each tuple ϵ and for the actual inputs I. This step is called *preprocessing*. Intuitively, preprocessing is the step of computing the encrypted initial parameters of the CSP (i.e. acceptance/feasibility value of a tuple from the point of view of each constraint), out of the provided secret inputs. Preprocessing prepares "the pairs" (y,f(y)) that encode the 0/1 values of the constraints. It is done by evaluating arithmetic circuits.

Similarly, instead of just reconstructing the assignments to variables in a solution ϵ, one will have to design and execute secure computations of the functions $o_k(\epsilon)$. This step is called *post-processing*. Intuitively, post-processing is the step of computing the outputs to be revealed to agents, from the obtained encrypted solution of the DisCSP and secret inputs. We show that in our cases this can also be done using simulations of arithmetic circuit evaluations.

Assume $\mathcal{A}$ is some algorithm using Shamir's secret sharing for securely finding a solution of a distributed CSP (with secret constraints known to some agents). The generic extension of the algorithm $\mathcal{A}$ to solve the DisCSP in the new framework is:

- **Preprocessing:** Share the secrets in I with Shamir's secret sharing scheme. Compute each $\phi_k(\epsilon_{|X_k}, I)$ for each tuple $\epsilon_{|X_k}$ and for the actual inputs I by designing it as an arithmetic circuit and simulating securely its evaluation. The public constraint ϕ_0 can be shared by any agent.

- Run the algorithm $\mathcal{A}$ as a black-box, for finding a solution $\epsilon*$ shared with Shamir's secret sharing scheme, for a DisCSP with parameters (i.e. constraints) shared with Shamir's secret sharing scheme.

- **Post-processing:** Compute each $o_i(\epsilon*)$ by designing it as an arithmetic circuit and simulating securely its evaluation. Reveal the result of $o_i(\epsilon*)$ only to A_i.

Pre- and post- processing for desk-mate problems

In the remaining part of the article we will prove that it is possible to design the needed preprocessing and post-processing to solve our example of DisCSPs, the desk-mates problem, using the general scheme defined above.

Preprocessing for the desk-mates problem. We assume the same choice of variables, as for the CSP formalization of this problem above. Let us now show how simple arithmetic circuits can implement the required preprocessing.

Each variable x_i specifies the index of the desk-mate associated to A_i. The input of each agent A_i is a preference value $P_{A_i}(j,k)$ for each ordered pair of agents $(A_j,\ A_k)$, and specifying whether A_i prefers A_j to A_k. $P_{A_i}(j,k)$=1 if and only if A_i prefers A_j to A_k. Otherwise $P_{A_i}(j,k)$=0. A constraint ϕ^{ij} is defined between each two variables, x_i and x_j. I.e. $\phi^{ij}[u,v]$ is the acceptance value of the pair of matches: $(A_i, A_u), (A_j, A_v)$. One synthesizes $m(m-1)/2$ such constraints:

$$\phi^{i,j}[u,v] = \begin{cases} 0 & \text{when } u{=}v \\ (1{-}P_{A_i}(v,u) * (1{-}P_{A_v}(j,i)))* & \\ \ (1{-}P_{A_j}(u,v) * (1{-}P_{A_u}(i,j))) & \text{when } u{\neq}v \end{cases}$$

The public constraint ϕ_0 (same as in Equation 2) restricts each pair of assignments:

$$\forall\epsilon, \epsilon{=}(\langle x_i, u\rangle, \langle x_j, v\rangle) : \phi_0(\epsilon) \stackrel{\text{def}}{=} ((u{=}j)\Leftrightarrow(v{=}i))\wedge(u \neq v)$$

ϕ_0 is known by everybody, and therefore there is no need to compute it with arithmetic circuits. The complexity of this preprocessing is $O(m^4)$ multiplications of secrets (for m^2 binary constraints with m^2 tuples each).

The desk-mates problem does not require any arithmetic circuit evaluation for the post-processing, as each agent A_i learns a value existing in the solution, $o_i(\epsilon) = \epsilon_{|\{x_i\}}$. The participants just reveal to A_i their shares of x_i in the solution.

Complexity

For a problem with size of the search space Θ and c constraints, the number of messages for finding all solutions with secure techniques similar to the one in (Herlea *et al.* 2001) is given by $(c-1)\Theta$ multiplications of shared secrets ($n(n{-}1)$ messages for each such multiplication). For the desk-mates problem modeled with the new framework, $\Theta{=}m^m$ and $c{=}1$ for the version with a single global constraint, or $c{=}m^2/2$ for the version with binary constraints. For the case with binary constraints, it yields a complexity of $O(m^{m+2})$. As mentioned before, the preprocessing has complexity $O(m^4)$ multiplications between shared secrets, resulting in a total complexity $O(m^2(m^m + m^2))$.

Solving the same problem with the same algorithm but modeled with the old DisCSP framework with private constraints, $\Theta = m^m 2^{m^3}$ and $c = m$, for one global constraint from each agent. There is no preprocessing, but the total complexity is $O(m^{m+1}2^{m^3})$. The new framework behaves better since $m << 2^{m^3}$. The comparison is similar for other secure algorithms, like MPC-DisCSP1 (see (Silaghi 2003)) whose complexity is given by $O(dm(c+m)\Theta)$ multiplications between shared secrets.

The Desk-mates Matcher Application

We have implemented the proposed framework as an application to Desk-mates Matcher. (Silaghi 2004).

Our web-application works as follows. An organizer of the computation, e.g. an instructor or a student, uses a web form to generate (for the included JAVA applets) parameters that are customized for the computation at hand. This process requires the organizer to input the size of the class, the names of the students, and a cryptographic public Paillier key provided by each student. Students can generate Paillier key pairs using the corresponding applet linked from the form, and keep the secret keys while handing the private ones to the organizer.

When the customized problem description is generated, a website is automatically built and provided for this problem instance. The organizer is offered an opportunity to email its URL to the students. The organizer can also specify which algorithm to be used for the computation.

Each student browses the received URL, and downloads the applet with customized parameters. The browser can verify the integrity of the applet. Each student provides the applet with his secret key, and inputs his secret preferences. Then he launches his applet into the computation. The applets retrieve each-other's network IP number and port by using a directory server installed on the same host as the web-application. The applets solve the problem securely, and display for each student only the name of her/his partner.

Conclusions

DisCSPs are a very active research area. Privacy has been recently stressed in (Meseguer & Jiménez 2000; Freuder, Minca, & Wallace 2001; Yokoo, Suzuki, & Hirayama 2002) as an important goal in designing algorithms for solving DisCSPs.

In this article we have investigated how versions of old and famous problems, stable matchings problems, can be solved such that the privacy of the participants is guaranteed except for what is leaked by the selected solution. Our approach uses secure simulations of arithmetic circuit evaluations and is therefore robust whenever no majority of the participants colludes to find the secret of the others, and when all agents follow the protocol.

We note that the desk-mates problems cannot be efficiently modeled (at least not in an obvious way) with existing distributed constraint satisfaction frameworks. We have therefore introduced a new distributed constraint satisfaction framework that can model such problems with the same search space size as the classic centralized CSP models. We have shown how some techniques for the existing frameworks can be adapted to problems modeled with the new DisCSPs, and we exemplify the model with the desk-mates problems. For m participants in the desk-mates problem, the size of the search space in the DisCSP model achieved with the new framework is $O(m^m)$ while the previous framework with private constraints yields DisCSP instances with a size of the search space of $O(m^m 2^{m^3})$. In existing secure algorithms for solving DisCSPs, the number of exchanged messages is fix and directly proportional to the search space size, making this property of a problem instance particularly relevant.

References

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

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

Gent, I., and Prosser, P. 2002. An empirical study of the stable marriage problem with ties and incomplete lists. In *ECAI 2002*, 141–145.

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

Irving, R., and Manlove, D. 2002. The stable roommates with ties. *Journal of Algorithms* 43(1):85–105.

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

Silaghi, M.-C.; Sam-Haroud, D.; and Faltings, B. 2000. Asynchronous search with private constraints. In *Proc. of AA2000*, 177–178.

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

Silaghi, M.-C. 2004. Secure distributed CSP solvers. http://www.cs.fit.edu/ msilaghi/secure/.

Wallace, R., and Silaghi, M.-C. 2004. Using privacy loss to guide decisions in distributed CSP search. In *FLAIRS'04*.

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

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

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