Private and Efficient Stable Marriages (Matching)[1]

Timothy Atkinson[†] and **Roman Bartak**[‡] and **Marius C. Silaghi**[†] and **Erdal Tuleu**[†] and **Markus Zanker**[§] [2]

Abstract. We provide algorithms guaranteeing high levels of privacy by computing uniformly random solutions to stable marriages problems. We also provide efficient algorithms extracting a non-uniformly random solution and guaranteeing t-privacy for any threshold t. The most private solution is expensive and is based on a distributed/shared CSP model of the problem. The most efficient version is based on running the Gale-Shapley algorithm after shuffling the men (or women) in the shared secret description of the problem.

We introduce an efficient arithmetic circuit for the Gale-Shapley algorithm that can employ a cryptographic primitive we propose for vector access with an arbitrary number of participants.

Participants want to find a stable matching as defined by their secret preferences and without leaking any of these secrets.

An additional advantage of the solvers based on secure simulations of arithmetic circuits is that it returns a solution picked randomly among existing solutions. Besides the fact that this increases privacy to a level of *requested t-privacy*, it also provides fairness to participants. A real implementation of a described secure solution usable by participants on distinct computers on the Internet is implemented (by students in a class assignment) and is available on our web-site.

1 Introduction

The stable marriages problem consists of matching pairs out of two distinct sets of participants [13]. One member of the pair should belong to the first set and the second member should belong to the second set. The matching is stable if whenever one participant wants to change her partner for a third one, the third participant prefers her current partner to the change. The participants have a secret preference (or tie) between any pair of potential partners.

Versions of these problems, without privacy requirements, have been long known and studied. Techniques for the stable marriages problem are used in US to assign hospitals to medical interns [37]. It is an example of constraint satisfaction problem (CSP). A 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 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 [41], or when the constraints are private to agents [33, 26, 32, 22, 17].

We show that the stable marriages problems can be modeled with the two known types of distributed CSP frameworks, but the obtained models are not efficient. A more efficient model is obtained using the MPC-DisCSP framework introduced in [30].

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 $D_i = \{v_1^i, ..., v_{d_i}^i\}$. $C = \{\phi_1, ..., \phi_c\}$ 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 $\langle x_i, v_k^i\rangle$ meaning that the variable x_i is assigned the value v_k^i.

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.

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 [45, 6, 42]. Some versions consider that each agent owns a constraint of the CSP [45, 38]. This constraint could model the private information of the agent [33]. Other versions consider that each agent owns the domain of a variable while the constraints are shared [41]. The secret domains can also model some private constraints.

None of the two approaches, namely private variables or private domains, can model *efficiently* the stable marriages problems. This is because their private data does not *directly* constrain the allocation of the natural shared resources. An indirect relation exists with such a constraint. Redundant variables need to be introduced in the system. The advantage of a framework based on shared secret CSPs will be stressed in this article, as it allows to avoid the redundant variables. Classic CSP models allow for naturally modeling complex preferences (e.g. ties, etc.).

2 Background

Related Work The stable marriages problem is an old and well studied challenge [13]. Modeling this problem with CSPs has been discussed in [14]. Distributed approaches to solving the problem also appeared in several works among which we mention [5]. We introduced the problem of privacy in stable marriages problem in [36] as an application of MPC-DisCSP1 algorithm. Other secure algorithms developed for the problem appear in [16] which also uses Paillier

[1] This is the full version of the paper at ECAI distributed CSP workshop, August 2006. Fragments from this article were originally part of [27, 35, 36].

[2] [†]Florida Institute of Technology, [‡]Charles University Prague, [§]University Klagenfurt

mixnets and, while being more efficient than our first versions, it cannot offer requested t-privacy.

In several general multi-party computation (MPC) frameworks, secrets s from an algebraic structure F are distributed among participants using sharing schemes. In a sharing scheme, each participant A_i gets a share denoted $[s]_i^F$, and at least t participants are required to reconstruct the secret from their shares. Arithmetic circuits can then be evaluated securely over these shares [3, 40, 15]. Our arithmetic circuits as well as our other protocols also work with MPC schemes where the secret are encrypted with a homomorphic public key cypher E allowing additions of plaintext by operations on ciphertexts [7], and whose secret key is distributed among t servers/participants.

Cryptographic Primitives on Shared Secrets We use the following primitives on secret shares:

- $bits(x)$. Transform the shared secret x (with ℓ bits) into a vector $[x]^B$ of ℓ shared secrets, $[x]^B = b_0, b_1, ..., b_\ell$, with possible values $\{0,1\}$ and representing the corresponding bits of x [8].
- $EXP(x, [y]^B)$. This primitive computes raises x at exponent y where y is shared on bits [9].
- $+, -, *, =, ==, \&\&, ||, <$. These operators are equivalent to the corresponding "C" operators but work on shared secrets [8].
- $a?b : c$. This operator is implemented as $a * (b - c) + c$.
- π=SHUFFLE(a). Generates and applies a secret random permutation π on vector a [29].
- UNSHUFFLE(b,π). Applies on vector b the inverse of the secret permutation π previously applied on vector a [29].
- , firstInArray(a). Sets all 1's in the array a to 0, except for the first 1 [31].
- firstInArrayIdx(a). Returns the index of the first 1 in a [28]. Can be implemented by first computing firstInArray(a), and then $\sum_{i=1}^{N}(ia[i])$.
- $y = a[\overline{x}]$. Reads in y the item at index x in the array a containing N shared secrets. Can be implemented with arithmetic circuits using N equality tests, $y = \sum_{i=1}^{N}((x == i) * a[i])$. [24, 25] gave an efficient version for 2-party computations. We propose next efficient algorithms for this operation with threshold t.
- $a[\overline{x}] = y$. Writes y at index x in the array of shared secrets a. We propose next an algorithm for this operation with threshold t computations.

3 Accessing arrays at secret index

This can be done in constant rounds with N equality tests, but that is asymptotically $log(|F|)$ more expensive in communication than the techniques proposed next.

3.1 Bit-based Access

A fast method we propose is based on bit decomposition and exponentiation with secret index [8, 9]. It works for accessing arrays with size $N < \ell$ where $\ell = \log_2(|F|)$. Given the shared secret index x for a vector of length N, first compute $d_0, d_1, ..., d_\ell = bits(EXP(2, [x]^B))$ by first running the $bits$ algorithm [8] followed by the secret exponentiation of [9], and followed again by the $bits$ algorithm of [8].

Now one can read the array item with $\sum_{i=0}^{N}(d_i * a[i])$.

One can write y in the array a at index x with
$a[i] = a[i] + (y - a[i]) * d_i, \forall i \in [0..N]$.

This version is much faster than the one above since it needs only two expensive $bits$ primitives instead of N of them.

3.2 Mixnet-based secret index access

The next protocol achieves the result in t rounds, where t is the number of supposed trusted servers/participants. First, we assume that $x \in [0..(N-1)]$ and is shared using an additive sharing with shares either from the set of integers Z or from Z_N. Transformations from any sharing to sharing of this form was discussed in [21, 1].

Read at secret index To perform the operation $y = a[\overline{x}]$ where a is an array of N shared secrets, one can use a mixnet related to the one we proposed in [29]. Each participant A_j encrypts his shares $[a_1]_j, .., [a_N]_j$ of the elements of a using a homomorphic encryption scheme E_j for which it holds the secret key and which allows for applying $(+)_F$ on its plaintext through some operation on ciphertexts [29]. All shares are then passed through a mixnet formed by the t participants holding shares of x. Each A_i generates a vector z of N random sharings of zero, and then for each input:

$$I_{j,i} = |E_j([a_1]_j), ..., E_j([a_N]_j)|$$

computes the output to be sent to the next agent

$$O_{j,i} = |E_j([a_1]_j + [z_1]_j), ..., E_j([a_N]_j + [z_N]_j)| <<< [x]_i$$

where $[x]_i$ is A_i's share of x and $<<< [x]_i$ denotes rotational shift with $[x]_i$ positions. In the case of MPCs based on homomorphic threshold encryption E, the mixnet is run on ciphertexts (operations remain the same but without involving shares):

$$O_i = |E(a_1 + 0), ..., E(a_N + 0)| <<< [x]_i$$

A_i can prove that he shifted the arrays and did not simply replaced them with new arrays, by generating an interactive zero knowledge proof. The zero knowledge proof is based on generating a set of K additional claims, consisting of vectors obtained with different z and different shifts.

$$C_{j,k} = |E_j([a_1]_j + [z_1^k]_j), ..., E_j([a_N]_j + [z_N^k]_j)| <<< s^k, k = [1..K]$$

The verifiers specify a challenge bit c_k for each k. For bits $c_k = 0$, the prover reveals s^k and all shares of z^k, showing that the claims $C_{*,k}$ are a rotation of the input. For bits $c_k = 1$, the prover reveals $s^k - [x]_i$ and all shares of $z - z^k$, showing that the claim is a rotation of the output.

At the end of the mix-net, the last agent in the chain broadcasts all encrypted shares in $O_{*,t}[1]$ and each participant decrypts its shares obtaining $a[\overline{x}]$.

Write at a secret index To perform the operation $a[\overline{x}] = y$ where a is an array of N shared secrets, one can use a bidirectional mixnet related to the one proposed in [29]. In Phase 1, each participant A_j encrypts his shares $[a_1]_j, .., [a_N]_j$ of the elements of a using a homomorphic encryption scheme E_j for which it holds the secret key and which allows for applying $(+_F)$ on its plaintext. All shares are then passed through a mixnet formed by the t participants holding shares of x. Each A_i generates a vector z of N random sharings of zero, and then for each input:

$$I_{i,j} = |E_j([a_1]_j), ..., E_j([a_N]_j)|$$

computes the output (to be sent to the next agent)

$$O_{i,j} = |E_j([a_1]_j + [z_1]_j), ..., E_j([a_N]_j + [z_N]_j)| <<< [x]_i$$

where $[x]_i$ is A_i's share of x and $<<< [x]_i$ denotes rotational shift with $[x]_i$ positions towards the left.

At the end of the mix-net, the t^{th} participant in the chain obtains as $O_{*,t}[1]$ the encrypted shares of $a[\overline{x}]$. Now each participant A_j sends to A_t its share of y encrypted with E_j, and A_t replaces $O_{j,t}[1]$ with $E_j([y]_j)$.

In Phase 2, the mix-net is now run in the reverse direction with $O_{*,t}$ as input. Each A_i generates a vector z' of N random sharings of zero, and then for each input:

$$I'_{j,i} = |E_j([a_1]_j), ..., E_j([a_N]_j)|$$

computes the output

$$O'_{j,i} = |E_j([a_1]_j + [z'_1]_j), ..., E_j([a_N]_j + [z'_N]_j)| >>> [x]_i$$

where $[x]_i$ is A_i's share of x and $>>> [x]_i$ denotes rotational shift with $[x]_i$ positions towards the right. The result $O'_{*,i}$ is the result vector a.

A_i can prove that he shifted the arrays and did not simply replaced them with new arrays, by generating an interactive zero knowledge proof. This proof also shows that the rotation is with the same number of positions and in the reverse direction as the first phase. The zero knowledge proof is based on generating a set of K additional claims, consisting of vectors obtained with different z' and the shifts of the corresponding claims at the first phase.

$$C'_{j,k} = |E_j([a_1]_k + [z'^k_1]_j), ..., E_j([a_N]_j + [z'^k_N]_j)| >>> s^k, k = [1..K]$$

The verifiers specify a challenge bit c_k for each k in [1..K]. For bits $c_k = 0$, the prover reveals s^k and all shares of z and z'^k, showing that at both phases the claims $C_{j,*}$ and $C'_{j,*}$ are a rotation of their inputs. For bits $c_k = 1$, the prover reveals $s^k - [x]_i$ and all shares of $z - z^k$ and $z' - z'^k$, showing that at both phases the claims $C_{j,*}$ and $C'_{j,*}$ are rotations of the output.

At the end of the mix-net, the last agent in the chain broadcasts all encrypted shares in $O'_{*,t}$ and each participant decrypts its shares obtaining the result a.

3.3 Efficient arithmetic circuit

Here we show a compilation of the stable marriages problem into a standard arithmetic circuit simulating the solution of [13].

The Algorithm 1 is equivalent to a circuit of size O(N^4). The arithmetic circuit in Algorithm 2 follows more exactly the Gale-Shapley and makes usage of array access for two of the N factors, in the asymptotic complexity.

4 DisCSP Models for Stable Marriages Problems

We employ 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 marriages problems. Basically we argue to the return to a more CSP-like framework where no direct association of the secret constraints to agents is required. Such a setting is enabled by secret sharing. The relation between secrets and participants is relevant at computation steps that are outside the CSP solution (during secret sharing and secret reconstruction).

algorithm *SM-GS-AC2(prefM, prefW)*
  π=SHUFFLE(prefM);
  for $(i = 0; i < N^2; ++i)$ **do**
    for $(k = 1; k \leq N; ++k)$ **do**
      $free[k] = (wife[k]{=}{=}0)$;
    $m = firstInArrayIdx(free)$;
    $match = (m \neq 0)$;
    $nIndex = proposed[\overline{m}] + 1$;
    $proposed[\overline{m}] = (match)?mIndex : 0$;
    $w = (match)?prefM[\overline{m}][\overline{nIndex}] : 0$; // select woman
    $H = (match)?h[\overline{w}] : 0$;
    $prefC = (match)?prefCrt[\overline{w}] : 0$;
    $prefN = prefW[\overline{w}][\overline{m}]$; // this is expensive ;
    $match = ((H!{=}0)\&\&(prefN < prefC))?0 : match$;
    $prefCrt[\overline{w}] = (match)?prefN : prefC$;
    $h[\overline{w}] = (match)?m : 0$;
    $wife[\overline{H}] = match?0 : wife[\overline{H}]$;
    $wife[\overline{m}] = match?w : 0$;
  UNSHUFFLE(wife,π); // apply on vector wife the inverse permutation of π; vector h can be recomputed from wife

Algorithm 2: Version of Algorithm 1 similar to Gale-Shapley

Stable Marriages The stable marriages problem is the problem of finding a set of matches between a set of females, $A_1, ..., A_m$, and a set of males, $B_1, ..., B_m$, such that if any person from the set of females, A_i, prefers some male, B_j, to the partner selected for her, then B_j prefers his current partner to A_i. If any male, B_i, prefers some female, A_j, to the partner selected for him, then A_j prefers her current partner to B_i.

The stable marriages problem is an instance of stable matchings [30] that can be modeled with a lower number of variables. A way of modeling the stable marriages problem as a CSP is to have one variable x_i for each female[3], specifying the index of the male assigned to her by the solution. The constraints are obtained by preprocessing the input of participants about their preferences. The fact that a person A_i prefers B_u to B_v is specified by the boolean constant (input) $P_{A_i}(u, v)$. The fact that B_i prefers A_u to A_v is specified by the boolean constant (input) $P_{B_i}(u, v)$. There is a constraint ϕ^{ij} between every pair of variables x_i and x_j. In first order logic notation, the constraint between each two variables x_i and x_j is:

$$\begin{aligned} \forall x_i, x_j : \phi^{ij}(x_i, x_j) \stackrel{\text{def}}{=} \; & (P_{A_i}(x_j, x_i) \Rightarrow P_{B_{x_j}}(j, i)) \wedge \\ & (P_{A_j}(x_i, x_j) \Rightarrow P_{B_{x_i}}(i, j)) \wedge \\ & (x_i \neq x_j) \end{aligned} \quad (1)$$

In this formulation, the preferences of an agent do not necessarily require a total order on the possible spouses, naturally modeling ties, incomplete lists, etc. Note that a total order is part of the common definition of the stable marriages problem [13, 37].

It is possible to extend the stable marriages problem to the case with an unequal number of males and females. In this case, it can be modeled either:

- as a usual instance of the stable matching problem [30], with one variable for the partner of each participant, each participant publicly preferring to be alone rather then with somebody of the same type, or
- with variables only for females (or males), where the variables

[3] Or male. Then, everything is defined symmetrically.

shared_secret$< [0..N] > proposed$[N+1](0); // previously proposed woman, by man index, initialized to 0
shared_secret$< [0..N] > prefCrt$[N+1](0); // preference for fiance, by woman index, initialized to 0
shared_secret$< [0..N] > wife$[N+1](0); // current fiancee, by man index, initialized to 0
shared_secret$< \{0, 1\} > free$[N](1); // current fiancee, by man index, initialized to 0
shared_secret$< [0..N] > h$[N+1](0); // current fiance, by woman index, initialized to 0
shared_secret$< \{0, 1\} > cont$; // bool: does the current man get engaged to his next preference?
shared_secret$< \{0, 1\} > test, match$; // bool: intermediary test
shared_secret$< [0..N] > w$; // the woman currently addressed
shared_secret$< [0..N] > prefC, prefN$; // preference of current woman for her fiance, resp. for new candidate
shared_secret$< [0..N] > H$; // previous fiance of the current woman
int i; // current round
int j; // index of man currently proposing
int k; // counter
algorithm *SM-GS-arithmetic-circuit (prefM, prefW)*
  shared_secret$< [1..N] > prefM$[N][N]; // the men's preferences in order
  shared_secret$< [1..N] > prefW$[N][N]; // the women's preferences by man index, bigger is better
  π=SHUFFLE(prefM); // shuffle matrix prefM, by a secret random permutation π on its first index
  for $(i = 0; i < N^2; ++i)$ **do**
    for $(j = 1; j \leq N; ++j)$ **do**
      $cont = 1 - (wife[j]{==}0)$; // if already engaged then skip this man
1.1 $proposed[j] = (cont)?proposed[j] : proposed[j] + 1$; // try next woman
      $w = (cont)?wife[j] : prefM[j][\overline{proposed[j]}]$; // selected woman among those willing
      $H = h[\overline{w}]$;
      $prefC = prefCrt[\overline{w}]$; // prefW[w][H]
      for *($k = 1; k \leq N; ++k$)* **do**
        $test = (k == w)$; // This loop can be replaced with array access for k=w
1.2 $cont = cont?cont : (test)?((H!{=}0)\&\&(prefW[k][j] < prefC)) : 0$;
        // $prefCrt[\overline{w}] = (cont)?prefCrt[\overline{w}] : prefW[\overline{w}][j]$ i.e., update preference of w
        $prefCrt[k] = (cont||(1 - test))?prefCrt[k] : prefW[k][j]$; // $prefCrt[\overline{w}] = (cont)?prefCrt[\overline{w}] : prefW[\overline{w}][j]$
        $h[k] = (cont||(1 - test))?h[k] : j$; // $h[\overline{w}] = (cont)?h[\overline{w}] : j$ i.e., set fiance of w to j
      for *($k = 1; k \leq N; k++$)* **do**
        $wife[k] = (cont||(k!{=}H))?wife[k] : 0$; // $wife[\overline{H}] = cont?wife[\overline{H}] : 0$ i.e., set fiancee of H to 0
      $wife[j] = cont?wife[j] : w$; // set fiance of j to w
  UNSHUFFLE(wife,π); // apply on vector wife the inverse permutation of π; vector h can be recomputed from wife

Algorithm 1: Stable Marriages: arithmetic circuit for the Gale-Shapley algorithm (two circuits, selected with find_next_rather_than_try_next).

have an additional value, 0, for specifying that the participant remains single.

For the second case, there is a global constraint:

$$\forall\epsilon, \phi(\epsilon) \stackrel{\text{def}}{=} (\forall i, k : ((k \neq \epsilon_{|\{x_i\}}) \wedge P_{A_i}(k, \epsilon_{|\{x_i\}})) \Rightarrow ((k \neq 0) \wedge (\exists j : (\epsilon_{|\{x_j\}} = k) \wedge P_{B_k}(j, i)))) \wedge (\forall q, t : \epsilon_{|\{x_q\}} \neq \epsilon_{|\{x_t\}}) \quad (2)$$

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.

4.1 DisCSP Benchmark: Stable Marriages

Modeling the stable marriages problem with DisCSPs with secret constraints that are known to some agents (e.g., AAS case). One can model the stable matching problem with secret constraints known to some agents [45, 34] 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 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).

4.2 Using MPC-DisCSP

In the previous part of this section we have exemplified CSP models for the stable marriage 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 use 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 all agents. We have introduced such a framework in [30]. We refine it here by removing the requirement that inputs come straight from participants and that outputs are revealed directly to participants, since this forbinds cases where the solution of a CSP is just part of a chain of intermediary computations that can comprise several CSPs (such as in auctions). In the new framework, secret inputs are assumed already shared at a previous step, and are not associated to a participant. Outputs are also delivered in shared form, and are not necessarily revealed to somebody immediately, but can be feed as inputs in other secure computations.

Definition 1 *A MPC Distributed CSP (MPC-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 *is a tuple of* α *shared secret inputs (defined on* F*). Each input* I *belongs to* F.

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. *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, $X_i \subseteq X$.

$O=\{o_1, ..., o_\omega\}$ *is the set of outputs. Let* m *be the number of variables.* $o_i : D_1 \times ... \times D_m \times I \rightarrow F$ *is a function receiving as parameter a solution and the inputs and returning a secret output (from* F*).*

The following theorems of [30] apply because this framework is more general (less specified) that the framework defined there.

Theorem 1 *The framework in the Definition 1 can model any distributed constraint satisfaction problems with private constraints [34].*

Theorem 2 *The framework in the Definition 1 can model distributed constraint satisfaction problems with private domains [41].*

Modeling the stable marriages problem as a MPC-DisCSP. To model the stable marriages problem as a MPC-DisCSP, one has an agent, A_i, for each female participant A_i in the problem description. Each participant B_j is mapped to an agent A_{m+j}. One has m variables, $x_1, ..., x_m$, modeling the partners of the agents $A_1, ..., A_m$. x_i specifies the index of the spouse assigned to A_i by the solution, or specifies 0, if she remains alone. The inputs I are given by the set of preferences $P_{A_i}(u, v)$ and $P_{B_i}(u, v)$, specifying whether A_i prefers B_u to B_v, respectively whether B_i prefers A_u to A_v, for each u and v. The set F for inputs and outputs is $\{true, false\}$.

The constraint ϕ^{ij} between every pair of variables x_i and x_j, is defined as in Equation 1. The output functions for $i \in [1..m]$ are defined as: $o_i(\epsilon) \stackrel{\text{def}}{=} \epsilon_{|\{x_i\}}$. Namely, each female learns only the index of the husband proposed to her. To return to each male A_{m+i} the identity of the spouse proposed the him, the corresponding output is $o_{m+i} \stackrel{\text{def}}{=} \{k | x_k = i\}$.

There is a public constraint:

$$\phi_0 \stackrel{\text{def}}{=} \forall i, j, x_i \neq x_j \tag{3}$$

5 Private Stable Marriages Solutions

Theorem 3 *Stable marriages problems can have several solutions.*

Proof. Namely take four agents with $P_{A_1}(1, 2)$, $P_{A_2}(2, 1)$, $P_{B_1}(2, 1)$, $P_{B_2}(1, 2)$. q.e.d. □

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 [26].

As it follows, if it is known that a certain problem has only one solution, then any technique is acceptable among either:

- Finding and returning all solutions using the technique in [20], or
- Returning only the first solution (e.g. by sequentially checking each tuple in lexicographical order until a solution is found).

Otherwise, strong privacy requirements make techniques returning a random solution [26] desirable, despite their potential of having a lower efficiency.

5.1 General Scheme

We will note that the main difference between the MPC-DisCSP framework, and the DisCSPs with secret constraints that are known to some agents, is that now the constraints need to be computed dynamically from secrets inputs.

The techniques solving DisCSPs with private constraints can be used as a black box, except for sharing and reconstruction steps. 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 replace the sharing and reconstruction with 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*.

Similarly, instead of just reconstructing the assignments to variables in a solution ϵ, one has to design and execute secure computations of the functions $o_k(\epsilon)$. This step is called *post-processing*.

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

5.2 Pre- and post- processing for stable marriages 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: stable marriages, using the general scheme defined above.

Preprocessing for the stable marriages problem. Some simple arithmetic circuits can implement the preprocessing for the stable marriages problem.

Each variable x_i specifies the index of the male associated to the female A_i. The input of each female A_i specifies a preference value $P_{A_i}(j,k)$ for each pair of males, (B_j, B_k). Each male B_i specifies a preference value $P_{B_i}(j,k)$ for each pair of females (A_j, A_k). $P_{A_i}(j,k)$=1 if and only if A_i prefers B_j to B_k. Otherwise $P_{A_i}(j,k)$=0. $P_{B_i}(j,k)$=1 if and only if B_i prefers A_j to A_k. Otherwise $P_{B_i}(j,k)$=0. A constraint ϕ^{ij} is defined between each two variables x_i and x_j. $\phi^{ij}[u,v]$ is the acceptance value of the pair of matches: $(A_i, B_u), (A_j, B_v)$. One synthesizes $m(m-1)/2$ constraints:

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

The public constraint ϕ_0 (same as in Equation 3) restricts each pair of assignments:

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

specifying that it is not possible for two persons to be associated to the same spouse in a solution.

Post-processing for the stable marriages problem. The stable marriages problem requires a post-processing phase to compute and reveal to each male the spouse proposed to him. Remember that the variables specify only the spouse for each female. The function $o_{m+i} \stackrel{\text{def}}{=} \{k | x_k = i\}$ can be computed with the following arithmetic circuit.

$$o_{m+i} = \sum_{k=1}^{m} ((x_k == i)?k : 0)$$

An implementation for this constraints, written as class assignment by students in Spring 2005, is available online at [19].

6 DisCSP Model based on Global Constraints

It can be noted that since in Equation 2 the variables are constrained to take distinct values, the arithmetic circuit can be written in a simple equivalent form:

$$\begin{aligned} \phi \qquad & (\langle x_{\epsilon_1}, u_1\rangle, ..., \langle x_{\epsilon_n}, u_n\rangle) = \\ & \begin{cases} 0, & \text{when } u_i{=}u_j \text{ and } u_i \neq 0 \\ \prod_{i=1}^{n}((1 - P_{A_i}(0, u_i)) * & \text{otherwise} \\ \prod_{k=1, k\neq u_i}^{m}(1 - P_{A_i}(k, u_i)* & \\ (1 - \sum_{j=1, u_j=k}^{n}(P_{B_k}(j,i))))) & \end{cases} \end{aligned}$$

The total number of multiplications needed to construct this global constraint is $O(mn^{m+1})$, namely mn multiplications for each of the n^m tuples. A public constraint for this problem is:

$$\phi_0(\langle x_{\epsilon_1}, u_1\rangle, ..., \langle x_{\epsilon_n}, u_n\rangle) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{when } u_i{=}u_j \text{ and } u_i \neq 0 \\ 1, & \text{otherwise} \end{cases}$$

Search Space size for DisCSP models Note that the complexity analysis here is based on the assumption of using only basic addition and multiplication secure primitives. It can be easily shown that by using other primitives, such as first-in-array, CSPs are solvable in a linear number of rounds [8].

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 [20] is given by $(c-1)\Theta$ multiplications of shared secrets ($n(n{-}1)$ messages for each such multiplication).

For the stable marriages problem modeled with the MPC-DisCSP 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 classic 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 MPC-DisCSP framework behaves better since $m << 2^{m^3}$. The comparison is similar for other secure algorithms, like MPC-DisCSP1 [26] whose complexity is given by $O(dm(c{+}m)\Theta)$ multiplications between shared secrets.

7 Conclusions

DisCSPs [4, 38, 18, 44, 11, 23] are a very active research area. Privacy has been recently stressed in [2, 12, 39, 10, 43] as an important goal in designing algorithms for solving DisCSPs.

In this article we have investigated how versions of an old and famous problem, the stable marriages problem [13, 37], can be solved such that the privacy of the participants is guaranteed except for what is leaked by the selected solution. Techniques for this problem are currently applied to college admissions and medical interns assignments in US. Our technique uses secure simulations of arithmetic circuit evaluations.

We note that the stable marriages problems can be modeled with existing distributed constraint satisfaction frameworks, but not very efficiently. We have therefore stressed the advantages of the MPC-DisCSP distributed constraint satisfaction framework that can model such problems with the same search space size as the classic centralized CSP models. For m participants in the stable matching problem, the size of the search space in the DisCSP model achieved with MPC-DisCSP is $O((m/2)^{m/2})$ 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 certain existing secure algorithms for solving DisCSPs, the number of exchanged messages is fix and directly proportional to the search space size. In other algorithms the number of rounds is constant but the size of the messages is proportional to the search space. This explains the importance of the size of the search space in an obtained CSP model. Both mentioned types of secure algorithms offering requested t-privacy make this property of a problem instance particularly relevant.

A more efficient solution (but selecting solutions with non-uniform randomness) is proposed with an arithmetic circuit simulating the Gale-Shapley algorithm on a shuffled version of the problem. Its complexity is $O(n^4)$.

REFERENCES

[1] T. Atkinson and M. Silaghi. Efficient array access. Cryptology ePrint Archive, May 2006.

[2] G. Bella and S. Bistarelli, 'Soft constraints for security protocol ananysis: Confidentiality', in *Third International Symposium on Practical Aspects of Declarative Languages*, number 1990 in LNCS, (2001).

[3] M. Ben-Or, S. Goldwasser, and A. Widgerson, 'Completeness theorems for non-cryptographic fault-tolerant distributed computating', in *STOC*, pp. 1–10, (1988).

[4] C. Bessière, A. Maestre, and P. Meseguer, 'Distributed dynamic backtracking', in *CP*, p. 772, (2001).

[5] I. Brito and P. Meseguer, 'Distributed stable matching problems', in *CP*, (2005).

[6] Z. Collin, R. Dechter, and S. Katz, 'On the feasibility of distributed constraint satisfaction', in *Proceedings of IJCAI 1991*, pp. 318–324, (1991).

[7] R. Cramer, I. Damgøard, and J.B. Nielsen, 'Multiparty computation from threshold homomorphic encryption', in *BRICS RS-00-14*, (2000).

[8] 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. `http://eprint.iacr.org`.

[9] Yevgeniy Dodis, Aleksandr Yampolskiy, and Moti Yung. Threshold and proactive pseudo-random permutations. Cryptology ePrint Archive, Report 2006/017, 2006. `http://eprint.iacr.org/`.

[10] B. Faltings and S. Macho-Gonzalez, 'Open constraint satisfaction', in *CP*, (2002).

[11] C. Fernàndez, R. Béjar, B. Krishnamachari, and C. Gomes, 'Communication and computation in distributed CSP algorithms', in *CP*, pp. 664–679, (2002).

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

[13] D. Gale and L.S. Shapley, 'College admissions and the stability of marriage', *American Mathematics Monthly*, **69**, 9–14, (1962).

[14] IP. Gent and P. Prosser, 'An empirical study of the stable marriage problem with ties and incomplete lists', in *ECAI 2002*, pp. 141–145, (2002).

[15] O. Goldreich, *Foundations of Cryptography*, volume 2, Cambridge, 2004.

[16] P. Golle, 'A private stable matching algorithm', in *FC*, (2006).

[17] R. Greenstadt, J. Pearce, E. Bowring, and M. Tambe, 'Experimental analysis of privacy loss in dcop algorithms', in *AAMAS*, pp. 1024–1027, (2006).

[18] Youssef Hamadi, *Traitement des problèmes de satisfaction de contraintes distribués*, Ph.D. dissertation, Université Montpellier II, Juillet 1999.

[19] J. Hamilton, M. Premkumar, and M. Silaghi. Private stable marriages implementation. `http://www.cs.fit.edu/~msilaghi/SMC/examples/stable-marriages`, April 2005.

[20] T. Herlea, J. Claessens, G. Neven, F. Piessens, B. Preneel, and B. Decker, 'On securely scheduling a meeting', in *Proc. of IFIP SEC*, pp. 183–198, (2001).

[21] E. 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`.

[22] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce, and P. Varakantham, 'Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling', in *AAMAS*, (2004).

[23] P.J. Modi, M. Tambe, W.-M. Shen, and M. Yokoo, 'Adopt: Asynchronous distributed constraint optimization with quality guarantees', *AIJ*, (2004).

[24] M. Naor and K. Nissim, 'Communication complexity and secure function evaluation', in *ECCC - Electronic Colloquium on Computational Complexity, Report TR01-062*, (2001).

[25] K. Nissim and R. Zivan, 'Secure discsp protocols - from centralized towards distributed solutions', in *DCR05 Workshop*, (2005).

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

[27] M.-C. Silaghi, 'Incentive auctions and stable marriages problems solved with privacy of human preferences', Technical Report TR-FIT-11/2004, Florida Institute of Technology, Melbourne, FL, (July 2004).

[28] M.-C. Silaghi, 'Hiding absence of solution for a discsp', in *FLAIRS'05*, (2005).

[29] 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`.

[30] 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).

[31] M.-C. Silaghi, G. Friedrich, M. Yokoo, and M. Zanker, 'Secure incomplete multi-party computation for distributed constraint problems', in *AAMAS-DCR Workshop*, (2006).

[32] M.-C. Silaghi and V. Rajeshirke, 'The effect of policies for selecting the solution of a DisCSP on privacy loss', in *AAMAS*, pp. 1396–1397, (2004).

[33] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings, 'Asynchronous search with aggregations', in *Proc. of AAAI2000*, pp. 917–922, Austin, (August 2000).

[34] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings, 'Asynchronous search with private constraints', in *Proc. of AA2000*, pp. 177–178, Barcelona, (June 2000).

[35] 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).

[36] Marius-Calin Silaghi. Secure multi-party computation for selecting a solution according to a uniform distribution over all solutions of a general combinatorial problem. Cryptology ePrint Archive, Report 2004/333, 2004. `http://eprint.iacr.org/`.

[37] S. Skiena, *Stable Marriages*, chapter 6.4.4, 245–246, AW, 1990.

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

[39] R.J. Wallace and E.C. Freuder, 'Constraint-based multi-agent meeting scheduling: Effects of agent heterogeneity on performance and privacy loss', in *DCR*, pp. 176–182, (2002).

[40] A. Yao, 'Protocols for secure computations', in *FOCS*, pp. 160–164, (1982).

[41] 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).

[42] 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).

[43] M. Yokoo, K. Suzuki, and K. Hirayama, 'Secure distributed constraint satisfaction: Reaching agreement without revealing private information', in *CP*, (2002).

[44] W. Zhang, G. Wang, and L. Wittenburg, 'Distributed stochastic search for constraint satisfaction and optimization: Parallelism, phase transitions and performance', in *PAS*, (2002).

[45] 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*, pp. 394–397, (1991).