# Secure Incomplete Multi-party Computation for distributed constraint problems

**M.-C. Silaghi**[†] **and G. Friedrich**[‡] **and M. Yokoo**[$] **and M. Zanker**[‡]
[†]Florida Institute of Technology, USA
[‡]University Klagenfurt, Austria
[$]Kyushu University, Japan

## Abstract

The algorithms we propose here are simple but our contribution consists in identifying the simple guidelines required for a high level of privacy. Achieving the highest level of privacy for secrets used in a distributed computation implies that the distributed computation (steps) should be independent of the value of these secrets. When the expected answer of a constraint satisfaction solver is either a solution or *no_solution*, then the previous assumption leads to algorithms that take always the computation time of the worst case. This is particularly disturbing for such NP-hard problems.

In this work we start from the observation that sometimes (specially for hard problems) users find it acceptable to receive as result not only a solution or the answer *no_solution* but also a failure with meaning *don't know*, or solutions proven optimal only within a subset of the problem space. More exactly, users accept *incomplete* solvers. It is argued in (Silaghi 2005b) that, for certain problems, privacy reasons lead users to prefer having an answer meaning *don't know* even when the secure multi-party computation could have proven *no_solution* (to avoid revealing that all alternatives are infeasible). While the solution proposed there is slower than complete algorithms, here we show secure incomplete solutions that are faster than complete solvers, allowing to address larger problem instances. We show that one can build time-aware instances where given a known amount of available time, we obtain an incomplete solver terminating in that time and offering a very high degree of privacy, namely *non-uniform requested t-privacy*.

## Introduction

Typical examples of distributed constraint satisfaction problems (CSPs) (Yokoo *et al.* 1998) and distributed constraint optimization problems (COPs) (Ali, Koenig, & Tambe 2005; Petcu & Faltings 2005) are meeting scheduling, resource allocation, time-tabling, auctions with several possible winners (Freuder, Minca, & Wallace 2001; Maheswaran *et al.* 2004). Such a problem can be defined by a set of variables and constraints on the satisfiable assignments to these variables. The set of all (satisfiable and unsatisfiable) simultaneous assignments of values to all variables defines the *search space* of the problem. An element of the search space is also referred here to as "an alternative to be considered as a solution to the problem", or simply *alternative*.

A complete CSP solver is one that never fails to find a solution when a solution exists. The result of such a solver is either a solution or the answer *no_solution*. CSPs can be very hard and therefore we do not have efficient complete secure multi-party computation solvers. Several complete secure solvers were proposed in the past for such problems, (Yokoo, Suzuki, & Hirayama 2002a; Herlea *et al.* 2001) and the highest levels of security required a computation time that is given by the worst possible case (over all possible values of the secrets) (Silaghi 2004). A complete COP solver is one returning an alternative with quality at least as good as any other alternative, and privacy requirements have similar implications as for CSPs (Silaghi & Mitra 2004; Silaghi, Faltings, & Petcu 2006).

It was shown that minimization of privacy loss often requires that the solution be picked randomly, preferably with a uniform distribution, among the existing solutions (Silaghi & Rajeshirke 2004).[1] Such a random selection can be achieved if the problem is shuffled prior to solving (Silaghi 2003; 2004). Two families of techniques were proposed for shuffling a shared description of a constraint problem, one based on mix-nets and one based on arithmetic circuits (Silaghi 2005c).

Sometimes, the security requirements themselves imply an incomplete solver (when the proof of unsatisfiability of the problem leads to unacceptable privacy loss, by revealing that all alternatives are infeasible) (Silaghi 2005b). The answer of such a solver is either a solution or *no_solution*. However, the solution proposed in (Silaghi 2005b) is actually slower than complete solutions. It first computes a solution with a complete secure solver and then it hides the solution with some probability.

In this work we show how the shuffling performed on problem descriptions prior to solving allows to build an incomplete secure stochastic multi-party solver where a high level of privacy is offered. The answers of the solver is either a solution or *don't know*, and nothing is revealed about the identity or properties of the set of alternatives that were

[1] We assume that a problem is solved only once. If the problem has to be solved a second time then the same random inputs are provided, guaranteeing that the same solution is returned.

not explored (except for its size). Notably, these algorithms are strictly faster than the corresponding complete versions and are parametrized with the percentage of the search space to be explored (the search space is the set of all alternatives that may or may not optimally satisfy the constraints).

By specifying the percentage of the search space to be explored for a given CSP/COP, one practically specifies the exact amount of computation (steps) that the solver should perform. The proposed techniques depend on whether the used shuffling primitive is based on mix-nets or on arithmetic circuits.

## Background

Constraint satisfaction problems (CSPs) as well as constraint optimization problems (COPs) are often addressed with stochastic and incomplete solvers. While the algorithms proposed here are described for distributed CSPs, they also apply for distributed COPs by following the extension in (Silaghi & Mitra 2004).

A Constraint Satisfaction Problem (X,D,C) is defined by a set of variables $X = \{x_1, ..., x_m\}$, a set of domains $D = \{D_1, ..., D_m\}$ where $D_i$ is the domain for $x_i$, and a set of constraints $C = \{\phi_1, ..., \phi_c\}$. Each constraint $\phi_j$ specifies the acceptable combinations of assignments of values to a subset $X_j$ of the variables. A tuple is a vector of assignments of values to distinct variables. A solution of the CSP is a tuple of assignments of values to all the variables and that satisfies all the constraints. The search space of the CSP is defined by the Cartesian product $D_1 \times ... \times D_m$. An element of the search space is called an *alternative*. The $i^{th}$ alternative is denoted by $\epsilon_i$.

A distributed CSP is a CSP $(X, D, C)$ where a set of participants $A = \{A_1, ..., A_n\}$ have secret shares of $C$, none of them knowing the whole set $C$.

Constraint optimization problems (COPs) differ from CSPs by the fact that the elements $\phi_i$ of $C$ are functions $\phi_i : X \rightarrow \mathbb{R}_+$ rather than predicates. The solution is an alternative maximizing the sum of these functions.

## Shuffling an array of shared secrets

Secure multi-party computations can securely evaluate any arithmetic circuit (Ben-Or, Goldwasser, & Widgerson 1988) or boolean circuit (Kilian 1988; Goldreich 2004) without revealing anything about the inputs except for what can be inferred from the output. 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. Each leaf is a constant.

**Secure computations with Shamir secret sharing** The secure multi-party simulation of arithmetic circuit evaluation proposed in (Ben-Or, Goldwasser, & Widgerson 1988) exploits Shamir's secret sharing (Shamir 1979). 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 $x$. A distinct non-zero number $\tau_i$ is publicly assigned to each participant $A_i$. Therefore, in order to share a secret number $s$ belonging to $A_k$ to $n$ participants $A_1, ..., A_n$, $A_k$ first selects $t-1$ random numbers $a_1, ..., a_{t-1}$ that will define the polynomial $f(x) = s + \sum_{i=1}^{t-1}(a_i x^i)$. 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 and the share $f(\tau_i)$ is often denoted $\langle s \rangle_i^{(t,n)}$. We will assume that all computations are performed in a field $\mathbb{Z}_q$ for some prime number $q$. 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 (resistant to the same number of colluders, $t-1$) (Ben-Or, Goldwasser, & Widgerson 1988; Yao 1982). For Shamir's technique, one knows to perform additions and multiplications when $t \leq (n-1)/2$. Since any $\lfloor n/2 \rfloor$ participants cannot find anything secret by colluding, such a technique is called $\lfloor n/2 \rfloor$-private (Ben-Or, Goldwasser, & Widgerson 1988).

**Secure computations with additive secret sharing** It is also known how to evaluate with computational securely any arithmetic circuit on additively shared secrets. This is a sharing scheme where, to distribute a secret $s$, $A_k$ distributes to each other participant $A_i$ a share $[s]_i$ consisting of a random number and uses as its own share $[s]_k = s - \sum_{i,i \neq k}[s]_i$ (Goldreich 2004) .

**Remark 1** *Note that, if for some unspecified reson the participants want to run again the solver on the very same problem, they obtain the same result as at previous runs, assuming that they provide the same inputs, (including the same secret permutations).*

**Shuffling with mix-nets** In (Silaghi 2003; 2004; 2005c) it is shown how a mix-net can shuffle a vector of shared secrets and can unshuffle a vector of the same size using the inverse permutations. Namely, each participant $A_k$ encrypts his share of each secret using a $(+ \mod q, X)$-homomorphic public encryption scheme $E_k$ for which it holds the secret key[2]. Then $A_k$ sends a vector holding each encrypted share to $A_1$. The vectors with the encrypted shares are passed along each participant in $A$, each of them applying the same secret permutation on all vectors. After applying its permutation each agent adds a shared 0 to each sharing of a secret using the homomorphism of the encryption. Each participant will provide the others with a zero-knowledge proof for the correctness of his shuffling (respectively unshuffling).

**Shuffling with arithmetic circuits** Assume that we have composable multi-party computation primitives (Kiltz 2005; Damgård *et al.* 2005) for computing:

[2] The $(+ \mod q, X)$-homomorphism of $E_k$ means that $E_k(m_1)E_k(m_2) = E_k(m_1 + m_2 \mod q)$. (Silaghi 2005c)

- $\delta_K(x, y)$: Kronecker's delta taking as parameter two shared secrets and returning a shared 1 when $x = y$ and a shared 0 otherwise
- $cmp(x, y)$ takes as parameter two shared secrets and returns a shared 1 when $x < y$ and a shared 0 otherwise
- $RS(m, M)$: random secret generator, generating a shared secret in the public interval $m, M$.
- $\vee_{k=1}^{\ell} x_i$: computes the shared secret result of applying logic $\vee$ on the vector of shared secrets $x_1, ..., x_\ell$ with values in $\{0, 1\}$.

## First in Array

The primitive $first(a[1..m], m)$ can be applied on a vector of $m$ shared secrets, $a[1..m] = a[1]$,...,$a[m]$, with values in the set $\{0, 1\}$, and replaces all its elements with shared 0, except for the first occurence of a 1. A version of the implementation in (Damgård *et al.* 2005), requiring a constant number of rounds, namely 17, and $20m$ multiplications, is described in the following. Implementations with less multiplications but linear or logarithmic number of rounds are straight-forward and are shown in (Silaghi 2003; 2004).

Let $\lambda = \lceil\sqrt{m}\rceil$. First the elements of $a$ are wrapped in a $\lambda \times \lambda$ matrix $b[i, j]$, such that:

$$b[i, j] = \begin{cases} a[i\lambda + j] & \text{if } (i-1)\lambda + j \leq m \\ 0 & \text{otherwise} \end{cases}$$

Then compute $x[i] = \vee_{j=1}^{\lambda} b[i, j]$ and $y[i] = \vee_{k=1}^{i} x[k]$ for $i \in [1..\lambda]$, and compute the row selector:

$$\mathit{row}[i] = \begin{cases} y[i] & \text{if } i = 1 \\ y[i] - y[i-1] & \text{if } i \in [2..\lambda] \end{cases}$$

Next one computes the selected row $r[j] = \sum_{i=1}^{\lambda} \mathit{row}[i] * b[i, j]$ and the column selector with $z[i] = \vee_{k=1}^{i} r[k]$ for $i \in [1..\lambda]$, and

$$\mathit{col}[j] = \begin{cases} z[j] & \text{if } j = 1 \\ z[j] - z[j-1] & \text{if } j \in [2..\lambda] \end{cases}$$

Finally, the changes in the input vector are performed with: $b[i, j] = \mathit{row}[i] * \mathit{col}[j]$ and $b$ is returned as result. Operations can be optimized to disregard the 0 elements added to $b$ at the beginning.

## Shuffling

It is possible to design an arithmetic circuit for shuffling secrets, e.g., using the Algorithm 3. This algorithm uses Algorithm 1 for a permutation of two elements on secret positions in a vector. The random permutation, a set of $k - 1$ swaps, is defined by a random vector of size $k - 1$ computed with Algorithm 2. Each swap is performed with Algorithm 1 that consists of a table look-up (which in particular cases, such as two-party computations, can be performed with specialized algorithms (Naor & Nissim 2001)), followed by a transformation placing the values in the final position. Unshuffling can be done with the Algorithm 4.

```
function Swap (s,i,r,m,M,k)
    //look-up for s[r] (could use specialized protocols);
    $s_r = \sum_{j=m}^{M}(\delta_K(r, j) * s[j])$;
    for $j \in (i, k]$ do
        $s[j] = s[j] + (s[i] - s[j]) * \delta_K(r, j)$;
    $s[i] = s_r$;
```

Algorithm 1: Swapping $i^{th}$ element with $r^{th}$ element for a public value $i$ and a secret value $r \in [m, M]$ in vector $s = \{s_1, ..., s_k\}$ of shared secrets

```
    function RandomVector(k)
        for $j = 1$ to $k - 1$ do
2.1         $r[j] = RS(j, k)$;
        return $r$;
```

Algorithm 2: Define a random permutation for shuffling a vector with $k$ shared secrets

```
function Shuffle(s,k,r)
    for $j = 1$ to $k - 1$ do
        Swap(s,j,r[j],j,k,k);
```

Algorithm 3: Shuffling a vector $s$ with $k$ shared secrets, and a random vector $r$ obtained with Algorithm 2

```
function Un-Shuffle(s,k,r)
    for $j = k - 1$ to $1$ do
        Swap(s,j,r[j],j,k,k);
```

Algorithm 4: Un-shuffling a vector $s$ with $k$ shared secrets, when the shuffling was defined by random secret vector $r$.

This permutation was shown in (Silaghi 2005c) to lead to a random shuffling (taken from a uniform distribution). Note that the random vector defining the permutation could have been built allowing each element to belong to any value between $1$ and $k$ at Line 2.1 in Algorithm 2. This would be computationally more expensive as it would require each call to the procedure $Swap$ to recompute all the elements of the vector to be shuffled (see Algorithm 5).

```
function Shuffle(s,k,r)
    for $j = 1$ to $k$ do
        Swap(s,j,r[j],1,k,k);
```

Algorithm 5: A less efficient shuffling of a vector $s$ with $k$ shared secrets, and a random vector $r$ where each element is obtained with $RS(1, k)$.

## Non-uniform requested t-privacy

Different privacy concepts and levels of privacy were identified for multi-party computations (Ben-Or, Goldwasser, & Widgerson 1988; Freuder, Minca, & Wallace 2001), and they are relevant for different computational models and assumptions. The t-privacy concept introduced in (Ben-Or, Goldwasser, & Widgerson 1988) is very well known and is applicable to many frameworks. It assumes a *passive attacker*, namely an attacker who communicates according to the protocol but gathers data and tries to analyze it for learning secrets.

**Definition 1** *A multi-party computation is t-private if a passive attacker controlling any at most $t$ participants cannot learn anything from the computation, except for what can be inferred from its outputs and prior knowledge.*

Security of a t-private scheme can be computational (if breaking it is computationally intractable for the attacker) or information theoretical (if even infinite computation power cannot infer anything about secrets since no information about them is contained in the data obtained by the attacker).

Given secret inputs $\sigma$, the prior knowledge $\Gamma$ of the $t$ colluders and a multi-party computation process $\Pi$ with answer $\alpha$, the technique is t-private if the probability distribution of the secrets is conditionally independent on $\Pi$ given answer $\alpha$ and knowledge $\Gamma$.

$$P(\sigma|\alpha,\Gamma,\Pi) = P(\sigma|\alpha,\Gamma) \tag{1}$$

However, many algorithms provide answers $\alpha$ that contain more information than what is actually needed. We typically decompose $\alpha$ in a desired data $\alpha^*$ and an algorithmic dependent unrequested data $\overline{\alpha}$. For example[3] the desired data can be an assignment of some variables satisfying secret constraints, and the unrequested data is what can be obtained from peculiarities of the used algorithm $\mathcal{A}$ (e.g., the solution is the first/last in some known order on alternatives).

**Definition 2** *An algorithm $\mathcal{A}$ achieves* requested t-privacy *if the probability distribution of the secrets is conditionally independent on $\Pi, \mathcal{A}$ and $\overline{\alpha}$ given requested data $\alpha^*$ and prior knowledge $\Gamma$.*

$$P(\sigma|\alpha,\Gamma,\Pi,\mathcal{A}) = P(\sigma|\alpha^*,\Gamma) \tag{2}$$

Examples of algorithms designed to be secure according to Equation 1 (Ben-Or, Goldwasser, & Widgerson 1988) but insecure according to the definition introduced in Equation 2 are described in (Silaghi 2003).

For problems with several solutions, *requested t-privacy* typically implies the return of uniformly random selected solutions whenever the problem may have more than one solution. Examples of techniques secure according to the introduced concept are (Silaghi 2004; 2005a). Sometimes uniform randomness in selecting the solution requires very expensive computations. Non-uniform randomness in selecting the solution, while shown to be often clearly better then deterministic approaches and often achievable with significantly reduced computation effort, is less secure (Silaghi 2004). The level of privacy achieved in that case is nevertheless interesting and worth its own definition. We propose to call it *non-uniform requested t-privacy*, and can be describes by the property:

**Definition 3** *An algorithm $\mathcal{A}$ achieves* non-uniform requested t-privacy *if for any secret $\overline{\sigma} \in \sigma$ that is not deterministically revealed given requested data $\alpha^*$ and prior knowledge $\Gamma$, it is also not deterministically revealed given $\Pi, \mathcal{A}$ and $\overline{\alpha}$.*

$$\forall\overline{\sigma} \in \sigma P(\overline{\sigma}|\alpha^*,\Gamma) < 1 \Rightarrow P(\overline{\sigma}|\alpha,\Gamma,\Pi,\mathcal{A}) < 1 \tag{3}$$

## MPC-DisCSP4

The multi-party computation technique, called MPC-DisCSP4 (Silaghi 2005b), extracts a random solution of a distributed CSP. An implementation of this technique is freely available online, as described in (Silaghi 2004). MPC-DisCSP4 uses general multi-party computation building blocks. General multi-party computation techniques can solve securely certain functions, one of the most general classes of solved problems being the arithmetic circuits. A distributed CSP is not a function. A DisCSP can have several solutions for an input problem, or can even have no solution. Two of the three reformulations of DisCSPs as a function (see (Silaghi & Rajeshirke 2004)) are relevant for MPC-DisCSP4:

$i$ A function DisCSP$^1$() returning the first solution in lexicographic order, respectively an invalid valuation $\tau$ when there is no solution.

$ii$ A probabilistic function DisCSP() which picks randomly a solution if it exists, respectively returns $\tau$ when there is no solution.

For privacy purposes only the $2^{nd}$ alternative is satisfactory. DisCSP() only reveals what we usually expect to get from a DisCSP, namely *some* solution. DisCSP$^1$() intrinsically reveals more (Silaghi & Rajeshirke 2004). MPC-DisCSP4 implements DisCSP() in five phases:

1. Share the secret parameters of the input DisCSP using polinomial or additive secret sharing. The value of each publicly possible assignment (allocation) is securely evaluated.
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 (Silaghi 2005c).
3. A version of DisCSP$^1$() where the operations performed by agents are independent of the input secrets (to avoid leaking the secrets), is executed by simulating arithmetic circuits evaluation with the techniques in (Ben-Or, Goldwasser, & Widgerson 1988; Goldreich 2004).
4. The solution returned by DisCSP$^1$() at Step 3 is translated into the initial problem formulation using a transformation that is inverse of the shuffling at Step 2 (Silaghi 2005c).
5. Construct the solution from its secret shares.

It is also possible and very simple to find all solutions (Herlea *et al.* 2001). However, when only a single solution is needed, this leaks a lot of information. At Step 3, MPC-DisCSP4 requires a version of the DisCSP$^1$() function whose cost is independent of the input, since otherwise the users can learn things like: *The returned solution is*

[3] With DisCSPs(Silaghi 2004).

*the only one, being found after unsuccessfully checking all other tuples, all other tuples being infeasible.* Since the used DisCSP$^1$() 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 secure techniques are available, notably MPC-DisCSP1 (Silaghi 2003), MPC-DisCSP2 (Silaghi & Mitra 2004), and MPC-DisCSP3 (Silaghi 2004). We call them generically MPC-DisCSPx. Techniques with a lesser privacy guarantee, namely where the number of computation steps depends on the secrets, are mentioned in (Silaghi 2002; Yokoo, Suzuki, & Hirayama 2002b; Silaghi 2005c; Nissim & Zivan 2005). In this paper we only address multi-party computations without trusted servers. A family of secure solvers based on trusted servers is proposed in (Yokoo, Suzuki, & Hirayama 2002b). Secure algorithms for optimization in COPs are proposed in (Silaghi & Mitra 2004; Silaghi, Faltings, & Petcu 2006).

### Hiding existence of solution

When no solution is found, all the participants learn that no alternative satisfies the constraints. For certain problems this leak of secrets may be considered unacceptable and a *don't know* answer is prefered to learning the infeasibility. But the *don't know* answer is believable only if the algorithm may indeed miss some solutions. An algorithm for missing the solution with some predefined probability $p$ is described in (Silaghi 2005b). It consists of computing a solution using a MPC-DisCSPx algorithm (see Figure 1) and then setting the assignments in the result to the invalid value 0 with a probability $p$.

### Incomplete algorithms

In the CSP world it is known that complete algorithms are ineffective for hard problem instances. For large problems, most applications apply incomplete stochastic search procedures (Yokoo & Hiramaya 1996; Zhang & Zhao 2002). With incomplete search, only a subset of the search space is analyzed. Typical examples of incomplete search are based on some type of hill climbing. With hill-climbing the solver starts with a random alternative and searches the neighbouring search space for solutions.

## Secure Incomplete Search

Let us finally detail our proposed techniques for a secure incomplete search, allowing to address hard problems. Our algorithms are very simple but our contribution consists in identifying that this simplicity is required for a high level of privacy. The idea is that *only an unknown subset of alternatives from the search space will be explored in a predefined number of $N$ steps*.

Incompleteness could be achieved by adding a public constraint that removes the remaining search space. However, to ensure privacy in case of failure (that the infeasibility of this particular sub-space is not revealed), we propose to take advantage of the shuffling of the whole problem. We select the subspace to be explored from the shuffled problem. This hides the exact search subspace that is analyzed and the only secret leaked in case of failure is that there are $N$ infeasible alternatives[4] (but they are not known). In case of success, no information is revealed about any tuples having been tested and found unacceptable before finding the returned solution.

### Secure Incomplete Search with Mix-nets

Each MPC-DisCSPx solving algorithm using mixnets can be modified into a corresponding secure incomplete search protocol that will be called Incomplete Multi-Party Computation for Distributed CSPs (IMPC-DisCSPx). Each IMPC-DisCSPx differs from the corresponding MPC-DisCSPx by the fact that only the first $N$ tuples of the shuffled search space are used to compute the shuffled solution. Each incomplete solver is parametrized by the number $N$ computation steps (of alternatives to be explored), $N$ beeing smaller or equal to the size of the search space. To be noted that an incomplete solver can be seen as a generalization of the corresponding complete solver, which is obtained when $N$ equals the size of the search space.

```
function IMPC-DisCSP4(N,(X,D,C))
    for i=1 to k do
        S[i]=$\prod_{\phi \in C} \phi(\epsilon_i)$;
    Shuffle(S) //using the mixnet;
    $first(S, N)$; // call “first in array” primitive;
    Un-Shuffle(S);
    // the solution can be extracted from S as in (Silaghi 2005b);
    return S;
```

Algorithm 6: IMPC-DisCSP4 for solving a CSP $(X, D, C)$ with $k$ alternatives allowed by the public constraints, and exploring $N$ alternatives.

**IMPC-DisCSP4** For example, IMPC-DisCSP4 is shown in Algorithm 6. IMPC-DisCSP4 requires $k(c-1)$ multiplications of secrets to build the vector $S$ and $2N$ multiplications of secrets to select the solution. Also, the shuffling and unshuffling require each $O(kn^2)$ expensive operations, $O(kn)$ for each participant. While IMPC-DisCSP4 leads to a reduction with up to $2k$ multiplications of secrets, the complexity remains the same, dictated by the shuffling.

It can be noted that the probability that a solution is lost can be fine tuned (with the same motivation as in (Silaghi 2005b)) by discarding the alternative $\epsilon_N$ with probability $p$. This can be done by multiplying each element $x[i]$ in the algorithm “first in array” with $cmp(RS(0, q-1), p * q)$.

**IMPC-DisCSP1** The incomplete algorithm IMPC-DisCSP1 obtained from MPC-DisCSP1 is more successful, and is shown in Algorithm 7. The function DisCSP1 called in Algorithm 7 is the arithmetic circuit proposed in (Silaghi 2003), with the modification that function `satisfiable` only integrates $N$ tuples (rather than the

[4] Or, more exactly, a set of tuples that can be analyzed in $N$ steps.

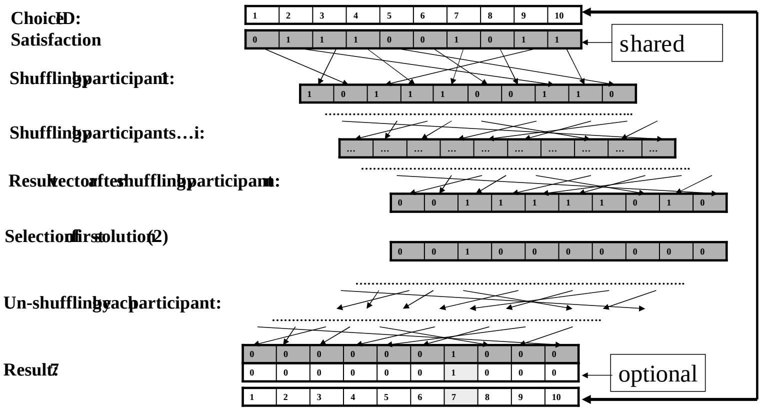

Figure 1: MPC-DisCSP4 using mix-nets

```
function IMPC-DisCSP1(N,(X,D,C))
    Shuffle(X,D,C) //using the mixnet;
    for i=1 to N do
        S[i]=$\prod_{\phi\in C}\phi(\epsilon_i)$;
    F=DisCSP1(N,(X,D,C),S);
    Un-Shuffle(F);
    return F;
```

Algorithm 7: IMPC-DisCSP1 for solving a CSP $(X, D, C)$ with $k$ alternatives and exploring $N$ alternatives.

$$F_j[k] = first([g_{j,1}(P)..g_{j,d}(P)], d)$$
$$p(\epsilon, P) = \prod_{\phi\in P}\phi(\epsilon) \text{ // using S[]}$$
$$g_{i,j}(P) = \texttt{satisfiable}(P \cup \Lambda_i^j \underset{k<i}{\cup} \Lambda^*_{k,P}, N)$$
$$\Lambda_i^j(\epsilon) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } x_i = v_j^i \text{ in valuation } \epsilon \\ 0 & \text{if } x_i \neq v_j^i \text{ in valuation } \epsilon \end{cases}$$
$$\Lambda^*_{k,P}(\epsilon) \overset{\text{def}}{=} F_k[\epsilon_{|x_k}] = \begin{cases} 1 & \text{if } x_k = v^k_{f_k(P)} \text{ in valuation } \epsilon \\ 0 & \text{if } x_k \neq v^k_{f_k(P)} \text{ in valuation } \epsilon \end{cases}$$

Figure 2: DisCSP1(N,P,S): arithmetic circuit for MPC-DisCSP1. It returns the set of vectors $F_j, \forall j \in [1..m]$. $\epsilon_{|x_k}$ stands for the value of $x_k$ in $\epsilon$.

whole search space). The result $F$ returned by DisCSP1 is a set of vectors, one for each variable. Each vector contains shared 0s on all positions, except for a 1 on the position corresponding to the value of the corresponding variable in the found solution. If there is no solution, then all elements of the vectors are 0.

The cost of IMPC-DisCSP1 is only $O(N(md + c))$ multiplications of secrets. Of these, $N(c - 1)$ are used to compute $S$. DisCSP1 calls `satisfiable` $md$ times, each of them requiring at most $O(N)$ multiplications. The cost of shuffling in IMPC-DisCSP1 can be small even for large and hard problems, if the maximum constraint arity (number of involved variables) is small.

```
procedure satisfiable(N,P) do
    return $\vee_{k=1}^{N} p(\epsilon_k, P)$
```

Algorithm 8: Function `satisfiable`: returns 0 or 1 function on whether at least one of the first $N$ tuples in the search space of $P$ is a solution to $P$, where $\epsilon_k$ is the $k^{th}$ tuple.

## Secure Incomplete Search with arithmetic circuits

The secure incomplete algorithms based on mixnets suffer from the fact that the cost of shuffling remains the same as for the complete approaches. This was particularly negative in the case of IMPC-DisCSP4 where the cost of the shuffling is the main cost.

This problem is reduced in algorithms with shuffling based on arithmetic circuits. Namely, with shuffling based on arithmetic circuits one does not need to compute the whole shuffling. With IMPC-DisCSP4, it is possible to only compute the first $N$ elements of the shuffled problem (see Algorithms 9, 10, and 11).

```
function Shuffle(s,k,r,N)
    for $j = 1$ to $N$ do
        Swap(s,j,r[j],j,k,k);
```

Algorithm 9: Shuffling a vector $s$ with $k$ shared secrets, and a random vector $r$ obtained with Algorithm 2

```
function Un-Shuffle(s,k,r,N)
    for $j = N$ to $1$ do
        Swap(s,j,r[j],j,k,k);
```

Algorithm 10: Un-shuffling a vector $s$ with $k$ shared secrets, when the shuffling was defined by random secret vector $r$.

It can be noted that in secure incomplete algorithms with shuffling based on arithmetic circuits we succeed to reduce the cost of shuffling and unshuffling from $O(k^2)$ to $O(kN)$

```
function IMPC-DisCSP4ac(N,(X,D,C))
    for i=1 to k do
        S[i]=$\prod_{\phi\in C}\phi(\epsilon_i)$;
    R=RandomVector(N);
    Shuffle(S,k,R,N) // use Algorithm 9;
    $first(S,N)$; // “first in array” algorithm;
    for i=N+1 to k do
        S[i]=0;
    Un-Shuffle(S,k,R,N);
    return S// the solution can be extracted from S as
    in (Silaghi 2005b);
```

Algorithm 11: IMPC-DisCSP4ac, solving a CSP $(X,D,C)$ with $k$ alternatives allowed by the public constraints, and exploring $N$ alternatives.

multiplications of secrets. With this improvement the complexity of IMPC-DisCSP4ac decreases, but remains high since $k$ is large for hard problems (can be exponential in the problem size).

In conclusion the most appropriate algorithm for Incomplete Search is IMPC-DisCSP1 which has polynomial space requirements and whose computational (time) complexity can be bounded to low values being linear in $N$ and in the problem size. Similarly to their complete versions, while the shuffling of IMPC-DisCSP4 is uniform, this does not hold for the shuffling of IMPC-DisCSP1, which after repeated use can lead to stochastic leaks revealed in (Silaghi 2004).

IMPC-DisCSP4ac (with arithmetic circuits) has a time complexity significantly smaller than MPC-DisCSP4 ($O(k(N+c))$ versus $O(k^2)$). This implies that the size of the problems solvable with IMPC-DisCSP4ac is larger than the size solvable with MPC-DisCSP4, which had the best complexity among secure complete algorithms.

**Remark 2 (IMPC-DisCSP1ac)** *Arithmetic circuit shuffling for IMPC-DisCSP1 works by separately permuting each domain (with a separate random vector for each of them). The improvement that can be brought is to only compute the permuted constraint elements that are part of the first $N$ tuples.*

*The shuffling for IMPC-DisCSP1 is less expensive. Therefore possible improvements in versions based on arithmetic circuit shuffling are less significant, not changing the time complexity.*

## Secure incomplete solvers from classic CSP solvers

It is known that any local computation can be compiled in an arithmetic circuit (Ben-Or, Goldwasser, & Widgerson 1988; Goldreich 2004), and any arithmetic circuit with a constant number of operations can be simulated using a secure computation. Therefore, naturally, several researchers attempted to sketch the immediate application of this theory for solving DisCSPs (Silaghi 2002; Yokoo, Suzuki, & Hirayama 2002a; Silaghi & Faltings 2002; Nissim & Zivan 2005). However, some problems were also noticed quite early:

- The compilation of most of the classic techniques seems to lead to very complex arithmetic circuits. The only such circuits completely developed so far are compilations of very simple classic techniques like generate and test (in MPC-DisCSP4), and variable elimination (in DPOP) (Silaghi, Faltings, & Petcu 2006).
- Standard solvers do not return a solution picked with uniform randomness among candidates, and this was shown to lead to unnecessary leaks of secrets (Silaghi & Rajeshirke 2004). Some secret randomness is needed, the best available privacy (namely requested t-privacy) being achieved by applying a uniform secret shuffling over all tuples prior to the selection of the first solution with a computation requiring a fixed set of operations (MPC-DisCSP4). Otherwise, at least some secret reordering is needed. Secret shuffling of domains was shown to guarantee non-uniform requested t-privacy for MPC-DisCSP1.
- The arithmetic circuits obtained from compiling most classic solvers have a number of steps that depends on secrets, therefore leading to insecure protocols. Again the exceptions identified in previous work is limited to algorithms such as generate and test (in MPC-DisCSP4), and the variable elimination in DPOP (Silaghi, Faltings, & Petcu 2006).

In this work we identify an additional type of algorithms that do overcome the last two problems stated above, and therefore can be compiled into secure solvers with *non-uniform requested t-privacy*. Namely:

**Theorem 1** *Any classic incomplete solver running in a predefined number of steps and preceded by a shuffling of the domains can be simulated with a secure protocol guaranteeing non-uniform requested t-privacy.*

The proof follows from the fact that each of the two problems identified above is overcome by construction. Due to the initial shuffling, each solution has a chance of being the first seen in the shuffled problem (see (Silaghi & Rajeshirke 2004)), and therefore being returned as answer. Moreover the number of steps does not depend on secrets.

Any classic (complete or incomplete) technique that does not run in a predefined constant number of steps, N, can be transformed into such an incomplete technique by:

- adding additional fake steps that change nothing after the solution is found and until a total of N steps are performed.
- if the solver needed more than N steps, stop after exactly N steps with the answer *don’t know*.

Note that the described solution, which guarantees *non-uniform requested t-privacy*, differs from the suggestions in (Nissim & Zivan 2005) which cannot provide such guarantees by the fact that:

- The problem (domains) are shuffled prior to search in our method.
- In our proposal the search is stopped after the predefined public number of steps, N, rather than a random number of steps after solving the problem.

Note that for certain algorithms employing dynamic reordering one may be able to prove that the initial shuffling becomes redundant, but this proof has to be done on a case by case basis.

For a classic CSP solver X, we propose to denote the secure incomplete solver obtained by compilation with our rules IMPC-X(N), being parametrized by the number of steps N. According to the above theorem any IMPC-X(N) guarantees non-uniform requested t-privacy.

## Conclusions

In this work we have proposed a new family of secure solvers for distributed constraint satisfaction and optimization problems. While existing techniques were complete but inapplicable to large instances, the new techniques can be used to securely address larger problems. Our algorithms are very simple but our contribution consists in noting that these simple guidelines are required for a high level of privacy in incomplete search.

We have proposed incomplete versions for each of the complete secure multi-party algorithms MPC-DisCSP1 and MPC-DisCSP4, based on shuffling with mix-nets or with arithmetic circuit. MPC-DisCSP1 is remarkable for its polynomial space requirements while MPC-DisCSP4 is remarkable for its low time complexity and for the uniform distribution in selecting solutions. The new versions only explore a subset of the search space of the problem, subset whose size is specified as a parameter. We have thus analyzed in detail three newly obtained versions: IMPC-DisCSP1, IMPC-DisCSP4, and IMPC-DisCSP4ac. Corresponding versions IMPC-DisWCSPx for distributed constraint optimization problems are obtained as in (Silaghi & Mitra 2004).

It is known that direct compilation into arithmetic circuits of many classic complete or incomplete CSP solvers does not lead to secure *non-uniform requested t-private* computations because their number of rounds depends on secrets. Here we show how to correctly develop corresponding secure incomplete versions of any classic solver X, obtaining a protocol denoted IMPC-X(N), namely by running on a shuffled instance of the shared problem an arithmetic circuit compilation stopped after a predefined number of steps, $N$. The protocols obtained this way do guarantee non-uniform requested t-privacy.

Similarly to its complete counterpart, IMPC-DisCSP1 requires only polynomial space. Unexpectedly, the versions obtained from MPC-DisCSP4 are much less appropriate for addressing large problems, but maintain the desirable property of selecting solutions with a uniform distribution. Among IMPC-DisCSP4 and IMPC-DisCSP4ac, the latter (based on arithmetic circuits) presents the largest speed-up in comparison to its complete version. The algorithm of choice for tackling large problems are therefore the ones based on MPC-DisCSP1 (IMPC-DisCSP1 and IMPC-DisCSP1ac), and their time complexity is linear in the problem size and in a parameter deciding the size of the explored search space.

## References

Ali, S.; Koenig, S.; and Tambe, M. 2005. Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In *AAMAS*.

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

Damgård, I.; Fitzi, M.; Nielsen, J. B.; and Toft, T. 2005. How to split a shared number into bits in constant round and unconditionally secure. Cryptology ePrint Archive, Report 2005/140. `http://eprint.iacr.org`.

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.

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

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.

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

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

Maheswaran, R.; Tambe, M.; Bowring, E.; Pearce, J.; and Varakantham, P. 2004. Taking DCOP to the real world: Efficient complete solutions for distributed event scheduling. In *AAMAS*.

Naor, M., and Nissim, K. 2001. Communication complexity and secure function evaluation. In *ECCC - Electronic Colloquium on Computational Complexity, Report TR01-062*.

Nissim, K., and Zivan, R. 2005. Secure discsp protocols - from centralized towards distributed solutions. In *DCR05 Workshop*.

Petcu, A., and Faltings, B. 2005. Approximations in distributed optimization. In *Principles and Practice of Constraint Programming CP 2005*.

Shamir, A. 1979. How to share a secret. *Comm. of the ACM* 22:612–613.

Silaghi, M.-C., and Faltings, B. 2002. A comparison of DisCSP algorithms with respect to privacy. In *AAMAS-DCR*.

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

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

Silaghi, M.-C.; Faltings, B.; and Petcu, A. 2006. Secure combinatorial optimization using dfs-based variable elimination. In *Symposium on AI and Maths*.

Silaghi, M.-C. 2002. *Asynchronously Solving Distributed Problems with Privacy Requirements*. PhD Thesis 2601, (EPFL). `http://www.cs.fit.edu/~msilaghi/teza`.

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

Silaghi, M. 2005a. Using secure discsp solvers for generalized vickrey auctions, complete and stochastic secure techniques. In *IJCAI05 DCR Workshop*.

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

Silaghi, M.-C. 2005c. Zero-knowledge proofs for mixnets of secret shares and a version of ElGamal with modular homomorphism. Cryptology ePrint Archive, Report 2005/079. `http://eprint.iacr.org`.

Yao, A. 1982. Protocols for secure computations. In *FOCS*, 160–164.

Yokoo, M., and Hiramaya, K. 1996. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)*, 401–408.

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. 2002a. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *CP*.

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

Zhang, W., and Zhao, X. 2002. Distributed breakout vs. distributed stochastic: A comparative evaluation on scan scheduling. In *Distributed Constraint Reasoning*, 192–201.