Secure DisCSP Protocols – From Centralized Towards Distributed Solutions

Kobbi Nissim and Roie Zivan[⋆]

Department of Computer Science
Ben-Gurion University of the Negev
Beer-Sheva, 84-105, Israel
{kobbi,zivanr}@cs.bgu.ac.il

Abstract. We present new protocols for secure distributed constraint satisfaction problems (DisCSPs). The presented protocols are the first to enable an oblivious use of advanced search techniques heuristics.
The first protocol is a *centralized* protocol, where two of the agents collect 'encrypted' data from all other parties, and obliviously perform a search algorithm. Our protocol improves on the previous solution of [YKH05] in several ways: It does not require introducing new agents into the protocol; it enables the use of non-trivial search techniques such as backjumping and ordering heuristics of variables and values; and, it completely eliminates information leakage to all agents.
Our second protocol makes the first steps toward a feasible *distributed* secured protocol for solving DisCSPs. Our protocol enables agents to concurrently perform non sequential (asynchronous) algorithms. It forms an alternative network, whose nodes are small groups (e.g. pairs) of agents, that is generated from the original DisCSP. Each node group obliviously performs the roles of all its members in the search algorithm. We also identify the communication pattern of the protocol as a possible leakage source, and suggest how to eliminate this leakage.
Finally, we discuss a hybrid solution that combines the centralized and distributed protocols and reduces the total communication cost.

1 Introduction

Distributed constraint satisfaction problems (*DisCSP*s) are composed of agents holding locals variables, and a constraints network that restricts the legal assignments to agents' variables. A solution to a DisCSP is an assignment to variables that is in agreement with all the constrains (cf. [Yok00,SGM96]). To achieve this goal, agents run a protocol where they check assignments to their and other agents' variables for consistency. Distributed CSPs are an elegant model for many every day combinatorial problems that are distributed by nature, such as *meeting scheduling* [WF02,ML04] in which agents attempt to schedule meetings parties according to their constrained personal schedule.

Constraint satisfaction is an NP-complete problem [Dec03], and hence one does not hope for an efficient worst-case solution. Instead, many studies suggest exhaustive search algorithms for *DisCSPs* [Yok00,ZM04,MZ03,BMBM05,SF05] which use search techniques and heuristics that prove to behave well in practice. Running an exponential complete search algorithm for polynomially many steps hence results either in a solution, a no-solution announcement or a failure to come to a conclusion whether a solution exists.

⋆ Supported by the Lynn and William Frankel center for Computer Sciences.

Privacy. A motivation for solving a constraint satisfaction problem *distributively* is the agents' need to keep their data private [Yok00,BM03,SF05]. In contrast to centralized CSP, where a single party holds the entire data, in distributed CSP agents may not need to reveal their entire inputs, and hence they may gain some privacy [BM03,Sil02]. However this results in limited privacy, which is inadequate in scenarios where the underlying data is sensitive.

The main privacy concern we address in this study is the information leaked by a protocol during run of the algorithm. Once the search algorithm is determined, one would like to make sure that no information, except for the search outcome, is conveyed to the participating parties. This is an instance of the very well researched cryptographic problem of secure multiparty computation, starting with the works of [Yao82,GMW87] [CCD88,BGW88]. Secure multiparty computation allows a group of n parties (agents) $A_1, \ldots, A_n$ to compute a function f of their corresponding private inputs $x_1, \ldots, x_n$. The outcome is that each of the agents learns $f(x_1, \ldots, x_n)$, but no other information (i.e. except for what is conveyed by $f(x_1, \ldots, x_n)$ and an agent's own input). This requirement is formalized by comparison with a hypothetical *ideal solution*, where the agents send their inputs to a trusted third party that in return communicates $f(x_1, \ldots, x_n)$. It is clear that in this ideal solution parties learn exactly what they should. It is required that any information that is leaked in a run of the real protocol is also leaked in a corresponding run of the hypothetical ideal solution. In other words, whatever computation over $x_1, \ldots, x_n$ a party may perform in a real implementation of the protocol, is *simulatable* in the ideal solution.

Fundamental results in cryptography show how to translate any function f into a secure computation of f [Yao82,GMW87,CCD88,BGW88]. These results follow by securely computing the outcome of each gate in a circuit computing f. The resulting protocol, is of communication and computation complexity proportional to the circuit complexity of f, which is too high for many applications of secure computation, and hence a lot of work is currently invested by cryptographers in finding efficient protocols for specific tasks, avoiding using the generic transformations as is. This approach is also taken in this study.

Previous work on secure DisCSP. Previous works [Sil03,SM04,YKH05] suggested harnessing cryptography, and in particular secure multiparty computation, in order to guarantee privacy. [Sil03,SM04] presented an arithmetic circuit solving CSP, and suggested using the protocol of Ben-Or et al. [BGW88] for generating a secure protocol for CSP. The benefit of this approach is that in a sense one achieves the most in privacy – apart from revealing a solution to the CSP problem, no other information is leaked. Furthermore, correctness and privacy are guaranteed even when a coalition of "bad" parties maliciously deviate from the protocol. However, the circuit size only depends on the input size, and hence the communication and computation of such a protocol is always the *worst-case* communication and computation, in particular, one cannot gain any efficiency from using smart heuristics. As CSP is NP-complete, it may be that even sub-exponential size circuits do not exist for CSP, and hence this approach is limited to solving only very small problems.

Yokoo et al. [YKH05] were the first to present a secure protocol implementing a specific search algorithm (chronological backtracking) for solving DisCSP. They avoid using a generic transformation, and instead present a specifically constructed protocol. The protocol proceeds in iterations (in each iteration a candidate solution is generated and checked), and hence allows for early termination once a solution is found. Early termination is very desirable, as even for simple heuristics, the running time on specific instances may be much lower than the worst-case running time. The protocol introduces new agents into the computation, that perform the computation in a ’centralized’ manner. The resulting protocol provably does not leak any information to the original DisCSP agents, but the solution found by the chronological backtracking algorithm. However, as we discuss in Section 3, the search pattern does leak to the server in charge of performing the search. This leakage is already significant in the case of chronological backtracking. Incorporating advanced search techniques and heuristics would generally worsen this leakage as these exploit the properties of the problem in order to increase efficiency.

1.1 This Work

The goal of this work is to further advance the construction of secure protocols for DisCSP. We present two main protocols for secure DisCSP – a ’centralized’ protocol and a ’distributed’ protocol – and discuss a hybrid protocol that combines the two.

A ’centralized’ protocol. Our first protocol uses two of the agents (denoted *servers*) to obliviously perform a search heuristic. The participating agents send their ‘encrypted’ information to the assigned servers which perform the computation obliviously. When the computation is completed, the severs return the ‘encrypted’ assignment to the other agents, who ‘decrypt’ them locally. We address some of the problems left open in [YKH05]. Our protocol *provably* does not leak *any* information to *any* party, except for what can be learned from the search heuristic result. Our construction may be easily modified to host a variety of search algorithms and heuristics which are known to improve the performance of CSP algorithms [Dec03]. In particular, by concealing the search pattern from the servers we ensure that this use of non trivial heuristics does not reveal any additional information. In contrast to the protocol of [YKH05], we assign the servers role to any two of the participating agents, eliminating the dependency on additional trusted servers and their availability.

We point out that any protocol that does not always run for the worst-case time may potentially leak information by revealing its running time. We present a simple technique for greatly reducing the amount of information leaked this way, with only a slight loss in efficiency.

Technically, we use different cryptographic tools from [YKH05] (see Section 4). The servers operate on *shares* (via a *secret-sharing* scheme) of the state of the search procedure, without any of them gaining any information about the actual state. We also use oblivious table lookup of [NN01].

Comparing the performance of our centralized protocol with that of [YKH05], we note that the computation and communication costs of a single iteration of our protocol are higher (roughly speaking, by a factor of $O(n)$). Thus, for Chronological Backtracking our protocol would be less efficient. We stress however, that these costs come with the benefit of completely eliminating information leakage. Furthermore, the usage of our protocol with an algorithm that is more efficient than Chronological Backtracking may significantly reduce the number of iterations, and hence completely eliminate the efficiency gap.

A ’distributed’ protocol. A common choice of designers of distributed CSP algorithms is to allow agents perform concurrently and asynchronously [Yok00,SF05,BMBM05,ZM04], as these have some benefits over performing a sequential assignment (synchronous) search. In a centralized search a few servers are heavily loaded, whereas the other agents are left idle. High communication cost may be incurred as all the information held by the agents needs to be sent (‘encrypted’) to the servers. In instances where only a small portion of each party’s input needs to be looked at in order to find a solution, a distributed protocol has the potential of being more efficient, as it may result in parties sending only ‘relevant’ portions of their inputs, summing up to a significantly lower communication costs than that of the centralized solution.

The main underlying idea in our distributed protocol is to create a new communication network in which each node represents a group of at least two agents of the original DisCSP. The agents in each such *node-group* share, via a secret sharing scheme, the states of the group agents. We note that the *communication pattern* of a protocol may also be a source of information leakage – If the pattern depends on the agents’ inputs, then it may help an attacker to learn sensitive information. As Current algorithms solving distributed CSPs were not constructed with this concern in mind, we present techniques for making the communication pattern input-oblivious.

A ’hybrid’ protocol. Finally, we note that it is sometimes beneficial to consider a trade-off between the centralized and distributed solutions. E.g. in terms of *total communication costs*, the distributed solution is more efficient when the number of iterations needed to find a solution is small. The centralized solutions becomes more efficient when the number of iterations is large. A strategy for an efficient protocol may hence be to start with the distributed solution, and to change gears and move to the centralized solution if the number of iterations gets large.

2 Distributed Constraint Satisfaction

A distributed constraints satisfaction problem – *DisCSP* – is composed of a set of k agents $A_1, ..., A_k$ and n constrained variables $X_1, \ldots, X_n$ over domains $D_1, ..., D_n$. A *binary constraint* (or *relation*) $R_{i,j}$ between two variables X_i and X_j is a subset of the Cartesian product of their domains: $R_{i,j} \subseteq D_i \times D_j$. More generally, for a subset $X_{i_1}, X_{i_2} ..., X_{i_d}$ of the constrained variables, a *constraint* is a subset of the Cartesian product of their respective domains $R \subseteq D_{i_1} \times D_{i_2} \times \cdots \times D_{i_d}$. In a DisCSP each agent is associated with a subset of the constrained variables. Agents hold (as their private inputs) relations that constrain the assignment to their associated variables with respect to the other variables. cf. [Yok00,SGM96]).

An assignment to a variable X_i is a pair $\langle X_i, val_i \rangle$, where $val_i \in D_i$. A *partial assignment* (or a compound label) is a collection of assignments to variables. A *solution* to a *DisCSP* is a partial assignment that includes exactly one assignment for every variable in $X_1, \ldots, X_n$, *and* satisfies all the constraints; i.e. non of the subsets of the complete assignment $< X_1, val_1 >, < X_2, val_2 > \ldots, < X_n, val_n >$ is in R. As in [YKH05] we assume all constraints are binary.

To solve a DisCSP problem, agents communicate to compute a solution (where each agent holds assignments to the variables she’s associated with). For our ‘centralized’ protocol we assume that the agents are connected by a complete network, and hence an agent may privately send messages to any of the other agents[1]. We also assume a synchronous communication environment in which all messages are sent according

[1] This is only for simplicity of presentation. This requirement may be easily relaxed using standard techniques.

to the synchronous pulses of a global clock and each message sent in clock pulse t is received at clock pulse $t+1$ [Lyn97]. If the environment is asynchronous, (i.e. no assumptions on the delivery time of a message beside that the delay is finite), synchronization is achieved by using a synchronization protocol [Dol00]. We further assume the domains of all agents are identical and include values $1, \ldots, m$ where m is polynomial in the problem size. This assumption was also used in [YKH05] and is compatible with many distributed problems such as *Meeting Scheduling* [WF02,ML04] and *Sensor CSP* [BDF$^+$05]. If the domains are not identical the union of all domain is a reasonable option that fits the model definitions.

3 The protocol of Yokoo et al

The protocol presented by [YKH05] is based on two cryptographic tools: *secret sharing* and *El-Gamal homomorphic encryption*. The protocol uses two external servers, and the parties send their entire data, encrypted, to these servers. From then on, the protocol is run in a 'centralized' manner by the servers.

In the initialization, each agent creates $n-1$ matrices of dimensions $m \times m$ which represent her constraints with each of the other $n-1$ agents: $R_{i,j}[v_1, v_2]$ contains an encryption of "1" if the assignment $\langle X_i, v_1\rangle, \langle X_j, v_2\rangle$ is consistent, otherwise $R_{i,j}[v_1, v_2]$ contains an encryption of a value different than 1. The agents permute the rows and columns of the matrices in order to conceal from the servers the actual values. The permuted and encrypted matrices are sent to the two external servers (a *Search Controller* and a *Decryptor*). Privacy is preserved as each of the servers cannot decrypt the matrices $R_{i,j}$ by herself (they may do so jointly).

The protocol runs in iterations, where the Search Controller holds a partial assignment to the constrained variables, that is known to be consistent, and tries to extend it. The search controller generated an 'encrypted' assignment to a variable that is not in the current partial assignment $\langle X_i, v\rangle$, and, jointly with the Decryptor, decides whether it is consistent with the current partial assignment. If it is - $\langle X_i, v\rangle$ is added to the partial assignment, and another variable would be considered in the next iteration. Otherwise, if the options for the current variable were not exhausted, a new value is to be checked for it; if they were exhausted, the variable that was last added to the partial assignment will be considered in the next iteration.

Privacy of the Protocol. It is clear, by the construction above, that no information (except for the running time) is leaked to the original agents of the DisCSP problem, merely by the fact they do not actually participate in the computation.

However, the search controller does learn some information about the inputs. Although assignment values are 'encrypted', the search controller learns the index-pattern of the current variable for which an assignment is tried, and the length pattern of the current partial assignment, throughout the search. This information is dependent on the structure of the inputs, and hence may in many cases leak information to the search controller. In general, it seems hard to a-priori determine which information would be leaked by these patterns. A simple example is the case where all values of the agent A_1 are constrained with all values of A_2. The search will fail of course, and the search controller will learn (from the search pattern) that the first two agents are in conflict.

The amount of information leaked by the search pattern strongly depends on the search algorithm and heuristics used. Introducing non trivial heuristics to the protocol of [YKH05] would increase the leakage. For example consider the addition of *Back-jumping* [Dec03] to the protocol. On each backtrack operation the search controller will jump to the last variable in the current partial solution, which is in conflict with the variable to whom it could not find a consistent assignment. Hence, by observing the search pattern the search controller would learn information on which variables were in conflict.

Efficiency of the Protocol. In each iteration the servers exchange a constant number of messages which include the information needed to encrypt the constraints checked. The communication cost of each iteration therefore is $O(k)$; the computation cost is $O(nk)$. Where n is the length of the current partial assignment the constraints are checked against and k is the security parameter for the encryption scheme.

4 Cryptographic background – Secure Multiparty Computation

Secure multiparty computation [Yao82,GMW87,CCD88,BGW88] enables two or more parties to compute a function of their joint inputs, restricting how parties may influence the computation and what they learn from it (about other parties' inputs). Advanced studies in this field supply tools for secret sharing of variables and oblivious computation of deterministic and random efficient functions.

In this section we briefly describe the main variants of the problem, and basic results and tools. The reader is referred to [Gol04] for a comprehensive review.

4.1 Adversary Model

The cryptographic literature deals mainly with two types of adversaries: *semi-honest* and *malicious*; these model how agents may behave in the protocol in order to tamper its privacy/correctness. We provide intuition and informal definitions of these models. We note that our protocols (as is the protocol in [YKH05]) are designed for semi-honest adversaries.

Semi-honest adversaries. In most of this work we address the case where the parties that participate in the protocol are *semi-honest* i.e. they follow the protocol as prescribed but may record all messages they get and their intermediate computations and randomness with the goal of subsequently deducing information not derivable solely from the protocol output. For simplicity, we give an informal definition for the two-party case, in the setting of deterministic functionalities.

Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a function. Let P be a two-party protocol for computing $(f_1(x,y), f_2(x,y)) = f(x,y)$ (note that the parties need not compute the same function). Protocol P for f is private if the view of party A_1 during a run of the protocol is simulatable, by a polynomial-time machine, given only her input x and what she should learn from running the protocol, namely $f_1(x,y)$. By *simulatable* we mean that the same (or computationally indistinguishable) distribution on messages that A_1 sees during a real execution of the protocol is efficiently generated by a polynomial-time algorithm – the *Simulator* S_1 – given x and $f_1(x,y)$. Similarly the view of A_2 is required to be simulatable by some polynomial-time simulator S_2 as $S_2(y, f_2(x,y))$.

Composition in the semi-honest model. It is generally easier to construct protocols for the semi-honest adversary model. In particular, there exist composition theorems that allow for modularizing the problem, and construct protocols by composing sub-protocols, that are each proved private with respect to semi-honest parties, into larger protocols.

Malicious Adversaries. The semi-honest model is of a *benign* adversary, that acts as prescribed in the protocol. On the other hand, a *malicious* adversary may deviate arbitrarily from its prescribed protocol. In particular, such adversaries may choose to run the protocol with fake inputs or to stop participating in the protocol. We skip the privacy definitions for this model as this work only deals with the semi-honest case.

Some work has to be done in order to understand the implication of the different adversary models in the realm of DisCSP. Regardless of that, the semi-honest model is important by itself, and one needs to solve the problem in this model to be able to solve it in the malicious adversary model. We note that general transformations of protocols that are secure in the semi-honest model into protocols that are secure in the malicious model exist [GMW87]. The transformation is via a compiler that 'forces' the parties to act semi-honestly. Hence, a possible design paradigm for secure protocols is to construct first a protocol for the semi-honest model and then compile it. The problem with this approach is that the compiler results in protocols that are inefficient in practice.

Colluding Adversaries. In the multi-party case, one has also to consider the effect of adversarial collusion on the security of the protocol. A collusion is usually modeled by a 'master-mind' adversary that learns their internal states and (in the case of malicious adversaries) controls how they act in the protocol. Through most of this work we do not deal with this concern.

Feasibility Results. Generic results in Secure Computation show that any function that is computable in polynomial time, may also be computed by a secure protocol (wrt semi-honest or malicious adversaries) with polynomial time and communication complexities (cf. [Yao82,GMW87]). In particular, we mention Yao's *garbled circuit* protocol for secure two-party computation of a function f. Represent f by a Boolean circuit C_f computing it. Yao's protocol securely computes the outcome of each circuit gate in the natural order induced by C_f. The resulting protocol, is a three-round protocol, of communication and computation complexity proportional to the *size* of C_f, which is at least as large as the input size of f (in bits). As we noted above, using Yao's transformation generally results in protocols that are not efficient enough, especially with large inputs or functions f that exhibit high circuit complexity. The transformation may be reasonable, however, for creating sub-protocols of the problems, those with small inputs, and simple functionality.

4.2 Basic Tools

We describe the cryptographic tools we use in the sequel. [2]

[2] *Notation*: We use $\in_R$ to denote selection from a set with uniform probability.

Oblivious Transfer. Oblivious Transfer (OT) is a specific case of secure function evaluation first suggested by Rabin. This is a protocol run between two parties. One party (the sender) holds as input a list of n values $x_0, \ldots, x_{n-1}$, and the other party (the chooser) holds as input an index i. The result of running the OTprotocol is that the chooser learns x_i, with no other information being leaked. I.e. in the case the values $x_0, \ldots, x_{n-1}$ are bits and $i \in \{0, \ldots, n-1\}$ we have $\mathrm{OT}_1^n((x_0, \ldots, x_{n-1}), i) = (\perp, x_i)$. An oblivious transfer protocol is a protocol that privately computes the function

Oblivious transfer serves as a basic block in our constructions. There are several efficient constructions of oblivious transfer protocols (cf. [AIR01,NP01,NP99]), under appropriate hardness assumptions.

Secret Sharing Secret sharing schemes enable distribution of a secret between a collection of agents such that only pre-designed quorums of agents are able to reconstruct the secret. An example is Shamir's secret sharing scheme [Sha79] that shares a secret s between n agents. The scheme chooses a random degree-t polynomial $p()$ satisfying $p(0) = s$ and distributes $p(i)$ to agent i. The scheme is a threshold scheme – any subset of t agents may combine their shares to compute s; On the other hand, the shares held by any coalition of less than t agents are independent of the secret s, hence any subset of less than t shares does not convey any information about s.

In our constructions, we use a simple scheme for sharing secrets between two parties. The secret s is assume to be in the range $\{0, \ldots, n-1\}$. One share of s is chosen at random, $s' \in_R \{0, \ldots, n-1\}$; the second share is set to $s'' = s + s'$ (mod n). Note that each of the shares is (on its own) a random number chosen from $\{0, \ldots, n-1\}$ *independently* of s. Given s' and s'' the secret is reconstructed as $s = s' + s''$ (mod n).

A special case that we state explicitly is of sharing a single bit ($n = 2$, $s \in \{0, 1\}$). Here, one share is chosen at random $s' \in_R \{0, 1\}$ and the second is set to $s'' = s \oplus s'$.

Oblivious Table Lookup. We will use the following table lookup protocol from [NN01]. Both the inputs to the protocol and its output are shared between two parties. Specifically, one party holds as input $s' = s'_0, \ldots, s'_{n-1}$ and $i' \in \{0, \ldots, n-1\}$; The other party holds $s'' = s''_0, \ldots, s''_{n-1}$ and $i'' \in \{0, \ldots, n-1\}$. These inputs are shares of a list $s = s_0, \ldots, s_{n-1}$ with $s_i = s'_i \oplus s''_i$ and an index $i = i' + i''$ (mod n).

The outcome of the protocol is a sharing of s_i. I.e. one party learns a random value $r_1 \in \{0, 1\}$ and the other learns $r_2 = r_1 \oplus s_i = r_1 \oplus s'_{i'+i''} \oplus s''_{i'+i''}$.

Figure 1 presents the protocol in details. It is easy to verify that $r' = e' \oplus z''_{i'} = e' \oplus e'' \oplus s''_{i'+i''} = e' \oplus e'' \oplus s''_i$ and similarly $r'' = e' \oplus e'' \oplus s'_i$, hence the outcome satisfies $r' \oplus r'' = s'_i \oplus s''_i = s_i$ as required.

5 A Centralized Protocol for Secure DisCSP

Our first solution is a centralized protocol in which two servers Srv_1, Srv_2 cooperate to securely solve the problem. Any two of the original agents may serve as the servers. The servers use the simple secret-sharing scheme from Section 4.2 to jointly hold the *state* of the protocol without any of them gaining any knowledge about it. The protocol proceeds in iterations, in each the state is obliviously updated by the servers. At the end of each iteration, the servers only learn if the protocol should be terminated. When the

INPUT: Alice holds $s' = s'_0, \ldots, s'_{n-1}$ and $i' \in \{0, \ldots, n-1\}$. Bob holds $s'' = s''_0, \ldots, s''_{n-1}$ and $i'' \in \{0, \ldots, n-1\}$.

1. Alice chooses an 'encryption key' $e' \in_R \{0, 1\}$. She 'encrypts' every item in s' with e and cyclically shifts the result for i' steps: $z'_j = e' \oplus s'_{j+i'}$ for $j \in \{0, \ldots, n-1\}$. See [NN01] for the security of this protocol. Similarly, Bob chooses $e'' \in_R \{0, 1\}$ and sets $z''_j = e'' \oplus s''_{j+i''}$.
2. Alice and Bob run two OT_1^n protocols in parallel:
   (a) In one protocol Alice is the chooser, with input i', Bob is the sender with input $z'' = z''_1, \ldots, z''_n$. Alice learns $z''_{i'}$ and sets $r' = e' \oplus z''_{i'}$
   (b) In the other protocol Bob is the chooser, with input i'' and Alice holds $z' = z'_1, \ldots, z'_n$. Bob learns $z'_{i''}$ and sets $r'' = e'' \oplus z'_{i''}$.
3. Alice and Bob locally output r', r'' resp.

Fig. 1. Oblivious table lookup protocol.

protocol terminates, every agent learns an assignment to her constrained variables (if a solution was found), and the number of iterations until termination.

We first describe a protocol for the *Chronological Backtracking* algorithm, and then show how other search algorithms and heuristics may be implemented. For the simplicity of the presentation, we assume that each agent A_i holds exactly one variable X_i.

Similarly to the protocol of [YKH05], each agent A_i, creates $n-1$ matrices of dimensions $(m+1) \times (m+1)$ which contain the constraints between its variable and each of the other agents variables as follows[3]:

$$R_{i,j}[v_1, v_2] = \begin{cases} 0 & v_1 = 0 \text{ or } v_2 = 0 \\ 0 & \langle i, v_1 \rangle \langle j, v_2 \rangle \text{ is consistent} \\ 1 & \langle i, v_1 \rangle \langle j, v_2 \rangle \text{ is inconsistent} \end{cases}$$

Figure 2 shows our (insecure) version of the Chronological Backtracking algorithm. We now describe how to implement each of its operations in a secure manner.

In the secure protocol, all variables (including the matrices $R_{i,j}$, the state variables $current, v_1, \ldots, v_n$ and the intermediate variables $val, conflict, term$) are shared by Srv_1, Srv_2 via the simple secret sharing scheme described in Section 4.2.

Initialization. Each agent A_i computes shares of $R_{i,j}$ by choosing a random Boolean matrix $R'_{i,j} \in_R \{0, 1\}^{(m+1) \times (m+1)}$ and setting $R''_{i,j} = R_{i,j} \oplus R'_{i,j}$. Agent A_i sends $R'_{i,j}$ to Srv_1 and $R''_{i,j}$ to Srv_2.

Termination. At the end of each iteration, the servers combine the shares of $term$ to decide whether to terminate the protocol. If $term = TRUE$, they send to each agent A_i the two shares of v_i. Agent A_i then combine the shares to compute her assignment.

[3] $R_{i,j}$ represents the constraints held by agent A_i wrt variables x_i, x_j. The case $v_1 = 0$ or $v_2 = 0$ is added for dealing with variables for which a partial assignment does not assign a value.

INPUT: Constraint matrices $R_{i,j}$ for $1 \leq i, j \leq n$ (we use the convention $R_{i,i} = 0^{(m+1)\times(m+1)}$).
STATE VARIABLES: $current \in \{0, \ldots, n-1\}$ – initially set to 1. $v_1, \ldots, v_n \in \{0, \ldots, m\}$ – initially all set to 0. $term \in \{TRUE, FALSE\}$ initially set to $FALSE$.
ASSUMPTION: $\langle 1, v_1\rangle, \ldots, \langle current - 1, v_{current-1}\rangle$ is a consistent partial assignment and $v_{current+1}, \ldots, v_n = 0$.

```
1. While ($\neg term$)
2.    Let $val = v_{current} + 1$
3.    Let $conflict = \bigvee_{j=1}^{n} (R_{current,j}[val, v_j] \vee R_{j,current}[v_j, val])$
4.    If ($conflict$ and $val = m$) let $val = 0$
5.    Let $v_{current} = val$
6.    If ($\neg conflict$) Let $current = current + 1$ otherwise, if ($val = 0$) let $current = current - 1$
7.    Let $term = ((current = 0) \vee (current = n + 1))$
```

Fig. 2. Chronological Backtracking.

The Iterations. One option is to construct a Boolean circuit C_{CBT} that computes an iteration of the Chronological Backtracking heuristic, and then use Yao's garbled circuit transformation [Yao82]. However, the input to the circuit is of size $O(n^2m^2)$, as it includes all the constrain matrices $R_{i,j}$, which is quite large. Instead, we choose to use the oblivious table lookup of [NN01], in order to reduce both communication and computation.

The functionality of Lines 1., 3., 4., 5., 6. can be described via quite compact circuits[4], hence we use Yao's garbled circuit construction for implementing them, as choosing more efficient constructions would results in an insignificant improvement. On the other hand, naively implementing line 2. using Yao's construction would require a circuit of size $\Omega(n \cdot m^2)$, resulting in a similar lower-bound on computation and communication. Using oblivious table lookup, we improve on the communication complexity, and reduce it to, roughly, $O(n \log m)$.

The privacy of the iteration functionality follows from composition theorems for the semi-honest case, and the security of the underlying sub-protocols. We defer the details to a later version of this work.

5.1 Implementing Other Heuristics

Unlike the protocol of [YKH05], our protocol conceals the search pattern from the participating agents, and hence allows introducing search techniques and heuristics which exploit the properties of the problem in order to gain efficiency. In particular, if the state s that has to be maintained by servers performing the algorithm is small, then our techniques allow for efficient lookup in s (using oblivious table lookup) and efficient 'update' of s (using the garbled circuit of [Yao82] would result in communication and computation complexity that is proportional to the size of s in bits). If a larger state needs to be maintained, than updates become costly, and the techniques from [NN01] for creating oblivious-write RAM machines may be used, incurring amortized update communication costs roughly $O(\log |s|)$.

The following examples demonstrate how improved search techniques and heuristics may be introduced into the protocol.

[4] Lines 3., 5., 6 require circuits of size $O(\log n)$. Lines 1., 4. require circuit of size $O(n \log m)$.

Adding Simple Backjumping. For that, the two servers may share an additional array $jump\text{-}to$ of n integers, which for each variable, holds the largest agent index that pruned a value from its domain. This array is updated, if needed, whenever an attempt to assign a variable with some value fails. On a backtrack operation, the state variable $current$ is updated to $jump\text{-}to[current]$

Dynamic Variable and Value Ordering Heuristics. Variable and value heuristics are known to be very beneficial in solving CSPs [Dec03]. To perform variable ordering the servers can instead of sharing the index $current$, share an array of $n + 1$ indices, initially set to all zeros. When the next variable is chosen by the heuristic[5], the servers update that variable's entry in the array with its index in the current partial assignment. This way the array holds the order in which the variables are assigned and performs backtrack accordingly. A similar construction may be used for the order of values. In the case of values, a boolean array for each variable is needed. A $FALSE$ entry represents a value that have not been assigned yet, and a $TRUE$ entry represents a pruned value.

5.2 Reducing Leakage due to Timing

Timing attacks are attacks that utilize information that is leaked by measuring the time it takes to compute a functionality. Timing attacks were presented on some cryptographic constructions, and were shown to leak sensitive information [Koc96]. In our setting, some information may be leaked by the protocol, just by counting the number of iterations T it takes to find a solution. We do not know how this information affects privacy, but there is a potential to a leakage of $\log T$ bits of information.

We present a very simple modification to our algorithm that may reduce this leakage significantly. Namely, we add a variable T to the state shared by the servers, and change the termination condition. T is initialized to 0 and counts the number of iterations for which the protocol was run. The iteration procedure is modified as follows. If the state variable $term$ is $FALSE$, then the iteration is performed as in Figure 2. Once $term$ is set to $TRUE$, idle iterations, where no change to variables $i, v_1, \ldots, v_n$ occur. The new termination condition requires that $term = FALSE$ *and* $T = \lceil (1 + \epsilon)^k \rceil$ for some integer k. The value $\epsilon > 0$ is a small constant. The running time of the modified protocol is hence at most $(1 + \epsilon)$ times that of the original protocol. Leakage due to timing is greatly diminished, and is bounded by $\log_2 \log_{1+\epsilon} T = O(\log \log T)$ bits of information.

6 Secure Distributed protocols

The cryptographic primitives which we introduced in Section 4 and used in Section 5, may also be used in constructing a distributed protocol, in which the (encrypted) problem data need not be transferred to a small collection of servers.

There are several reasons why such protocols are desirable. First, they may allow the usage of *concurrent* asynchronous computation techniques which are commonly used in solving DisCSPs [Yok00,Sil02,MZ03,ZM04,NSHF04]. A second reason is that they may balance the computation and communication between the agents of the DisCSP which were idle during most of the centralized protocol. Third, eliminating the need to transfer the input data to servers.

[5] We do not describe how the decision is made since it is specific to the heuristic chosen.

In this section, we present a framework for constructing secure distributed protocols. Assuming that agents need relatively small space for conducting a distributed CSP protocol, we generate an *alternative network* in which every node represents a pair of agents, that run a secure protocol similarly to that presented in the previous section.

6.1 Alternative Network

Given a network of n agents, we first generate an alternative network, whose nodes represent two or more agents of the original network, such that every agent is assigned exactly to one node. We note that in order to prevent any information leakage due to the alternative network structure, the construction is oblivious of the agents' inputs. It may, however, depend on the specific algorithm performed. For simplicity of presentation, we assume that n is even, and that every node of the alternative network represents exactly two agents. Two nodes, N_i and N_j of the alternative network share an edge in the alternative network iff at least one of the agents of node N_i in the original $DisCSP$ is linked with one of the agents of node N_j. Assuming the $DisCSP$ is connected, this would mean the alternative network is also connected. See Figure 3 for an example.

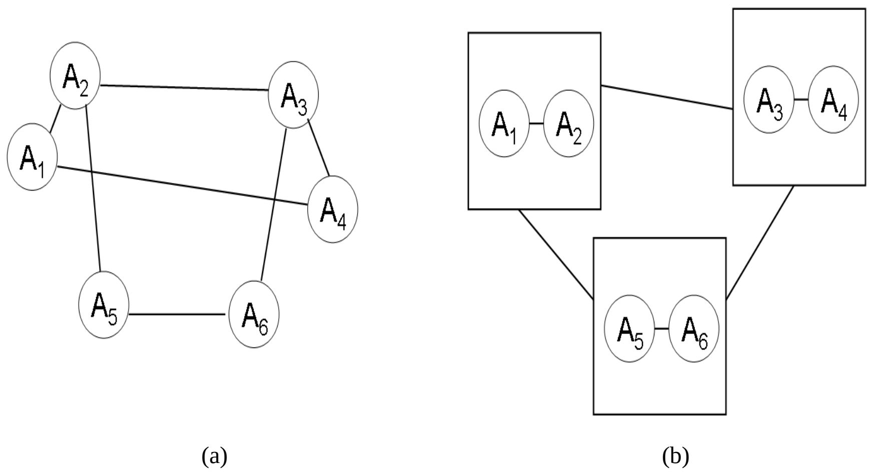

Fig. 3. (a)A standard network. (b) An alternative network.

We will have the agents of every node in the alternative network secretly share their respective states in the original DisCSP. We note that the agents need not share their input constrains $R_{i,j}$[6]. The agents in each node run a secure protocol that, in each iteration considers the received messages, and the internal state and computes shares of the updated state, as well as of messages that need to be sent to other nodes in the alternative network, using techniques similar to that presented in Section 5. Similarly to our centralized protocol, the agents of the original DisCSP are not aware of their state in the search, and hence do not learn anything during the computation. We briefly describe the major points in the construction.

[6] I.e. agent A_i holds the 'share' $R_{i,j}$, and the other agent in the node holds the 'share' 0.

6.2 The Communication Pattern.

In the distributed case, we have to deal with a potential source of information leakage that does not usually exist in the two-party case, namely, the communication pattern. We note that even if messages are sent encrypted, so that their content is not readable by an adversary, the fact that a message was sent at time (or iteration) t may leak information – as it may help distinguishing inputs for which a message is sent at time t from those where that does not happen. We may consider here two types of adversaries: the original agents of the network, and an external viewer of the network. The agents see only some of the messages, thus it is sufficient to make each of the individual communication patterns seen by agents input-oblivious. An external viewer that learns the entire communication pattern may even pose a worse privacy threat.

To the best of our knowledge, the problem of directly constructing input-oblivious communication DisCSP protocols was not previously addressed. A related problem is that of *anonymous message delivery*, where one hides the senders and receivers in a communication network. In particular, using anonymous message delivery for sending messages in a protocol results in hiding its communication pattern. A possibility is to use the techniques of [BD03]. Their solution define several 'buses' (these are messages that may contain shorter messages in 'seats') that run in pre-specified 'routs'. The main parameters of such a scheme are the time it takes to deliver a message and the communication costs. An extreme solution is that on every clock pulse, messages are passed between all nodes of the network in both directions. This results in delivery time $O(1)$, and communication $O(n^2)$. Another extreme point is to have a single bus that traverses the network (e.g. using a spanning tree). This results in delivery time $O(n)$, and communication $O(n^2)$ [7]. Beimel and Dolev present several ways to choose bus-routs to obtain other tradeoffs between these parameters. We note that although a constant load of n^2 messages seems high, this is the load of common DisCSP algorithms [Yok00].

How nodes communicate. Logically, nodes of the alternative network receive messages from other nodes, do some computation that results in updating their local state, and eventually send messages to other nodes. The agents that correspond to a node are responsible of performing the computation obliviously. In particular, to (logically) send a message from one node of the alternative network to another, shares of these messages are computed by agents of the originating node (each share is held by a single agent), and are sent to their 'twin'-agents in the receiving nodes (again, each share is received by a single agent).

7 A Hybrid Protocol

In traditional DisCSP, the question whether to run a distributed or a centralized protocol affects both privacy and efficiency. In our setting, the privacy concern is eliminated (wrt to non-colluding semi-honest adversaries). Hence, one should only consider the efficiency of the protocols when deciding which to use. There are several efficiency parameters that may be taken into account, of which we only address the total communication costs of the protocol.

As in the distributed protocol parties need not send information about their entire inputs $R_{i,j}$, we have that if the number of iterations needed to solve a DisCSP is small,

[7] Here n is the number of assignments ('seats') transferred by messages

the distributed solution would generally be more efficient, in terms of total communication, when compared with the centralized solution (that requires sending $O(n^2m^2)$ bits during its initialization). On the other hand, the total communication costs of running a single iteration are generally higher in the distributed solution, and hence, when the number of iterations is high the centralized protocol becomes more efficient.

A strategy for running the protocols may hence be to start running the distributed protocol, and move to a centralized protocol if the number of iterations grows too high. A reasonable turning point is when the total communication costs of the distributed protocol exceed those of the initialization for the centralized protocol.

8 Summary

This work considered the problem of constructing efficient secure protocols for DisCSP. We showed that the direction pointed to by [YKH05] is productive, and improved on previous results by (i) eliminating the leakage in the protocol by [YKH05], (ii) eliminating the need for external servers and, (iii) enabling the use of advanced search techniques and heuristics. The overhead of running a single iteration of our protocol is a factor $O(n)$ greater than that of [YKH05]. However, as we allow the incorporation of sophisticated heuristics, the gap may be bridged (by lowering the overall number of iterations), retaining the other benefits of our construction. We also showed how distributed protocols for secure DisSCP may be constructed and addressed the related problem of constructing input-oblivious communication pattern DisCSP protocols. Finally, we discussed how our distributed and centralized solutions may be combined.

References

[ACM88] *Proc. Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, 2–4 May 1988.

[AIR01] Bill Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology—EUROCRYPT 2001*, Innsbruck, Austria, May 2001.

[BD03] A. Beimel and S. Dolev. Buses for anonymous message delivery. *Journal of Cryptology*, 16:1:25–39, 2003.

[BDF$^+$05] R. Bejar, C. Domshlak, C. Fernandez, , K. Gomes, B. Krishnamachari, B.Selman, and M.Valls. Sensor networks and distributed csp: communication, computation and complexity. *Artificial Intelligence*, 161:1-2:117–148, January 2005.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In ACM [ACM88], pages 1–10.

[BM03] I. Brito and P. Meseguer. Distributed forward checking. In *Proc. CP-2003*, pages 801–806, September, Ireland, 2003.

[BMBM05] C. Bessiere, A. Maestre, I. Brito, and P. Meseguer. Asynchronous backtracking without adding links: a new member in the abt family. *Artificial Intelligence*, 161:1-2:7–24, January 2005.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In ACM [ACM88], pages 11–19.

[Dec03] Rina Dechter. *Constraints Processing*. Morgan Kaufman, 2003.

[Dol00] S. Dolev. *Self Stbilization*. MIT Press, 2000.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, 25–27 May 1987.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume II Basic Applications*. Cambridge University Press, 2004.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Lecture Notes in Computer Science*, 1109:104–113, 1996.

[Lyn97] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Series, 1997.

[ML04] A. Meisels and O. Lavee. Using additional information in discsp search. In *Proc. 5th workshop on distributed constraints reasoning, DCR-04*, Toronto, 2004.

[MZ03] A. Meisels and R. Zivan. Asynchronous forward-checking for distributed csps. In W. Zhang, editor, *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2003.

[NN01] M. Naor and K. Nissim. Communication complexity and secure function evaluation. *ECCC – Electronic Colloquium on Computational Complexity, Report TR01-062*, 2001.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. 31st Annual ACM Symposium on Theory of Computing*, pages 245–254, Atlanta, Georgia, May 1999.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SIAM Symposium on Discrete Algorithms (SODA)*, pages 448–457, Washington, D.C., January 2001.

[NSHF04] T. Nguyen, D. Sam-Hroud, and B. Faltings. Dynamic distributed backjumping. In *Proc. 5th workshop on distributed constraints reasoning DCR-04*, Toronto, September 2004.

[SF05] M. C. Silaghi and B. Faltings. Asynchronous aggregation and consistency in distributed constraint satisfaction. *Artificial Intelligence*, 161:1-2:25–54, January 2005.

[SGM96] G. Solotorevsky, E. Gudes, and A. Meisels. Modeling and solving distributed constraint satisfaction problems (dcsps). In *Constraint Processing-96*, pages 561–2, New Hamphshire, October 1996.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sil02] M. C. Silaghi. *Asynchronously Solving Problems with Privacy Requirements*. PhD thesis, Swiss Federal Institute of Technology (EPFL), 2002.

[Sil03] M. C. Silaghi. Solving a distributed csp with cryptographic multi-party computations, without revealing constraints and without involving trusted servers. In *Proc. Workshop on Distributed Constraint Reasoning DCR-03 IJCAI*, Acapulco, Mexico, 2003.

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

[WF02] R. J. Wallace and E. Freuder. Constraint-based multi-agent meeting scheduling: effects of agent heterogeneity on performance and privacy loss. In *Proc. 3rd workshop on distributed constrait reasoning, DCR-02*, pages 176–182, Bologna, 2002.

[Yao82] A.C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, 3–5 November 1982. IEEE.

[YKH05] M. Yokoo, K.Suzuki, and K. Hirayama. Secure distributed constraints satisfaction: Reaching agreement without revealing private information. *Artificial Intelligence*, 161:1-2:229–246, January 2005.

[Yok00] M. Yokoo. Algorithms for distributed constraint satisfaction problems: A review. *Autonomous Agents & Multi-Agent Sys.*, 3:198–212, 2000.

[ZM04] R. Zivan and A. Meisels. Concurrent dynamic backtracking for distributed csps. In *CP-2004*, pages 782–7, Toronto, 2004.