A Privacy Preserving Collusion Secure DCOP Algorithm

Tamir Tassa[1] , **Tal Grinshpoun**[2] and **Avishay Yanai**[3]
[1]The Open University, Ra'anana, Israel
[2]Ariel University, Ariel, Israel
[3]Bar-Ilan University, Ramat Gan, Israel
tamirta@openu.ac.il, talgr@ariel.ac.il, ay.yanay@gmail.com

Abstract

In recent years, several studies proposed privacy-preserving algorithms for solving Distributed Constraint Optimization Problems (DCOPs). All of those studies assumed that agents do not collude. In this study we propose the first privacy-preserving DCOP algorithm that is immune to coalitions, under the assumption of honest majority. Our algorithm – PC-SyncBB – is based on the classical Branch and Bound DCOP algorithm. It offers constraint, topology and decision privacy. We evaluate its performance on different benchmarks, problem sizes, and constraint densities. We show that achieving security against coalitions is feasible. As all existing privacy-preserving DCOP algorithms base their security on assuming solitary conduct of the agents, we view this study as an essential first step towards lifting this potentially harmful assumption in all those algorithms.

1 Introduction

Constraint optimization [Meseguer and Larrosa, 1995] is a powerful framework for describing optimization problems in terms of constraints. In many practical domains, such as Meeting Scheduling [Maheswaran *et al.*, 2004], Mobile Sensor nets [Farinelli *et al.*, 2008], and the Internet of Things [Lezama *et al.*, 2017], the constraints are enforced by distinct participants (agents). Hirayama and Yokoo [1997] termed such problems as Distributed Constraint Optimization Problems (DCOPs). Various algorithms for solving DCOPs have been proposed, some of which are complete [Gershman *et al.*, 2009; Hirayama and Yokoo, 1997; Mailler and Lesser, 2004; Modi *et al.*, 2005; Petcu and Faltings, 2005; Yeoh *et al.*, 2010], and some are incomplete [Farinelli *et al.*, 2008; Katagishi and Pearce, 2007; Ottens *et al.*, 2017; Zhang *et al.*, 2005].

The main motivation for DCOP research stems from the inherent distributed structure of many real-world problems, and the *privacy* concerns that are associated with this distribution. Léauté and Faltings [2013] offered the basic definitions of privacy in this framework. The four notions of privacy that they describe are: agent privacy, topology privacy, constraint privacy and decision privacy (see Section 2). Several studies considered a solution of DCOPs in a manner that preserves (some of) those privacy types.

This line of research began with the work of Silaghi and Mitra [2004]. They proposed a privacy-preserving solution to Distributed Weighted Constraint Satisfaction Problems (DisWCSPs); those are distributed problems that are similar to DCOPs, but differ from them in the distribution model and, consequently, in the related privacy targets. Their solution is strictly limited to small scale problems since it depends on an exhaustive search over all possible assignments. As their solution is based on the BGW protocol [Ben-Or *et al.*, 1988], it is immune against coalitions that involve less than half of the agents.

All of the subsequent studies considered DCOPs. The main motif in those studies was to develop privacy-preserving versions of existing algorithms. Greenstadt et al. [2007] devised a version of the DPOP algorithm [Petcu and Faltings, 2005], called SSDPOP. Léauté and Faltings [2013] proposed three privacy-preserving versions of DPOP that differ in their privacy guarantees and in their runtime performance. Grinshpoun and Tassa [2016] developed P-SyncBB, a privacy-preserving version of the complete search algorithm SyncBB [Hirayama and Yokoo, 1997]. Tassa et al. [2017] presented P-Max-Sum, a privacy-preserving version of the incomplete inference-based Max-Sum algorithm [Farinelli *et al.*, 2008]. Lastly, Grinshpoun et al. [2019] devised P-RODA, a secure implementation of region-optimal algorithms.

All of the above described works based their security on assuming *solitary* conduct of the agents. Alas, subsets of agents may try to collude and combine the information which they have in order to infer information on other agents. In this paper we suggest the first privacy-preserving DCOP algorithm that is immune against such coalitions. We depart from the SyncBB algorithm and devise a privacy-preserving algorithm that simulates its operation and provides topology, constraint and decision privacies, even in the presence of a coalition of agents, under the assumption of an honest majority (i.e., the size of the coalition is smaller than half the number of agents).

2 Definitions and Assumptions

A Distributed Constraint Optimization Problem (DCOP) is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a set of agents $A_1, A_2, \ldots, A_n$, $\mathcal{X}$ is a set of variables $X_1, X_2, \ldots, X_m$, $\mathcal{D}$ is a set of finite domains $D_1, D_2, \ldots, D_m$, and $\mathcal{R}$ is a set

of relations (constraints). Each variable X_i takes values in the domain D_i, and it is held by a single agent. Each constraint $C \in \mathcal{R}$ defines a non-negative cost for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times \cdots \times D_{i_k} \rightarrow [0, q]$, for some $1 \leq i_1 < \cdots < i_k \leq m$, and a publicly known maximal constraint cost q.

An *assignment* is a pair including a variable, and a value from that variable's domain. We denote by a_i the value assigned to the variable X_i. A *partial assignment* (PA) is a set of assignments in which each variable appears at most once. A constraint $C \in \mathcal{R}$ is *applicable* to a PA if all variables that are constrained by C are included in the PA. The cost of a PA is the sum of all applicable constraints to the PA. A *full assignment* is a partial assignment that includes all of the variables. The goal in Constraint Optimization Problems is to find a full assignment of minimal cost.

For simplicity, we assume that each agent holds exactly one variable, i.e., $n = m$. We let n denote hereinafter the number of agents and the number of variables. We consider a binary version of DCOPs, in which every $C \in \mathcal{R}$ constraints exactly two variables and takes the form $C_{i,j} : D_i \times D_j \rightarrow [0, q]$. These assumptions are customary in DCOP literature [Modi *et al.*, 2005; Petcu and Faltings, 2005].

Léauté and Faltings [2013] have distinguished between four notions of privacy. The notions of privacy which our proposed algorithm respects are: (a) *Topology privacy* – hiding from each agent the topological structures in the constraint graph beyond his[1] own direct neighborhood in the graph; (b) *Constraint privacy* – hiding from each agent the constraints in which he is not involved; and (c) *Decision privacy* – hiding from each agent the final assignments to other variables.

Like in all prior art on privacy-preserving DCOP algorithms, we too assume that the agents are semi-honest, namely, they follow the prescribed protocol but try to glean more information than allowed from the protocol transcript. In contrast to prior art, we assume that (less than half of the) agents may collude in order to combine their inputs and messages received during the execution of the protocol, for the purpose of extracting private information on other agents.

3 A Secure Synchronous Branch and Bound

Synchronous Branch-and-Bound (SyncBB) [Hirayama and Yokoo, 1997] was the first complete algorithm for solving DCOPs. SyncBB operates in a completely sequential manner, a fact that inherently renders its synchronous behavior. It assumes a static public ordering of the agents, $A_1, \ldots, A_n$. The search space of the problem is traversed by each agent assigning a value to his variable and passing the *current partial assignment* (CPA) to the next agent in the order, along with the current cost of the CPA. After an agent completes assigning all values in the domain to his variable, he *backtracks*, i.e., he sends the CPA back to the preceding agent. To prevent exhaustive traversal of the entire search space, the agents maintain an *upper bound*, which is the cost of the best solution that was found thus far. The algorithm keeps comparing the costs of CPAs and the current upper bound, in order to *prune* the search space.

[1] We use the masculine form for simplicity.

Herein we devise a secure implementation of SyncBB, called PC-SyncBB (Privacy-preserving and Collusion-resistant SyncBB). Another secure implementation of SyncBB, called P-SyncBB, was previously introduced by Grinshpoun and Tassa [2014; 2016]. The two algorithms are fundamentally different. While in P-SyncBB agents are exposed to sensitive information such as assignments of other agents, costs of CPAs, and the value of the upper bound, PC-SyncBB totally avoids such information disclosure. Hence, the outline of PC-SyncBB is simpler than that of P-SyncBB, because there is no need to implement mechanisms for preventing illegal inferences that can be deduced from such information. On the other hand, as in PC-SyncBB much less information is revealed, and as PC-SyncBB is designed to be resistant to coalitions (while P-SyncBB's security is jeopardized already when two agents collude), the secure multiparty computational tasks in PC-SyncBB are much harder. Hence, the cryptographic approach taken in PC-SyncBB is completely different, and it is much more involved than the corresponding one in P-SyncBB.

3.1 Preliminaries

General Assumptions and Notations

The design of PC-SyncBB is based on the following general assumptions:

- There is a static public ordering of the agents, $A_1, \ldots, A_n$.
- The upper bound on the cost of any possible solution is $q_\infty := \binom{n}{2}q + 1$, and it is known to all agents. In addition, all agents agree upfront on an integer S greater than $2q_\infty$.
- For every pair of indices $1 \leq t < k \leq n$, $\Gamma(t, k)$ is a Boolean predicate that equals `true` iff X_t and X_k are constrained. Then, $I_k^- := \{t : 1 \leq t < k \text{ and } \Gamma(t, k)\}$ and $I_k^+ := \{t : k < t \leq n \text{ and } \Gamma(k, t)\}$ are sets containing the indices of all agents that precede/follow A_k in the order and whose variable is constrained with X_k. We also let $I_k := I_k^- \cup I_k^+$.

Value Ordering

Each agent A_k maintains two value orderings over his domain D_k. Each of those orderings can be described by a vector of length $|D_k|$. The first ordering, denoted $\mathbf{u}_k$, is fixed and known to all agents A_t such that $t \in I_k$. Then if A_t and A_k are constrained, they can describe their constraint $C_{t,k}$ as a matrix $M_{t,k}$ of $|D_t|$ rows and $|D_k|$ columns, where the value in the r-th row and s-th column is

$$M_{t,k}(r, s) = C_{t,k}(\mathbf{u}_t(r), \mathbf{u}_k(s)) \,. \tag{1}$$

The second ordering, denoted $\mathbf{w}_k$, is generated at random by A_k whenever he begins a new traversal over his domain. That ordering, which is kept secret from all other agents, determines the order in which that agent will scan the values in his domain during that stage of the search. Agent A_k generates such an ordering each time a CPA is passed to him from the preceding agent A_{k-1}.

Internal Variables

Every agent A_k maintains the following variables:

- sCPA_k is an array of length n that holds additive shares in the cost of the CPA. Assume that agents A_t and A_k are constrained and that $C_{t,k}$ is applicable to the CPA. Then the cost of the CPA includes, as one of its addends, the value $C_{t,k}(X_t, X_k)$. In such a case $\text{sCPA}_k(t)$ and $\text{sCPA}_t(k)$ will both store random values in $\mathbb{Z}_S$ so that

$$\text{sCPA}_t(k) + \text{sCPA}_k(t) = C_{t,k}(X_t, X_k) \bmod S\,. \quad (2)$$

  If, on the other hand, $C_{t,k}$ is *not* applicable to the CPA (i.e. the CPA does not include X_k or X_t or both), then $\text{sCPA}_k(t) = \text{sCPA}_t(k) = 0$. In view of the above, the overall cost of the CPA, at any stage of the algorithm's run, equals

$$Cost(CPA) = \sum_{k=1}^{n} \sum_{t \in I_k} \text{sCPA}_k(t) \bmod S\,. \quad (3)$$

- sUB_k holds an additive share in the upper bound (the cost of the best full assignment that was discovered thus far). Each such share is random and uniformly distributed over $\mathbb{Z}_S$. At any stage of the algorithm's run,

$$UpperBound = \sum_{k=1}^{n} \text{sUB}_k \bmod S\,. \quad (4)$$

- p_k is a pointer to a value in the ordering $\mathbf{w}_k$. The current assignment to X_k is given by $\mathbf{w}_k(p_k)$.
- $OptimalSetting_k$ stores the assignment to X_k in the currently best full assignment that was found thus far.

3.2 The PC-SyncBB Algorithm

The PC-SyncBB algorithm is given in Algorithm 1, which we proceed to describe.

The Procedure init

Every agent A_k initializes all entries in his vector sCPA_k as well as p_k to zero (Lines 1-2). Then, every agent A_k, $k > 1$, initializes sUB_k to zero, while A_1 initializes it to q_∞ (Lines 3-6). Such settings imply that $\sum_{k=1}^{n} \text{sUB}_k = q_\infty \bmod S$, in agreement with Eq. (4) (since the initial upper bound is set to q_∞). Finally, the procedure init triggers the search by having A_1 call the procedure assign_CPA (Line 7).

The Procedure assign_CPA

If this procedure is called when $p_k = 0$, it means that A_k now begins a new traversal over his domain. Hence, in such a case he generates a new random ordering, $\mathbf{w}_k$, of D_k (Lines 8-9). In order to move to the next value in $\mathbf{w}_k$, A_k increments the pointer p_k (Line 10). If p_k becomes greater than $|D_k|$ it means that the domain D_k was already fully scanned, so A_k performs the procedure backtrack (discussed below) in order to return the search torch back to the preceding agent A_{k-1} (Lines 11-12). Otherwise, A_k assigns $v := \mathbf{w}_k(p_k)$ to X_k (Line 14). Consequently, as X_k has a new value, the CPA's cost is changed, so new random shares of that cost must be computed. This is done by calling the sub-protocol update_shares_in_CPA(k, v) (Line 15), which recomputes $\text{sCPA}_k(t)$ and $\text{sCPA}_t(k)$, for all $t \in I_k^-$, so that the right-hand side of Eq. (3) equals the new CPA's cost. (We discuss that sub-protocol in Section 3.3.)

We now separate the discussion according to the index k of the operating agent. If $k = n$, then a new full assignment is reached. It is needed to compare its cost, which equals $\sum_{k=1}^{n} \sum_{t \in I_k} \text{sCPA}_k(t) \bmod S$, Eq. (3), to the current upper bound, $\sum_{k=1}^{n} \text{sUB}_k \bmod S$, (Eq. (4)). This comparison must be done in a secure manner. To that end, A_n in-

Algorithm 1 – PC-SyncBB (executed by agent A_k)

```
procedure init
 1: sCPA_k(t) ← 0 for all 1 ≤ t ≤ n
 2: p_k ← 0
 3: if k > 1 do
 4:     sUB_k ← 0
 5: else
 6:     sUB_k ← q_∞
 7:     assign_CPA()

procedure assign_CPA
 8: if p_k = 0 do
 9:     Generate a new random ordering of D_k into w_k
10: p_k ← p_k + 1
11: if p_k > |D_k| do
12:     backtrack()
13: else
14:     X_k ← v := w_k(p_k)
15:     update_shares_in_CPA(k, v)
16:     if k = n do
17:         if compare_CPA_cost_to_upper_bound() = true do
18:             broadcast(NEW_OPTIMUM_FOUND)
19:         assign_CPA()
20:     else
21:         if compare_CPA_cost_to_upper_bound() = false do
22:             assign_CPA()
23:         else
24:             send(CPA_MSG) to A_{k+1}

procedure backtrack
25: if k > 1 do
26:     sCPA_k(t) ← 0 for all t ∈ I_k^-
27:     send(ZERO_SHARE_MSG,k) to A_t for all t ∈ I_k^-
28:     send(BACKTRACK_MSG) to A_{k-1}
29: else
30:     broadcast(COMPLETE)

when received (NEW_OPTIMUM_FOUND) do
31: sUB_k ← Σ_{t∈I_k} sCPA_k(t)
32: OptimalSetting_k ← X_k
when received (CPA_MSG) do
33: p_k ← 0
34: assign_CPA()
when received (ZERO_SHARE_MSG,k') do
35: sCPA_k(k') ← 0
when received (BACKTRACK_MSG) do
36: assign_CPA()
when received (COMPLETE) do
37: X_k ← OptimalSetting_k
38: Terminate
```

vokes compare_CPA_cost_to_upper_bound (Line 17), a secure multi-party sub-protocol that we discuss in Section 3.4. It returns `true` if the cost of the current full assignment is lower than the upper bound, namely, if

$$\sum_{k=1}^{n}\sum_{t\in I_k} \text{sCPA}_k(t) \bmod S < \sum_{k=1}^{n} \text{sUB}_k \bmod S\,, \qquad (5)$$

and `false` otherwise. If the current full assignment does improve the upper bound, then A_n broadcasts the message NEW_OPTIMUM_FOUND (Line 18). Upon receiving such a message, every agent A_k stores the sum of his current shares, $\sum_{t\in I_k}$ sCPA$_k(t)$, in sUB$_k$ and he also stores the current assignment of X_k in $OptimalSetting_k$ (Lines 31-32). Finally, whether the current full assignment is a new optimum or not, A_n calls the procedure assign_CPA again in order to test the next value in his domain (Line 19).

If $k < n$, the agents examine the possibility to prune the search space: they first check whether the CPA's cost is already greater than or equal to the upper bound, by invoking compare_CPA_cost_to_upper_bound (Line 21). If it returns `false` then Eq. (5) does not hold, i.e., the cost of the CPA is already greater than or equal to the upper bound. In such a case there is no point in pursuing the current path in the search space, so A_k calls the procedure assign_CPA again in order to test the next value in his domain (Line 22). Otherwise, A_k passes the torch onward to A_{k+1} (by sending him the message CPA_MSG in Line 24) in order to continue the search over CPAs with the current k-prefix. When A_{k+1} receives the message CPA_MSG, he zeroes the pointer p_{k+1} to his domain D_{k+1}, in order to start traversing all values in D_{k+1} as possible extensions to the current k-CPA, and then he calls the procedure assign_CPA (Lines 33-34).

The Procedure backtrack

When agent A_k, $k > 1$, executes the procedure backtrack, he does two things. First, he zeros all entries in sCPA$_k$ (Line 26) and sends a ZERO_SHARE_MSG message, with his index k, to all agents that precede him and are constrained with him (Line 27). Any such agent, upon receiving the ZERO_SHARE_
MSG message, zeroes the relevant share in his own array (Line 35). As a result of the above two actions, Eq. (3) still holds for the reduced CPA that is obtained after this backtracking. Afterwards, A_k sends a BACKTRACK_MSG message to A_{k-1} (Line 28). When the latter receives that message, he calls assign_CPA in order to change the assignment of his variable to the next value in his domain and proceed the search with the new modified CPA (Line 36).

When A_1 performs backtrack, it means that he completed a traversal of D_1, and, consequently, the entire search space $(D_1 \times \cdots \times D_n)$ was scanned. Therefore, the algorithm terminates with the last optimum found being the global optimum. In such a case A_1 broadcasts the message COMPLETE (Line 30). When receiving such a message, every agent A_k assigns to his variable X_k the value $OptimalSetting_k$ (which was his assignment in the last optimal solution that was found) and then he terminates (Lines 37-38).

3.3 The Sub-protocol update_shares_in_CPA

Before starting PC-SyncBB, each of the agents A_k, $k < n$, creates a key pair in a Paillier cipher [Paillier, 1999] and sends the corresponding public key to A_t for all $t \in I_k^+$. Denote by $\mathcal{E}_k$ the encryption function in A_k's cipher and by ν_k the corresponding modulus. Then $\mathcal{E}_k$ is a function from $\mathbb{Z}_\nu$ to $\mathbb{Z}^*_{\nu^2}$ and it is additively homomorphic, in the sense that for every two plaintexts x and y, $\mathcal{E}_k(x+y) = \mathcal{E}_k(x)\cdot\mathcal{E}_k(y)$, where addition is modulo ν and multiplication is modulo ν^2. The Paillier cipher is probabilistic, in the sense that the encryption function depends also on a random string (so that every plaintext x has a large number of possible ciphertexts $\mathcal{E}_k(x)$). It is known to be semantically secure [Goldwasser and Micali, 1982].

After creating $\mathcal{E}_k$, A_k computes a vector $\mathbf{z}_k^1$ of length $|D_k|$ where $\mathbf{z}_k^1(1) = \mathcal{E}_k(1)$ and $\mathbf{z}_k^1(i) = \mathcal{E}_k(0)$ for all $2 \le i \le |D_k|$. It is important to compute the latter $|D_k|-1$ encryptions with $|D_k|-1$ independently selected random strings. Then, A_k defines the vectors $\mathbf{z}_k^i = CRS\left(\mathbf{z}_k^{i-1}\right)$, for $2 \le i \le |D_k|$, where $CRS(\cdot)$ is a circular right-shift by one position of the vector entries. Hence, $\mathbf{z}_k^i$ encrypts the vector $(0,\ldots,0,1,0,\ldots,0)$ where the 1 appears in the ith entry, $1 \le i \le |D_k|$. Given the manner in which those vectors were computed and the probabilistic and semantic security properties of the Paillier cipher, a polynomially-bounded adversary who gets any random sequence of those vectors (i.e. $\mathbf{z}_k^{i_1}, \mathbf{z}_k^{i_2}, \ldots$) will not be able to distinguish between the $\mathcal{E}_k(1)$ and the $\mathcal{E}_k(0)$ entries in them (with a non-negligible probability of success).

We are now ready to describe the sub-protocol update_shares_in_CPA (Algorithm 2). It is triggered by A_k whenever he assigns a new value v to his variable, X_k. When that happens, it is needed to update the shares of all agents $A_1,\ldots,A_k$ so that the validity of Eq. (3) is maintained. The shares that should be modified in wake of such an assignment are sCPA$_k(t)$ and sCPA$_t(k)$ for all $t \in I_k^-$. Those shares will be modified so that, in view of Eq. (2), the sum of sCPA$_k(t)$ and sCPA$_t(k)$, for any fixed $t \in I_k^-$, will equal $C_{t,k}(X_t, X_k)$ for the current assignments of X_t and X_k (X_k's assignment equals v, and it is passed to the sub-protocol as an input).

Assume that $t \in I_k^-$. Then the contribution of the pair X_t and X_k to the CPA is $M_{t,k}(r,s)$, where $\mathbf{u}_t(r) = X_t$ and $\mathbf{u}_k(s) = X_k$ (see Eq. (1)). Recall that A_t does not know s while A_k does not know r. In order to compute the new respective shares, sCPA$_k(t)$ and sCPA$_t(k)$, so that Eq. (2) holds, these two agents perform the following computation.

When A_t performed last time the procedure assign_CPA and set there the current assignment to X_t, he called update_shares_in_CPA (Algorithm 2), see Line 15 in PC-SyncBB. In Line 8 of Algorithm 2 he sent to all agents in I_t^+ the vector $\mathbf{z}_t^j$ which encodes his current assignment. Going back to the present, when A_k executes update_shares_in_CPA he holds a vector $\mathbf{z}_t$ that he received from A_t for every $t \in I_k^-$. That vector equals $\mathbf{z}_t^r$, where r is the index in $\mathbf{u}_t$ in which the current assignment to X_t is stored. Even though A_k cannot infer from $\mathbf{z}_t$ the current value of X_t, he can still correctly update his shares vis-a-vis A_t. To that end, he computes

$$y_t := \prod_{i=1}^{|D_t|} \mathbf{z}_t(i)^{[(M_{t,k}(i,s)-\rho) \bmod S]}\,, \qquad (6)$$

Algorithm 2 The sub-protocol update_shares_in_CPA

when received k, the index of the agent A_k that invokes the procedure, and v, A_k's current assignment
1: **for all** $t \in I_k^-$ **do**
2: A_k selects uniformly at random $\rho \in \mathbb{Z}_S$.
3: A_k computes y_t as given in Eq. (6), where $\mathbf{z}_t$ is the vector that A_k received from A_t in the last time.
4: A_k sends the computed y_t to A_t.
5: A_t sets $\text{sCPA}_t(k) \leftarrow \mathcal{E}_t^{-1}(y_t)$.
6: A_k sets $\text{sCPA}_k(t) \leftarrow \rho$.
7: **if** $k < n$ **do**
8: A_k sends to all A_t where $t \in I_k^+$ the vector $\mathbf{z}_k^j$ where j is the index for which $\mathbf{u}_k(j) = v$.

where s is the index of the entry in $\mathbf{u}_k$ that holds v – the current assignment to X_k, and ρ is a value selected uniformly at random (independently for each A_t) from $\mathbb{Z}_S$. The key observation here is the following.

Lemma 3.1 *The homomorphism of $\mathcal{E}_t$ implies that $y_t = \mathcal{E}_t([(M_{t,k}(r,s) - \rho) \text{ mod } S])$.*

Next, A_k sends y_t to A_t who decrypts it and stores it in $\text{sCPA}_t(k)$. In view of Lemma 3.1, A_t obtains $\text{sCPA}_t(k) = (M_{t,k}(r,s) - \rho) \mod S$ whereas A_k sets $\text{sCPA}_k(t) = \rho \mod S$ (Algorithm 2, Lines 5-6). Those two uniformly random shares satisfy $\text{sCPA}_t(k) + \text{sCPA}_k(t) = M_{t,k}(r,s) \mod S$, which fulfils the required equality in Eq. (2).

The above described updates are carried out by A_k and A_t for all $t \in I_k^-$. After completing all those updates, the updated shares satisfy Eq. (3).

3.4 compare_CPA_cost_to_upper_bound

The sub-protocol compare_CPA_cost_to_upper_bound verifies the inequality in Eq. (5). Agent A_k, $1 \leq k \leq n$, holds two integers modulo S: $a_k := \sum_{t \in I_k} \text{sCPA}_k(t) \bmod S$ and $b_k := \text{sUB}_k \bmod S$. The goal is to determine whether $\alpha := \sum_{k=1}^n a_k \bmod S$ is smaller than $\beta := \sum_{k=1}^n b_k \bmod S$ or not.

To this end, we make use of a secure multi-party computation (MPC) protocol [Yao, 1982]. An MPC protocol for some function f allows a set of n distrustful parties $A_1, \ldots, A_n$, where A_i possesses a private input x_i, to compute $y \leftarrow f(x_1, \ldots, x_n)$, while preserving the privacy of the parties. Namely, at the end of the protocol the parties learn y, but nothing beyond that on inputs of other parties. Almost all practical MPC protocols work with the underlying Boolean/Arithmetic circuit as the computational model. Therefore, to securely compute the function f, the parties first need to agree on a Boolean/Arithmetic circuit C that implements f. The runtime of such computations depends on n and the size $|C|$ of the circuit C (the number of gates in it).

In PC-SyncBB, we managed to shrink the usage of a general purpose MPC protocol to the computation of Eq. (5). Specifically, we use the protocol of Ben-Efraim and Omri [2017] for the function $f(x_1, \ldots, x_n)$ where A_k's input is $x_k = b_k - a_k \mod S$ and the function f returns `true` if Eq. (5) holds and `false` otherwise.

The protocol of Ben-Efraim and Omri [2017] is secure under the assumption that less than $n/2$ parties collude. It proceeds in two phases, called "offline" and "online". In the offline phase the parties do not yet know their inputs, but can prepare the raw materials required for the computation. The online phase begins once the parties know their inputs. In practice, this allows the parties to perform the offline phase in advance, even without having their inputs (or even knowing the domains), and perform the fast online phase once the inputs are ready. Due to page limitations, we defer the detailed description of this sub-protocol to the full version.

3.5 Properties of PC-SyncBB

The main properties of this algorithm are stated below.[2]

Theorem 3.2 *PC-SyncBB is complete and sound.*

Theorem 3.3 *PC-SyncBB provides constraint-, topology-, and assignment/decision-privacy. Even if any subset $\mathcal{B} \subsetneq \mathcal{A}$ of agents collude, where $|\mathcal{B}| < n/2$, they would not be able to infer information on (values or existence of) constraints between two agents outside the coalition, or on value assignments or final decisions of such agents.*

It is important to note that, like *all* preceding papers on privacy-preserving solution of DCOPs, our algorithm does not guarantee *perfect* privacy, as it may leak some very benign information on the constraint graph topology. While achieving perfect privacy is possible, in theory, in any multi-party computation, it is very hard to do so while maintaining practicality. Hence, in almost all studies that deal with privacy-preserving solutions of practical problems, one accepts benign information leakages.

4 Experimental Evaluation

We begin by evaluating the runtime of the compare_CPA_cost_to_upper_bound sub-protocol, which is a central and computationally expensive part of PC-SyncBB. For efficiency and reproducibility we used the original implementation of Ben-Efraim and Omri [2017].[3] The executions were over LAN with EC2 machines of type c5.large in Amazon's North Virginia data center with every agent running on a separate machine. We measured performance for various values of n, where q (the maximum value of a single binary constraint) is set to 100. Hence, as the maximum cost of any solution is $q_\infty := \binom{n}{2}q + 1$ and S is set to be the smallest power of 2 greater than $2q_\infty$, then n fully determines $\ell = \log S$ and, consequently, also the size of the circuit C that the protocol uses. Table 1 gives, for each n, the bit-length ℓ of the agents' inputs, the overall circuit size (number of gates), and the average runtimes over 100 executions for the offline and online phases of the protocol.

Now we turn to the runtime performance evaluation of the full PC-SyncBB algorithm. In order to asses the toll of privacy preservation, we compare PC-SyncBB (offline+online) to other algorithms that maintain the Branch & Bound structure – P-SyncBB [Grinshpoun and Tassa, 2016] that preserves privacy only under the assumption of non-colluding

[2] Proofs are given at http://arxiv.org/abs/1905.09013.

[3] https://github.com/cryptobiu/Protocols/tree/master/Concrete_Efficiency_Improvements_to_Multiparty_Garbling_with_an_Honest_Majority

n	5	7	9	11	13	15	17	19
ℓ	11	13	13	14	14	15	15	16
$\|C\|$	184	336	448	610	732	924	1056	1278
offline	6.7	12.4	20.2	32.3	47.2	72.0	94.3	135.3
online	0.51	0.85	1.3	1.6	2.4	2.5	2.7	3.6

Table 1: Bit length, circuit size and runtime (msecs) of the Ben-Efraim-Omri as a function of n.

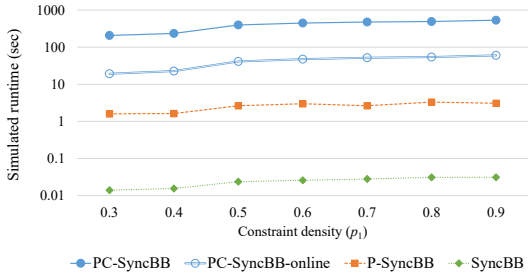

Figure 1: Varying p_1 (random DCOPs).

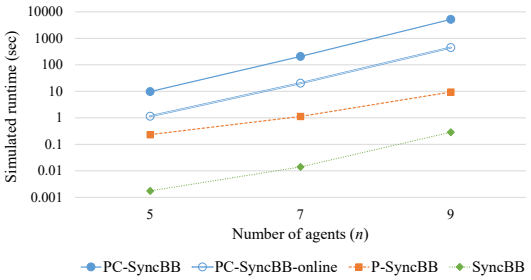

Figure 2: Varying n (random DCOPs).

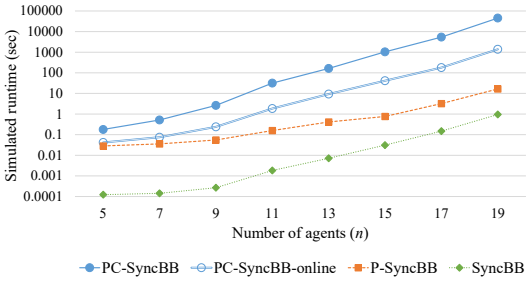

Figure 3: Varying n (graph coloring problems).

agents, and the basic insecure SyncBB [Hirayama and Yokoo, 1997]. We also present separately the online requirements of PC-SyncBB. The algorithms were implemented and executed in the AgentZero simulator [Lutati *et al.*, 2014], running on a hardware comprised of an Intel i7-6820HQ processor and 32GB memory, except for the calls to the compare_CPA_cost_to_upper_bound procedure that were executed on the machines from Amazon's North Virginia data center, in order to simulate as realistically as possible a truly distributed environment. We followed the *simulated time* [Sultanik *et al.*, 2008] approach in all the subsequent experiments. The results are shown in a logarithmic scale and are the average over 50 problem instances (for each setting/benchmark).

The first benchmark consists of unstructured randomly generated DCOPs on which we perform two experiments. In the first experiment, presented in Figure 1, we fix the number of agents to $n = 7$ and the domain sizes to 6, and vary the constraint density $0.3 \leq p_1 \leq 0.9$. (Using lower density values $p_1 < 0.3$ results in unconnected constraint graphs.) It is clear that constraint density only mildly affects the runtime performance of all the evaluated algorithms. However, the toll of privacy preservation is evidently high, with each layer of protection adding about two orders of magnitude to the runtime. Specifically, the online part of PC-SyncBB requires about one order of magnitude more time than P-SyncBB.

In the second experiment, shown in Figure 2, we fix the constraint density to $p_1 = 0.3$ and the domain sizes to 6, and vary the number of agents $5 \leq n \leq 9$. Here and in the following scalability experiments we use a cutoff time of 30 minutes for online PC-SyncBB. It is clear that the number of agents has a major effect on the performance of all the evaluated algorithms, in accordance with known results regarding the scalability of Branch & Bound algorithms in computationally hard problems. Interestingly, P-SyncBB scales slightly better, probably due to its inherent use of sorted value ordering.

Similar scalability phenomena are also observed in more structured benchmarks. Figure 3 depicts the runtime performance on distributed 3-color graph coloring problems ($p_1 = 0.4$, $5 \leq n \leq 19$) in which each pair of equal values of constrained agents imposes a random and *private* cost. The structure in these problems lies in the diagonal constraint matrices between every pair of neighboring agents. An experiment on scale-free networks, generated according to the Barabási-Albert model [Barabási and Albert, 1999], produced almost the same graph as in Figure 2 and is thus omitted.

5 Conclusion

We proposed herein PC-SyncBB, the first privacy-preserving DCOP algorithm which is secure against coalitions. We analyzed the properties of the algorithm and evaluated its performance. Our experiments demonstrate that PC-SyncBB is feasible for moderately-sized problems.

A major limitation on scalability is due to the protocol of Ben-Efraim and Omri [2017] that is invoked by compare_CPA_cost_to_upper_bound. In the future we intend to explore other directions that could yield much more efficient implementations of that sub-protocol. If those constructions prove more efficient, we intend to consider more challenging settings, with larger coalitions or malicious agents. While raising the security bar usually increases runtime and communication costs, it is important to come up with such solutions in order to enlarge and diversify the toolkit available for implementations in various application settings. Finally, as all existing privacy-preserving DCOP algorithms base their security on assuming solitary conduct of the agents, we view this study as an essential first step towards lifting this potentially harmful assumption in all those algorithms. In particular, it is necessary to develop privacy-preserving and collision-secure implementations of other DCOP algorithms (e.g., inference-based), and especially of incomplete algorithms that could offer better scalability.

References

[Barabási and Albert, 1999] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[Ben-Efraim and Omri, 2017] A. Ben-Efraim and E. Omri. Concrete efficiency improvements for multiparty garbling with an honest majority. In *LATINCRYPT*, 2017.

[Ben-Or *et al.*, 1988] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.

[Farinelli *et al.*, 2008] A. Farinelli, A. Rogers, A. Petcu, and N.R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*, pages 639–646, 2008.

[Gershman *et al.*, 2009] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding for distributed COPs. *Journal of Artificial Intelligence Research*, 34:61–88, 2009.

[Goldwasser and Micali, 1982] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377, 1982.

[Greenstadt *et al.*, 2007] R. Greenstadt, B. Grosz, and M.D. Smith. SSDPOP: improving the privacy of DCOP with secret sharing. In *AAMAS*, pages 171:1–171:3, 2007.

[Grinshpoun and Tassa, 2014] T. Grinshpoun and T. Tassa. A privacy-preserving algorithm for distributed constraint optimization. In *AAMAS*, pages 909–916, 2014.

[Grinshpoun and Tassa, 2016] T. Grinshpoun and T. Tassa. P-SyncBB: A privacy preserving branch and bound DCOP algorithm. *Journal of Artificial Intelligence Research*, 57:621–660, 2016.

[Grinshpoun *et al.*, 2019] Tal Grinshpoun, Tamir Tassa, Vadim Levit, and Roie Zivan. Privacy preserving region optimal algorithms for symmetric and asymmetric DCOPs. *Artificial Intelligence*, 266:27–50, 2019.

[Hirayama and Yokoo, 1997] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In *CP*, pages 222–236, 1997.

[Katagishi and Pearce, 2007] H. Katagishi and J.P. Pearce. Kopt: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In *DCR*, 2007.

[Léauté and Faltings, 2013] Thomas Léauté and Boi Faltings. Protecting privacy through distributed computation in multi-agent decision making. *Journal of Artificial Intelligence Research*, 47:649–695, 2013.

[Lezama *et al.*, 2017] F. Lezama, J. Palominos, A.Y. Rodríguez-González, A. Farinelli, and E. Munoz de Cote. Agent-based microgrid scheduling: An ICT perspective. *Mobile Networks and Applications*, 2017.

[Lutati *et al.*, 2014] Benny Lutati, Inna Gontmakher, Michael Lando, Arnon Netzer, Amnon Meisels, and Alon Grubshtein. Agentzero: A framework for simulating and evaluating multi-agent algorithms. In *Agent-Oriented Software Engineering*, pages 309–327. Springer, 2014.

[Maheswaran *et al.*, 2004] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce, and P. Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS*, pages 310–317, 2004.

[Mailler and Lesser, 2004] R. Mailler and V.R. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS*, pages 438–445, 2004.

[Meseguer and Larrosa, 1995] P. Meseguer and J. Larrosa. Constraint satisfaction as global optimization. In *IJCAI*, pages 579–584, 1995.

[Modi *et al.*, 2005] P.J. Modi, W.M. Shen, M. Tambe, and M. Yokoo. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.

[Ottens *et al.*, 2017] B. Ottens, C. Dimitrakakis, and B. Faltings. Duct: An upper confidence bound approach to distributed constraint optimization problems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(5):69, 2017.

[Paillier, 1999] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, pages 223–238, 1999.

[Petcu and Faltings, 2005] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[Silaghi and Mitra, 2004] M.C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In *IAT*, pages 531–535, 2004.

[Sultanik *et al.*, 2008] E. Sultanik, R. N. Lass, and W. C. Regli. DCOPolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In *AAMAS (demos)*, pages 1667–1668, 2008.

[Tassa *et al.*, 2017] T. Tassa, T. Grinshpoun, and R. Zivan. Privacy preserving implementation of the Max-Sum algorithm and its variants. *Journal of Artificial Intelligence Research*, 59:311–349, 2017.

[Yao, 1982] A.C. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.

[Yeoh *et al.*, 2010] W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Journal of Artificial Intelligence Research*, 38:85–133, 2010.

[Zhang *et al.*, 2005] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 161(1-2):55–87, 2005.