Eighth International Conference on Principles and Practice of Constraint Programming (CP-2002)

Secure Distributed Constraint Satisfaction: Reaching Agreement without Revealing Private Information

Makoto Yokoo[1], Koutarou Suzuki[2] and Katsutoshi Hirayama[3]

[1] NTT Communication Science Laboratories, NTT Corporation
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0237, Japan
url: www.kecl.ntt.co.jp/csl/ccrg/members/yokoo/
e-mail: yokoo@cslab.kecl.ntt.co.jp
[2] NTT Information Sharing Platform Laboratories, NTT Corporation
1-1 Hikari-no-oka, Yokosuka, Kanagawa 239-0847, Japan
url: info.isl.ntt.co.jp/~koutarou/
e-mail: koutarou@isl.ntt.co.jp
[3] Kobe University of Mercantile Marine
5-1-1 Fukae-minami-machi, Higashinada-ku, Kobe 658-0022, Japan
url: www-jo.ti.kshosen.ac.jp/~hirayama/
e-mail: hirayama@ti.kshosen.ac.jp

Abstract. This paper develops a secure distributed Constraint Satisfaction algorithm. A Distributed Constraint Satisfaction Problem (DisCSP) is a CSP in which variables and constraints are distributed among multiple agents. A major motivation for solving a DisCSP without gathering all information in one server is the concern about privacy/security. However, existing DisCSP algorithms leak some information during the search process and privacy/security issues are not dealt with formally. Our newly developed algorithm utilizes a public key encryption scheme. In this algorithm, multiple servers, which receive encrypted information from agents, cooperatively perform a search process that is equivalent to a standard chronological backtracking. This algorithm does not leak any private information, i.e., neither agents nor servers can obtain any additional information on the value assignment of variables that belong to other agents.

1 Introduction

A Distributed Constraint Satisfaction Problem (DisCSP) is a constraint satisfaction problem in which variables and constraints are distributed among multiple agents. Since various application problems in multi-agent systems can be formalized as DisCSPs, there have been many works on this topic in the last decade [5, 7, 10, 13–15].

One major motivation for solving a DisCSP without gathering all information in one server is the concern about privacy/security, i.e., the knowledge of the problem each agent has is private information and revealing such information

to a server or other agents is not desirable. Consequently, we cannot gather all information in a single server and solve the problem by using centralized CSP techniques. In a DisCSP, a variable value can be considered as an action/plan that an agent will take. It is natural that an agent does not want to reveal information on possible plans or the final plan it will take to other agents.

For example, a problem of scheduling multiple meetings among multiple participants can be formalized as a DisCSP as follows. Each agent/participant has one variable that corresponds to each meeting. The domain of a variable is possible dates and time slots. There exist equality constraints among variables that represent the same meeting and belong to different agents (i.e., they must meet the same day/time). Also, there exist inequality constraints between multiple variables that belong to the same agent (i.e., a person cannot attend multiple meetings at the same time). Also, an agent has unary constraints on its variables (i.e., he/she has personal schedules that prevent him/her from attending a meeting). In this problem domain, it is clear that a person would not be happy to make such private information public.

However, existing DisCSP algorithms leak some information during the search process and privacy/security issues have not yet been dealt with formally. For example, in the asynchronous backtracking algorithm [14], each agent exchanges a tentative value assignment with each other. If the current assignment does not satisfy constraints, these agents change their assignments and perform backtracking in a certain order. During this search process, an agent can obtain some information on possible values of variables that belong to other agents. Also, an agent can learn the final value assignment of these variables.

When applying this algorithm to the meeting scheduling problem, we can assume each agent makes proposals on the date and time slot of the meeting and negotiates with other agents. The fact that an agent proposes a certain date reveals that he/she does not have any personal schedule on that date. If an agent declines a certain date, this means that he/she has a personal schedule or the date conflicts with some other meeting. Such private information is leaked during the search process.

On the other hand, in the research community on information security and cryptography, there have been many works on multi-party protocols, which deal with performing various computations based on private information of participants, while keeping the private information secret [4, 9]. However, as far as the authors are aware, there has been virtually no work on solving combinatorial optimization problems (including CSPs as a special case) by utilizing information security techniques, with the notable exception of the authors' recent works on secure dynamic programming [12, 16].

In this paper, we develop a secure DisCSP algorithm that utilizes information security techniques. As far as the authors are aware, this is the first research effort that combines the two growing research fields, i.e.. constraint satisfaction and information security.

In this paper, we say that an algorithm does not leak any private information if an agent cannot obtain any additional information on the value assignment of

variables that belong to other agents. In a meeting scheduling application, this means that each participant cannot know the scheduled dates of the meetings he/she will not attend. Also, he/she cannot obtain any information on the private schedules of other participants.

In our newly developed secure DisCSP algorithm, multiple computational servers are used to implement a standard chronological backtracking; thus, this algorithm is guaranteed to be complete. Each agent only knows the value assignment of its own variables and cannot obtain any additional information on the value assignment of variables that belong to other agents. Also, computational servers cannot get any information on the value assignment of any variables.

In the rest of this paper, we first show the formal definition of DisCSP and secure algorithms (Section 2). Next, we describe a public key encryption scheme, which is a basic tool used in our secure DisCSP algorithm (Section 3). Then, we describe the details of our newly developed secure DisCSP algorithm (Section 4). Finally, we examine the characteristics of this algorithm (Section 5) and discuss related works (Section 6).

2 Formalization

A DisCSP can be formalized as follows.

- There exist agents $1, 2, \ldots, I$.
- There exist variables $x_1, x_2, \ldots, x_n$. Each variable belongs to one agent. The fact that x_i belongs to agent a is represented as $\mathit{belongs}(x_i, a)$.
- All variables have a common domain $\{1, 2, \ldots, m\}$. This domain of variables is common knowledge among agents.
- There exist unary constraints on one variable and binary constraints between two variables.
- We assume constraints are represented as *nogoods*. A unary nogood $(x_i = d_i)$ represents the fact that the value assignment d_i to variable x_i violates the constraint. A binary nogood $(x_i = d_i, x_j = d_j)$ represents the fact that the assignment of $x_i = d_i$ and $x_j = d_j$ violates the constraint.
- These constraints are the private information of agents. More specifically, the unary constraints on x_i, which belongs to agent a, are known only by agent a. Let us denote a set of these unary constraints as $C^a_{x_i}$. Also, the binary constraints between x_i and x_j, which belong to agent a and agent b, respectively, are distributed between agents a and b. Let us denote the constraints agent a knows as $C^a_{x_i,x_j}$ and the constraints agent b knows as $C^b_{x_i,x_j}$.
- A solution of DisCSP $D = (x_1 = d_1, x_2 = d_2, \ldots, x_n = d_n)$ is a value assignment of all variables that satisfies all unary and binary constraints.

Let us show an example. Figure 1 shows an example of the well-known n-queens problem, where $n = 4$. If we assume there exists an agent that corresponds to a queen of each row and these queens try to find their positions so that they do not kill each other, this problem can be formalized as a DisCSP. We call this problem the distributed 4-queens problem. More specifically, there are four

agents $1, 2, 3, 4$. Each agent i has one variable x_i with domain $\{1, 2, 3, 4\}$. In this problem, there is no unary constraint. Let us assume that for $i < j$, $C^i_{x_i,x_j}$, i.e., the constraint agent i knows, consists of diagonal constraints, e.g., $C^2_{x_2,x_3}$ contains nogood $(x_2 = 1, x_3 = 2)$, etc., and $C^j_{x_i,x_j}$ consists of column constraints, e.g., $C^2_{x_1,x_2}$ contains nogood $(x_1 = 1, x_2 = 1)$, etc.

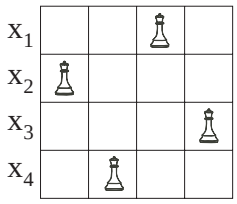

Fig. 1. Distributed 4-Queens Problem

This formalization is based on the traditional definition introduced by [13, 14]. One minor difference is that in [13, 14], it is assumed that the domain of a variable is different and the domain is private information, while in our formalization, the entire domain is common knowledge but an agent can have private unary constraints. These two formalizations are mutually interchangeable and there are no fundamental differences. Another slight difference is that in [13, 14], binary constraints between variables x_i and x_j are assumed to be the common knowledge of agents a and b, who own these variables. This is a special case of our definition where $C^a_{x_i,x_j}$ and $C^b_{x_i,x_j}$ are identical.

In [10], an alternative formalization of DisCSP is presented in which variables and domains are common knowledge and constraints are private information. Our formalization is based on [14], i.e., the constraints an agent has are restricted to the constraints that are related to its own variables.

We define the fact that when solving a DisCSP using a particular algorithm, the algorithm does not leak private information of an agent to other agents as follows.

– After a solution is found, each agent has the same knowledge on the assignment of other agents as the case where the agent obtains (a part of) the solution directly from an oracle without performing the algorithm.

  More specifically, let us denote an assignment of any subset of k variables as $D_k = (x_{j_1} = d_{j_1}, x_{j_2} = d_{j_2}, \ldots, x_{j_k} = d_{j_k})$. Also, let us define $p_a(D_k)$ as the estimated probability of agent a, in that the final solution is a superset of D_k, after a solution is found by performing the algorithm. Also, let us define $p^{\text{oracle}}_a(D_k)$ as the estimated probability of agent a, in that the final solution is a superset of D_k, after obtaining the value assignment of its own variables from the oracle. This definition means $p_a(D_k) = p^{\text{oracle}}_a(D_k)$ for all D_k.

When obtaining the value assignment of its own variables from the oracle, the agent obtains certain information on the value assignment of other variables. For example, assume x_1 belongs to agent a and x_2 belongs to agent b. If agent a knows there exists an inequality constraint between x_1 and x_2, agent a can infer that $p_a((x_j = d_i)) = 0$ if x_i is assigned to d_i. Also, if there exists an equality constraint, it can infer that $p_a((x_j = d_i)) = 1$. The above condition means that the algorithm does not leak any unnecessary information, i.e., no additional information is leaked besides the information that can be inferred by obtaining a part of the solution.

Basically, the above definition requires that there must be no information leak on unary constraints. It is clear that if a solution is obtained from the oracle, an agent can obtain no information on the unary constraints of other agents. If agent a learns some information on unary constraints, e.g., a learns nogood $(x_j = d_j)$ by performing an algorithm, $p_a((x_j = d_j)) = 0$ cannot be equal to $p_a^{\text{oracle}}((x_j = d_j))$, except when $p_a^{\text{oracle}}((x_j = d_j)) = 0$.

Furthermore, if an algorithm requires a third party other than the agents who originally have variables (we call such an actor a *server*), we define the fact that when solving a DisCSP using a particular algorithm, the algorithm does not leak any private information of an agent to a server as follows. We assume a server does not have any a priori knowledge of the problem.

- A server has no knowledge on the obtained solution after performing the algorithm. More specifically, for an assignment of any subset of k variables $D_k = (x_{j_1} = d_{j_1}, x_{j_2} = d_{j_2}, \ldots, x_{j_k} = d_{j_k})$, let us denote the estimated probability of a server, in that the obtained solution is a superset of D_k, as $p_{\text{server}}(D_k)$. This definition means $p_{\text{server}}(D_k) = 1/m^k$, where k is the number of variables in D_k. This condition means all assignments look equally probable for a server.

This condition requires that a server cannot learn any information on unary/binary constraints.

3 Preliminaries

In this section, we describe a basic tool for our implementation, i.e., an *indistinguishable*, *homomorphic*, and *randomizable* public key encryption scheme. In the rest of this paper, we use ElGamal encryption [2], which has all of these properties, for describing our algorithm. However, our algorithm can be implemented using other encryption methods that also have these properties.

- Public key encryption: In public key encryption, the key used for encryption is public, so anybody can create ciphertext $E(M)$ from plaintext M. On the other hand, the key used for decryption is kept secret and only the one who has the secret key can obtain M from $E(M)$.
- ElGamal encryption: ElGamal encryption is one instance of public key encryption. Let $q, p = 2q+1$ be primes and $G =< g >\subset \mathbf{Z}_p^*$ be a cyclic group of

order q generated by g, where $\mathbf{Z}_p$ denotes a set of integers from 0 to $p-1$ and $\mathbf{Z}_p^*$ denotes a set of integers that are in $\mathbf{Z}_p$ and prime to p. The secret key is $s \in \mathbf{Z}_q$ and the corresponding public key is $g, y = g^s$. ElGamal encryption is based on the assumption of the hardness of the discrete logarithm problem (DLP), i.e., to find s from (g, g^s) is computationally infeasible. Please note that we use modulo p arithmetic.

Anyone can encrypt message $M \in G$ just by using the public key $g, y = g^s$, i.e., choose random number $r \in \mathbf{Z}_q$ and create ElGamal ciphertext $E(M) = (A = g^r, B = y^r M)$.

One who knows the secret key, $s \in \mathbf{Z}_q$, can decrypt ciphertext $E(M) = (A = g^r, B = y^r M)$, i.e., compute $B/A^s = M$.

- Indistinguishable encryption: In ElGamal encryption, $E(M)$ is created using random number r. Thus, if the same plaintext is encrypted twice using different random numbers, these two ciphertexts look totally different and we cannot know whether the original plaintexts are the same or not without decrypting them.
- Homomorphic encryption: Encryption E is homomorphic if $E(M_1)E(M_2) = E(M_1 M_2)$ holds. If we define the product of ciphertexts $E(M_1) = (A_1, B_1)$ and $E(M_2) = (A_2, B_2)$ by $E(M_1)E(M_2) = (A_1 A_2, B_1 B_2)$, ElGamal encryption E is homomorphic encryption. By using this property, we can obtain the product of two plaintexts by taking the product of two ciphertexts without decrypting them.
- Randomization: In ElGamal encryption, one can create a new randomized ciphertext $E(M) = (Ag^{r'}, By^{r'})$ with random value r' from the original ciphertext $E(M) = (A = g^r, B = y^r M)$. This is equivalent to making a product of $E(1) = (g^{r'}, y^{r'})$ and $E(M)$. If we assume that the Decision Diffie-Hellman (DDH) problem is infeasible, one cannot determine whether a ciphertext is a randomized ciphertext of the original ciphertext or not.
- Multiple Servers: By utilizing secret sharing techniques, we can make each server to have only a share of the secret key; thus, any collusion of t (or less than t) servers cannot decrypt E [9].

  In the preparation phase, secret key s and public key y are generated in a distributed way [8] and each distributed server has only a share of the secret key. The decryption is performed in a distributed fashion by each distributed server that has a share of the secret key.

  For example, let us consider a simplest case where the total number of servers is two and $t = 1$, i.e., there are servers 1 and 2. If these two servers cooperate, they can decrypt E while a single server cannot. Servers 1 and 2 generate their own shares of the secret keys s_1, s_2, respectively. The secret key is $s = s_1 + s_2$, but each server does not know s. They exchange $y_1 = g^{s_1}$ and $y_2 = g^{s_2}$ with each other and obtain public key $y = y_1 \cdot y_2 = g^{s_1+s_2} = g^s$.

  When decrypting $E(M) = (A = g^r, B = y^r M)$, these servers calculate A^{s_1} and A^{s_2} and exchange the results with each other. Then, by calculating $B/(A^{s_1} \cdot A^{s_2}) = B/A^{s_1+s_2} = B/A^s$, these servers can obtain plaintext M. Note that the secret key s is kept secret to these servers even after the decryption; they need to cooperate to decrypt another ciphertext.

4 Secure DisCSP Algorithm

In this section, we show the details of our newly developed algorithm. When describing our algorithm, we put the following assumptions for notation simplicity.

- Each agent i has exactly one variable x_i.
- There exist binary constraints between all pairs of variables.

Relaxing these assumptions and extending the algorithm to general cases is rather straightforward.

4.1 Basic Ideas

In our newly developed secure DisCSP algorithm, computational servers called a search-controller, decryptors, and value-selectors are used. There exist multiple (at least two) decryptors. Each of these decryptors has a share of the secret key of E as described in Section 3, and public key y for E is generated by these decryptors distributedly. There exists one value-selector for each variable/agent. Also, there exists one search-controller.

The main search procedure is performed by the search-controller and value-selectors. Each agent first encodes unary/binary constraints and passes the information to the servers. Then, these servers obtain a solution and return the value assignment to each agent. By utilizing these servers, we can guarantee that the information each agent can get is the same as the case where the agent obtains its value assignment directly from an oracle.

On the other hand, when utilizing servers, we must make sure that these servers do not obtain any information of the obtained solution. The definition introduced in Section 2 is so strict that it requires that a server cannot learn the fact that two variable values are different/equal.

In our secure DisCSP algorithm, this requirement is satisfied by introducing the following methods: 1) constraints are encoded by using a public key encryption scheme, 2) agents cooperatively perform renaming/permutation of variable values, 3) the search procedures are distributed among multiple servers, i.e., a search-controller, decryptors, and value-selectors.

Figure 2 (a) shows the flow of the proposed algorithm. In the rest of this section, we describe the details of these procedures.

4.2 Encoding Constraints

Each agent needs to encode its unary/binary constraints so that subsequent renaming/permutation is possible and so that the search-controller and value-selectors cannot understand but decryptors can cooperatively decrypt.

To satisfy this goal, for each value k of variable x_i, agent i represents its unary/binary constraints using an $n \times m$ constraint matrix. We denote this matrix as $A_{i,k}$. The element of this matrix $A_{i,k}(j,l)$ is defined as follows. $E(1)$ and $E(z)$ denote the encryption of 1 and common public element $z(\neq 1)$, respectively. z is chosen so that $z^c \bmod p \neq 1$ holds for all c, where $0 < c < q$. We also assume $2(n-1) < q$ holds.

- For $j \neq i$:
  - $A_{i,k}(j,l) = E(z)$ if $x_i = k$ and $x_j = l$ are inconsistent, i.e., nogood($x_i = k, x_j = l$) is in $C^i_{x_i,x_j}$, or k violates i's unary constraint, i.e., nogood($x_i = k$) is in $C^i_{x_i}$.
  - $A_{i,k}(j,l) = E(1)$ if $x_i = k$ and $x_j = l$ are consistent.
- For $j = i$ and $l \neq k$: $A_{i,k}(j,l) = E(1)$.
- For $j = i$ and $l = k$: $A_{i,k}(j,l) = E(E_i(k))$, where E_i is i's encryption function. E_i can be any encryption scheme, i.e., it does not need to be a public key encryption scheme. We assume $E_i(k) \in G$ and $E_i(k) \neq 1$.

Figure 2 (b) shows a constraint matrix for value 1 of x_2 encoded by agent 2 in the distributed 4-queens problem. Note that agent 2 knows column constraints with x_1 and diagonal constraints with x_3 and x_4. The j-th row for $j \neq i$ represents binary constraints, i.e., the position that is in conflict with $x_2 = 1$ is filled by $E(z)$ (otherwise filled by $E(1)$). As described in Section 3, E is indistinguishable, which means that each $E(z)$ (or $E(1)$) looks totally different and we cannot know whether the original plaintexts are the same or not without decrypting them.

The i-th row encodes the information that this matrix is on value k. This information is used to obtain the actual value assignment from the solution on renamed/permuted variable values.

4.3 Renaming/Permutation

Next, agents perform the permutation of columns on each constraint matrix. This means that agents transform the original problem into a new problem in which variable values are renamed. More specifically, for each variable value k, k is renamed as $\pi(k)$, where π is a permutation of m elements, i.e., a bijective map from $\{1, 2, \ldots, m\}$ to $\{1, 2, \ldots, m\}$. Note that the domain of all variables is common and agents perform the same permutation for all variables.

To make sure that no agent can know the result of the permutation, each agent sequentially performs the permutation one by one. As a result, no agent knows the result of the total permutation. By utilizing randomization, we cannot know the result of the permutation even if we compare the matrices before and after the permutation.

More specifically, agent j has its own permutation function $\pi_j(\cdot)$. The combined permutation function, i.e., $\pi(\cdot)$ is given by $\pi_n(\pi_{n-1}(\ldots \pi_1(\cdot)\ldots))$.

The detailed procedure is as follows.

- For each k, each agent i makes public $(i, A_{i,k})$.
- From 1 to n, each agent j sequentially applies permutation function π_j to these matrices. More specifically, agent 1 first applies the permutation function $\pi_1(\cdot)$ to columns for all $A_{i,k}$ and makes public the result, where each element of the matrix is randomized by multiplying $E(1)$. Next, agent 2 applies the permutation function $\pi_2(\cdot)$, and so on. Let us denote the final result as $(i, A'_{i,k})$.

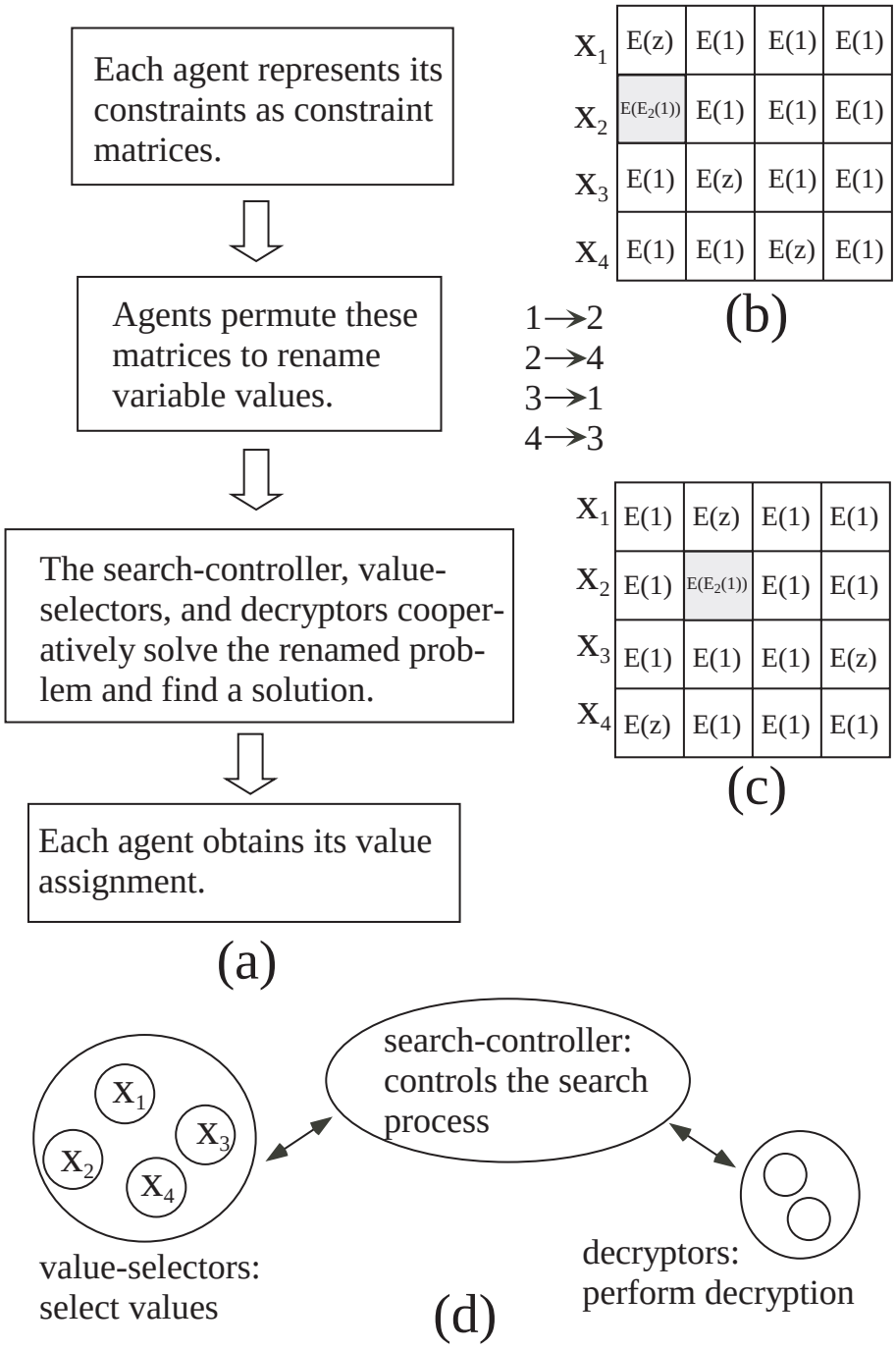

Fig. 2. Overview of Secure DisCSP Algorithm

– Decryptors cooperatively decrypt the i-th row of $A'_{i,k}$. If the k'-th element is not equal to 1, i.e., it is equal to $E_i(k)$, then decryptors send $(k', E_i(k), A'_{i,k})$ to value-selector i. Note that $k' = \pi(k)$.

Figure 2 (c) shows the result of the permutation $1 \rightarrow 2$, $2 \rightarrow 4$, $3 \rightarrow 1$, $4 \rightarrow 3$ for the constraint matrix described in Figure 2 (b).

By using the above procedure, each value-selector i gets the constraint matrices associated with x_i, where variable values are renamed/permuted by $k \rightarrow k'$, where $k' = \pi(k)$.

4.4 Search Procedure

In this section, we show the search procedure performed by the search-controller, decryptors, and value-selectors. The search-controller controls the search process. For each variable x_i, there exists value selector i. Value-selector i selects a renamed variable value for x_i. The decryptors perform decryption (Figure 2 (d)).

Partial_Solution $\leftarrow \{\}$;
send **start-round** messages to all value-selectors;
wait until receiving all $v(i, j)$;
SELECT-VARIABLE: select variable x_i which is not in *Partial_Solution*;
if all variables are already included in *Partial_Solution*
then inform value-selectors that the current assignment is a solution;
  terminate the procedure;
else *pos* $\leftarrow i$;
  CHECK: $P \leftarrow \prod_j v(pos, j) \cdot v(j, pos)$, where $x_j \in$ *Partial_Solution*;
  send P to decryptors and wait for the decryption of P.
  if the decryption of P is 1
  then add x_{pos} to *Partial_Solution*, goto **SELECT-VARIABLE**;
  else
    ASK-NEXT-VALUE: send **set-next-value** message
      to value-selector *pos* and wait for its reply;
    when received **ok** message **do**
      send **start-round** messages to all value-selectors;
      wait until receiving all $v(i, j)$; goto **CHECK**; **end do**;
    when received **backtrack** message **do**
      if *Partial_Solution* is empty
      then announce that there exists no solution;
        terminate the procedure;
      else remove variable $x_{i'}$ that was added to to *Partial_Solution*
          most recently from *Partial_Solution*;
        pos $\leftarrow i'$; goto **ASK-NEXT-VALUE**;
      end if; **end do**; **end if**; **end if**;

Fig. 3. Procedure for Search-Controller

Procedure for Search-Controller

Figure 3 shows the procedure for the search-controller. The search-controller performs the described search procedure that is equivalent to a standard chronological backtracking. *Partial_Solution* represents a set of variables that are in a partial solution. All constraints among variables within the partial solution are satisfied. If a variable that is not in the partial solution does not satisfy a constraint with a variable in the partial solution, the search-controller asks the value-selector of the variable to change its current assignment. If the value-selector has no other value, then the value of the variable that was most recently added to the partial solution is changed (backtracking).

In the procedure described in Figure 3, if all constraints are satisfied, then P is $E(1)$, otherwise, P is $E(z^c)$, where c is the number of violated constraints, since E is homomorphic. Also, since z is chosen so that $z^c \bmod p \neq 1$ for all $0 < c < q$ and $2(n-1) < q$, z^c will not be equal to 1.

Procedure for Value-selector

Figure 4 shows the procedure for a value-selector i. Only this value-selector knows the current assignment of variable x_i. To check the constraints, when value-selector i chooses d_i, it sends a randomized j-th row of $A'_{i,k}$ in $(d_i, E_i(k), A'_{i,k})$ to value-selector j. For example, if value-selector 2 chooses 2, it sends the first row of the matrix of Figure 2 (c), i.e., $[E(1), E(z), E(1), E(1)]$ to value-selector 1. If value-selector j chooses d_j for its variable x_j, it chooses the d_j-th element of the communicated vector. For example, if value-selector 1 chooses 2, it chooses the second element of $[E(1), E(z), E(1), E(1)]$, i.e., $E(z)$. If the decryption of this element (which is performed by decryptors) is 1, this means the current assignment of x_i and x_j satisfies the unary and binary constraints of agent i. In this case, since the decryption is z, the current assignment of x_1 and x_2 does not satisfy the constraints of agent 2.

Procedure for Decryptors

Each decryptor j has a share of secret key s, i.e., s_j, where $s = \sum_j s_j$. The number of decryptors depends on the required level of security, i.e., by using $t+1$ decryptors, even if t (or less than t) decryptors collude to obtain the private information of agents, they cannot decrypt constraint matrices directly.

If decryptors are asked to decrypt $E(M) = (A, B)$, each decryptor j calculates A^{s_j} and exchanges the results. By calculating $B/\prod_j A^{s_j} = B/A^s$, decryptors can obtain M.

Obtaining a Solution

One instance of a solution obtained under the permutation $1 \rightarrow 2$, $2 \rightarrow 4$, $3 \rightarrow 1$, $4 \rightarrow 3$ is $d_1 = 1, d_2 = 2, d_3 = 3, d_4 = 4$.

When each value-selector is informed by the search-controller that the current assignment is a solution, it sends $E_i(k)$ in $(d_i, E_i(k), A'_{i,k})$, where d_i is the current assignment, to agent i, i.e., $E_1(3)$ to agent 1, $E_2(1)$ to 2, $E_3(4)$ to 3, and $E_4(2)$

when initialized **do**
  $Done \leftarrow \{\}$, $d_i \leftarrow$ a randomly chosen value from $1, \ldots, m$; **end do**;

when received **start-round** message **do**
  send the following message for all value-selectors $j \neq i$
    $V(i,j)$, where $V(i,j)$ is a randomized j-th row of $A'_{i,k}$ in $(d_i, E_i(k), A'_{i,k})$,
      i.e., $[A'_{i,k}(j,1), \ldots, A'_{i,k}(j,m)]$; **end do**;

when received $V(j,i)$ from value-selector j **do**
  send $v(j,i)$, which is a randomized d_i-th element of $V(j,i)$,
    to the search-controller; **end do**;

when received **set-next-value** message **do**
  add d_i to $Done$,
  if all values are already included in $Done$,
  then set $Done \leftarrow \{\}$;
    randomly choose new value d_i,
    send **backtrack** message to the search-controller;
  else randomly choose new value d_i that is not in $Done$;
    send **ok** message to the search-controller; **end do**;

Fig. 4. Procedure for Value-Selector i

to 4. By decrypting $E_i(k)$, agent i obtains the value k for its variable x_i, which is a part of the final solution.

5 Algorithm Characteristics

5.1 Security

It is clear that this algorithm does not leak any additional information to other agents. Since constraint matrices are encrypted, an agent cannot get any information during the permutation phase. An agent does not participate in the search phase. Therefore, the information an agent can get is the same as the case where the agent obtains a part of the final solution from an oracle.

Next, we show that the algorithm does not leak any information to servers, i.e., the search-controller, decryptors, and value-selectors. Since variable values are renamed by permutation, each value-selector cannot know the actual real value it is selecting. Also, the renamed assigned value is known only to one value-selector. Therefore, neither a value-selector, the search-controller, nor an decryptor is able to get any information on whether the values of two variables are the same/different. Also, unless all decryptors collude, decryptors cannot decrypt a constraint matrix directly.

Although the search-controller observes the search process, this information is useless for updating the estimated probability that a particular assignment is a

part of the final solution. For example, let us assume the simplest case, i.e., there exist two variables x_1, x_2, whose domain is $\{1, 2\}$. If the search-controller adds x_1 to *Partial_Solution* first, when a solution is found, there are four possible scenarios, i.e., 1) a solution is obtained immediately, 2) a solution is found after changing x_2's value once, 3) a solution is obtained after changing x_2 twice and performing backtracking, 4) a solution is obtained after changing x_2's value twice and performing backtracking, then changing x_2's value again. Regardless of which pattern is observed, each of the four possible solutions, i.e., $(x_1 = 1, x_2 = 1)$, $(x_1 = 1, x_2 = 2)$, $(x_1 = 2, x_2 = 1)$, and $(x_1 = 2, x_2 = 2)$, is equally probable.

Also, a value-selector observes only partial information of the search process. Even if the value-selector can get the same information as decryptors, it still cannot obtain any additional information on a particular assignment being a part of the final solution.

If we do not perform renaming/permutation, value-selector i can learn the final value assignment of variable x_i. Also, if we do not distribute processing among the search-controller, value-selectors, and decryptors, i.e., solve a problem using a centralized server, although this server cannot know the actual value assignments because of renaming/permutation, the server can tell whether two variable values are the same/different.

5.2 Communication Costs

The search procedure of the secure DisCSP algorithm is equivalent to a standard chronological backtracking. The number of rounds, i.e., the number of times that the search-controller, value-selectors, and decryptors exchange messages, becomes m^n in the worst case.

For each round and for each constraint $C^i_{x_i,x_j}$, value-selector i communicates an m-element vector to value-selector j, and value-selector j communicates one element of the vector to the search-controller. Also, decryptors must communicate with each other to decrypt a ciphertext.

The required communication costs are clearly much larger than existing DisCSP algorithms [5, 7, 10, 13–15]. However, this seems somewhat inevitable if we wish to preserve the privacy of agents.

6 Discussion

In most of the existing DisCSP algorithms, each agent exchanges the tentative value assignment with other agents, and final value assignments are made public [5, 13–15]. In [10], an alternative formalization of DisCSP is presented in which variables and domains are common knowledge and constraints are private information. In the algorithm presented in [10], agents obtain a solution by explicitly communicating information on constraints to each other.

In [7], an algorithm called distributed forward-checking algorithm is presented in which an agent communicates possible domains of variables that belong to other agents rather than its own assignments. This is similar to our

idea of communicating a row in a constraint matrix. However, in [7], encryption techniques are not used so private information is leaked during the search process.

One promising application field of secure DisCSP algorithms is meeting scheduling. In [3], the trade-off between the efficiency of an algorithm and privacy of agents in meeting scheduling problems is discussed. In [6], a secure meeting scheduling protocol that utilizes information security techniques is developed. However, this protocol is specialized for a meeting scheduling problem in which only a single meeting is scheduled. By applying our newly developed algorithm, multiple meetings can be scheduled simultaneously.

It is well known that any combinatorial circuit can be computed securely by using general-purpose multi-party protocols [1, 4]. Therefore, if we can construct a combinatorial circuit that implements a constraint satisfaction algorithm, in principle, such an algorithm can be executed securely (thus we do not need to develop a specialized secure protocol for DisCSPs). However, implementing a combinatorial circuit that executes a constraint satisfaction algorithm is not easy, and the obtained circuit would be very large. Note that we need to create a general purpose logic circuit to solve CSPs, not specialized hardware to solve a particular problem instance (such as those discussed in [11]).

Furthermore, to execute such a general-purpose multi-party protocol, for each computation of an AND gate in the circuit, the servers must communicate with each other. Using such a general purpose multi-party protocol for a distributed constraint satisfaction problem is not practical at all due to the required communication costs.

7 Conclusions

In this paper, we developed a secure DisCSP algorithm. Our newly developed algorithm utilizes an indistinguishable, homomorphic, and randomizable public key encryption scheme. In this algorithm, multiple servers, which receive encrypted information from agents, cooperatively perform a search process and obtain an encrypted solution. Then, a part of the encrypted solution is sent to each agent. By using this algorithm, the private information of an agent is not leaked during the search process to other agents or servers.

In this paper, we developed an algorithm whose performance is equivalent to a basic chronological backtracking as a first step in developing secure DisCSP algorithms. Our future works include 1) analyzing the robustness of the developed algorithm against collusions of servers and agents, 2) developing new algorithms that are more computationally efficient and require less communication costs without sacrificing security/privacy.

References

1. Ben-Or, M., Goldwasser, S., and Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation, *Proceedings of 20th ACM Symposium on the Theory of Computing* (1988) 1–10

2. ElGamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Transactions on Information Theory*, Vol. IT-31, No. 4, (1985) 469–472
3. Freuder, E. C., Minca, M., and Wallace, R. J.: Privacy/Efficiency Tradeoffs in Distributed Meeting Scheduling by Constraint-based Agents, *Proceedings of IJCAI-01 Workshop on Distributed Constraint Reasoning* (2001)
4. Goldreich, O., Micli, S., and Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority, *Proceedings of 19th ACM Symposium on the Theory of Computing* (1987) 218–229
5. Hamadi, Y., Bessière, C., and Quinqueton, J.: Backtracking in Distributed Constraint Networks, *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI-98)* (1998) 219–223
6. Herlea, T., Claessens, J., Neven, G., Piessens, F., Preneel, B., and De Decker, B.: On Securely Scheduling a Meeting, Dupuy, M. and Paradinas, P. eds., *Trusted Information - The New Decade Challenge, Proceedings of IFIP SEC* (2001) 183–198
7. Mesequer, P. and Jiménez, M. A.: Distributed Forward Checking, *Proceedings of CP-00 Workshop on Distributed Constraint Satisfaction* (2000)
8. Pedersen, T.: A Threshold Cryptosystem without a Trusted Party, *Proceedings of EUROCRYPT '91* (1991) 522–526, Lecture Notes in Computer Science 547
9. Shamir, A.: How to share a secret, *Communications of the ACM*, Vol. 22, No. 11, (1979) 612–613
10. Silaghi, M.-C., Sam-Haroud, D., and Faltings, B. V.: Asynchronous Search with Aggregations, *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)* (2000) 917–922.
11. Suyama, T., Yokoo, M., Sawada, H., and Nagoya, A.: Solving Satisfiability Problems using Reconfigurable Computing, *IEEE Transactions on VLSI*, Vol. 9, No. 1, (2001) 109–116
12. Suzuki, K. and Yokoo, M.: Secure Combinatorial Auctions by Dynamic Programming with Polynomial Secret Sharing, *Proceedings of Sixth International Financial Cryptography Conference (FC-02)* (2002)
13. Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving, *Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems* (1992) 614–621
14. Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K.: The Distributed constraint satisfaction problem: formalization and algorithms, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 5, (1998) 673–685
15. Yokoo, M. and Hirayama, K.: Distributed Breakout Algorithm for Solving Distributed Constraint Satisfaction Problems, *Proceedings of the Second International Conference on Multi-Agent Systems*, MIT Press (1996) 401–408
16. Yokoo, M. and Suzuki, K.: Secure Multi-agent Dynamic Programming based on Homomorphic Encryption and its Application to Combinatorial Auctions, *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002)* (2002): (to appear)