セキュアマルチエージェント動的計画法の設計と組合せオークションへの適用

鈴木 幸太郎, 横尾 真

This paper presents a secure dynamic programming protocol that utilizes homomorphic encryption. By using this method, multiple agents can solve a combinatorial optimization problem among them without leaking their private information to other agents. More specifically, in this method, multiple servers cooperatively perform dynamic programming procedures for solving a combinatorial optimization problem by using the private information sent from agents as inputs. Although the severs can compute the optimal solution correctly, the inputs are kept secret even from the servers. Furthermore, we discuss the application of this protocol to various types of combinatorial auctions, i.e., multi-unit auctions, linear-good auctions, and general combinatorial auctions.

1 Introduction

In multi-agent systems, multiple autonomous agents sometimes need to solve a combinatorial optimization problem by using their private information. For example, in a combinatorial auction where multiple goods are auctioned simultaneously, agents need to find a combination of bids for disjoint sets of goods, so that the sum of the bidding prices is maximized. This problem is called the winner determination problem and has recently become a very active research field [5][13][14][15].

If there exists a fully trusted agent, e.g., the participants can trust the auctioneer, it is possible to gather all private information relevant to the combinatorial optimization problem at this trusted agent; thus this agent can solve the problem using any available centralized optimization technique.

However, we cannot take it for granted that there exists such a trusted agent. For example, in a standard first-price sealed-bid auction [12], where the highest bidder wins and pays his/her own price, the auctioneer might collude with a particular participant and reveal information about incoming bids to that participant during the auction.

If we use a strategy-proof mechanism [12][24], such as a second-price sealed bid (Vickrey) auction, where the highest bidder wins and pays the second highest price, the information of the other participants' bids becomes useless; thus we can discourage collusion between the auctioneer and bidders. However, in a second-price sealed-bid auction, the auctioneer can increase his/her revenue by fabricating a fake bid whose price is very close to the highest bid.

Secure Multi-agent Dynamic Programming:Design and Application to Combinatorial Auctions

Koutarou Suzuki, 日本電信電話株式会社, NTT 情報流通プラットフォーム研究所, NTT Information Sharing Platform Laboratories, koutarou@isl.ntt.co.jp

Makoto Yokoo, 日本電信電話株式会社, NTT コミュニケーション科学基礎研究所, NTT Communication Science Laboratories, yokoo@cslab.kecl.ntt.co.jp

We can utilize various cryptographic technologies so that while accepting incoming bids, the auctioneer cannot learn bidding prices. For example, bidders first submit encrypted bids, and then give the auctioneer the decryption keys after the bids are closed. However, the auctioneer can utilize the information of bids for future auctions. For example, the auctioneer can learn the behavior/preference of a certain participant from the past auctions and commit similar fraud based on this information, or the auctioneer might reveal/sell such private information to others. Varian [22] described this problem as follows: *"Even if current information can be safeguarded, record of past behavior can be extremely valuable, since historical data can be used to estimate the willingness to pay. What should be the appropriated technological and social safeguards to deal with this problem?"*.

This paper aims to provide a solution to this problem by utilizing indistinguishable, homomorphic public key encryption scheme [3]. More specifically, multiple servers cooperatively solve a combinatorial optimization problem by using the private information sent from agents as inputs. Although the severs can compute the optimal solution correctly, the inputs are kept secret even from the servers. For example, in a combinatorial auction, multiple auction servers can solve the winner determination problem, i.e., they can find the combination of bids so that the sum of the bidding prices is maximized. However, the information of bids that are not part of the optimal solution is kept secret from the auction servers. More specifically, we develop a method for securely executing dynamic programming procedures [1], which are very effective and widely applied to various combinatorial optimization problems.

The application of the proposed secure dynamic programming protocol is not limited to combinatorial auctions. In various applications, it is quite possible that multiple agents are willing to solve a combinatorial optimization problems cooperatively, but they are still concerned with privacy. For example, if a multiple network carriers cooperatively provide a particular service, they might need to find the most cost effective way for providing the service by combining their resources. However, it is quite natural that these carriers are not willing to reveal all their private information to each other. By using our secure dynamic programming protocol, these carriers can find the optimal solution without revealing their private information.

The rest of this paper is organized as follows. In Section 2, we briefly describe the overview of dynamic programming techniques. In Section 3, we detail our newly developed secure dynamic programming protocol. Furthermore, in Section 4, we describe a way of applying the proposed method to various types of combinatorial auctions, i.e., multi-unit auctions, linear auctions, and general combinatorial auctions. In Section 5, we discuss its relation to existing techniques.

2 Dynamic Programming

Dynamic programming [1] was developed by R. Bellman during the late 1950's. Dynamic programming is a powerful method that can be applied to various combinatorial optimization problems.

In the following, we use the problem of finding the longest path in the one-dimensional directed graph described in Figure 1 to illustrate the concept of dynamic programming.

This graph consists of nodes $0, 1, 2, \ldots, m$ with directed links among them. A link is represented as (j, k), where $j < k$. For each link (j, k), the weight of the link $w(j, k)$ is defined. The goal is to find the longest path from initial node 0 to terminal node m, i.e., to find a path from 0 to m so that the sum of the weights of links are maximized. For simplicity, we assume for each node j (where $0 \leq j < m$),

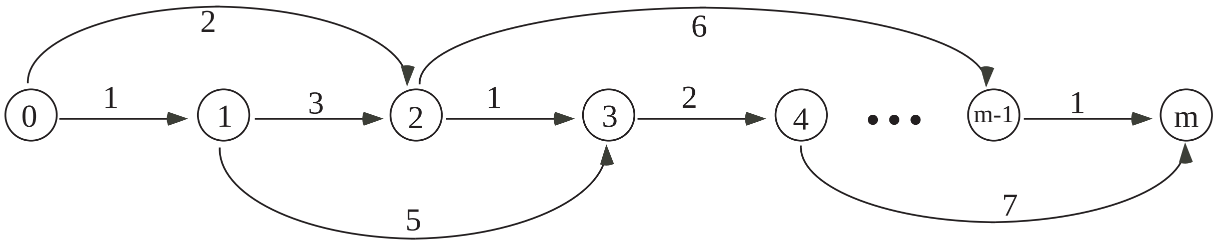

図 1 Example of one-dimensional directed graph (I)

there exists at least one link that starts from j, i.e., there is no dead-end node except m.

One notable characteristic of this problem is as follows. Assume P is the longest path from 0 to m. Then, for any node j which is on P, the last half of P, i.e., the part of P from j to m, is also a longest path from j to m. This characteristic is called the *principle of optimality*. This feature enables us to find the optimal solution of the original problem from the optimal solutions of sub-problems.

More specifically, we can obtain the length of the longest path from 0 to m by solving the following recurrence formula from node $m-1$ to 0. In this formula, $f(j)$ represents the length of the longest path from j to m. We call $f(j)$ an *evaluation value* of node j. For terminal node m, $f(m)$ is defined as 0. For initial node 0, $f(0)$ represents the optimal solution, i.e., the length of the longest path from 0 to m.

$$f(j) = \max_{(j,k)}\{w(j,k) + f(k)\}$$

When calculating this formula, for each node j, we record the link (j,k) that gives the evaluation value $f(j)$, i.e., the link that gives $\max_{(j,k)}\{w(j,k) + f(k)\}$. We can construct the longest path by following these recorded links from 0 to m. Generalizing this technique to other types of graphs, e.g., two-dimensional multistage networks (which are used for multi-unit auctions described in Section 4), or general directed acyclic graphs, is rather straightforward.

In the rest of this paper, we describe our secure dynamic programming protocol based on this path-finding problem in a one-dimensional directed graph. As discussed in Section 4, generalizing the proposed protocol to general cases is also straightforward.

3 Secure Dynamic Programming

In this section, we present our secure dynamic programming protocol and discuss its security and efficiency.

3.1 Preliminaries

Before describing our protocol, 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 [3], which has all of these properties, for describing our protocol. However, our protocol can be implemented using other encryption methods (e.g., Paillier encryption [10]) 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 plain text 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.

The secret key is $x \in \mathbf{Z}_q$ and the corresponding public key is $g, y = g^x$.

Anyone can encrypt message $M \in \mathbf{Z}_p$ just using public key $g, y = g^x$, 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 secret key $x \in \mathbf{Z}_q$ can decrypt ciphertext $E(M) = (A = g^r, B = y^r M)$, i.e., compute $B/A^x = M$.

Let us show a very simple example.

Public keys: $p = 23, g = 5, y = 2$

Secret key: $x = 2$

Please note that this setting is intended only to show a simple, easy to follow explanation; p is too small and obtaining secret key x from the public keys is easy. In standard settings, p is chosen from about 768 to 1024-bit primes.

For encrypting plaintext 5, if we choose random value $r = 5$, we obtain ciphertext $E(5) = (g^5, y^5 M) = (5^5 \bmod 23, 2^5 \cdot 5 \bmod 23) = (20, 22)$. For decrypting $E(5) = (20, 22)$, we can obtain $22/20^2 \bmod 23 = 22/9 \bmod 23 = 5$.

- 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 looks totally different and we cannot know whether the original plaintexts are the same or not without decrypting them.

  For example, if we choose random value $r = 6$ for encrypting plaintext 5, we obtain ciphertext $E(5) = (g^6, y^6 M) = (5^6 \bmod 23, 2^6 \cdot 5 \bmod 23) = (8, 21)$. This ciphertext looks totally different from another ciphertext $E(5) = (20, 22)$.

  Formally, if we assume that the Decision Diffie-Hellman (DDH) problem [18] is infeasible, ElGamal encryption E is indistinguishable encryption. More specifically, from $(M_0, M_1, E(M_b))$ where $b \in \{0, 1\}$, one cannot find $b \in \{0, 1\}$ with probability greater than one half. This means ciphertext $E(M)$ leaks no information about plaintext M.

- 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 this property, we can take the product of two plaintexts by taking the product of two ciphertexts without decrypting them.

  For example, given $E(5) = (20, 22)$ and $E(5) = (8, 21)$, we can obtain $E(5 \cdot 5 \bmod 23 = 2) = (20 \cdot 8 \bmod 23, 22 \cdot 21 \bmod 23) = (22, 2)$. Note that this operation can be done without decrypting these ciphertexts. By decrypting $(22, 2)$, we can obtain $2/22^2 \bmod 23 = 2/1 = 2$.

- Randomization: In ElGamal encryption, one can create a new randomized ciphertext $E(M) = (Ag^s, By^s)$ with random value s from the original ciphertext $E(M) = (A = g^r, B = y^r M)$. This is equivalent to make a product of $E(1) = (g^s, y^s)$ and $E(M)$. If we assume that the DDH problem is infeasible, one cannot decide whether a ciphertext is a randomized ciphertext of the original ciphertext or not.

  For example, given $E(5) = (20, 22)$, and $E(1) = (9, 12)$, where $r = 10$, the resulting product is $E(5)E(1) = E(5) = (20 \cdot 9 \bmod 23, 22 \cdot 12 \bmod 23) = (19, 11)$. $E(5) = (20, 22)$ and $E(5) = (19, 11)$ look totally different so we cannot know whether the original plaintexts are the same or not without decrypting them.

3.2 Overview

The basic idea of our secure dynamic programming protocol is as follows:

- Multiple servers (called evaluators), each of which corresponds to one node of a graph, cooperatively execute dynamic programming.

- Each evaluator of node i knows only its evaluation value $f(i)$ and does not know any weight of any link.

To realize this secure dynamic programming protocol, we have to answer the following question: how can we determine the maximum of weights, and add a constant to a weight without revealing the weights themselves? The decision on how to represent and encrypt the weight is crucial to making these tasks feasible.

We are going to explain our representation that makes these tasks feasible.

- Vector representation: We represent weight w $(1 \leq w \leq n)$ by encrypted weight $\vec{e}(w)$ that is the following vector of ciphertexts:

$$\begin{aligned} \vec{e}(w) &= (e_1, \ldots, e_n) \\ &= (\underbrace{E(z), ..., E(z)}_{w}, \underbrace{E(1), ..., E(1)}_{n-w}), \end{aligned}$$

  where $E(1)$ and $E(z)$ denote the encryption of 1 and common public element $z(\neq 1)$, respectively. n is chosen so that it is large enough to represent the length of the longest path. z is chosen so that $z^k \bmod p \neq 1$ for $0 < k < q$.

  For example, we can represent weight $w = 1$ by encrypted weight $\vec{e}(w)$ using $z = 5$, $n = 4$:

$$\begin{aligned} \vec{e}(w) &= (E(z), E(1), E(1), E(1)) \\ &= ((5, 10), (2, 4), (10, 8), (4, 16)), \end{aligned}$$

  if we choose r for each element as $1, 2, 3, 4$, respectively. Since each encryption is done independently, i.e., we use different r for encrypting each element, they are indistinguishable from each other; we cannot tell w without decrypting each element.

- Add a constant: We can add a constant f to encrypted weight $\vec{e}(w) = (e_1, \ldots, e_n)$ without decrypting $\vec{e}(w)$ nor learning w. By shifting and randomizing $\vec{e}(w)$, we can obtain

$$\vec{e'}(w + f) = (\underbrace{E(z), \ldots, E(z)}_{f}, e'_1, \ldots, e'_{n-f})$$

  where e'_j is a randomization of ciphertext e_j. Due to randomization, one can obtain no information about constant f from $\vec{e}(w)$ and $\vec{e'}(w + f)$.

  For example, we can add a constant $f = 1$ to $\vec{e}(w) = ((5, 10), (2, 4), (10, 8), (4, 16))$ by shifting $\vec{e}(w)$ to right by 1. Using $E(z) = (11, 7)$, where $r = 9$, we obtain $((11, 7), (5, 10), (2, 4), (10, 8))$. Furthermore, we mask the 2-nd, 3-rd, and 4-th elements by taking products with $E(1)$, i.e., $(9, 12), (22, 1), (18, 2)$, where $r = 10, 11, 12$, respectively. Finally, we obtain:

$$\begin{aligned} &\vec{e'}(w + f) \\ &\quad = (E(z), E(z)E(1), E(1)E(1), E(1)E(1)) \\ &\quad = ((11, 7), (5, 10) \cdot (9, 12), (2, 4) \cdot (22, 1), \\ &\qquad (10, 8) \cdot (18, 2)) \\ &\quad = ((11, 7), (22, 5), (21, 4), (19, 16)) \\ &\quad = (E(z), E(z), E(1), E(1)) \end{aligned}$$

  Please note that we can perform these operations without decrypting $\vec{e}(w)$. Also, if we compare $\vec{e}(w)$ and $\vec{e'}(w + f)$, we cannot know the amount of the shift without decrypting them.

- Find the maximum: We can find the maximum of encrypted weights $\vec{e}(w_i) = (e_{1,i}, \ldots, e_{n,i})$ without leaking information about the weights that are not the maximum as follows. Consider the componentwise product of all weights

$$\prod_i \vec{e}(w_i) = (\prod_i e_{1,i}, \ldots, \prod_i e_{n,i}).$$

  Observe that, due to the homomorphic property, the j-th component of this vector has the following form

$$c_j = \prod_i e_{j,i} = E(z^{S(j)})$$

  where $S(j) = \#\{i \mid j \leq w_i\}$ is the number of weights that are equal or greater than j. Notice that $S(j)$ monotonically reduces as j increases. To find the maximum of these weights, we decrypt c_j and check whether decryption $D(c_j)$

is equal to 1 or not from $j = n$ to $j = 1$ until we find the largest j s.t. $D(c_j) \neq 1$. This j is equal to $\max_i\{w_i\}$, i.e., the maximum of the weights.

For example, we can represent another weight $v = 3$ by encrypted weight

$$\begin{aligned}\vec{e}(v) &= (E(z), E(z), E(z), E(1)) \\ &= ((20, 22), (8, 21), (17, 19), (16, 3)),\end{aligned}$$

if we choose r for each element as $5, 6, 7, 8$, respectively.

To find $\max\{w + f, v\}$, we create the product of $\vec{e'}(w + f)$ and $\vec{e}(v)$, i.e.,

$$\begin{aligned}&\vec{e'}(w + f) \cdot \vec{e}(v) \\ &= (E(z)E(z), E(z)E(z), E(1)E(z), E(1)E(1)) \\ &= ((11, 7) \cdot (20, 22), (22, 5) \cdot (8, 21), \\ &\quad (21, 4) \cdot (17, 19), (19, 16) \cdot (16, 3)) \\ &= ((13, 16), (15, 13), (12, 7), (5, 2)) \\ &= (E(z^2), E(z^2), E(z), E(1)).\end{aligned}$$

By decrypting the 4-th element, we obtain $B/A^x = 2/5^2 \bmod 23 = 2/2 \bmod 23 = 1$. By decrypting the 3-rd element, we obtain $B/A^x = 7/12^2 \bmod 23 = 7/6 \bmod 23 = 5 = z$. Thus we can be convinced that $\max\{w + f, v\}$ is equal to 3.

Using these processes, we can find the maximum of weights, and add a constant to a weight; thus we can perform dynamic programming procedures using this vector representation.

3.3 Protocol

Now, we will present details of our protocol. There is a weight publisher $P_{(j,k)}$ for each link (j, k), and an evaluator T_j for each node j. In an auction setting, a weight publisher corresponds to a bidder, and an evaluator corresponds to a part of the multiple auction servers.

The basic idea of this protocol is as follows. The weight publisher $P_{(j,k)}$ encrypts its weight $w(j, k)$ using T_j's encryption function. Evaluator T_k (who cannot decrypt this information) then calculates $w(j, k) + f(k)$. Next, evaluator T_j calculates $f(j)$ using this information without knowing $w(j, k)$.

The details of the protocol can be described as follows. For a while, we assume that each evaluator acts honestly, i.e., it does not try to decrypt the information it does not need to know to execute the protocol. We show how to prevent the evaluator from learning the information it does not need to know by introducing multiple evaluators for each node in Section 3.5.

- *Preparation* : Evaluator T_j of node j generates a secret key and a public key of encryption function E_j of node j.
- *Publish weight* : Weight publisher $P_{(j,k)}$ of link (j, k) decides its weight $w(j, k)$ $(1 \leq w(j, k) \leq n)$ and publishes encrypted weight: $\vec{e}_{(j,k)}(w(j, k)) = (\underbrace{E_j(z), ..., E_j(z)}_{w(j,k)}, \underbrace{E_j(1), ..., E_j(1)}_{n-w(j,k)})$ of its weight $w(j, k)$ using encryption function E_j of node j.
- *Find optimal value* : Evaluator T_j of node j determines $f(j) = \max_{(j,k)}\{w(j, k) + f(k)\}$ from $\vec{e'}_{(j,k)}(w(j, k) + f(k))$ using the method described in Section 3.2. Evaluator T_j then creates $\vec{e'}_{(i,j)}(w(i, j) + f(j))$ from $\vec{e}_{(i,j)}(w(i, j))$ using the method described in Section 3.2, and publishes it. By iterating these procedures, we can obtain optimal value $f(0)$, i.e., the length of the longest path.

  Please note that since T_j has the secret key of encryption E_j, it can decrypt $\vec{e}_{(j,k)}(w(j, k))$ or $w(j, k)$. We discuss how to prevent T_j from illegally learning $w(j, k)$ in Section 3.5.
- *Find optimal path* : After finding the optimal value, we trace the nodes back to determine the optimal path. Assume that evaluator T_{i_0} announces that node j_0 attains maximum $f(i_0)$. Evaluator T_{j_0} who knows maximum $f(j_0)$ then finds node k_0 that attains maximum $f(j_0)$, i.e., $w(j_0, k_0)+f(k_0) = f(j_0) = \max_{(j_0,k)}\{w(j_0, k)+f(k)\}$,

by decrypting the $f(j_0)$-th component of $\vec{e'}_{(j_0,k)}(w(j_0,k)+f(k))$ for all k and checking that the decryption of the component is equal to 1 or z (if it is equal to z, the node in the one that attains the maximum). Evaluator T_{j_0} then announces that node k_0 attains maximum $f(j_0)$. By iterating these procedures, we can find the optimal path.

3.4 Example

Here, we give an example of protocol execution using the one-dimensional directed graph described in Figure 2. There are four nodes $\{0,1,2,3\}$, four links $\{(0,1),(1,2),(1,3),(2,3)\}$, with weights $w(0,1)=3, w(1,2)=2, w(1,3)=1, w(2,3)=0$.

First, weight publishers $P_{(0,1)}, P_{(1,2)}, P_{(1,3)}, P_{(2,3)}$ publish encrypted weights (where $n=6$).

$\vec{e}_{(0,1)}(w(0,1))$
$\quad = (E_0(z), E_0(z), E_0(z), E_0(1), E_0(1), E_0(1)),$
$\vec{e}_{(1,2)}(w(1,2))$
$\quad = (E_1(z), E_1(z), E_1(1), E_1(1), E_1(1), E_1(1)),$
$\vec{e}_{(1,3)}(w(1,3))$
$\quad = (E_1(z), E_1(1), E_1(1), E_1(1), E_1(1), E_1(1)),$
$\vec{e}_{(2,3)}(w(2,3))$
$\quad = (E_2(1), E_2(1), E_2(1), E_2(1), E_2(1), E_2(1)),$
respectively.

By decrypting $\vec{e}_{(2,3)}(w(2,3))$, evaluator T_2 knows $f(2)$ is 0, and creates

$$\vec{e'}_{(1,2)}(w(1,2)+f(2)) = (E_1(z), E_1(z), E_1(1), E_1(1), E_1(1), E_1(1))$$

by randomizing $\vec{e}_{(1,2)}(w(1,2))$.

To find $\max\{w(1,2)+f(2), w(1,3)\}$, evaluator T_1 creates the product of $\vec{e'}_{(1,2)}(w(1,2)+f(2))$ and $\vec{e}_{(1,3)}(w(1,3))$, i.e.,

$$\vec{e'}_{(1,2)}(w(1,2)+f(2)) \cdot \vec{e}_{(1,3)}(w(1,3)) = (E_1(z^2), E_1(z), E_1(1), E_1(1), E_1(1), E_1(1)),$$

and decrypts the 6-th, 5-th, 4-th, 3-rd, and 2-nd components and finds z. Thus T_1 is convinced that $f(1)=\max\{w(1,2)+f(2), w(1,3)\}$ is 2. T_1 then shifts $\vec{e}_{(0,1)}(w(0,1))$ by $f(1)$, randomizes it to yield:

$$\vec{e'}_{(0,1)}(w(0,1)+f(1)) = (E_0(z), E_0(z), E_0(z), E_0(z), E_0(z), E_0(1))$$

that represents

$f(0)=w(0,1)+\max\{w(1,2)+f(2), w(1,3)\}=5.$

To find the optimal path, evaluator T_1 decrypts the $f(1)$-th(=2-nd) component of $\vec{e'}_{(1,2)}(w(1,2)+f(2))$ and $\vec{e}_{(1,3)}(w(1,3))$ and obtains z and 1, respectively. Thus T_1 is convinced that link $(1,2)$ attains $\max\{w(1,2)+f(2), w(1,3)\}=2$.

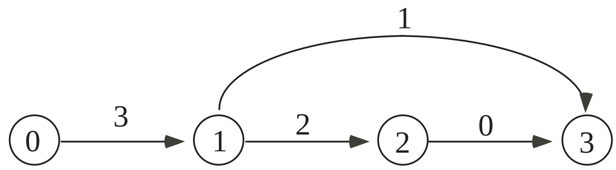

図 2 Example of one-dimensional directed graph (II)

3.5 Security

In this section, we discuss the security of our protocol. To classify the obtained degree of security, let us define the levels of information leakage as follows:

level 0: No information about weights, except the weights of the links in the optimal path, is leaked.

level 1: Besides the information leakage of level 0, one evaluation value $f(j)$, where node j is not on the optimal path is leaked.

level k: Besides the information leakage of level 0, evaluation values $f(j_1),\ldots,f(j_k)$, where nodes $j_1,\ldots,j_k$ are not on the optimal path, are leaked.

level m: Besides the information leakage of level 0, evaluation values $f(j)$ for all nodes $0 \le j < m$ are leaked.

A passive adversary is defined as the one who tries to obtain secret information based on the published information of the protocol and all information available to collusive participants. We assume that all collusive participants perform the protocol properly, i.e., they cannot actively manipulate the protocol.

From the indistinguishability of ElGamal encryption, one can obtain no information about weight w from $\vec{e}(w)$. Due to randomization, one can obtain no information about constant f from $\vec{e}(w)$ and $\vec{e'}(w+f)$. Hence, if a passive adversary cannot collude with any evaluator, our protocol guarantees level 0 information leakage, i.e., our protocol leaks no information about weights except the optimal path.

On the other hand, if the adversary can collude with evaluator T_j, the adversary can access to all information about $w(j,k)$, since T_j knows the secret key of E_j. To prevent this, we can implement the function of evaluator T_j by plural evaluators $T_j^1, T_j^2, \ldots$ and introduce a threshold value t, so that when $t+1$ evaluators for node j cooperate, they can decrypt E_j, but cannot do so if there are less than or equal to t evaluators.

More specifically, each distributed evaluator $T_j^1, T_j^2, \ldots$ has only a share of the secret key; thus any collusion of t (or less than t) evaluators cannot decrypt E_j [17]. In the preparation phase, the secret and public key are generated in a distributed way [11] and each distributed evaluator $T_j^1, T_j^2, \ldots$ has only a share of the secret key. To find the optimal value and path, the decryption is performed in a distributed fashion by each distributed evaluator $T_j^1, T_j^2, \ldots$ that has a share of the secret key, through a secure channel.

By introducing multiple evaluators for each node, our protocol guarantees that the information leakage is at most level 1 even if a passive adversary collude with one evaluator T_j^i, i.e., the adversary learns only evaluation value $f(j)$. In general, the information leakage of level 1 can be quite acceptable since the adversary cannot discover any additional weights of links from $f(j)$ except the optimal path. An exception occurs when node j has only one link (j,m). In this case, the evaluation value $f(j,m)$ is equal to $w(j,m)$. If this is the case, we can prevent the leakage of this information by making weight publisher $P_{(j,m)}$ act as an evaluator for node j. Furthermore, if a passive adversary can collude with t evaluators, our protocol guarantees that the information leakage is at most level t.

表 1 communication complexity

	pattern	round	volume
publish weight	$P_{(j,k)} \rightarrow BB$	l	n
find optimal value	$T_j \leftrightarrow BB$	m	$n \times v$
find optimal path	$T_j \leftrightarrow BB$	m	1

If the participants can accept level m information leakage, then we can drastically reduce the required number of servers, i.e., we need to use only $t+1$ evaluators, while the original protocol requires $m(t+1)$ servers. Each evaluator T^i does the operations of T_j^i for all j in the original protocol. Even if a passive adversary colludes with t evaluators, the adversary can obtain no more information than level m.

Even if the obtained level of information leakage is m, i.e., evaluation values $f(j)$ for all j are leaked, the adversary can learn the exact value of weight $w(j,k)$ only if (j,k) is the only link directed from j. In that case, $f(j) - f(k) = w(j,k)$. Otherwise, the adversary cannot learn the exact value of any weight except those in the optimal path.

3.6 Efficiency

Table 1 shows communication pattern, round complexity, and data volume per round of each step of our protocol. Here, l is the number of links, m is the number of nodes, v is the maximum number of links for a node, and n is the number of components of an encrypted weight that must be greater than the length of the longest path. We consider that there is bulletin board BB, and that each agent sends data to BB to publish it, and accesses to BB to find the published data.

Since the round complexity is proportional to the number of links or nodes, our scheme can handle large numbers of auction items. However, since the data volume and computational complexity is proportional to the range of weights, our scheme becomes costly when the total sum of the prices is large.

4 Application to Combinatorial Auctions

In this section, we discuss the application of our secure dynamic programming protocol to several types of combinatorial auctions, i.e., multi-unit auctions, linear-good auctions, and general combinatorial auctions.

4.1 Multi-unit Auctions

In multi-unit auctions, m units of an identical item are auctioned. Each bidder B_k $(1 \leq k \leq N)$ declares his/her bidding price $b_k(j)$ for each quantity j, where $1 \leq j \leq m$. The goal is to find the allocation that maximizes the sum of the bidding prices.

The optimal allocation in a multi-unit auction can be obtained by solving the following recurrence formula [20][21].

$$f((1,j)) = b_1(j),$$
$$f((k,s)) = \max_{0 \leq j \leq s} \{f((k-1, s-j)) + b_k(j)\}.$$

In this formula, $f((k,s))$ represents the value of the optimal solution of a sub-problem, i.e., optimally allocating s units among k participants, i.e., bidders $B_1, B_2, \ldots, B_k$. The optimal solution is given by $f((N,m))$.

Applying our secure dynamic programming protocol to this problem is rather straightforward. We use plural evaluators for each node (k,s). Furthermore, for node $(1,j)$, its evaluation value $f((1,j))$ represents the bidding price of Bidder B_1 for j units. To prevent the leakage of this information, bidder B_1 should act as an evaluator for node $(1,j)$.

4.2 Linear-Good Auctions

In a linear-good auction [13][20], there exist m goods $G = \{1, 2, \ldots, m\}$. These goods are sequentially ordered. Each bidder $B_k (1 \leq k \leq N)$ bids his/her bidding price $b_k([l_k, u_k])$ for an interval $[l_k, u_k] \subseteq G$ of goods, i.e., a bidder wants to obtain a continuous sequence of goods. We assume each bidder bids for a single interval (or equivalently bids for several intervals independently). The result of the auction is the allocation of the m goods that maximizes the sum of all bidders' bidding prices.

Auctions for linear goods can be used for time scheduling (e.g., for the allocation of time slots in a conference room), or for the allocation of a one-dimensional space (e.g., for parts of a seashore), or spectrum right auctions in which each bidder (wireless carrier) wants a continuous frequencies to minimize interference between carriers [8].

This problem can be directly mapped into the problem of finding the longest path in a one-dimensional graph. Consider the graph with $m+1$ nodes $\{0, 1, 2, \ldots, m\}$, i.e., there exists an initial node 0 and nodes that correspond to goods. For Bidder B_k, who bids his/her bidding price $b_k([l_k, u_k])$ for interval $[l_k, u_k]$, we represent his/her bid as a link between $(l_k - 1, u_k)$ and set its weight as $b_k([l_k, u_k])$. Also, we add a dummy link between $(j, j+1)$ for each $0 \leq j < m$, whose weight is 0, to prevent a node from becoming a dead-end.

The allocation of the m goods that maximizes the sum of all bidders' bidding prices corresponds to the longest path from node 0 to node m, and can be securely computed by our secure dynamic programming protocol. Also, our method can handle multiple links between two nodes.

The idea of linear-good auctions can be generalized to route auctions, in which each bidder bids for a path in a general graph. The result of the auction is the path from the start node to the destination node that maximizes/minimizes the sum of all bidders' bidding prices.

One application of route auctions can be a transportation task assignment problem, in which the auctioneer wants to transport his/her cargo from a starting city to a destination city. There are several transportation companies. Each company bids his/her price to carry the cargo for a path (which can be only a part of the total journey), and the auctioneer chooses the combination of paths that minimizes the total cost. This problem can be formalized as finding a shortest path in a graph, and so can be solved using the secure dynamic programming protocol presented in this paper.

4.3 General Combinatorial Auctions

In a general combinatorial auction, there exist multiple different goods $G = \{1, 2, \ldots, m\}$. Each bidder B_k bids his/her price $b_k(S)$ for each subset/bundle $S \subseteq G$. The goal is to find the allocation of goods that maximizes the total price.

As described in [13], this problem can be solved by dynamic programming as follows. For each bundle S, $b(S)$ represents the highest price of the bundle. For simplicity, we assume that each bundle has at least one bid (otherwise, we can put a dummy bid with price 0). For each bundle $S \subseteq G$, we create a node $(S, |S|)$, where $|S|$ represents the number of goods in S. For each node $(S, |S|)$, we place the following directed links:

- $((S, |S|), (\{\}, 0))$, where $w((S, |S|), (\{\}, 0)) = b(S)$
- $((S, |S|), (C, |C|))$, where $C \subset S$ and $|C| \geq |S|/2$, and $w((S, |S|), (C, |C|)) = b(S \setminus C)$.

We show an example of nodes and links of a combinatorial auction, where $G = \{1, 2, 3\}$ in Figure 3.

The optimal allocation is given by the longest path from the initial node (G, m) to the terminal node $(\{\}, 0)$. It is clear that this problem can be solved using the secure dynamic programming protocol presented in this paper. Our method can handle multiple links between two nodes. Therefore, we don't need to pre-select the bid with the highest price for each bundle.

One obvious disadvantage of this approach is that the number of nodes becomes very large, i.e., 2^m. However, this seems somewhat inevitable if we are to solve this problem using dynamic programming, since the winner determination problem of a general combinatorial auction is NP-complete [13].

5 Discussions

There have been various works on secure auction servers and protocols [4][7][9]. However, as far as the authors know, there has been no other research on secure dynamic programming protocols nor on secure combinatorial auction servers/protocols based on dynamic programming techniques.

Kikuchi [7] developed a secure $M+1$-st price auction protocol, in which multiple units of an identical item are auctioned, but each participant is assumed to require only one unit. Suzuki and Abe [19] also developed a secure $M+1$-st auction protocol based on Homomorphic Encryption. Our method can be applied to the cases where each participant wants to buy multiple units.

It is well known that any combinatorial circuit can be computed securely using general-purpose multi-party protocols [2][6]. Therefore, if we can construct a combinatorial circuit that implements a dynamic programming algorithm, in principle, such an algorithm can be executed securely (thus we don't need to develop a specialized secure protocol for dynamic programming). However, to execute such a general-purpose multi-party protocol, for each computation of an AND gate in the circuit,

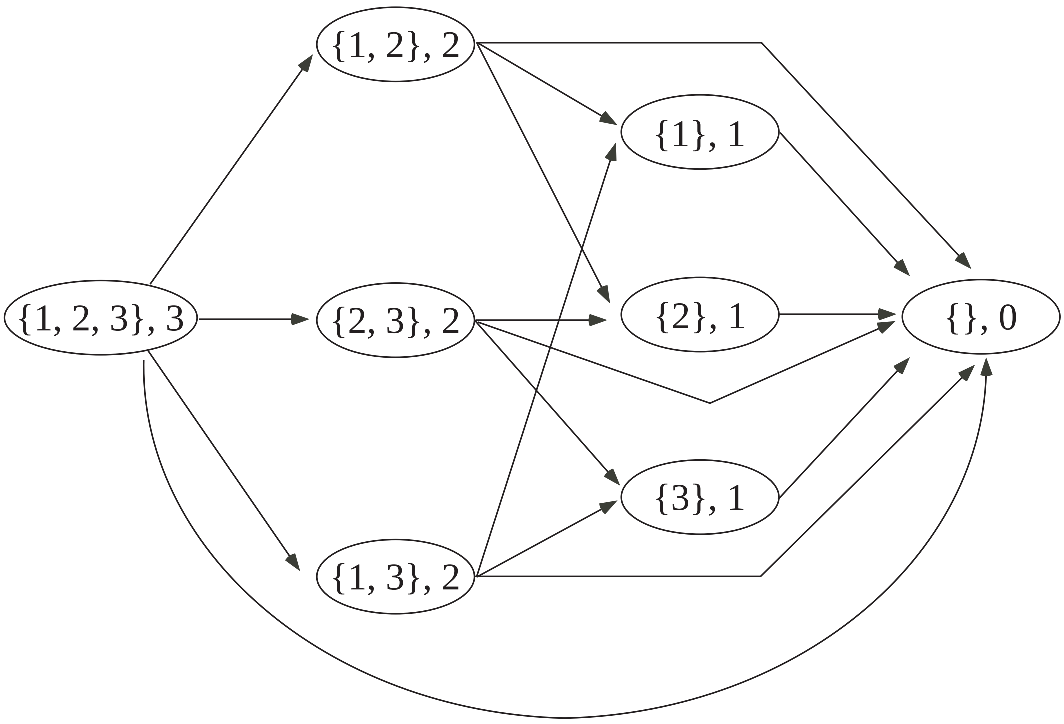

図 3 Example of graph for combinatorial auction

the evaluators must communicate with each other. Using such a general purpose multi-party protocol for a large-scale dynamic programming application is not practical at all due to the required communication costs.

Naor et al. [9] proposed a general method for executing any auction protocol including combinatorial auction protocols based on a technique called the garbled circuit [23]. This method does not require interactive communications among multiple evaluators. However, to apply this method to a dynamic programming application, we first need to construct a combinatorial circuit that implements a dynamic programming algorithm, then scramble this circuit so that an agent executing this circuit cannot learn the actual contents of the circuit. Using this method for large-scale dynamic programming applications, including combinatorial auctions, would be rather difficult.

6 Conclusion

In this paper, we presented a secure dynamic programming protocol that utilizes homomorphic encryption. By using this method, multiple servers/evaluators cooperatively perform dynamic programming procedures for solving a combinatorial optimization problem by using the private information sent from agents as inputs. Although the evaluators can compute the optimal solution correctly, the inputs are kept secret even from the servers. Furthermore, we discussed its application to several combinatorial auctions, i.e., multi-unit auctions, linear-good auctions, and general combinatorial auctions.

Although dynamic programming is a very powerful, widely-used combinatorial optimization method, applying dynamic programming techniques to general combinatorial auctions is not feasible for large-scale problems, since it requires an exponential number of nodes. Currently, we are investigating the protocols for securely computing average-case efficient algorithms [5][15][16].

参考文献

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of 20th ACM Symposium on the Theory of Computing*, pages 1–10, 1988.

[3] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. on Information Theory*, IT-31(4):469–472, 1985.

[4] M. K. Franklin and M. K. Reiter. The design and implementation of a secure auction server. *IEEE Transactions on Software Engineering*, 22(5):302–312, 1986.

[5] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computation complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 548–553, 1999.

[6] O. Goldreich, S. Micli, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of 19th ACM Symposium on the Theory of Computing*, pages 218–229, 1987.

[7] H. Kikuchi. (M+1)st-Price auction protocol. In *Proceedings of Financial Cryptography 2001*, 2001.

[8] T. Matsui and T. Watanabe. Sealed bid multi-object auctions with necessary bundles and its application to spectrum auctions. In *Proceedings of the 4th Pacific Rim International Workshop on Multi-agents (PRIMA-2001)*, pages 78–92. Springer-Verlag, 2001. Lecture Notes in Artificial Intelligence 2132.

[9] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99)*, 1999.

[10] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Proceedings of EUROCRYPT '99*, pages 223–238, 1999.

[11] T. Pedersen. A threshold cryptosystem without a trusted party. *Proceedings of EUROCRYPT '91*, pages 522–526, 1991. Lecture Notes in Computer Science 547.

[12] E. Rasmusen. *Games and Information*. Blackwell, 1994.

[13] M. H. Rothkopf, A. Pekeč, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[14] Y. Sakurai, M. Yokoo, and K. Kamei. An efficient approximate algorithm for winner determination in combinatorial auctions. In *Proceedings of the Second ACM Conference on Electronic Commerce (EC-00)*, pages 30–37, 2000.

[15] T. Sandholm. An algorithm for optimal winner determination in combinatorial auction. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 542–547, 1999.

[16] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. CABOB: A fast combinatorial algorithm for optimal combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 1102–1108, 2001.

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

[18] M. Stadler. Publicly verifiable secret sharing. *Proceedings of EUROCRYPT '96*, pages 190–199, 1996. Lecture Notes in Computer Science 1070.

[19] K. Suzuki and M. Abe. M+1-st price auction using homomorphic encryption. Technical Report ISEC 2001-11, IEICE, 2001.

[20] M. Tennenholtz. Some tractable combinatorial auctions. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 98–103, 2000.

[21] S. van Hoesel and R. Müller. Optimization in electronic markets: examples in combinatorial auctions. *Netnomics*, 3:23–33, 2001.

[22] H. R. Varian. Economic mechanism design for computerized agents. In *Proceedings of the First Usenix Workshop on Electronic Commerce*, 1995.

[23] A. C. Yao. How to generate and exchange secrets. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[24] M. Yokoo, Y. Sakurai, and S. Matsubara. Robust combinatorial auction protocol against false-name bids. *Artificial Intelligence*, 130(2):167–181, 2001.