Collaboratively Solving the Traveling Salesman Problem with Limited Disclosure

Yuan Hong, Jaideep Vaidya, Haibing Lu, Lingyu Wang

To cite this version:

Yuan Hong, Jaideep Vaidya, Haibing Lu, Lingyu Wang. Collaboratively Solving the Traveling Salesman Problem with Limited Disclosure. 28th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec), Jul 2014, Vienna, Austria. pp.179-194, 10.1007/978-3-662-43936-4_12 . hal-01284854

HAL Id: hal-01284854
https://inria.hal.science/hal-01284854

Submitted on 8 Mar 2016

Collaboratively Solving the Traveling Salesman Problem with Limited Disclosure

Yuan Hong[†], Jaideep Vaidya[‡], Haibing Lu[§], and Lingyu Wang[#]

[†]SUNY-Albany, [‡]Rutgers University, [§]Santa Clara University, [#]Concordia University
[†]hong@albany.edu, [‡]jsvaidya@business.rutgers.edu, [§]hlu@scu.edu,
[#]wang@concordia.ca

Abstract. With increasing resource constraints, optimization is necessary to make the best use of scarce resources. Given the ubiquitous connectivity and availability of information, collaborative optimization problems can be formulated by different parties to jointly optimize their operations. However, this cannot usually be done without restraint since privacy/security concerns often inhibit the complete sharing of proprietary information. The field of privacy-preserving optimization studies how collaborative optimization can be performed with limited disclosure. In this paper, we develop privacy-preserving solutions for collaboratively solving the traveling salesman problem (TSP), a fundamental combinatorial optimization problem with applications in diverse fields such as planning, logistics and production. We propose a secure and efficient protocol for multiple participants to formulate and solve such a problem without sharing any private information. We formally prove the protocol security under the rigorous definition of secure multiparty computation (SMC), and demonstrate its effectiveness with experimental results using real data.

Keywords: Privacy, Secure Multiparty Computation, Optimization

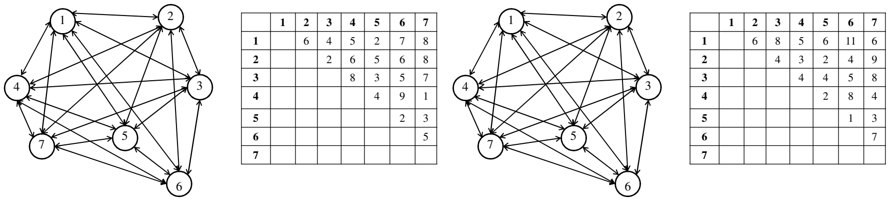
1 Introduction

Collaboration amongst different parties occurs frequently in the modern business world. Given the increasing resource constraints, it makes sense for different companies to jointly optimize their operations in delivering, production planning, scheduling, inventory control, etc. Indeed, joint optimization has led to significant savings when successfully carried out. However, such collaboration is normally the exception, rather than the rule. The reason for this is the high degree of trust required, wherein proprietary data has to be shared with an external party which can then carry out the optimization. To deal with this, privacy-preserving solutions have been developed to enable collaborative optimization for several specific problems [1–5]. In this paper, we focus on the traveling salesman problem (TSP). TSP is a fundamental optimization problem, and can be used in many applications such as logistics, planning, and production. We first show how a simple two-party collaborative TSP can be formulated [1]:

Example 1. There are two shipping companies, denoted $Alice$ and Bob, which offer delivery services among seven cities, City $1, \dots, 7$. Figure 1 shows the city connectivity and corresponding delivery cost for both companies.

A client E_C wants to decide which shipping company to employ in order to ship their goods to a list of cities (e.g., $1, 2, 3, 5$, and 6) with the lowest overall cost. However, E_C is reluctant to let either $Alice$ or Bob know its list of cities before signing the contract, and $Alice$ and Bob also do not want to share all their delivery cost information with E_c or other parties. How can E_C make this decision under such privacy concerns?

The prior solution [1] is to let E_C securely solve "Two" two-party TSPs with $Alice$ and Bob "respectively", and then choose the lower cost obtained from $Alice$ and Bob.



	1	**2**	**3**	**4**	**5**	**6**	**7**
1		6	4	5	2	7	8
2			2	6	5	6	8
3				8	3	5	7
4					4	9	1
5						2	3
6							5
7							

(a) $Alice$'s Service Network and Cost

	1	**2**	**3**	**4**	**5**	**6**	**7**
1		6	8	5	6	11	6
2			4	3	2	4	9
3				4	4	5	8
4					2	8	4
5						1	3
6							7
7							

(b) Bob's Service Network and Cost

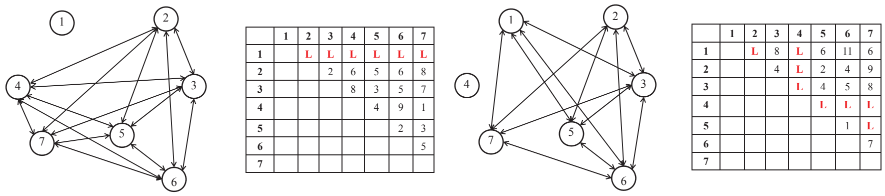
Fig. 1. Two-party Collaboration in [1]

While this enables E_C to choose the lowest cost provider, it does not enable the lowest cost overall, since it forces one company to be used to do all of the shipments. In this paper, we tackle the privacy concerns in a multiparty TSP (rather than two-party TSP [1]) in which the global minimum cost can be further reduced from all the participants, detailed as below.

Cost Reduction. Assume that the optimal solutions derived for E_C from Alice and Bob in Figure 1 are 1) "$Alice : 1 \rightarrow 5 \rightarrow 6 \rightarrow 2 \rightarrow 3 \rightarrow 1$" with total cost 2+2+6+2+4=16, and 2) "$Bob : 1 \rightarrow 5 \rightarrow 6 \rightarrow 2 \rightarrow 3 \rightarrow 1$" with total cost 6+1+4+4+8=23. Per the work in [1], E_C then employs $Alice$ to deliver their goods to the destinations due to $16 < 23$. However, Bob indeed offers cheaper shipping rates among some cities, e.g., $5 \rightarrow 6 \rightarrow 2$. If E_C can employ both shipping companies to deliver goods, even though on the same route "$1 \rightarrow 5 \rightarrow 6 \rightarrow 2 \rightarrow 3 \rightarrow 1$", the global cost could be $2 + 1 + 4 + 2 + 4 = 13 < 16 < 23$. As more shipping companies participate in the collaboration, significant cost saving can be realized.

Practicability and Availability. In reality, many companies need to make decisions for the delivery of considerable number of cities. From the economic perspective, they normally do not always contract with only one shipping company for all of their destinations. Then, the two-party TSP in [1] clearly cannot meet the practical cost-minimizing demand of E_C. Besides this, such TSP has many other drawbacks. Specifically, if neither $Alice$ nor Bob is able to complete the delivery for E_C by their own: e.g., $Alice$ cannot deliver any shipping to City 1 while Bob cannot deliver shipping between City 1 and 2 (Figure 2), then the model in [1] cannot provide a solution.

To address the above limitations, we formulate and securely solve a novel cost-reducible collaborative TSP, which enables the client to find the global minimum cost from one or more shipping companies, and enables the shipping companies not to lose



	1	2	3	4	5	6	7
1		L	L	L	L	L	L
2			2	6	5	6	8
3				8	3	5	7
4					4	9	1
5						2	3
6							5
7							

	1	2	3	4	5	6	7
1		L	8	L	6	11	6
2			4	L	2	4	9
3				L	4	5	8
4					L	L	L
5						1	L
6							7
7							

(a) $Alice$'s Service Network and Cost

(b) Bob's Service Network and Cost

Fig. 2. Limited Shipping Service Network ("L" means unavailable service)

their clients for their service limitation. Indeed, securely solving a fundamental optimization problem like TSP gives great insights to all its the applications with limited information disclosure. With similar settings, our problem formulation and solver can be equally applicable to the proprietary information protection in many other real-world applications such as arranging school bus routes for one or more outsourced companies to pickup the children, scheduling service calls at cable firms, and manufacturing circuit board by drilling holes with machines from different entities [6]. Therefore, the main contributions of this paper are summarized as below:

- We propose a new efficient secure communication protocol to solve the novel cost-reducible collaborative TSP under semi-honest adversarial model – all parties follow the protocol but they are curious to derive private information from each other.
- We give end-to-end security proof for our secure communication protocol under the rigorous definition of Secure Multiparty Computation (SMC) [7, 8], whereas only the security of some building blocks was proven in the two-party TSP [1].
- We demonstrate the effectiveness of our approach with experiments on real data.

The remainder of this paper is organized as follows. We first briefly review the related literature in Section 2. Then we define the problem in Section 3, and present the secure communication protocol in Section 4. We give security and cost analysis in Section 5. Section 6 demonstrates the experimental results. Finally, we conclude this paper and discuss the future work in Section 7.

2 Related Work

We briefly review some of the relevant work on privacy-preserving collaborative (viz. distributed) optimization. Li and Atallah [9] addressed the collaborative linear programming problem between two parties where the objective function and constraints can be arbitrarily partitioned, and proposed a secure simplex method for such problem based on homomorphic encryption and scrambled circuit evaluation. Vaidya [10] proposed a secure revised simplex approach with homomorphic encryption and secure comparison, which is more efficient than Li and Atallah's approach [9]. Catrina and Hoogh [11] presented a solution to solve distributed linear programs based on secret sharing. The protocols utilized a variant of the simplex algorithm and secure computation with fixed-point rational numbers, optimized for such application.

Apart from a direct cryptographic protocol, another typical approach is to transform the original problem into a different space, solve it in that transformed space, and then reconstruct the solution. Du [12] and Vaidya [13] transformed the linear programming problem by multiplying a monomial matrix to both the constraint matrix and the objective function, assuming that one party holds the objective function while the other party holds the constraints. Bednarz et al. [14, 15] pointed out a potential attack to the above transformation approach, which has been resolved in [16, 3]. In addition, Mangasarian presented two transformation approaches for horizontally partitioned linear programs [17] and vertically partitioned linear programs [18] respectively. Li et al. [19] extended the transformation approach [17] for horizontally partitioned linear programs with equality constraints to inequality constraints. Hong and Vaidya identified a potential inference attack to Mangasarian and Li's transformation based approach, and revised the transformation with significantly enhanced security guarantee in [20].

There has also been work on creating privacy-preserving solutions for collaborative combinatorial optimization problems (especially those that are NP-hard). Hong et al. [4] presented a privacy-preserving approach for the well known graph coloring problem. The solution is based on tabu search. The work most relevant to our problem is that of Sakuma et al [1] who proposed a genetic algorithm for securely solving two-party distributed traveling salesman problem (TSP). They consider the case that one party holds the cost vector/matrix while the other party holds the tour vector/matrix. The TSP that is completely partitioned among multiple parties has been discussed but not solved in [1]. In this paper, we consider a cost-reducible collaborative TSP amongst multiple (more than two) parties, and securely solve it with simulated annealing based protocol, which facilitates us to formally prove the security for the approach. Our approach falls under the framework of secure multiparty computation [7, 21, 22, 8], wherein any function can be securely computed without revealing anything to each party except its input, output, and anything that can be derived from them.

3 Problem Formulation

3.1 TSP and Simulated Annealing

TSP is a NP-hard optimization problem that is defined as follows[23]: given a set of cities and the distances between every pair of cities, finding the shortest route that visits all the cities exactly once and returns to the original city (minimizing overall distance). Note that in a more general sense, distance between cities can be replaced with cost.

Simulated annealing is a generic probabilistic meta-heuristic that is widely used to solve this NP-hard problem when the input is large [23]. Simulated annealing was inspired from thermodynamics, based on the behavior of metals cooling and annealing [24]. The basic idea behind simulated annealing is to move from one state (solution) to another state (a neighboring solution), until a good enough solution is found, or the given computation budget has been exhausted. Note that the move is probabilistic in that we may move to a worse solution or choose to stay in the same state even if a better solution has been found some of the times. In TSP, given the traveling route, denoted as x, the objective function of simulated annealing is to minimize $f(x)$ and the algorithm iteratively moves from solutions to their neighboring solution. At each stage, the probability of making a transition from a solution to the neighboring solution is based on an acceptance probability function for two solutions and a sensitivity parameter "Temperature" T. The algorithm is briefly summarized as below:

1. initialize a solution x by randomly selecting a traveling route.
2. randomly pick a neighboring solution x' of x by 2-Opt neighborhood [25], swapping the visited order of two cities. For example, if $x =$"1$\rightarrow 3 \rightarrow 2 \rightarrow 4$", then $x =$"$1 \rightarrow 3 \rightarrow 4 \rightarrow 2$" is its neighboring solution.
3. decide whether to move to the new solution with the probability computed from $f(x)$, $f(x')$ and the sensitivity parameter T. If yes, update the current solution.
4. repeat step 2,3 until achieving the iteration threshold or a satisfactory solution.

3.2 Cost-reducible Collaborative TSP

In a TSP, $G = (V, E)$ is a complete undirected graph where V and E represent the set of cities and (cost) weighted edges respectively. Cost-reducible collaborative TSP involving k shipping companies and one client, where all involved shipping companies successively "Relay" the goods for client E_C on the overall route for further cutting the cost. Then, we define it as k-Relaying Traveling Salesman Problem (k-RTSP).

Definition 1 (k-RTSP). *Assume n cities, and k shipping companies $P_1, \dots, P_k$ that hold k different cost matrices/vectors $y_1, \dots, y_k$, and a client E_C who needs to visit a subset of cities once (finally returning to the original city, as in a Hamiltonian Cycle). Then, k-RTSP is defined as: find the optimal traveling route and minimum total cost for E_C's cities, where every segment of the traveling route is served by the shipping company quoting the cheapest rate among all k shipping companies.*

k-RTSP's optimal traveling route is jointly computed by k shipping companies and the client. If $k = 1$, k-RTSP turns into a two-party TSP [1] since no relay is required.

Vectors in *k*-RTSP Intuitively k shipping companies' cost vectors can be written as:

Table 1. k Shipping Companies' Cost Vectors

$$\boxed{\begin{array}{c} y_1 = (y^1_{(1,2)}, \dots, y^1_{(1,n)}, y^1_{(2,3)}, \dots, y^1_{(2,n)}, \dots, y^1_{(n-1,n)}) \\ \vdots \\ y_k = (y^k_{(1,2)}, \dots, y^k_{(1,n)}, y^k_{(2,3)}, \dots, y^k_{(2,n)}, \dots, y^k_{(n-1,n)}) \end{array}}$$

As shown in Figure 2, if one shipping company does not serve delivery between two cities, e.g., P_1 does not deliver between City 1 and 2, we let $y^1_{(1,2)}$ be a sufficiently large number L. Similarly, client's traveling route vector can be expressed as a boolean vector $x = (x_{(1,2)}, \dots, x_{(1,n)}, x_{(2,3)}, \dots, x_{(2,n)}, \dots, x_{(n-1,n)})$ where $x_{(i,j)} = 1$ means e_{ij} is included in E_C's traveling route;otherwise 0.

Note that the length of vectors x and $y_1, \dots, y_k$ is the total number of city pairs $d = n(n-1)/2$. For simplicity of notations, we use $j = 1, \dots, d$ to indicate the index of $n(n-1)/2$ elements in each of the $k+1$ vectors.

Cost Function and Solution In k-RTSP, every $x_{(i,j)} = 1$ in x (e_{ij} is included in the route) is assigned to a shipping company with cheapest cost (we denote this process as "Route Assignment"). Then x can be drilled down to k boolean vectors for k different shipping companies, for example as below:

Table 2. Route Assignment for Traveling Route Vector

	$x = (0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0)$
Alice	$x_1 = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$
Bob	$x_2 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0)$
Carol	$x_3 = (0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

Denoting $x_{ij} \in \{0, 1\}$ ($i = 1, \dots, k$ and $j = 1, \dots, d$) as whether shipping company P_i delivers goods for the jth corresponding pair of cities or not, the length-d vector x is then drilled down to k length-d vectors $x_1, \dots, x_k$ in route assignment. Thus, $\{x_1, \dots, x_k\}$ is the solution of k-RTSP, which is finer-grained than x. Moreover, the cost function of k-RTSP can be derived as $f(x_1, \dots, x_k) = \sum_{i=1}^{k} x_i \cdot y_i = \sum_{i=1}^{k} (\sum_{j=1}^{d} x_{ij} y_{ij})$.

4 Privacy-preserving Algorithm

Simulated annealing is an efficient meta-heuristic for conventional TSP, thus we build the secure solver for k-RTSP by securing the simulated annealing algorithm. First, we consider the extension of the algorithm to solve k-RTSP (no security), all the parties repeat the following procedures until they find a near-optimal solution:

Client E_C proposes a solution $x = \{x_1, \dots, x_k\}$ and its neighboring solution $x' = \{x'_1, \dots, x'_k\}$ to all the shipping companies $P_1, \dots, P_k$, then $P_1, \dots, P_k$ jointly compute the energy of two solutions $f(x_1, \dots, x_k)$ and $f(x'_1, \dots, x'_k)$, and finally compute the probability of moving from x to x':

$$Prob = \min\{1, exp(-\frac{f(x'_1, \dots, x'_k) - f(x_1, \dots, x_k)}{T})\} \tag{1}$$

If x' outperforms x, solution will move from x to x' with $Prob = 1$; if x' is worse than x, there still exists a probability to move from x to x' (to avoid local optimum).

More importantly, all parties should jointly compute the result without revealing any private information in every iteration. To achieve this, we present a secure communication protocol to solve k-RTSP without private information disclosure.

4.1 Building Blocks

Simulated annealing iteratively computes the energy of various pairs of neighboring solutions and decides whether to move or not. Thus, the secure communication protocol based on it also repeatedly calls some secure functions, which are considered as *Building Blocks* of the protocol. We briefly describe them below:

Secure Scalar Product. The cost function of k-RTSP is given as $f(x_1, \dots, x_k) = \sum_{i=1}^{k} x_i \cdot y_i$ where client E_C holds the traveling route vectors $x_1, \dots, x_k$ and shipping companies $P_1, \dots, P_k$ hold $y_1, \dots, y_k$ respectively. Thus, we implement a secure scalar product protocol based on Paillier's Homomorphic Cryptosystem [26] to securely compute the function (Algorithm 1). *Notice that every party only holds a random number – the sum of the random numbers is the scalar product which is also unknown to everyone besides the private inputs.*

Algorithm 1 Secure Scalar Product

```
Input: Traveling route vector $x = \{x_1, \dots, x_k\}$, cost vectors $y_1, \dots, y_k$
Output: $f(x_1, \dots, x_k) = \sum_{i=1}^{k} x_i \cdot y_i$ co-held by all $k+1$ parties
Client $E_C$ creates a public/private key pair $(pk, sk)$, and encrypts $x_1, \dots, x_k$ to
  $Enc_{pk}(x_1), \dots, Enc_{pk}(x_k)$ with its $pk$
$E_C$ sends $Enc_{pk}(x_1), \dots, Enc_{pk}(x_k)$ and $pk$ to all $k$ shipping companies $P_1, \dots, P_k$
for each party $P_i, i = 1, \dots, k$ do
    $P_i$ generates a random integer $r_i$, encrypts it with the public key $pk$, computes the en-
    crypted scalar product as: $Enc_{pk}(x_i)^{y_i} * Enc_{pk}(r_i) = Enc_{pk}(x_i \cdot y_i + r_i)$ , and sends
    it back to $E_C$
$E_C$ decrypts $Enc_{pk}(x_i \cdot y_i + r_i)$ with its private key $sk$ and obtains a random share $s_i =$
  $x_i \cdot y_i + r_i$
  {Finally, $E_C$ privately holds random numbers $\forall i \in [1, k], s_i = x_i \cdot y_i + r_i$, and $\forall i \in [1, k], P_i$
  privately holds random numbers $-r_i$. The sum of all the shares $f(x_1, \dots, x_k) = \sum_{i=1}^{k} x_i \cdot y_i$
  is unknown to all parties.}
```

Secure Comparison. In simulated annealing, given temperature T, Equation 1 is used to determine if solution $x = \{x_1, \dots, x_k\}$ should be moved to $x' = \{x'_1, \dots, x'_k\}$ or not. In every iteration, $\min\{1, exp(-\frac{f(x'_1, \dots, x'_k) - f(x_1, \dots, x_k)}{T})\}$ should be compared with a random number $\eta \in [0, 1)$ [27]: if $\min\{1, exp(-\frac{f(x'_1, \dots, x'_k) - f(x_1, \dots, x_k)}{T})\} > \eta$, then move $x = \{x_1, \dots, x_k\}$ to $x' = \{x'_1, \dots, x'_k\}$; otherwise, not.

We employ FairplayMP [28] to securely compare the outputs of two functions, thus we need to compare: $f(x_1, \dots, x_k) - f(x'_1, \dots, x'_k)$ and $T \log \eta$, where the inputs for $f(x_1, \dots, x_k)$ and $f(x'_1, \dots, x'_k)$ are the random shares held by all $k+1$ parties (generated from Algorithm 1).

Note that, if $f(x_1, \dots, x_k) - f(x'_1, \dots, x'_k) > T \log \eta$, client E_C moves $x = \{x_1, \dots, x_k\}$ to $x' = \{x'_1, \dots, x'_k\}$ regardless of whether x' is better than x or not; else, the move does not occur. Note that if the temperature T is lowered, simulated annealing algorithm only accepts moving from x to a worse solution x' with closer energy $f(x'_1, \dots, x'_k)$ and $f(x_1, \dots, x_k)$. This guarantees the accuracy of the meta-heuristic.

In secure comparison, similarly, each party cannot learn any input from each other, and only client E_C knows the comparison result "$>$" or "$\leq$".

4.2 Two-level Secure Simulated Annealing (TSSA)

Different from the traditional TSP, k-RTSP has two categories of neighboring solutions since up to k possible costs are available for every pair of cities in G. First, like the well-known TSP, we can find the 2-Opt [25] neighboring solution x' by permuting the visited order of two cities, e.g., $x = 1 \rightarrow 2 \rightarrow 3$ and $x' = 1 \rightarrow 3 \rightarrow 2$. Second, for every solution, e.g., $x = 1 \rightarrow 2 \rightarrow 3$, each route segment ($1 \rightarrow 2$ and $2 \rightarrow 3$) can be potentially assigned to Alice, Bob or Carol in "Route Assignment". Then, $x = 1 \rightarrow 2 \rightarrow 3$ (with a particular route assignment) should have $3^2 - 1 = 8$ neighboring solutions by choosing different combinations of shipping companies, reflected in $\{x_1, \dots, x_k\}$. Ideally, if $y_1, \dots, y_k$ are known to client E_C, E_C can simply select the shipping company with the lowest cost on each segment of the route. However, in secure k-RTSP model, such information is masked and absolutely unknown to E_C. Therefore, we have to run meta-heuristics again to find the optimal route assignment for every solution (traveling route) generated in neighboring route search.

In summary, while securely running the protocol, a top-level simulated annealing is called to search traveling route vector x. For every new solution x, bottom-level simulated annealing will be executed to search the neighborhoods in "Route Assignment". As soon as a near-optimal solution of "Route Assignment" for x is found, x *and its optimal route assignment* will be updated as the current solution. After that, top-level simulated annealing continues to repeatedly traverse x's neighboring traveling route vectors. Therefore, we denote this two level meta-heuristic based protocol as *Two-level Secure Simulated Annealing (TSSA)*.

4.3 Secure Communication Protocol

Our secure communication protocol (TSSA shown in Algorithm 2) ensures that any party cannot learn any private information from each other. More specifically, at Line 1, the protocol is initialized; the loop for top-level simulated annealing is executed between Line 2-22; at Line 3-5, E_C drills down current solution (x, derived from top-level simulated annealing) and initializes the bottom-level simulated annealing, whose loop executes between Line 6-13; in the bottom-level loop, current solution (bottom level) is updated at Line 10-11, and the bottom-level cooling is implemented at Line 13; At Line 19-20 in the top-level loop, current solution for TSSA is updated, and the top-level cooling is implemented at Line 22. TSSA algorithm searches the optimal traveling route vector where the embedded bottom-level simulated annealing is called to search the optimal route assignment for every solution. For any traveling route x and its neighboring solution x' obtained in the top-level simulated annealing, their corresponding optimal route assignments are found by separate bottom-level simulated annealing respectively. Thus, the current best route (with its optimal route assignment) moves toward the optimal traveling route (with the optimal route assignment) of k-RTSP. Here, it is worth noting that:

Algorithm 2 Two-level Secure Simulated Annealing

```
Input: Client $E_C$'s initial traveling route $x$; Shipping companies cost vectors $y_1, \ldots, y_k$; Initial
    temperature $T_1, T_2$; Cooling coefficient $\rho_1, \rho_2$; $\eta \in [0, 1)$
Output: Near-optimal route $best\{x\}$ and the route assignment $best\{x_1, \ldots, x_k\}$
 1: $iter_1 \leftarrow 0$; $best\{x\} \leftarrow x$; $best\{x_1, \ldots, x_k\} \leftarrow \{x_1, \ldots, x_k\}$
    {Top-level simulated annealing searches the optimal route with optimal route assignment}
 2: while $iter_1 < max\_iter_1$ do
 3:     $iter_2 \leftarrow 0$
        {Bottom-level simulated annealing for the optimal route assignment of $best\{x\}$}
 4:     $E_C$ drills down $best\{x\}$ with a random route assignment: $\{x_1, \ldots, x_k\}$
 5:     $best_1\{x_1, \ldots, x_k\} \leftarrow \{x_1, \ldots, x_k\}$
 6:     while $iter_2 < max\_iter_2$ and $route\_assigned(best\{x\}) = 0$ do
 7:         $E_C$ gets $best_1\{x_1, \ldots, x_k\}$'s random neighboring route assignment: $\{x'_1, \ldots, x'_k\}$
 8:         Call Algorithm 1 twice to securely compute $f(x_1, \ldots, x_k) = \sum_{i=1}^{k} x_i \cdot y_i$ and
            $f(x'_1, \ldots, x'_k) = \sum_{i=1}^{k} x'_i \cdot y_i$ among all $k + 1$ parties
 9:         Call secure comparison (FairplayMP [28]) to compare $f(x_1, \ldots, x_k) - f(x'_1, \ldots, x'_k)$
            and $T_2 \log \eta$ among all $k + 1$ parties
10:         if $f(x_1, \ldots, x_k) - f(x'_1, \ldots, x'_k) > T_2 \log \eta$ then
11:             $best_1\{x_1, \ldots, x_k\} \leftarrow \{x'_1, \ldots, x'_k\}$
12:         $iter_2 + +$
13:         $T_2 \leftarrow \rho_2 T_2$ (cooling down after several iterations)
14:     $route\_assigned(best\{x\}) = 1$
15:     $E_C$ gets a random neighboring route of $best\{x\}$: $x'$
        {Bottom-level simulated annealing for the optimal route assignment of $x'$}
16:     All $k + 1$ parties repeat Step 3-15 to obtain the near-optimal route assignment for $x'$:
        $best_2\{x_1, \ldots, x_k\}$ and $route\_assigned(x') = 1$
17:     Call Algorithm 1 twice to securely compute $f(best_1\{x_1, \ldots, x_k\})$ and
        $f(best_2\{x_1, \ldots, x_k\})$ among all $k + 1$ parties
18:     Call secure comparison (FairplayMP [28]) to compare $f(best_1\{x_1, \ldots, x_k\})$ -
        $f(best_2\{x_1, \ldots, x_k\})$ and $T_1 \log \eta$ among all $k + 1$ parties
19:     if $f(best_1\{x_1, \ldots, x_k\}) - f(best_2\{x_1, \ldots, x_k\}) > T_1 \log \eta$ then
20:         $E_C$ updates: $best\{x\} \leftarrow x'$; $route\_assigned(best\{x\}) = 1$; $best\{x_1, \ldots, x_k\} \leftarrow$
            $best_2\{x_1, \ldots, x_k\}$
21:     $iter_1 + +$
22:     $T_1 \leftarrow \rho_1 T_1$ (cooling down after several iterations)
23: Return $best\{x\}$ and $best\{x_1, \ldots, x_k\}$
```

- To compute the energy of any solution, all parties securely compute the scalar product using Algorithm 1, and each party holds a share of the result. To determine whether move or not, all parties implement FairplayMP [28] to securely compare the functions with their input shares. Finally, only client E_C knows the comparison result and whether move or not.
- For top and bottom-level simulated annealing, we use different sensitivity parameters: initial temperature T_1 and T_2, cooling coefficient ρ_1, ρ_2 (note that the temperature will be lowered after several iterations). We also setup different maximum number of iterations for top and bottom-level simulated annealing respectively max_iter_1 and max_iter_2.

– In order to improve efficiency, we define an indicator $route\_assigned(x) \in \{0, 1\}$ to avoid running the route assignment for every route x twice in the protocol – as a neighboring solution and the current solution respectively. If the optimal route assignment for x has been found in previous iterations, we let $route\_assigned(x) = 1$; Otherwise, 0. At Line 6, we examine the status of $route\_assigned(x)$ before going to the bottom-level simulated annealing.
– At Line 23, E_C learns only the near-optimal route $best\{x\}$ and its near-optimal route assignment $best\{x_1, \ldots, x_k\}$ (as the output of the protocol). We do not allow E_C to learn the total optimal cost during executing the protocol (before all parties contracting) because of some potential malicious inference attack slightly going beyond the semi-honest model: if the minimum cost is revealed to E_C, then E_C can use any two cities as the input to get the minimum cost (viz. the cost between two known cities, which is the corresponding shipping company's proprietary information). More severely, the minimum cost of every pair of cities and the corresponding shipping company might be inferred by E_C by repeating such malicious attack for multiple times. Although the SMC/protocol security is not violated in the above attacking scenario under semi-honest adversarial model, we still unreveal the total optimal cost for mitigating such risk.

5 Security and Cost Analysis

A formal security proof can be provided under the framework of Secure Multiparty Computation (SMC). Under the framework of SMC, a secure protocol reveals nothing in semi-honest model if all the messages received by every party can be simulated in polynomial time by knowing only the input and output of the SMC protocol [7, 8].

Theorem 1. *TSSA protocol reveals only the near-optimal traveling route $best\{x\}$ and the route assignment $best\{x_1, \ldots, x_k\}$ to client E_C in semi-honest model.*

Proof. We first look at the steps that do not need communication between different parties in the protocol. Notice that all the candidate solutions are proposed by E_C in TSSA, then most of the steps are locally implemented by E_C, e.g., finding the neighboring solution (either the traveling route or the route assignment), updating the current solution based on the comparison, and reducing the temperature in simulated annealing. These steps can be simulated by simply executing those steps.

In addition, we must simulate each party's view (all the received messages in the protocol) that requires communication in polynomial time. More specifically, client E_C and k shipping companies $P_1, \ldots, P_k$ iteratively communicate with each other in Secure Scalar Product (Algorithm 1) and Secure Comparison. We now examine the messages received by each party.

Client E_C's view: First, while calling the secure scalar product computation every time, E_C receives k encrypted random shares. k random shares are the actual messages received by E_C in those steps. W.l.o.g., E_C gets $s_i = x_i \cdot y_i + r_i$ from shipping company P_i. All the random shares generated in all iterations can be simulated by generating a random from the *uniform probability distribution over $\mathcal{F}$*, assuming that s_i is scaled to fixed precision over a closed field, enabling such a selection. Thus, $Prob[s_i = t] = Prob[r_i = t - s_i] = \frac{1}{\mathcal{F}}$, and all the shares can be simulated in polynomial time.

Second, E_C receives a series of comparison results from FairplayMP [28]. To simulate the sequence of comparison results ("$>$" or "$\leq$"), the inverse step of the simulated annealing can be utilized. Specifically, E_C starts from the near-optimal traveling route, and then finds the given neighboring solutions in sequence by running TSSA inversely (note that temperature increase can be imposed to tune the sensitivity of the moving probability in the inverse optimization). While comparing $f(x'_1, \ldots, x'_k) - f(x_1, \ldots, x_k)$ and $T_2 \log \eta$ in the bottom-level simulated annealing, if the result is "$>$", the simulator outputs an "1", otherwise "0". Now we discuss how to simulate them in polynomial time.

Recall that all the searched solutions are known to E_C in sequence, but the energy of any state (which is the sum of the local random shares) is unknown to E_C since E_C does not know the cost vectors from every shipping company. Fortunately, since E_C knows its final traveling route $best\{x\}$, we can use the same simulator in [10] to simulate a cost function in polynomial time. Then, the energy of two compared states (solutions) can be polynomially simulated as well simply because both solutions are regarded as the input for E_C (E_C proposes the candidate solutions). Consequently, we can simulate "1" or "0" for two reasons: 1) the probability of generating each of them is deterministic with Equation 1, and 2) another parameter η is uniformly distributed in $[0, 1)$. Therefore, a sequence of such comparison results in E_C's view can be simulated in polynomial time. Similarly, the sequence of comparison results for top-level simulated annealing can be simulated with the same polynomial machine.

In summary, applying the Composition Theorem [8], client E_C learns only the near-optimal traveling route $best\{x\}$ and the route assignment $best\{x_1, \ldots, x_k\}$.

Shipping Company $\forall i \in [1, k], P_i$*'s View*: In the protocol, every shipping company only receives the random shares in secure scalar product computation and E_C's public key pk. As analyzed above, the random shares can be simulated in polynomial time using the same machine as E_C's random share. Therefore, applying the Composition Theorem [8], every shipping company only learns the public key pk in the protocol. This completes the proof.

Besides the protocol security guaranteed by the SMC theory, it would be useful if we can simultaneously resolve the inferences from the messages received before and after the move. Coincidentally, simulated annealing provides excellent mechanism to naturally mitigate such inference attack. Specifically, unlike many other meta-heuristics, simulated annealing runs probabilistically and allows moving from the current solution to a *worse neighboring solution*. Due to the above uncertainty and the unrevealed overall cost in any solution, it is difficult for E_C to infer any private information, e.g., which solution outperforms its neighboring solution. Thus, the inference attack (which actually does not compromise the SMC protocol security) could be mitigated.

Cost Analysis: Given a k-RTSP with n cities, the maximum number of iterations of the top and bottom level simulated annealing is given as max_iter_1 and max_iter_2 respectively. For simplicity of notation, we denote them as $O(m)$. We now discuss the communication and computation cost required in the TSSA protocol.

Communication Cost. In TSSA protocol, only secure scalar product computation and secure comparison request multiparty communication. First, while calling the secure scalar product computation, it needs one round communication between E_C and each shipping company. Then, the communication cost of total secure scalar product computation is $O(2m^2 * k * n(n-1)/2) = O(m^2n^2k)$ messages of bit communication. Second, the number of communication messages in every secure comparison is equal to the number of computing parties [29]. Then, the communication cost of total secure comparison is $O(m(2m+1)k) = O(m^2k)$ messages of bit communication. Moreover, public key pk is delivered from E_C to all k shipping companies (every party can use the same public key in all iterations, thus pk can be considered as offline cost). Therefore, the communication complexity of the protocol is $O(m^2n^2k)$.

Computation Cost. First, E_C locally finds the neighboring solution, moves the solution and updates the temperature in both top and bottom-level simulated annealing. The computation cost of the above process is ignorable compared to cryptographic work. Second, if we estimate the runtime for a single secure scalar product computation and a single secure comparison as t_s and t_c respectively, the total computation cost based on cryptography can be written as $(2m^2 * k) * t_s + m(2m+1) * k * t_c \approx 2m^2k(t_s + t_c)$.

6 Experimental Validation

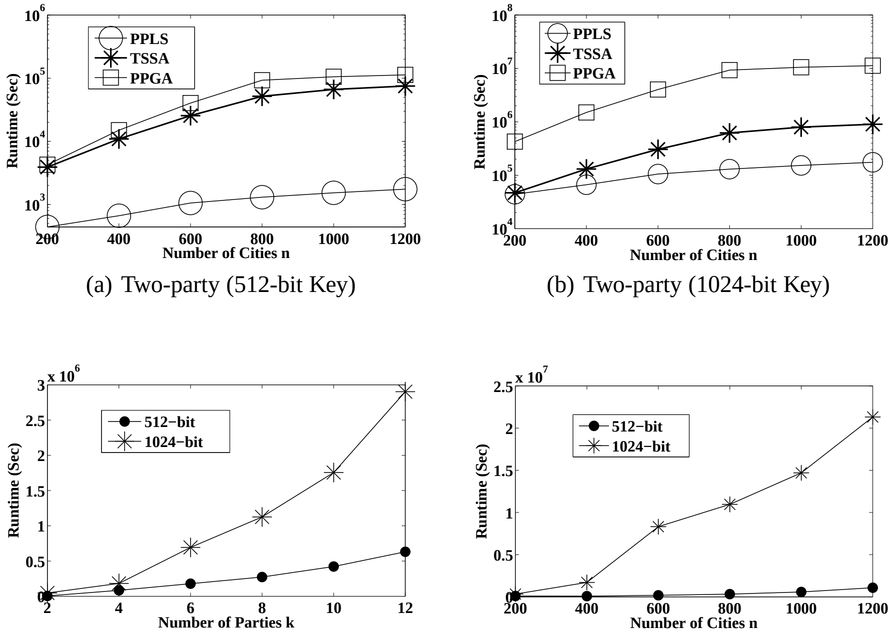
We first present the experimental setting and then discuss the results.

Datasets. We conduct experiments on four real datasets collected from National Imagery and Mapping Agency [6]. Each dataset is derived from a country (Canada, Japan, Italy and China), where the cost of travel between cities is the Euclidean distance computed from the coordinates. We randomly select six sets of cities from each of the four datasets with the size (number of cities n) $200, 400, 600, 800, 1000$ and 1200. This matches the experimental setup in [1], which makes it comparable for the two-party case. Note that we repeat every experiment 10 times by changing the initial city in every test, and average all the results returned from all 10 experiments running on each of the 4 different datasets.

Problem Setup. To formulate k-RTSP, we need to set the costs for the k parties. To do this, we generate $k-1$ noise values by sampling from the gaussian distribution with mean 0 and an appropriate variance (1/3 of the cost). Then, we obtain k different costs for every pair of cities (the original plus the $k-1$ noise added values) and randomly assign them to the k different parties, where k is selected as $2, 4, 6, 8, 10$ and 12.

Meta-heuristics Setup. We initialize the temperature for two-level simulated annealing and the cooling parameters as $T_1 = 1000, \rho_1 = 50\%$ and $T_2 = 1000, \rho_2 = 20\%$. If no better solution can be found in 50 iterations, we apply cooling. If no better solution can be found in 20 times cooling, we terminate the meta-heuristics (note that the termination criteria can be alternatively established as a maximum threshold for the total number of iterations).

Cost Estimation of Large Scale Input. For small and medium scale input, we can directly capture the computation cost. Since our protocol iteratively utilizes the cryptographic building blocks, we can estimate the computation cost for large scale input.



(a) Two-party (512-bit Key)

(b) Two-party (1024-bit Key)

(c) Varying k (n=600)

(d) Varying n (k=6)

Fig. 3. Computation Cost Evaluation

The runtime can be obtained through multiplying the cost of running a unit building block by the overall required count. We do this for the tests with runtime more than 10^4 seconds (approx. 3 hours).

6.1 Computation Cost Comparison (Two-party)

Under the same experimental setup for the two party TSP, we compare the computation cost of our two-level secure simulated annealing (TSSA) with Sakuma and Kobayashi's two algorithms – Privacy-preserving local search (PPLS) and Privacy-preserving genetic algorithm (PPGA) [1]. Since only two parties are involved, the bottom-level simulated annealing would not be triggered in TSSA.

We set the key length in Paillier's Homomorphic cryptosystem [26] as 512 bits and 1024 bits, and the size of the circuit (number of gates) in FairplayMP [28] as 512 (a benchmark setup). Figure 3(a) and 3(b) show the runtime of TSSA, PPLS and PPGA for the key length of 512 bits and 1024 bits respectively. As expected, the computation cost of TSSA lie between that of PPLS and PPGA. This is consistent with the fact that 2-Opt local search performs efficiently yet produce the worst result [30, 28]. More specifically, when the number of cities n increases from 200 to 1200, with 512-bit key, TSSA consumes approximately 1.5 hour – 20 hours; with 1024-bit key, TSSA spends approximately 20 hours – 5 days. The computation cost increases polynomially in both scenarios.

6.2 Results of k-RTSP

In the multiparty case, we evaluate the performance of solving k-RTSP with TSSA in two facts: the efficiency of the solver and the quality of k-RTSP's optimal solution. Efficiency can be reflected as the computation cost (viz. runtime) of the algorithm. Figure 3(c) and 3(d) demonstrate the scalability of our TSSA algorithm. The runtime grows polynomially as the number of shipping companies k or the number of cities n increases. Since TSSA is provably secure for all $k + 1$ parties, the most expensive runtime shown in the figure (1200 cities, key length of 1024-bit, $k = 6$) is still tolerable (around 10 days).

On the other hand, we define two measures to evaluate the quality of k-RTSP's optimal solutions: "Cost Reduction Ratio" $= 1 - \frac{c}{c_{\max}}$ and "Heuristic Error" $= \frac{c}{c_{\min}} - 1$. Given a specific traveling route to visit all cities, $c_{\max}$ denotes the maximum possible cost (the highest cost is chosen between every pair of consecutively visited cities on the route), c is the optimal solution derived from TSSA algorithm and $c_{\min}$ denotes the minimum possible cost (the lowest cost is chosen between every pair of consecutively visited cities on the route).

Table 3. Quality of Near-optimal Solutions

(a) Cost Reduction Ratio

Inputs	$k = 2$	$k = 4$	$k = 6$	$k = 8$	$k = 10$
$n = 200$	0.272	0.256	0.245	0.23	0.211
$n = 400$	0.301	0.292	0.285	0.269	0.241
$n = 600$	0.324	0.318	0.312	0.303	0.293
$n = 800$	0.339	0.335	0.328	0.32	0.317
$n = 1000$	0.352	0.344	0.339	0.336	0.332
$n = 1200$	0.361	0.353	0.348	0.344	0.339

(b) Heuristic Error

Inputs	$k = 2$	$k = 4$	$k = 6$	$k = 8$	$k = 10$
$n = 200$	0.025	0.032	0.041	0.063	0.075
$n = 400$	0.038	0.047	0.054	0.073	0.084
$n = 600$	0.052	0.066	0.085	0.094	0.111
$n = 800$	0.081	0.085	0.116	0.128	0.137
$n = 1000$	0.127	0.142	0.154	0.164	0.183
$n = 1200$	0.161	0.171	0.18	0.192	0.199

TSSA protocol gives good results for k-RTSP if $CostReRatio$ is high and $Error$ is low. To validate this on large-scale inputs, we simplify TSSA by running it without cryptographic work, which does not distort the returned near-optimal solutions. Table 3 present the "Cost Reduction Ratio" and "Heuristic Error", where $k = 2, \ldots, 10$ and $n = 200, \ldots, 1200$. First, the cost reduction ratio increases with the increase of the problem size. This fact is true because, when the client needs to visit more cities, the maximum possible cost becomes extremely high, then great saving can be realized. With the growth of k, maximum possible cost would be slightly higher for the same k-RTSP, however the heuristic cannot always find the route assignment for every pair of cities. Then, cost reduction ratio decreases as k goes large. Second, the heuristic error represents the difference between the near-optimal solution obtained by TSSA and the minimum possible cost. The data in Table 3(b) implies that the error grows while enlarging either k or n. This is the nature of heuristics.

Finally, cost reduction ratio between 0.211 and 0.361 shows a considerable saving in delivery, and the heuristic error (which primarily reflects the error produced in bottom-level simulated annealing) between 0.25 and 0.199 shows the effectiveness of our TSSA meta-heuristic algorithm.

7 Conclusion and Future Work

In this paper, we formulate and study the privacy issues in the cost-reducible collaborative TSP where the global cost can be further reduced by allowing multiple companies to collaboratively provide a shared solution. We then proposed an effective privacy-preserving approach (TSSA protocol) to securely derive the near-optimal solution within a reasonable time. We formally proved the security of the TSSA protocol and validated the efficiency as well as the quality of the optimal solution with real data. In the future, we will further mitigate the inference attack in addition to the SMC protocol security by proposing schemes that require multiple parties to jointly propose neighboring solutions (like [4]), and also improve the efficiency of the communication protocol by either developing secure solvers based on the recent cryptographic tools (e.g., Sharemind [31], PICCO [32]) or incorporating problem transformation into the algorithm without compromising security. We also intend to generalize the simulated annealing based communication protocol to solve more NP-hard problems with limited information disclosure.

References

1. J. Sakuma and S. Kobayashi, “A genetic algorithm for privacy preserving combinatorial optimization,” in *GECCO*, pp. 1372–1379, 2007.
2. C. Clifton, A. Iyer, R. Cho, W. Jiang, M. Kantarcioglu, and J. Vaidya, “An approach to identifying beneficial collaboration securely in decentralized logistics systems,” *Management & Service Operations Management*, vol. 10, no. 1, 2008.
3. Y. Hong, J. Vaidya, and H. Lu, “Secure and efficient distributed linear programming,” *Journal of Computer Security*, vol. 20, no. 5, pp. 583–634, 2012.
4. Y. Hong, J. Vaidya, H. Lu, and B. Shafiq, “Privacy-preserving tabu search for distributed graph coloring,” in *SocialCom/PASSAT*, pp. 951–958, 2011.
5. Y. Hong, J. Vaidya, and S. Wang, “A survey of privacy-aware supply chain collaboration: From theory to applications,” *Journal of Information Systems*, To Appear, 2014.
6. “http://www.math.uwaterloo.ca/tsp/,”
7. A. C. Yao, “How to generate and exchange secrets,” in *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, (Los Alamitos, CA, USA), pp. 162–167, IEEE, IEEE Computer Society, 1986.
8. O. Goldreich, *The Foundations of Cryptography*, vol. 2, ch. Encryption Schemes. Cambridge University Press, 2004.
9. J. Li and M. J. Atallah, “Secure and private collaborative linear programming,” in *Proceedings of the 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pp. 1 – 8, Nov.17–20 2006.
10. J. Vaidya, “A secure revised simplex algorithm for privacy-preserving linear programming,” in *AINA ’09: Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications*, 2009.
11. O. Catrina and S. de Hoogh, “Secure multiparty linear programming using fixed-point arithmetic,” in *ESORICS*, pp. 134–150, 2010.

12. W. Du, *A Study of Several Specific Secure Two-party Computation Problems*. PhD thesis, Purdue University, West Lafayette, Indiana, 2001.
13. J. Vaidya, "Privacy-preserving linear programming," in *SAC*, pp. 2002–2007, 2009.
14. A. Bednarz, N. Bean, and M. Roughan, "Hiccups on the road to privacy-preserving linear programming," in *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, WPES '09, (New York, NY, USA), pp. 117–120, ACM, 2009.
15. A. Bednarz, *Methods for Two-party Privacy-preserving Linear Programming*. PhD thesis, The University of Adelaide, Adelaide, Australia, 2012.
16. Y. Hong, J. Vaidya, and H. Lu, "Efficient distributed linear programming with limited disclosure," in *DBSec*, pp. 170–185, 2011.
17. O. L. Mangasarian, "Privacy-preserving horizontally partitioned linear programs," *Optimization Letters*, vol. 6, no. 3, pp. 431–436, 2012.
18. O. L. Mangasarian, "Privacy-preserving linear programming," *Optimization Letters*, vol. 5, no. 1, pp. 165–172, 2011.
19. W. Li, H. Li, and C. Deng, "Privacy-preserving horizontally partitioned linear programs with inequality constraints," *Optimization Letters*, vol. 7, no. 1, pp. 137–144, 2013.
20. Y. Hong and J. Vaidya, "An inference-proof approach to privacy-preserving horizontally partitioned linear programs," *Optimization Letters*, vol. 8, no. 1, pp. 267–277, 2014.
21. O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game - a completeness theorem for protocols with honest majority," in *Proceedings of the 19th ACM Symposium on the Theory of Computing*, (New York, NY), pp. 218–229, ACM, 1987.
22. M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *In Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pp. 1–10, 1998.
23. C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.
24. S. Kirkpatrick, C. D. Gelatt, and M. P. V. Jr., "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
25. G. A. Croes, "A method for solving traveling salesman problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, 1958.
26. P. Paillier, "Public key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592*, pp. 223–238, 1999.
27. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, p. 1087, 1953.
28. A. Ben-David, N. Nisan, and B. Pinkas, "Fairplaymp: a system for secure multi-party computation," in *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, (New York, NY, USA), pp. 257–266, ACM, 2008.
29. I. Ioannidis and A. Grama, "An efficient protocol for yao's millionaires' problem," in *Hawaii International Conference on System Sciences (HICSS-36)*, (Waikoloa Village, Hawaii), pp. 205–210, Jan. 6-9 2003.
30. B.-I. Kim, J. lk Shim, and M. Zhang, "Comparison of tsp algorithms," *Project for Models in Facilities Planning and Materials Handling*, 1998.
31. D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *ESORICS*, pp. 192–206, 2008.
32. Y. Zhang, A. Steele, and M. Blanton, "Picco: a general-purpose compiler for private distributed computation," in *ACM Conference on Computer and Communications Security*, pp. 813–826, 2013.