An Approach to Identifying Beneficial Collaboration Securely in Decentralized Logistics Systems

by Chris Clifton
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

An Approach to Identifying Beneficial Collaboration Securely in Decentralized Logistics Systems

Chris Clifton • Ananth V. Iyer • Richard Cho • Wei Jiang •
Murat Kantarcıoğlu • Jaideep Vaidya

Krannert School of Management and Department of Computer Sciences, Purdue University, 403 West State Street, West Lafayette, Indiana 47907-2056

clifton@cs.purdue.edu • aiyer@mgmt.purdue.edu • rcho@gte.net • wjiang@cs.purdue.edu • kanmurat@cs.purdue.edu • jsvaidya@rbs.rutgers.edu

The problem of sharing manufacturing, inventory or capacity to improve performance is applicable in many decentralized operational contexts. However, solution of such problems commonly requires an intermediary or a broker to manage information security concerns of individual participants. Our goal is to examine use of cryptographic techniques to attain the same result without the use of a broker. To illustrate this approach, we focus on a problem faced by independent trucking companies that have separate pickup and delivery tasks and wish to identify potential efficiency enhancing task swaps while limiting the information the companies must reveal to identify these swaps. We present an algorithm that finds opportunities to swap loads without revealing any information except the loads swapped, along with proofs of the security of the protocol. We also show that it is incentive compatible for each company to both follow the protocol correctly as well as provide their true data. We apply this algorithm to an empirical dataset from a large transportation company and present results that suggest significant opportunities to improve efficiency through Pareto improving swaps. This paper uses cryptographic arguments in an operations management problem context to show how an algorithm can be proven incentive compatible as well as demonstrate the potential value of its use on an empirical dataset.

1. Introduction

Coordination of decisions and sharing capacity across independent decision makers can provide significant cost reductions in many operations management problem contexts. In decentralized environments, there are numerous instances where individual companies would like to swap tasks or loads to gain operational efficiencies. Keskinocak & Tayur (2001) suggest that e-marketplaces can play the role of central coordinators. They suggest, for example, a marketplace where customers post orders for paper products and manufacturers offer capacity that is shared across multiple orders to generate efficiency. They also describe an example where independent manufacturers who have production capability for a variety of products with associated setups swap customer demands to reduce setups. Another example discussed involves intermediaries that provide consolidation and capacity trading opportunities for shippers and carriers. Similarly, Kalagnanam, Trumbo & Lee (2000) describe a problem faced by steel companies that have to deal with surplus inventories of steel coils with specific properties. They provide an optimization model that can be solved by a coordinator who attempts to optimally allocate the surplus inventory. Keskinocak & Tayur (2001) identify that a key requirement for an intermediary is that it will have to be an independent, trustworthy entity who will keep all information confidential and who will create benefits that are equitably distributed.

The examples above suggest that one reason for use of a broker or a marketplace is that it prevents disclosure of proprietary information across competitors. However, this information must still be revealed to the broker in order for coordination to occur. In addition, the broker will need to cover his costs by charging a margin, the data collected by the broker has to be secured from leakage and the broker's incentives have to be aligned with individual company goals. In many contexts, independent companies are reluctant to share proprietary information, unless absolutely necessary, in order to protect their customer base. However, companies do realize that some sharing of data may be necessary to compensate for market fragmentation. In such contexts, it would be ideal to devise a system where all of the data are available for use but only data that is absolutely is shared. required for coordination.

The cryptographic community has shown that a trusted third party is not required – it is possible to compute functions without disclosing private data to any party (Yao 1986, Goldreich, Micali & Wigderson 1987). The result is that no party learns more than they would if a broker arranged the transactions, *and no broker is required.* Our goal is to apply state of the art techniques from data encryption to logistics problems to automate the task performed by the broker. Companies learn no more than with an honest broker. However, the broker is eliminated. In fact, for the specific problem and solution given in this paper, we prove that no party learns more than the minimum needed to accomplish the desired efficiency gains. The benefit to shipping companies and shippers is the ability to reduce collaboration costs and improve the efficiency of the overall system.

One reason for this research is that we believe that an examination of operations problems with their specific contexts can provide opportunities for development of algorithms that exploit problem structure. In addition, by focusing on algorithms that are embedded in commercial codes, we suggest that there is potential for rapid practical adoption. Finally, there are conceptual issues regarding data leakage vs. efficiency that we suggest as a rich research area, both in operations management as well as in the computer security research community.

1.1 A Specific Problem Context – The Trucking Industry

In order to provide a complete treatment of our approach, we examine a specific problem context involving truck routing. Truck transport is a \$462 billion industry in the US (Wilson & Delaney 2003). However, the industry is extremely fragmented with the largest company accounting for less than 5% of the market. The main source of inefficiency in this industry is the "deadhead" miles or miles driven empty. The primary reason for this inefficiency is the spatial nature of this industry, i.e., for a truck to pickup a load, it physically has to be at that location. When the truck is done, it ends up at the physical drop off point and may have to travel to a new location to be useful. Intuitively, if transport companies swap some of their loads, there is the potential for Pareto improving savings (i.e., neither company faces a higher cost and at least one company faces a lower cost).

However, attempts to collaborate and thus swap loads to get more efficient routes are often discouraged by a desire of individual companies to "share only if beneficial". In addition, legal restrictions, dealing with anti-trust issues, frown upon information sharing and collaboration that can be potentially anti-competitive. However, even anti-trust considerations do permit competing firms to engage in limited information sharing and collaboration that is clearly efficiency enhancing. As reported in the Wall Street Journal, "It is permissible for carriers to cooperate in certain ways. For instance, if two of them both carry chemicals for a given producer on the same route, they may pool their capacity for the purpose of operational efficiency. ... But cooperating to divide up markets or to affect prices would clearly fall outside these permitted arrangements" (Bandler 2003).

If transport companies resort to using a broker to swap loads, the first step is for each company to independently identify loads it would like to swap. These potential loads are provided to a broker who now sees all available loads. As a result, parties will only make things available that they view as likely to be picked up in a swap. *The key difference in our approach is that all available loads are provided by all companies. The algorithm is executed in a distributed manner by each company and at no time is the data entrusted to any third party. In addition, the secure protocols used in the algorithm guarantee that no information other than the swapped loads that improve efficiency are revealed to each company, and all such efficiency-improving swaps are made.* This results in both lower information disclosure and higher efficiency than a traditional broker-mediated model.

Why would companies want such an approach? One motivation comes from the transport companies themselves as part of their desire to protect proprietary information while achieving maximum efficiency. However, another motivation may come from shippers who could demand such a protocol be used by their associated carriers to ensure that efficiency is enhanced while preventing any collusion regarding data that is not explicitly required to be shared and that may reduce competitiveness of the carrier market.

Thus, in this paper, we do the following:

1. We provide an algorithm that ensures that sharing takes place only if each company sees its costs reduced and that the sharing scheme ensures that all potential players can engage to identify cost reducing swaps, while ensuring no information is shared other than what can be concluded from the final swapped points.

2. Show how incentive compatibility can be proven; by demonstrating on this algorithm that honesty on the part of the collaborators is ensured by guaranteeing that either it can be detected that one participant is cheating (and thus gets thrown out of future collaborations) or that the cheating is not incentive compatible, i.e., the cheater is worse off.

3. We show that the algorithm, implemented in a decentralized manner, affords the globally optimal split of loads for a specific setting.

4. We apply the algorithm proposed to an empirical dataset from a transportation company that provided us with 11 weeks of pickup and delivery data. While the algorithm proposed is a heuristic in the context of vehicle routing, it uses the state of the art techniques in data encryption and secure multi-party computation techniques that *guarantee* that the security requirements are met. The empirical data suggests the delivered value of the algorithm. Empirical results suggest the potential to reduce costs by over 15% based on application of the algorithm.

One goal of this paper is to provide an application of cryptographic techniques as an enabler of operational efficiency in a decentralized ownership environment. We also suggest the associated methodological issues as a potentially fruitful area of research in operations management. In addition, several combinatorial problems can be solved by using the space filling curve as a heuristic. For such problem contexts, our paper provides a template to generate secure implementations.

The next section gives a formal treatment of the algorithm, along with proofs that it achieves a one-dimensionally optimal result, that nothing is disclosed that is not obvious from the result, and that cheating is not incentive compatible i.e., it is detrimental to the cheater. In Section 2.5, we show that the algorithm can be securely used among multiple parties. In Section 3, we provide results when this approach is applied to a set of real shipping transactions. In the Appendix, we provide notation and formal proofs of the theorems in the text.

2. Problem Description

Formally, the general problem is as follows. We have N independent transport companies with company i having m_i points located in a two dimensional plane that must be served by a truck. The goal is to identify a sequence of enquiries and swaps between pairs of companies that result in:

1. a set of points that when swapped between the companies guarantees that no company is worse off,

2. no information is shared other than what can be concluded from the final swapped points,

3. the algorithm is polynomial in running time, and

4. any cheating by either party during the execution of the algorithm is either detected by the other party, or results in a less efficient solution for the dishonest party, thus providing the incentive to truthfully follow the algorithm.

2.1 Using SpaceFilling Curves

We first map the initial two dimensional problem to one dimension using a space filling curve. Figure 1(a) shows the existing routes for two parties, Figures 1(b) and 1(c) present a space transformation process via a Hilbert spacefilling curve. The basic idea of using a space filling curve to develop heuristics for combinatorial problems is described in Bartholdi III & Platzman (1988). Since it is a one to one mapping from two dimensions to one dimension, points identified for swap in one dimension provide a unique pointer to the corresponding original point location in two dimensions. Bartholdi and Platzman show that using the space filling curve to develop heuristics results in the following properties: (a) 25% worse than optimal worst case performance for planar traveling salesman problems (Bartholdi III & Platzman 1982), (b) solutions within one second that have a gap of less than 34% more than the best approach for large problems using 2 months of computing time (see http://www.isye.gatech.edu/~jjb/mow/mow.html), and (c) implemented logistics packages such as the ARC/Info Geographical Information System, the CAPS Logistics Toolkit of Baan Systems, and other commercial systems managing 2-dimensional data (see http://www.isye.gatech.edu/~jjb/mow/mow.html).[1]

Why map to one dimension? In general, the vehicle routing problem is an NP-hard optimization problem, even without worrying about privacy/security. While proper choice of heuristics may give good solutions directly on the two dimensional problem, choosing those heuristics requires an understanding of the characteristics of the data – and sharing this information violates goal 2. However, in one dimension the optimal solution is tractable: search for the best solution requires a logarithmic number of steps.

2.2 A Swap Algorithm

The goal of the algorithm is to identify points that can be swapped between two parties, whose locations are on a line, that results in (a) both parties decreasing their costs and (b) finding the maximum number of swaps that will optimize the total distance traveled. Note that in one dimension, given the number of points to swap, the specific points that will guarantee the lowest cost is apparent i.e., swap the farthest points for each location. We begin with the assumption that each party has selected an end point. This choice may be influenced by their current hub locations – a natural approach would be to draw a space-filling curve with endpoints at the hubs. A list of symbols and notation is provided in Appendix A.1.

Let the k^{th} extreme point $Extreme\_POS(X, d_x, k)$ refer to the k^{th} farthest point from the endpoint d_x in the points X. Given a number of points k, the parties compare their k^{th} extreme points. If the locations do not cross, k is a lower bound on the number of points to swap to decrease costs and the parties try $2k$. Once the locations cross, k is an upper bound, and the search continues between the upper and lower bounds until the optimal number of points to swap is found. A formal statement of the algorithm is provided as Algorithm 1. (Note that Aggarwal, Mishra & Pinkas (2004) present a protocol for securely computing the

[1]Note that our approach will be to incorporate the encryption processes within this heuristic.

k^{th}-ranked element among multiple parties. The problem is different, as we do not know k in advance; the distinctions are discussed further in Appendix B.4.)

Algorithm 1 OROD: One-Dimensional Relative Outlier Detection

```
Require: $S^1$, $O^1$,
1: $lbound \leftarrow 0$
2: $ubound \leftarrow \infty$
3: $i \leftarrow 1$
4: {Lines 5 through 13 determines the maximal size of $i$}
5: while $(ubound - lbound > 1)$ do
6:     if $Extreme\_POS(S^1, l, i) > Extreme\_POS(O^1, r, i)$ {Defined in Appendix A.2.} then
7:         $lbound \leftarrow i$
8:     else
9:         $ubound \leftarrow i$
10:    end if
11:    $i \leftarrow min\left(i * 2, \left\lfloor \frac{lbound+ubound}{2} \right\rfloor\right)$
12: end while
13: return $i$
```

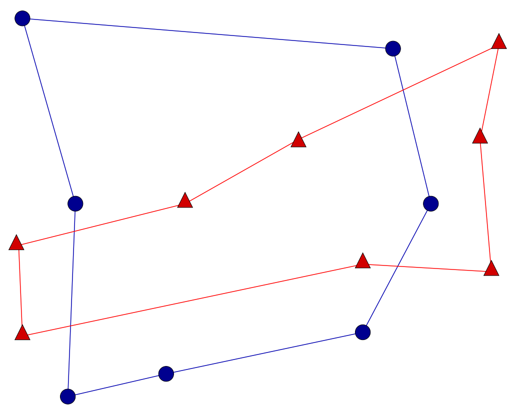
2.2.1 Illustration of OROD Execution

We now provide an example to illustrate the algorithm. Let $\bigcirc$ and $\triangle$ be two parties, with original routes shown in Figure 1(a). Figures 1(b) and 1(c) present a space transformation process via a Hilbert curve. Let $\bigcirc$ start from the left end and $\triangle$ start from the right end. We now show the execution of the algorithm from $\bigcirc$'s point of view.

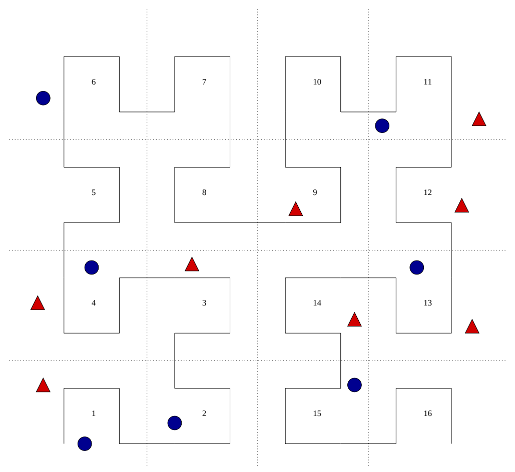
Initially, $lbound \leftarrow 0$, $ubound \leftarrow \infty$ and $i \leftarrow 1$. Since the difference between $lbound$ and $ubound$ is greater than 1, the execution enters the *while* loop. $Extreme\_POS(\bigcirc, left, 1)$ returns 15, which is greater than the index returned from $Extreme\_POS(\triangle, right, 1) = 1$, so $lbound \leftarrow 1$ and $i \leftarrow 2$. In the next iteration, $Extreme\_POS(\bigcirc, left, 2) = 13$ is still greater $Extreme\_POS(\triangle, right, 2) = 3$, so $lbound \leftarrow 2$ and $i \leftarrow 4$. $Extreme\_POS(\bigcirc, left, 4) = 6 < Extreme\_POS(\triangle, right, 4) = 9$, so $ubound \leftarrow 4$ and $i \leftarrow 3$. $Extreme\_POS(\bigcirc, left, 3) = 11 > Extreme\_POS(\triangle, right, 3) = 4$, so $lbound \leftarrow 3$ and $i \leftarrow 3$. since $ubound - lbound = 1$, the execution exits with 3 as the number of points to be swapped. Thus $\bigcirc$ swaps its (least desired) points in locations 11, 13 and 15 for $\triangle$'s points in 1, 3 and 4. The line is thus partitioned into two parts, each assigned to one end. Given the one to one mapping preserved by the spacefilling curve, the corresponding three swapped points can be identified in the original two dimensions. The resulting tours are shown in Figure 1(d).

Theorem 1. *The OROD algorithm terminates with the optimal number of points swapped, i.e., after swapping, there are no two points that could be swapped to give a lower tour length.*

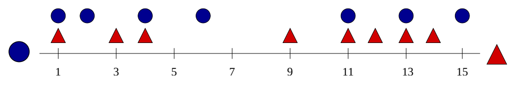
Proof: See Appendix C.



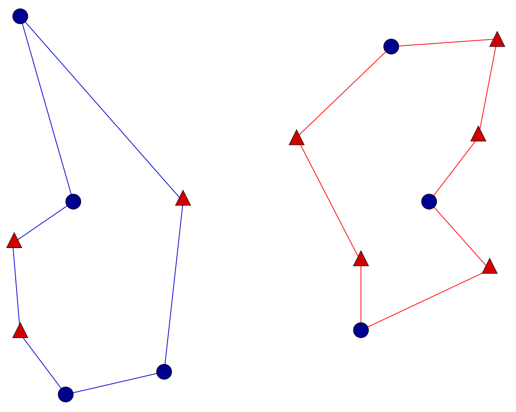
(a)



(b)



(c)



(d)

Figure 1: Execution of OROD protocol between two parties

2.3 Secure Execution

We next focus on secure execution of the algorithm OROD. The goal in this section is to ensure that no "excess" information is revealed by either party, other than what can be deduced from the final swaps. Since Algorithm 1 only exchanges information in step 6, we want to perform the comparison of the *Extreme_POS* functions from both parties without disclosing anything except the comparison result. As we shall show, the comparison results at each step can be deduced from the final swaps, so the result of the comparison does not disclose excess information.

The idea of the secure version of the algorithm is to provide a mechanism to check if it is okay to swap a certain number of points, without either side learning anything except if that many points will be a beneficial swap i.e., parties do not learn where the points are.[2]

To demonstrate this, we start with the view in Figure 1(c), so the reader (but not △) can see both sides. ◯ builds a set of boxes, each containing either $>$ or $\leq$. The boxes correspond to positions on a line, and a box entry at a specific position has a $>$ if ◯ will benefit from swapping the given number of points (indicated by i) if every one it receives is at or to the left of that position. Figure 2 shows the values of the boxes for checking if swapping a single point is okay. If ◯ receives points to the left of position 15, it benefits (since it gives up a point at 15.) However, if it were to receive a point at position 15 or 16, it would be better not to swap. If it were to swap one point, △ would give up its point at position 1. It therefore opens the box at position 1 (its leftmost point) - since this is $>$, ◯ (and △) will gain from swapping one point.

This process is repeated to find the right number of points to swap. Figure 3 shows the test for $i = 2$; △ looks at position 3 (where its second extreme point is located) and finds it is okay to swap two points. Figure 4 shows $i = 4$; looking at position 9, △ finds that four points given by ◯ would include at least one to the left of this point, so it isn't a beneficial swap. All that is now left is testing a swap of 3 points - this is shown in Figure 5. Since this is okay ($>$), and four points does not work, the final result is to swap three points.

The key to the security of the process is that ◯ does not know which box △ opens, and △ only learns the value in one box. (This is accomplished through a cryptographic protocol described in Section 2.3.1.) Thus, the only thing learned from the protocol is the value "$>$".

For this example, secure implementation implies that since the final result (Figure 1(d)) shows △ swapping three points, knowing the final result both parties could conclude that swapping one point is okay. While they have learned something new *at this point*, nothing is learned from this protocol that ◯ and △ would not learn from giving all of their data to an honest broker (proved later). This is accomplished without the need for an honest broker. For example, from what △ sees during the execution of the protocol (the shaded boxes only), △ knows that ◯ has fewer than four points to the right of position 9, and at least three to the right of position four. But these would be obvious even with an honest broker: △ learns of the ◯s at 11, 13, and 15 from the swap, and knows that there cannot be another

[2]We describe this using direct *oblivious transfer*; however, the actual implementation would use a secure comparison approach with constant complexity (Yao 1986) rather than the linear complexity of the method described.

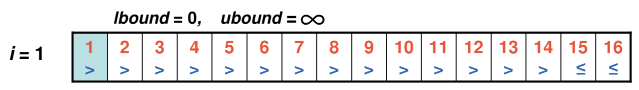


Figure 2: Example of execution, iteration 1

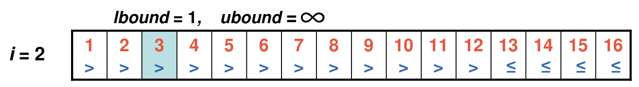


Figure 3: Example of execution, iteration 2

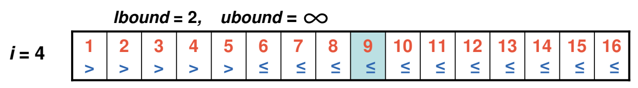


Figure 4: Example of execution, iteration 3

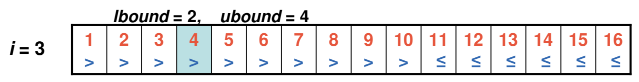


Figure 5: Example of execution, iteration 4

◯ to the right of 9 or it would have been swapped as well. Thus, the algorithm ensures that sharing of beneficial information is identified by ◯ and △ with no other information being revealed in the process.

How do we verify secure execution of OROD formally? Section 2.3.1 provides details.

2.3.1 Securely Opening the Box

We now describe an approach to make sure that only one of the N electronic "boxes" offered by ◯ (in the algorithm OROD) is opened by △, and that ◯ cannot learn which box is opened. The cryptography community refers to this problem as 1 *out of N oblivious transfer*; it has been the subject of extensive research. Here, we will describe a simple 1 out of N oblivious transfer (OT_1^N) protocol from Naor & Pinkas (1999) and Naor & Pinkas (2001).

For simplicity, we first describe 1 out of 2 oblivious transfer (OT_1^2). In this problem, ◯ has two electronic boxes (labeled 0 and 1); △ wants to open the box σ ($\sigma = 0$ or 1) without revealing to ◯ which box was opened. In the following protocol, let σ be the index of the box that △ wants to open, and let B_0 and B_1 be the contents of boxes 0 and 1 respectively. (For simplicity, we show the contents as a single bit; B_0 and B_1 take values 0 or 1).

The protocol is described below:

1. ◯ generates and publishes three numbers:

   C**:** a random number between $1, \ldots, p-1$,

   p**:** a large prime number, and

   g**:** the generator of p's multiplicative group, i.e., every number between 1 and $p-1$ can be written as $g^k \bmod p$ for some k between $1, \ldots, p-1$.

2. △ picks a random number k between $1, \ldots, p-1$, sets $P_\sigma = g^k \mod p$ and $P_{1-\sigma} = (C/P_\sigma) = C{*}{P_\sigma}^{-1} \mod p$ where ${P_\sigma}^{-1}$ is the multiplicative inverse of P_σ (i.e ${P_\sigma}^{-1}{*}P_\sigma = 1 \mod p$) and sends P_0 to ◯.

3. ◯ finds P_1 by evaluating $C/P_0 \bmod p$, creates $E_0 = (g^{r_0} \bmod p, M_0 = (H((P_0)^{r_0} \bmod p) \oplus B_0))$, $E_1 = (g^{r_1} \bmod p, M_1 = (H((P_1)^{r_1} \mod p) \oplus B_1))$, by randomly choosing r_0, r_1 between $1, \ldots, p-1$, and sends E_0, E_1 to △. Note that E_0, E_1 are both pairs of numbers. The hash function H is described in Appendix B.2.

4. △ computes $T_\sigma = H((P_\sigma)^{r_\sigma} \mod p) = H((g^{r_\sigma})^k \mod p)$ and gets B_σ by computing $T_\sigma \oplus M_\sigma$. It cannot compute the other value due to the cryptographic assumptions outlined in Appendix B.2.

In the above protocol the choice of △ (σ) is not revealed because all ◯ receives is either $g^k \bmod p$ or $C/g^k \bmod p$, where k is chosen randomly. Since operations are done in mod p, both $g^k \bmod p$ or $C/g^k \bmod p$ values are uniformly distributed between $0, \ldots, p-1$. Therefore, ◯ does not see anything more than a random number. △ learns nothing by receiving the random C, or (because of the random oracle hash function) from inspecting E_0 or E_1.

While $\triangle$ can decrypt E_σ to obtain the final result, by the original Diffie-Hellman assumption (see Appendix B.2) it cannot decrypt the other box. If $\triangle$ could decrypt both E_0 and E_1, it means that it knows a efficient way to find $(P_{1-\sigma})^{r_{1-\sigma}} = g^{k' r_{1-\sigma}} \bmod p$ (Note that preceding value is needed to open the other box.) given $P_{1-\sigma} = g^{k'} \bmod p$ (since g is a generator we can write $P_{1-\sigma} = g^{k'} \bmod p$ for some k') and $g^{r_{1-\sigma}} \bmod p$, this violates the Diffie-Hellman assumption.

To clarify the above protocol, we give a simple example.[3]

Example 1. *Let the shared hash function H be $H(0) = 1, H(1) = 0, H(2) = 1, H(3) = 1, H(4) = 0$.*

1. *$\bigcirc$ has $B_0 = 1$, $B_1 = 0$, also it generates and publishes three numbers: $C = 4$, $p = 5$ and $g = 2$ (Note that $2^0 = 1 \bmod 5, 2^1 = 2 \bmod 5, 2^2 = 4 \bmod 5, 2^3 = 3 \bmod 5$)*

2. *$\triangle$ wants to learn the value of B_0 ($\sigma = 0$). $\triangle$ picks a random number $k = 3$, sets $P_0 = 2^3 = 8 = 3 \bmod 5$, and $P_1 = (C/P_0) = 4/3 = 4 * 2 = 8 = 3 \bmod 5$ (note that ${P_0}^{-1} = 2$ is the multiplicative inverse of $P_0 = 3$ taken $\bmod 5$; $P_0 * P_0^{-1} = 3 * 2 = 1 \bmod 5$). $\triangle$ sends P_0 to $\bigcirc$.*

3. *$\bigcirc$ independently calculates P_1 by evaluating $4/P_0 = 4 * 2 = 3 \bmod 5$, and randomly choosing $r_0 = 4, r_1 = 3$ calculates $g^{r_0} = 2^4 = 16 = 1 \bmod 5$, $g^{r_1} = 2^3 = 8 = 3 \bmod 5$, $(P_0)^{r_0} = 3^4 = 81 = 1 \bmod 5$, $(P_1)^{r_1} = 3^3 = 27 = 2 \bmod 5$, $M_0 = H(1) \oplus B_0 = 0 \oplus 1 = 1$, and $M_1 = H(2) \oplus B_1 = 1 \oplus 0 = 0$. $\bigcirc$ sets $E_0 = (1, 1)$ and $E_1 = (3, 1)$, and sends E_0, E_1 to $\triangle$.*

4. *$\triangle$ computes $T_0 = H((P_0)^{r_0} \bmod 5) = H((3^4) \bmod p) = H(81 \bmod 5) = H(1) = 0$ and gets B_0 by computing $T_0 \oplus M_0 = 0 \oplus 1 = 1$.*

Thus $\triangle$ gets to know the contents of the box 0 (equal to 1) while $\bigcirc$ does not know which box was opened by $\triangle$. This shows the secure 1 out of 2 oblivious transfer. The approach above can be extended to deal with 1 out of n oblivious transfer. We provide details in Appendix B.3.

2.4 Formal Statements Regarding Security of the Algorithm

Formally, we divide the security into two parts. First, if each party follows the protocol correctly, we show that the algorithm is secure. Second, we show that each party is best off if they follow the protocol correctly.

A secure implementation requires us to prove that nothing is learned by either party during the execution except what can be deduced from the final points swapped. To prove

[3] We would like to stress that 1024 bit or larger p and C values would be used in practice for secure applications.

this, we show it is possible to build a "simulator" that uses only the final results and the data known by each individual party to identify the result of the comparison. The proof requires verification that the results at each step can indeed be deduced from the result. The logic that follows is that no new information is revealed at each step of the execution of the algorithm. This provides a formal proof of security of the algorithm *if each party were to follow the protocol provided.*

Theorem 2. *Algorithm 1 is secure under the semi-honest definition of Secure Multiparty Computation.*

Proof: See Appendix D.

The theorem above formally establishes that the OROD algorithm results in a secure implementation.

The Secure Multiparty Computation literature also provides techniques secure against *malicious* parties; i.e., parties that deviate from the protocol can be detected. However, even with security against malicious parties one can "cheat" by faking ones input and running the protocol honestly. We thus focus on the concept of *incentive compatibility* i.e., it is in each party's self interest to follow the protocol provided with correct input. In other words, trying to learn more is discouraged by the fact that the party is worse off or is detected as cheating. (Similar issues were explored in Dodis, Halevi & Rabin (2000), who suggest that finding problems where incentive compatibility simplifies protocols is an interesting research area; this is discussed further in Appendix E.) Formally, we have to show that it is incentive compatible for each party to follow the protocol. Note that we will assume that if a party does not benefit from violating the protocol, it will follow the protocol.

Theorem 3. *The OROD protocol is incentive compatible, i.e., any failure to correctly follow the protocol results in the dishonest party being worse off, or leaves the result unaffected and gives no additional information to the dishonest party.*

Proof: See Appendix E.

2.5 Execution of OROD among Multiple Parties

Define the optimal result from the execution of OROD among multiple parties to be the perfect partitioning among all parties' one-dimensional datasets. In other words, any two of parties' ranges of their one-dimensional datasets are disjoint. This results in the minimal global tour (in one dimension).

We note one problem: it is not possible to draw a space-filling curve such that every party's depot is uniquely at the opposite end of the line from every other party. The result is that a globally optimal partitioning may not be optimal (or even an improvement) for every party; we lose the incentive compatibility Theorem 3. However, if we assign a total ordering to the parties and use this to determine which end of the line they get in any pairwise operation, repeated execution does converge to a minimum global tour.

Theorem 4. *Execution of OROD among k-parties eventually reaches optimal: k partitions (where $k \geq 2$).*

Proof: See Appendix C.

However, this multiple party execution of OROD is *not* secure under the definitions of SMC: parties may see intermediate results (swapped points) that eventually are passed to another party. To be secure, each party should see only those points that it starts with, and that it ends with.

However, with one additional constraint we can utilize the two-party OROD protocol as a subroutine to build a secure multi-party protocol. The constraint is that once a party has received a swapped point, it can never swap it again:

- Constraint *MP*: whenever OROD is executed between any two parties, the points swapped between them should be removed from consideration for their subsequent execution with other parties.

Informally, this constraint says that a subcontractor may not further subcontract the work; this is likely to be a reasonable restriction for other reasons (companies want some quality control by knowing who will actually perform their work.)

Theorem 5. *Executing OROD among multiple parties is secure provided that constraint MP is satisfied.*

We do not provide a formal proof of this theorem since it follows the same steps as Theorem 2. According to Theorem 2, a single execution of OROD is secure. In addition, since constraint *MP* must be satisfied, no swapped points are considered for the subsequent executions, and all intermediate swapped points are part of the final result. Therefore, each party's view during the execution of OROD among multiple parties can be simulated by the party's input and output, following the same steps as in the proof of Theorem 2.

Also note that securely executing OROD among multiple parties merely requires each party executes OROD with the other parties once, provided that each execution of OROD satisfies constraint *MP*. The reasoning follows: suppose party i has already executed OROD once with all other parties, and assume that i executes OROD with party j (where $i \neq j$) for a second time. Since constraint *MP* is satisfied, the points considered during this second execution between i and j should merely be their original points. Based on Theorem 1, the first execution of OROD between i and j creates two perfect partitions of their original points. In other words, if some of their original points were swapped after the second execution of OROD, these points would have been swapped after the first execution. Consequently, when constraint *MP* is guaranteed, any subsequent execution of OROD between i and j after the first execution does nothing.

3. Experimental Analysis

In this section, we present results obtained when the algorithms described in this paper were applied to an empirical dataset provided by a trucking company, representing LTL (Less

Than Truckload) loads during the first quarter of 2003. The purpose was to observe the empirical effect of running the algorithm on a dataset where we have (a) individual points in two dimensions, and (b) pickup and drop off pairs associated with each shipment. The dataset contains 81,842 data points (including both pickup and delivery points) spanning 11 weeks. One issue in the dataset is that in the actual problem, what we have to swap is *loads* i.e., pickup and delivery pairs while the algorithm picks *points* to swap. Thus the results of the empirical tests provide an indication of the performance of the algorithm under more realistic problem conditions.

When the algorithms were executed for two companies, we randomly assigned some pick-up points to company A and the rest to company B. Based on the assigned pick-up points, each pickup and delivery pair for each week was assigned to company A or B. Within a given weekly data set, we considered a single truck completing all pick-ups and then all deliveries. Actual distance being dependent on the starting point of each company, we used the left-most pickup point in one dimension as the starting point of company A and the right-most point for company B. Before swapping any points, we obtained a pre-swap tour length (in two dimensions) of the pick-up and delivery for each company by using the 2-opt algorithm (Lin & Kernighan 1973). We also used the same optimization method to calculate the tour length after swapping.

We considered three swapping methods for our analysis. One is Pair Swapping (PS), where we swap only if both pick-up and delivery points are beyond a certain point. Another is Average Swapping (AS), where companies swap data based on the average of pick-up and delivery points. As the number of delivery points is more than three times the number of pick-up points in the dataset, we also consider Delivery Swapping (DS) based on the delivery point. Figure 6 gives the savings in total tour length by week. Although none of our swapping method guarantees positive savings, the average savings in total distance are 0.6% for PS, 5.7% for AS and 17.8% for DS. The results thus show that DS performs best in our empirical test. The main reason for this result is that the major distance driven in our dataset is the trip to delivery points from primary pickup locations, as there are more deliveries than pickups. This long tour has the most opportunity for improvement.

Figure 7 gives the savings by party – again it is clear that destination swapping, by optimizing the longer tour, gives the best results. It also shows an interesting real-life situation. Because *pairs* are being swapped, rather than individual points, it is possible that the tour length increases. The optimality result of Theorem 1 holds for a single tour, but becomes a heuristic when both pickup and destination must be swapped. However, it is an effective heuristic: In this data, destination swapping always gives a global benefit and almost always benefits both parties.

3.1 Empirical Results of multi-party collaboration

We also evaluated the secure multi-party protocol of Section 2.5 on the dataset. The experiments were conducted under two different settings: execution among three parties and execution among five parties. Because of the constraint *MP*, different execution orderings may produce different results. Therefore, under each assignments of the data (a trial), we

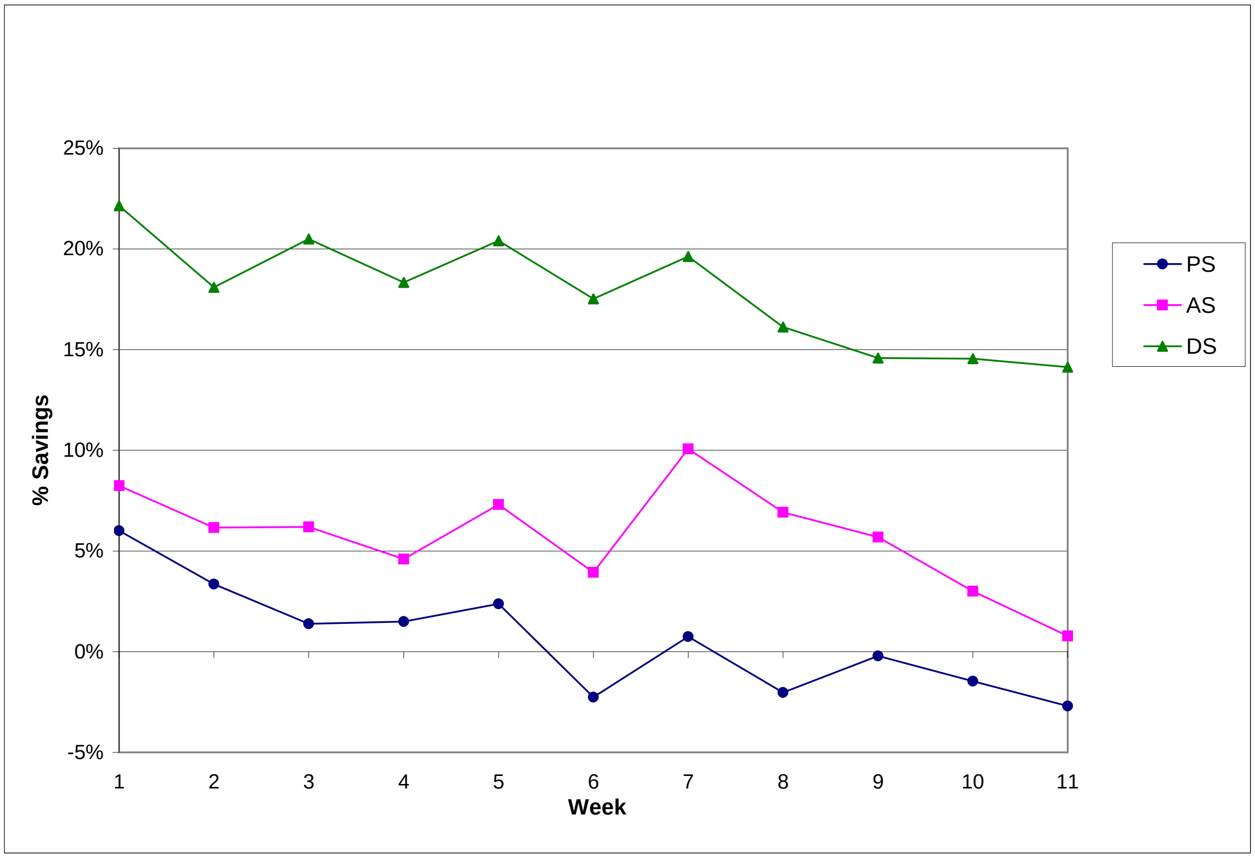


Figure 6: Savings in Total Distance

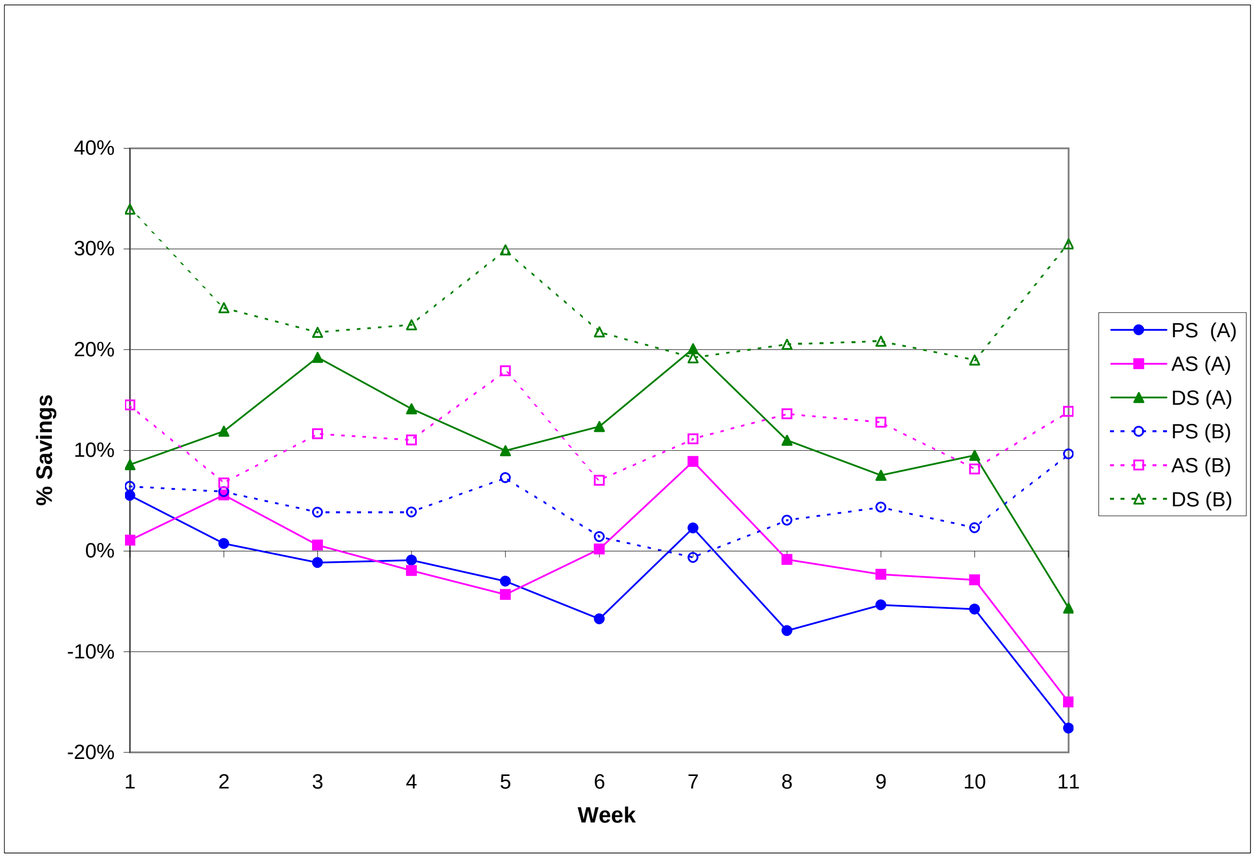


Figure 7: Savings in Individual Distance

	Trial 1			Trial 2			Trial 3			Overall
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	average
C1	16.48%	35.00%	24.02%	23.09%	31.91%	26.16%	15.52%	23.22%	19.20%	23.12%
C2	9.85%	27.11%	16.48%	-6.35%	8.69%	1.22%	1.98%	14.40%	8.54%	8.75%
C3	24.40%	29.45%	24.40%	27.65%	31.09%	27.65%	31.49%	35.62%	31.49%	27.85%

Table 1: Percentage Savings by company and trial for swap among three companies.

tried multiple execution orderings. This was repated for three trials. We present the best case, the worst case and the average savings (in percent) across all possible orderings for each trial.

Data for all experiments presented here comes from the 1470 pickup and delivery pairs in the first week of the data set. Each party is given the same number of pick-up/delivery pairs. Based on the superior two-party performance of the Delivery Swapping strategy, we chose that as the heuristic.

3.1.1 Empirically measured Savings with Three Companies

In this section, we consider the case when the weekly data was split (randomly) across three companies for a single trial. As described earlier, there are six possible sequences of two party algorithm executions, which are represented by the possible permutations of the ordering of three companies i.e., {(C1 C2), (C1 C3), (C2 C3)}, {(C1 C2), (C2 C3), (C1 C3)}, { (C1 C3), (C1 C2), (C2 C3)}, {(C1 C3), (C2 C3), (C1 C2)}, {(C2 C3), (C1 C2), (C1 C3)},{ (C2 C3), (C1 C3), (C1 C2)}. In the description above, (C1 C2) denotes the execution of the two-party protocol between C1 and C2 with constraint *MP* being satisfied. The secure multiparty algorithm was executed for all six of these possible sequences. This was repeated for three trials.

Table 1 shows statistics for each trial, and provides the maximum, minimum and average savings for each of the three companies (C1, C2 and C3) across the 6 possible execution orderings. For some execution orderings and initial data assignments, a company may not see benefits in the original problem context (e.g., see the minimum % change for C2 in Trial 2). But, in most cases, on average, every company observes cost reductions varying from 8.7 % to 27.9 %.

3.1.2 Empirically measured Savings with Five Companies

We ran the same experiment with five companies. Thus, we divided up the data randomly into five equal subsets, one for each company. Since each company tries to swap with every other company, we have S={(C1 C2), (C1 C3), (C1 C4), (C1 C5), (C2 C3), (C2 C4), (C2 C5), (C3 C4), (C3 C5), (C4 C5)} i.e., 10 possible pairwise swaps. Because of the number (10!) of possible orderings of these swaps, we randomly chose 10 orderings for each of the three trials.

The following steps were executed for each trial:

- Randomly partition the weekly dataset into five sub-datasets (one per company).

	Trial 1			Trial 2			Trial 3			Overall
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	average
C1	4.59%	19.50%	15.75%	18.50%	30.59%	24.63%	13.30%	26.09%	18.82%	19.73%
C2	6.70%	26.77%	16.77%	0.66%	20.53%	12.27%	-13.89%	14.39%	-0.13%	9.64%
C3	-8.87%	28.78%	5.58%	3.08%	26.59%	15.23%	-4.35%	22.52%	8.59%	9.80%
C4	4.35%	34.60%	15.41%	-14.57%	14.67%	-4.11%	0.00%	36.96%	15.85%	9.05%
C5	27.83%	44.90%	33.17%	7.43%	35.46%	27.39%	25.77%	36.43%	31.34%	30.63%

Table 2: Percentage Savings by company and trial for swap among five companies.

- Execute the secure multi-party protocol 10 times, each time randomly choosing the ordering to run. (a random permutation of all elements in S.)
- Record statistics of each of the 10 random execution orderings.

The above steps were repeated three times (trials). Table 2 provides, for each company, and for each trial, the minimum, maximum and average savings across the 10 executions. The table shows that the average savings vary from 9.05 to 30.63 %. As observed earlier, there are cases where companies do not see improvements for a particular sequence of orderings. Note that the algorithms we described always guarantee savings in the space filling curve mapped dimension. The results show that the gap between the spacefilling curve and its link to the pickup and delivery travel distance in the two dimensional problem context may result in an increase in travel distance in the original travel metric in some cases.

4. Conclusion / Future Work

We have described an algorithm that enables independent companies to identify opportunities for collaboration without sharing unnecessary data. The theoretical analysis shows that the algorithm ensures that no information can be inferred except that from the shared data, and that truthtelling is incentive compatible. In addition, application of the algorithm to an empirical dataset indicate a substantial potential impact. Papers such as Dodis, Halevi & Rabin (2000) in the cryptography literature state that creation of protocols where it is incentive compatible to correctly follow the protocol and truthfully provide the data are interesting theoretical challenges. Our paper provides an example of an algorithm designed for an operations management problem context. We suggest that exploration of applied operations management problem contexts can provide a rich source of research problems as well as potential for practical applications.

The specific problem described in this paper is a step towards a greater integration of data encryption techniques and theory to problems in operations management. Another example is given by Atallah, Elmongui, Deshpande & Schwarz (2003), which addresses capacity allocation and auctions – a supplier/consumer interaction rather than the peer to peer interaction presented in this paper. An interesting open research question concerns the potential tradeoff between the use of a heuristic approach to problem solution that admits a tight security property vs. a closer to optimal algorithm that permits data leakage. We leave such exploration to future research.

Another area for research is applying the concept of incentive compatibility to secure multiparty computations. Current secure multiparty computation models do not take the reward structure of the parties into account, leading to interesting results where the most secure model (malicious model) may not provide the needed security in practice. Integrating this concept into evaluation of an algorithm, as in Section E, leads to new opportunities for practical application of secure collaborative computation.

A. Algorithm Details

A.1 Notations

$S^k = \{s_1^k, s_2^k, \cdots, s_m^k\}$, a set of m k-dimensional points
$O^k = \{o_1^k, o_2^k, \cdots, o_n^k\}$, a set of n k-dimensional points
$SF \equiv$ A space filling curve
$[a, b] \equiv$ The range of SF

A.2 Formal Definitions of Extreme Points

Definition 1. (Extreme Points with respect to X and d_x)

- X is a set of points, d_x is a direction (i.e. left or right) related to X
- An extreme point is a point in X that is the furthest away from d_x
- i^{th} extreme point is a point in X that is the i^{th} furthest away from d_x

Definition 2. ($Extreme\_POS(X, d_x, i)$)

- X is a set of points, d_x is a direction (i.e. left or right) and i is an integer
- The function returns the position of the i^{th} extreme points with respect to X and d_x
- If $i > |X|$, the function returns a position beyond the range limits (e.g., $-\infty$, $+\infty$) in the direction d_x.

Formally, $Extreme\_POS(X, d_x, i)$ is:

```
Reorder $X$ in ascending order according to position
**if** $d_x = right$ **then**
    Return the position of $i^{th}$ item of $X$ ($+\infty$ if $i > |X|$)
**else**
    Return the position of the $(|X| + 1 - i)^{th}$ item of $X$ ($-\infty$ if $i > |X|$)
**end if**
```

B. Secure Multiparty Computation

Substantial work has been done on secure multiparty computation. The key result is that a wide class of computations can be done securely under reasonable assumptions. Any function that can be represented by a polynomial circuit in terms of the number of input bits can be evaluated in reasonable time. We give a brief overview of this work, concentrating on material that is used in the paper. The definitions given here are from Goldreich (2004*a*). For simplicity, we concentrate on the two party case. Extending the definitions to the multiparty case is straightforward.

B.1 Definition of Secure Multiparty Computation

The basic idea behind showing that a protocol is secure is that no party can learn more than in a "trusted broker" model, where all parties give their input to the broker, and the broker computes and returns the result. The problem with the trusted broker model is that the broker learns everything. A Secure Multiparty Computation (SMC) accomplishes this *without* a broker learning anything.

The problem is that in the trusted broker model, the only information a party receives is the final result. However, in a real computation, more information must be exchanged (for example, the results of the *Extreme_POS* comparison). The goal is to prove that the information exchanged does not reveal anything other than what can be inferred from the final results.

The security of an algorithm is proved as follows: If a party, given its own input and the final result, can *simulate* the information received during an actual execution of the protocol, then it hasn't learned anything from the protocol. The proof of Theorem 2 demonstrates such a proof by simulation.

Note that the values chosen by the simulator may not be the same as the values seen during any *particular* execution of the protocol. What matters is that over many executions of the protocol on the same inputs, the *distribution* of values seen (which varies based on the parties choice of random numbers, such as encryption keys) is the same as that generated by simulation.

Formal definitions for secure multiparty computation, and the proof that the simulation argument in fact guarantees security equivalent to the "trusted broker" model, can be found in Goldreich (2004*a*).

In summary, a secure multiparty protocol will not reveal more information to a particular party than the information that can be induced by looking at that party's input and the output.

B.2 Cryptographic Definitions

Security of the oblivious transfer relies on certain function being infeasible to compute and the ability to simulate values seen during the protocol by choosing a random number. Two key assumptions used are given below.

Computational Diffie-Hellman Assumption (Diffie & Hellman 1976): Assume that p is a very large prime number and g is the generator of its multiplicative group (i.e., every number between $1 \dots p-1$ can be written as $g^k \bmod p$ for some k between $1 \dots p-1$). The computational Diffie-Hellman Assumption states that given $g^a \bmod p$ and $g^b \bmod p$ (Note that a and b is not given), there is no efficient way to compute $g^{ab} \bmod p$.

Random Oracle Assumption: In the construction of the protocol, we will use a cryptographic Hash function H. We assume that this function is known to all parties (e.g., SHA (NIS 1995)) and it maps its input to what appears to be (is computationally indistinguishable from) a random output. The computationally indistinguishability of the output of commonly used cryptographic hash functions from a random output is not formally provable (yet), but is a common assumption in the cryptography literature.

The Computational Diffie-Hellman Assumption is the basis for the Diffie-Hellman key exchange protocol, and it is similar to the assumptions relied on by other public-key cryptosystems. The Random Oracle assumption is also relied on by many cryptosystems (Goldreich 2004*b*). If either does not hold, many commonly-used encryption schemes could be broken.

B.3 1 out of n Oblivious Transfer

We now provide an approach to extend the 1 out of 2 protocol to generate a 1 out of n protocol. The basic idea is that the values in the boxes are masked using an exclusive or with $\log n$ bits. Each bit is an encrypted value, with two keys for each bit. One key is used if that bit in the binary representation of the desired box number is 0, the other if it is 1. Encryption requires two inputs, the key and value to be encrypted; for cryptographic reasons, the value being encrypted must be different for each box, but known to both parties – the box number itself meets this criteria.

$\bigcirc$ sends the value EB_i (exclusive or of encryptions and contents) for every box to $\triangle$. To recover the key, $\triangle$ chooses a 0 or 1 as the value it will be able to recover for each bit (corresponding to the binary representation of the desired box number.) Each bit of the key is transferred using 1 out of 2 oblivious transfer; $\triangle$ will only be able to recover the key where all the bits match the actual box to be opened, and this will only correctly decrypt the desired box number. For any other box, the result of at least one bit would appear random (because of trying to get the "wrong" bit using 1 out of 2 oblivious transfer for that bit.)

We show this protocol for 16 boxes (4 bits); assume that $\bigcirc$ has $B_1 \dots B_{16}$ and $\triangle$ wants to learn B_7. We also use an encryption function E (e.g., DES (NIS 1988)) as the hash function H meeting the random oracle assumption; as will be seen below, we need to invert the hash as part of this protocol.

1. $\bigcirc$ generates 4 key pairs

$$(K_1^0, K_1^1), (K_2^0, K_2^1), (K_3^0, K_3^1), (K_4^0, K_4^1)$$

where each $K_j^{\sigma[j]}$ are is a randomly chosen key for E. For $1 \leq i \leq 16$, with binary representation $(\sigma[1], \sigma[2], \sigma[3], \sigma[4])$, $\bigcirc$ creates $EB_i = B_i \oplus E_{K_1^{\sigma[1]}}(i) \oplus E_{K_2^{\sigma[2]}}(i) \oplus E_{K_3^{\sigma[3]}}(i) \oplus E_{K_4^{\sigma[4]}}(i)$

2. $\bigcirc$ sends all EB_i to $\triangle$.

3. Since 7 (the box number $\triangle$ wants to open) has binary representation 0111, using four executions of OT_1^2, $\triangle$ learns $K_1^0, K_2^1, K_3^1, K_4^1$.

4. $\triangle$ retrieves B_7 by calculating $EB_7 \oplus E_{K_1^0}(7) \oplus E_{K_3^1}(7) \oplus E_{K_3^1}(7) \oplus E_{K_4^1}(7)$

It can easily be proven that if the encryption scheme and OT_1^2 are secure, then the above algorithm is secure. Since it only retrieves one key from each pair, $\triangle$ accurately decrypts at most one message.

For simplicity of presentation, we have shown a construction that scales linearly (since $\bigcirc$ sends the encrypted value of every box). However, there are known logarithmic time protocols for secure comparison that can be used for step 6 of Algorithm 1; for details see Yao (1986), Goldreich (2004*a*), or Ioannidis & Grama (2003).

B.4 Related work: k^{th} ranked element

Aggarwal, Mishra & Pinkas (2004) present a protocol for securely computing the k^{th}-ranked element among lists held by multiple parties. A special case of our problem, in which the number of points of each party is known to all, can be solved by computing the k^{th} ranked element. Knowing the number of points held by (in particular) the party at the "low" end of the line gives the value to use for k; their protocol would give the location of this k^{th} element. This location would partition the line, the parties would then know to swap points to the right (left) of that location to get the optimal solution (see Lemma 3.1.)

We do not make the assumption that the number of points held by each party is public; we limit the information disclosed to *only* the swapped points. Without sharing knowledge of the number of points belonging to at least one party, finding the k^{th} element will not solve the problem. While it would be possible to use the k^{th} ranked element protocol iteratively to find the appropriate number of points to swap, this would require revealing the location of the $k + 1^{st}$ (non-swapped) point. Thus, their protocol is not suitable for our application.

C. Proof of Correctness

Theorem 1. *The OROD algorithm terminates with the optimal number of points swapped, i.e., after swapping, there are no two points that could be swapped to give a lower tour length.*

Proof. First, note that partitioning the line gives the lowest tour length: with the line partitioned, neither party needs to drive past the partition, and any swap would require that each party drive past the partition, extending the tour length (this is shown in Lemma

3.1). A simple inductive proof demonstrates that the process terminates with the number of swaps required to achieve this optimal (in one dimension) result.

First, note that it is easy to see that $lbound \leq i \leq ubound$ always holds.

We will show that the following always hold.

1. swapping fewer than *lbound* points does not partition the line, and

2. swapping *ubound* or more points does not partition the line.

First, they are trivially true initially (since we cannot swap a negative number or ∞ points).

If $Extreme\_POS(S, l, i) > Extreme\_POS(O, r, i)$ then swapping fewer than i points will not partition the line (since at least the i^{th} points would still be on the wrong side of the partion). In this case, *lbound* gets set to i, which does not violate 1. Since *ubound* is unchanged, 2 still holds.

If $Extreme\_POS(S, l, i) \leq Extreme\_POS(O, l, i)$ then swapping i or more points will not partition the line, since the i^{th} points are on the wrong side of the partition. In this case, *ubound* gets set to i, which does not violate 2; and since *lbound* is unchanged, 1 continues to hold.

If $ubound - lbound = 1$ then by 1 and 2, exactly *lbound* points must be exchanged. Since at this point $i = \lfloor \frac{lbound+lbound+1}{2} \rfloor = lbound$, the result is exactly the number that must be exchanged to partition the line.

All that remains is to show that the algorithm does terminate. While $ubound = \infty$, i doubles, and at some point must exceed the number of points on the wrong side of the partition (since the number of points is finite.) At this point, *ubound* gets set to a finite number. As long as $ubound - lbound > 1$, $lbound < \lfloor \frac{lbound+lbound+1}{2} \rfloor < ubound$, and $lbound < i * 2$, forcing i to be set such that $lbound < i < ubound$. Since at the next iteration, either *lbound* or *ubound* is set to i, either *lbound* increases or *ubound* decreases. Since the values are integral, a finite number of iterations are possible before $lbound + 1 \geq ubound$, and the algorithm terminates. □

The number of iterations can be verified to be logarithmic in the number of points swapped.

Theorem 4. *Execution of OROD among k-parties eventually reaches optimal: k partitions (where $k \geq 2$).*

Proof. This is proven through strong induction on the number of parties.

Base case: $k = 2$. True from Theorem 1.

Inductive step: Let integer $t > 2$, and assume the claim is true for all $k < t$. We need to prove that the claim is true when $k = t$.

Let $k = t - 1$. From the induction hypothesis, the execution among the k parties creates k partitions. Then let another party (say l) join the *k-parties* and i be any party among the *k-parties* or k partitions. Leave i out and execute the protocol among the remaining $k - 1$ parties plus l. Then we have a new set of k partitions.

Considering i with the other parties, if the protocol is executed between i and any one of the parties whose dataset range not overlapping that of i, the protocol does nothing. Let p be the number of parties whose dataset ranges overlapping that of i.

If $p < k$, the execution of the protocol among these p parties plus i eventually leads to a set of disjoint partitions. These partitions, plus the previously disjoint partitions with i, form $k + 1$ disjoint partitions.

If $p = k$, during the execution between p and either the left-most or right-most party, the partition of the left most or the right most party will shrink. Since there are a finite number of points for each party, the number of times the partition can shrink is finite. Therefore, eventually, no party's partition overlaps with that of the left most or the right most party. The execution among the rest of the parties will lead to k partitions. Adding the partition of either the left most or the right most party, we have $k + 1$ disjoint partitions. As a result, the claim is true when $k = t$.

Because the base case and inductive step are true, the claim is true for all $k \geq 2$. □

D. Proof of Security

Theorem 2. *Algorithm 1 is secure under the semi-honest definition of Secure Multiparty Computation.*

Proof. Communication occurs only at line 6. To prove the protocol is secure, we only need to show that given the information received at the end, the intermediate comparison results can be simulated. (The comparison is computed in a secure manner as described in Section 2.3.1, so it reveals nothing but the comparison result.)

Algorithm 2 gives the simulator for line 6 for the party at the left end of the line, S^1. The simulator works as follows: let n be the number of points to swap and i be an integer. If i is less than n, S^1 knows i has yet to be maximal, and the protocol will double the size of i. On the other hand, if i is greater than n, S^1 knows i is too big to be maximal, and the protocol will decrease the size of i to a value that is in the middle of its upper and lower bounds. If i is the same as n, the protocol terminates.

From the simulator, it is obvious that the shared results from the protocol, plus a party's input, are sufficient to precisely simulate each execution of the protocol. Therefore, because the simulation process and execution of the protocol are computationally indistinguishable, the OROD protocol is secure. □

E. Proof of Honesty

Cryptographic approaches and game theoretic solutions are complementary to each other. Specifically, Dodis, Halevi & Rabin (2000) explicitly demonstrate how cryptographic approaches can be used to solve game theoretic problems by showing how to construct a two-player game that achieves the same payoffs as a mediated two-player game (mediated

Algorithm 2 Simulator for OROD Protocol

```
Require: $S^1$, $n$
 1: $i \leftarrow 1$
 2: $lbound \leftarrow 0$
 3: $ubound \leftarrow \infty$
 4: while $(i \neq n)$ do
 5:     if $i < n$ then
 6:         $Known : Extreme\_POS(S^1, l, i) > Extreme\_POS(O^1, r, i)$
 7:         $lbound \leftarrow i$
 8:         $i \leftarrow i * 2$
 9:     else
10:         $Known : Extreme\_POS(S^1, l, i) < Extreme\_POS(O^1, r, i)$
11:         $ubound \leftarrow i$
12:         $i \leftarrow \left\lfloor \frac{lbound+ubound}{2} \right\rfloor$
13:     end if
14: end while
```

by an "honest broker"). We have shown how this approach can be used to solve a practical problem, and now prove that the payoff for this problem guarantees incentive compatibility.

There are two levels at which incentive compatibility can be shown. The first is to show that it is incentive compatible to provide the correct data for the algorithm. This is the traditional definition used in economics. The second is to show that it is optimal for each party to correctly follow the protocol i.e., any attempt to gain more information by deviating from (compromising the security of) the protocol is self defeating or will result in discovery by the other party. This is related to the distinction between "semi-honest" and "malicious" in Secure Multiparty Computation definitions. A protocol that does not meet the first level (giving incorrect data) is vulnerable even if it meets the standard of security against malicious parties. A protocol that meets the second form of incentive compatibility need not be secure against malicious parties; it will be in the party's best interest to follow the protocol, and the semi-honest security ensures that no excess information is disclosed. We will show that our protocol meets the first type (correct data), and with the exception of the secure comparison (for which we describe protocols secure against a malicious party), our protocol also meets the second. Thus we ensure both correct results and prevent disclosure of information by ensuring that it is in a party's best interest to follow the protocol with correct data.

We first assume that the space filling curve heuristic is jointly agreed on by all parties because it is already built into several available mechanisms for vehicle routing. It is possible that the heuristic may cause an optimal one-dimensional result to be non-optimal, and possibly not beneficial for all parties, in two dimensions. For one party to know that the heuristic is "bad" for a particular dataset requires a knowledge of the other party's data; we assume that parties do not know each other's information. By the same argument, a party would not be able to manipulate the choice of heuristic to its advantage.

We now show that cheating in the protocol results in a non-optimal one-dimensional solution, and that such a solution is in neither party's best interest. We start with the following lemma.

Lemma 3.1. *If both parties swap an equal number of points, each will achieve the greatest benefit if the points swapped partition the line.*

Proof. Formally, the cost for a party P, $Cost(P)$ is

```
{Let $S$ represent the list of points owned by $P$}
Reorder $S$ in ascending order according to position
$Cost = |Position(S_{|S|}) - Position(S_d)|$
```

Let C be the dishonest party and H be the honest party. Let C_o be C's points before executing the protocol, C_t be C's points after a correct execution of the protocol and C_c be C's points after a false execution. H has the corresponding sets as well. Note that $|C_t| = |C_c| = |C_o|$, since an equal number of points are given up and received.

Let $C_{t|C|}$ be the farthest point for C in the true set and $C_{c|C|}$ be the farthest in the false set. Assume $Position(C_{c|C|}) < Position(C_{t|C|})$. Since $C_{t|C|} \notin C_c$, $C_{t|C|} \in H_c$. Since $|H_c| = |H_t|$, $\exists X \in H_t - H_c$, and since no points are lost, $X \in C_c$.

Since the line is partioned in a true execution (Theorem 3), $Position(C_{t|C|}) < Position(H_{t|H|})$. But this gives

$$Position(C_{c|C|}) < Position(C_{t|C|}) < Position(H_{t|H|}) \leq X.$$

This gives a contradiction: $Position(C_{c|C|})$ cannot be the farthest point for C. □

Theorem 3. *The OROD protocol is incentive compatible, i.e., any failure to correctly follow the protocol results in the dishonest party being worse off, or leaves the result unaffected and gives no additional information to the dishonest party.*

Proof. Theorem 1 shows that the algorithm produces a partition. A partition is optimal for *both* parties: If a dishonest party gets a point on the wrong side of the partition, it must travel beyond the partition, extending its tour length. We will show that any cheating leads the cost (defined as above) to increase, thus producing an incentive for honesty.

First, note that a dishonest party cannot "invent" false points to swap. If a dishonest party tries to give away a point that does not really exist, the cheating will be detected when the honest party goes to pick up the non-existent load. Contractual obligations can enforce disincentives to such cheating. The same holds true for giving up fewer or more points than agreed upon by the algorithm.

This leaves two avenues for dishonest behavior:

- Cause the algorithm to arrive at the wrong value, and
- Give away different points than those determined by the algorithm.

From Lemma 3.1, we have shown that any cheating that causes points to be swapped other than those swapped in a correct execution of the algorithm will be disadvantageous to the dishonest party.

Any cheating in the comparison at step 6 of Algorithm 1 does not reveal any information to C. At most, C can cause the comparison to give the wrong result. Looking at the proof of Theorem 1, we see that at any point, the correct number of items to swap *true* lies in the range $lbound \leq true < ubound$. Suppose the correct comparison result is $>$; then we are supposed to set $lbound \leftarrow i$ – so we know $true \geq i$. If C causes the comparison result to be wrong ($\leq$), it would instead set $ubound \leftarrow i$. However, since *ubound* can never increase, and $true \geq i$, the result of the algorithm will not be the true result. An analogous construction works with *lbound*. Therefore, any dishonesty in the execution of the algorithm either causes both parties to come to the wrong answer, or if C behaves incorrectly in another step, to come to different answers. As discussed before, if C and the honest party give up a different number of points, C will be detected to have cheated and contractual disincentives would apply. By Lemma 3.1, swapping the wrong number of points is disadvantageous.

The second possibility is that a dishonest party comes to the correct number to swap, but gives up the wrong points. Assume a point X is kept that should have been swapped. The same approach used in Lemma 3.1 can be used to show that $position(X) > position(C_{t|C|})$, increasing the cost for the cheating party.

Thus, any cheating either increases the cost for the dishonest party, or is discovered and provable by the honest party. Honesty is thus incentive compatible. $\square$

References

Aggarwal, Gagan, Nina Mishra & Benny Pinkas. 2004. Secure Computation of the k^{th}-Ranked Element. In *Proceedings of IACR Eurocrypt (EUROCRYPT04).* Interlaken, Switzerland: pp. 40–55.

Atallah, Mikhail J., Hicham G. Elmongui, Vinayak Deshpande & Leroy B. Schwarz. 2003. Secure Supply-Chain Protocols. In *IEEE International Conference on E-Commerce.* Newport Beach, California: pp. 293–302.
*http://ieeexplore.ieee.org/xpl/citationdwnld.jsp?arNumber=1210264

Bandler, James. 2003. "How Seagoing Chemical Haulers May Have Tried to Divide Market." The Wall Street Journal.

Bartholdi III, John J. & Loren K. Platzman. 1982. "An O(N log N) planar travelling salesman heuristic based on spacefilling curves." *Operations Research Letters* pp. 121–125.

Bartholdi III, John J. & Loren K. Platzman. 1988. "Heuristics Based on Spacefilling Curves for Combinatorial Problems in Euclidean Space." *Management Science* 34(3):157–160.

Diffie, Whit & Martin Hellman. 1976. "New Directions In Cryptography." *IEEE Transactions on Information Theory* IT-22(6):644–654.

Dodis, Yevgeniy, Shaih Halevi & Tal Rabin. 2000. A Cryptographic Solution to a Game Theoretic Problem. In *Advances in Cryptology – CRYPTO 2000*. Springer-Verlag.

Goldreich, Oded. 2004*a*. *The Foundations of Cryptography*. Vol. 2 Cambridge University Press chapter General Cryptographic Protocols.
*http://www.wisdom.weizmann.ac.il/ oded/PSBookFrag/prot.ps

Goldreich, Oded. 2004*b*. *The Foundations of Cryptography*. Vol. 2 Cambridge University Press chapter Encryption Schemes.
*http://www.wisdom.weizmann.ac.il/ oded/PSBookFrag/enc.ps

Goldreich, Oded, Silvio Micali & Avi Wigderson. 1987. How to Play any Mental Game - A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing*. pp. 218–229.
*http://doi.acm.org/10.1145/28395.28420

Ioannidis, Ioannis & Ananth Grama. 2003. An Efficient Protocol for Yao's Millionaires' Problem. In *Hawaii International Conference on System Sciences (HICSS-36)*. Waikoloa Village, Hawaii: pp. 205–210.

Kalagnanam, J., M. Trumbo & H. S. Lee. 2000. "The Surplus Inventory Matching Problem in the Process Industry." *Operations Research* 48(4).

Keskinocak, Pinar & Sridhar Tayur. 2001. "Quantitative Analysis for Internet-Enabled Supply Chains." *Interfaces* 31(2):70–89.

Lin, S. & Brian W. Kernighan. 1973. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem." *Operations Research* 21(2):498–516.

Naor, Moni & Benny Pinkas. 1999. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. Atlanta, Georgia, United States: ACM Press pp. 245–254.

Naor, Moni & Benny Pinkas. 2001. Efficient Oblivious Transfer Protocols. In *Proceedings of SODA 2001 (SIAM Symposium on Discrete Algorithms)*. Washington, D.C.: .

NIS. 1988. Data Encryption Standard (DES). Technical Report FIPS PUB 46-2 National Institutes of Standards and Technology.
*http://www.itl.nist.gov/fipspubs/fip46-2.htm

NIS. 1995. Secure Hash Standard. Technical Report FIPS PUB 180-1 National Institutes of Standards and Technology.
*http://www.itl.nist.gov/fipspubs/fip180-1.htm

Wilson, Rosalyn & Robert V. Delaney. 2003. "14th Annual "State of Logistics Report"©: The Case for Reconfiguration.".
*http://www.cassinfo.com/bob_pc_2003.html

Yao, Andrew C. 1986. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*. IEEE pp. 162–167.