An Efficient and Secure Shortest Path Protocol for MPC, and its Application to Restricted Combinatorial Auctions

Abdelrahaman Aly, Sara Cleemput

imec-COSIC, KU Leuven, ESAT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee (Belgium)
firstname.lastname@esat.kuleuven.be

Abstract. We propose a protocol to securely compute a solution to the (single source) Shortest Path Problem, based on Dijkstra's algorithm and Secure multi-party Computation. Our protocol is a data oblivious abstraction that improves state of the art by Aly et al. [FC 2013 & ICISC 2014] and offers perfect security against both semi-honest and malicious adversaries. We achieve this without the use of more complex cryptographic techniques such as Oblivious RAM, Moreover, it can easily be adapted to form a subroutine in other combinatorial mechanisms and give, on this regard, an example on how it can help solve certain combinatorial auctions, although we believe it can be used for applications on traffic and network analysis, among others. Finally, we demonstrate the efficiency of our protocol by experiments and benchmarking.

Keywords: shortest path problem, combinatorial auctions, secure multi-party computation

1 Introduction

The Shortest Path problem (SPP), i.e. computing the shortest path between two vertices in a graph, is a common subroutine in various applications. In many settings data related to the computation, such as elements of its configuration, topology or associated weights, is held by competing parties. Real life examples include telecommunication networks for banking, and restricted topology combinatorial auctions modeled as graphs. In such environments, different parties could gain a competitive advantage by obtaining privately held information. Therefore, mechanisms to ensure correctness and fairness are required.

In combinatorial auctions [1], participants can bid for individual items or for any sub-set of items. This is particularly relevant if there are interdependencies between the items, or if they naturally form a set, e.g. airport time slot allocations. In this paper we consider the case where the number of combinations of items is polynomially limited and all combinations are pre-agreed by the auctioneers. Thus, the combinations can be expressed as a graph with a restricted topology, where the path that expresses the optimal combination of bids for the seller can be found by solving an SPP [2]. In an ideal setting, a trusted party (auctioneer) receives the (secret) bids and returns the optimal result. In reality a trusted party is difficult to find, making this an ideal setting for multi-party Computation (MPC), which encompasses a wide collection of techniques that allow any set of parties to jointly compute any function without disclosing privately held inputs.

In this work we introduce an MPC-based data-oblivious protocol to securely solve the single source SPP. Using the findings of Aly et al. [3,4], we further adapt Dijkstra's algorithm. We consider all information related to the graph (except its topology) to be privately held. The result of our computation is the length of the path and/or the path composition; the parties decide whether these are disclosed. Our protocol can be used to find solve certain combinatorial auctions. Moreover, it can offer perfect security and its multiplicative complexity is one order of magnitude lower than the current state of the art [3,4], matching the $\mathcal{O}(|V|^2)$ of the original Dijkstra protocol.

1.1 Related Work

Several protocols to securely solve linear programming problems using the simplex algorithm have been proposed in the literature [5,6,7]. Toft [6] pointed out several security weaknesses in the protocols by Li and Atallah [5] and presented termination conditions and methods for the algorithm. The Catrina and Hoogh [7] method implements the simplex algorithm as well, but includes optimized support for rational numbers.

Aly et al. [3,4] have introduced several data-oblivious protocols to solve the SPP, including the adaptations of Dijkstra's that this work improves. However, their bound on the number of multiplications (i.e., effective work) is cubic, whereas we only require a quadratic number of multiplications. Brickell and Shmatikov [8] introduced a protocol for the SPP in a two-party setting against semi-honest adversaries. In contrast, our solution is not limited to the two-party case and also provides security against active adversaries. The Breadth-First-Search (BFS) proposed by Blanton et al. [9] provides complexity bounds for a special case of the SPP, the non-weighted graph. Conversely, we consider the general case where the graph is weighted.

An example of a mechanism to achieve oblivious distributed storage is ORAM. Wang et al. [10], for example, used ORAM in a client-server configuration to extract information without letting the server learn any access pattern by the use of data-structures. Moreover, recent advances on ORAM have led to the development of secure two-party computation protocols using ORAM. Liu et al. [11] use oblivious data structures to securely solve the shortest path problem and other combinatorial problems.

Furthermore, Keller and Scholl [12] implemented Dijkstra's algorithm using Oblivious RAM (ORAM) based data-structures matching the $\mathcal{O}(|V|^2)$ complexity of the original algorithm. However, their results show that, for certain graph sizes, the results provided by Aly et al. [3] can out-perform their ORAM-based implementation, as ORAM's intrinsic overhead exceeds any asymptotic advantage.

1.2 Notation and Security

We follow the graph notation introduced by Aly et al. [3,4]. Furthermore, we make use of the square bracket notation, for secret shared values and consider all inputs to be elements of $\mathbb{Z}_q$ where q is a sufficiently large prime or RSA modulus. Complexity is measured in terms of round complexity (multiplicative depth or latency) and multiplicative complexity (amount of work or throughput) of the whole protocol. To represent negative numbers, we follow the convention in the literature, i.e. the upper halve of the field represents negative numbers. Vectors and matrices are represented by capital letters e.g. E, where $|E|$ denotes its size. Finally, some common tasks used throughout our protocols are denoted as follows: **i).** $[z] \leftarrow_{[c]} [x] : [y]$ is the conditional operator, an arithmetic replacement for the flow instruction for branching. $[c]$ represents a selection bit: $[z]$ takes the value of $[x]$ if $[c] == 1$ and $[y]$ otherwise. This simple construction requires only one communication round. **ii).** `exchange`$(i, j, [X])$ swaps the elements in the i-th and j-th position of vector X. This operation is not cryptographic in nature.

The *Security of MPC protocols* is typically defined in the context of an existent ideal MPC functionality, as in [13]. We model this functionality as an **arithmetic black box** or $\mathcal{F}_{ABB}$. The $\mathcal{F}_{ABB}$ allows us to store secret values and perform basic operations on them. Furthermore, values can be extracted and made publicly available. We extend the basic functionality provided by the $\mathcal{F}_{ABB}$ by adding secure comparisons to its arithmetic operations. We can use this to build more complex functionality, that we consider as part of the $\mathcal{F}_{ABB}$ as well. We consider that we have access to such $\mathcal{F}_{ABB}$ and that it provides perfect security. In practice, we assume the functionality is available by means of various protocols. The basic functionality offered by our $\mathcal{F}_{ABB}$ is the following:

- $[x] \leftarrow$ `share`(x) is used to store values on the $\mathcal{F}_{ABB}$.
- $x \leftarrow$ `open`$([x])$ is used to extract values from the $\mathcal{F}_{ABB}$ and make them publicly available. In practice inputs are reconstructed using the underlying MPC functionality.
- $[c] \leftarrow [x] \overset{?}{<} [y]$ returns a secret shared $\{0, 1\}$ value. If $[x]$ is smaller than $[y]$ the function returns $[1]$ and otherwise $[0]$.

Furthermore, we extend our arithmetic black box with the following complex MPC constructions. Note that, in practice both of them can be achieved under the same security model than the rest of the functionality associated to our $\mathcal{F}_{ABB}$:

- $[z] \leftarrow$ `max`$([E])$ returns the secret shared representation of the maximal value in E and its associated index value. This can be easily achieved using the $[x] \overset{?}{<} [y]$ functionality.
- $[E] \leftarrow$ `permute`$([E])$ returns a secret shared random permutation of E e.g. [14].

On protocols for the $\mathcal{F}_{ABB}$ functionality:

Comparisons: Comparisons are an essential building block of our protocols. They have been studied extensively in the literature. Indeed, several methods providing various levels of security have been proposed, ranging from perfect security [15] to statistical security [16,17,7], as there is always a trade-off between security and performance. As stated before, comparisons are more expensive than multiplications, due to their inherent complexity.

Secure Permutations: This problem has recently been studied in the literature by employing sorting networks, permutation networks and permutation matrices [18,12]. To the best of our knowledge, this functionality can easily be achieved in $\mathcal{O}(n \cdot log(n))$, n being the size of the vector to be permuted, where the output is drawn from an (almost) uniform distribution [14].

Sorting Mechanisms: In order for the combinatorial auction mechanism to work, we need to be able to sort the bids according to price. Several efficient, secure sorting algorithms have been proposed in the literature [19,20,21]. However, the approach by Hamada et al. [22], securely random shuffling the vector before sorting, is most appropriate to our application, since it allows us to use a traditional sorting algorithm without revealing the values. We use this latter technique in our Dijkstra construction.

Note that this modular approach to constructing secure functionality over MPC, is a common practice, and can be proven secure under the hybrid model proposed by Canetti [23,24]. In our case, it suffices to define our security under the stand-alone model as follows:

Definition 1. *Let π be a real protocol implemented in a multi-party setting. We say π is secure if, for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}_{\mathcal{A}}$ such that the $\mathtt{VIEW}_{\pi}(P_i)$ of any party P_i controlled by the adversary $\mathcal{A}$ and the $\mathtt{VIEW}_{\mathcal{F}_{SP}}(P_i)$ controlled by the simulator $\mathcal{S}_{\mathcal{A}}$ cannot be distinguished (with non-negligible probability).*

2 Privacy Preserving SPP

Let $G = (V, E)$ be a directed graph without negative cycles. We can then solve the SPP from a source vertex s to any other vertex of G. G can be represented as a weighted adjacency matrix U where U_{ij} is the weight of edge $(i, j) \quad \forall (i, j) \in E$. This can be formalized as follows [25]:

$$min \sum_{v,w \in E} c_{v,w} x_{v,w}, \tag{1}$$

subject to

$$\sum_{w:(v,w)\in E} x_{v,w} - \sum_{w:(w,v)\in E} x_{w,v} = \begin{cases} n-1 & for\ v = s \\ -1 & \forall v \in V \backslash \{s\} \end{cases} \tag{2}$$

$$x_{v,w} \geqslant 0 \qquad \forall (v,w) \in E. \tag{3}$$

On plain-text, this problem can be solved in polynomial time by the use of a textbook version of Dijkstra's Shortest Path Algorithm [25]. Note, however, that given the restrictions put in place by the data being secret shared among the parties, the algorithm has to be adapted.

We now introduce the underlying intuition of our protocol: U is obliviously permuted before protocol execution. We then assign temporary labels to each vertex in G (i.e. each row in $[U]$). As in [3,4], our protocol then proceeds to identify the most suitable vertex to explore. However, unlike them, we are able to open the temporary label, as the label itself does not convey any information other than the position of the next row in the now permuted matrix $[U]$, to be analyzed. This technique is somewhat similar to the the `shuffling before sorting` technique introduced by [22]. The protocol then continues by using the data stored in the indicated row to update the distance vector, while the vertex is added to the shortest path. Weights in U are stored in secret shared form, as is the secret shared source vertex $[s]$. Our protocol's output is the set of vertices on the shortest path $[\alpha]$ and their corresponding distances $[D]$.

2.1 Non-Disclosure Oblivious Dijsktra Protocol

The original protocols [3,4] use expensive mechanisms to extract the next vertex in G to be analyzed. This overhead adds an extra order of magnitude to the multiplicative complexity of the protocols, i.e. $\mathcal{O}(|V^3|)$ (for complete graphs). As in [3,4], we consider all inputs (except $|V|$ or an upper bound) to be secret shared, to be integer and to be bounded by q in such a way no overflow occurs. Protocol 1 shows the necessary changes to the implementation. We make the same assumptions as in [4], that is to say, w.l.o.g. we consider G to be a complete graph.

Our protocol follows the logic behind the original Dijkstra's algorithm: it starts at the source vertex, explores all outgoing edges, updates distances towards adjacent vertices, chooses the "closest" vertex, adds it to the shortest path and repeats the process until the whole graph has been explored. More specifically, Protocol 1 works as follows: in lines $1-3$ the output vectors $[\alpha]$ and $[D]$ are initialized. The element in $[D]$ corresponding to the source node is initialized as $[0]$, all others are initialized as $[\top]$ (a constant greater than any input, but much smaller than q, to avoid overflow). We have included a permutation (see line 4), which has a complexity of $\mathcal{O}(n \cdot log(n))$ [14] (in this context, n stands for vector size). The original vertex identifiers (the $[P]$ vector in our protocol) are jointly permuted as well. Thus, in lines $6-11$, we can use the indexes of the permuted inputs to identify the best vertex to explore next, without revealing any information related to the vertex itself. This allows us to perform the `exchange` operation (see line 13) in the clear. The protocol can then track $[P]$ and its state (lines $14-19$). This last step is similar to what was introduced by [3,4]. Thus, we can achieve quadratic complexity in the number of multiplications (work), comparisons and rounds.

Protocol 1: Optimized Non-Disclosure Dijkstra Protocol ($\pi_{\mathtt{SP}}$)

```
Input: secret shared edge weights $[U]_{i,j}$ for $i, j \in \{1, ..., |V|\}$, encoding vector
    $[S]$ where $S_i = 0$ if $i \neq s$ ($s$ being the source vertex) and 1 otherwise.
Output: The vector of predecessors $\alpha$ and the vector of distances $[D]$.
1  for $i \leftarrow 1$ to $|V|$ do
2      $[\alpha]_i \leftarrow i$; $[D]_i \leftarrow_{[S_i]} [0] : [\top]$; $[P]_i \leftarrow [i]$;
3  end
4  $([P], [D], [U]) \leftarrow$ permute$([P], [D], [U])$;
5  for $i \leftarrow 1$ to $|V|$ do
6      $[d'] \leftarrow [\top]$;
7      for $j \leftarrow |V|$ to $i$ do
8          $[c] \leftarrow [D]_j \overset{?}{<} [d']$;
9          $[v] \leftarrow_{[c]} j : [v]$;
10         $[d'] \leftarrow_{[c]} [D]_j : [d']$;
11     end
12     $v \leftarrow$ open$([v])$;
13     exchange$(i, v, [P], [D], [U])$;
14     for $j \leftarrow i + 1$ to $|V|$ do
15         $[a] \leftarrow [D]_i + [U]_{i,j}$;
16         $[c] \leftarrow [a] \overset{?}{<} [D]_j$;
17         $[D]_j \leftarrow_{[c]} [a] : [D]_j$;
18         $[\alpha]_i \leftarrow_{[c]} [P]_j : [\alpha]_i$;
19     end
20 end
```

Complexity: The multiplicative complexity of Protocol 1 is dominated by the permutation. As stated before, the complexity of a secure oblivious vector permutation is $\mathcal{O}(n \cdot \log(n))$. However, as we are permuting a matrix instead, our protocol requires $\mathcal{O}(|V|^2 \cdot log(|V|))$ secure multiplications (amount of work). Such multiplications can be parallelized achieving $\mathcal{O}(|V|^2)$ rounds of communication. Furthermore, Protocol 1 contains two additional multiplications in line 17 and 18, which can also be parallelized. The `exchange` operation does not influence the complexity of the protocol, as it is done over publicly available information.

Security Analysis: Our protocol does not disclose any private information during its execution. More precisely, the call to `open`$([v])$ (in line 12 of Protocol 1) does not reveal the original index position of the analyzed vertex, since the vertices are uniformly (and obliviously) permuted. The *Achievable Security* of our protocol is the same as that of the underlying MPC protocols that implement it i.e., we can achieve perfect security assuming honest majorities and secure channels for the active and passive case [26]; or cryptographic security assuming dishonest majorities for the active and passive case [27,28]. More formally, we first proceed to define our ideal functionality as follows:

Definition 2 (Ideal Functionality $\mathcal{F}_{SP}$). *Let $G = (V, E)$ be a connected directed graph. Let the elements of the weighted adjacent matrix U and the source vertex s be elements of $\mathbb{Z}_q$, and let both be privately held inputs. The ideal functionality $\mathcal{F}_{SP}$ receives both $[U]$ and $[s]$ and returns the shortest path $[\alpha]$ and the distances $[D]$ to an adversary $\mathcal{A}$ who is in control of the corrupted parties via the Simulator $\mathcal{S}$.*

We now proceed to prove security for Protocol 1 (denoted as $\pi_{\mathtt{SP}}$) as follows:

Theorem 1. *The protocol π_{SP} securely implements $\mathcal{F}_{SP}$ in the $\mathcal{F}_{ABB}$ framework.*

Proof. The disclosed intermediate values v do not convey any information to the adversary, as they are indexes of the permuted matrix (uniformly distributed on V). Furthermore, the protocol flow only depends on publicly available values, i.e. the upper bound on the number of vertices and the v values. Hence, integrity is guaranteed by the underlying MPC protocol. The simulation of the complete protocol can be achieved by calling the simulators available for the atomic operations in the order fixed by the protocol flow. Since the real and ideal views for the atomic operations are themselves equal (as the real functionality is implemented by the $\mathcal{F}_{ABB}$), $\mathtt{VIEW}_{\pi_{SP}}(P_i) \equiv \mathtt{VIEW}_{\mathcal{F}_{\mathtt{SP}}}(P_i)$, $\forall\ P_i \in P$ where P is the set of all parties. $\square$

3 Privacy-Preserving Combinatorial Auctions

Combinatorial auctions are a common mechanism for exchanging different subsets of items. We explore how to use our protocol in this setting. More specifically, how to find the maximal path for a topology restricted combinatorial auction as depicted by Vangerven et al. [2], but in this case, without the need for a central auctioneer. A similar case was studied by Nojoumian and Stinson [29], using MPC and dynamic programming. However, unlike theirs, our auction scheme offers perfect security with no information leakage for both the active and passive scenario.

3.1 Auction Mechanism

We formulate the problem as a directed graph, where suitable bids are represented as edges and the vertices are combinations of items. The possible combinations are pre-agreed by the sellers and are not required to be exhaustive. Buyers can submit bids for any of the pre-agreed combinations of items. We only consider the case of single-item combinatorial auctions (we mean by this that each item is unique). Participants, inputs and outputs can be described as follows:

Buyers/Bidders: Set of parties interested in placing bids for one or several combinations of items. The set of all buyers is denoted by B.

Sellers: Set of parties interested in selling one or more items in various pre-agreed combinations. The set of all sellers is denoted by A, where A_i is the i-th seller.

Auctioneer (Automated): Party in charge of running the auction. The auctioneer receives the bids and computes an outcome that maximizes the total selling price whilst preserving privacy. In our setting this role is carried out by a set of computational parties e.g. $B \cup A$. These computational parties will execute our protocol, guaranteeing privacy under the non-collusion assumption, i.e. there exist at least as many honest parties as the underlying MPC protocol requires.

Combination Graph: Set of the possible item combinations pre-agreed by the sellers, represented as a graph $G = (V, E)$. Note that we assume for the graph to be acyclic and directed. Each vertex $V_i \quad \forall i \in V \backslash \{s, t\}$ represents a combination of items and each edge E_i signals a possible transition from one vertex (i.e. a set of items) to another. A path on the graph represents the transition from the set of all items (the source s) to $\{\emptyset\}$ (the sink t).

Bids: Each bid contains a secret shared weighted matrix $[U]^p$, whose elements U_{ij} equal the bid price for each desired edge (i, j) and $\top$ in all other locations.

Maximal Path: When the auction is expressed as an acyclic graph, the Maximal Path is the chain of vertices maximizing the seller's profit. The union of combinations represented by these vertices must be a valid sub-set of all items. Note that we can find such path by solving the SSP when the weights are multiplied by -1 [2].

Computation parties: We consider them to be any set of parties executing the protocols. This set could be, for instance $A \cup B$, or any subset of the parties with some stake in the protocol. However, how this set is composed is orthogonal to the problem at hand and could include as many parties as desired.

Figure 3.1 shows an example of this kind of auction modeling for 4 items on 3 combinations.

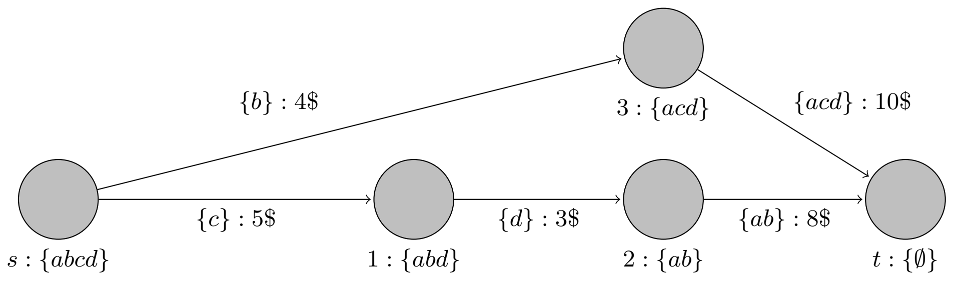

Fig. 1. Restricted topology combinatorial auction modeled as a graph

The protocol to solve restricted topology combinatorial auctions is as follows:

Prerequisites: The sellers make the pre-agreed topology of G available to all bidders in B. Bids are then transmitted to the computational parties in secret shared form.

1. We extend the notation to denote the set of all bid prices (weights) for the i-th edge of G as E_i. The computational parties calculate the maximum bid price for each edge i by calculating $\texttt{max}(E_i)$ and assign it as the secret shared weight associated to the i-th edge. The protocol adds a label $[p]$ to the corresponding edge to mark the origin of the highest bid for this edge.

2. A secret shared unique weighted matrix $[U]$ is produced from the weights selected in the previous step. Each position $[U_{ij}]$ stores the weight and the label $[p]$.

3. The output of the bid is calculated by using [2].In other words, the computational parties obtain a maximum path in G by using the function $[P], [D], [k] \leftarrow \pi_{SP}([U], s, t)$ as a subroutine. The output, in this case $[P]$ is the set of all accepted combinations and $[D]$ the set of all accepted bid prices with their associated $[p]$ label.

4. The shares of $[P]$, $[D]$ and the respective labels $[p]$ are sent to all parties in A and B at the same time for reconstruction.

Note: Integrity, correctness and fairness are guaranteed by the underlying MPC protocols used to implement this functionality and the protocol outlines by [2]. The complexity and security of this basic construction is dominated by the function `maxPath` (implemented by Protocol 1).

The complexity and security of this basic construction are the same as those of the function `maxPath`. Hence, we refer the reader to the complexity analysis provided in Section 2.

4 Computational Experiments

We conducted basic experiments using the open source MPC Framework provided by Aly [30]. This library provides C++ implementations for all basic functionalities of the arithmetic black box abstraction. It uses BGW [26,31] as its basic underlying MPC protocol and the Catrina and Hoogh inequality protocol [32] for comparisons. We use the simple and well known Batcher odd–even mergesort network as a permutation function. The MPC framework [30] is based on NTL [33] and GMP libraries. One shortcoming of the framework is that it separates the computations and the data transmission into two independent threads, thus all internal operations are executed sequentially, hence a multiplication equals one communication round. However, given that the main gains of our protocol are in the multiplicative complexity (amount of work), this allows us to better visualize the performance gains obtained. The framework is secure only against passive adversaries, since it does not implement any Verifiable Secret Sharing (VSS) mechanism. This is issue, however is orthogonal to the security of our protocols.

Our tests simulated a set of three parties with 32-bit inputs. We evaluated two different instances of our protocol: with and without calling the `permute` operation. Thus we were able to measure the overhead caused by the permutation, which is the main adaptation of our protocol. We also measured the execution time of the Dijkstra protocol introduced by [4], which already improved upon [3], for benchmarking. We ran our tests on complete graphs of various sizes, ranging from 4 to 32 vertices. We evaluated our protocols using a 2*2*10-cores Intel Xeon E5-2687 at 3.1 GHz CPUs. The results can be seen in Figure 2.

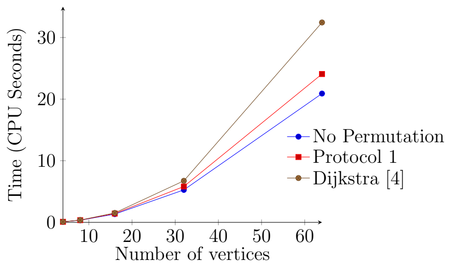

Fig. 2. Dijkstra CPU Times

Our protocol was able to solve a 4-vertex instance in 0.0927 s., whereas it required 5.8 s. for a 32-vertex instance. In contrast, the protocol introduced by Aly and Van Vyve required 0.0950 s. and 6.7 s. respectively. We could observe that a slight overhead caused by the permutation is present, despite of this, our protocol out-perform the state of the art. More specifically, on the overhead, on the 4-vertex instance, the permutation represented an increase by 0.0015s. and an increase by 0.551s. for the 32-vertex instance. As expected, the results show a decrease of computational cost with respect to the state of the art. Finally, we can also see how the cost of disclosing the inputs at the end of the computation (the cost of permuting the rows of the weighted matrix of the graph) has a slight impact on the performance.

5 Conclusions and Future Work

This paper introduces an improved mechanism to solve the SPP in a privacy friendly manner. In this work we achieve quadratic complexity (of the amount of work) by adapting the oblivious Dijkstra protocol proposed by Aly et al. [3,4]. We eliminate the need for expensive vertex extraction mechanisms, at the cost of an oblivious permutation. Additionally, we demonstrate how our protocol can be applied to (topology restricted) combinatorial auctions. Future work includes secretly computing the consensus on the graph configuration and explore other potential applications in financial scenarios as well as in networking and other relevant areas of study in network flows.

Acknowledgements This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070

References

1. de Vries, S., Vohra, R.V.: Combinatorial auctions: A survey. Volume 15., Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, INFORMS (July 2003) 284–309
2. Vangerven, B., Goossens, D.R., Spieksma, F.C.: Winner determination in geometrical combinatorial auctions. Volume 258., Elsevier (2017) 254–263
3. Aly, A., Cuvelier, E., Mawet, S., Pereira, O., Van Vyve, M.: Securely solving simple combinatorial graph problems. In: Financial Cryptography. (2013) 239–257
4. Aly, A., Van Vyve, M.: Securely solving classical network flow problems. In Lee, J., Kim, J., eds.: Information Security and Cryptology - ICISC 2014. Volume 8949 of Lecture Notes in Computer Science., Springer International Publishing (2015) 205–221
5. Li, J., Atallah, M.J.: Secure and private collaborative linear programming. In: Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on. (Nov 2006) 1–8
6. Toft, T.: Solving linear programs using multiparty computation. In Dingledine, R., Golle, P., eds.: Financial Cryptography and Data Security, Berlin, Heidelberg, Springer Berlin Heidelberg (2009) 90–107
7. Catrina, O., de Hoogh, S.: Secure multiparty linear programming using fixed-point arithmetic. In: ESORICS. (2010) 134–150
8. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: ACM CCS. CCS '07, ACM (2007) 498–507
9. Blanton, M., Steele, A., Alisagari, M.: Data-oblivious graph algorithms for secure computation and outsourcing. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. ASIA CCS '13, New York, NY, USA 207–218

10. Wang, X., Nayak, K., Liu, C., Shi, E., Stefanov, E., Huang, Y.: Oblivious data structures. Cryptology ePrint Archive, Report 2014/185 (2014) `http://eprint.iacr.org/`.
11. Liu, C., Huang, Y., Shi, E., Katz, J., Hicks, M.: Automating efficient ram-model secure computation. In: 35th IEEE Symposium on Security and Privacy. (2014)
12. Keller, M., Scholl, P.: Efficient, oblivious data structures for mpc. In Sarkar, P., Iwata, T., eds.: Advances in Cryptology – ASIACRYPT 2014, Berlin, Heidelberg, Springer Berlin Heidelberg (2014) 506–525
13. Damgård, I., Nielsen, J.B.: Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In: CRYPTO. Volume 2729 of LNCS., Springer (2003) 247–264
14. Czumaj, A., Kanarek, P., Kutylowski, M., Lorys, K.: Delayed path coupling and generating random permutations via distributed stochastic processes. SODA '99, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics 271–280
15. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: TCC. (2006) 285–304
16. Toft, T.: Sub-linear, secure comparison with two non-colluding parties. In: Public Key Cryptography. (2011) 174–191
17. Lipmaa, H., Toft, T.: Secure equality and greater-than tests with sublinear online complexity. In: ICALP (2). (2013) 645–656
18. Laur, S., Willemson, J., Zhang, B.: Round-efficient oblivious database manipulation. In Lai, X., Zhou, J., Li, H., eds.: Information Security. Volume 7001 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 262–277
19. Goodrich, M.T.: Randomized shellsort: A simple data-oblivious sorting algorithm. Volume 58., New York, NY, USA, ACM (December 2011) 27:1–27:26
20. Jónsson, K.V., Kreitz, G., Uddin, M.: Secure multi-party sorting and applications. Cryptology ePrint Archive, Report 2011/122 (2011) `https://eprint.iacr.org/2011/122`.
21. Goodrich, M.T.: Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in o(n log n) time. In: Proceedings of the 46th Annual ACM Symposium on Theory of Computing. STOC '14, New York, NY, USA, ACM (2014) 684–693
22. Hamada, K., Kikuchi, R., Ikarashi, D., Chida, K., Takahashi, K.: Practically efficient multi-party sorting protocols from comparison sort algorithms. In: ICISC. (2012) 202–216
23. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology **13**(1) (2000) 143–202
24. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS '01. (2001) 136–145
25. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1993)
26. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC, ACM (1988) 1–10
27. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO. Volume 7417 of LNCS., Springer (2012) 643–662
28. Keller, M., Orsini, E., Scholl, P.: Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. CCS '16, New York, NY, USA, ACM (2016) 830–842

29. Nojoumian, M., Stinson, D.: Efficient sealed-bid auction protocols using verifiable secret sharing. In Huang, X., Zhou, J., eds.: Information Security Practice and Experience. Volume 8434 of Lecture Notes in Computer Science. Springer International Publishing (2014) 302–317
30. Aly, A.: Network Flow Problems with Secure Multiparty Computation. PhD thesis, Universté catholique de Louvain, IMMAQ (2015)
31. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing. PODC '98, New York, NY, USA 101–111
32. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: SCN. (2010) 182–199
33. Shoup, V.: NTL: A library for doing number theory (2001)