Parallel Privacy-preserving Computation of Minimum Spanning Trees

Mohammad Anagreh[1,2], Eero Vainikko[1] and Peeter Laud[2]
[1]*Institute of Computer Science, University of Tartu, Narva maantee 18, Tartu, Estonia*
[2]*Cybernetica, Mäealuse 2/1, Tallinn, Estonia*

Keywords: Privacy-preserving Computation, Multi-party Computation (SMC), Single-Instruction-Multiple-Data (SIMD), Protocol of Reading Private Array, Prim's Algorithm, Minimum Spanning Tree, Sharemind.

Abstract: In this paper, we propose a secret sharing based secure multiparty computation (SMC) protocol for computing the minimum spanning trees in dense graphs. The challenges in the design of the protocol arise from the necessity to access memory according to private addresses, as well as from the need to reduce the round complexity. In our implementation, we use the single-instruction-multiple-data (SIMD) operations to reduce the round complexity of the SMC protocol; the SIMD instructions reduce the latency of the network among the three servers of the SMC platform. We present a state-of-the-art parallel privacy-preserving minimum spanning tree algorithm which is based on Prim's algorithm for finding a minimum spanning tree (MST) in dense graphs. Performing permutation of the graph with sharemind to be able to perform the calculation of the MST on the shuffled graph outside the environment. We compare our protocol to the state of the art and find that its performance exceeds the existing protocols when being applied to dense graphs.

1 INTRODUCTION

A spanning tree of a connected graph is the subgraph that spans it (i.e. it contains all vertices of the graph) and is a tree. For weighted graphs, its Minimum Spanning Tree (MST) is a spanning tree of it, whose sum of edge weights is as small as possible. The MST finding algorithms in graphs are widely used in many fields in computer sciences such as computer networks, maps technology, bioinformatics, and other computations. Privacy-preserving minimum spanning tree algorithms have not been yet studied enough due to the novelty of the technology and high computational cost in secure multiparty computation (SMC) protocols. These protocols provide secure implementations for the *arithmetic black box* (ABB) (Damgård and Nielsen, 2003) abstraction, inside which the privacy-preserving operations are performed without leaking anything about the results of the main and intermediate computations. The private input in the ABB comes from the participants in the computation before starting to perform secure computation. The operations of the ABB use its internal, private memory to store the data during the processing (Laud and Kamm, 2015). Each operation incurs significant latency, due to the communication between the computing nodes of the SMC platform. These features increase the computational costs of the total running time of an application in the SMC platform. This challenge leads to the need for performing the privacy-preserving computation efficiently, reducing the running time while still keeping the computation secure. Especially, the gap of the problem will increase if we apply the protocols for finding a minimum spanning tree to large graphs. In this work, we propose a parallel privacy-preserving minimum spanning tree algorithm for graphs of large size, the algorithm should be fit with SMC protocols, secure and reducing the running time of finding the final results.

In general, finding the minimum spanning tree in privacy-preserving manner proceeds by applying a standard MST algorithm on top of the ABB. Once the input data has been stored in the ABB, its partition among several parties is handled through the protocol set by some way to process the data in parallel to reduce the round complexity of finding the result. There are several standard algorithms for finding a minimum spanning tree regardless of the form of the input data as an adjacency list or matrix. One of the oldest algorithms for finding a minimum spanning forest in the case of a graph that is not connected is Borůvka's Algorithm (Boruvka, 1926). The most well-known MST algorithms are Prim's minimum spanning tree algorithm for weighted undirected graph (Prim, 1957) and Kruskal Algorithm (Kruskal, 1956).

Anagreh, M., Vainikko, E. and Laud, P.
Parallel Privacy-preserving Computation of Minimum Spanning Trees.
DOI: 10.5220/0010255701810190
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 181-190
ISBN: 978-989-758-491-6

In this paper, we will present the implementation of the minimum spanning tree algorithm for the sparse, or dense representations of several graphs on top of state-of-the-art SMC protocol sets. Our implementation for finding the minimum spanning tree is based on Prim's algorithm and the parallel oblivious reading subroutine by Laud (Laud, 2015). We use the Sharemind SMC platform (Bogdanov et al., 2008; Bogdanov et al., 2012b) as the protocol set, which uses secret sharing to get the private input and to represent the intermediate values of the computation. The research contribution is presenting a parallel privacy-preserving algorithm of the Prim's minimum spanning tree algorithm for different kinds of graphs where the number of the vertices and the number of edges are public, but the edges themselves — their end-points and weights — are private. In all of our implementations of the proposed algorithm, we benchmark them and their parts on several graphs with different sizes and we compare our results with previous works.

The organization of the paper is as follows. Section 2 briefly presents the related work. Section 3 gives the background on SMC and its abstraction, the ABB. Section 4 presents the Parallel privacy-preserving Prim's algorithm. Section 5 describes our implementation, gives the benchmarking results, and compares them to related work. The last section 6 concludes the paper, discussing its results and the future work.

2 RELATED WORK

There exists a significant body of work on privacy-preserving graph algorithms using SMC. The privacy-preserving computations allow the private data from many sources to be used in a computation. Indeed, without SMC, parties will avoid sharing their personal data for a number of reasons. Consequently, many researchers have proposed efficient privacy-preserving methods for solving different problems in different fields of computer science such as privacy-preserving shortest path (Aly and Cleemput, 2017; Ramezanian et al., 2018; Wu et al., 2016), privacy-preserving data mining (Mendes and Vilela, 2017; Lindell and Pinkas, 2000; Agrawal and Srikant, 2000; Bogdanov et al., 2012a), privacy-preserving set matching and intersection (Freedman et al., 2004; Saldamli et al., 2019), privacy-preserving statistical data analysis (Bogdanov et al., 2014a) and many others.

There has also been work on optimizing the calculation of finding a minimum spanning tree to reduce the time complexity of the algorithms. Chung et al. (Chung and Condon, 1996) proposed a parallel method for finding a minimum spanning tree based on the sequential version of Borůvka's Algorithm. In their implementation, they used four different kinds of graphs — random graphs, random geometric graphs, structured graphs, and TSP graphs on asynchronous, distributed-memory machines. The method is not fit to be run in privacy-preserving architecture because of the distributed memory.

A simple parallel algorithm for finding a minimum spanning tree for undirected weighted graph $G = (V, E)$ on EREW PRAM is proposed by (Johnson and Metaxas, 1992). The time complexity of their algorithm is $O(\log^{3/2} n)$ using $n + m$ processors, where $n = |V|$ is the number of the vertices and $m = |E|$ is the number of the edges. An important innovation in this algorithm is to extract necessary information about elements without explicitly shrinking components. The algorithm is faster by a factor of $\sqrt{\log n}$ than any deterministic algorithm. The algorithm is designed to be run over the EREW PRAM machine, which is not suitable for huge graphs that occupy a lot of memory, especially if the algorithm is modified to be compatible with the SIMD approach as our algorithm in this paper.

Vineet et al. (Vineet et al., 2009) presented a minimum spanning tree algorithm on Nvidia GPUs under CUDA. The proposed algorithm is a recursive formulation of Borůvka's Algorithm for a huge undirected graph. The size of the graphs they use in the implementation reaches 5 million vertices and 30 million edges. The result shows that the speedup of the algorithm is 50 times over CPU and around 9 times over the best GPU implementation for finding the MST. It is an efficient algorithm that gives the result within 1 second for a huge graph. Another minimum spanning tree algorithm on Nvidia GPU under CUDA is invented (Wang et al., 2010). This algorithm is based on Prim's algorithm using the newly developed GPU-based Min-Reduction data-parallel primitives. The result shows that the speed-up is 2 times on GPU over CPU implementation and 3 times over non-primitive implementation. Both proposed algorithms over GPU are not fit to be run in our Sharemind SMC platform.

A minimum-weight degree-constraint spanning tree algorithm is proposed by Boldon et al. (Boldon et al., 1996). They used a massively-parallel SIMD machine, MasPar MP-1, to implement the four heuristics for approximate solutions to the d-MST problem. The parallel implementation method is designing a suitable PRAM algorithm then implement it directly in the MasPar MP-1. The result shows that the graph with 5000 vertices and 12,5 million edges can be processed in less than 10 seconds.

In (Suraweera and Bhattacharya, 1992), a parallel algorithm for finding a minimum spanning tree for a weighted undirected graph is proposed, the time complexity is $O(\log m)$. The parallel algorithm in this paper is based on the modification of the sequential algorithm in (Suraweera, 1989) and the Klein's algorithm (Klein and Stein, 1990). The implementation using $O(m+n)$ processors is presented for the SIMD machine where m and n are the numbers of edges and vertices respectively. The optimization in this work achieves a speed-up of $O((m \log\log n)/\log m)$

In our work, we are interested in optimizing the calculation of the minimum spanning tree using the SIMD approach in privacy-preserving manner. The motivation is, that the privacy-preserving minimum spanning tree has not been studied enough yet. In (Rao and Singh, 2020), two privacy-preserving minimum spanning tree algorithms in the semi-honest model are proposed. One of them is a privacy-preserving MST algorithm based on Prim's algorithm ,and the other is a privacy-preserving MST algorithm based on Kruskal's algorithm. Both proposed privacy-preserving MST algorithms implemented on top of Yao's garbled circuit protocols (Yao, 1982; Demmler et al., 2015; Liu et al., 2015). The structure of the graph is public but the weights of the edges are private. It would be more secure if the whole structure of the graph would be private, and this is one of the most significant issues in our work.

The privacy-preserving minimum spanning tree algorithm based on the Awerbuch-Shiloach algorithm (Awerbuch and Shiloach, 1987) is proposed by Laud (Laud, 2015). He proposed privacy-preserving protocols to perform in parallel many reads or writes of the elements from the private vectors, according to private addresses. The implementation of the privacy-preserving minimum spanning tree algorithm with protocols of private read or write had not been investigated before. In our paper, we use the same protocols for private read or write for finding the privacy-preserving minimum spanning tree based on Prim's algorithm using the SIMD approach in the Sharemind SMC platform efficiently.

3 PRELIMINARIES

Secure Multi-party Computation (SMC/SMPC) is a cryptographic protocol that enables a group of n-parties with their private inputs $(x_0, x_1, ...x_{n-1})$ to jointly compute a function $(y_0, y_1, ...y_{n-1}) = f(x_0, x_1, ..., x_{n-1})$ and receive the output, without any party learning the private inputs of other parties and the operations of the function. The secure execution of the function f can take place by applying one of the different approaches to SMC, e.g. garbled circuit (Yao, 1982), homomorphic encryption (Henecka et al., 2010) or secret-sharing (Burkhart et al., 2010; Damgård et al., 2009), or a combination of those. We expect the privacy-preserving computation protocols to provide privacy and integrity for the data they operate on, and for the parties of the protocol. To show the existence of these properties in a composable manner, the general-purpose model for the analysis of cryptographic protocols is called the universally composable security (UC) framework (of SMC) (Canetti, 2000).

Universally composable security means that the real protocol is *at least as secure as* an ideal functionality, where the adversary cannot perform any attacks by design and by definition, except for certain unavoidable attacks like failing to respond to messages. One system is at least as secure as another system, if everything that could happen to a user of the first system (or: everything that this user could experience) due to the actions of some adversary interfering with the execution of that system and that user, could also happen to the same user if it uses the second system instead, again taking into account the adversarial activities. If a real protocol is at least as secure as some ideal functionality, then we can build a bigger system on top of that ideal functionality, and analyze the security of that system on the basis of the properties of that functionality; the analysis remains valid if the functionality is replaced by the real protocol (Canetti, 2000).

The Arithmetic Black Box (ABB) is a very useful ideal functionality, abstracting the notion of secure multiparty computation (Laud and Kamm, 2015). ABB allows complex privacy-preserving computations to be securely described without going into the details of protocols and private data representations, i.e. the ABB allows the data and operations to be considered private against the adversaries and the parties. The parties refer to the data stored in the ABB only through *handles*, they cannot access the private data itself inside ABB, even the private data that came from parties or what has been computed from it, i.e. its a black box against parties. Protocol parties can perform some operations with them and return the result by a new handle. In some cases, the protocol parties may need to get some private values stored in ABB. Using a special *declassification* command, parties can get the actual data pointed to by some handle. Note that in privacy analyses, we have to argue that a declassifying command gives no novel information to the computing parties. It's important to note that ABB only responds if it receives the same instruction from all computing parties in the ideal functionality

(Laud and Kamm, 2015).

In our proposed algorithm description below, the privacy-preserving MST algorithm is built on top of an ABB. We will use the common notation $[\![v]\!]$ to denote that value v is stored in ABB, and the notation $[\![\vec{v}]\!]$ to denote the private vector of $(v_1,...,v_n)$ is also stored in ABB, and accessed only through a handle. A write-up of $[\![v]\!]+[\![w]\!]$, or $[\![v]\!]\cdot[\![w]\!]$, or $v\cdot[\![w]\!]$ denotes the invocation of an ABB command (i.e. a cryptographic protocol) to cause the computation of the sum of two private values, or the product of two private values, or the multiplication of a private value with a public value. Our ABB implementation also provides operations to compare two private values, and the *choice*-operation: the expression **if** $[\![b]\!]$ **then** $[\![x]\!]$ **else** $[\![y]\!]$ denotes the selection of either $[\![x]\!]$ or $[\![y]\!]$, depending on the value of the private boolean $[\![b]\!]$. In case some of the operands are vectors, this write-up denotes the invocation of several copies of protocols in SIMD manner.

Regarding the invocation costs of the protocols to perform operations with private values, it is important to note that our representation of private values is homomorphic with respect to linear operations, meaning that the addition of two private values or the multiplication of a private value with a public value requires no communication between the computing parties, and are hence considered to have the complexity 0. In our algorithms below, we also use some higher-level operations for which Sharemind has particularly efficient implementations; these operations can be thought of as extra commands of the ABB (Laud and Kamm, 2015). There exists a private representation of *permutations* of n elements, which can be applied to vectors of length n. We denote the application of a permutation $[\![\sigma]\!]$ to a vector $[\![\vec{v}]\!]$ with $\mathsf{apply}([\![\sigma]\!],[\![\vec{v}]\!])$. It results in a vector $[\![\vec{w}]\!]$, where $w_i = v_{\sigma(i)}$ for all $i \in \{1,\ldots,n\}$. It is also possible to apply the inverse of σ to $\vec{v}$; we denote this operation by $\mathsf{unApply}([\![\sigma]\!],[\![\vec{v}]\!])$. Finally, there is an operation to generate random private permutations. In our implementation, we use the protocols by (Sven et al., 2011) to implement apply and unApply. These implementations work in constant rounds and a linear number of communicated values.

We also use the parallel private read subroutine by (Laud, 2015), which is implemented as a pair of commands. The command $\mathsf{prepareRead}(n,[\![\vec{I}]\!])$ takes as arguments the length of the array from which the reading is done, as well as the indices of the elements that we want to read. This command returns some private data, which is consumed by the command performRead, which also takes the actual array of length n as an argument. The implementation of prepareRead works in logarithmic number (with respect to $n+|\vec{I}|$) of rounds and linearithmic communication. In contrast performRead only requires a constant number of rounds and linear communication, hence the parallel private read subroutine works best if the same set of private indices is used for reading many sets of values, from different vectors.

4 PRIVACY-PRESERVING MINIMUM SPANNING TREE

An *undirected graph* $G=(V,E)$ is a pair of a set of vertices V and edges E, where $E \subseteq V \times V$, and the order of the components of the elements of E does not matter. The edges indicate a two-way relationship among two vertices $u \in V$ and $v \in V$, such that each edge $e=(u,v) \in E$ can be traveled in both directions.

In a *weighted graph*, each edge e in the graph G has been assigned an integer value called *weight* denoted $w(e)$ or $w(u,v)$. A *subgraph* of $G=(V,E)$ is any graph $G'=(V',E')$, where $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. The weight of a (sub)graph is the sum of the weights of all of its edges. A tree is a connected graph with no cycles, i.e. the removal of any edge will disconnect a tree. A *spanning tree* T of an undirected graph G is a subgraph that is a tree that includes all of the vertices of G. Given a weighted undirected graph G, we are interested in finding the minimum spanning tree for the graph, i.e. a spanning tree with minimal possible weight.

An undirected graph $G=(V,E)$ with weighted edges and n-vertices $\{0,1,....,V_{n-1}\}$ can be represented as a data structure in different ways. The *adjacency matrix* of G is a matrix with size $|V| \times |V|$, where the elements of the matrix are the weights $w(u,v)$, where $u \in V$ indexes the rows and $v \in V$ indexes the columns. As the size of the representation is proportional to $|V|^2$, independently of $|E|$, we call it a *dense representation*.

A *dense graph* is a graph $G=(V,E)$ in which the number of edges $|E|$ is close to the maximal number of edges, i.e. $|E| = O(|V|^2)$.

In a *sparse graph*, the number of edges E is close to the possible minimal number of edges for the graph to be connected, $|E| = O(|V|)$. **Prim's Algorithm for MST**. Prim's algorithm maintains two sets of vertices. The first set M (which our privacy-preserving implementation represents as a vector of booleans $\vec{M}$ of length $|V|$, with the value false meaning that the corresponding vertex is included in the set) contains the vertices already included in the MST. The set M is actually public, for reasons that become apparent shortly. For the rest of the vertices, we record in the

vector $[\![\vec{K}]\!]$ the length of the shortest edge that connects them with some vertex in M. At every step of Prim's algorithm, we find the minimum-weight edge between the sets M and $V\setminus M$, and include it in the MST, updating the set M. In the parent vector $[\![\vec{P}]\!]$ (of length $|V|$), we record the tree we are building — for each vertex v, the value $P[v]$ denotes a vertex, such that $(P[v],v)$ was added to the tree when v was an element of $V\setminus M$.

In our algorithm, depicted in Alg. 1, the number of vertices is public, while the adjacency matrix of the graph $\mathbf{G}$ is private. The absence of an edge between two vertices is denoted by some ∞-value that is larger than any length of an actual edge. Besides the adjacency matrix, our algorithm also takes a starting vertex s from which to start building the tree. In general, this vertex can be selected arbitrarily.

Our algorithm starts (lines 2–10) by applying a random, private permutation to hide the identities of the vertices. By permuting the identities of vertices, we will hide the order, in which the vertices will be added to the set M. This is similar to (Aly and Cleemput, 2017) and is performed for similar reasons — to avoid expensive, private data dependent memory accesses in the following steps. In order to permute the vertices, we generate a random private permutation $[\![\sigma]\!]$ of length n. We permute all rows, as well as all columns of $[\![\mathbf{G}]\!]$, using $[\![\sigma]\!]$; all rows can be permuted in parallel, and similarly for columns. We write $\mathbf{G}[u,\star]$ for the u-th row and $\mathbf{G}[\star,v]$ for the v-th column. The last step in the permutation part is finding the new identity of the starting vertex s. For this, we apply the inverse of $[\![\sigma]\!]$ to the identity vector $[0,1,\ldots,n-1]$ (which will be classified in the process), and take its s-th element. This element may be declassified — it is just a random number picked from the set $\{0,\ldots,n-1\}$.

In the main part (lines 11–28) of the algorithm, we first (lines 11–14) set up the working arrays $[\![\vec{K}]\!]$ and $\vec{M}$, as well as the array $[\![\vec{P}]\!]$ that records the parents of each vertex in the MST. We will then run a loop for n iterations, such that in each iteration we add one vertex to the MST. The loop starts (lines 16–22) by looking for the vertex $u' \in V\setminus M$ that is closest to the vertices in M. We create a list $[\![\vec{L}]\!]$ of all vertices not in M, together with their distance from M. Here *NIL* denotes the empty list and cons adds another element (a pair) to the list. We will then use the function minL to find the pair with the smallest first component. The function minL, depicted in Alg. 2, has a completely standard shape; it applies the associative operation to a list in a tree-like manner. After finding the pair with the smallest distance, we take its second component

Data: Number of vertices n, starting vertex s, edge weights $[\![\mathbf{G}]\!]$ (a $n\times n$ array)
Result: Minimum Spanning Tree $[\![\vec{T}]\!]$

```
1  begin
2      [[σ]] ← randPerm(V)
3      forall u ∈ {0,...,n−1} do
4          [[G'[u',⋆]]] ← apply([[σ]], [[G[u,⋆]]])
5      end
6      forall v ∈ {0,...,n−1} do
7          [[G'[⋆,v']]] ← apply([[σ]], [[G'[⋆,v]]])
8      end
9      [[Ī]] ← unApply([[σ]], [0,1,..,n−1])
10     s' ← declassify([[I[s]]])
11     [[K]] ← ∞
12     [[K[s']]] ← 0
13     [[P[s']]] ← s'
14     M ← true
15     for idx = 0 to n−1 do
16         [[L]] ← NIL
17         for i = 0 to n−1 do
18             if M[i] then
19                 [[L]] ← cons(([[K[i]]], [[i]]), [[L]])
20             end
21         end
22         u' ← declassify(second(min([[L]])))
23         M[u'] ← false
24         [[D]] ← [[G'[u,⋆]]]
25         [[C]] ← ([[D]] < [[K]])?
26         [[K]] ← if M ∧ [[C]] then [[D]] else [[K]]
27         [[P]] ← if M ∧ [[C]] then u' else [[P]]
28     end
29     [[R]] ← prepareRead(n, [[Ī]])
30     [[T]] ← performRead([[P]], [[R]])
31     return [[T]]
32 end
```

Algorithm 1: Prim's Algorithm.

— the identity of the vertex. We may declassify this vertex, due to the random permutation on vertex identities that we performed at the beginning.

The main part continues (lines 23–27) by updating the arrays $[\![\vec{K}]\!]$ and $[\![\vec{P}]\!]$. Here $[\![\vec{D}]\!]$ (the reading of which now requires no memory accesses by private addresses) contains the lengths of edges incident to u'. Lines 25–26 compute the conditions to update $[\![\vec{K}]\!]$, and line 27 uses the same conditions to update $[\![\vec{P}]\!]$ for the vertices adjacent to u', and still a part of $V\setminus M$. Note that operations in lines 25–27 are vectorized, which improves the running time of the protocols implementing them.

Data: List of pairs of private values $[\![\vec{w}]\!]$
Result: the element of $[\![\vec{w}]\!]$ with the minimal first component

```
1  begin
2  |  m ← length([[w]])
3  |  if m = 1 then return [[w[0]]]
4  |  begin in parallel
5  |  |  ([[e]], [[i]]) ← minL(left_⌊m/2⌋([[w]]))
6  |  |  ([[f]], [[j]]) ← minL(right_⌈m/2⌉([[w]]))
7  |  end
8  |  if [[e]] ≤ [[f]] then
9  |  return ([[e]], [[i]])
10 |  else
11 |  return ([[f]], [[j]])
12 end
```

Algorithm 2: minL: minimal first component pair.

The last part in the algorithm (lines 29–30) is getting the real value of the MST by applying Laud's protocol (Laud, 2015) for private reading. It makes use of the vector $[\![\vec{I}]\!]$ that contains the original order of the vertices. Then algorithm finds the MST by performing two sub-routines of the protocol, prepareRead and performRead. In detail, for vector $[\![\vec{P}]\!]$ with length n and vector $[\![\vec{I}]\!]$ with same length, then the vector of minimum spanning tree $[\![\vec{T}]\!]$ is given by $\mathsf{performRead}([\![\vec{P}]\!], \mathsf{prepareRead}(n, [\![\vec{I}]\!]))$.

Time Complexity. There are two kinds of communication-related complexities for secure multiparty computation applications. This is the reason why application in secure multiparty computation has high latency. The first kind which is related to the nature of the algorithm and how many times the algorithm is going to be iterated to perform the calculation, is called the *Bandwidth*. The second kind which is responsible for increasing the latency of the calculation is the *Round Complexity*. Let n denote the number of the vertices in the given graph **G**, and m the number of the edges. The structure of private data we use in our algorithm is an adjacency matrix, so the number of the edges m will be no more than $O(n^2)$. The privacy-preserving Prim's minimum spanning tree algorithm requires $O(n^2)$ bandwidth and $O(n \log n)$ rounds (where the size of a single value is assumed to be a constant). Indeed, the initial permutation of vertices takes $O(n^2)$ bandwidth, but only a constant number of rounds (Sven et al., 2011). The main loop is executed n times, each time requiring $O(n)$ bandwidth due to the SIMD-operations on vectors of length n, and $O(\log n)$ rounds due to the function minL. In the end, the private reading operation requires $O(n \log n)$ bandwidth and $O(\log n)$ rounds.

Security and Privacy. Our algorithm is built on top of a universally composable ABB. If there were no declassification statements in the algorithm, then its composition with a SMC protocol set that is a secure implementation of the ABB would inherit the security and privacy properties of that protocol set (Laud and Kamm, 2015). Our algorithm contains declassification statements. Nevertheless, we can state the following security theorem.

Theorem. Suppose that our SMC protocol set implements an ABB for k computing parties that is secure against an active [resp. passive] adversary that corrupts at most t computing parties. Then, an active [resp. passive] adversary that runs in parallel with k parties executing Alg. 1 and this SMC protocol set to implement private computations, corrupting at most t parties, will not learn anything about the inputs to the algorithm beside the number of vertices n of the graph, and the starting vertex s.

Proof. We need to construct a simulator that takes the view of the adversary in the ideal world, and returns the view of the adversary in the real world. The simulator runs a copy of the ideal functionality for the ABB inside it. In the ideal world, the adversary only receives the numbers n and s. In the real world, by corrupting a number of parties, the adversary sees a number of things:

- the inputs n and s;
- the *handles* to the private values, stored in the variables marked with $[\![\cdot]\!]$ in the algorithm;
- the declassified values;
- if the adversary is active: the reactions of the ABB to the attempts of the corrupted parties to deviate from Alg. 1.

It is not necessary to consider the values the real-world adversary sees (through corrupted parties) during the execution of the SMC protocols implementing the ABB, nor the effects of any deviation from these protocols by the corrupted parties — this is taken care of by the composition theorem of the universal composability framework.

The simulator gets the numbers n and s from the ideal-world adversary. It can compute the handles itself, as their values (in contrast to the values they're pointing to inside the ABB) are public. The simulator learns the commands that all k parties submit to the ABB. If a corrupted party deviates from Alg. 1 and its subroutines, the simulator learns this, as well as the reaction of the ABB.

In order to simulate the declassified values, the simulator generates a random permutation of the numbers $0, 1, \ldots, n-1$, releases the first of them at the declassification in line 10, and releases them one

by one (thus repeating the first one) at the declassifications in line 22. Indeed, Alg. 1 randomly permutes the vertices at the beginning, keeping the permutation itself private, hence out of the view of the adversary. Thus the order, in which the vertices are relaxed (i.e. added to the set M) in the main loop, is a random order.

5 RESULT AND EXPERIMENTAL

5.1 Benchmarking Results for Related Work

Recently, a little bit of benchmarking results for privacy-preserving minimum spanning tree algorithms have been already documented. The benchmarking of finding the minimum spanning tree in privacy-preserving computation using a PRAM (Parallel Random Access Machine) algorithm by Awerbuch and Shiloach has been implemented (Laud, 2015). The implementation used the protocols for reading and writing private arrays (same Protocol we use in this paper). He reported running time in time logarithmic to the size of the graph, the number of processors is based on the number of edges in the graph. In detail, a dense graph with 2000 vertices (and 1999k edges) is benchmarked in his work, the running time is more than 10^4 seconds. In the sparse graphs that are used for benchmarking, the number of edges is only 6 times the number of vertices.

In (Rao and Singh, 2020), in sequential implementation, they presented privacy-preserving MST by implementing two algorithms separately, Prim and Kruskal algorithms. There is no real implementation in their work, the time complexity for both algorithms is $O(m \log n)$.

5.2 Privacy-preserving Prim MST Experiments

In this work, we have implemented the parallel Prim's minimum spanning tree algorithm on the Sharemind secure multiparty computation platform. We used the single-instruction-multiple-data instructions supported by the SecreC high-level language (Bogdanov et al., 2014b) to write the code of the implementation using 32-bit integers for weights and vertices of the graphs. The three-party protocol set secure against one passively corrupted party is used among the three computing nodes in Sharemind. The computing nodes are run on a cluster of three computers connected with each other, where each computer is 12-core 3 GHz CPU with Hyper-Threading running Linux and 48 GB of RAM, connected by an Ethernet local area network with a link speed of 1 Gbps. The parallel calculation is done by performing the single-instruction-multiple-data approach, in which the private data of the graph is shaped as vectors in order to perform the calculation for the multiple data in one single instruction. The data of the graphs is represented in secret-shared manner (both the inputs and the outputs) among the three servers of Sharemind.

Table 1: Running time (in seconds) of privacy preserving Prim's algorithm.

G	Vertex	Edge	Perm	Loop	PefR	Total
Sparse	**20**	**75**	0.02	0.2	0.02	**0.24**
	50	**150**	0.08	0.7	0.02	**0.81**
	50	**250**	0.09	0.71	0.02	**0.81**
	50	**1k**	0.08	0.72	0.02	**0.83**
	200	**1k**	0.81	6.07	0.04	**6.91**
	200	**5k**	0.85	6.05	0.04	**6.93**
	1k	**5k**	15.3	110.6	0.1	**126**
	3k	**10k**	136.3	920.1	0.2	**1056.7**
	3k	**15k**	137.1	914.8	0.2	**1052.1**
	3k	**50k**	133.2	920.3	0.3	**1053.8**
	3k	**100k**	136.3	910.3	0.2	**1046.9**
	5k	**20k**	375.4	2478.9	0.4	**2854.7**
	5k	**50k**	371.2	2516.2	0.4	**2887.8**
	5k	**100k**	375.0	2466.0	0.4	**2814.5**
Dense	**50**	**1225**	0.09	0.75	0.02	**0.86**
	100	**4950**	0.23	1.9	0.02	**2.18**
	250	**31.1k**	1.2	8.9	0.04	**10.14**
	500	**124.7k**	4.3	31.3	0.07	**35.67**
	1k	**499.5k**	14.5	107.2	0.1	**121.9**
	2k	**1999k**	61.3	413.6	0.2	**475.1**
	5k	**12497k**	375	2502	0.4	**2879.5**
	10k	**49.9M**	1573	10069	0.9	**11642.9**
Planar	**100**	**200**	0.3	2.1	0.03	**2.43**
	100	**300**	0.25	2.3	0.03	**2.58**
	500	**1000**	4.1	32.2	0.06	**36.36**
	500	**1500**	4.1	32.1	0.06	**36.26**
	1k	**2k**	16.6	111.5	0.1	**128.2**
	1k	**3k**	15.5	109.6	0.1	**125.2**
	2k	**4k**	58.4	418.1	0.2	**476.6**
	2k	**6k**	57.4	416.2	0.2	**473.8**
	3k	**6k**	133.4	913.5	0.2	**1047.2**
	3k	**9k**	136.5	909.2	0.3	**1046.0**
Big	**8500**	**300k**	1.1k	7.1k	0.6	**8.2k**
	9500	**500k**	1.4k	8.8k	0.8	**10.2k**
	10k	**1M**	1.5k	9.8k	0.7	**11.3k**
	20k	**5M**	5.9k	39.1k	2.1	**45.0k**
	20k	**10M**	6.1k	39.6k	2.3	**45.7k**
	30K	**5M**	13.4k	89.8k	3.6	**103.2k**

The execution time of our parallel privacy-preserving prim's minimum spanning tree algorithm depends on the number of the vertices in the graph, the number of the edges has no influence. We report the execution time (in seconds) for running on sev-

Table 2: Total Bandwidth (in MB) of the three servers of Sharemind cluster in running different graphs.

Graph		Server-1		Server-2		Server-3	
Vertex	**Edge**	Time(S)	Band.(MB)	Time(S)	Band.(MB)	Time(S)	Band.(MB)
50	**300**	0.81	10.0	0.81	9.7	0.81	9.9
50	**1225**	0.84	12.1	0.84	11.9	0.84	10.1
100	**1K**	2.21	35.9	2.27	35.0	2.21	34.2
200	**5K**	7.10	129.4	7.10	123.5	7.10	124.5
200	**19.9K**	7.10	128.3	7.10	124.9	7.10	127.0
1k	**2K**	129.9	3554.4	129.9	3075.3	129.7	3183.7
1k	**3K**	130.1	2820.6	130.1	2733.9	130.1	2799.4
1k	**40K**	129.1	2973.1	129.2	2809.7	129.2	2880.5
1k	**499K**	131.2	2887.8	131.2	2798.1	131.2	2863.6
3k	**6K**	1065.7	37064	1065.9	31230	1065.8	29723
3k	**9K**	1069.5	24983	1069.4	24000	1069.7	24770
5k	**1M**	2944	71666	2944	67993	2945	69664
5k	**12.4M**	2982	72636	2981	69055	2983	71023
10k	**49.9M**	11745	346330	11745	304688	11750	312186

eral graphs with different sizes in Table 1. It is important to note that our algorithm uses the adjacency matrix as the representation of the private data of the graphs, i.e. the data structure of the dense graph is used but for the different kinds of the graphs depends on the number of the vertices and edges. Three different kinds of graphs we used in our implementation are sparse, dense, and planar. We split the running time into three groups of the calculation to analyze the real behavior of the algorithm. The three groups are Permutation-operation which is shuffling the rows and columns in the graph before finding the MST, Loop-operation is for finding MST, and the last part performRead-operation for reading the private array.

In the first group of the graphs, we used the data of the sparse graph with dense representation in the implementation. The smallest graph is a graph with 20 vertices and 75 edges, the number of edges is around 3x times the number of vertices. The graphs in the group are based on the number of edges which is given by $m = xn$. The biggest graphs we processed are the graphs where the number of edges is around 20x and 33x times the number of vertices. The result shows that the number of edges has no influence in the running time of the algorithm because of using the SIMD. In the group of dense graphs, the number of edges is given by $n(n-1)/2$, where n is the number of vertices. The result shows our algorithm for finding MST is the fastest algorithm in comparison with a previous algorithm as shown in section 5.1. The third group of the graphs is a planar graph, where the number of the edges is 2 or 3 times the number of the vertices. Big graphs are also implemented in our algorithm, the result shows how our algorithm is efficient for finding MST in the private calculation for big graphs that have up to ten million edges.

Parallel efficiency of the algorithm is better for dense graphs than sparse ones. In Table 2 we present the algorithm's time in seconds and bandwidth consumption results. The bandwidth is measured as the total amount of communication (in and out) in megabytes measured at each server running in parallel with two other servers in the cluster. It can be observed from the table that in general, bandwidth reflects well the measured time values with some slight exceptions. These exceptions can be caused by randomness of the generated graphs having different edges and different weights with given set vertices.

6 CONCLUSIONS

In this work, we have shown how to use the state-of-the-art algorithmic techniques to implement the privacy-preserving version of the classical minimum spanning tree algorithm. We use the Prim's minimum spanning tree algorithm for finding MST for the dense graph representation but for different kinds of graphs. Our implementation is a novel method, its running time has never been achieved before especially for dense graphs.

The size of the input graph plays a big role in the performance of the privacy-preserving minimum spanning tree algorithm, particularly the number of vertices in the graph. Using single-instruction-multiple-data makes the number of edges less significant in the performance of the privacy-preserving MST. We use SIMD instructions as much as possible to restate the edges and vertices in private vectors, particularly in relaxing the edges and in the procedure of finding the minimum first component pair. Also, in

order to keep the privacy of the computation, we use the permutation procedure to mask the real identities of the vertices. The private reading protocol is used to return the real value of the vertices after finished the private calculation.

The future work on parallel privacy-preserving minimum spanning tree algorithms may focus on more parallelization opportunities for minimum spanning tree algorithms especially for sparse representation, not just for sparse data as in this paper. Also, it may include the study of more MST algorithms that may have an algorithmic structure that can be parallelized efficiently to reduce the round complexity more. The ability to use multiple-instruction-multiple-data to reduce the round complexity of the MST algorithm may also be useful.

ACKNOWLEDGEMENT

We would like to express our very great appreciation to Dr.Benson Muite from the institutes of Computer Science at the University of Tartu, for his valuable and constructive suggestions during this work. This work was supported by European Regional Development fund through EXCITE-the Estonian Centre of Excellence in ICT Research.

REFERENCES

Agrawal, R. and Srikant, R. (2000). Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450.

Aly, A. and Cleemput, S. (2017). An improved protocol for securely solving the shortest path problem and its application to combinatorial auctions. Cryptology ePrint Archive, Report 2017/971. https://eprint.iacr.org/2017/971.

Awerbuch, B. and Shiloach, Y. (1987). New connectivity and msf algorithms for shuffle-exchange network and pram. *IEEE Transactions on Computers*, (10):1258–1263.

Bogdanov, D., Jagomägis, R., and Laur, S. (2012a). A universal toolkit for cryptographically secure privacy-preserving data mining. In *Pacific-Asia Workshop on Intelligence and Security Informatics*, pages 112–126. Springer.

Bogdanov, D., Kamm, L., Laur, S., Pruulmann-Vengerfeldt, P., Talviste, R., and Willemson, J. (2014a). Privacy-preserving statistical data analysis on federated databases. In *Annual Privacy Forum*, pages 30–55. Springer.

Bogdanov, D., Laud, P., and Randmets, J. (2014b). Domain-polymorphic programming of privacy-preserving applications. In *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security*, pages 53–65.

Bogdanov, D., Laur, S., and Willemson, J. (2008). Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer.

Bogdanov, D., Niitsoo, M., Toft, T., and Willemson, J. (2012b). High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418.

Boldon, B., Deo, N., and Kumar, N. (1996). Minimum-weight degree-constrained spanning tree problem: Heuristics and implementation on an simd parallel machine. *Parallel Computing*, 22(3):369–382.

Boruvka, O. (1926). On a minimal problem. *Práce Moravské Pridovedecké Spolecnosti*, 3:37–58.

Burkhart, M., Strasser, M., Many, D., and Dimitropoulos, X. (2010). Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. *Network*, 1(101101).

Canetti, R. (2000). Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202.

Chung, S. and Condon, A. (1996). Parallel implementation of bouvka's minimum spanning tree algorithm. In *Proceedings of International Conference on Parallel Processing*, pages 302–308. IEEE.

Damgård, I., Geisler, M., Krøigaard, M., and Nielsen, J. B. (2009). Asynchronous multiparty computation: Theory and implementation. In *International workshop on public key cryptography*, pages 160–179. Springer.

Damgård, I. and Nielsen, J. B. (2003). Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Annual International Cryptology Conference*, pages 247–264. Springer.

Demmler, D., Schneider, T., and Zohner, M. (2015). Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*.

Freedman, M. J., Nissim, K., and Pinkas, B. (2004). Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*, pages 1–19. Springer.

Henecka, W., K ögl, S., Sadeghi, A.-R., Schneider, T., and Wehrenberg, I. (2010). Tasty: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462.

Johnson, D. B. and Metaxas, P. (1992). A parallel algorithm for computing minimum spanning trees. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 363–372.

Klein, P. and Stein, C. (1990). A parallel algorithm for eliminating cycles in undirected graphs. *Information Processing Letters*, 34(6):307–312.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50.

Laud, P. (2015). Parallel oblivious array access for secure multiparty computation and privacy-preserving mini-

mum spanning trees. *Proceedings on Privacy Enhancing Technologies*, 2015(2):188–205.

Laud, P. and Kamm, L. (2015). Stateful abstractions of secure multiparty computation. *Applications of Secure Multiparty Computation. Cryptology and Information Security*, 13:26–42.

Lindell, Y. and Pinkas, B. (2000). Privacy preserving data mining. In *Annual International Cryptology Conference*, pages 36–54. Springer.

Liu, C., Wang, X. S., Nayak, K., Huang, Y., and Shi, E. (2015). Oblivm: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy*, pages 359–376. IEEE.

Mendes, R. and Vilela, J. P. (2017). Privacy-preserving data mining: methods, metrics, and applications. *IEEE Access*, 5:10562–10582.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401.

Ramezanian, S., Meskanen, T., and Niemi, V. (2018). Privacy preserving shortest path queries on directed graph. In *2018 22nd Conference of Open Innovations Association (FRUCT)*, pages 217–223. IEEE.

Rao, C. K. and Singh, K. (2020). Securely solving privacy preserving minimum spanning tree algorithms in semi-honest model. *International Journal of Ad Hoc and Ubiquitous Computing*, 34(1):1–10.

Saldamli, G., Ertaul, L., Dholakia, K., and Sanikommu, U. (2019). An efficient private matching and set intersection protocol: Implementation pm-malicious server. In *Proceedings of the International Conference on Security and Management (SAM)*, pages 16–22. The Steering Committee of The World Congress in Computer Science, Computer ....

Suraweera, F. (1989). A fast algorithm for the minimum spanning tree. *Computers in industry*, 13(2):181–185.

Suraweera, F. and Bhattacharya, P. (1992). A parallel algorithm for the minimum spanning tree on an simd machine. In *Proceedings of the 1992 ACM annual conference on Communications*, pages 473–476.

Sven, L., Jan, W., and Bingsheng, Z. (2011). Round-efficient oblivious database manipulation. In Lai, X., Zhou, J., and Li, H., editors, *Information Security, 14th International Conference, ISC 2011, Xi'an,China, October 26-29, 2011. Proceedings*, volume 7001 of *Lecture Notes in Computer Science*, pages 262–277. Springer.

Vineet, V., Harish, P., Patidar, S., and Narayanan, P. (2009). Fast minimum spanning tree for large graphs on the gpu. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 167–171.

Wang, W., Guo, S., Yang, F., and Chen, J. (2010). Gpu-based fast minimum spanning tree using data parallel primitives. In *2010 2nd International Conference on Information Engineering and Computer Science*, pages 1–4. IEEE.

Wu, D. J., Zimmerman, J., Planul, J., and Mitchell, J. C. (2016). Privacy-preserving shortest path computation. *arXiv preprint arXiv:1601.02281*.

Yao, A. C. (1982). Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE.