Parallel Privacy-Preserving Shortest Paths by Radius-Stepping

Mohammad Anagreh[1,2], Eero Vainikko[1], Peeter Laud[2]
[1]*Institute of Computer Science, University of Tartu, Narva maantee 18, Tartu, Estonia*
[2]*Cybernetica, Mäealuse 2/1, Tallinn, Estonia*
{mohammad.anagreh, eero.vainikko}@ut.ee, peeter.laud@cyber.ee

***Abstract*—The radius-stepping algorithm is an efficient, parallelizable algorithm for finding the shortest paths in graphs. It solved the problem in Δ-Stepping algorithm, which has no known theoretical bounds for general graphs. In this paper, we describe a parallel privacy-preserving method for finding Single-Source Shortest Paths (SSSP). Our optimized method is based on the Radius-Stepping algorithm. The method is implemented on top of the Secure Multiparty Computation (SMC) Sharemind platform. We have re-shaped the radius-stepping algorithm to work on vectors representing the graph in a SIMD manner, in order to enable a fast execution using the secret-sharing based SMC protocol set of Sharemind. The results of the real implementation show an efficient method that reduced the execution time hundreds of times in comparison with a standard case of the privacy-preserving radius-stepping and Δ-Stepping algorithms.**

I. Introduction

Recently, the users of the navigation apps are concerned about location privacy. As well as, the map providers give the mapping information conservatively that might contain sensitive information. This problem opens the door for many researchers to start thinking about finding the shortest paths in privacy-preserving computation considering the performance, especially for using a big graph.

Secret-sharing based secure multiparty computation (SMC) protocols often lead to the most efficient privacy-preserving applications, at least when efficiency is measured in terms of bandwidth. One of the most important challenges in sharing-based protocols is the round complexity, and hence the effects of network latency. The network latency in SMC protocol is a challenge to many researchers for reducing the round complexity of private computation in SMC platform [1], [2]. Many researchers have been solving this problem by parallel computation [3], [4]. In this work, we are interested to perform the privacy-preserving parallel computation for finding the SSSP for a private graph.

The shortest paths problem is one of the most basic problems in different applications using graph-shaped data; the problem will be more complicated in privacy-preserving manner. We have had a special interest in planar graphs, and have been looking for existing SSSP algorithms for planar graphs, of which there exist a few, to be adapted to privacy-preserving computations. The linear-time algorithm for SSSP in planar graphs with a non-negative edge is proposed by Henzinger et al. [5] with time complexity $\mathcal{O}(n)$. Lipton at al. [6], proposed an algorithm for finding the SSSP for a planar graph with arbitrary real-valued edge weights, the time complexity is $\mathcal{O}(n^{3/2})$. The shortest path algorithm for n-vertex planar graph with real-valued weights [7] is proposed with time complexity $\mathcal{O}(n \log^3 n)$. The Δ-stepping algorithm [8] is a SSSP algorithm for arbitrary directed graphs with non-negative edges in the sequential and parallel counterparts with $\mathcal{O}(n \log n + m)$ time complexity. The shortcoming in the Δ-Stepping algorithm is, that the algorithm has no known theoretical bounds on general graphs, bounds can increase n-times in each iteration. The Radius-Stepping algorithm is proposed in [9] to solve the bounds' problem in Δ-Stepping with $\mathcal{O}(m \log n)$ time complexity. It has the best tradeoff between work and depth bounds, by using a variable instead of fixed-size increases of the *radius*, it has a proven bounds on the number of steps. In this work, we adapted the Radius-Stepping algorithm for privacy-preserving computations by reducing the number of general iterations in the algorithm. In our work, the main idea is to use vectors to store the edges and vertices to perform the main operations of the Radius-Stepping in SIMD. We use the Secure Multiparty Computation platform *Sharemind* [10] which is based on secret sharing over arbitrary rings, for our implementation of the Radius-Stepping algorithm in both sequential and parallel fashion. Parallel computation reduces the number of iterations, which will reduce the latency among the three miners (i.e. computing parties) in the Sharemind platform while they run Sharemind protocols [11].

II. Related Work

Recently, privacy-preserving solutions for various kinds of particular computational problems have been proposed, such as privacy-preserving minimum spanning tree [3], privacy-preserving data mining [12], statistical analysis [13], [14]. Researchers have been conducting some research in privacy-preserving shortest path to keep the computation secure. Ramezanian et al. [15] proposed a privacy-preserving all-pairs shortest path (APSP) algorithm. The proposed algorithm is an extended version of the Floyd-Warshall algorithm and it is used in a novel protocol they proposed, that enables privacy-preserving path queries on directed graphs. The time complexity in the extended version will increase with the latency in the SMC platform. In [16], the authors proposed a new algorithm for privacy-preserving APSP and another

algorithm for the SSSP. They also proposed two new algorithms for the privacy-preserving set union. The algorithms are used in the privacy-preserving shortest path in the semi-honest module. The cryptographic version of privacy-preserving data mining is closely related to the technique in this paper. In general, the work is efficient, but the work is presented in the sequential fashion that will increase the latency in privacy-preserving computations. An efficient protocol for privacy-preserving shortest paths computing for navigation is proposed in [17]. The cryptographic protocol for navigation on city streets that provides privacy for both the service provider's routing data and client's location in the street.

In our work in this paper, we focus on optimizing the single-source shortest path algorithm (Radius-Stepping algorithm) and applying the same optimized algorithm in privacy-preserving computations. Our protocols are *compositional*, i.e. we expect that the components of the graph have already been secret-shared between the computing parties of the SMC protocol. The result is likewise delivered in secret-shared manner, and can be used in subsequent steps of a larger application.

III. Preliminaries

A. Secure Multiparty Computation

The cryptographic techniques assure the security and the integrity of the storage and the communication between the participants, even while the adversary has taken over a fraction of them. The general-purpose model for the analysis of cryptographic protocols with strong security properties is called the framework of universal composability [18].

The framework of universal composability considers a set of interacting Turing machines running in parallel and sending data among them, including the adversarial Turing machines. The framework states when a set of Turing machines *securely implements* another set — this happens when any behavior observed by an *environment* machine while interacting with the first set can be matched by the second set. The value of the framework is in the composition theorem. A protocol Π may securely implement an ideal functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. If Ξ is another protocol that securely implements the functionality $\mathcal{F}$, then the composition of Π and Ξ securely implements $\mathcal{G}$.

The Arithmetic Black Box (ABB) is an ideal functionality $\mathcal{F}_{\mathcal{ABB}}$ that performs calculations with private data. It allows parties to store the private data handed over to it, performs the operations based on users' instructions, and sends certain values back to users if a sufficient number of them request it. Let us suppose, a party sends a command $\mathbf{store}(v)$ to the ideal functionality to do some calculation, where v is some value. The functionality $\mathcal{F}_{\mathcal{ABB}}$ receives and stores the value v, then assigns a fresh *handle* h to that value by storing the pair (h, v). Finally, the ideal functionality sends h to all parties. To perform the computations without revealing any knowledge about the intermediate result, the ABB waits for a command $(\mathsf{perform}, op, h_1, \ldots, h_k)$ from all (or sufficiently many) computing parties. It looks up the values $v_1, \ldots, v_k$ stored with the handles $h_1, \ldots, h_k$, applies the operation op on them, obtaining a value v, stores it under a new handle h, and returns h to all parties. To learn a value stored under the handle h, all (or sufficiently many) parties send the command $(\mathsf{declassify}, h)$ to the ABB, which then looks up the pair (h, v) and responds with v [19]. The protocol sets that may be used to securely implement an ABB may be based on either threshold homomorphic encryption [20], garbled circuits [21], or secret sharing [10]. These implementations can offer protection against an honest-but-curious or malicious party. In our algorithms built on top of an ABB, we use the notation $[\![v]\!]$ to denote that v is a value that is stored inside the ABB, and accessed only through a handle. Any operations performed with such values will take place by having commands issued to the ABB.

B. Radius-Stepping Algorithm

Our contribution in this paper is to optimize the Radius-Stepping algorithm to run efficiently with reduced time complexity. The algorithm finds the SSSP for the weighted undirected graph $\mathbf{G}$, where the graph is represented in the adjacency matrix as input to the algorithm. The Radius-Stepping algorithm and Δ-Stepping algorithms have the same basic structure, both algorithms are a hybrid algorithm of the Dijkstra and Bellman-Ford algorithms. The steps of the Bellman-Ford algorithm are inner operations in the Dijkstra algorithm. The Dijkstra part visits vertices in increasing distance from the source s and settling each vertex $v \in V$ in $\mathbf{G}$. Instead of visiting one vertex v at a time, Radius-Stepping visits the vertices in steps. The algorithm increases the radius centered at vertex s from d_{i-1} to d_i, and all vertices $v \in |V|$ in the annulus $d_{i-1} < d(s, v) < d_i$ will be settled. The Bellman-Ford parts (repeat to until) performs the settling of the vertices in sub-steps. In Δ-Stepping algorithm, the radius will be increased by a fixed amount in each step in the round distance $d_i = d_{i-1} + \Delta$. This increase can require $\Theta(n)$ sub-steps in the worst case. As well as, in some cases, it requires

Data: $G = (V, E)$, vertex radii $r(\cdot)$, source vertex s
Result: The graph distance $\delta(\cdot)$ from s
begin
  $\delta(\cdot) \leftarrow +\infty$, $\delta(s) \leftarrow 0$
  foreach $v \in N(s)$ **do**
    $\delta(v) \leftarrow w(s, v), S_0 \leftarrow \{s\}, i \leftarrow 1$
  while $|S_{i-1}| < |V|$ **do**
    $d_i \leftarrow min_{v \in V \setminus S_{i-1}}\{\delta(v) + r(v)\}$
    repeat
      foreach $u \in V \setminus S_{i-1}$ *s.t* $\delta(u) \leq d_i$ **do**
        foreach $v \in N(u) \setminus S_{i-1}$ **do**
          $\delta(v) \leftarrow \min\{\delta(v), \delta(u) + w(u, v)\}$
        end
      end
    until *no* $\delta(v) \leq d_i$ *was updated*
    $S_i = \{v \mid \delta(v) \leq d_i\}$
    $i = i + 1$
  end
  return $\delta(\cdot)$
end

Algorithm 1: Radius-Stepping

$\mathcal{O}(mn)$ work because each sub-step will work on the same set of vertices and their edges. In the Radius-Stepping algorithm, this problem is solved efficiently, by proposing a new round distance d_i in each round.

IV. Privacy-Preserving Shortest Paths

In this section, we describe our algorithm for parallel privacy-preserving SSSP, called SIMD-RADIUS-STEPPING, and present its performance analysis. The SIMD-Radius-Stepping algorithm is presented in Algorithm 2. In details, the algorithm works as follows: The input to the algorithm is a weighted, undirected graph $\mathbf{G} = (V, E)$ presented by an adjacency matrix, source vertex s, and the value of the radius for every vertex in the graph, given as a function $r : V \to \mathbb{R}^+$. In general, the algorithm has the same basic structure as the Radius-Stepping algorithm, but we represent the algorithm using vectors to let the algorithm perform the calculation in SIMD and in privacy-preserving manner.

Data: Number of vertices and edges n, m
Data: Adjacency matrix $[\![\mathbf{G}]\!] \in \mathbb{Z}^{n\times n}$
Data: Vertex radii $[\![\vec{r}]\!] \in \mathbb{N}^n$, source vertex s
Result: The distances $[\![\vec{\delta}]\!]$ from s
begin
  $[\![\vec{S}]\!] = false$ // length of $\vec{S}$ is n
  $[\![\vec{\delta}]\!] \leftarrow [\![\mathbf{G}[s, \star]]\!]$
  $[\![S[s]]\!] \leftarrow true$, $[\![\delta[s]]\!] \leftarrow 0$, $[\![D]\!] \leftarrow 0$
  repeat
    $[\![\vec{\delta'}]\!] \leftarrow$ **Bellman-Ford-Step**$(n, [\![\mathbf{G}]\!], [\![\vec{S}]\!], [\![\vec{\delta}]\!], [\![D]\!])$
    $[\![B]\!] \leftarrow ([\![\vec{\delta'}]\!] \neq [\![\vec{\delta}]\!])$ // a private Boolean
    $[\![\vec{\delta}]\!] := [\![\vec{\delta'}]\!]$
    $[\![\vec{S'}]\!] \leftarrow ([\![\vec{\delta}]\!] \leq D) :\in \mathbf{kind}(true, false)$
    $[\![D']\!] \leftarrow \mathbf{min}(\mathbf{choose}([\![\vec{S}]\!], [\![\infty]\!], [\![\vec{\delta}]\!] + [\![\vec{r}]\!]))$
    $[\![\vec{S}]\!] := \mathbf{choose}([\![B]\!], [\![\vec{S}]\!], [\![\vec{S'}]\!])$
    $[\![D]\!] := \mathbf{choose}([\![B]\!], [\![D]\!], [\![D']\!])$
  until declassify$(\bigvee \neg[\![\vec{S}]\!])$
  return $\delta(\cdot)$
end

Algorithm 2: SIMD-Radius-Stepping

We store the set S (Alg. 1) as a vector of private Booleans $[\![\vec{S}]\!]$, and the current mapping δ as a vector of private integers $[\![\vec{\delta}]\!]$. The algorithm starts by initializing the Boolean vector $[\![\vec{S}]\!]$ with $false$-value, and the vector of graph distance $[\![\vec{\delta}]\!]$ with weights of the source vector s with all vertices $v \in V$. As well as, the first element in vector $[\![\vec{S}]\!]$ and $[\![\vec{\delta}]\!]$ is the source vertex s by $true$-value and by 0-value, respectively. The algorithm has one repeat-until loop, started by finding the shortest path for one step by Alg. 3, until settling all vertices with radius less than D. The Bellman-Ford-step algorithm starts by reshaping the elements inside a vectors for $[\![\vec{\delta}]\!]$ and $[\![\vec{S}]\!]$ into row and column vectors. Later, the algorithm updates the $[\![\vec{\delta'}]\!]$ by finding the minimum values of the $[\![\overrightarrow{\delta_B}]\!]$ which is the shortest path for each vertex in the graph. The updating in the vector $[\![\overrightarrow{\delta_B}]\!]$ is by either finding the minimum weights of elements in the $[\![\overrightarrow{\delta_{Row}}]\!]$ and the summation of the elements in $[\![\overrightarrow{\delta}_{Col}]\!]$ with edges' weights of the graph $[\![\vec{G}]\!]$, or by getting the elements of the $[\![\overrightarrow{\delta_{Row}}]\!]$. The updating of the $[\![\overrightarrow{\delta_B}]\!]$ by the elements of the vector $[\![\overrightarrow{M}]\!]$ is based on satisfying the three conditions of bounding the number of sub-steps as presented in Alg. 3. The operations of the Bellman-Ford-Step algorithm is organized to be performed by single-instruction-multiple-data, using SIMD decreases the round complexity of performing the Bellman-Ford sub-steps. To keep tracing over the whole vertices in the graph is by applying the remain operations in the algorithm.

The private Boolean B is to save the comparison result between the vector $[\![\vec{\delta}]\!]$ and $[\![\vec{\delta'}]\!]$. The vector of visited set $[\![\vec{S'}]\!]$ in Alg. 2 represents the condition in the repeat-until in Alg.1. The vertices which are not visited yet will be stored in the vector $[\![\vec{S}]\!]$ after finding the minimum of the $[\![\vec{\delta}]\!]$ + $[\![\vec{r}]\!]$. The last operations in the algorithm are updating the vector of set $[\![\vec{S}]\!]$ and the value of D; the updating is based on the state of the Boolean B.

Data: Number of vertices n
Data: Adjacency matrix $[\![\mathbf{G}]\!] \in \mathbb{Z}^{n\times n}$
Data: Current values of $[\![\vec{S}]\!]$, $[\![\vec{\delta}]\!]$, $[\![D]\!]$ in Alg. 2
Result: The updated distance $[\![\vec{\delta'}]\!]$ from s
begin
  for *every vertex* $u \in \{0, 1, \ldots, n-1\}$ **do**
    for *every vertex* $v \in \{0, 1, \ldots, n-1\}$ **do**
      $[\![\mathbf{S}_{Row}[u, v]]\!] \leftarrow [\![S[u]]\!]$
      $[\![\mathbf{S}_{Col}[u, v]]\!] \leftarrow [\![S[v]]\!]$
      $[\![\boldsymbol{\delta}_{Row}[u, v]]\!] \leftarrow [\![\delta[u]]\!]$
      $[\![\boldsymbol{\delta}_{Col}[u, v]]\!] \leftarrow [\![\delta[v]]\!]$
    end
  end
  $[\![\overrightarrow{con}]\!] \leftarrow (\neg[\![\mathbf{S}_{Row}]\!])\&(\neg[\![\mathbf{S}_{Col}]\!])\&([\![\boldsymbol{\delta}_{Col}]\!] \leq [\![D]\!])$
  $[\![\boldsymbol{\delta}_{\mathrm{B}}]\!] \leftarrow \mathbf{choose}([\![\overrightarrow{con}]\!], \mathbf{min}([\![\boldsymbol{\delta}_{Row}]\!], [\![\boldsymbol{\delta}_{Col}]\!] + [\![\mathbf{G}^{\mathrm{T}}]\!]), [\![\boldsymbol{\delta}_{Row}]\!])$
  forall $u \in \{0, \ldots, n-1\}$ **do**
    $[\![\delta'[u]]\!] \leftarrow \mathbf{min}\{[\![\boldsymbol{\delta}_{\mathrm{B}}[u, \star]]\!]\}$
  end
  return $[\![\vec{\delta'}]\!]$
end

Algorithm 3: Bellman-Ford-Step

Time Complexity is based on the radius r. The radius r is used to determine the number of the steps and sub-steps in the calculation. The number of the iterations is based on the number of the visited vertices ($\forall S \in V$). There are three cases of the radii r. In the case that $r(v) = 0$, the time complexity of the algorithm is $\leqslant \mathcal{O}(n + \log m)$. If $r(v) = \infty$, the time complexity is $\leqslant \mathcal{O}(\log n)$. As well as, the time complexity is $\leqslant \mathcal{O}(\log n + \log m)$ if the $r(v) = \delta$.

Security and privacy of Alg. 2 follows from the discussions in [19]. As long as the algorithms built on top of the ABB do not declassify anything, the algorithm retains the security and privacy properties of the underlying implementation of the ABB, including its resistance against semi-honest or malicious adversaries. Alg. 2 does contain declassification statements, and their outcomes cannot be directly deduced from the public parameters of the graph. We leak the number of iterations made in Alg. 2, this number depends on a multitude of parameters, including the structure of the graph and the used radii. It is probably possible to minimize this leakage, while still

TABLE I
RUNNING TIMES (IN SECONDS) OF PRIVACY-PRESERVING SIMD-RADIUS-STEPPING ALGORITHM FOR SPARSE, DENSE AND UNWEIGHTED GRAPHS

Weighted Graph (Sparse)					Weighted Graph (Dense)					Unweighted Graph				
n	m	Serial	Parallel	Speed-up	n	m	Serial	Parallel	Speed-up	n	m	Serial	Parallel	Speed-up
500	6k	13062	203.9	64x	50	1225	74.6	0.6	124x	50	1225	71.8	0.2	359x
500	10k	35346	186.5	189x	75	2775	247.7	1.2	206x	100	2k	357	0.7	510x
900	20k	77580	825	94x	100	4950	593.9	1.5	395x	200	19.9k	4769	4.3	1109x
1k	40k	–	920	–	300	44850	16059	19.4	827x	500	20k	26372	14.8	1781x
2k	40k	–	10206	–	500	124.7k	74453	78.6	947x	1k	20k	–	75.9	–
5k	5M	–	14.2k	–	1k	499.5k	–	188.4	–	5k	12.4M	–	2304	–
10k	15M	–	91.3k	–	10k	49.9M	–	48.9k	–	10k	49.9M	–	9087	–

obtaining the benefits of iterating only as long as something is changing, by experimentally determining a good upper bound of the iterations, and always performing at least this number of iterations in Alg. 2.

V. BENCHMARKING RESULTS

Recently, some, but not too many benchmarking results for privacy-preserving shortest path algorithms have been reported. The benchmarking of the SSSP algorithms, Dijkstra and Bellman-Ford have been reported by Aly et al. [23]. For Dijkstra's algorithm on 128 vertices (dense graph), they reported a run time around an hour, while for Bellman-Ford algorithm is around 8 hours on 64 vertices. As well as, they reported 20 seconds for Dijkstra's algorithm with a 64-vertex graph. The operations of oblivious RAM (ORAM) on top of the SPDZ protocol set [24] are implemented by Keller et al. [25], the protocol used to implement a privacy-preserving Dijkstra's algorithm. A sparse graph with 2000 vertices implemented, running time in a couple of minutes. For a dense graph with 500 vertices, the running time is a couple of hours. The garbled circuits are used to privately evaluate Dijkstra's algorithm [26]. They reported running times of ca. 15 minutes for 100-vertex graphs (their parallel implementation handles 32 circuits at the same time on a 32-core server). As well as, garbled circuits are used to evaluate Dijkstra's algorithm on sparse graphs by Liu et al. [27], they employed oblivious priority queues to increase efficiency. The result in the implementation is estimated that together with JustGarble [28], they used a graph with 1000-vertex and 3000-edge, the execution time is around 20 minutes.

In this work, we have implemented the proposed algorithms on Sharemind SMC platform for both versions of the algorithm, sequential and SIMD. Sharemind benchmarks are run on a cluster of three computers connected with each other with a three-party protocol set secure against one passively corrupted party; SecreC [22] high-level language is used to write the codes of the implementations. The servers in the cluster have a 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 implementation of Sharemind does not make use of multiple cores. Private values are represented by additively sharing them among the three servers over the ring $\mathbb{Z}_{2^{32}}$.

We use different types of weighted undirected graphs in our implementation, sparse and dense with varying sizes. We also used unweighted graphs with different sizes in the implementation. We report the running time (in seconds) of the privacy-preserving SIMD-Radius-Stepping algorithm in the Table I for several sparse, dense and unweighted graphs with different sizes. The result shows that the speed-up is increased by increasing the size of the input graph, the speed-up of the SIMD-Radius-Stepping algorithm is scalable.

TABLE II
RUNNING TIMES OF SIMD-RADIUS-STEPPING FOR PLANAR-LIKE GRAPHS

Graph	Vertex	Edge	DELTA-S	STAND-RS	SIMD-RS
$\|E\| = 2\|V\|$	20	40	2.28	1.62	0.24
	40	80	6.84	8.4	0.54
	80	160	89.2	54.6	4.6
	120	240	254	170	13.4
$\|E\| = 2.5\|V\|$	20	50	5.28	1.74	0.24
	40	100	20.7	9.42	0.84
	80	200	113.7	56.7	5.6
	120	300	296.1	178.5	9.48
$\|E\| = 4\|V\|$	20	80	10.1	2.28	0.24
	40	160	30.96	10.7	0.9
	80	320	109.2	66.8	3.78
	120	480	360	192	9.5

One of the most impressive uses of our proposed algorithm is finding the privacy-preserving shortest path for dense graphs. The results show that the speed-up for parallel implementation of the dense and unwighted graph may be in hundreds of times. This is the motivation of our work, that by using our SIMD-Radius-Stepping algorithm, we can find the shortest path for a huge graphs in privacy-preserving computation in less running time in comparison with the running time of the serial version of the algorithm.

Another series of tests that we report in Table II considers graphs whose number of edges (for the given number of vertices) is similar to planar graphs. In this series, we implemented three algorithms for privacy-preserving single-source shortest path algorithms: the Radius-Stepping and Δ-Stepping algorithms, and our SIMD-Radius-Stepping algorithm. There are three groups of the graphs that have been benchmarked. The groups are based on the ratio between the number of edges and the number of vertices of the graph.

Algorithm 2 has a declassification statement in a location that causes some details of the graph to be leaked through the running time of the algorithm. The running time is characterized by the time it takes to run a single iteration of that algorithm (which only depends on the number of vertices and edges of the graph), and the number of iterations the algorithm

does (which depends on the structure of the graph, as well as on the radius $\mathbf{r}$). In Table III, we report the average number of iterations for random weighted graphs with given number of vertices and edges, for three possible choices of the radii — either infinite, or zero, or randomly generated. The average execution time has been reported, too. We see that for these choices, the first choice consistently beats the others.

TABLE III
RUNNING TIMES OF SIMD-RADIUS-STEPPING WITH RADII CASES

Graph	Size		$\mathbf{r} = \infty$		$\mathbf{r} = 0$		$\mathbf{r} = rnd$	
	n	m	Iter	Time	Iter	Time	Iter	Time
Sparse	50	400	6	0.5	44	3.9	8	0.66
	100	2k	5	1.3	41	9.8	9	2.1
	200	2k	9	8.1	100	94.5	20	17.3
	500	20k	6	29.4	121	597.1	27	134.9
	1k	200k	6	106.5	69	1266	19	451.1
Dense	50	1225	5	0.4	24	2.1	6	0.5
	100	4950	5	1.1	25	6.1	8	1.9
	200	19.9k	5	4.5	27	22.8	8	8.1
	500	124.7k	5	23.6	31	153.4	10	49.1
	1k	499.5k	5	87.8	38	690	15	256
Planar-like	50	150	8	0.7	70	6.2	10	0.8
	100	200	9	2.3	165	40.1	23	5.8
	100	300	9	2.1	155	38.9	21	5.0
	200	400	13	11.1	380	276.6	40	43.6
	200	600	11	9.8	251	260.5	41	35.1
	500	1500	13	63.5	691	3420	100	489.6

VI. CONCLUSION

We have presented the parallel technique of finding the SSSP in the privacy-preserving computation using SIMD as much as possible to reduce the round compliexity. The running times show that the proposed method is faster than Radius-Stepping, and Δ-Stepping algorithm. In particular, our implementation of a parallel single-source shortest path is the faster, compared to previous work. Future work is studying different shortest path algorithms using SIMD.

ACKNOWLEDGEMENT

This work was supported by European Regional Development fund through EXCITE-the Estonian Centre of Excellence in ICT Research, and by ETAG through grant PRG920.

REFERENCES

[1] J. Katz, R. Ostrovsky, A. Smith. *Round efficiency of multi-party computation with a dishonest majority*. In International Conference on the Theory and Applications of Cryptographic Techniques, (pp. 578-595). Springer, Berlin, Heidelberg, 2003.

[2] J. Katz, C.Y. Koo. *Round-efficient secure computation in point-to-point networks*. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 311-328). Springer, Berlin, Heidelberg, 2007.

[3] P. Laud, *Parallel Oblivious Array Access for Security Multiparty Computation and Privacy Preserving Minimum Spanning Trees.* Proc. on Privacy Enhancing Technologies (2): pp.188-205, 2015.

[4] E. Boyle, K.M. Chung, and R. Pass. *Large-scale secure computation: Multi-party computation for (parallel) RAM programs*. In Annual Cryptology Conference (pp. 742-762). Springer, Berlin, Heidelberg, 2015.

[5] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian. *Faster shortest-path algorithms for planar graphs.* J. Comput. Syst. Sci., 55(1):3-23, 1997.

[6] R. J. Lipton, D. J. Rose, and R. E. Tarjan. *Generalized nested dissection.* SIAM Journal on Numerical Analysis, 16: pp.346–358, 1979.

[7] J. Fakcharoenphol and S. Rao. *Planar graphs, negative weight edges, shortest paths, and near linear time.* J. Comput. Syst. Sci., 72(5):868-889,2006.

[8] U. Meyer and P. Sanders. Δ-*stepping: a parallelizable shortest path algorithm.* Journal of Algorithms, 49(1), pp.114-152, 2003.

[9] G. Blelloch, Y. Gu, Y. Sun and K. Tangwongsan. *Parallel shortest paths using radius stepping*. In Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, pp. 443-454, 2016.

[10] D. Bogdanov, S. Laur, and J. Willemson. *A framework for fast privacy-preserving computations*. In S. Jajodia and J. Lopez, editors, ESORICS, volume 5283 of Lecture Notes in Computer Science, p. 192-206. Springer, 2008.

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

[12] D. Bogdanov, R. Jagomägis and S. Laur. *A universal toolkit for cryptographically secure privacy-preserving data mining.* In Pacific-Asia Workshop on Intelligence and Security Informatics, pp. 112-126. Springer, Berlin, Heidelberg, 2012.

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

[14] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk and R. Talviste. *Students and taxes: a privacy-preserving study using secure computation.* Proceedings on Privacy Enhancing Technologies, (3), pp.117-135, 2016.

[15] S. Ramezanian, T. Meskanen, and V. Niemi. *Privacy Preserving Shortest Path Queries on Directed Graph*. In 2018 22nd Conference of Open Innovations Association (FRUCT),IEEE, pp.217-223, 2018.

[16] J. Brickell and V. Shmatikov. *Privacy-preserving graph algorithms in the semi-honest model—*. In International Conference on the Theory and Application of Cryptology and Information Security, Springer, Berlin, Heidelberg, pp.236-252, 2005.

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

[18] R. Canetti. *Universally composable security: A new paradigm for cryptographic protocols*. In FOCS, pages 136- 145. IEEE Computer Society, 2001.

[19] P. Laud. *Stateful abstractions of secure multiparty computation*. Peeter Laud and Liina Kamm (eds.), Applications of Secure Multiparty Computation. IOS Press, Cryptology and Information Security, 13, pp.26-42, 2015.

[20] I. Damgård and J. B. Nielsen. *Universally composable efficient multiparty computation from threshold homomorphic encryption.* In D. Boneh, editor,CRYPTO, volume2729 of Lecture Notes in Computer Science, pp.247-264.Springer, 2003.

[21] D. Demmler, T. Schneider, and M. Zohner. *ABY - A frame-work for efficient mixed-protocol secure two-party computation.* In 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California,USA, February 8-11, 2014. The Internet Society, 2015.

[22] D. Bogdanov, P. Laud and J. Randmets. *Domain-polymorphic programming of privacy-preserving applications.* In Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security, pp. 53-65, 2014.

[23] A. Aly, E. Cuvelier, S. Mawet, O. Pereira, and M. Vyve. *Securely Solving Simple Combinatorial Graph Problems.* In International Conference on Financial Cryptography and Data Security, pp.239-257. Springer, Berlin, Heidelberg, 2013.

[24] I, Damgård, V. Pastro, N. Smart and S. Zakarias. *Multiparty computation from somewhat homomorphic encryption.* In Annual Cryptology Conference, pp.643-662. Springer, Berlin, Heidelberg, 2012.

[25] M. Keller and P. Scholl. *Efficient, oblivious data structures for MPC.* In International Conference on the Theory and Application of Cryptology and Information Security. Springer, pp.506-525, 2014.

[26] H. Carter, B.Mood, P. Traynor and K. Butler. *Secure outsourced garbled circuit evaluation for mobile devices.* Journal of Computer Security, 24(2), pp.137-180, 2016.

[27] C. Liu, X.S. Wang. K. Nayak. Y. Huang and E. Shi. *Oblivm: A programming framework for secure computation.* In 2015 IEEE Symposium on Security and Privacy, pp.359-376. IEEE, 2015.

[28] M. Bellare, V.T. Hoang, S. Keelveedhi and P. Rogaway. *Efficient garbling from a fixed-key block cipher.* In 2013 IEEE Symposium on Security and Privacy, pp.478-492. IEEE, 2013.