Middle East Technical University
Department of Computer Engineering
Wireless Systems, Networks and Cybersecurity (WINS) Laboratory

Style Guide for Writing Technical Reports

CENG519 Network Security
2021-2022 Spring
Term Project Report

Prepared by
Burak Bolat
Student ID: e2237097
burak.bolat@metu.edu.tr
Computer Engineering
12 June 2022

Abstract

Homomorphic Encryption (HE) became a requirement for secure and private outsourced computation of concealed sensitive data. Although it provides privacy, it comes with an inevitable heavy load. Briefly, the workflow of HE is encrypting sensitive data which will be used in outsourcing computation without decoding and decrypting the result of the computation. In this study, CKKS, which is an approximate homomorphic encryption scheme, is used for computation. Graphs are used in numerous manners in computer science from connectivity of people in a social medium to representation of molecules. For the cases where the input is sensitive like the topology of person's related people or the topology of GSM stations, sending the connectivity information without encryption is a vulnerability. On the other side, analysis on a server is demanded. For instance, a GSM cluster desires to compute availability check of each node and a node knows only its' neighbour nodes. There should be a server that receives parts of the graph from nodes and computes connectivity. In such case, each graph encrypts its adjacency for sending to server and server checks the connectivity. There are various circumstances parallel to the instance. This study aims to compute the number of non-connected nodes for a homomorphically encrypted non-directed graph. I present two approaches, namely search-based and path length-based, and analysis for them. For each approach, analysis mainly includes different topologies and computation time for HE processes. This study also proposes interfaces for client and server since overall computation requires large scales of polynomials.

Table of Contents

List of Figures

1 Introduction

Outsource computation and distributed computation are attracting attention with the demand from many field from medicine to communication. The main concern is confidentiality of the data. For instance, revealing one's healt status or disease makes attackers motivate to abuse that information. At the first sight, reveal of the information is not ethical, but, the information may be used to sell medicament or for the worser cases, it allows threathening the owner of information. Examples have a wide range for various sensitive data such as topology of network, trained machine learning model, dataset of machine learning.

Homomorphic Encryption (HE) is designed to protect privacy of the sensitive data. It provides user to perform computations without decrypting it. At the first step, the data is encrypted with the public key of the user so that receiver can authenticate the sender and can not decrypt the data. At the server side, computation are performed and result is returned encrypted. When the owner receives the result, it can be decrypted with private key. While there are exact and approximate approaches of HE [1], CKKS [2], an aproximate polynomial based scheme, is used for encrypted operations.

In this work, a graph algorithm to process encrypted adjacency list is examined. To motivate, graph algorithms have wide range of usage that can be listed as: financial services, manufacturing, data regulation, marketing and machine learning [3]. Number of non-reachable node (NoNRN) problem is the main concern of the paper. Note that, this problem is a generic approach as it does not depend any specific domain. On the other side, it has multiple overlapping solutions with other graph algorithms that is solving a graph problem is correlated with solving others.

A crucial point to notice is CKKS does not provide some basic logic operations. First, comparison can not implemented in CKKS which is commonly used in graph algorithms to branch and terminate. CKKS lacks indexing which is intuitive from safety and privacy point. Graph algorithms also use indexing in order to mark visited nodes, get neighbours of a desired node.

In the scope of this work, above issues are solved by communication with client via secure interface. Simply, server handles a step and client slightly modifies the result and resends it. Doing so the algorithm become less secure but otherwise algorithm suffer from two hardship. One is running N , number of nodes, many iteration in a graph analytic program exploits the required bit, which results in bit modulus error. The second is since there is no comparison, server can not early terminate the computation. Moreover, I provide two analysis as execution times of one run of N runs and execution times of cummulative runs.

Miscrosoft Seal [4] is hired for HE library and Microsoft EVA [5] compiler for HE which provides a Python wrapper is hired for simulating the algorithm and encryptions.

Main points are:

- Algorithms for a graph problem, namely number of non-connected nodes, with HE
- Trade offs for two approaches
- Communication interface and analysis with it

2 Background and Related Work

2.1 Background

This section provides the background information.

2.2 Related Work

Homomorphic Encryption has gained more emphasis. Munjal et al. published a review on HE please see for detailed explanation of HE and state of the art methods [6]. Louis et al. conducted a review for machine learning methods [7]. There are considerably high works on medicine since the data privacy is crucial. Wood et al. provided a newer review of HE for medicine and machine learning [8]. In [9], Froelicher et al. proposed novel federated analysis of medical data and they discussed how separate parts can contribute to computation.

Counting the number of non-connected nodes of a node has multiple solutions. It can be solved with depth first search, breath first search and finding paths with length 1 to N, where N is the number of nodes in the graph.

To the best of my knowledge, there is no work on homomorphical graph search algorithm which handles all search operations in server. Nevertheless, matrix multiplication algorithms have attracked the attention from machine learning community[10].

3 Main Contributions

3.1 Search Based Approach

```
Implements: Search One Step
Uses: encryptedMsg Instance: graph
Input: graph, N
Output: reval
Events: GetMask, FilteredNeighbours

GetMask: ($row$) do                          ▷ Matrix of NxN, (i,j) = 1 if graph(0, i) = 1
 1: $mask = [0.0] \cdot vecLength$                   ▷ vecLength is preset $4096 * 4$
 2: $mask[row] = 1$
 3: $mask = mask \cdot graph$                        ▷ mask(i,j)=1, if graph(0,i) = 1
 4: $mask = mask + (mask >> i)$ for i in range(0, $N/2$)

FilteredNeighbours: (graph) do
 5: $allMasks \leftarrow \emptyset$
 6: $allMasks = allMasks + GetMask(row)$ for row in range(0, N)
 7: $filteredNeighbour \leftarrow allMasks \cdot graph$

BFSHeadOneStep: (graph) do       ▷ Compute one step further neighbours for the head
    node.
 8: $reval \leftarrow \emptyset$
 9: $fNeighbours \leftarrow FilteredNeighbours(graoh)$
10: $reval = reval + (fNeighbours << (N \cdot k))$ for k in range (1, N)
```

Algorithm 1: Search One Step : one run of a BFS computation for the head node.

3.2 Matrix Multiplication Based Approach

In this section, I present client-side operations of matrix multiplication based approach which is highly similar to client-side of the previous search based approach. Matrix multiplication psuedocode is not given here but implemented in the code. For details of homomorphic efficient matrix multiplication, please see [10].

Implements: Clientside
Uses: Graph **Instance**: graph
Input: graphAsAdj, N
Output: number of non-connected nodes
Events: Serialize, Encrypt, CheckChange, OneStep, CountNonConnected, Stablize

```
Stabilize: (graph) do
 1: If $graph(i) > 0.7$:
 2:     $graph(i) = 1$
 3: Else:
 4:     $graph(i) = 0$
 5: $graph \leftarrow Serialize(graphAsAdj)$
 6: $oldHead \leftarrow \emptyset$

One Pass Loop: (graph, oldHead) do
 7: $graphEnc \leftarrow Encrypt(graph)$
 8: $retval \leftarrow OneStep(graphEnc)$
 9: If $OneStep()$ is $BFSHeadOneStep()$:
10:     $retval \leftarrow Stabilize(retval)$
11: If not $CheckChange(retval, oldHead)$ :
12:     Return $CountNonConnected(retval)$
13: $oldHead \leftarrow retval[: N]$          ▷ Get only the head adjacency list, other rows are thrash
```

Algorithm 2: Clientside : operations that are performed to early stop and prevent bit explosion of CKKS program

4 Results and Discussion

4.1 Methodology

Number of non-connected nodes problem has multiple solution as mentioned earlier. The following sections explain details of approaches. Number of non-connected nodes will be denoted as NoNCN.

To define the problem I state given a graph $\mathcal{G}$ with non-directed edges between $\mathcal{N}$ nodes, adjacency matrix is $\mathcal{A} \in \mathcal{R}^{\mathcal{N}\times\mathcal{N}}$, where $\mathcal{A}_{i,j}$ denotes the connection between $\mathcal{N}_i$ and $\mathcal{N}_j$.

4.1.1 Search Based Homomorphic NoNCN

The most general and the simplest approach is traversal approaches starting from the head node. During traversal mark the nodes encountered in the visited list which initialized all False's. Consequently return the number of False's in the visited list. Apart from classical solution, homorphic twin of this solution will be presented. To see overall approach please see 1. Note that HE keeps the polynomial form of the graph and there can not be separate visited list. In order to solve that, adjacency list itself modified such that row of the head node (first node) is the visited list.

The method starts with creating masks $\mathcal{M}_i$ for each node i. Since there is no comparison, $\mathcal{A}_{0,i}$ is detected by indicator vector *indic*. Indicator is $[0..0] \in \mathcal{R}^{16384}$, where 16384 is the vector size of encrypted message, vector with only one element $indic_i = 1$, as I create i^{th} node's mask. Multiplication of *indic* and *graph* gives the vector of *indic* back or all zeros with respect to $\mathcal{A}_{0,i}$. Consequently, I get a vector with one element is 1 or not. In order to create the mask from that, I used shifted sums. In case $\mathcal{A}_{0,i} = 0$ since $indic = [0..0]$, shifted sum is also 0 vector. In case $\mathcal{A}_{0,i} = 1$, $\mathcal{M}_i = 1$ and it is shifted $\frac{\mathcal{N}}{2}$ times while at each time it is summed with it's previous version. An simplified example 4.1.1, i.e. without padding 0s up to 16384:

$$\begin{aligned}
\mathcal{M}_i^0 &= [1,0,0,0] \\
\mathcal{M}_i^1 &= [1,0,0,0] + [0,1,0,0] = [1,1,0,0] \\
\mathcal{M}_i^2 &= [1,1,0,0] + [0,0,1,1] = [1,1,1,1]
\end{aligned}$$

$\mathcal{M}^k$ defines the k^{th} time of shifting and vector is shifted to left 2^k. However if $\mathcal{N} \neq 0 \ mod(2)$, mask can not be computed as 1s with shifted sums. The restriction comes from a step ahead. At the next step of algorithm, all generated masks are summed, thus, each mask should only affect its row in $\mathcal{M} \in \mathcal{R}^{\mathcal{N}\times\mathcal{N}}$. Again, an simplified example 4.1.1 is given where $\mathcal{N} = 5$:

$$\begin{aligned}
\mathcal{M}_i^0 &= [1,0,0,0,0] \\
\mathcal{M}_i^1 &= [1,0,0,0,0] + [0,1,0,0,0] = [1,1,0,0,0] \\
\mathcal{M}_i^2 &= [1,1,0,0,0] + [0,0,1,1,0] = [1,1,1,1,0] \\
\mathcal{M}_i^3 &= [1,1,1,1,0] + [0,1,1,1,1] = [1,2,2,2,1]
\end{aligned}$$

At the third step $\mathcal{M}_i^3$, in order not to affect other domains, instead of shifting 2^3, it is shifted to left as $\mathcal{M}_i^2 >> 1$.

All masks are computed separately. Note that each mask starts from i^{th} location and after mask computation it occupies a range from i to $i + \mathcal{N}$. Before summation, each mask is shifted to the appropriate position in $\mathcal{M}$. That is, i^{th} mask is shifted from $(i, \mathcal{N} + i)$ to $(i \cdot \mathcal{N}, i \cdot \mathcal{N} + \mathcal{N})$.

At this point, notice that this method is close to matrix multiplication. This paragraph shortly explains that. Since the graph is non-directed, its adjacency matrix is a symmetric matrix. Thus, $\mathcal{A} = \mathcal{A}^T$. Masks $\mathcal{M}$ corresponds to multiplying each row with the corresponding column. As a result, it seems like row by row element multiplication, but, it is identical to row by column multiplication.

After multiplication, filtered neighbours $\mathcal{F} = \mathcal{M} \cdot \mathcal{A}$ is computed with one missing operation. Essentially, it is computing the head's row of matrix multiplication. All rows of multiplication is shifted left so that they overlap with the first row and summed. Overall example is given below.

$$\mathcal{A}^0 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \mathcal{M} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathcal{A} \cdot \mathcal{M} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$(\mathcal{A} \cdot \mathcal{M})_{shifted} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathcal{A}^1 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$\mathcal{A}^1$ is the one step of homomorphic algorithm. Client communication is discussed under 4.1.3.

4.1.2 Matrix Multiplication Based Homomorphic NoNCN

NoNCN problem can be seen as finding paths whose length are from 1 to $\mathcal{N}$. Finding such paths, neigbhour of the head node is all reachable nodes from the head. Section starts with the classical formulation of the paths of length 1 to i. Then, homomorphic approach will be explained. Same notation with 4.1.1 will be used.

Power i of an adjacency matrix computes paths of length i.

$$\mathcal{A}^i = \mathcal{A}^{i-1} \times \mathcal{A}^0$$

However, this form computes the paths of length exactly i. In order to compute paths of length from 1 to i, adding self connections at each step to the $\mathcal{A}$ is enough. Doing so, $\mathcal{A}^0$ connections also preserverd in the $\mathcal{A}^i$. Consequently, the overall idea has such a simple structure.

$$\mathcal{A}^i = (\mathcal{A}^{i-1} + \mathcal{I}) \times \mathcal{A}^0$$

Homomorphic matrix multiplication is proposed at [10]. It is not duplicated here and treated as a close method since algorithm itself requires long explanation and formulation. At each step, identity is added to the given graph and multiplied with itself. This result is returned to the client. As can be understood, classical version and the homomorphic version has tight relation. However, one more implementation detail is inserted to the matrix multiplication. In the cited work, vectors of the homomorphic program is given as $\mathcal{A} \in \mathcal{R}^{\mathcal{N}^2}$, i.e. in the size of adjacency list in this paper's scope. Unlike proposed method, $16384 = 4096 \times 4$ is picked for this project as safe margin. The issue reveals from the rotation operations of the proposed matix multiplication. Adjacency matrix is duplicated and shifted $\mathcal{N}^2$ to right, then added to the original graph vector in order to handle rotations. Note that rotations for the matrix multiplication are saturated, nevertheless, there are trash values at the end of graph vector. These values are affordable since all operations are left rotation or can be transformed to left rotation in the matrix multiplication. On the other pass, when client resends $\mathcal{A}^i$, garbage values are cleared on the server side.

4.1.3 Client-Server Interface

Two approaches explained above both having their own issues. In this section, interface for matrix multiplication based will be mentioned earlier and followed by search based method issues.

Matrix multiplication for one pass includes multiplications in itself that increases programs *bit modulus*. *Bit modulus* is the required bits to find a prime number $p \ mod \ 2N = 1$. Multiplication which increases depth of the program increases the required bit. A few more discussion is included at 4.1.4. This issue is common for both approaches since both depends N many times looping. As a solution, one step computed graph is returned at the end of each step. Client checks whether the head node $\mathcal{A}_0$ is changed or not. If client does not decide to stop, it resend the graph to the server to compute the a step further paths.

Search based approach has another issue, namely numerical explosion. As can be seen at 4.1.1, masks may have greater numbers than 1 and there are $\mathcal{N}$ many multiplication. At the end, all rows are summed. Thus, small numerical deviations gets bigger. Also, CKKS can not maintain such deviations under control and 0 values diverges to the higher values. Therefore, client maps each values to 1 and 0 at the end of each pass. That is the only difference from the matrix based interface.

4.1.4 Implementation Details

In this section implementation details namely scaling settings and multi core implementation will be investigated.

Apart from all the discussions above input and output scales takes significant role. Input scales are increased to take bit modulus under control. Notice that larger input scales implies a larger noise budget, hence more encrypted computation capabilities which is commented in the code of SEAL [4].

Another point on the scaling is 'lazy_relinearize' config in the porvided code template. Dividing code in to the functions may force lazy relinearization to relinearize the outputs of the functions. Thus, methods are divided into as many meaningful functions as possible. That solved bit modulus errors.

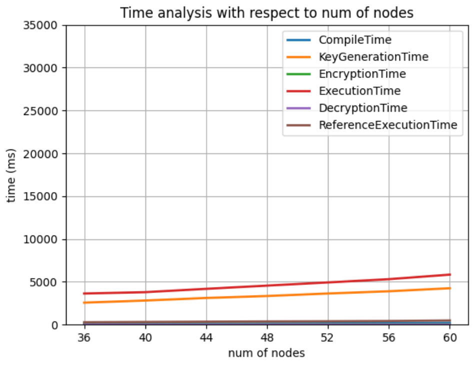
4.1.5 Discussion on Approaches

Search based approach runs faster since it does not consider remaining rows other than the head node. It fulfills the remaining adjacency graph $\mathcal{A}^k_{i\neq 0}$ with trash values as it uses $\mathcal{A}^0$ without caring other than head. Thus, it seems more secure.

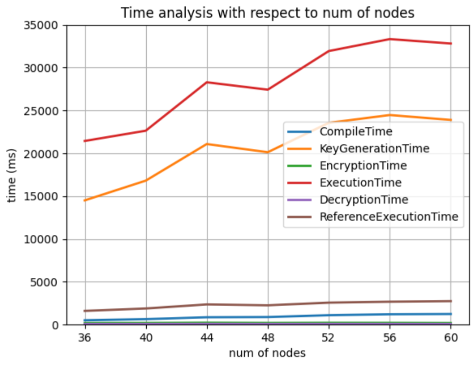
On the other side, if one needs a number of non-connected graphs for all nodes matrix multiplication based approach seems valuable. It is slower than former approach and returns paths for all nodes. That makes it less secure, however, desirable choice for some tasks.

4.2 Results

Experiments are run on graphs $\mathcal{G}$ with $\mathcal{N} \in [36, 60]$. For each node count, 100 experiments are conducted. Measured variables are CompileTime, KeyGenerationTime, EncryptionTime,



(a) Time analysis of metrices with Search Based.

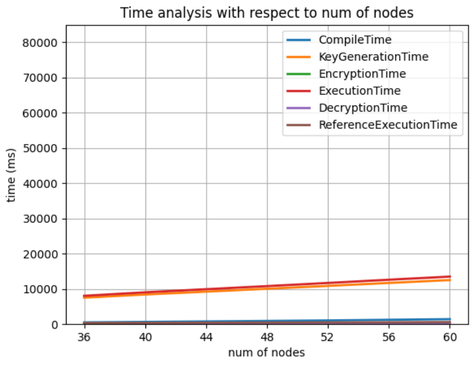


(b) Total time analysis of metrices with Search Based.

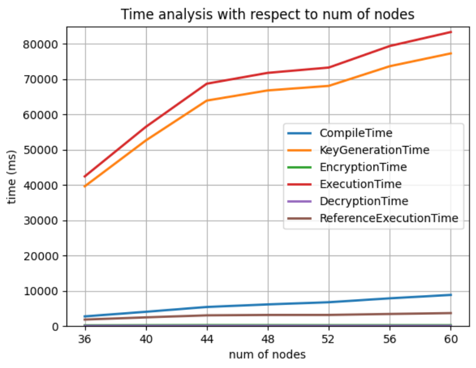
Figure 1 . Time analysis of one pass and all passes put in the same scale. Search Based approach is investigated. $\mathcal{N}$ ranges from 36 to 60, nonetheless, times scale up to nearly 5 times.

ExecutionTime, DecryptionTime, ReferenceExecutionTime and Mse. Most of the variables does not give clear idea since they take short time. However, ExecutionTime gives valuable sights. On the other hand, Mse does not variates since client stabilizes results before resending.

At the figure 1 , the scaling between total (b) time and one pass (a) time is relatively small. This is the benefit of early stopping. Otherwise, time scales with the $\mathcal{N}$. This case holds for the matrix multiplication at 3 .



(a) Time analysis of metrices with Matrix Mult Based.

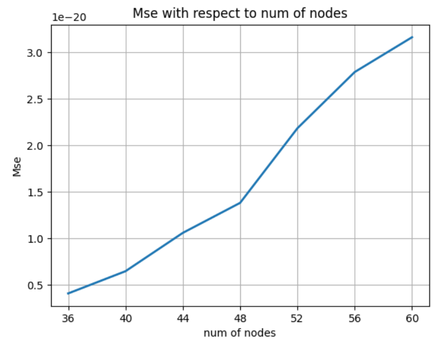


(b) Total time analysis of metrices with Matrix Mult Based.

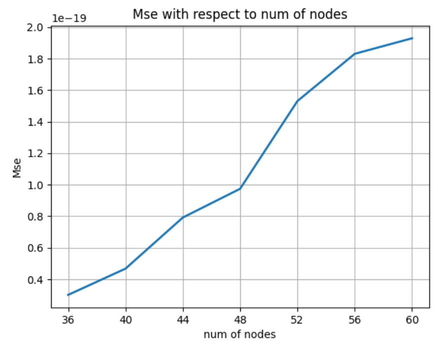
Figure 2 . Time analysis of one pass and all passes put in the same scale. Matrix Multiplication Based approach is investigated. $\mathcal{N}$ ranges from 36 to 60, nonetheless, times scale up to nearly 5 times.

At 1 and 3 , it is seen that ExecutionTime and KeyGenerationTime are steady (even decreases in some cases). This is an interesting result. That may be resulted from probability of having edge ($p = 0.05$) while generation of graph is kept constant during tests. Therefore, having more nodes means the probability of reaching maximum connection is faster since there are more nodes with the same probability of having edge. As a result, lines gets steadier as the $\mathcal{N}$ increases. The varianced version of the plots are also investigated, yet, they are not presented since plots become unreadable. This is a drawback of early stopping. In case the graph $\mathcal{G}$ generated more linear, required pass increases with the linearity of the graph. Consequently, variance rises when this kind of graphs are created.

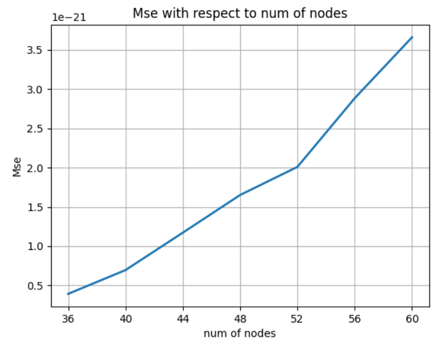
The MSE plots are given here but does not implies a lot. As expected the same attitude with other variables is hold in MSE plots. Linearity can be seen from the figures. Opposite to the early stopped times, MSE slope rises with the increasing number of nodes. This is also expected since multiplications and additions focus larger areas and the number of 0s in the serialized vector given to the graph analitic program decreases.



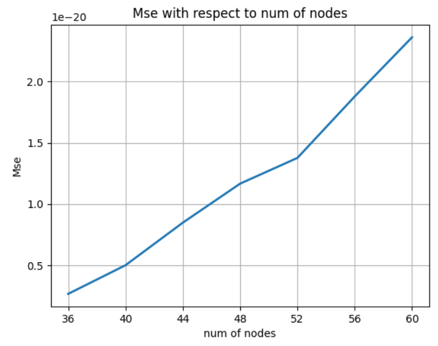
(a) MSE of one pass Search Based. Note that scale is 10 times smaller than total MSE.



(b) MSE of total passes Search Based.



(c) MSE of one pass Matrix Mult Based. Note that scale is 10 times smaller than total MSE.



(d) MSE of total passes Matrix Mult Based.

Figure 3 . MSE analysis of Search Based (top) and Matrix Mult Based (bottom). Each approachede is analysed in terms of one pass (left) and total passes (right).

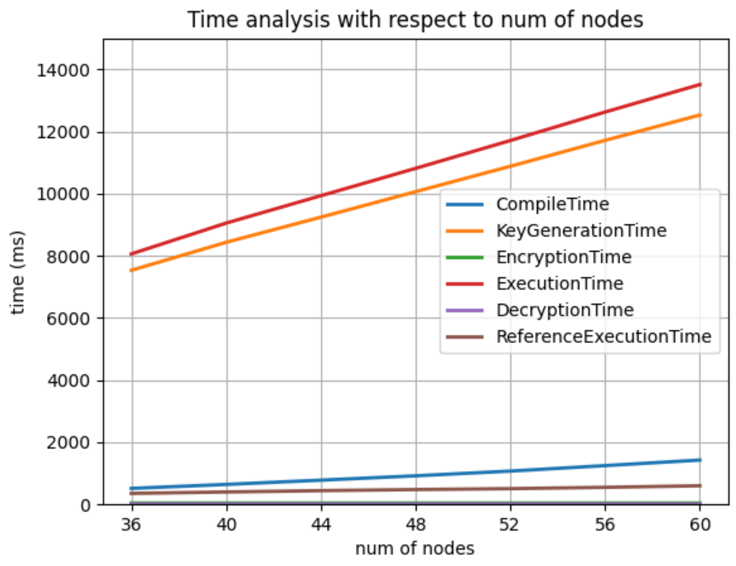


Figure 4 . Time analysis of One Pass in big scale.

At the 4 , which is a closer look to 3 , linearity is more obvious. Indeed, the increase is consistent, unlike total cases. As a result, the former guess on probability of having edges at 4.2 looks right. Lastly, Matrix Multiplication and Seach Based methods compared at 5 . Clearly, Matrix Multiplication Based approach starts with more time consumption and the ratio of that consumption swells with the increase of nodes.

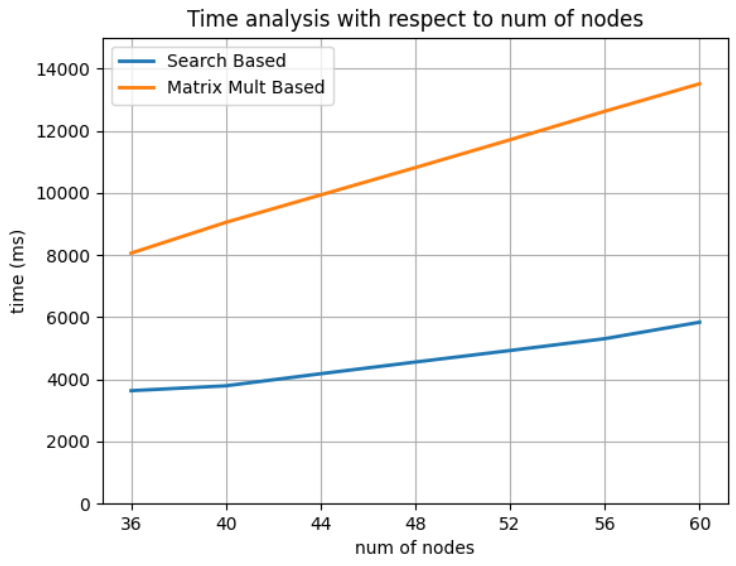


Figure 5 . ExecutionTime comparison of Search and Matrix Mult Based approaches.

4.3 Discussion

The main discussion is clearified at 4.1.5. Client side operations make approaches weaker in the security view. Stabilization of the numbers for Search Based approach may be done at the server. Rescaling methods may be revisited to relinearize the one pass in the server so that all loops can be done at server. More experiments can be conducted with different having edge probability p and comparison between approaches can be investigated further.

5 Conclusion

In this report, a graph algorithm, namely number of non-connected nodes, is investigated. The homomorphic methods are proposed and discussed in terms of weaknesses and performance. Further directions are brought on the table.

References

[1] L. J. M. Aslett, P. M. Esperança, and C. C. Holmes, "A review of homomorphic encryption and software tools for encrypted statistical machine learning," 2015. [Online]. Available: https://arxiv.org/abs/1508.06574

[2] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, p. 409–437.

[3] O. Corporation. (2021) 17 use cases for graph databases and graph analytics. [Online]. Available: https://www.oracle.com/a/ocom/docs/graph-database-use-cases-ebook.pdf

[4] "Microsoft SEAL (release 4.0)," https://github.com/Microsoft/SEAL, Mar. 2022, microsoft Research, Redmond, WA.

[5] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, "EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation.* ACM, jun 2020. [Online]. Available: https://doi.org/10.1145%2F3385412.3386023

[6] K. Munjal and R. Bhatia, "A systematic review of homomorphic encryption and its contributions in healthcare industry," *Complex Intelligent Systems*, pp. 1--28, 05 2022.

[7] L. J. M. Aslett, P. M. Esperança, and C. C. Holmes, "A review of homomorphic encryption and software tools for encrypted statistical machine learning," 2015. [Online]. Available: https://arxiv.org/abs/1508.06574

[8] A. Wood, K. Najarian, and D. Kahrobaei, "Homomorphic encryption for machine learning in medicine and bioinformatics," *ACM Comput. Surv.*, vol. 53, no. 4, aug 2020. [Online]. Available: https://doi.org/10.1145/3394658

[9] D. Froelicher, J. R. Troncoso-Pastoriza, J. L. Raisaro, M. A. Cuendet, J. S. Sousa, J. Fellay, and J.-P. Hubaux, "Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption," *bioRxiv*, 2021. [Online]. Available: https://www.biorxiv.org/content/early/2021/02/25/2021.02.24.432489

[10] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1209–1222. [Online]. Available: https://doi.org/10.1145/3243734.3243837