PRESERVING PRIVACY WITH HOMOMORPHIC ENCRYPTION IN 5G NETWORK DATA ANALYTICS FUNCTION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BERKER ACIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2023

Approval of the thesis:

PRESERVING PRIVACY WITH HOMOMORPHIC ENCRYPTION IN 5G NETWORK DATA ANALYTICS FUNCTION

submitted by **BERKER ACIR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Ertan Onur
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Prof. Dr. Ertan Onur
Computer Engineering, METU _____

Prof. Dr. İbrahim Körpeoğlu
Computer Engineering, Bilkent University _____

Date: 08.09.2023

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Berker Acır

Signature :

ABSTRACT

PRESERVING PRIVACY WITH HOMOMORPHIC ENCRYPTION IN 5G NETWORK DATA ANALYTICS FUNCTION

Acır, Berker
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Ertan Onur

September 2023, 81 pages

Recently, cloud computing services have found widespread use. Offloading computations to public clouds has many benefits albeit harming the privacy of users and data. Homomorphic encryption (HE) facilitates cloud computing services that can do computations over encrypted data in the cloud without requiring decryption and this enables privacy-preserving applications. In this thesis, we propose to offload delay-tolerant tasks of the Network Data Analytics Function (NWDAF) which plays a key role in autonomous network management and security in 5G networks using heuristic and machine learning (ML) techniques. Not all methods such as deep neural networks (DNN), and complex analysis algorithms can be deployed in the core network due to their high computational requirements. Besides, telecommunication companies may opt to outsource the analytics to experienced third-party providers rather than implementing them by themselves. By offloading Network Data Analytics Function (NWDAF) tasks to the cloud, telecommunication companies can concentrate on expanding their core capabilities, and reduce their capital and operational expenditures.

To support the proposed idea, two of the Network Data Analytics Function (NWDAF) use cases are selected and it is shown that homomorphic encryption can be employed

for preserving data privacy in both use cases. Selected use cases are user equipment (UE) abnormal behavior or anomaly detection, and user equipment (UE) mobility analytics and expected behavior prediction. For the former use case, various DL models are trained with plaintext data and it is demonstrated that inferencing on encrypted data is as accurate and precise as plaintext inferencing. For the latter use case, an approach for confidentially finding islands (connected components) in a graph is proposed and shown that how this approach can be used for mobility analytics and expected behavior prediction. Various performance evaluation results are quantified and it shows that privacy-preservation can be achieved with a cost of computation overhead.

Keywords: 5G, Deep Learning, Homomorphic Encryption, Network Data Analytics Function, Privacy Preserving

ÖZ

5G ŞEBEKE VERİ ANALİTİĞİ FONKSİYONU'NDA HOMOMORFİK ŞİFRELEME İLE MAHREMİYETİN KORUNMASI

Acır, Berker
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Ertan Onur

Eylül 2023 , 81 sayfa

Son dönemlerde bulut bilişim hizmetleri yaygın olarak kullanılmaktadır. Hesaplamaların herkese açık bulutlarda yapılması, birçok fayda sağlamasına rağmen, kullanıcıların ve verilerinin mahremiyetini riske atmaktadır. Homomorfik şifreleme (HE) yöntemi, bulut bilişim hizmetlerinin şifrelenmiş veriler üzerinde doğrudan hesaplama yapılabilmesini mümkün kılar ve bu da veri mahremiyetinin korunmasını sağlar. Şebeke Veri Analitiği Fonksiyonu (NWDAF), buluşsal (heuristic) veya makine öğrenimi (ML) tekniklerini kullanarak 5G şebekelerinde otonom ağ yönetimi ve güvenliği konusunda önemli bir rol oynar. Bu çalışmada, Şebeke Veri Analitiği Fonksiyonu'nun (NWDAF) gecikmeye toleranslı görevlerinin, homomorfik şifreleme (HE) kullanılarak kullanıcı ve veri gizliliğini korurken üçüncü taraf bulut sağlayıcılarına yüklenebileceği fikrini öneriyoruz. Fakat, derin sinir ağları (DNN) veya karmaşık analiz algoritmaları gibi yöntemler, yüksek hesaplama gereksinimleri nedeniyle çekirdek ağda konuşlandırılamayabilir. Ayrıca telekomünikasyon şirketleri, analitik methodlarını kendileri uygulamak yerine deneyimli üçüncü taraflara devretmeyi tercih edebilir. Bunun sonucunda da, telekomünikasyon şirketlerinin temel yeteneklerini genişletme-

sine ve sermaye ile operasyonel harcamalarını azaltmaya odaklanmasına olanak sağlayacaktır.

Bu çalışmada önerilen fikri desteklemek adına, iki farklı Şebeke Veri Analitiği Fonksiyonu (NWDAF) kullanım senaryosu seçilmiştir ve homomorfik şifreleme ile veri gizliliğin korunabileceği gösterilmiştir. Kullanıcı ekipmanı (UE) için anormal davranış veya anomali tespiti ve kullanıcı ekipmanı (UE) için hareketlilik analitiği ve beklenen davranış tahmini kullanım senaryoları seçilmiştir. İlk senaryo için, çeşitli derin öğrenme (DL) modellerini düz metin verileriyle eğitiyoruz. Daha sonra, şifrelenmiş veriler üzerinde çıkarımın düz metin çıkarımı kadar doğru ve kesin olduğu gösteriliyor. Diğer senaryo için, şifreli bir çizgede veri gizliliğini bozmayacak bir şekilde adaları (bağlı bileşenleri) bulma yöntemi sunuluyor, ve bu yöntemin hareketlilik analitiği ve beklenen davranış tahmini için nasıl kullanılabileceği gösteriliyor. Sunduğumuz yöntemlerin çeşitli perfomans ölçümleri ile analizi sunuluyor, ve performans düşüşü karşılığında mahremiyetin korunabileceğini gösteriliyor.

Anahtar Kelimeler: 5G, Derin Öğrenme, Homomorfik Şifreleme, Şebeke Veri Analitiği Fonksiyonu, Mahremiyetin Korunması

To my father, whose memory lives on, and to my mother, my constant support.

ACKNOWLEDGMENTS

To begin with, I want to express my deepest gratitude to my supervisor, Prof. Dr. Ertan Onur, for his unwavering support and valuable guidance throughout my Master of Science program.

I would also like to thank my thesis committee members, Prof. Dr. İbrahim Körpeoğlu and Assoc. Prof. Dr. Pelin Angın, for serving on my thesis committee and providing helpful feedback and valuable contributions.

I am grateful to Turkcell Technology for supporting my master's studies within the framework of 5G and Beyond Joint Graduate Support Programme coordinated by Information and Communication Technologies Authority.

I would also like to thank my teammates in METU Falcons, chiefly Göksel Üstündağ and Yiğit Adaş, and my friends, mostly Alpkan Sözer, Berkant Bayraktar, İlker Ayçiçek, Yunus Emre Saçma, and Yusufcan Manav, for their motivation and support through my studies.

I would also like to sincerely thank my family for their unconditional love and never-ending support. I am grateful to my late father, Özer Acır, and my mother, Naime Acır, for making sacrifices and never giving up their best. Also, I want to thank my brother, Dinçer Acır, for helping me and I wish him the best in his journey as a Mechanical Engineering student at METU and afterward.

Most importantly, I am thankful for having my dearest love, İrem Aydın, in my life as I would not have been able to complete this thesis without her wisdom, support, and guidance. She has been there to hold my hand and cheer me up whenever I have been lost and down. Also, s*paw*cial thanks to our cats, Dardanel and Pirinç, for their fluffy emotional support.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiv

LIST OF FIGURES xv

LIST OF ABBREVIATIONS xviii

CHAPTERS

1 INTRODUCTION 1

1.1 Motivation and Problem Definition 1

1.2 Contributions and Novelties 3

1.3 The Outline of the Thesis 5

2 BACKGROUND INFORMATION 7

2.1 Related Work 7

2.2 Preliminaries 10

2.2.1 5G Networks 10

2.2.1.1 5G Service-Based Architecture (SBA) and Network Functions (NF) 11

2.2.1.1.1 Network Data Analytics Function (NWDAF) . . 12

2.2.2 Homomorphic Encryption (HE) 14

2.2.2.1 Types of Homomorphic Encryption 16

2.2.2.2 Tools: Libraries, Compilers, Frameworks 17

2.2.2.2.1 The Simple Encrypted Arithmetic Library (SEAL) 17

2.2.2.2.2 Intel's Graph Compiler for Artificial Neural Networks with HE backend (nGraph-HE) 18

2.2.2.2.3 The Encrypted Vector Arithmetic Language and Compiler (EVA) 19

3 PRIVACY-PRESERVING ANOMALY DETECTION FOR A GROUP OF USER EQUIPMENT (UE) 21

3.1 Privacy Preservation with Deep Learning 22

3.1.1 Dataset 23

3.1.2 Deep Learning Models 26

3.2 Results and Discussion 28

3.2.1 Methodology 29

3.2.2 Impact of Polynomial Modulus Degree (N) 30

3.2.3 Impact of the Security Level 31

3.2.4 Impact of Deep Learning Layers and Parameter Count 31

3.2.5 Impact of Complex Packing 32

3.3 Evaluation of the Privacy Preservation 33

4 PRIVACY-PRESERVING MOBILITY ANALYTICS FOR USER EQUIPMENT (UE) 35

4.1 Finding the Islands in a Graph 36

4.2 HE-Friendly Methodology 37

4.3 Implementation . . . 39
4.3.1 Methods . . . 42
4.3.1.1 Reducing Ones . . . 43
4.3.1.2 Adjacency Matrix Representation . . . 44
4.3.1.3 Distances Matrix Calculation . . . 44
4.3.1.4 Finding Islands . . . 45
4.3.2 Implementation Details . . . 46
4.3.2.1 Vectorized Implementation . . . 48
4.3.2.2 Parallelized Implementation . . . 49
4.3.2.3 Parallelized Implementation without Loops . . . 50
4.3.3 Implementation Comparisons . . . 51
4.4 Results . . . 53
4.4.1 Plots of Reduce Ones EVA Program . . . 54
4.4.2 Plots of Compute Distances EVA Programs . . . 54
4.4.3 Comparison of EVA Programs with Only One Input . . . 59
4.4.4 Amortized Runtime Comparison of EVA Programs . . . 63
4.5 Discussion . . . 64
5 CONCLUSION . . . 69
REFERENCES . . . 71
APPENDICES
A RUNNING TIMES OF ALL CONFIGURATIONS . . . 79

LIST OF TABLES

TABLES

Table 3.1 Averaged and amortized time results of all experiments. 34

Table 3.2 Average accuracy and precision results of all experiments. 34

Table 4.1 Methods and information about whether they are executed in the client or the server. 52

Table 4.2 The number of times that EVA Programs are called in the implementations where n^2 is the size of input matrix/matrices. 52

Table 4.3 Input sizes, vector sizes, and the number of inputs that are used in EVA Programs of the implementations where n^2 is the size of input matrix/matrices and 2^m is the number of different input matrices in the Parallelized Implementations. 53

LIST OF FIGURES

FIGURES

Figure 1.1 Privacy-preserving NWDAF architecture where NWDAF offloads computations to third-party cloud computing services with homomorphic encryption. 4

Figure 2.1 5G service-based architecture. 12

Figure 2.2 Network data analytics function (NWDAF) in 5G networks. . . . 14

Figure 2.3 Data processing in cloud computing services compared by traditional and homomorphic encryption. The figure is redrawn from Microsoft's documentation titled "Homomorphic encryption with SEAL". . 16

Figure 3.1 NWDAF architecture where MTLF either securely deploys trained model to the cloud or initiates secure model training on the cloud, and AnLF makes inferences on the cloud models with encrypted inputs. . . 22

Figure 3.2 One day sample redrawn from the dataset of Sevgican et al. . . . 25

Figure 3.3 An example of raw and pre-processed input and output data. . . . 26

Figure 3.4 An example of a fully connected neural network (FCNN) architecture drawn with software from LeNail et al. 28

Figure 3.5 Running times of a DL model with 5 layers and 452 parameters with 128-bit security where three different polynomial modulus degrees (N) are compared. 31

Figure 3.6 Running times of a DL model with 5 layers and 888 parameters with a polynomial modulus degree of 14 where three different security levels are compared. 32

Figure 3.7 Running times of 4 different DL models with layer counts of 5 and 7, approximately 450 and 880 parameters with polynomial modulus degree of 15 and 192-bits security level where numbers of parameters and layers are compared. 32

Figure 3.8 Running times of a DL model with 7 layers and 1200 parameters with polynomial modulus degree of 15 and 256-bits security level where complex packing feature is compared. 33

Figure 4.1 Example cellular network topologies. 36

Figure 4.2 Example 2D matrices together with their island counts. 37

Figure 4.3 Directed graph with 4 nodes together with its adjacency matrix and partial calculation of its distances matrix for the first row. 38

Figure 4.4 Sample 3×3 input matrix and its distances matrix. 39

Figure 4.5 Vectorized and parallelized approaches with their input and output differences. 41

Figure 4.6 2D matrices and their conversion into reduced matrices. 43

Figure 4.7 2D matrices and their conversions into adjacency matrices. . . . 44

Figure 4.8 Input matrix and its reduced version with distances matrix. Note that 1 value in the distances matrix means that there is at least one path from the cell in the row to the cell in the column. 45

Figure 4.9 Timings of all Reduce Ones EVA program implementations by input matrix and vector sizes. 56

Figure 4.10 MSE scores of all Reduce Ones EVA Programs implementations by input matrix and vector sizes. 57

Figure 4.11 Timings of all Compute Distances EVA Programs implementations by input matrix and vector sizes. 58

Figure 4.12 MSE scores of all Compute Distances EVA Programs implementations by input matrix and vector sizes. 59

Figure 4.13 MSE score comparisons of EVA Programs by their implementations. 60

Figure 4.14 Timings comparisons of all Reduce Ones EVA Program implementations when there is only one input. 61

Figure 4.15 Timings comparisons of all Compute Distances EVA Program implementations when there is only one input. 62

Figure 4.16 Total runtimes of all EVA Program implementations when there is only one input. 63

Figure 4.17 Amortized timing comparisons of all Reduce Ones EVA Program implementations. 65

Figure 4.18 Amortized timings comparisons of all Compute Distances EVA Program implementations. 66

Figure 4.19 Amortized total runtimes of all EVA Programs implementations. 67

Figure A.1 Averaged running times of inferencing on both plaintext and ciphertext data for each configuration. 80

Figure A.2 Averaged amortized running times of inferencing on both plaintext and ciphertext data for each configuration. 81

LIST OF ABBREVIATIONS

2D	Two Dimensional
3GPP	3rd Generation Partnership Project
5GB	5G and Beyond
5GS	5G System
ADRF	Analytics Data Repository Function
AF	Application Functions
AI	Artificial Intelligence
AMF	Access and Mobility Management Function
AnLF	Analytics Logical Function
API	Application Programming Interface
AUSF	Authentication Server Function
CCL	Connected-Component Labeling
CDR	Charging Data Record
CHF	Charging Function
DCCF	Data Collection Coordination Function
DL	Deep Learning
DNN	Deep Neural Network
eMBB	Enhanced Mobile Broadband
EVA	Encrypted Vector Arithmetic
FCNN	Fully Connected Neural Network
FHE	Fully Homomorphic Encryption
FL	Federated Learning
HE	Homomorphic Encryption
IoT	Internet of Things

LR	Logistic Regression
LSTM	Long Short-Term Memory
MIMO	Multiple-Input/Multiple-Output
ML	Machine Learning
MME	Mobility Management Entity
mMTC	Massive Machine-Type Communications
mmWave	Millimeter Wave
MTLF	Model Training Logical Function
NF	Network Function
NFV	Network Function Virtualization
NWDAF	Network Data Analytics Function
OAM	Operation, Administration and Maintenance
PCF	Policy Control Function
PCRF	Policy and Charging Rules Function
PHE	Partially Homomorphic Encryption
QoE	Quality of Experience
QoS	Quality of Service
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RRU	Remote Radio Unit
SBA	Service-Based Architecture
SBI	Service-Based Interface
SDN	Software-Defined Networking
SEAL	Simple Encrypted Arithmetic Library
SGD	Stochastic Gradient Descent
SMF	Session Management Function
SWHE	Somewhat Homomorphic Encryption

TS	Technical Specification
UE	User Equipment
URLLC	Ultra-Reliable Low Latency Communications
XGBoost	eXtreme Gradient Boosting

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Considering the challenges and envisaged network usage, 5G and beyond mobile networks (5GB) are designed to support the three main use case scenarios: enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communications (URLLC), massive Machine Type Communications (mMTC) [1]. eMBB focuses on providing high user data rates with very high traffic capacity for wide area networks. URLLC is for meeting strict network requirements for reliability, latency, and availability. mMTC consists of connecting a vast number of devices with relatively low data rates.

The emergence of new applications with varying requirements and an increasing number of connected devices pose challenges to both core and mobile networks [1]. These challenges pertain to the need for flexibility and scalability, as current mobile network functions are strongly coupled with specialized hardware [2]. To address this issue, 5G mobile networks follow a service-oriented architecture where network elements are decomposed and modularized into network functions that provide their services through restful web interfaces [3–5]. This approach utilizes software-defined networking (SDN) [6] and network function virtualization (NFV) [7] to decouple the network control and data planes and implement network functions in virtual machines or containers running on commodity hardware. By leveraging cloud and edge computing, 5G networks achieve enhanced flexibility, scalability, and deployability com-

pared to traditional networks, allowing them to meet diverse application requirements through network programmability [8].

However, the above-mentioned flexibility and programmability come at the cost of increased management complexity for network operators. To tackle this issue, network automation using data-driven analytics is widely adopted. To this end, the Network Data Analytics Function (NWDAF) has been standardized in Release 15 of 3GPP [9] and continues to be improved in subsequent releases. NWDAF automatically collects network statistics, metrics, and events to produce insights about the network. It takes snapshots of the network dynamically, measures the performance of services and functions, processes and models the data, and shares analytics results with other network functions. Machine learning (ML), data mining, and heuristic techniques are used to create meaningful network and user information from the collected data [10, 11]. This way, NWDAF facilitates autonomous network operation, efficient resource management, enhanced service quality, reliability, and security.

However, in some cases, ML models, especially Deep Learning (DL) models or computationally intensive algorithms may not be deployable in the core network due to their high computational needs. Additionally, network operators may prefer to delegate the implementation of ML models and algorithms to third parties with expertise in these areas rather than implementing them themselves. In such situations, NWDAF could rely on models that are trained or deployed in public clouds or algorithms offered by public cloud services. However, using public cloud computing services can raise significant data privacy concerns since cloud services may be vulnerable to insider attacks [12] and data breaches [13].

While most studies on NWDAF assume that network providers cannot rely on third-party cloud computing for their data analytics services due to data privacy concerns, recent advances in cryptography have made it possible to make calculations over encrypted data without violating data privacy. Offloading NWDAF functionalities, particularly delay-tolerant ones, to third-party cloud providers can enable network operators to be more agile, concentrate on their core capabilities while reducing hardware requirements, and keep up with the ever-growing demands of resources [14], thanks to cloud elasticity and the ability to provision resources on demand. To preserve data

privacy, homomorphic encryption (HE) scheme can be employed to eliminate the risk of exposing sensitive data.

To support the proposal of offloading delay-tolerant tasks of NWDAF to public cloud computing services, two of NWDAF's use cases are selected and solved with HE schemes while preserving data privacy. One of the selected use cases is user equipment (UE) abnormal behavior or anomaly detection while the second one is user equipment (UE) mobility analytics and expected behavior prediction. Both use cases are essential for 5G network security and service quality.

1.2 Contributions and Novelties

By the specifications of NWDAF, data-driven paradigms and approaches such as artificial intelligence (AI) and machine learning (ML) will be the primary sources for providing data analytics for the network's past, present, and future states. In conjunction with this aspect, and considering the plethora of studies in the literature on privacy-preserving DL employing HE schemes, privacy-preserving inferencing with DL will be used in the former use case (UE abnormal behavior or anomaly detection).

A method is devised for confidentially finding connected components in a 2D binary matrix. Then, it is used for the latter use case (UE mobility analytics and expected behavior prediction) since the method can be extended to work on any graphs as long as they are undirected.

In this thesis, the architecture depicted in Figure 1.1 is devised where NWDAF instances make use of third-party cloud computing services while preserving data and user privacy with homomorphic encryption (HE). The reason for that can be telecommunication companies might prefer to outsource analytics implementations to experts in the industry. In addition, telcos may not want to deploy dedicated hardware for AI/ML-related tasks within their network infrastructure, which contradicts the central philosophy of 5G networks: Software-defined networking (SDN) and network function virtualization (NFV). Two of the NWDAF use cases are selected to support

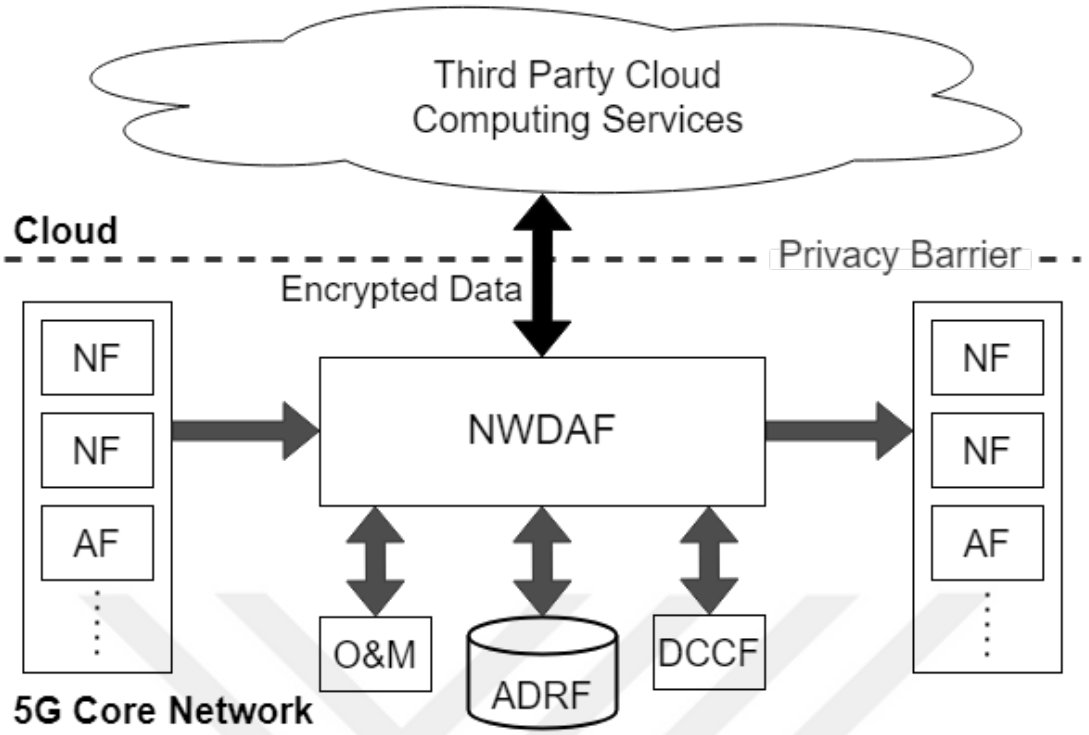


Figure 1.1: Privacy-preserving NWDAF architecture where NWDAF offloads computations to third-party cloud computing services with homomorphic encryption.

the proposal, and their privacy-preserving solutions with HE schemes are shown with implementation details, simulation results, discussions, and future work.

The NWDAF use case of detecting abnormal user behavior is selected to show the proposal's feasibility, and various DL models are trained with plaintext data. Then, inferencing on both plaintext and ciphertext inputs is evaluated in terms of accuracy, precision, and performance together with the limitations of HE. The trade-off between preserving user and data privacy and performance degradation is assessed. This demonstrates that NWDAF can leverage DL models deployed on public cloud servers while maintaining data privacy using HE.

In recent years, there have been significant efforts in developing efficient HE schemes, libraries, compilers, and frameworks, and privacy-preserving studies involving HE are mainly focused on ML and DL. This thesis also aims to solve a graph problem that can be used within cloud computing services while preserving privacy, and the solution can be used in real-world applications such as mobility analytics for mobile

devices in cellular networks and connected-component labeling in computer vision. It can be solved on the cloud using HE, even when dealing with sensitive data.

To the best of our knowledge, this is one of the first works that offers the use of third-party cloud computing services for 5G's NWDAF while preserving privacy with HE schemes and providing a privacy-preserving approach with HE for finding connected components in an undirected graph.

1.3 The Outline of the Thesis

This work is organized as follows. Chapter 2 shows related works to both of the problems studied in this thesis in Section 2.1 and summarizes preliminaries related to the problem such as 5G networks and HE in Section 2.2. Two of the selected NWDAF use cases are presented in Chapter 3 and Chapter 4.

Chapter 3 presents the NWDAF anomaly detection use case for a group of UE. Section 3.1 shows the proposed architecture of privacy-preserving anomaly analytics on NWDAF. Section 3.2 introduces an evaluation of the methodology, describes performed experiments, and discusses obtained results. Section 3.3 evaluates the proposed architecture in light of conducted experiments.

Chapter 4 presents the NWDAF mobility analytics use case and associates the use case with the finding the islands in a graph problem, which is shown in detail in Section 4.1. Section 4.2 introduces the HE-friendly methodology for solving the problem while confirming the HE limitations. Section 4.3 shows introduced methods in detail, their implementation details, and further comparison of the implementations. Section 4.4 presents the simulation results while Section 4.5 discusses the results and mentions possible improvements on the implementations.

Finally, Chapter 5 concludes this work together with the summary of this thesis and points out future research directions and possible improvements of both works.

CHAPTER 2

BACKGROUND INFORMATION

2.1 Related Work

In the last decade, there has been extensive research on introducing AI and ML capabilities to both core and cellular networks for 5G and beyond networks. The works of [15–17, 17] emphasize the need for using AI in cellular networks as increased data rates, reduced latencies, and the ability to support a massive number of connected devices will bring new challenges for network control, management, and orchestration. They highlight revolutionary techniques in the 5G cellular networks and discuss which and how AI/ML techniques (i.e., supervised, semi-supervised, unsupervised, reinforcement, federated learning) can be utilized in the cellular networks, together with their challenges.

Network data analytics services were first introduced in 3GPP Rel-15 in 2017 [9], and it has been improving since then with each release cycle of 3GPP's technical reports (TR) and technical specifications (TS) [10, 11]. In [18], authors present their functional NWDAF implementation that works on open-source 5G network software, and they analyze the data collected by NWDAF and draw insights from NF interactions. Sevgican et al. provide a summary of NWDAF and a novel NWDAF-compatible implementation in their work [19].

Anomaly detection is a widely studied topic, especially in ML and DL areas. In [20],

Yuan et al. provide an extensive literature review of anomaly detection techniques in 5G with or without considering NWDAF. The reviewed anomaly detection techniques include supervised, semi-supervised, and unsupervised ML techniques and heuristic approaches. Each has its advantages and disadvantages. Supervised learning methods such as DL can be effective in achieving more accurate detection but require lots of labeled data. Semi-supervised and unsupervised learning methods are particularly useful when there is insufficient or no labeled data. However, their performances can be insufficient when the data have high intra-class differences.

Since the latest specifications allow multiple instances of NWDAF in the network, FL seems one of the promising methods for network analytics. The central NWDAF could be overwhelmed when a large number of UEs or The Internet of Things (IoT) devices request data analytics at the same time. To overcome this problem, Distributed Learning methods [21], especially Federated Learning (FL) [22], are suggested to be employed in the NWDAF architecture [23]. Also, Yu et al. [24] propose a framework architecture that combines both NWDAF and FL while touching upon HE, which can be used for higher security. In [25], authors propose introducing partially homomorphic encryption (PHE) into NWDAF working with FL. By that, the privacy of both local data and shared model weights between NWDAF instances are preserved. Even though using FL can balance the load on the NWDAF instances while achieving high accuracy and preserving privacy, it will bring extra communication costs to the networks with limited bandwidth capabilities due to frequent parameter exchanges between instances.

Although ML and DL heavily depend on data, the literature still lacks comprehensive datasets for NWDAF use cases, even though network data analytics services were introduced in late 2017. In [19], Sevgican et al. address this issue and provide a publicly available synthetic dataset[1] which can be used for two of the standardized NWDAF use cases: network load level prediction and network load level anomaly detection. Details of the dataset are discussed in Section 3.1.1 and how it is utilized in this thesis. The authors also provide supervised ML solutions for network load prediction and network anomaly classification. Linear regression (LR), long short-term memory (LSTM), and recursive neural network (RNN) are used for prediction,

[1] https://github.com/sevgicansalih/nwdaf_data

whereas logistic regression and eXtreme Gradient Boosting (XGBoost) are used for anomaly detection [19].

The concept of fully homomorphic encryption (FHE) was first introduced by Rivest et al [26] in the 1970s. The concept remained just an idea as it was considered impossible or impractical for more than three decades until Craig Gentry's breakthrough with the first fully homomorphic encryption (FHE) scheme in 2009 [27] and his following work [28]. Subsequently, there has been a surge in interest regarding this subject, which has led to the emergence of novel research and advancements on FHE such as Brakerski/Fan-Vercauteren (BFV) [29], Brakerski-Gentry-Vaikuntanathan (BGV) [30], a fast fully homomorphic encryption scheme over the torus (TFHE) [31] and Cheon, Kim, Kim, and Song (CKKS) [32]. Improvements in both theory and implementations together with general hardware advances have paved the way for the use of HE in practice with performance improvements of up to five orders of magnitude [33]. For example, the multiplication time of ciphertexts has been decreased from 30 minutes to less than 20 milliseconds.

Authors of [34] and [35] provide extensive information on HE schemes in their surveys. In a similar manner, the survey [36] has comprehensive information on FHE schemes and mentions preliminaries and related techniques used in FHE. Advances in FHE have brought numerous real-world applications in various domains, and they have also paved the way for the development of the state-of-the-art libraries, compilers, and frameworks that enable those without expertise in cryptography to develop their own secure and privacy-preserving applications. Both surveys of [36] and [33] present state-of-the-art HE tools such as libraries, compilers, frameworks, and accelerators in detail while providing details of applications. Both of the works also address the escalating synergy between HE and ML. In their assessments, the authors of [33] elaborate on these libraries, compilers, and frameworks and how they are utilized in their evaluations [2].

Solutions to finding connected components in a graph [37] and CCL problems [38] date back to the 1970s. The problem can be solved by applying any of the graph traversal algorithms on each island. In each traversal, an island is explored, and then,

[2] https://github.com/MarbleHE/SoK

the traversal algorithm is called again on the next unexplored land piece until there are no unexplored land pieces left. The number of calls gives the number of islands in this problem. Also, the CCL algorithms can be used for solving the problem [39]. However, neither of those algorithms can be directly implemented on encrypted matrices or graphs while employing HE schemes.

2.2 Preliminaries

This section provides the necessary context and information to comprehend the scope of this thesis with brief background information on 5G networks and homomorphic encryption.

2.2.1 5G Networks

The fifth-generation networks have extended capabilities compared to their predecessors, 4G LTE networks. They are designed to have higher data rates up to several gigabits per second, and lower latencies while providing reliable services to an increased number of connected devices simultaneously. These are available with the use of different frequency bands such as millimeter waves (mmWave) and lower-frequency bands, advanced technologies such as massive Multiple-Input Multiple-Output (MIMO), beamforming, edge computing, and SDN/NFV.

Telecommunication operators can divide the network into virtual networks depending on their purposes, thanks to SDN and NFV. This innovative feature is called network slicing. SDN separates the control and data planes from each other, whereas they are tightly coupled with each other in traditional networks with routers and switches. With SDN, a central controller orchestrates the network by making decisions about how data traffic should flow while communicating with switches and routers to configure them based on the current network requirements. Consequently, it allows network management to be more agile and efficient and makes it easier to adapt to evolv-

ing traffic patterns, implement policies, and optimize network resources. On the other hand, NFV abstracts NFs that traditionally run on dedicated hardware away into software runs on standard servers or even the cloud. Virtualization allows NFs to be deployed and managed more dynamically and efficiently. It enhances scalability while reducing the cost by eliminating the need for specialized hardware. Furthermore, it enables the quick deployment of new network services without requiring physical changes to the infrastructure.

2.2.1.1 5G Service-Based Architecture (SBA) and Network Functions (NF)

The 5G networks have a service-based architectural (SBA) framework, which defines how network services or network functions (NFs) are organized and communicate with each other within the 5G network. SBA is a departure from the monolithic architectural styles of previous generations, and it provides flexibility, modularity, loose coupling, scalability, and agility in deploying and managing services within a 5G ecosystem. Figure 2.1 illustrates 5G SBA where each NF exposes its own services through a service-based interface (SBI) by employing a RESTful interface. Each NF may also make use of other NFs' APIs in order to function properly.

Network functions (NFs) used in previous generations are either replaced with new NFs, split into separate NFs, or combined into new NFs in 5G SBA in order to clarify the delineation between control and user planes. For example, Policy and Charging Rules Function (PCRF) and Charging Data Record (CDR) are replaced with Policy Control Function (PCF) and Charging Function (CHF), respectively. On the other hand, Mobility Management Entity (MME) is split into Access and Mobility Management Function (AMF), Authentication Server Function (AUSF), and Session Management Function (SMF). Besides introducing NFs for functionalities of previous generations, there are entirely new NFs such as NWDAF for extending the capabilities of the 5G core network further. NFs play a vital role in ensuring the network to provide high-speed data transfer, low latency, reliable connectivity, and support for a wide range of applications and services that define the capabilities of the 5G networks.

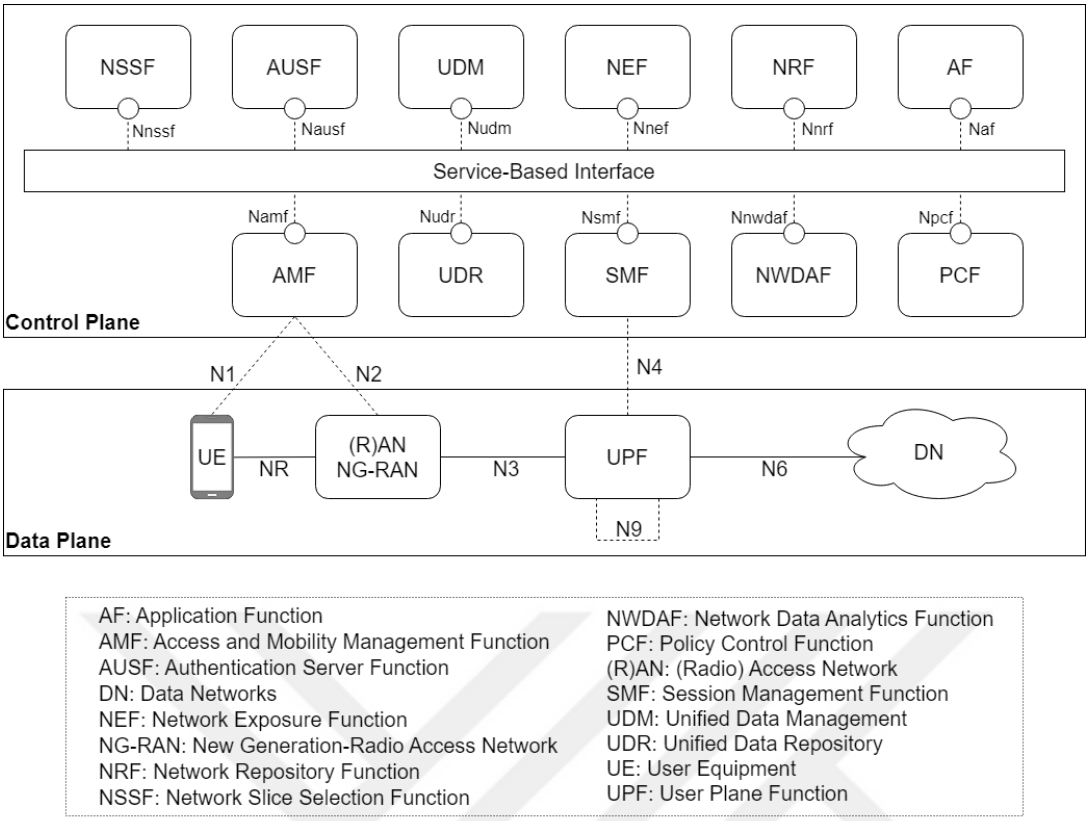


Figure 2.1: 5G service-based architecture [40].

2.2.1.1.1 Network Data Analytics Function (NWDAF)

NWDAF was first defined as a standard for using network data analytics in 5G networks with Release 15 [9]. As of Release 17 [11], it consists of the following logical functions for leveraging AI/ML: Analytics Logical Function (AnLF) and Model Training Logical Function (MTLF). AnLF is responsible for performing inferences deriving and exposing analytics. AnLF requires the models from MTLF whose responsibility is to train and deploy ML models and expose training services. Due to the dynamic nature of the network and data, MTLF self-monitors provided models' performances and initiates re-training or re-designing processes when required. In addition, Release 17 states that there can be more than one NWDAF instance in the network, and an instance has either AnLF, MTLF, or both functionalities combined. An NWDAF instance can be the consumer of another instance. For example,

NWDAF instances with only AnLF can make use of provided models from other instances with MTLF.

NWDAF collects, processes, and analyzes vast amounts of data generated by various NFs and UEs in order to enhance network security and performance and improve quality of service (QoS) and quality of experience (QoE) by applying advanced data analytics and AI/ML techniques to provide insights, intelligence, or predictions in real-time. With 3GPP Release 16 [10] in 2020, the following use cases are officially described in the technical specification:

- Network Conditions:
  - Current congestion information and prediction for a specific location, device, or group of devices
  - Load analytics information and prediction for specific NF instances or types
  - Network performance computation and prediction
  - Network slice instance load level computation and prediction
  - QoS sustainability—reporting and predicting QoS change
- Device Behavior:
  - UE abnormal behavior or anomaly detection
  - UE communication analytics and pattern prediction
  - UE mobility analytics and expected behavior prediction
- Service Experience:
  - Application service experience computation and prediction
  - Service experience computation and prediction for an application or network slice

3GPP's technical specifications (TS) define in detail how NFs expose their services to other NFs and how NFs interact with the services of other NFs. As illustrated in Figure 2.2, NWDAF collects data or analytics information from various sources such

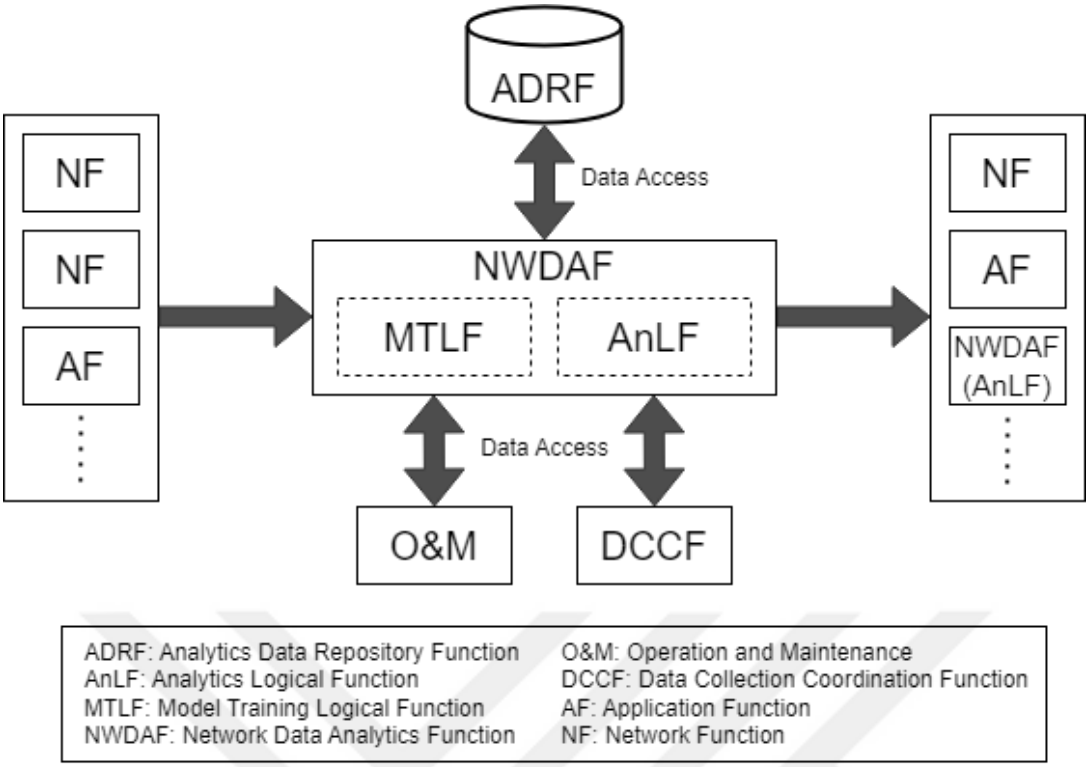


Figure 2.2: Network data analytics function (NWDAF) in 5G networks [41].

as Application Functions (AF), Analytics Data Repository Function (ADRF), Data Collection Coordination Function (DCCF), Operation, Administration and Maintenance (OAM), and then infer insights, or predictions from collected data. After that, NWDAF publishes its own analytics information to related subscribers which can be any NF, AF, or even AnLF of another NWDAF.

2.2.2 Homomorphic Encryption (HE)

Most of the encryption schemes include the following three functionalities: key generation, encryption, and decryption. Symmetric-key encryption schemes use the same secret key for both encryption and decryption, whereas public-key encryption schemes use two keys separately: a public key for encryption and a secret key for decryption. Therefore, public-key encryption schemes allow anyone who knows the public key to encrypt data, but only those who know the secret key can decrypt and

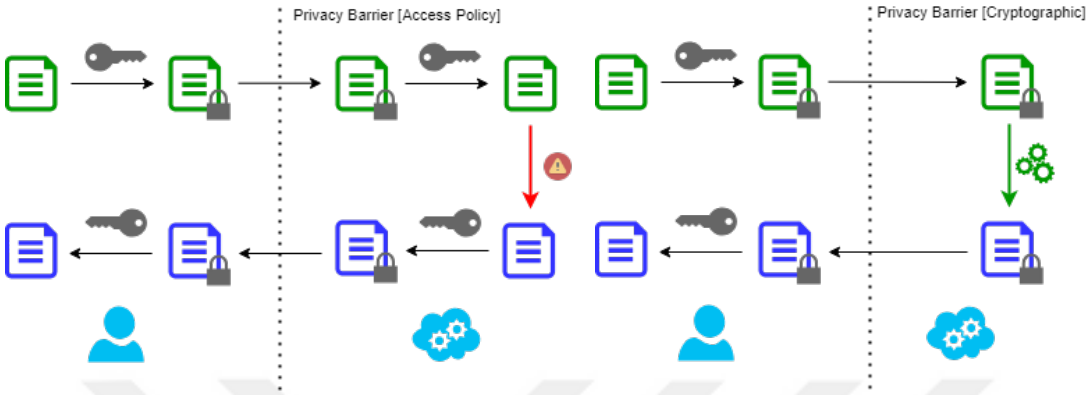
read the data. Symmetric-key encryption can be used for efficiently encrypting very large amounts of data and enables secure outsourced cloud storage and computing capabilities. However, any outsourced computation has to decrypt the data first before any computations can take place. Even though cloud services implement access policies to prevent unauthorized principals accessing data [42], it still does not guarantee the privacy of data or users in the presence of both internal or external threats [12] and data breaches [13].

HE allows for computations to be done directly on encrypted data, which is beneficial for data privacy on cloud computing services. The results of these encrypted computations remain encrypted, and only the data owner can decrypt them [43]. However, HE does have its limitations and drawbacks. It restricts the use case scenarios since only some computations on encrypted data are possible. Additionally, it is impossible to draw insights from encrypted data. There is also a significant performance overhead, which makes it infeasible to run very large and complex algorithms homomorphically.

Figure 2.3 contains data processing steps in cloud computing compared by both traditional and homomorphic encryption. Traditional encryption has its privacy barrier via access policies. However, that does not guarantee data privacy while carrying on computations over the data since the encrypted data needs prior decryption before computations and subsequent encryption after the computations. Contrary to traditional encryption, HE has a cryptographic privacy barrier where only data owners with private keys can decrypt the encrypted data, guaranteeing data privacy on the cloud.

While homomorphic encryption holds excellent potential for enabling privacy preserving cloud computing, it does have its limitations and drawbacks. This technology only allows for certain computations on encrypted data, which limits and complicates its usage. For example, it is impossible to gain insights from encrypted data due to the nature of the encryption. Furthermore, homomorphic encryption requires a significant amount of computational resources, making it infeasible to run large and complex algorithms homomorphically. Apart from its performance overhead, data encrypted with homomorphic encryption is much larger than unencrypted data, re-

sulting in a memory overhead. Therefore, it may not be practical to encrypt entire large databases with this technology. However, homomorphic encryption can still be helpful in scenarios where strict privacy requirements prohibit unencrypted cloud computations.



(a) Traditional (b) Homomorphic

Figure 2.3: Data processing in cloud computing services compared to traditional and homomorphic encryption. The figure is redrawn from Microsoft's documentation titled "Homomorphic encryption with SEAL" [44].

One of the main practical limitations of homomorphic encryption schemes is the multiplicative depth of circuits, which restricts the complexity of computations that can be performed over encrypted data. It is important to note that homomorphic encryption schemes are inherently malleable, meaning that they inherently have weaker security properties than non-homomorphic schemes. Malleability is a property of some cryptographic algorithms. An encryption algorithm is malleable if it is possible to transform a ciphertext into another ciphertext that decrypts to a related plaintext. For example, given an encryption of a plaintext m, it is possible to generate another ciphertext that decrypts to $f(m)$, for a known function f, without necessarily knowing or learning m [45].

2.2.2.1 Types of Homomorphic Encryption

Homomorphic refers to homomorphism in algebra, where each function in the plaintext space has an equivalent one in the ciphertext space and vice verse. Homomorphic

encryption includes multiple types of encryption schemes that can perform different classes of computations over encrypted data [43]. Some common types of homomorphic encryption are:

- **Partially homomorphic encryption** (**PHE**) supports only one type of operation (e.g., addition or multiplication) infinite times on the ciphertext. RSA [46], El-Gamal [47] and Paillier [48] are a few of the well-known PHE encryption schemes.
- **Somewhat homomorphic encryption** (**SWHE**) supports a limited number of addition and multiplication operations on the ciphertext [29].
- **Leveled fully homomorphic encryption** allows the evaluation of any operations over the ciphertext but there is pre-determined depth that limits operations [30].
- **Fully homomorphic encryption** (**FHE**) allows the evaluation of any operations infinite times over the ciphertext [49].

2.2.2.2 Tools: Libraries, Compilers, Frameworks

As described in section 2.1, studies on FHE paved the way for state-of-the-art tools such as HElib [50], SEAL [51], PALISADE [52], Lattigo [53], FHEW [54], TFHE [31], ALCHEMY [55], Encrypt-Everything-Everywhere (E^3) [56], EVA [57], CHET [58], nGraph-HE [59], SEALion [60] etc. that abstract cryptographic parts that require expertise in both cryptography and high-performance numerical computation. The following sections provide a brief overview of the tools employed in this study. For both problems, SEAL is the library running beneath the used tools.

2.2.2.2.1 The Simple Encrypted Arithmetic Library (SEAL)

In 2015, Microsoft SEAL [51] was introduced as a homomorphic encryption library implemented in C++ that allows operations such as additions, multiplications, shifts, and negations to be performed on encrypted integers and real numbers. However,

certain operations, such as encrypted comparison or sorting, cannot be evaluated on encrypted data using HE, limiting the library's implementation to only specific privacy-critical computation programs.

The library offers different homomorphic encryption schemes with varying properties, including the Brakerski-Gentry-Vaikuntanathan (BGV) [30] and the Brakerski-Fan-Vercauteren (BFV) [29] schemes which allow modular arithmetic to be performed on encrypted integers, and the Cheon, Kim, Kim, and Song (CKKS) [32] scheme which enables additions and multiplications on encrypted real or complex numbers but yields only approximate results. For applications that require precise values, such as financial and mathematical applications, the BGV and BFV schemes are more appropriate, while the CKKS scheme is more suitable for applications such as summing up encrypted real numbers, evaluating machine learning models on encrypted data, or computing distances of encrypted locations. SEAL offers leveled-FHE operations and does not implement bootstrapping for the offered schemes. Bootstrapping is a technique for resetting the noise level of a ciphertext to a fixed lower level. The source code is available under an MIT license, is well documented, and includes a wide range of examples.

2.2.2.2.2 Intel's Graph Compiler for Artificial Neural Networks with HE backend (nGraph-HE)

In this work, Intel's graph compiler for Artificial Neural Networks with HE-backend (nGraph-HE) [59,61,62], which supports the CKKS encryption scheme implemented by SEAL and presented in 2019, is adopted. The framework efficiently transforms neural networks into computational graphs for private inferencing and converts computations between the nodes into related FHE operations defined in SEAL. nGraph-HE also applies FHE-specific optimizations (e.g., constant folding, SIMD-packing, and complex-packing), graph-level optimizations (e.g., lazy rescaling and depth aware encoding), and run-time optimizations (e.g., bypassing special plaintext values). However, it requires the user to define FHE-related configuration parameters for rescaling operations. The motivation behind selecting nGraph-HE in this work is the fact that nGraph-HE is well-integrated with TensorFlow DL libraries, and it simplifies work-

ing on pre-trained neural networks in order to maintain privacy while making private inferences by abstracting HE operations such as key generation, encryption, decryption, key maintenance. nGraph-HE is available under the Apache License v2.0 and its repository is well-documented and includes examples.

2.2.2.2.3 The Encrypted Vector Arithmetic Language and Compiler (EVA)

EVA [57] was introduced in 2019, and it is an input language created explicitly for vector arithmetic and targets private arithmetic calculations using the CKKS scheme implemented by SEAL. Essentially, EVA serves as a compiler for FHE computations, automating the more complex aspects of the encryption such as key generation, encryption, decryption, parameter selection, ciphertext maintenance, etc. It provides users with a simple way to write programs that operate on encrypted data without needing to worry about encryption parameter selection, rescaling insertion, relinearization, and having access to the secret key. EVA converts the program into a term graph while transforming it with graph rewriting rules. However, depth-reducing transformations are not taken into account. While EVA can be used for any vectorized applications, its primary focus is on neural network inferencing.

CHAPTER 3

PRIVACY-PRESERVING ANOMALY DETECTION FOR A GROUP OF USER EQUIPMENT (UE)

The focus of this chapter is on one of the use cases of NWDAF, namely the detection of the Abnormal Behaviour Information (anomaly) for a Group of UEs, which is believed to be crucial for network security and service quality as hijacked or misused UEs can be the cause of abnormal behavior, e.g., abnormal traffic patterns, unexpected UE locations, unexpected transaction dispersion amounts, and wrong destination addresses. However, 3GPP does not provide details on how NWDAF will identify these abnormalities, leaving the implementation up to service providers.

The architecture shown in Figure 3.1 is devised for Abnormal Behaviour Detection for a Group of UEs where NWDAF instances make use of third-party cloud computing services while preserving data privacy. MTLF's secure model deployment to cloud services or its secure model training initiation process is not detailed in this work for the sake of conciseness. Therefore, model training is done on plaintext data, and the work is focused on AnLF's secure model inferencing conducted on encrypted data using homomorphic encryption. The synthetic dataset[1] generated in the work of Sevgican et al. is used in this part. The performance of the proposed architecture under various configurations is evaluated, and the results are compared with the results of the work of [19] in terms of accuracy and precision.

[1] https://github.com/sevgicansalih/nwdaf_data

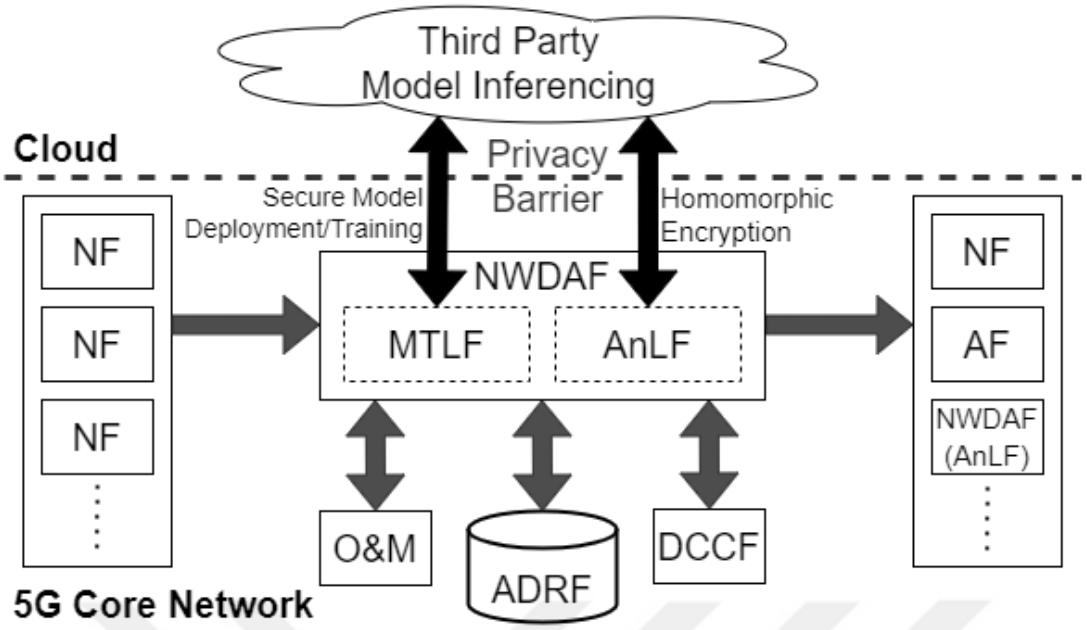


Figure 3.1: NWDAF architecture where MTLF either securely deploys trained model to the cloud or initiates secure model training on the cloud, and AnLF makes inferences on the cloud models with encrypted inputs.

3.1 Privacy Preservation with Deep Learning

Even though the nGraph-HE framework simplifies private inferencing, there are still engineering challenges related to the framework, SEAL, and HE. One of the challenges comes from HE, where HE computations are, by definition, data-independent, and this restricts branching on data. Meaning that calculations depending on the value of data cannot be done. This also prevents the usage of significant DL methods, such as the ReLU activation function or the minimum and maximum pooling layers in convolutional neural networks. Such DL models trained under these constraints are referred to as HE-friendly models. Another challenge comes from configuring SEAL which offers leveled HE operations that limit the multiplicative depth of DL models. As a result of this, complex DL models (e.g., RNN, LSTM) or DL models with a high number of layers cannot be used without proper configuration. A wrong configuration of the parameters will result in different private inferencing results from what is

expected since the library does not implement bootstrapping which is a crucial technique for refreshing ciphertexts to reduce the noise to a fixed level. Even though the trained DL model is capable of detecting anomalies, it can predict incorrectly due to the wrong configuration of HE parameters.

3.1.1 Dataset

The synthetic dataset used has the following fields:

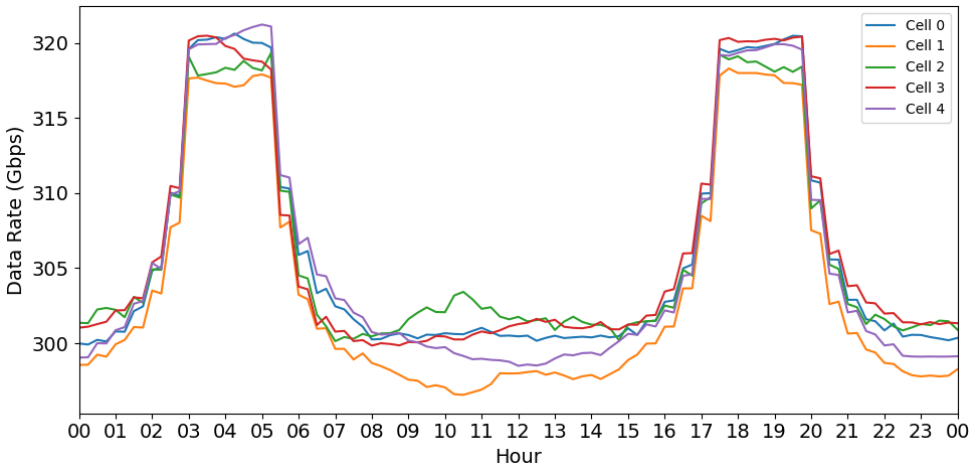
- **Time period**: 15-minutes interval
- **Cell identifier**: Network area information of remote radio unit (RRU) cell
- **Subscription category**: Platinum, gold, and silver
- **User equipment type**: IoT devices, vehicles, cell phones, smartwatches, and tablet computers
- **Data rate**: Aggregated data rate per cell identifier, subscription category, user equipment type in terms of Gbps
- **Anomaly information**: Whether aggregated data rate has anomaly or not

A row of the data contains the information about the data rate of a group of personal equipment within the specific subscription category connected to a particular cell in a specific time period, together with whether there is an anomaly or not. There are nearly 1.3 million rows in the dataset covering almost half a year. Sevgican et al. used fixed topology with 5 cells together with their adjacency information, and they simulated the mobility of user equipment when generating the data while considering personal equipment type, subscription category, and time of day. For example, the mean handover ratio of IoT devices is much lower than that of the vehicle due to the nature of this equipment. The mean handover ratios are the lowest between 22:00 and 06:00 as humans are less likely to move around at night time. Sevgican et al. also considered rush hours (07:00-09:30 and 16:00-20:00), where personal equipment has

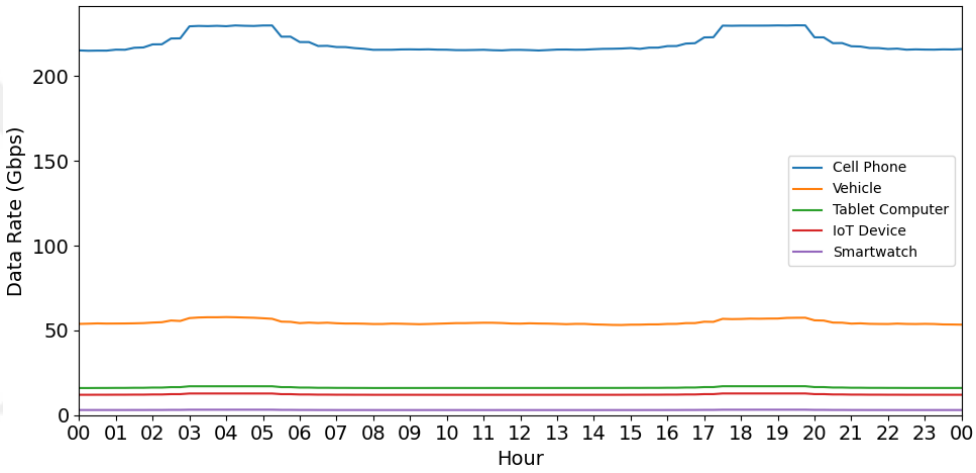
lower handover ratios compared to other times of the day except nighttime since the increased traffic affects our mobility and daily lives. In addition, data rates vary between subscription categories and personal equipment. For example, user equipment with platinum subscription category have higher data rates compared to ones with lower subscription categories, and people tend to buy better subscription categories for their cell phones rather than for their IoT devices.

Cell phones have the highest data rate, while smartwatches have the lowest data rate. The authors generate the dataset by aggregating each type of personal equipment with their subscription category within the cell. Then, they simulate the mobility of each personal equipment with their subscription categories with specified handover ratios between adjacent cells, and their aggregated data rates are used in the dataset. In addition, they introduced anomalies to specific time periods which results in unexpected increases in network traffic, and the anomaly in data rates fades and stabilizes in time.

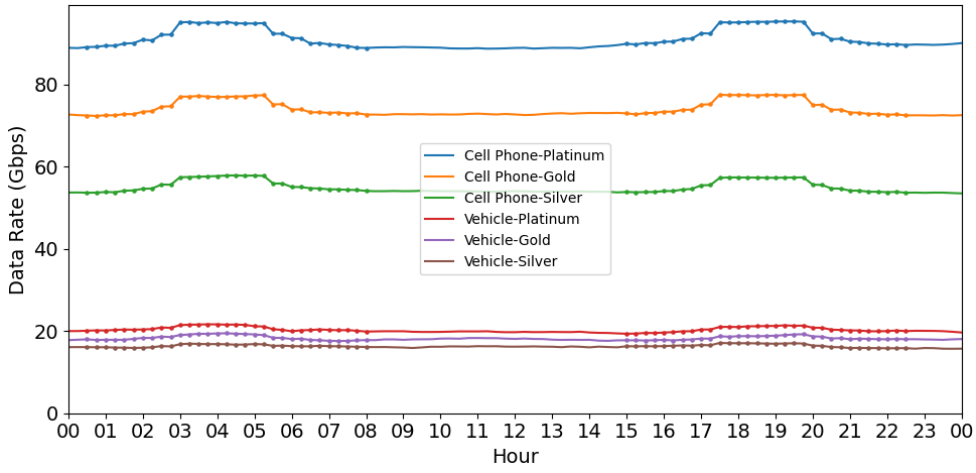
Figure 3.2 contains one day sample from the dataset. Aggregated data rates of each cell can be seen in Figure 3.2a, where the spikes represent anomalies in the data. In Figure 3.2b, the data rates of each personal equipment type connected to cell 1 are shown. The data rate of cell phones is the highest among personal equipment types. Figure 3.2c shows the data rates of cell phones and vehicles connected to cell 1, and their data rates are separated by their subscription categories. Dots on the curves show that there is an anomaly in the data and the anomaly diminishes with time as the data rates decrease.



(a) Aggregated data rates of each cell.



(b) Aggregated data rates of each equipment type connected to cell 1.



(c) Data rates of cell phones and vehicles by their subscription categories together with anomalies.

Figure 3.2: One day sample redrawn from the dataset [19].

3.1.2 Deep Learning Models

The dataset has been pre-processed before being used in the DL models. Time period and data rate values are normalized between zero and one. Also, RRU cell, subscription category, and personal equipment identifiers are converted into one-hot vectors. The main reason for the pre-processing of the dataset is to decrease the overfitting and underfitting of HE-friendly neural networks due to their limited capabilities. It is observed that normalizing the data and using one-hot vectors for categorical data had improved the trained models' accuracy and also decreased overfitting and especially underfitting probabilities. Since HE-friendly neural networks cannot directly use one of the most essential parts of the deep learning models, namely the ReLU activation function, the data was underfitting to the DL models. Even though convolutional layers have not been used in this work, HE-friendly DL models could not directly make use of some of the pooling types, such as minimum and maximum pooling, since branching is not possible on the encrypted data when HE is employed.

An example of raw and pre-processed input and output data can be seen in Figure 3.3. Raw input and output data have sizes of 5 and 1, respectively, whereas pre-processed input and output data have sizes of 15 and 2, respectively. The example data belongs to the vehicles with the platinum subscription category connected to cell 4, and their aggregated data rate/load equals 19.9095 Gbps where there is no anomaly.

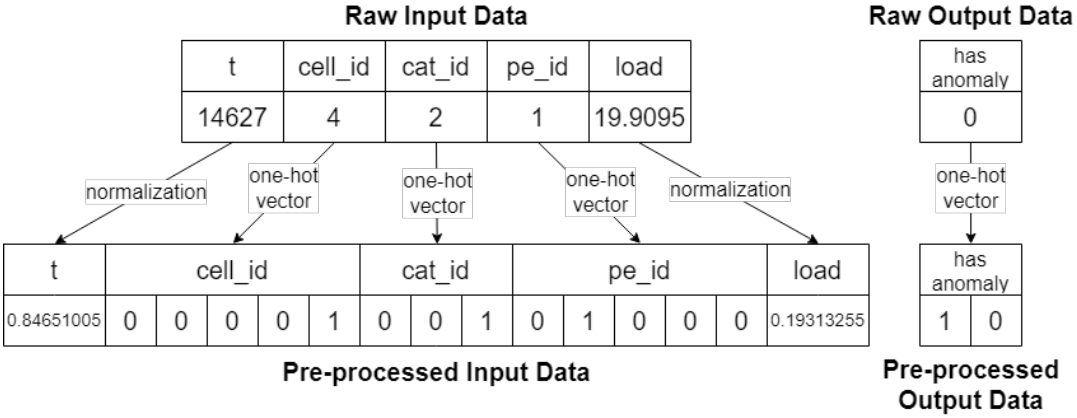


Figure 3.3: An example of raw and pre-processed input and output data.

Considering the restrictions of HE and DL, the models in this work make use of dense layers followed by polynomial activation functions except for the last layer. Outputs of the last layer are used in categorical cross-entropy loss function with the Adam optimizer with default values (0.001 learning rate, 0.9 $\beta 1$ and 0.999 $\beta 2$) as a stochastic gradient descent (SGD) method. The models take input data such as normalized time period, normalized load, RRU cell identifier as a one-hot vector, subscription category as a one-hot vector, and personal equipment information as a one-hot vector to predict whether the given data has an anomaly or not. Polynomial activation function has two learnable parameters that calculate $y = w_1 \cdot x + w_2$ where w_1 and w_2 are learnable weights, x and y are input and output, respectively. All of the weights (i.e., learnable parameters) have been initialized with the He normalization, also known as Kaiming initialization introduced by He et al. in [63]. Again, both L1 and L2 regularization methods are applied to kernel and bias weights in order to reduce overfitting and underfitting.

Figure 3.4 contains an example of a fully connected neural network (FCNN) architecture drawn with software from [64]. The architecture summarizes the DL models used in this work where the polynomial activation function follows every hidden layer. Input and output layers have 15 and 2 nodes respectively since the data is pre-processed.

Another limitation of the HE is that a limited number of multiplication operations can performed on the encrypted data as the error (noise) on ciphertext accumulates with each multiplication and the result will eventually become unreliable and impossible to decrypt to the intended value. Hence, the number of dense and activation layers in the DL models is kept as low as possible while the models are capable of classifying anomalies.

After model training and validation with plaintext data, the nGraph-HE framework is used for plaintext and ciphertext inferencing. First, the framework converts Tensorflow computations into computational graphs, and depending on the experiment type (i.e., plaintext or ciphertext inferencing), the framework converts computations on the graph into related operations using SEAL. It also does optimization on both computation graphs and HE calculations. For example, multiplications between inputs and weights are converted into HE multiplication operations between ciphertext

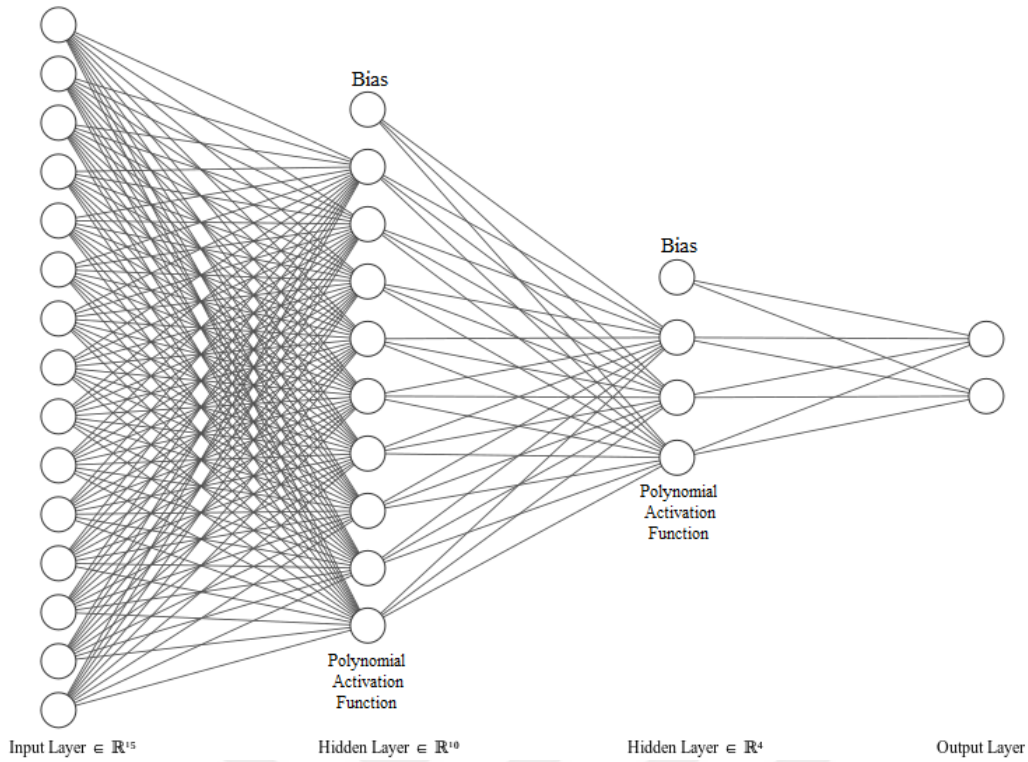


Figure 3.4: An example of a fully connected neural network (FCNN) architecture drawn with software from [64].

and plaintext implemented by SEAL. Then, the framework handles key generation, encryption, and decryption processes of input if the framework is configured to work with the HE scheme.

3.2 Results and Discussion

The ultimate goal of this part is to show that encrypted data can be used for inferencing and achieve similar results to previous works that use plaintext data. It is proposed that some of the functionalities of NWDAF in 5G networks can be offloaded to third-party cloud providers while preserving data privacy. In this perspective, the nGraph-HE framework is used to test trained DL models for inferencing on encrypted data. In addition to comparing ciphertext inferencing with plaintext inferencing in terms of success (accuracy, precision) and computational time overhead, experiments with

various configurations are conducted to analyze the impact of various configuration parameters on the results and runtimes.

3.2.1 Methodology

We used 70% of the dataset for training, 21% for validation, and 9% for evaluating our models' performances on both plaintext and ciphertext data. Multiple neural network models with similar architectures but with a different number of layers and parameters are evaluated together with a combination of encryption-related parameters. There are a total of 102 different experiment configurations where each configuration was run at least 20 times, and the results such as running times, accuracy, and precision were averaged.

nGraph-HE and SEAL-related encryption parameters are as follows:

- **Polynomial modulus degree** represents the degree of a power-of-two cyclotomic polynomial. Increasing the polynomial modulus degree allows us to increase the maximum number of input data, meaning that more encrypted data can be processed for anomaly detection.
- **Coefficient modulus** is an integer modulus that is constructed as a product of multiple distinct prime numbers whose bit lengths are described in the configuration.
- **Security level** represents the security level of the encryption scheme, which can be 128, 192, and 256 bits.
- **Complex packing** enables usage of both imaginary and real parts of complex components of the plaintext encoding map, resulting in doubling the throughput since two plaintext values are decrypted into one complex value.

The choice of encryption parameters significantly affects the performance, capabilities, and security of the encryption scheme. Using a larger polynomial modulus

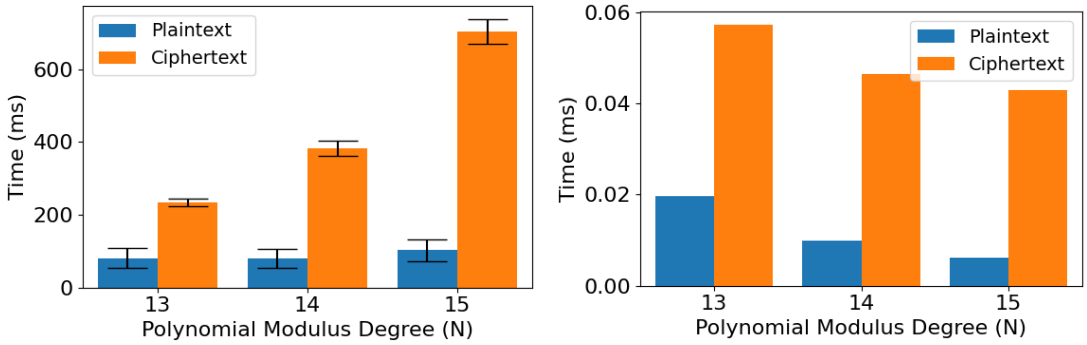
degree together with complex packing allows us to work on more elements without decreasing the security level with the performance degradation.

Experiments were run on an Intel Xeon Silver 4114 server with a 2.20 GHz CPU and 64 GB RAM. In the experiments, different models are trained and the resulting models are used with different configurations in plaintext and ciphertext inferencing. There are 6 different models where there are 2 different layer counts together with 3 different parameter count options. Models with 5 and 7 layers were considered sufficient for this task. The former model has 3 dense layers followed by 2 activation layers except for the last (output) layer. In contrast, the latter model has 4 dense layers followed by 3 activation layers except the last layer. The number of layers can be increased when dealing with much more complex problems. Still, polynomial modulus and coefficient modulus parameters must be configured accordingly since arithmetic operations on the encrypted data accumulate the error, and the results may deviate after a certain number of operations due to lack of bootstrapping. In the sequel, the experimental results are presented to analyze the impact of various HE parameters on the runtime.

Further details of all configurations can be found in Appendix A, where averaged runtimes and amortized running times of all experiments are illustrated in Figure A.1 and Figure A.2, respectively.

3.2.2 Impact of Polynomial Modulus Degree (*N*)

Figure 3.5 shows the averaged and amortized runtimes of a model with 5 layers and 452 trainable parameters, and the ciphertext inferencing is done with the 128-bit security level and three different polynomial modulus degrees (N): 13, 14, and 15. Since the polynomial modulus degree determines the maximum input size (2^{N-1}), it is observed that increasing the polynomial modulus degree increases the total runtime. However, the amortized runtime of input decreases with the increase of polynomial modulus degree.



(a) Averaged

(b) Amortized

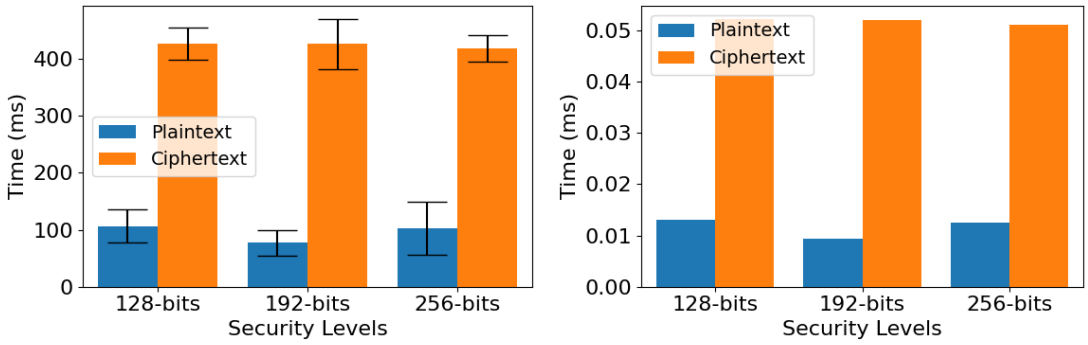
Figure 3.5: Running times of a DL model with 5 layers and 452 parameters with 128-bit security where three different polynomial modulus degrees (N) are compared.

3.2.3 Impact of the Security Level

Figure 3.6 illustrates the averaged and amortized runtimes of a model with 5 layers and 452 trainable parameters, and the ciphertext inferencing is done with polynomial modulus degree of 14 and three different security levels: 128-bits, 192-bits, and 256-bits. It is observed that changing the security level does not affect the total runtime. It is vital to keep the same number of the coefficient modulus. Otherwise, the configuration with a higher number of coefficient modulus will perform slower regardless of its security level.

3.2.4 Impact of Deep Learning Layers and Parameter Count

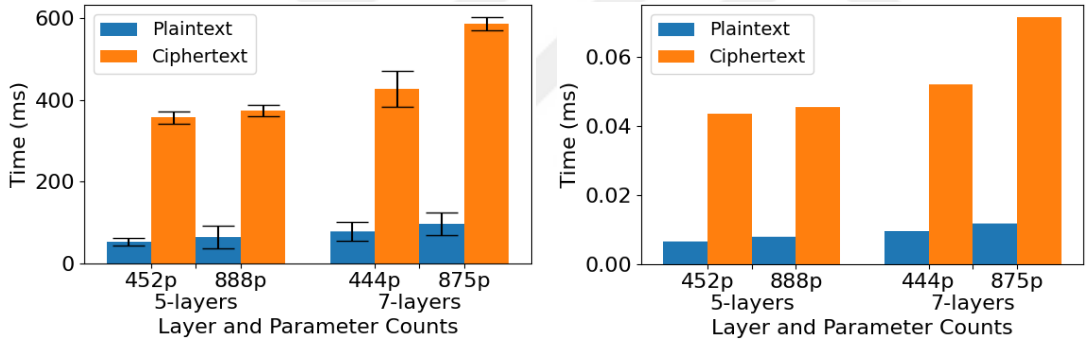
In Figure 3.7, the averaged and amortized runtimes of four different models with 5 and 7 layers where their trainable parameter counts are close to 450 and 880 are presented, and the ciphertext inferencing is done with polynomial modulus degree of 15 and security level of 192-bits. It is observed that introducing new layers into the DL models or increasing the parameter count in the DL models increases the running times. The parameter count increases the number of arithmetic operations (i.e., mul-



(a) Averaged (b) Amortized

Figure 3.6: Running times of a DL model with 5 layers and 888 parameters with a polynomial modulus degree of 14 where three different security levels are compared.

tiplication, summation) executed on the encrypted data. However, the layer count has much more impact on the running times as it also increases the multiplicative depth of the encrypted data.



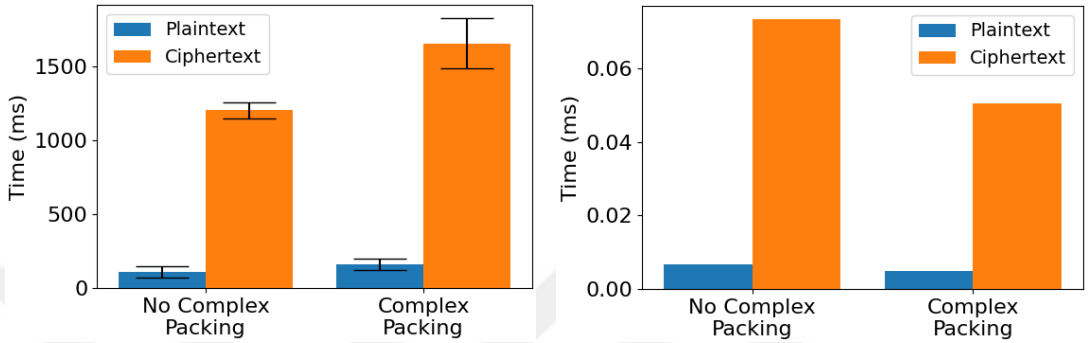
(a) Averaged (b) Amortized

Figure 3.7: Running times of 4 different DL models with layer counts of 5 and 7, approximately 450 and 880 parameters with polynomial modulus degree of 15 and 192-bits security level where numbers of parameters and layers are compared.

3.2.5 Impact of Complex Packing

Figure 3.8 presents the averaged and amortized runtimes of testing the complex pack-

ing option with a model of 7 layers and 1200 parameters where ciphertext inferencing is done with polynomial modulus degree of 15 and 256-bit security level. When complex packing is disabled, the maximum input number is equal to 2^{N-1} where N is the polynomial modulus degree and equals 15. The maximum input number is doubled when complex packing is enabled. Therefore, averaged running time increases with enabling complex packing. Yet, complex packing reduces the amortized running time.



(a) Averaged (b) Amortized

Figure 3.8: Running times of a DL model with 7 layers and 1200 parameters with polynomial modulus degree of 15 and 256-bits security level where complex packing feature is compared.

3.3 Evaluation of the Privacy Preservation

The average time for training a DL model with plaintext data together with the validation step is 24.98 ± 2.87 seconds where it takes 5 epochs to train a DL model. As shown in the experiment results, inferencing encrypted data with HE increases computational time cost due to the nature of HE. Averaged and amortized runtimes of inferencing on plaintext and ciphertext data are shown in Table 3.1. Privacy-preserving inferencing with HE increases computational runtimes by a magnitude of 5 in terms of amortized runtimes. In contrast, it increases running time by the magnitude of 6 on average without considering amortized running times. Note that the results are averaged for all experiments. Meaning that runtimes will be increased or decreased

depending on the selected configuration parameter and DL model size/complexity.

Average accuracy and precision values for training, validation, and testing of all experiments are shown in Table 3.2. Average plaintext and ciphertext testing accuracy and precision values are equal to 0.776 ± 0.01. The average approximation value between plaintext and ciphertext inferencing is $1.271 \times 10^{-8} \pm 2.59 \times 10^{-8}$. It can be concluded that the accuracy and precision difference between plaintext and ciphertext output values is negligible.

Table 3.1: Averaged and amortized time results of all experiments.

	Plaintext	**Ciphertext**
Time	103.47 ± 44.57 ms	653.47 ± 376.13 ms
Amortized	0.01 ± 0.006 ms	0.049 ± 0.018 ms

Table 3.2: Average accuracy and precision results of all experiments.

	Training	**Validation**	**Plaintext & Ciphertext**
Accuracy	0.775 ± 0.002	0.775 ± 0.006	0.776 ± 0.01
Precision	0.775 ± 0.004	0.774 ± 0.006	0.776 ± 0.01

Sevgican et al. state in [19] that their base logistic regression (LR) algorithm achieves 0.556 accuracy and 0.769 precision while their XGBoost algorithm achieves 0.626 accuracy and 0.773 precision. Average plaintext and ciphertext testing accuracy and precision values are 0.776 ± 0.01. Comparing the results of both works, it can be concluded that using encrypted data with HE for inferencing does not decrease the trained model's accuracy and precision. However, it introduces a running time overhead. The abnormal behavior detection for a group of UEs can be executed in a public cloud while preserving data privacy. However, the use of HE is restricted to the analysis of historical data in this work, and it might require further performance enhancements in HE-based schemes for use in real-time use cases.

CHAPTER 4

PRIVACY-PRESERVING MOBILITY ANALYTICS FOR USER EQUIPMENT (UE)

By the technical specifications of 3GPP, NWDAF performs data analytics to provide UE mobility statistics or predictions by collecting UE mobility-related information from NFs and OAM. The focus of this chapter is to provide mobility analytics for user equipment (UE) while NWDAF makes use of third-party cloud computing services whilst preserving data and user privacy with FHE. In the rest of this chapter, a general privacy-preserving methodology is presented in detail, and it is proposed to be used for UE mobility statistics. With the privacy-preserving methodology, connected components in any undirected graph can be discovered and the discovered connected components can be used for UE mobility analytics. For example, it can provide information on UE mobility anomalies where UE connects to different cellular networks within an impossible time window (i.e., UE connected to a cell in Ankara connects to a cell in Istanbul).

The topology of cellular networks can be represented with island representation where each cellular grid can be represented with a land piece, and it forms an island or connected component with its neighboring cells. Then, the network service provider can check the mobility activities of its user equipment. As shown in figure 4.1, there are example topologies for cellular networks in Ankara and Istanbul. The network service provider sends its encrypted network topologies to a third-party cloud computing service, which computes connected components from the encrypted topologies. After that, the network service provider can analyze UE mobilities or check whether the connected UE shows suspicious activities, such as connecting to a base station in a

different topology.

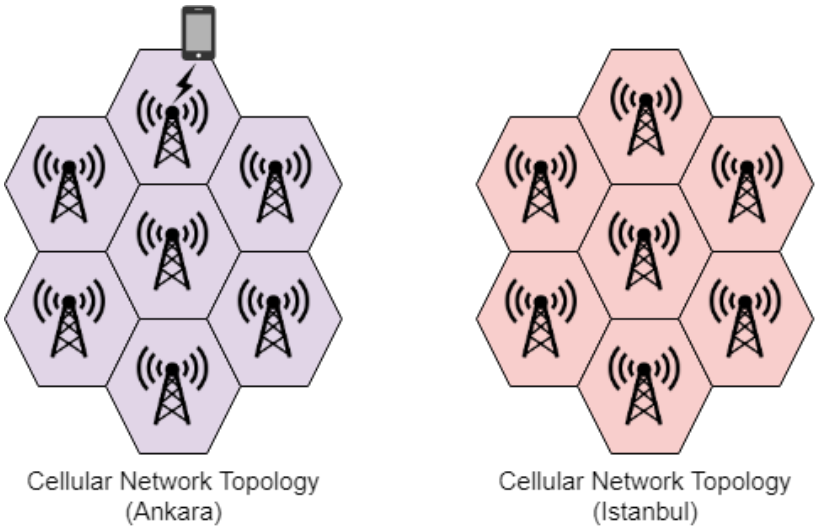
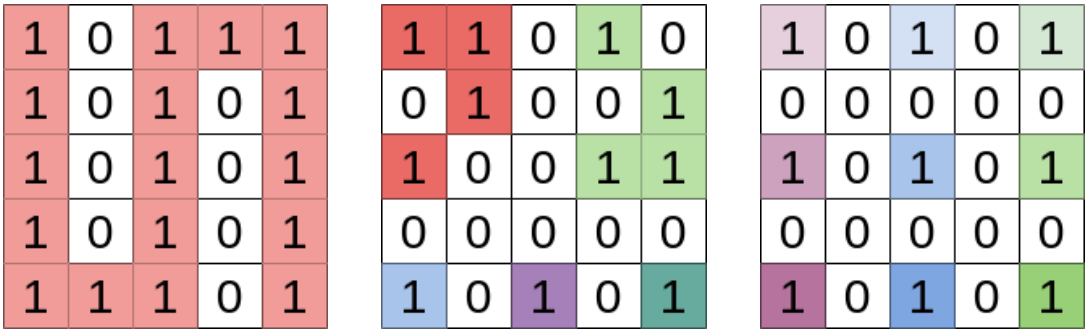


Figure 4.1: Example cellular network topologies.

4.1 Finding the Islands in a Graph

The problem is finding the islands in the given two-dimensional (2D) $0-1$ (binary) matrix where 1s denote land pieces whereas 0s indicate water. Each land piece can be connected to 8 neighboring land pieces and a group of connected land pieces forms an island. In Figure 4.2, there are three different 2D $0-1$ matrices together with their island counts. This is a variation of the problems of finding the connected components in an undirected graph and connected-component labeling (CCL). A connected component of an undirected graph is a subgraph in which every two vertices are connected to each other by at least one path, and there are no such vertices that are connected to any other vertices outside of the subgraph.

For an $N \times N$ sized $0-1$ (binary) matrix $\mathbf{M}$, the cell with the coordinate (x, y) where $0 \leq x \leq N-1$ and $0 \leq y \leq N-1$ are denoted as $\mathbf{M}[x, y]$, and $\mathbf{M}[x, y]$ represents a land piece if its value is equal to 1. Otherwise, it represents water. For cell with the coordinate (x, y), surrounding eight cells with the coordinates (x-1, y-1), (x, y-1), (x+1, y-1), (x-1, y), (x+1, y), (x-1, y+1), (x, y+1), (x+1, y+1) are called the



(a) 1 island (b) 5 islands (c) 9 islands

Figure 4.2: Example 2D matrices together with their island counts.

8-neighbors of the cell. Two land cells p and q are on the same island if there is a path that consists of land cells $m_1, m_2, \ldots, m_n$ such that $m_1 = p$, $m_n = q$ and m_i and m_{i+1} are 8-neighbors of each other for all $1 \leq i \leq n - 1$. Even though the problem is straightforward, finding islands in an encrypted $N \times N$ sized $0 - 1$ matrix while preserving data privacy with HE is another challenge.

4.2 HE-Friendly Methodology

Even though the problem can be solved with a bounded (at most number of nodes) number of traversal calls, any of the traversal algorithms cannot be used with HE since branching on encrypted data is not possible with HE naturally. In graph traversal algorithms, it is required to check the values of the cells in order to decide whether to include those cells in the connected component or not. However, checking the cell's value is not possible in HE since that value will be encrypted and cannot be decrypted by the computation service. Thus, a HE-friendly approach is required to solve this problem.

Adjacency matrix representation of the 2D matrix is needed as it provides adjacency information, and the distances matrix is calculated from the adjacency matrix. After the distances matrix is obtained, it will provide reachability information between the nodes.

Let **M** be a 2D binary input matrix with size of $n \times n$, **A** and **D** be the adjacency and distances matrix representations of **M** with size of $N \times N$ where N equal to n^2 respectively. Let denotations $\mathbf{G}[i]$ and $\mathbf{G}[i, j]$ represent the i^{th} row, the value at i^{th} row and j^{th} column of the 2D matrix **G** respectively. The distance matrix **D** initially equals the adjacency matrix **A**, and it will provide reachability information between nodes after the calculations. Then, each row of the distances matrix **D** is updated for $t = 1, 2, \ldots, \lceil N/2 \rceil$ as $\mathbf{D}^{t+1}[r] = \sum_{c=1}^{N} \mathbf{D}^{t}[r, c] \cdot \mathbf{A}[c]$, where each row is iteratively updated $\lceil N/2 \rceil$ times, which is for repeating the summation for the worst-case scenario. $\mathbf{D}^1$ equals **A**, and $\mathbf{D}^{\lceil N/2 \rceil}$ equals the distance matrix **D** after the iteration. $\sum_{c=1}^{N} \mathbf{D}^{t}[r, c] \cdot \mathbf{A}[c]$ will be done $\lceil N/2 \rceil$ times. $\lceil N/2 \rceil$ is equal to or greater than the length of the longest possible path in a square matrix. After every row of the distances matrix is updated, it is possible to find the islands.

In Figure 4.3, there is an example of calculating the distances matrix for the node p without doing any repetition. Even though the graph is directed and not a 2D matrix, it might be a good example to demonstrate the calculation of the distances matrix. The calculation done in the figure is just $\mathbf{D}[1] = \sum_{c=1}^{4} \mathbf{D}[1, c] \cdot \mathbf{A}[c]$. At the beginning, node *p* can only reach node *q*, but after the calculations, it can be seen that node *p* can reach every other node including itself.

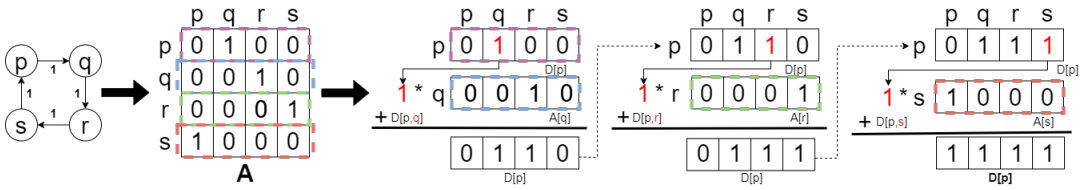


Figure 4.3: Directed graph with 4 nodes together with its adjacency matrix and partial calculation of its distances matrix for the first row.

Islands can be formed from the distances matrix as it contains the connectivity information of every node such that it can be checked whether a node is connected to another node or not. If $\mathbf{D}[n \cdot i + j, n \cdot x + y]$ is greater than 0, that means the node with (i, j) coordinate can reach the node with (x, y) coordinate meaning that those nodes belong to the same island. Figure 4.4 includes an example with 3×3 input matrix and its binary distances matrix. There are 5 land pieces whose coordinates are $(0, 0)$, $(0, 2)$, $(1, 2)$, $(2, 0)$ and $(2, 1)$. Only by examining the input and the distances

matrices, it is possible to infer that there are 2 islands where one of them consists of only a node with coordinate $(0, 0)$ indicated with red color, and the other island, indicated with purple color, is formed by the rest of the nodes

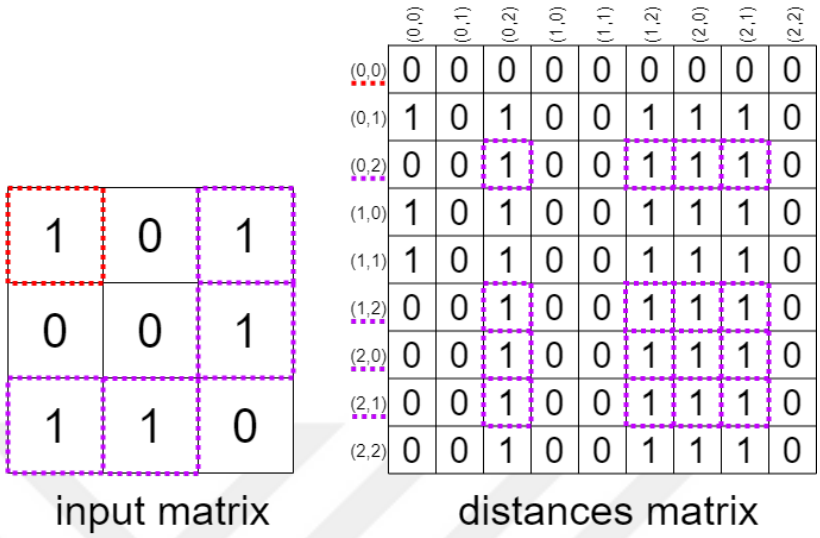


Figure 4.4: Sample 3×3 input matrix and its distances matrix.

The problem and the HE-friendly methodology can be easily extended to any $N \times M$ sized 2D matrices or any undirected graphs.

4.3 Implementation

Details of HE-friendly methodology for the problem are explained in the previous section. In this section, approaches for the FHE solution are discussed and implementation details of those approaches are detailed. EVA compiler backed with the CKKS scheme of SEAL is used for the implementation of the aforementioned methodology. EVA program works on encrypted vectors and only supports basic vector operations such as addition, multiplication, negation, and shift. While implementing the methodology, it is observed that EVA can be used in two different ways in terms of feeding the encrypted input into the EVA program. One of them is a vectorized approach whereas the other one is a parallelized approach. In the vectorized approach, encrypted input data is used in the same vector while each element of encrypted input is

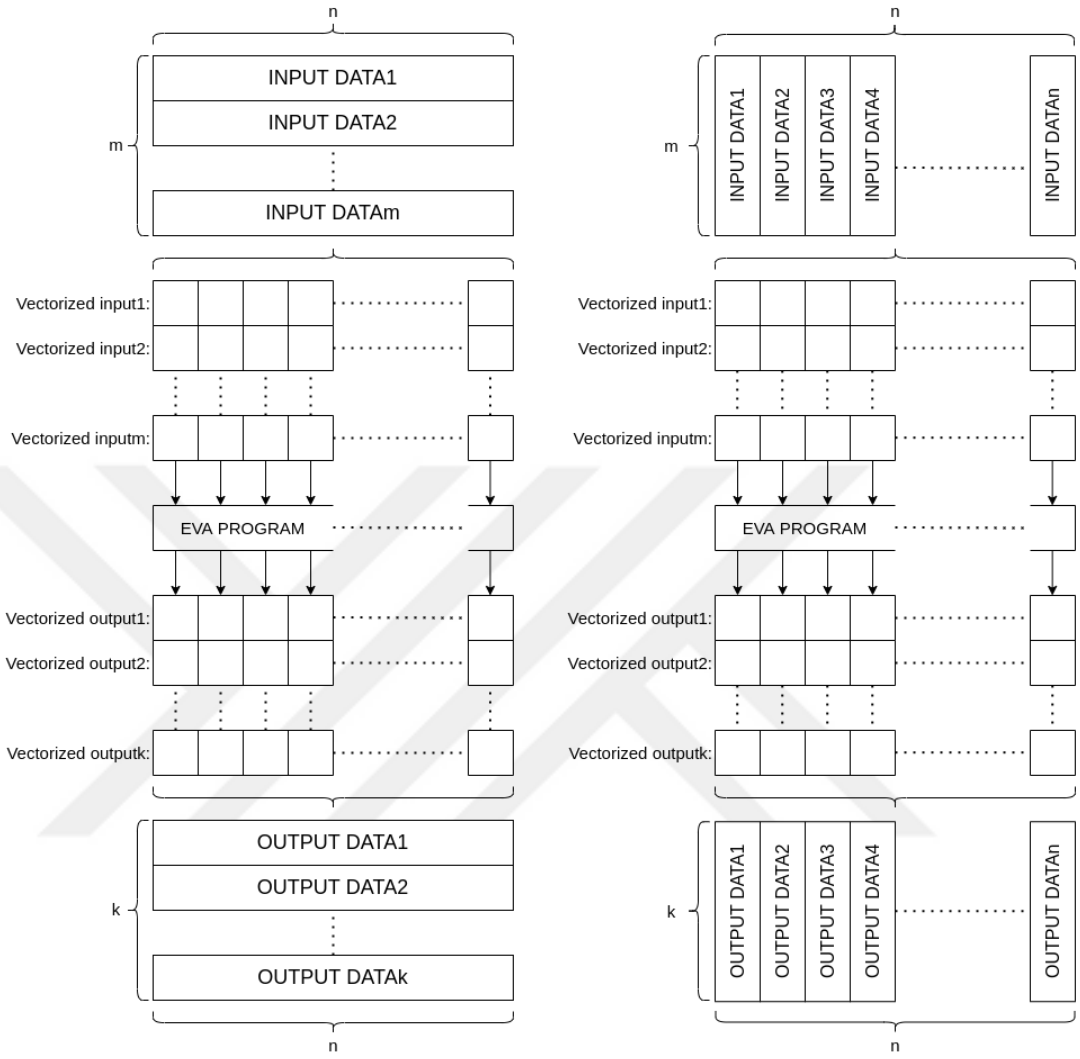
used in different vectors in the parallelized approach. Differences between those two approaches can be seen in Figure 4.5.

In the FHE implementation, the client first converts the 2D $0-1$ input matrix into its adjacency matrix representation. (For the parallelized approach, the client can omit the adjacency matrix conversion and leave the conversion process to the server.) Then, it generates public and private keys according to the HE scheme and desired security level. Then, it encrypts the adjacency matrix (or input matrix) with the generated public key. Afterward, the client sends the encrypted adjacency matrix (or input matrix) together with the plaintext information of the number of cells in the input to the server. The server runs the related EVA program that calculates $\mathbf{D}^{t+1}[r] = \sum_{c=1}^{N} \mathbf{D}^{t}[r,c] \cdot \mathbf{A}[c]$ defined in Section 4.2 for every row in order to obtain the distances matrix of the input adjacency matrix (after converting the input matrix to its adjacency matrix representation). It sends the calculated distances matrix back to the client. Later, the client decrypts the distances matrix using the secret key and obtains the plaintext distances matrix of the input. Finally, the client can find the islands using only the input matrix and the received distances matrix.

For sanity check, there is an additional and optional evaluation step that runs the same EVA program on the plaintext data directly. This is needed since the CKKS scheme does approximate the values, and if inadequate encryption parameters (such as small modulus coefficients with wrong security level) are used the resulting values can differ a lot from the expected values. Therefore, the decrypted distances matrix received from the server and the reference distances matrix obtained from the evaluation step are compared in terms of their Mean Squared Error (MSE) scores. Lesser the MSE means the EVA program works smoothly.

Yet, the server cannot obtain the distances matrix due to insufficient bit modulus as the size of the input 2D matrix is increased due to the growing number of multiplications over encrypted values in the calculation $\mathbf{D}^{t+1}[r] = \sum_{c=1}^{N} \mathbf{D}^{t}[r,c] \cdot \mathbf{A}[c]$ defined in Section 4.2. That is one of the drawbacks of FHE which is multiplicative depth limitation. The reason for that is the fact that the library (SEAL) underneath the EVA compiler offers leveled-FHE operations in order to overcome the performance penalties of bootstrapping [33] by choosing sufficiently large parameters to evaluate

the entire computation without bootstrapping. However, there is a cut-off point after which the bootstrapping technique is more efficient.



(a) Vectorized Approach

(b) Parallelized Approach

Figure 4.5: Vectorized and parallelized approaches with their input and output differences.

This issue is overcome in the implementations by introducing additional communication steps between the server and the client such that the distance matrix is calculated after $N.\lceil N/2 \rceil$ times ciphertext exchange where N is the number of nodes.

The EVA, SEAL, and CKKS-related configuration parameters are as follows: Balance

reduction on, always rescaling, lazy relinearization, 128-bits of security, 30 fixed-point input scale, and the maximum ranges of 30-bit coefficients modulus in outputs [42].

Depending on the approach, the implementations are tested with 2D square matrices of sizes 1, 4, 9, 16, and 25. A randomly initialized 2D $0-1$ matrix is used in each test, and tests are simulated 100 times. Compile, key generation, encryption, execution, decryption, reference execution times, and MSE scores are recorded for each test. Their averages and standard deviations are used in the results.

4.3.1 Methods

Implementing the FHE-friendly methodology with EVA and SEAL is not as simple as coding it with one of those programming languages since branching on encrypted values is not possible and EVA makes calculations with vectors. Meaning that the problem needs to be solved using only vectors which introduces another level of complexity for implementation. For that reason, the following methods are devised:

1. Reduce the number of 1s in the 2D input matrix (Optional Method)

2. Obtain the adjacency matrix from the 2D input matrix or reduced matrix

3. Calculate the distances matrix from the adjacency matrix

4. Find the islands from the input matrix or the reduced matrix and distances matrix

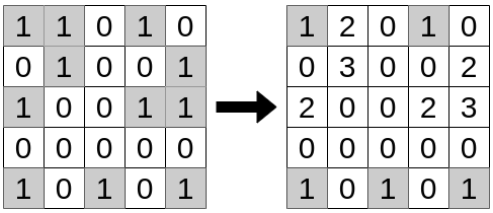
In the following subsections, the methods above are explained in detail. For simplicity, small-sized matrices (such as 2×2, 3×3) are used as example input for visualization purposes.

4.3.1.1 Reducing Ones (Optional)

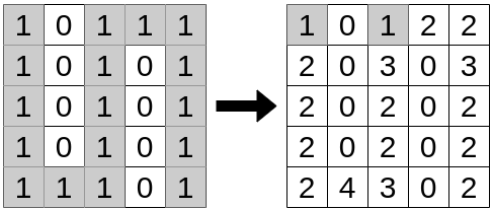
This step is optional as islands can be found with either the original input matrix or the reduced matrix. The main reason for this method is to reduce the ones in the input matrix so that the client can solve the problem with much fewer computations after it receives the distances matrix from the server. The input matrix is converted into the reduced matrix with the following calculation:

Let G be a boolean 2D input matrix of size $n \times n$, R be the reduced matrix of size $n \times n$, notation $\mathbf{G}[i, j]$ represent the cell value of i^{th} row and j^{th} column in matrix G and out of bound elements of matrices are assumed to be zero (e.g. $\mathbf{G}[x, y]$ equals to zero if $x \notin [1, n]$ or $y \notin [1, n]$). So, each element of $\mathbf{R}[i, j]$ can be calculated as $\mathbf{G}[i, j] \cdot \Big(\mathbf{G}[i-1, j-1] + \mathbf{G}[i-1, j] + \mathbf{G}[i-1, j+1] + \mathbf{G}[i, j-1] + \mathbf{G}[i, j]\Big)$, where the number of 1 valued cells is decreased as the $\mathbf{R}[i, j]$ value is the multiplication of the corresponding cell in G, $\mathbf{G}[i, j]$, with the summation of some neighboring cells (such as left ($\mathbf{G}[i, j-1]$), upper left ($\mathbf{G}[i-1, j-1]$), upper ($\mathbf{G}[i-1, j]$) and upper right ($\mathbf{G}[i-1, j+1]$) neighbors) together with the cells own value ($\mathbf{G}[i, j]$). Figure 4.6 contains two examples of this calculation.

The reduced matrix can also be used to find the approximate number of islands or to set an upper bound for island counts. Multiple reduce ones steps with different neighboring settings and matrix traversal methods can be used for approximation. The output with the minimum number of ones can be used as an approximate number of islands. The approximation might be useful for some cases, but the upper bound for the approximation ratio might need to be found.



(a) Input matrix with 11 ones and 5 islands is converted into the reduced matrix with 5 ones.

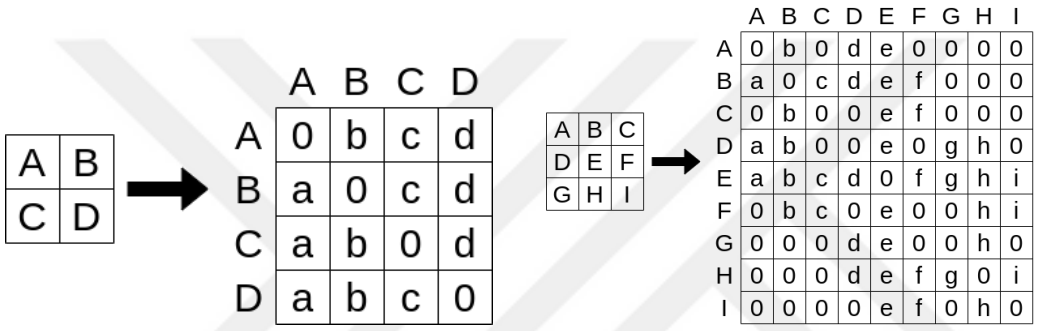


(b) Input matrix with 17 ones and 1 island is converted into the reduced matrix with 2 ones.

Figure 4.6: 2D matrices and their conversion into reduced matrices.

4.3.1.2 Adjacency Matrix Representation

The input matrix or the reduced input matrix must be converted into its adjacency matrix representation to find the distances between each cell or node. Each cell in the 2D matrix is treated as a node so that the $n^2 \times n^2$ sized adjacency matrix is created from the $n \times n$ sized input matrix. Cell with (i, j) coordinates in the input matrix has the index value of $i \cdot n + j$ in rows and columns of the adjacency matrix. In Figure 4.7, there are examples of adjacency matrix conversions.



(a) 2×2 matrix and its 4×4 adjacency matrix. (b) 3×3 matrix and its 9×9 adjacency matrix.

Figure 4.7: 2D matrices and their conversions into adjacency matrices.

4.3.1.3 Distances Matrix Calculation

Since rows of the adjacency matrix give 1-hop distance information, it is possible to calculate distances from each node to every other reachable node. After distances are obtained, the distances matrix is used to check whether there is a path from a node to other nodes. The distances matrix is initialized with the adjacency matrix, and then the rows of the distances matrix are updated with the calculation $\mathbf{D}^{t+1}[r] = \sum_{c=1}^{N} \mathbf{D}^{t}[r, c] \cdot \mathbf{A}[c]$ defined in Section 4.2.

4.3.1.4 Finding Islands

In this method, the client has an input matrix, reduced matrix (optionally), and distances matrix. Using these, the client can find the number of islands. The distances matrix has the connectivity information of every node so that the client can check whether a node is connected to another node. The client starts by checking 1 valued node/cell and it populates every reachable 1 valued node/cell from that cell. Every populated/grouped 1's represents an island. Using a reduced matrix instead of an input matrix helps to save additional computations on the client side since it reduces the number of 1s that need to be checked in the input matrix.

Notation of $\mathbf{D}[(i,j),(x,y)]$ is same as $\mathbf{D}[i \cdot n + j, x \cdot n + y])$ for $n \times n$ sized matrix D.

Figure 4.8 includes an example input matrix and its reduced version together with the distances matrix. Without using the reduced input matrix, the client needs to check 5 land (1-valued) cells with the coordinates of $(0,0)$, $(0,2)$, $(1,2)$, $(2,0)$ and $(2,1)$ in order to form islands.

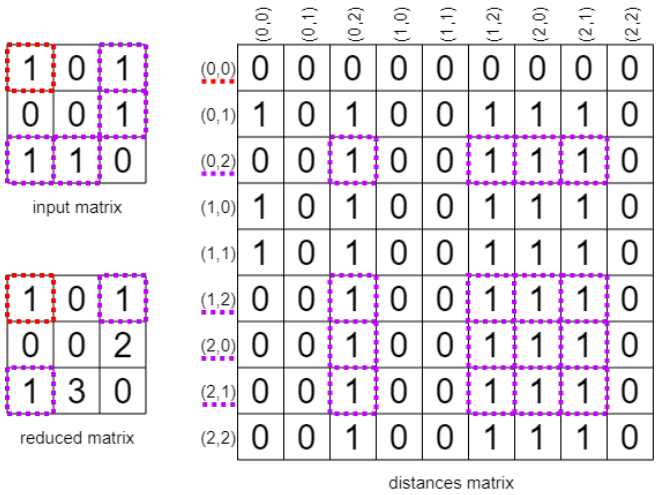


Figure 4.8: Input matrix and its reduced version with distances matrix. Note that 1 value in the distances matrix means that there is at least one path from the cell in the row to the cell in the column.

1. The client checks whether $(0, 0)$ has a path to any other cells and decides there is no path to other cells since every value in $\mathbf{D}[(0, 0)]$ is equal to 0. Then, $(0, 0)$ forms an island by itself.

2. There are 4 cells with the coordinates of $(0, 2)$, $(1, 2)$, $(2, 0)$ and $(2, 1)$ left, and then then client checks whether $(0, 2)$ has path to any other cells. Since $\mathbf{D}[(0, 2), (1, 2)]$, $\mathbf{D}[(0, 2), (2, 0)]$ and $\mathbf{D}[(0, 2), (2, 1)]$ are not equal to zero, those 4 cells form an island.

3. There are 2 islands.

With the reduced input matrix, the client can form islands by only checking the distance information of 3 cells with the coordinates of $(0, 0)$, $(0, 2)$, and $(2, 0)$.

1. The client checks whether $(0, 0)$ has a path to any other cells and decides there is no path to other cells since every value in $\mathbf{D}[(0, 0)]$ is equal to 0. Then, $(0, 0)$ forms an island by itself.

2. There are 2 cells with the coordinates of $(0, 2)$ and $(2, 0)$ left, and then the client checks whether $(0, 2)$ has path to any other cells. Since $\mathbf{D}[(0, 2), (2, 0)]$ is not equal to zero, those 2 cells form an island.

3. There are 2 islands.

Using the reduced input matrix reduces the number of 1-valued cells which are also used in forming islands. As a result, there is less computation on the client side.

4.3.2 Implementation Details

Details of the FHE-friendly methodology for the problem are explained in the previous sections. In this part, FHE programs written with EVA compiler backed with CKKS scheme of SEAL library are explained in detail. There are 2 types of EVA Programs implementing the Reduce Ones and Compute Distances methods and 3 different approaches: *Vectorized*, *Parallelized*, and *Parallelized without Loops*. For the

approaches, the EVA Programs have slight differences in terms of implementation but the underlying principles are the same as explained in the previous methods section. Before describing the EVA programs, these are the common concepts in the programs:

- **Client:** The client first generates public and private keys with defined configuration parameters. Then, it encrypts the input data which will be used by the EVA program on the server. After the data is encrypted, it is sent to the server which executes the compiled version of the EVA program over the encrypted data. Finally, the received results are decrypted with the private key. For the sanity checking and the evaluation of the FHE programs, there is an evaluation process that runs the same EVA program on the data itself without any encryption. This evaluation process is needed as the CKKS scheme does approximate the values, and if inadequate encryption parameters (such as small modulus coefficients with wrong security level) are used the resulting values can differ a lot from the expected values. Therefore, the EVA programs' output and reference output are compared in terms of their Mean Squared Error (MSE) scores. Lesser the MSE means the EVA program works as intended.
- **Server:** The server runs the compiled version of the EVA program on encrypted inputs and sends back encrypted outputs of the program to the client. The server never decrypts the encrypted inputs regardless of the required computations.

There are 3 implementations for solving the problem: **Vectorized** Implementation, **Parallelized** Implementation, and **Parallelized** Implementation **without Loops**. Implementation details and their differences are explained in the following subsections. As illustrated in previously shown Figure 4.5, EVA Programs can be implemented with the two approaches: **Vectorized** and **Parallel**.

- **Vectorized:** Multiple input data are vectorized and each vector element is passed into the EVA Program. Therefore, the EVA Program is needed to be written accordingly as the program will process only vector elements. Shift operations are much more suitable for this approach as shifting enables accessing adjacent vector elements of the same input data. One drawback of this approach

is that the EVA Program receives elements from different inputs rather than the same input data.

  - Each output is equal to the multiplication of its input index with its input.

- **Parallelized:** Multiple input data are parallelized instead of vectorized and each input data is passed to the EVA Program. Therefore, the EVA Program needs to be written accordingly since the whole input will be processed in the program. In the EVA Program, whole input data can be converted into *numpy* array with the data type of *eva.Expr* and computations in the program can be done over that array. In the end, the *numpy* array must be converted back to the dictionary compatible with EVA Program's Output. Even though computations of the program are done with *numpy*, there are still computation limitations such as broadcast operations between *numpy* arrays with *eva.Expr* data types, which result in exception/error.

  - Each output is equal to multiplication of $[1, 2, \ldots, n-1, n]$ with its input.

4.3.2.1 Vectorized Implementation

Input matrix of size $n \times n$ is converted into vector whose size is power of 2 and bigger than or equal to $(n+2)^2$ where plus 2 comes from the zero paddings. Then, **Reduce Ones EVA Program** is deployed to convert the input matrix into the reduced input matrix. Then, from the input matrix, the client itself creates the adjacency matrix and the client itself creates two more vectors from the adjacency matrix: distances matrix and extended matrix. Those 3 vectors' size is the power of 2 and bigger than or equal to n^4. There are n^4 elements in the adjacency matrix since there are n^2 nodes in the 2D input matrix. The distances and the extended matrices are initialized from the adjacency matrix at the beginning.

With the adjacency, the distances, and the extended matrices as input vectors, **Compute Distances EVA Program** is run exactly $\lceil n^2/2 \rceil \cdot n^2$ times. $\lceil n^2/2 \rceil$ represents the length of the longest possible path in the input matrix and n^2 is for calculating

distances for each element/node. With every iteration distances and extended matrices are updated. Note that the distance matrix is updated within the EVA Program at each iteration. Then, the client adjusts the extended matrix for the next iteration from the distances matrix and the next element/node.

After the client has obtained the (optional) reduced input and distance matrices from the server, the islands can be found on the client side. $n \times n$ is the size of the input matrix. In the vectorized implementation:

- **Reduce Ones EVA Program** is called just once with 1 vector with size of $2^{\lceil 2 \cdot \log(n+2) \rceil}$.
- **Compute Distances EVA Program** is called $\lceil n^2/2 \rceil \cdot n^2$ times with 3 vectors with size of $2^{\lceil 4 \cdot \log(n) \rceil}$.
- Yet, the problem is solved for only one instance.

4.3.2.2 Parallelized Implementation

2^m input matrices whose sizes are $n \times n$ is converted into n^2 vectors with size of 2^m. Then, **Reduce Ones EVA Program** is deployed to convert all input matrices into reduced input matrices. In the program, *numpy* matrices are constructed from the inputs where each value is the type of *eva.Expr* and calculations are pretty straightforward as they are done with *numpy*[1]. After the calculations are done, the *numpy* arrays are converted back to the appropriate output dictionary.

Then, **Compute Distances EVA Program** is deployed to calculate the distances matrix. This program converts the input vectors into *numpy* input matrices. From those matrices, adjacency matrices[2] are obtained and used further to get distances matrices.

[1] Data type of *eva.Expr* has built-in operations such as multiplication with scalar or another *eva.Expr*, addition with a scalar or another *eva.Expr*, negation, shift, etc. Those operations enable the use of *eva.Expr* data type with *numpy* and makes the implementations much easier. Even though there are built-in defined operations, some of the operations with the *numpy* such as broadcast operation can cause exceptions/errors.

[2] Direct conversion of input matrix into adjacency matrix causes an exception regarding the use of scalar 0 values in it as SEAL library prohibits multiplication of encrypted value with scalar 0 which will result in transparency of ciphertext. In order to solve this, a zero input vector is used where each element is the encrypted

After the distance matrices are calculated, they are converted back to the appropriate output dictionary.

After the client has obtained the (optional) reduced input matrices and distances matrices, the number of islands for multiple inputs can be found on the client side. $n \times n$ is the size of input matrices and 2^m is the vector sizes and the number of different input graphs/matrices. In the parallelized implementation:

- **Reduce Ones EVA Program** is called just once with n^2 vectors with size of 2^m.
- **Compute Distances EVA Program** is called just once with n^2 vectors with size of 2^m.
- The problem is solved for 2^m instances.

4.3.2.3 Parallelized Implementation without Loops

Reduce Ones step is the same as in the Parallelized Implementation. But, the Compute Distances part is different. Even though **Compute Distances EVA Program** of Parallelized Implementation is straightforward and easy with the usage of *numpy*, it has a crucial problem: Multiplicative depth grows exponentially with the increase of n, and the implementation cannot compute matrices with sizes bigger than 2×2 (e.g. 3×3, 4×4 etc.) due to insufficient bit modulus. In order to tackle this problem, loops in the **Compute Distances EVA Program** are moved to the client side. So, the client will call this program in the server $\lceil n^2/2 \rceil \cdot n^2$ times. This approach is pretty similar to the approach in the Vectorized Implementation where $\lceil n^2/2 \rceil$ represents the length of the longest possible path in the input matrices and n^2 is for calculating the distance for each element. Also, input vectors differ from the previous implementation as the adjacency and distances matrices are created on the client side and passed as input to

version of zero values. This might cause security problems as it resembles the known plaintext attacks. If that is the case, conversion into adjacency matrix can be done on the client side and sent to the server afterward as done in the Vectorized Implementation in order to solve the security problem.

the **Compute Distances EVA Program**. The distance matrices are updated at each iteration/program call depending on each row value.

After the client has obtained the (optional) reduced input matrices and the distance matrices, the number of islands for multiple inputs can be found on the client side. $n \times n$ is the size of input matrices and 2^m is the vector sizes and the number of different input graphs/matrices. In the parallelized implementation:

- **Reduce Ones EVA Program** is called just once with n^2 vectors with size of 2^m.
- **Compute Distances EVA Program** is called $\lceil n^2/2 \rceil \cdot n^2$ times with $2 \cdot n^4$ vectors with size of 2^m.
- The problem is solved for 2^m instances.

4.3.3 Implementation Comparisons

In this section, the similarities or differences of the three implementations that are described previously are discussed. In Table 4.1, solution steps and where they are executed in the implementations are shown. Reduce Ones and calculation of Distances Matrix methods are done by the server in all of the implementations. However, the conversion of the input matrix into the adjacency matrix is done only by the server in Parallelized Implementation. This step is done together with the calculation of the Distances Matrix. With the additional EVA Program (such as *Obtain Adjacency Matrix EVA Program*), the adjacency step can be passed to the server in the Parallelized Implementation without Loops. However, such a thing may not be possible for the Vectorized Implementation. Finding the islands step cannot be done in the EVA Program as it requires checking the connectivity of 1-valued cells.

Table 4.1: Methods and information about whether they are executed in the client or the server.

Method\Implementation	**Vectorized**	**Parallelized**	**P. w/out Loops**
Reduce Ones	Server	Server	Server
Adjacency Matrix	Client	Server	Client
Distances Matrix	Server	Server	Server
Finding Islands	Client	Client	Client

The number of times that the implementations call each EVA Program is shown in Table 4.2. Reduce Ones EVA Program is called only once in every implementation. However, Compute Distances EVA Program is called $\lceil n^2/2 \rceil \cdot n^2$ times in both Vectorized Implementation and Parallelized Implementation without Loops where n^2 is the size of the input matrix/matrices.

Table 4.2: The number of times that EVA Programs are called in the implementations where n^2 is the size of input matrix/matrices.

EVA Program\Implementation	**Vectorized**	**Parallelized**	**P. w/out Loops**
Reduce Ones	1	1	1
Compute Distances	$\lceil n^2/2 \rceil \cdot n^2$	1	$\lceil n^2/2 \rceil \cdot n^2$

Input sizes, vector sizes, and number of inputs of implementations' EVA Programs are shown in Table 4.3. n^2 is the size of input matrix/matrices and 2^m is the number of different input matrices. Vectorized Implementation works on only one input matrix whereas Parallelized Implementations can work on multiple input matrices. In Vectorized Implementation, input sizes to the Reduce Ones and Compute Distances EVA Programs are 1 and 3 respectively while this is not the case for Parallelized Implementations since the whole input matrix or distances matrix are used as input to EVA Programs.

Table 4.3: Input sizes, vector sizes, and the number of inputs that are used in EVA Programs of the implementations where n^2 is the size of input matrix/matrices and 2^m is the number of different input matrices in the Parallelized Implementations.

Implementation	**Vectorized**		**Parallelized**		**P. w/out Loops**	
EVA Program	**Reduce**	**Compute**	**Reduce**	**Compute**	**Reduce**	**Compute**
Input Size	1	3	n^2	n^2	n^2	$2 \cdot n^4$
Vector Size	$2^{\lceil 2 \cdot \log(n+2) \rceil}$	$2^{\lceil 4 \cdot \log(n) \rceil}$	2^m	2^m	2^m	2^m
Number of Inputs	1		2^m		2^m	

4.4 Results

FHE comes with a computational overhead, meaning computations done over encrypted data will be much slower than the computations on plaintext data. In this section, averaged simulation results of the implementations are shown in terms of compile, key generation, encryption, execution, decryption, reference execution times, and MSE scores for different input sizes.

Means and standard deviations of timings and scores of the EVA programs by the input matrix and vector sizes are shown in the next subsections. From the figures, it can be said that increasing the vector size slightly helps to decrease MSE scores. As expected, increasing the input matrix size increases all the measured timings. In each Compute Distances EVA Program, 30 bits are used for output ranges and input scales. In the case of high MSE scores, tuning those values might help to reduce the scale of errors on ciphertexts and so do MSE scores.

As previously mentioned, the EVA programs in our implementation might be called more than once depending on the approach. Therefore, the total running times of the EVA programs have also been analyzed as the EVA Program might be called multiple times depending on the square of the n where n is the number of nodes. Therefore, runtimes of the EVA Programs have an exponential relation with respect to input size as expected from the calling times. Also, note that the implementation

can be finished in several hours as the input matrix size increases. Apart from the total running times, amortized running times of the EVA programs have also been analyzed since parallelized implementations can take more than one input.

4.4.1 Plots of Reduce Ones EVA Program

Means and standard deviations of timings and scores of Reduce Ones EVA Programs by the input matrix and vector sizes are shown in Figures 4.9 and 4.10. From the figures, it can be said that increasing the vector size helps to decrease compile, key generation times, and MSE scores while increasing encryption and reference execution times. As expected, increasing the input matrix size increases all the measured timings. However, it greatly affects the encryption, execution, and decryption timings for the Parallelized Implementations due to the increasing number and size of input vectors to the programs. In each Compute Distances EVA Program, 30 bits are used for output ranges and input scales. In the case of obtaining high MSE scores, increasing these values helps to reduce the scale of errors on ciphertexts, and so do MSE scores.

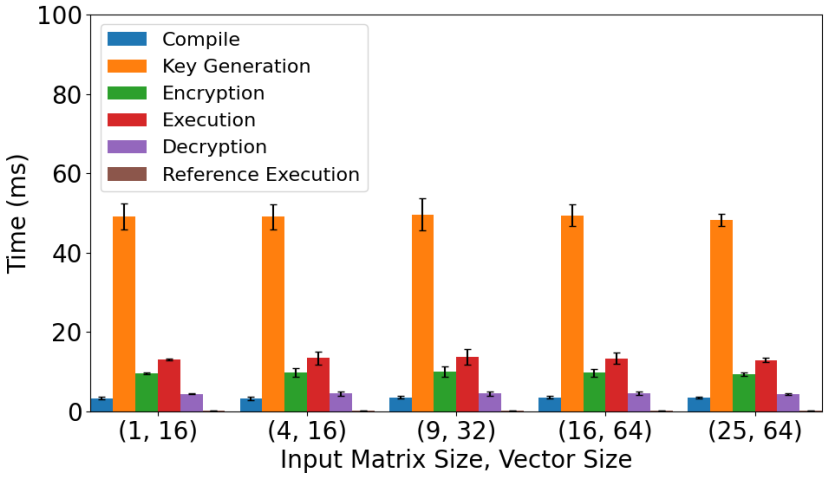
Note that the y-axes of timing plots are the same in the figures, which means each implementation's Reduce Ones EVA Program runs in the same time scale even though there are up to 10 times timing differences between the implementation programs.

4.4.2 Plots of Compute Distances EVA Programs

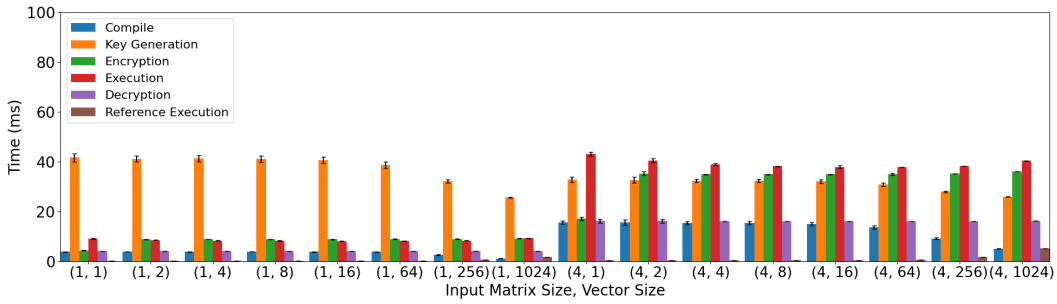
Means and standard deviations of timings and scores of Compute Distances EVA Programs by the input matrix and vector sizes are shown in Figures 4.11 and 4.12. From the figures, it can be said that increasing the vector size slightly helps to decrease MSE scores. As expected, increasing the input matrix size increases all the measured timings. However, it greatly affects the encryption, execution, and decryption timings for the Parallelized Implementations due to the increasing number and size of input

vectors to the programs. From the plots, it can be seen that Parallelized Implementation without Loops has an overhead related to the input vectors to the programs as encryption timings are the most time-consuming part of the program. Other than that, Parallelized Implementation has an overhead related to the execution which is expected because the program is used to calculate the distances matrix at one call. The other programs are called multiple times to calculate the distance matrices. Also, MSE scores of Parallelized Implementations are much higher than the Vectorized Implementation due to the high number of multiplications in the parallelized programs. In each Compute Distances EVA Program, 30 bits are used for output ranges and input scales. In the case of high MSE scores, increasing those values helps to reduce the scale of errors on ciphertexts and so do MSE scores.

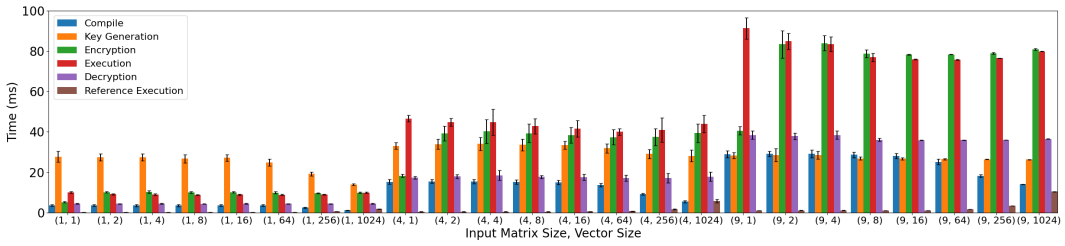
Note that the y-axes of timing plots are not the same in the figures, which means the Vectorized program runs faster than all others, and the Parallelized program is running faster than the program without loops. Also, it is important to keep in mind that the Compute Distances EVA Program of Parallel Implementation is called just once to calculate distances whereas other programs are called multiple times. Comparisons of implementations in terms of total and amortized runtimes are also plotted in the next subsections. Another reminder is that Parallelized Implementations work on multiple inputs while Vectorized Implementation runs for only one input.



(a) Vectorized Implementation

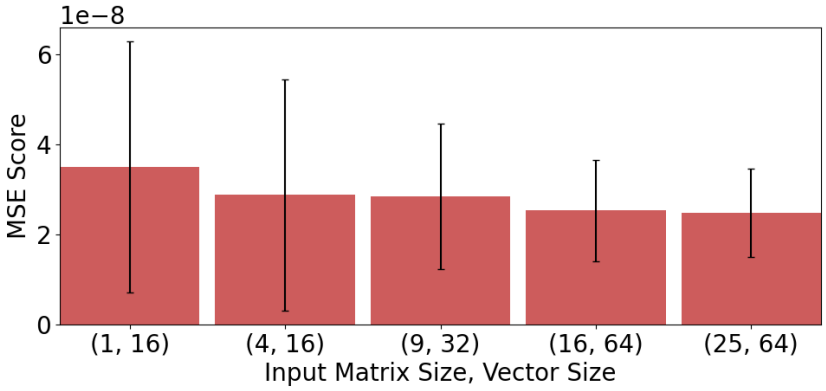


(b) Parallelized Implementation

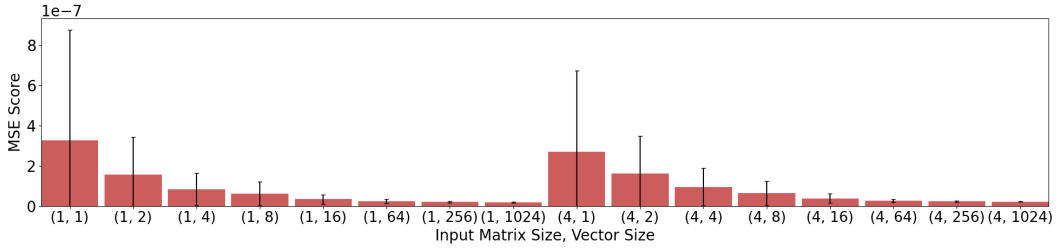


(c) Parallelized Implementation without Loops

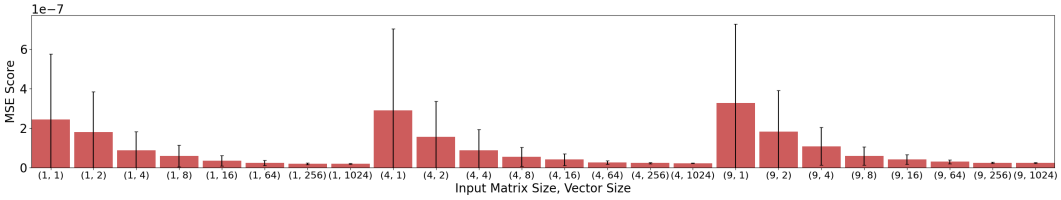
Figure 4.9: Timings of all Reduce Ones EVA program implementations by input matrix and vector sizes.



(a) Vectorized Implementation

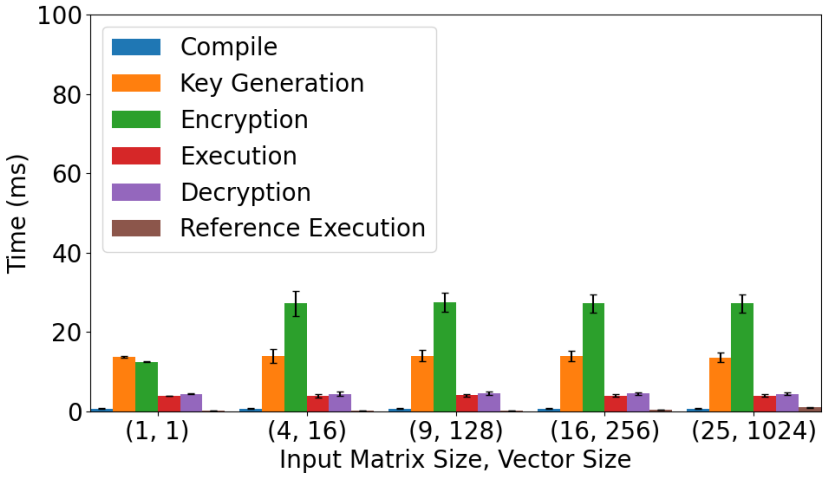


(b) Parallelized Implementation

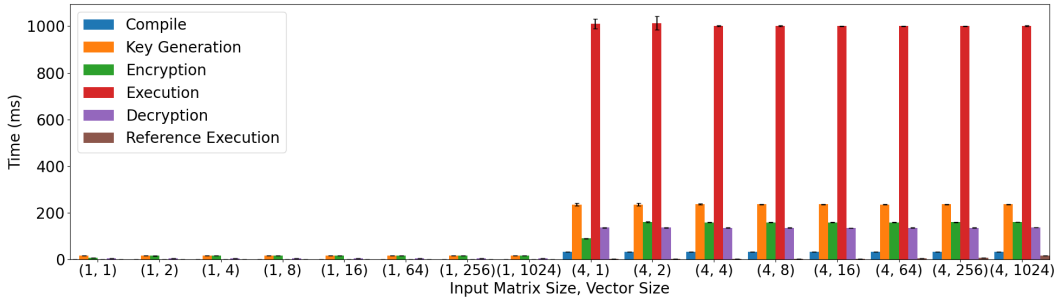


(c) Parallelized Implementation without Loops

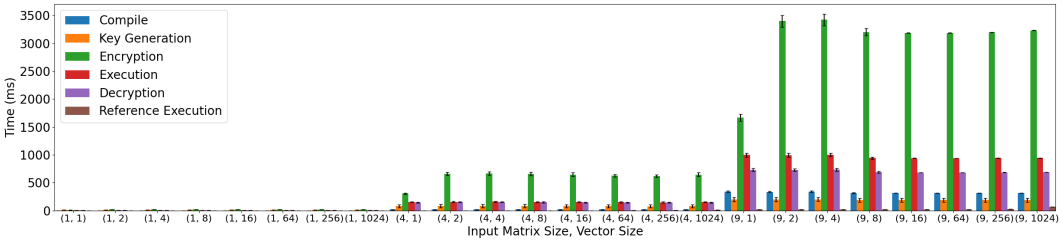
Figure 4.10: MSE scores of all Reduce Ones EVA Programs implementations by input matrix and vector sizes.



(a) Vectorized Implementation

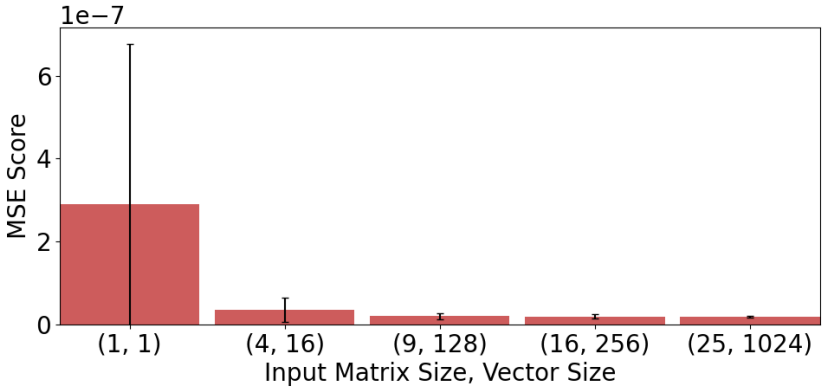


(b) Parallelized Implementation

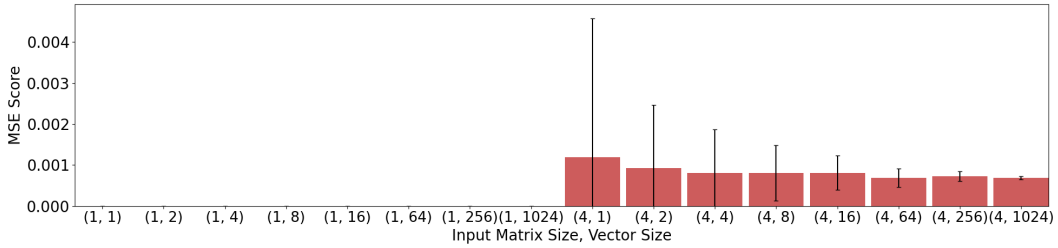


(c) Parallelized Implementation without Loops

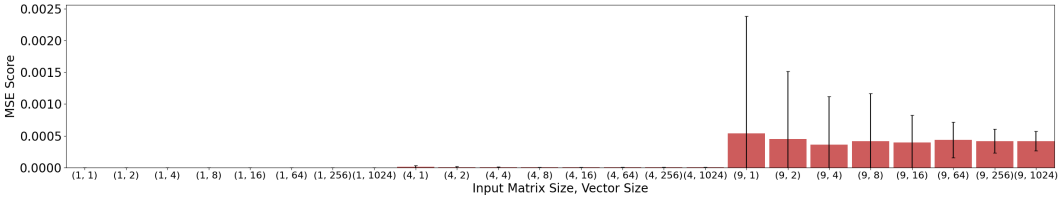
Figure 4.11: Timings of all Compute Distances EVA Programs implementations by input matrix and vector sizes.



(a) Vectorized Implementation



(b) Parallelized Implementation



(c) Parallelized Implementation without Loops

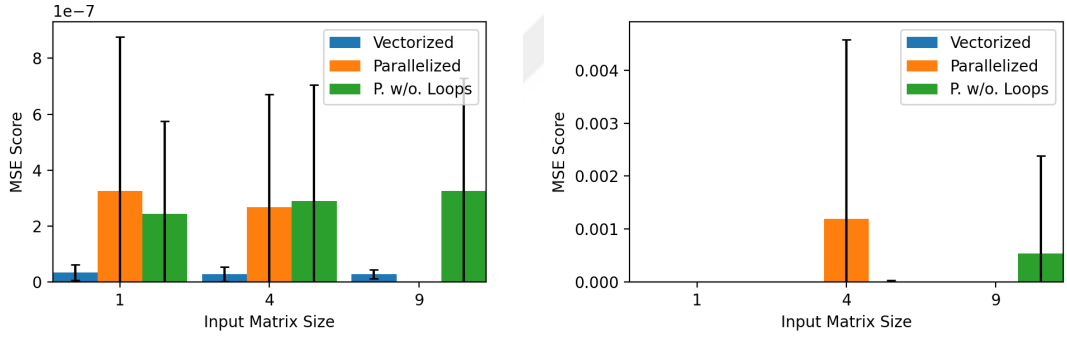
Figure 4.12: MSE scores of all Compute Distances EVA Programs implementations by input matrix and vector sizes.

4.4.3 Comparison of EVA Programs with Only One Input

Timings and MSE scores of EVA Programs are plotted with respect to the implementations when there is only one input in Figure 4.13, Figure 4.14, and Figure 4.15. The input matrix size and the number of inputs are the same for all of the three implementations. So, only one input matrix is used and input matrix sizes are 1×1,

2×2, and 3×3. Unfortunately, Parallelized Implementation cannot work with matrix sizes above 2×2 due to the multiplicative depth limitations of using leveled-FHE in SEAL. Also, the vector size is 1 for Parallelized Implementations due to using only one input.

Vectorized Implementation is faster than Parallelized Implementation in almost every timings and MSE scores except for key generation times in Reduce Ones EVA Programs. In terms of Compute Distances EVA Programs, Vectorized Implementation is much faster than the other implementations. Parallelized Implementations run with similar timings in terms of Reduce Ones EVA Programs. Yet, Parallelized Implementation is only faster in terms of encryption and decryption time, which can be explained with a reduced number of input vectors to Compute Distance EVA Program. Parallelized Implementation's MSE scores are much greater than the rest due to having greater multiplicative depth, although this cannot be seen clearly from the plots.



(a) Reduce Ones

(b) Compute Distances

Figure 4.13: MSE score comparisons of EVA Programs by their implementations.

As mentioned in Subsection 4.3.3, EVA programs can be called more than once depending on the implementation. Therefore, the total running times of the EVA programs should also be analyzed. Reduce Ones EVA Programs are called only once for every implementation. Still, only Compute Distances EVA Program of Parallelized Implementation is run once, whereas the other implementations are called $\lceil n^2/2 \rceil . n^2$ times.

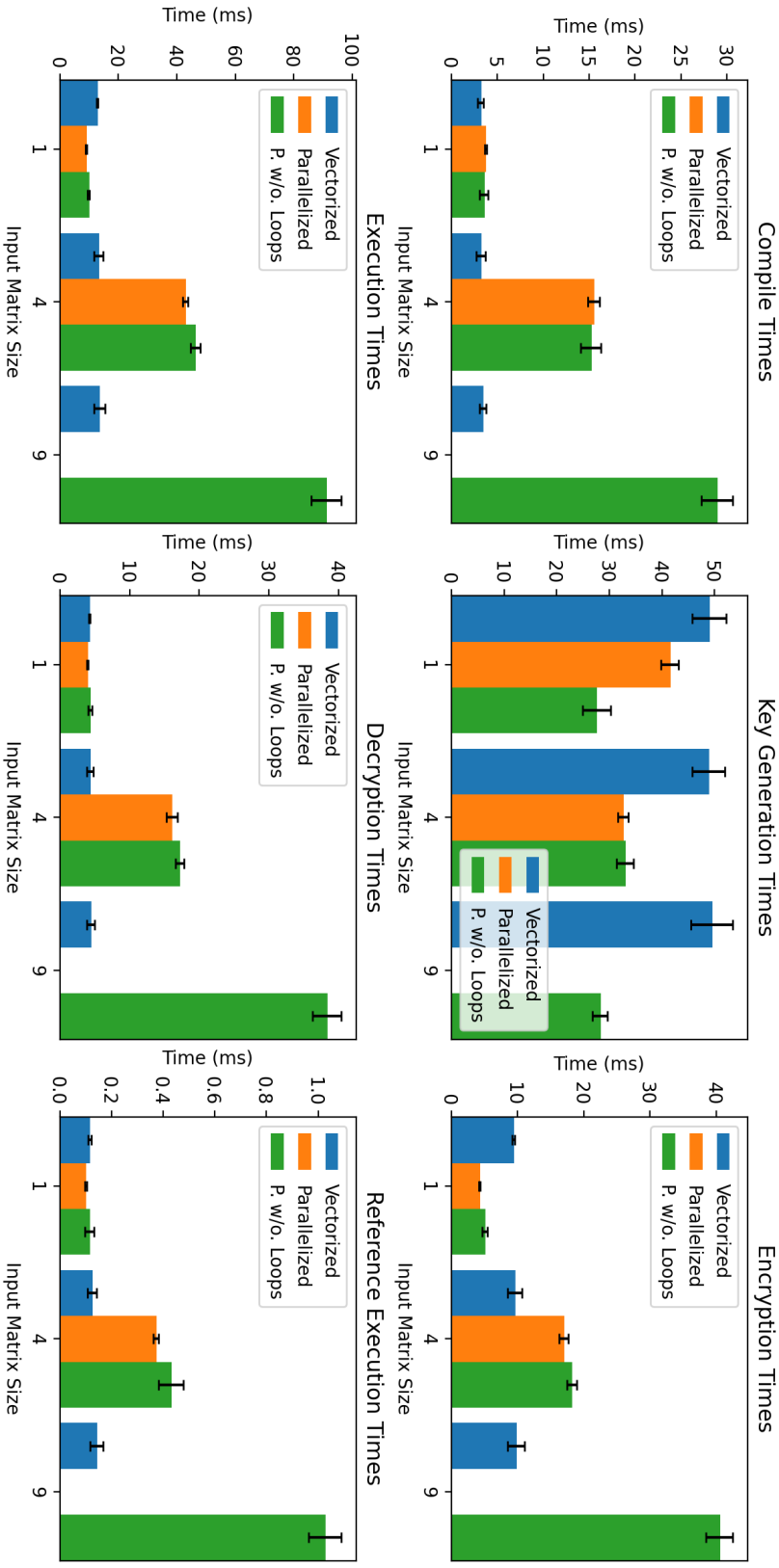


Figure 4.14: Timings comparisons of all Reduce Ones EVA Program implementations when there is only one input.

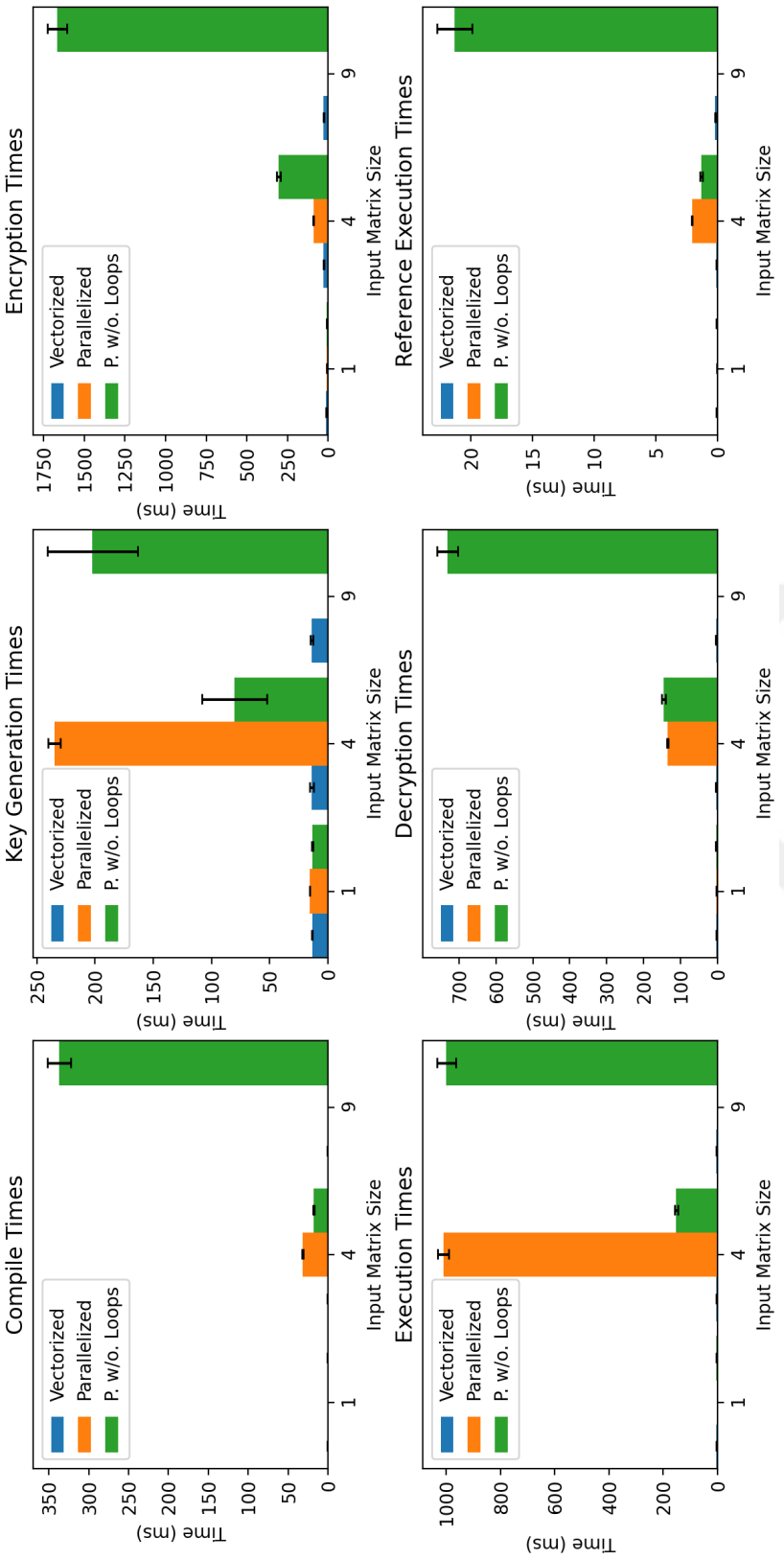
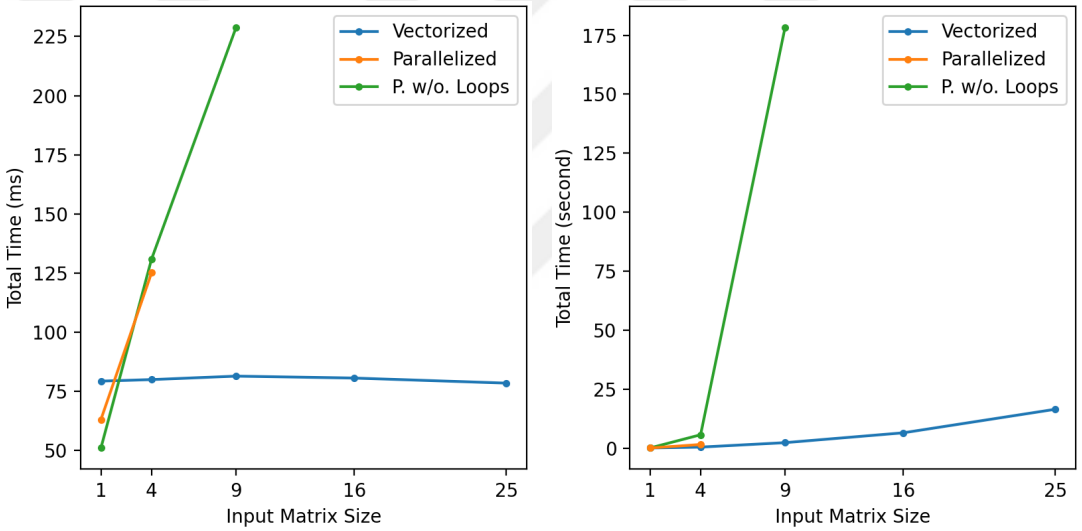


Figure 4.15: Timings comparisons of all Compute Distances EVA Program implementations when there is only one input.

In Figure 4.16, the total running times of each implementation with only one input are shown. By looking at the figure, Vectorized Implementation seems faster than Parallelized Implementations if each implementation is run with only one input (i.e. vector size is 1 for Parallelized Implementations). Even though Reduce Ones EVA Programs are called only once, Parallelized Implementations seem to run in $O(n)$ instead of $O(1)$ which is similar to Vectorized Implementation runtime behavior. Compute Distances EVA Programs (except Parallelized Implementation) seem to have exponential runtime with respect to input size. This or the opposite cannot be said about Parallelized Implementation's runtime due to lack of data. Also note that Compute Distances EVA Programs can be finished in several hours as the input matrix size is increased and Parallelized Implementation without Loops seems to be much slower than Vectorized Implementation.



(a) Reduce Ones

(b) Compute Distances

Figure 4.16: Total runtimes of all EVA Program implementations when there is only one input.

4.4.4 Amortized Runtime Comparison of EVA Programs

In the previous subsections, every implementation's EVA Programs are compared

with each other when they work on only one input. However, unlike Vectorized Implementation, Parallelized Implementations can handle more than one input. The amortized comparisons of each EVA Program implementation are presented in Figure 4.17 and Figure 4.18. For the amortized analysis, total running times are divided by 1024 (the number of inputs, which is the maximum vector size for Parallelized Implementations in the simulation) and 1 for Vectorized Implementation. From the amortized runtimes of EVA Programs in the figures, it can be said that increasing the vector size decreases the amortized runtimes for Parallelized Implementations.

If the total runtimes of each implementation are considered as in Figure 4.19, Vectorized Implementation is slower than the Parallelized Implementations. However, further experiments and simulations with larger input matrix sizes are required to decide this for larger input matrix sizes. Furthermore, Vectorized Implementation can be further improved to work on multiple input data.

4.5 Discussion

In the implementations, the adjacency representation of the 2D input matrix is converted into an input vector, which might have an upper size limit coming from either the compiler or its library. This can be a problem when dealing with bigger-sized inputs. Another limitation comes from multiplicative depth which causes insufficient bit modulus problems as the number of multiplications grows. This limitation can be solved by optimizing the algorithms or dividing the whole computation into smaller computations as done in this work. Yet, that solution comes with a critical drawback: computation overhead. Considering the exponential relation between the running time and the input size, it could take more than days to solve the problem for bigger inputs, which refutes the goal of this work.

In order to reduce error accumulation throughout communications between the client and the server, values in the matrices are rounded to the closest integers before encryption. For example, a cell value in the distances matrix could be approximated to 0.99 instead of 1, and if it is calculated with a similar error rate $\lceil N/2 \rceil \cdot N$ times, then it will be equal to 0.63 for $N = 9$ and 0.27 for $n = 16$.

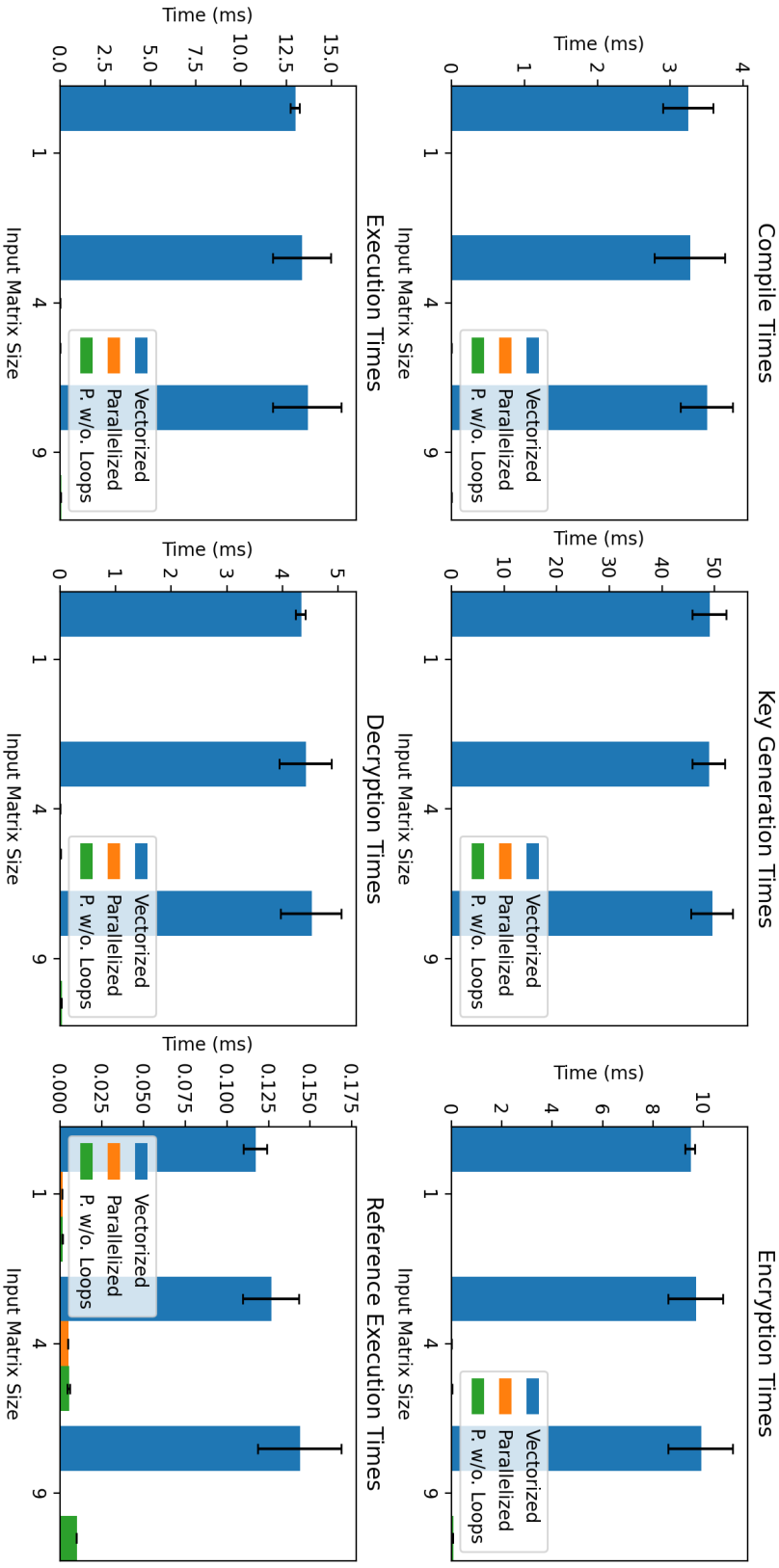


Figure 4.17: Amortized timing comparisons of all Reduce Ones EVA Program implementations.

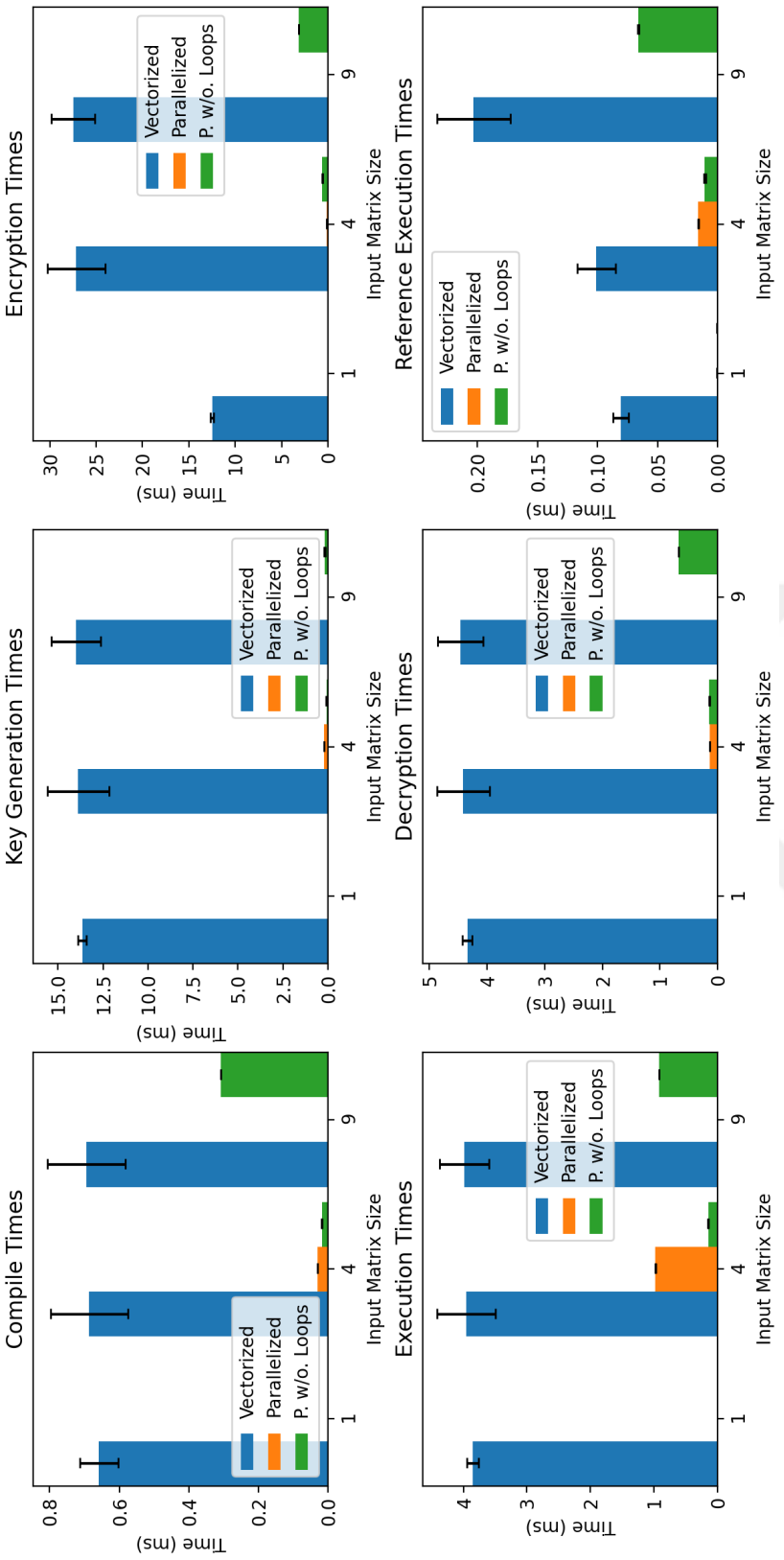
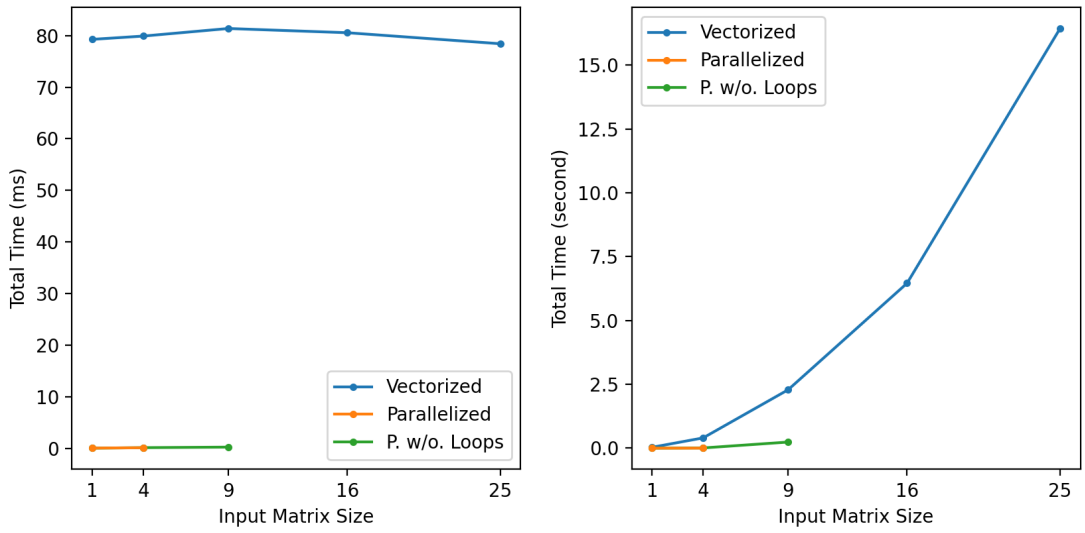


Figure 4.18: Amortized timings comparisons of all Compute Distances EVA Program implementations.



(a) Reduce Ones

(b) Compute Distances

Figure 4.19: Amortized total runtimes of all EVA Programs implementations.

There could be a few optimizations in the implementations. First of all, the client is called $\lceil N/2 \rceil \cdot N$ times. This increases the total running time greatly. The client could reduce the number of server calls by checking the input matrix and skipping the calculation of 0-valued cells as their distance information won't be helpful at all. Secondly, the client can check consecutive distance matrices and decide to stop earlier. For example, if every 1-valued cell can reach every other 1-valued cell, the client can stop the iteration earlier as the client has enough information to solve the problem correctly. Thirdly, another optimization can be reducing the input vector sizes so that the server will do fewer computations. The client could ignore 0-valued cells when creating an adjacency matrix representation of the input 2D $0-1$ matrix. In addition, Vectorized Implementation can be altered to work on multiple input matrices. It is believed that these optimizations can reduce the amortized running times of the implementations.

Further experiments on the problem can be conducted using different HE tools that support FHE schemes with the bootstrapping technique.

Vectorized Implementation is the fastest if the solution requires working on only one input. Otherwise, Parallelized Implementations seem to run faster in terms of

amortized running time analysis (where Parallelized Implementations use 1024 inputs whereas Vectorized Implementation uses 1 input). However, Vectorized Implementation can be improved to work on multiple input data, which may change this situation. Also, Vectorized Implementation does not have multiplication limitations as in Parallelized Implementation because it has simpler EVA Programs compared to other EVA program implementations.

Finding the connected islands problem can be solved using FHE. However, it is important to keep in mind that running times of implementations can be more than hours depending on the input matrix size due to computation limitations and overhead that come with FHE.

CHAPTER 5

CONCLUSION

This thesis proposes offloading delay-tolerant tasks of NWDAF to third-party (public) cloud service providers while preserving data privacy using homomorphic encryption schemes. This would allow telecommunication operators to be more agile, focus on their core capabilities, and reduce capital and operational expenditures. In order to support the proposal, two of the NWDAF use cases are selected and HE-friendly methodologies are provided for the solutions.

Even though NWDAF was first introduced in late 2017, the literature still lacks network data that can be used for NWDAF use cases. In light of this shortcoming, a synthetic NWDAF dataset containing abnormal behavior information for a selected group of UEs is used, and various deep-learning models are trained and tested with plaintext and ciphertext data. It is shown that inferencing with ciphertext data is as successful as inferencing with normal plaintext data. Its accuracy and precision are similar to or slightly better than the results of previous works with the same data. In addition, it is evaluated that both runtime and amortized running time effects of encryption and DL parameters through extensive experiments and shown that preserving privacy increases both averaged amortized runtimes and running times by 5 and 6 times, respectively. In the future, this work can be extended to use ciphertext data for both model training and inferencing in the cloud and evaluate communication costs, security threats, and computational overhead. These extensions will further affirm our belief that both model training and analytical functionalities of NWDAF instances can be offloaded to third-party cloud providers while preserving the privacy of network and user data.

This thesis also presents a HE solution to the problem of finding the islands in the encrypted graph which can be extended to mobility analytics of UEs. HE-friendly methodology for solving the problem is explained in detail, and then, details of the FHE solution implemented with EVA and SEAL are explained. Different implementations are compared with each other in terms of compile, key generation, encryption, execution, decryption and reference execution times, total running times, and approximation rates by MSE scores. There are a few optimizations that can improve the implementations which might result in the running time reduction. Optimizations and improvement of the Vectorized Implementation where it can process more than one input data could be the extension of this part. Also, further experiments should be done with larger input matrices and vector sizes.

REFERENCES

[1] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5G: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE JSAC*, vol. 35, no. 6, pp. 1201–1221, 2017.

[2] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[3] I. F. Akyildiz, P. Wang, and S.-C. Lin, "Softair: A software defined networking architecture for 5G wireless systems," *Computer Networks*, vol. 85, pp. 1–18, 2015.

[4] H. Wang, S. Chen, H. Xu, M. Ai, and Y. Shi, "Softnet: A software defined decentralized mobile network architecture toward 5G," *IEEE Network*, vol. 29, no. 2, pp. 16–22, 2015.

[5] F. Mademann, "The 5G system architecture," *Journal of ICT Standardization*, pp. 77–86, 2018.

[6] T. Chen, M. Matinmikko, X. Chen, X. Zhou, and P. Ahokangas, "Software defined mobile networks: concept, survey, and research directions," *IEEE Communications Magazine*, vol. 53, no. 11, pp. 126–133, 2015.

[7] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[8] L. Ma, X. Wen, L. Wang, Z. Lu, and R. Knopp, "An SDN/NFV based framework for management and deployment of service based 5G core network," *China Communications*, vol. 15, no. 10, pp. 86–98, 2018.

[9] 3GPP, "5G System; Network Data Analytics Services; Stage 3," Technical Spec-

ification (TS) 29.520, 3rd Generation Partnership Project (3GPP), May 2019. Version 15.3.0.

[10] 3GPP, "Architecture enhancements for 5G System (5GS) to support network data analytics services," Technical Specification (TS) 23.288, 3rd Generation Partnership Project (3GPP), July 2020. Version 16.4.0.

[11] 3GPP, "Architecture enhancements for 5G System (5GS) to support network data analytics services," Technical Specification (TS) 23.288, 3rd Generation Partnership Project (3GPP), May 2022. Version 17.4.0.

[12] A. J. Duncan, S. Creese, and M. Goldsmith, "Insider attacks in cloud computing," in *Proc. of the IEEE TRUSTCOM*, pp. 857–862, 2012.

[13] L. Cheng, F. Liu, and D. Yao, "Enterprise data breach: causes, challenges, prevention, and future directions," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 5, p. e1211, 2017.

[14] N. Salhab, R. Rahim, R. Langar, and R. Boutaba, "Offloading network data analytics function to the cloud with minimum cost and maximum utilization," in *Proc. of the ICC 2020*, pp. 1–6, 2020.

[15] R. Li, Z. Zhao, X. Zhou, G. Ding, Y. Chen, Z. Wang, and H. Zhang, "Intelligent 5G: When cellular networks meet artificial intelligence," *IEEE Wireless Communications*, vol. 24, no. 5, pp. 175–183, 2017.

[16] X. You, C. Zhang, X. Tan, S. Jin, and H. Wu, "AI for 5G: research directions and paradigms," *Science China Information Sciences*, vol. 62, pp. 1–13, 2018.

[17] R. Shafin, L. Liu, V. Chandrasekhar, H. Chen, J. Reed, and J. C. Zhang, "Artificial intelligence-enabled cellular networks: A critical path to beyond-5G and 6G," *IEEE Wireless Communications*, vol. 27, no. 2, pp. 212–217, 2020.

[18] A. Chouman, D. M. Manias, and A. Shami, "Towards supporting intelligence in 5G/6G core networks: NWDAF implementation and initial analysis," in *2022 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 324–329, 2022.

[19] S. Sevgican, M. Turan, K. Gökarslan, H. B. Yilmaz, and T. Tugcu, "Intelligent network data analytics function in 5G cellular networks using machine learning," *Journal of Communications and Networks*, vol. 22, no. 3, pp. 269–280, 2020.

[20] Y. Yuan, C. Gehrmann, J. Sternby, and L. Barriga, "Insight of anomaly detection with NWDAF in 5G," *2022 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pp. 1–6, 2022.

[21] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.

[22] M. Isaksson and K. Norrman, "Secure federated learning in 5G mobile networks," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1–6, 2020.

[23] Y. Jeon, H. Jeong, S. Seo, T. Kim, H. Ko, and S. Pack, "A distributed NWDAF architecture for federated learning in 5G," in *2022 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–2, 2022.

[24] L. Yu, L. Xin, and M. Guo, "A safe architecture of 5G network intelligence based on federated learning and NWDAF," *Proceedings of the 4th International Conference on Advanced Information Science and System*, 2022.

[25] C. Zhou and N. Ansari, "Securing federated learning enabled NWDAF architecture with partial homomorphic encryption," *IEEE Networking Letters*, pp. 1–1, 2023.

[26] R. L. Rivest, L. Adleman, M. L. Dertouzos, *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[27] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

[28] C. Gentry, "Computing arbitrary functions of encrypted data," *Commun. ACM*, vol. 53, p. 97–105, mar 2010.

[29] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive, Paper 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[30] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[31] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption library," August 2016. https://tfhe.github.io/tfhe/.

[32] 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), pp. 409–437, Springer International Publishing, 2017.

[33] A. Viand, P. Jattke, and A. Hithnawi, "SoK: Fully homomorphic encryption compilers," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1092–1108, 2021.

[34] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, jul 2018.

[35] P. Martins, L. Sousa, and A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective," *ACM Comput. Surv.*, vol. 50, dec 2017.

[36] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[37] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Commun. ACM*, vol. 16, p. 372–378, jun 1973.

[38] A. Rosenfeld, "Connectivity in digital pictures," *J. ACM*, vol. 17, p. 146–160, jan 1970.

[39] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: A review of state-of-the-art algorithms," *Pattern Recognition*, vol. 70, pp. 25–43, 2017.

[40] A. Kunz, L. Nkenyereye, and J. Song, "5G evolution of cellular IoT for V2X," *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 1–6, 2018.

[41] A. Ghosh, A. Maeder, M. P. J. Baker, and D. Chandramouli, "5G evolution: A view on 5G cellular technology beyond 3GPP release 15," *IEEE Access*, vol. 7, pp. 127639–127651, 2019.

[42] K. Laine, "Simple encrypted arithmetic library 2.3. 1," *Microsoft Research https://www. microsoft. com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1. pdf*, 2017.

[43] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption." Cryptology ePrint Archive, Paper 2015/1192, 2015.

[44] J. Contreras, "Homomorphic encryption with SEAL." `https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/articles/homomorphic-encryption-seal`. Accessed: (26.08.2023).

[45] D. Dolev, C. Dwork, and M. Naor, "Nonmalleable cryptography," *SIAM Journal on Computing*, vol. 30, no. 2, pp. 391–437, 2000.

[46] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, 1978.

[47] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, pp. 469–472, 1984.

[48] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Application of Cryptographic Techniques*, 1999.

[49] C. Gentry, "A fully homomorphic encryption scheme," 2009.

[50] S. Halevi and V. Shoup, "Algorithms in helib," in *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*, pp. 554–571, Springer, 2014.

[51] "Microsoft SEAL (release 4.1)." `https://github.com/Microsoft/SEAL`, Jan. 2023. Microsoft Research, Redmond, WA.

[52] "PALISADE Lattice Cryptography Library (release 1.11.5)." `https://palisade-crypto.org/`, 2021.

[53] C. V. Mouchet, J.-P. Bossuat, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, "Lattigo: A multiparty homomorphic encryption library in go," in *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, no. CONF, pp. 64–70, 2020.

[54] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in *International Conference on the Theory and Application of Cryptographic Techniques*, 2015.

[55] E. Crockett, C. Peikert, and C. Sharp, "ALCHEMY: A language and compiler for homomorphic encryption made easy," *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[56] E. Chielle, O. Mazonka, N. G. Tsoutsos, and M. Maniatakos, "E3: A framework for compiling C++ programs with encrypted operands," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 1013, 2018.

[57] 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*, PLDI 2020, (New York, NY, USA), p. 546–561, Association for Computing Machinery, 2020.

[58] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. E. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: an optimizing compiler for fully-homomorphic neural-network inferencing," *Proceedings of the 40th ACM SIG-*

PLAN Conference on Programming Language Design and Implementation, 2019.

[59] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *Proc. of the 16th ACM CF*, CF '19, (NY, USA), p. 3–13, 2019.

[60] T. van Elsloo, G. Patrini, and H. Ivey-Law, "SEALion: a framework for neural network inference on encrypted data," *ArXiv*, vol. abs/1904.12840, 2019.

[61] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "nGraph-HE2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'19, (New York, NY, USA), p. 45–56, Association for Computing Machinery, 2019.

[62] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," ARES '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[63] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

[64] A. LeNail, "NN-SVG: Publication-ready neural network architecture schematics," *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019.

APPENDIX A

RUNNING TIMES OF ALL CONFIGURATIONS

Configuration names have the format of *N14-192-L5-p0452* or *N14-192-L5-p0452-C* in Figure A.1 and Figure A.2. Configuration *N14-192-L5-p0452* has a polynomial modulus degree of 14 with 192 bits of security level and DL model with 5 layers and 452 weights (parameters) whereas configuration *N14-192-L5-p0452-C* has the same properties with the addition of complex packing enabled. Available configuration parameters are in the below:

- Polynomial Modulus Degrees: 13, 14, or 15
- Security Levels: 128-bits, 196-bits, or 256-bits
- Complex Packing: Enabled or not
- DL Layer Counts: 5 or 7
- DL Parameter/Weight Counts: ~ 200, ~ 450, ~ 850, and ~ 1200

Note that some of the combinations are not possible due to restrictions related to HE-configurations in nGraph-HE and SEAL.

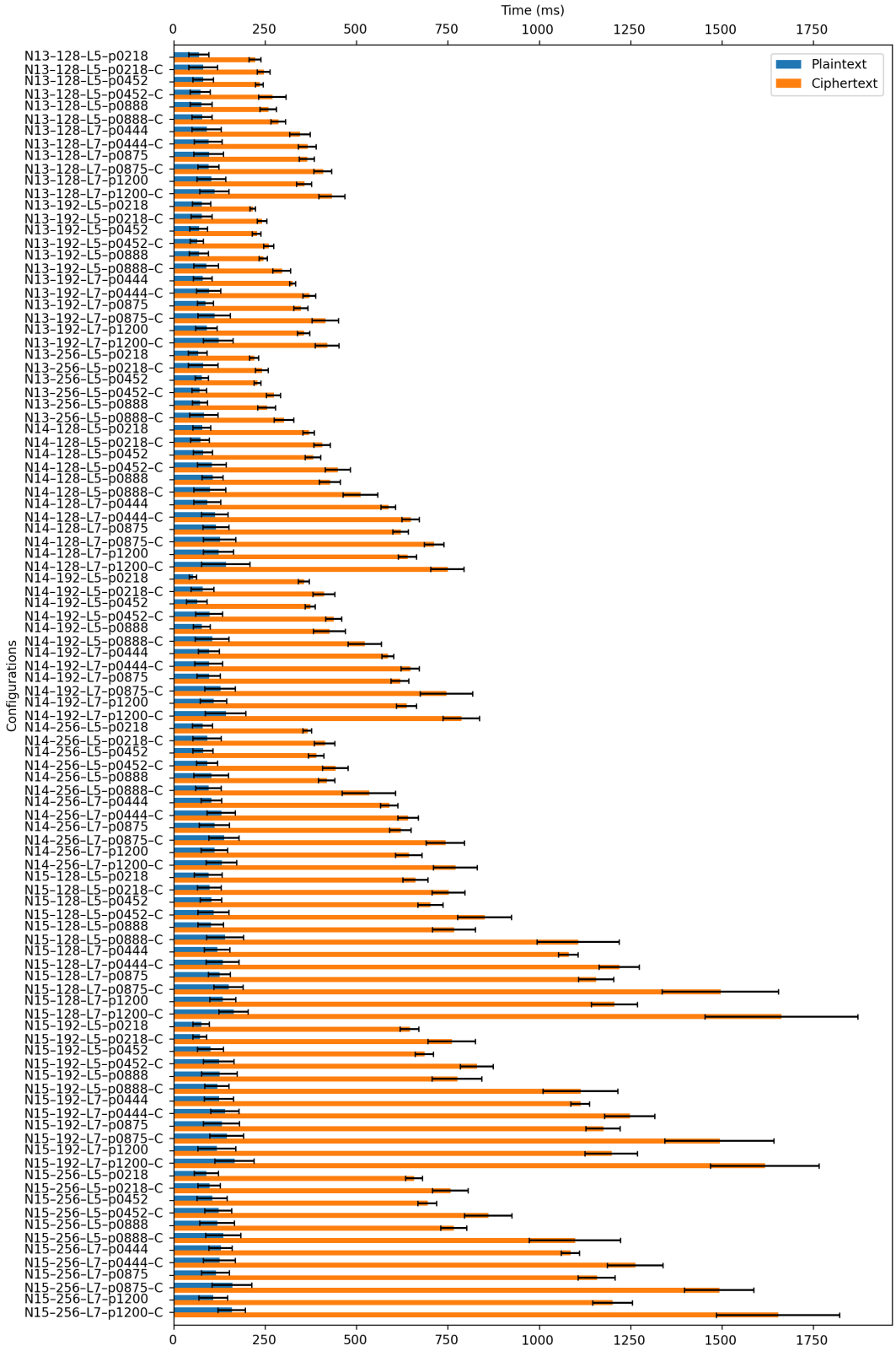


Figure A.1: Averaged running times of inferencing on both plaintext and ciphertext data for each configuration.

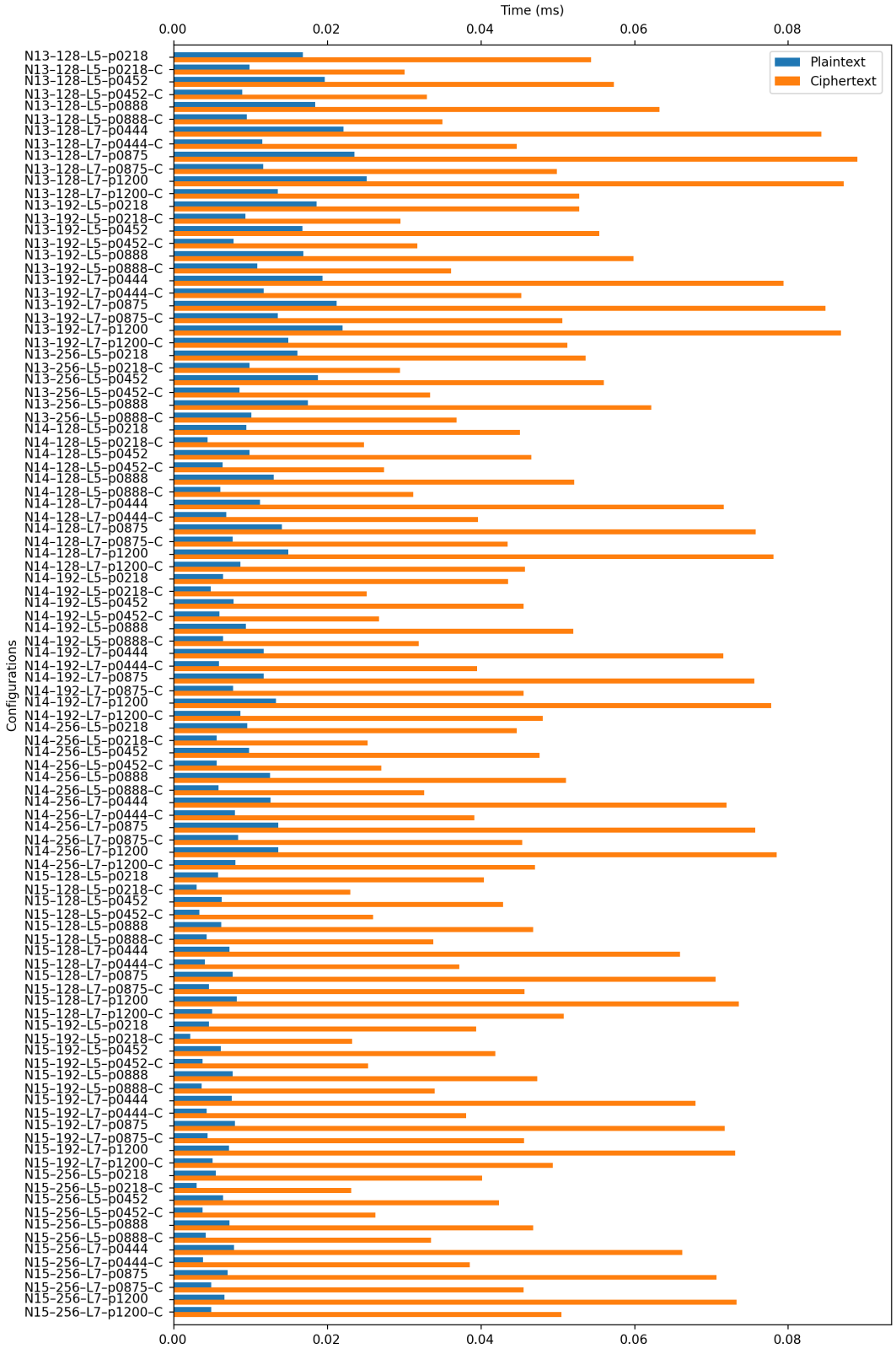


Figure A.2: Averaged amortized running times of inferencing on both plaintext and ciphertext data for each configuration.