COMPARISON OF METHODS OF PRIVACY-PRESERVING CLASSIFICATION BASED ON MACHINE LEARNING ALGORITHMS FOR INTRUSION DETECTION

by
CEREN YILDIRIM

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July 2022

ABSTRACT

COMPARISON OF METHODS OF PRIVACY-PRESERVING CLASSIFICATION BASED ON MACHINE LEARNING ALGORITHMS FOR INTRUSION DETECTION

CEREN YILDIRIM

COMPUTER SCIENCE AND ENGINEERING M.Sc. THESIS, JULY 2022

Thesis Supervisor: Prof. Erkay Savaş

Keywords: homomorphic encryption, network intrusion, machine learning classifiers, intrusion detection systems

As cyberattacks have become more prevalent and sophisticated, designing and developing intrusion detection systems (IDS) has turned out to be an increasingly challenging task. Machine learning-based intrusion detection systems offer a solution for fast, adaptable and accurate detection of intrusion incidents. However, depending on who is evaluating the classifier, this requires the IDS provider and the user to share the confidential network data and the evaluation model, putting both parties at risk of privacy violations. The homomorphic encryption technique proposes a solution to overcome such privacy issues, by allowing manipulation of encrypted data without requiring a decryption key. Using this technique, the parties may encrypt their private input (e.g., network data or evaluation model) before sharing it with an untrusted party for evaluation. As the homomorphic encryption technique may impose a prohibitively high computational overhead, the homomorphically executed classifiers must be designed to retain the detection abilities of the actual classifiers while minimizing the total computation overhead and multiplicative depth of the circuit that implements the classifiers. This thesis compares the performance of different machine learning-based classifiers for network intrusion detection and also evaluates different encryption scenarios. The overall detection accuracy, time performance, and security and privacy concerns of different implementations are assessed and discussed.

ÖZET

İZİNSİZ GİRİŞ TESPİTİ İÇİN MAKİNE ÖĞRENMESİ ALGORİTMALARINA DAYALI GİZLİLİĞİ KORUYAN SINIFLANDIRMA YÖNTEMLERİNİN KARŞILAŞTIRILMASI

CEREN YILDIRIM

BİLGİSAYAR BİLİMİ VE MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ,
TEMMUZ 2022

Tez Danışmanı: Prof. Dr. Erkay Savaş

Anahtar Kelimeler: homomorfik şifreleme, izinsiz ağa giriş, makine öğrenmesi temelli sınıflandırma, saldırı tespit sistemleri

Siber saldırıların yaygınlaşması ve daha karmaşık hale gelmesiyle, izinsiz giriş tespiti sistemlerini tasarlamak giderek daha zor bir hale gelmektedir. Makine öğrenmesi tabanlı saldırı tespit sistemleri, izinsiz girişlerin hızlı, uyarlanabilir ve doğru tespiti için bir çözüm sunmaktadır. Ancak bu, sınıflandırıcıyı kimin değerlendirdiğine bağlı olarak, saldırı tespit sistemi sağlayıcısının ve kullanıcının, gizli ağ verilerini ve değerlendirme modelini paylaşmasını gerektirir. Sonuç olarak, her iki taraf için de bir gizlilik ihlali riski ortaya çıkar. Homomorfik şifreleme tekniği, şifrelenmiş verilerin bir şifre çözme anahtarı gerektirmeden işlenmesine izin vererek, bu tür gizlilik sorunlarının üstesinden gelmek için bir çözüm sunmaktadır. Bu tekniği kullanarak taraflar, değerlendirme için güvenilmeyen bir tarafla paylaşmadan önce bilgilerini şifreleyebilir. Bununla birlikte, homomorfik şifreleme tekniği maliyeti çok yüksek olabilecek işlemsel ek bir yüke yol açar. Bu nedenle asıl sınıflandırıcıların tespit yeteneklerinden ödün vermeden, toplam işlemlerin ve sınıflandırıcı devrelerin çarpma derinliğinin en aza indirilmesi hedeflenerek homomorfik sınıflandırıcılar tasarlanmaktadır. Bu tez, izinsiz ağa giriş tespiti için farklı makine öğrenme tabanlı sınıflandırıcıların performanslarını karşılaştırmakta ve ayrıca farklı şifreleme senaryolarını değerlendirmektedir. Farklı uygulamaların genel tespit doğruluğu, zamanlama performansı ve güvenlik endişeleri değerlendirilmekte ve tartışılmaktadır.

ACKNOWLEDGEMENTS

I would like to start by thanking my supervisor Prof. Erkay Savaş, who has supported me tremendously throughout this journey. His guidance, insight, and patience have been incredibly valuable to me, and I am honored to have had the opportunity to work under his supervision during my master's studies.

Furthermore, I would like to extend my sincere thanks to my thesis jury members Prof. Albert Levi and Assoc. Prof. Cihangir Tezcan, for their valuable comments and feedback.

I would also like to express my gratitude to The Scientific and Technological Research Council of Turkey (TÜBİTAK). This thesis was supported by TÜBİTAK under grant 118E725.

Finally, I would like to thank my parents, Mustafa and Derya, for their endless support every step of the way. They have always been my biggest believers, and not a day goes by that I do not feel privileged to have such a loving and caring family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BE Both Encrypted 15, 34, 35, 41, 42, 44, 45

DE Data Encrypted 15, 34, 41, 42, 44, 48, 50, 52, 53

DO Data Owner . . viii, 15, 16, 20, 21, 26, 43, 44, 45, 46, 47, 48, 49, 50, 52, 53, 54, 55, 59, 64

ME Model Encrypted 15, 34, 41, 42, 44, 45, 47, 50, 52, 53

MO Model Owner . viii, 15, 16, 20, 21, 42, 44, 45, 46, 47, 48, 49, 50, 52, 53, 59, 64

1. INTRODUCTION

With the growing number of cyberattacks, Intrusion Detection Systems (IDS) have become more relevant than ever before. The prevalence and diversity of network intrusions have become an immediate concern with the advancement of the internet. An intrusion detection system is a tool or software that monitors and analyzes the network traffic to warn the user about possible malicious activities. Machine learning-based IDSs evaluate the network traffic over a pre-constructed model and classify the activity as benign or malicious. Machine learning can offer fast, adaptable, and highly accurate intrusion detection solutions as it can learn from past data and intrusion incidences. Thus, several machine learning techniques were proposed for intrusion detection, and their performances were compared in previous studies in the literature (Tait, Khan, Alqahtani, Shah, Khan, Rehman, Boulila & Ahmad, 2021; Tsai, Hsu, Lin & Lin, 2009).

However, one of the crucial challenges of using IDS is protecting the privacy of the user's network data and the evaluation model due to the fact that IDS is provided by other parties as a service. In order to perform classification, either the user has to share their data with the IDS provider, or the IDS provider has to share the evaluation model with the user. Furthermore, if the classification is performed over cloud computing, a third party would have to access both the user data and the IDS model. The user might be reluctant to share their confidential network data with untrusted parties. Similarly, the IDS provider would not want to reveal any information about their classification model to users or any other untrusted parties as the model is a propriety piece of information that is one of its core business value propositions. Homomorphic encryption proposes a solution to this challenge by allowing the manipulation of encrypted data without having to decrypt it first. Furthermore, a fully homomorphic encryption (FHE) scheme enables any number of addition and multiplication operations to be performed on encrypted data. Hence, classification using machine learning models can be achieved with encrypted data, thus allowing users and the model owners to conceal their data from each other, intruders, and other third parties involved in the computation process.

In the dissertation by (Karaçay, 2019), three rule-based classifiers (Decision Trees, Neural Network, and Naive Bayes) were proposed for privacy-preserving intrusion detection systems. In the proposed scenario, the data owner (user) evaluates their network activity over the encrypted model received from the IDS provider. In this thesis, two more machine learning algorithms are proposed for classification: XGBoost and SVM. Furthermore, three different data encryption scenarios are introduced. In the first scenario, the user's data is encrypted while the model is open (and possibly the IDS provider runs the homomorphic evaluation); in the second scenario, the user's data is open, and the model is encrypted (the user runs the homomorphic evaluation); and in the third scenario, both the user data and the model is encrypted. The last scenario is included to accommodate the case that a third party running a cloud service is entrusted with running the homomorphic evaluation. The performance and security of all five classifiers are compared against each other under the proposed data encryption scenarios. In addition, two different network intrusion data sets are used for the evaluation of the classifiers.

As homomorphic operations have significant run-time overhead, it becomes critical to map the classification algorithms to arithmetic or logic circuits for homomorphic evaluation. The mapping technique must minimize the number of total operations and more importantly, the depth of the multiplication operations without compromising the detection accuracy, as multiplication is one of the most common and expensive operations when it is performed homomorphically. Different classification techniques are implemented based on the learning model's structure.

In this thesis, each classifier is implemented for the introduced encryption scenarios. Classification performances and running times of each implementation are presented and assessed. Later, the security of each implementation is discussed, and possible attack scenarios and remediations are proposed. This thesis provides a brief overview and comparison of different homomorphic classification approaches for intrusion detection. The applicability of different implementations in different use case scenarios is evaluated and discussed.

2. BACKGROUND INFORMATION

In this chapter, we provide background information on intrusion detection systems, homomorphic encryption, and machine learning algorithms. We also give a brief literature review relevant to the subject matter of this dissertation.

2.1 Intrusion Detection Systems

An intrusion detection system (IDS) is a tool or software that can detect suspicious activities in a network or system (Bace & Mell, 2001; Scarfone, Mell & others, 2007). The aim of an IDS is to warn the user of possible threats by monitoring the host systems or the network activity. The most common types of IDSs are network intrusion detection systems (NIDS) and host intrusion detection systems (HIDS).

NIDSs constantly monitor the network traffic to identify any unknown and possibly malicious activities. They consist of system sensors that are placed in key locations in the network to monitor and analyze the activity. An example NIDS is shown in Figure 2.1, where an IDS sensor is placed next to the external firewall. This placement allows the detection of attacks that might not be infiltrated by the initial defenses on the network.

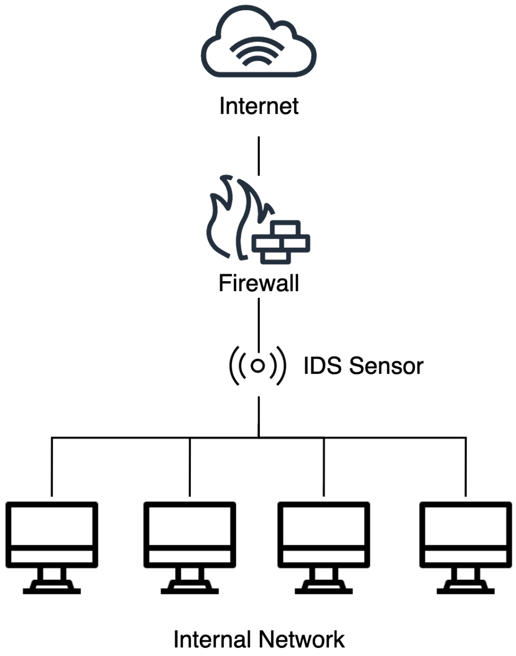

Figure 2.1 An example network intrusion detection system

HIDSs monitor the events of a single host to detect possibly malicious activities. It considers the internal system of the host, such as the system logs, running processes, etc., as well as the network traffic for the given host.

There are different detection approaches that IDSs can employ. Among the most popular ones are signature-based and anomaly-based detection approaches.

A signature-based IDS uses a predetermined set of characteristics that were extracted from previously known attacks to decide whether or not a particular network activity is malicious. However, the signature-based approach is limited when it comes to detecting new attacks since the signatures are based on previously identified malicious network activities.

An anomaly-based IDS uses a set of predetermined values that are attributed to normal network traffic. If the network activity diverges from this expected behavior, it will be marked as malicious.

2.2 Homomorphic Encryption

Given plaintext messages m_1, m_2, $\ldots$, m_i and their encryptions $E(m_1)$, $E(m_2)$, $\ldots$, $E(m_i)$, homomorphic encryption allows a function f to be evaluated on the encrypted messages without having to decrypt the messages first (Gentry, 2009). The result of the computation is a ciphertext, $E(f(m_1, m_2, \ldots, m_i))$, which can only be decrypted by the party who owns the decryption key. This encryption technique is especially useful if an untrusted party is involved in the computation process and the data owner wants to protect their sensitive data. The untrusted party can compute the result of the function f without ever seeing the content of the user's data. The result can later be decrypted by the data owner, who holds the decryption key. One of the several applications of homomorphic encryption is in the health sector, where the patient's confidential data (such as gene sequences, diagnostic test results, and other personal information) is evaluated by an institution for predictive analysis. To overcome privacy concerns, the institutions responsible for the evaluation of sensitive data can perform computations on encrypted data.

There are three most well-known homomorphic encryption schemes. These are partial homomorphic encryption, somewhat homomorphic encryption, and fully homomorphic encryption.

Partially homomorphic encryption (PHE) allows a limited set of operations to be performed on the encrypted message (Morris, 2013). Either addition or multiplication operation can be performed based on the encryption algorithms of choice. This limits the scope of computations that can be realized with this type of homomorphic encryption scheme.

Somewhat homomorphic encryption (SWHE) allows using both addition and multiplication operations within the same cryptographic scheme, but the number of operations one can perform without decrypting the message is limited (Belland, Xue, Kurdi & Chu, 2017) due to the increasing noise in the ciphertext that renders the ciphertext undecipherable, as explained subsequently.

The fully homomorphic encryption (FHE) allows multiple types of operations to be performed an unlimited amount of times, at least theoretically (Gentry, 2009). Each multiplication and addition operation adds some noise to the ciphertext, which makes the ciphertext indecipherable after a certain number of operations. Fully homomorphic encryption resolves this issue with a technique called *bootstrapping*.

Bootstrapping allows the decryption of a ciphertext homomorphically. As a result, if the noise of a ciphertext grows too much, it can be decrypted homomorphically to create a new ciphertext with a lower level of noise amount.

The batching method in the context of SWHE or FHE allows one to pack multiple plaintext data in slots (in fact, encoded as a single polynomial) before the encryption operation. As a result, the method can be utilized to perform the same homomorphic operation on all data items in those slots in the packed ciphertext; a technique usually known as Single Instruction, Multiple Data (SIMD) (Brakerski, Gentry & Halevi, 2013; Smart & Vercauteren, 2014). The SIMD technique can speed up the homomorphic computation process significantly as it allows computations to be performed on multiple slots simultaneously. In this thesis, the proposed encoding and classification algorithms rely on the batching method. Additionally, multiple input records can be evaluated concurrently using this method.

2.2.1 Fully Homomorphic Encryption Schemes

In this section, some of the widely used schemes and their implementations for fully homomorphic encryption are discussed. In practice, they are commonly used as SWHE schemes; as they support bootstrapping, they can be used as FHE schemes. Two of the most popular schemes are the Cheon, Kim, Kim, and Song scheme (CKKS) (Cheon, Kim, Kim & Song, 2017) and the Brakerski/Fan-Vercauteren scheme (BFV) (Fan & Vercauteren, 2012). The CKKS scheme supports floating-point arithmetic and allows only approximate computing, modifying the fraction of the ciphertext at each operation (Jiang & Ju, 2022). On the other hand, the BFV scheme is limited to integer arithmetic operations.

There are several libraries developed to implement FHE schemes. The Microsoft SEAL library (SEAL, 2021) implements both BFV and CKKS schemes, while the PALISADE library supports a broader range of FHE schemes (PALISADE, 2020).

In this thesis, Microsoft SEAL's BFV encryption scheme was used for homomorphic evaluations (Laine, 2017). The BFV scheme is semantically secure based on the hardness assumption of the Ring Learning with Errors (RLWE) problem. In the BFV encryption scheme plaintext space and ciphertext space are $\mathcal{R}_t$ and $\mathcal{R}_q$, respectively. The message is encoded in the plaintext space $\mathcal{R}_t = \mathbb{Z}_t[x]/(x^n + 1)$, where they are represented as polynomials of degree less than n and their coefficients are in modulus t. The encoded messages (plaintexts) are encrypted into ciphertexts of two polynomials, namely $\mathcal{R}_q \times \mathcal{R}_q$, where q stands for the ciphertext (coefficient) modulus. Each newly generated ciphertext contains a small noise component, which increases with each homomorphic operation applied to it, with the multiplication operation causing the most significant noise growth.

A parameter λ is selected to specify the security level of the encryption algorithm. The default security parameter of SEAL is set to 128 bits. Roughly speaking a given security level λ for an encryption scheme implies that breaking it requires a computational effort in the order of 2^λ steps. The coefficient moduli t, q, and the ring dimension n are selected based on the specified security level. The SEAL library also provides hard-coded default parameters for security levels of 128-bit, 192-bit, and 256-bit. The secret key sk is generated using the security parameter λ, and the public key pk is generated using sk. Additionally, a set of evaluation keys ek is generated, which is used in homomorphic operations such as multiplication and relinearization. While only the owner knows the secret key sk; pk and ek are open to public access as they are used in homomorphic operations that can be performed by anyone. A message can be encrypted using pk, and an encrypted message is decrypted using sk. The BFV scheme supports many homomorphic operations, including addition, multiplication, and rotation. After each multiplication operation, the ciphertext grows in size. For instance, while an original ciphertext consists of two polynomials in $\mathcal{R}_q$, a homomorphic multiplication increases the number of such polynomials to three. Even though a large ciphertext can be decrypted as it is, the computational performance will decrease as the ciphertext size increases. As a result, the ciphertext needs to be scaled back to its original size using the relinearization operation. There are other operations supported in the SEAL library in addition to those mentioned above: e.g., plain operations (`multiply_plain`, `add_plain`, etc., where one of the two operands is plaintext) and permutation over the values in the ciphertext slots using Galois automorphism.

2.3 Machine Learning Algorithms

In this section, we will briefly introduce five different machine learning algorithms, which are utilized to develop IDS models in this thesis.

2.3.1 Decision Trees

Decision trees are a supervised machine learning algorithm that uses a tree structure composed of decision rules at every branch node of the tree to make predictions. The outcomes of decision rules in the tree result in the selection of a leaf node. A decision tree consists of multiple decision nodes at which we perform comparison operations between a predetermined feature value and the input value for the corresponding feature. Based on the outcome of the comparison, either the left child node or the right child node is taken until finally, a leaf node is reached. Each leaf node holds a prediction value. The outcomes of these decision nodes for an input x_i lead to one of the leaf nodes, which contains the prediction result $\hat{y_i}$ for the given input.

2.3.2 XGBoost

XGBoost is a decision-tree-based machine learning algorithm that utilizes gradient boosting (Chen & Guestrin, 2016). An XGBoost model consists of classification and regression trees (CARTs). For each tree, parent nodes hold a comparison for deciding which node to follow next, and each leaf node has a score value l_j where j represents the jth leaf. Outcomes of the decisions for each tree are followed to reach a leaf score. The corresponding leaf scores from all the trees in the ensemble are added up to achieve a prediction score, which is used to make the decision. For binary classification, the classification is made based on the sign of the prediction score. The prediction of an input x_i on the XGBoost model can be represented as shown in the Formula 2.1.

$$\hat{y_i} = \sum_{k=1}^{m} f_k(x_i), \;\; f_k \in F \tag{2.1}$$

Here, m stands for the number of trees in the model, f_k is a tree function that outputs a leaf score, F is the set of all CARTs, and $\hat{y_i}$ is the model prediction (Chen & Guestrin, 2016).

2.3.3 Support Vector Machines

Support Vector Machine (SVM) is a supervised machine learning algorithm that is used for classification and regression problems. The algorithm works by constructing a hyperplane that optimally separates different classes from one another (Cortes & Vapnik, 1995). For multiclass classification, methods such as *one-against-all* and *one-against-one* have been proposed. These methods work by combining multiple binary classifiers (Hsu & Lin, 2002). In the *one-against-one* approach, every two class is separated using a hyperplane; and in the *one-against-all* approach, a hyperplane is used to separate each class from the rest of the classes. For linearly non-separable samples, kernel tricks might be used to map the inputs to a higher dimensional space and make the samples separable.

Given a set of training points $(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \ldots, (\boldsymbol{x_\ell}, y_\ell)$, $\boldsymbol{x_i}$ represents a data point (a d dimensional vector, where d stands for the number of features) and y_i represents its corresponding label. The training set is said to be linearly separable if a normal vector $\boldsymbol{w}$ and an offset value b exist such that the condition in Equation 2.2 is satisfied.

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1, \quad i = 1, ..., \ell \tag{2.2}$$

Here, the dimension of the weight vector $\boldsymbol{w}$ is equal to the dimension of the input space, which is the number of features d.

An optimal hyperplane that separates the training data points with the maximal margin is shown in Equation 2.3,

$$\boldsymbol{w_0} \cdot \boldsymbol{x} + b_0 = 0 \tag{2.3}$$

where ($\boldsymbol{w_0}$, b_0) is the unique set of arguments that maximizes the distance between the classes in the training data.

2.3.4 Naive Bayes

Naive Bayes is a simple classification algorithm that is based on Bayesian probability (Leung, 2007). It is considered to be highly scalable, and the classes are assumed to be conditionally independent.

Given a data point $x = (x_1, x_2, \ldots, x_d)$, where d is the number of independent features, it calculates $P(c_j|x)$ where c_j stands for the different classes that the data point might belong to. The data point is classified as the class that gives the highest probability as a result. Using the Bayes' Theorem, this probability can be shown as

$$P(c_j|x) = \frac{P(x|c_j)P(c_j)}{P(x)}. \tag{2.4}$$

2.3.5 Neural Network

Neural networks are a collection of nodes (neurons) connected to each other in a layered structure (Rumelhart, Hinton & Williams, 1986). The connection between the nodes is called edges, and each holds a weight value. Each neural network has an input layer, one or more hidden layers, and an output layer. The input is used to initialize the units of the input layer. The units in the successive layers apply a linear function on the outputs from the connected units of the previous layer to generate a total input. Later, a non-linear function is applied to the total input to produce the given unit's output, which is then forwarded to the following layer. The neural networks are trained using data points with known labels. The prediction of the network and the actual label are compared to generate an error value, which is then used to adjust the weights of the connections so that the model's prediction accuracy is improved.

2.3.6 Evaluation Metrics

The classification performances of the introduced algorithms are evaluated using the following metrics: accuracy, precision, recall, and false alarm rate (Kumar, 2014). If an instance is classified as positive, it is classified as malicious activity; otherwise, it is classified as normal activity. Given the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), the evaluation metrics are calculated as given below.

Accuracy: The accuracy rate is the ratio of correctly classified instances to the total number of instances. The accuracy of a classifier is calculated using Equation 2.5.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.5}$$

Precision: The precision metric is the ratio of correctly classified positive instances to all the instances classified as positive. It is calculated using Equation 2.6.

$$Precision = \frac{TP}{TP+FP} \tag{2.6}$$

Recall (Detection Rate): The recall metric is the ratio of correctly classified positive instances to the number of actual positive instances. It is calculated using Equation 2.7.

$$Recall = \frac{TP}{TP+FN} \tag{2.7}$$

False Alarm Rate (FAR): The FAR is the ratio of negative instances, which are incorrectly classified as positive, to the total number of negative instances. It is calculated using Equation 2.8.

$$FAR = \frac{FP}{TN+FP} \tag{2.8}$$

2.4 Literature Review

In the studies by Tsai et al. (2009) and Tait et al. (2021), the performances of several machine learning classifiers were compared for the purpose of intrusion detection. The authors in (Bost, Popa, Tu & Goldwasser, 2014) propose classification protocols for algorithms such as decision trees and Naive Bayes on encrypted data.

This thesis builds on top of a previous study conducted in (Karaçay, 2019), where homomorphic encryption is used to overcome privacy concerns, and a rule-based classification technique is used to evaluate machine learning algorithms. This is achieved by extracting rules from trained machine learning models to be used for classification. Rule-based solutions for intrusion detection have also been used in previous studies. The rule-based Naive Bayes and Neural Network algorithms introduced by Karaçay (2019) were constructed based on the works of Alashqur (2015) and Goodman, Higgins, Miller & Smyth (1992).

This thesis additionally introduces a privacy-preserving XGBoost classifier which was proposed in (Mağara, 2022), as the XGBoost algorithm generally performs better than other algorithms proposed for classification. Instead of the rule-based classification technique, the model as a whole is encoded in such a way that allows the user data to be evaluated homomorphically.

In the literature, several other approaches were taken to construct a privacy-preserving XGBoost classifier. Meng & Feigenbaum (2020) introduce a privacy-preserving XGBoost interface, which uses somewhat homomorphic encryption (SWHE) and order-preserving encryption (OPE) schemes for classification. The OPE scheme is used for the homomorphic comparison of the feature values. Deforth, Desgroseilliers, Gama, Georgieva, Jetchev & Vuille (2021) propose an algorithm to train and evaluate gradient boosted tree models using a multi-party computation setting. Xu, Li, Wang, Luo & Guo (2021) proposes a multisource distributed training model based on the Paillier homomorphic encryption (Paillier, 1999), which introduces a transfer learning IDS based on encrypted XGBoost.

This thesis, on the other hand, focuses on the evaluation of the XGBoost model instead of the training process. Additionally, we choose to use the SWHE scheme for privacy-preserving purposes, including for evaluating the comparison operations.

Also, a privacy-preserving linear SVM algorithm is introduced in this thesis. Binary classification with SVM is relatively easy compared to the other algorithms since it only requires the input to be multiplied with a normal vector $\boldsymbol{w}$, then added to a coefficient value b. Hence, no encoding or approximation techniques are necessary for the classification. The algorithm's simplicity allows for significantly fast classification even with homomorphic encryption. Several studies on privacy-preserving SVM classification exist in the literature; however, the current literature focuses primarily on more sophisticated non-linear classification algorithms (Laur, Lipmaa & Mielikäinen, 2006; Park, Lee, Cheon, Lee, Kim & Byun, 2019; Teo, Han & Lee, 2013). However, due to its simplicity, high speed, and relatively good accuracy, the linear SVM is investigated and analyzed in this thesis.

3. METHODOLOGY

In this thesis, the performances of different classifiers based on machine learning algorithms are evaluated considering three encryption scenarios. The model and user data are encoded in such a way that classifiers can be evaluated homomorphically. Three of these algorithms, namely Decision Trees, Naive Bayes, and Neural Network, are evaluated using a rule-based approach. In the rule-based approach, rules are extracted from the training data set, and the inputs are compared with the extracted rules to reach a classification decision. On the other hand, in XGBoost classifiers, a gradient boosting framework for classification, a different encoding methodology is used so that the input and the model are represented using bit strings. The final algorithm, Support Vector Machines (SVM), uses simple integer arithmetic for classification. These five algorithms are evaluated in terms of performance using different encryption scenarios that are introduced in this chapter.

3.1 Data Sets

For the evaluation of the algorithms that are described in this section, two different network intrusion data sets are used. The first one is the University of New Brunswick ISCX 2012 Intrusion Detection Evaluation Data Set (ISCXIDS2012) (Shiravi, Shiravi, Tavallaee & Ghorbani, 2012). This data set contains seven days of network activity for the system under inspection. For this study, only network activity from the days Saturday and Sunday were used. The chosen subset contains normal activity, brute-force attacks, and infiltration of the network from the inside. The original data set contains 18 features, with labels included.

The second data set is the KDD Cup 1999 data set (KDD'99) (Cup, 1999), which contains 42 features with labels included. It is one of the most widely used and popular intrusion detection evaluation data sets. However, the KDD'99 data set is also criticized for being an outdated data set that contains a large number of redundant records (Tavallaee, Bagheri, Lu & Ghorbani, 2009). Nonetheless, we have decided to use this data set since it is large data (both in terms of total feature numbers and total records) that allows us to test the scalability of our approach, especially for the SVM and XGBoost algorithms.

3.2 Data Pre-processing

To begin with, the redundant records on the data sets were removed. Especially the KDD'99 data set suffers from a large number of redundant records. This affects the model and testing accuracy. Hence, the redundant records are eliminated from the data set before any data processing takes place. Also, for the KDD'99 data set, different types of attacks (e.g., DoS, Probe, R2L) are mapped to a single attack label since only binary classification techniques are investigated in this thesis.

The WEKA machine learning framework is used for preprocessing the data sets (Holmes, Donkin & Witten, 1994). The original data sets were resampled without replacement, and a uniform distribution was applied so that a suitable number of training and test records were selected for evaluation. These data sets were separated as train and test sets by dividing the original sets into 70% and 30% portions, respectively.

After this, the next step is to discretize the data and eliminate unnecessary features to ensure that the data is suitable to be trained and tested on the classification algorithms.

The categorical features with a high number of values that would severely impact the performance of the classifiers were eliminated beforehand. The data were discretized based on the categorical feature with the highest number of values. For example, for the ISCX data set, the `sourceTCPFlagsDescription` feature takes 24 values. Since it is a categorical feature and cannot be discretized any further, the rest of the numerical features are discretized so that the highest number of values is 24 at most for all features. Similarly, for the KDD'99 data set, the number of categories (also referred to as "bins") was set to be 11, at most. The equal frequency binning method was used to discretize the feature values, meaning the data was divided into bins that had similar amounts of samples.

In order to eliminate irrelevant features and minimize the number of features we are working with, feature selection was employed on the data sets. The information gain (IG) evaluation method was used to eliminate the features of less importance (Kent, 1983). A feature is said to be more informative as its information gain value increases. For the Decision Tree, SVM, and XGBoost classifiers, the threshold value for the information gain was selected to be 0.05. For the Naive Bayes and Neural Network algorithms, the number of features is determined based on the number of rules the algorithms generate and the overall performance received from these rules as an excessive number of rules can endanger the feasibility of homomorphic evaluation.

3.3 Encryption Scenarios

Three different encryption scenarios are considered in this study:

1. Model owner encrypts the model (Model encrypted)
2. Data owner encrypts their data (Data encrypted)
3. Data owner encrypts their data, and the model owner encrypts the model (Both encrypted)

Model encrypted (ME): In the first scenario, the model owner (MO) encrypts the model and sends it to the data owner (DO). The DO uses the homomorphic encryption scheme to evaluate their plaintext input on the encrypted model. The encrypted result is sent back to the MO to be decrypted. If the result indicates an intrusion, the MO will inform the DO. This is the only scenario that has been investigated in the dissertation of Karaçay (2019).

Data encrypted (DE): In the second scenario, the DO encrypts their private data and sends it to the MO. The MO uses the homomorphic encryption scheme to evaluate the data using their open model. Then, the encrypted result is sent back to the DO to be decrypted. As a result, if there is an intrusion in their network, the DO will be informed by the result.

Both encrypted (BE): In the third scenario, both the model and the input data are encrypted. In this case, we assume that a third party will be performing the homomorphic evaluations. The DO encrypts their data, and the MO encrypts the model. The encrypted entities are sent to the third party that is responsible for evaluating the DO's data using the MO's model. The encrypted result is sent back to the parties. This scenario could also be implemented using multi-key homomorphic encryption, in which the result is encrypted by the keys of both DO and MO (Chen, Dai, Kim & Song, 2019). However, a single-key encryption technique is used in this thesis.

In Figure 3.1, the data transaction between the parties for all three encryption scenarios is illustrated.

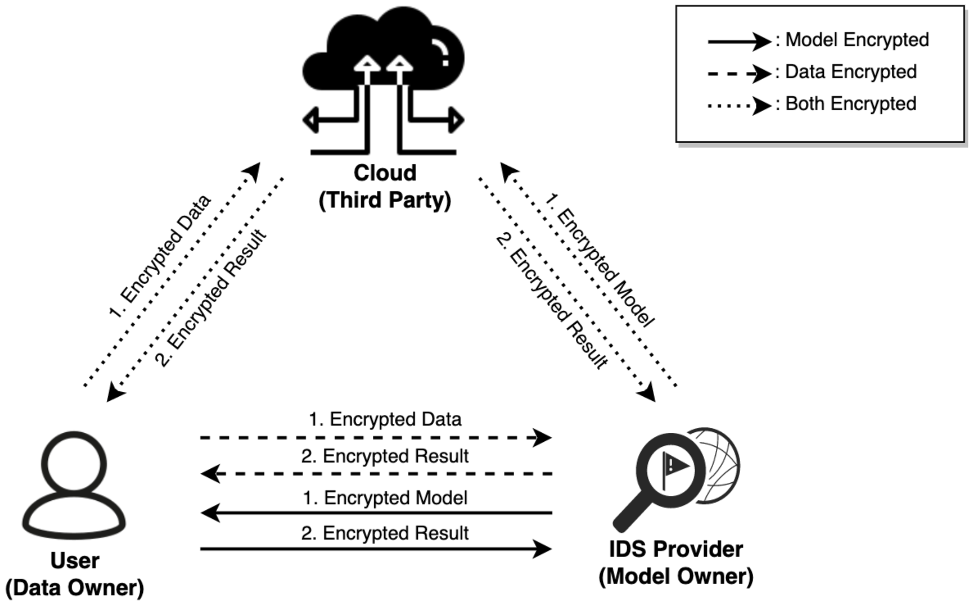

Figure 3.1 Three encryption scenarios

These three scenarios are considered for the different classifiers. The performance measurements of the different scenarios are compared with each other using both single-threaded and multi-threaded software implementations. In the latter case, we show that using parallel execution, we can accelerate the classification significantly as data points are processed independently.

3.4 Classifier Descriptions

In this section, we will provide the details of the mapping techniques we use for the classifiers to obtain the circuits for efficient homomorphic evaluations of the classification models.

3.4.1 XGBoost Classifier

XGBoost is a machine learning algorithm that uses decision-tree ensembles for classification. Unlike the Decision Tree, Naive Bayes, and Neural Network classifiers that are considered rule-based models; because of the structure of the XGBoost model, it is not feasible to extract rules for different classes. The XGBoost classification method discussed in this section has been introduced in the thesis by (Mağara, 2022). An XGBoost model consists of multiple trees. Each tree holds "scores" on its leaf nodes, and each parent node holds a value that is compared against a specific feature value of the input. The sum of the resulting leaf values for all trees is used to make a classification decision.

In Figure 3.2, two of the total 128 trees in the model generated for the KDD'99 data set are illustrated. Each tree in the model is evaluated separately for the input. For example, the root node of the first tree on the left checks whether the input value for the `count` feature is less than eight or not. If this condition holds, then the comparison in the left node is performed, which is "`dst_bytes` $<$ 1". If this condition also holds, the resulting leaf node will be the first one on the left, with the value of -0.584. All 128 trees will lead to a different leaf node with different values based on the input. The final classification score is the summation of the resulting leaf values for all trees.

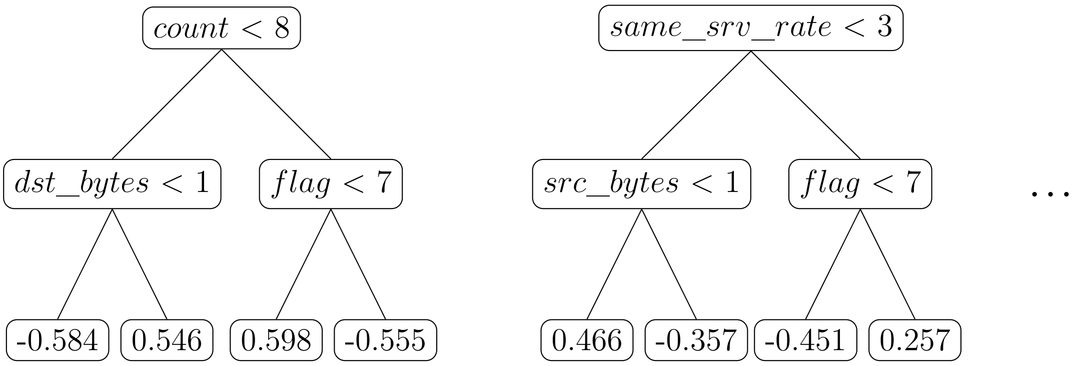

Figure 3.2 2/128 trees of the XGBoost model constructed for the KDD'99 data set.

The model generated for this study contains trees of depth two. In a tree, each parent node holds a less-than comparison operation that compares a feature value of the input with a predetermined value. If the feature value of the input is less than the node value, the comparison on the left child node is performed. Otherwise, the comparison on the right child node is performed. Based on the result of these comparisons, we reach a leaf node, which holds a decimal value. The resulting leaf values for each tree on the model are summed up to reach a final classification score. For binary classification, the input is classified as a malicious activity if the final score is a positive number. Otherwise, the input is classified as a normal activity if it is a negative number.

Normally, the leaf values of an XGBoost model consist of decimal values between -1 and 1. However, since the BFV scheme, which only works with integers, is used for homomorphic evaluation, these decimal values are multiplied with a power of 10 to approximate an integer score. It is made sure that this approximation is sufficient and makes no difference in the classification decision.

In order to classify an input, initially, the node comparison results for every parent nodes of all trees are calculated. Afterward, these node comparison results are used to obtain the scores from all trees, which, are, finally, are summed up to reach the final classification score.

3.4.1.1 Encoding

In order to realize the comparison operations on the parent nodes using homomorphic encryption, the model and the test values should be encoded in such a way as to facilitate fast and efficient homomorphic evaluation of the XGBoost trees.

The input data and the model are encoded using different methods. If a feature is compared with t different values on the model, that feature can take $t+1$ different values in our model. As a result, both the input values and the comparison values are represented using t bits.

The values that are taken by features are regrouped ("binned" or placed in bins) according to the node comparison values on the model to minimize the number of bits used in the encoding. For example, in the KDD'99 data set, all the features are discretized into 11 equal frequency bins. However, the model does not necessarily consider all the 11 values of a feature as a split point on the tree; it might consider only a subset of these values. We can simply show these categories as integers from 0 to 10. For example, the feature `dst_bytes` in the data set can take one of the following values:

$$(-inf - 2] : 0,\ (2 - 139.5] : 1,\ (139.5 - 326.5] : 2,\ (326.5 - 349.5] : 3,$$
$$(349.5 - 579.5] : 4,\ (579.5 - 1029.5] : 5,\ (1029.5 - 1588.5] : 6,\ (1588.5 - 2455] : 7,$$
$$(2455 - 4216.5] : 8,\ (4216.5 - 9285.5] : 9,\ (9285.5 - inf) : 10$$

However in the model, this feature values are compared only with the following five values: 1, 7, 8, 9, 10 (e.g., $dst\_bytes < 7$). In this case, it will be sufficient for the DO to encode their data with respect to the comparison values on the model. As a result, the input can have six distinct values, and the number of bits used to encode the value of `dst_bytes` can be reduced from 10 bits to 5 bits.

On the other hand, since we are utilizing the batching method with homomorphic encryption, the bit count used for all features should be the same and also be a power of 2. That means the feature with the highest number of binned values will be the feature that determines the bit size (and the number of values) for the rest of the features. Consequently, dummy values will be generated for features that contain a smaller number of binned values than the specified number of bits. For example, for the `dst_bytes` feature, it suffices for us to use only 5 bits for encoding. However, we need 8 bits to encode all features since the bit size must be a power of 2. As a result, for `dst_bytes` feature, three more random comparison values will be selected and used as a part of the encoding. While this causes extra bits to be used for encoding, increasing the number of bits might help the protection of the model's privacy. Using all the available bits to represent a feature, we share less information about the model, and the fact that the `dst_bytes` feature takes only five comparison values on the model is not disclosed.

After three more values are added, the following values will be the model's node comparison values for the `dst_bytes` feature: 1, 2, 3, 5, 7, 8, 9, 10. Then, we can define the encoding method for node comparisons.

Node value encoding: *The node comparison values are encoded using one-hot encoding, such that the most significant (i.e., the leftmost) bit corresponds to the smallest feature value.*

For the feature `dst_bytes`, the values on the model will be encoded as 10000000, 01000000, 00100000, 00010000, 00001000, 00000100, 00000010 and 00000001. For instance, when the node comparison operation is < 1, the node value will be encoded as 10000000. Similarly, when it is less than 9, then the node value encoding will be 00000010.

Now, we can explain the method used to encode the DO's input. Following the same example, the DO's data will be binned into nine intervals:

$$(-\infty,1),\ [1,2),\ [2,3),\ [3,5),\ [5,7),\ [7,8),\ [8,9),\ [9,10)\ [10,\infty).$$

The intervals are indexed from the largest to smallest, where the first index is 1 for the largest interval, and incremented by one.

Input encoding: *DO's input $\bar{x}$ is encoded feature-wise as $2^{i-1}-1$ if $\bar{x}$ is in the i^{th} largest interval.*

Hence, in the example the intervals for `dst_bytes` will be encoded as 11111111, 01111111, 00111111, 00011111, 00001111, 00000111, 00000011, 00000001 and 00000000, respectively. For instance, if $\bar{x} \in [7,8)$, it will be encoded as $x = 00000111$, which will result in bit sequence with a Hamming weight of 1, when it is bitwise multiplied with the node value encoding of the comparisons < 8, < 9, and < 10. Conversely, the same operation will yield an all-zero bit string for the comparisons with the value of 7 and less.

Note also that if the comparison values on the model are decimal numbers, they are rounded up to the next integer. For example, if a comparison such as $f < 7.5$ exists on the model, then 7.5 will be considered as 8 since there are no possible input values (categories) between the numbers 7.5 and 8.

3.4.1.2 Preparation of the input

For an XGBoost model made up of m trees, there exists m different root, left and right nodes where each node holds a comparison value. These values are compared against specific feature values of the DO's input. Each parent node holds a less-than comparison operation such as "`Destination Port` < 14.5", "`Source Port` < 2.5".

Ideally, the MO does not want to reveal which features are compared in each tree. However, in our proposed evaluation scenario, the DO's data is expected to be ordered in the same way feature comparisons are given in the XGBoost model.

Example 1. Consider an XGBoost model that consists of 3 trees ($m = 3$).

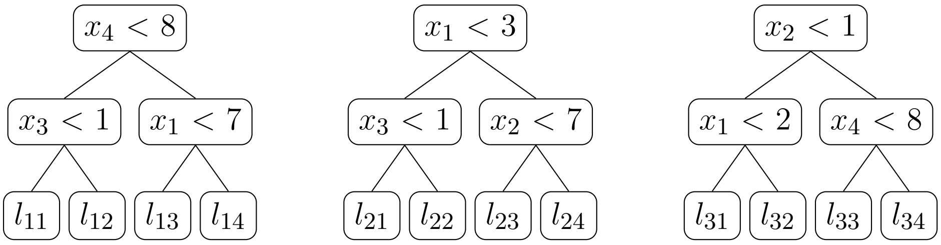

Figure 3.3 Example XGBoost model consisting of 3 trees.

In order to evaluate the DO's input on the model using the batching method, the input's feature value encodings must be given in the same order as the model. Hence, the DO's input is expected to be given in three arrays in the form of:

$$\begin{aligned} x_{root} &= x_4 \mid x_1 \mid x_2 \\ x_{left} &= x_3 \mid x_3 \mid x_1 \\ x_{right} &= x_1 \mid x_2 \mid x_4 \end{aligned}$$

However, if we require the DO to generate these arrays for evaluation, they would learn which features are used for comparison in each independent tree. This would reveal an essential information about the model to the DO.

In order to protect the model, we do not want to share the this additional information that reveals the features compared in each tree. To achieve that, the DO is expected to store all the feature values for their input in separate arrays called *input feature arrays* of length $m \times t$, where m and t stands for the number of trees and the number of bits used to represent the feature values, respectively. Hence, an input feature array contains the encoded value of the same feature m times.

The input feature arrays should be evaluated homomorphically so that the output is a single array that contains all feature values of the input ordered in the same way as the model. To achieve that, the MO is expected to generate arrays called *feature masks* along with the model itself. Feature masks are d arrays of length $m \times t$, which contain 1's in the slots where the given feature is present on the model, and 0's in the remaining slots. These masks are multiplied with their corresponding input feature vectors so that the value of the given feature is only present on the correct slots according to the model. Once each feature mask is applied to the corresponding input feature array, the resulting arrays are summed up to generate a single ordered input array.

A feature mask F_i is generated for all features f_i for $i = 1, 2, \ldots, d$ and it can take either the value of 0 or $2^t - 1$, which is repeated m times. The DO's input feature arrays are shown as X_1, X_2, ..., X_d, where d is the number of features. These arrays are multiplied by the corresponding feature masks generated by the MO. Finally, the resulting arrays are added up so that the features of the user's data are ordered in the same way as the features are ordered in the model. The Equation 3.1 will be used to reach the ordered input array x.

$$x = \sum_{k=1}^{d} (F_i \cdot X_i) \tag{3.1}$$

Example 2. Let us consider the model given in Example 1, which consists of 3 trees ($m = 3$), and assume that the feature values are encoded using 4 bits ($t = 4$) and we perform the comparisons for the root nodes of these trees. In the first tree, the feature f_4's value is compared; in the second tree, the feature f_1's value is compared; and finally, in the third tree, feature f_2's value is compared. Note that the feature f_3 is not used in the root nodes of any of the trees in this example.

Assuming this data set has four features in total (f_1, f_2, f_3, f_4), the following masks will be generated for each feature by the MO.

F_1: 0000 | 1111 | 0000

F_2: 0000 | 0000 | 1111

F_3: 0000 | 0000 | 0000

F_4: 1111 | 0000 | 0000

The user's input feature arrays are given below, where each x_i stands for the input encoded value of the feature f_i.

$$X_1 = x_1 \mid x_1 \mid x_1$$

$$X_2 = x_2 \mid x_2 \mid x_2$$

$$X_3 = x_3 \mid x_3 \mid x_3$$

$$X_4 = x_4 \mid x_4 \mid x_4$$

The ordered user input, which results after Formula 3.1 is applied, is given as:

$$x_{root} = x_4 \mid x_1 \mid x_2.$$

As all these operations are performed homomorphically, any party (including the DO and a third party) can perform them without learning the order formed inside the ciphertext.

Since three nodes exist (root, left, right) for a tree of depth two, the same operation will also be performed for the right and left nodes so that we obtain x_{left} and x_{right}. Consequently, for a single input value, three arrays for each node type (root, left, right) of size $m \times t$ will be generated.

Moreover, to improve the performance of the evaluation, multiple input records are stored in a single array. A ciphertext that contains N slots due to batching is capable of storing $N/(m \times t)$ number of input arrays. Therefore, the batching technique allows the evaluation of multiple inputs concurrently using a single ciphertext.

3.4.1.3 Node Comparison

Using Equation 3.2, the less-than comparison of each node is tested.

$$z = \sum_{i=0}^{t-1} (x_{k,i} \cdot y_i) \tag{3.2}$$

Here, a t-bit input feature value (for the feature f_k) represented as $(x_{k,t-1}||x_{k,t-2}||\dots||x_{k,1}||x_{k,0})$ and a t-bit node value represented as $(y_{t-1}||y_{t-2}||\dots||y_1||y_0)$ are given as inputs. If the output z is 1, the user's value is less than the model's value. In this case, the left subtree will be taken. Otherwise, the right subtree will be taken.

Since batching is used for homomorphic evaluation, an ordered input vector (for the root, left or right nodes) can be evaluated over its corresponding model vector concurrently. As a result, Equation 3.2 will translate to one multiplication and $\log_2(t)$ addition operations for a single node comparison. The result for each comparison will be stored in the $(t \times i)$th slot of the array given that $i = 0, 1, \dots, N/t - 1$.

Example 3. Assume a scenario where feature values are encoded using 4 bits ($t = 4$), and the XGBoost model consists of 3 trees ($m = 3$). Consider that x_{root} stands for an ordered input vector for the root nodes, and y_{root} stands for the vector that holds the feature comparison values for the root nodes. The node comparison for a single input can be achieved as follows:

$$
\begin{array}{cccc}
 & x_{root} & = & 1111 \mid 0000 \mid 0111 \\
\text{x} & y_{root} & = & 0100 \mid 0001 \mid 1000 \\
\hline
 & int_1 & & 0100 \mid 0000 \mid 0000 \\
+ & int_1 << 1 & & 1000 \mid 0000 \mid 0000 \\
\hline
 & int_2 & & 1100 \mid 0000 \mid 0000 \\
+ & int_2 << 2 & & 0000 \mid 0000 \mid 0000 \\
\hline
 & z_{root} & = & 1{*}{*}{*} \mid 0{*}{*}{*} \mid 0{*}{*}{*}
\end{array}
$$

Here, * stands for intermediary calculation values that are not important for the result calculation process. Before decrypting the result, these intermediary values will be masked as they may leak sensitive information.

3.4.1.4 Result Calculation

After the comparison result of each node is calculated, these results are used to get the score of the correct leaf node using the following equations.

$$
\begin{aligned}
&c_1 = z_{root} \times z_{left} \times l_1 \\
&c_2 = z_{root} \times (1 - z_{left}) \times l_2 \\
&c_3 = (1 - z_{root}) \times z_{right} \times l_3 \\
&c_4 = (1 - z_{root}) \times (1 - z_{right}) \times l_4 \\
&Score = c_1 + c_2 + c_3 + c_4
\end{aligned}
$$

Here, $z_{root}, z_{left}, z_{right}$ values are the comparison results for the root, left, and right nodes respectively. The l_i values are the leaf scores of the tree, numbered from leftmost to rightmost. c_i holds the leaf value if and only if its corresponding leaf node l_i is reached given the comparison results z.

For example, if the root node comparison is satisfied z_{root} will be 1. If the left node comparison is satisfied, z_{left} will also be 1. As a result c_1 will be equal to l_1 and c_2, c_3, c_4 will all be equal to 0.

Hence, the summation of all four c values will give us the value of the achieved leaf node for a single tree without exposing which leaf (or which feature value combination) results in the said value.

The operations above are simplified to minimize the number of multiplication operations, and the final version of the operation is shown in Equation 3.3, given

$p_1 = l_1 - l_2$, $p_2 = l_2 - l_4$, $p_3 = l_4 - l_3$, and $p_4 = l_4$.

$$Score = (z_{root} - 1) \times p_3 \times z_{right} + (z_{left} \times p_1 + p_2) \times z_{root} + p_4 \tag{3.3}$$

Once the scores of all trees are calculated, they are summed up to reach a final classification score. The tree scores in the array are summed up using the logarithmic summation method. The number of trees is either selected to be a power of 2, or $\hat{m} - m$ empty trees are appended to the arrays to allow slot-wise logarithmic summation of the tree scores ($\hat{m}$ stands for m's next power of 2). As a result, in a ciphertext each $(i \times \hat{m})$th slot, where $i = 0, 1, \dots, N/\hat{m} - 1$, holds the classification score of an input. Finally, the intermediary values generated as a result of logarithmic summation are masked by adding a random integer R selected from the range of $[0,\ p)$.

3.4.2 Support Vector Machines (SVM) Classifier

Support Vector Machines (SVM) is a supervised machine learning algorithm that aims to create an optimal hyperplane between training samples that belong to different classes by finding a normal vector $\boldsymbol{w}$. An offset value b is used such that the classes are separated with a maximal margin. Given that the input contains d values, $\boldsymbol{w}$ and the input $\boldsymbol{x}$ are vectors of size d for binary classification.

Formula 3.4 is used to classify a data point x.

$$y = (\boldsymbol{w} \cdot \boldsymbol{x} + b) \tag{3.4}$$

If the resulting label y is a negative value, then the input is classified as regular network activity. Otherwise, it is classified as malicious.

The normal vector $\boldsymbol{w}$ and the offset value b are extracted from the model. For binary classification, the size of the vector $\boldsymbol{w}$ is equal to the number of features d in the data set, and a single offset value is used.

The dot product of $\boldsymbol{w}$ and $\boldsymbol{x}$ in the Formula 3.4 translates to a single multiplication operation and $\log_2(\hat{d})$ addition operations with the batch encoding technique, where $\hat{d}$ stands for d's next power of 2. The d number of values of the normal vector $\boldsymbol{w}$ is stored in a single array, and similarly, the d number of values of the input $\boldsymbol{x}$ vector are also stored in a single array. The multiplication of these two arrays, followed by the logarithmic summation of the array elements, results in the dot product of $\boldsymbol{w}$ and $\boldsymbol{x}$. Finally, with the addition of the offset value b, the classification result y_i is achieved. The intermediary values that are obtained as a result of logarithmic addition are masked with the addition of random integers selected from the range of $[0,\ p)$. Hence, one multiplication operation and $\log_2(\hat{d}) + 2$ addition operations suffice to classify a single data point. Additionally, the batching method allows the storage of $N/\hat{d}$ inputs in a single ciphertext, further speeding up the evaluation process.

The simplicity of the SVM model evaluation makes the computation time faster than any other machine learning algorithm investigated in this thesis and allows to classify a greater number of test data points homomorphically in a short amount of time while still yielding a high accuracy rate. Additionally, we used a subset of the KDD'99 data set containing 1,059,932 data points to evaluate the performance and accuracy of the SVM implementation with a large volume of data.

3.4.3 Rule-based Classifiers

In this thesis, three rule-based classifiers (Decision Tree, Neural Network, and Naive Bayes) that were introduced in the dissertation of Karaçay (2019) are also evaluated in the three aforementioned encryption scenarios. Using each algorithm, a number of malicious signatures (which will be referred to as *rule signatures*) are generated. These signatures are then compared against the input to decide whether or not the input is malicious. If the input matches any of the signatures, it will be considered a malicious activity.

The DO's input will be used to generate a *record signature*, and the classifier will be given as a set of *rule signatures*. Here, each feature is transformed into the categorical data type, whereby it can be equal to a certain number of values. This approach is similar to the one used in our XGBoost method. However, in rule-based classifiers, the predicates always check the equality, whereas it is a less-than comparison in XGBoost.

The length of each signature is $d \times t$ where d stands for the number of total features and t stands for the number of total values a feature takes. A *record signature* will be given as

$$v_{11},\, v_{12},\, \ldots,\, v_{1t} |\ v_{21},\, v_{22},\, \ldots,\, v_{2t}\ |\ \ldots |\ v_{d1},\, v_{d2},\, \ldots v_{dt}.$$

Here, the one-hot encoding technique is used to represent the values each feature takes. For all features f_j where $j = 1, 2, \ldots, d$, if the feature f_j of the input is equal to its ith value, then the v_{ij} bit will be set to 1. The other bits for the jth feature will be set to 0. This way, the resulting record signature will represent the input itself.

This record signature will be compared against a set of rule signatures. Each classifier described below will generate a number of rule signatures that will be used to detect malicious inputs.

For the rule signatures, the value of *don't care* will be assigned to the t bits of the features that are not used in detection. Based on a predicate, which captures one of the conditions in the malicious activity, if a feature is known to take a certain value v_{ij}, that slot will be set to 1. Similarly, if a value is known not to equal a specific value v_{ij}, that slot will be set to 0. Since there are three possible states a feature value can take (value holds, value does not hold, and don't care), 2 bits are used to encode these states. The values 0, 1 and *don't care* are encoded with [0,0], [1,0] and [1,1], respectively.

The comparison of a record signature with a rule signature is achieved using Equation 3.5.

$$\boldsymbol{\tau} \leftarrow (\mathbf{s} \vee \mathbf{M_i}[\mathbf{1}]) \oplus \mathbf{M_i}[\mathbf{0}] \tag{3.5}$$

In this thesis, we adopt the notation used in (Karaçay, 2019). Here $\boldsymbol{\tau}$ represents the result vector that holds the comparison value of the rule and the record signatures. $\mathbf{s}$ represents the record signature and $\mathbf{M_i}$ represents one of the rule signatures. Since the feature values in the rule signatures are encoded using two bits, $\mathbf{M_i}[\mathbf{0}]$ stands for the first bit of the feature values while $\mathbf{M_i}[\mathbf{1}]$ stands for the second bit of the feature values.

If the result of this equation $\boldsymbol{\tau}$ is equal to $(0)^{1\times\lambda}$ where λ stands for the length of the signatures, that means the rule signature matches the record signature.

The entire array $\boldsymbol{\tau}$ is summed up using logarithmic summation. As a result, the first index holds 0 if the signatures match, otherwise it holds a non-zero value. The index that holds the result is multiplied with a random integer in the range of $[0,\ p)$, and the intermediate indices are multiplied with 0 to prevent information leakage. The result for each rule signature is placed in a random index of the ciphertext using the shift operation.

Once again, as was the case for other algorithms, the batching technique allows for the classification of multiple inputs with a single ciphertext. A ciphertext can contain $N/\widehat{(d \times t)}$ rule signatures, where $\widehat{(d \times t)}$ stands for $(d \times t)$'s next power of 2.

3.4.3.1 Decision Tree Classifier

A decision tree consists of parent and leaf nodes, where each leaf node holds a classification value and each parent node holds a feature comparison value. In each parent node, a feature value of the input is checked against a value of the given feature. If the comparison holds, the left subtree is taken. If it does not hold, the right subtree is taken. The resulting leaf value is the classification result. In Figure 3.4, an example model is given.

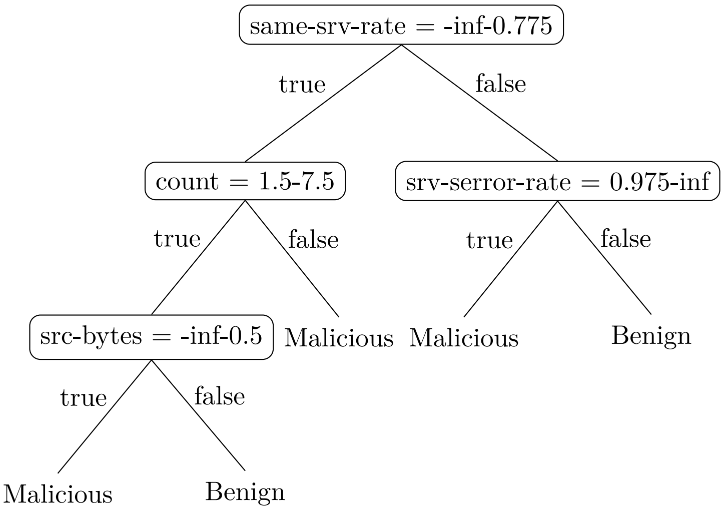

Figure 3.4 An example decision tree model.

In the decision tree model, each route from the root node to different leaf nodes is referred to as a path. The rule signatures are based on the paths on the model. In a path, certain input features are compared with particular values of that feature. Not all features are necessarily checked in a given path. That is why we also have *don't care* conditions for features that are not considered on a given path. The paths that lead to a "Malicious" leaf, will be the rule signatures for the decision tree classifier. For the model in Figure 3.4, the following paths are the rule signatures:

- **Rule 1:** (same-srv-rate = -inf-0.775) $\wedge$ (count = 1.5-7.5) $\wedge$ (src-bytes = -inf-0.5)
- **Rule 2:** (same-srv-rate = -inf-0.775) $\wedge$ (count $\neq$ 1.5-7.5)

- **Rule 3:** (same-srv-rate $\neq$ -inf-0.775) $\wedge$ (srv-serror-rate = 0.975-inf)

Two different binary strings, stored in matrix $\boldsymbol{M}$, will be generated for each rule signature. The first column of the matrix $\boldsymbol{M}$ will hold the first bit of the encodings $\mathbf{M}[\mathbf{0}]$, and the second column will hold the second bit of the encodings $\mathbf{M}[\mathbf{1}]$.

$$\mathbf{M}[\mathbf{0}] : v_{11}, v_{12}, \ldots, v_{1t} | \; v_{21}, v_{22}, \ldots, v_{2t} \; | \ldots | \; v_{d1}, v_{d2}, \ldots v_{dt}$$

$$\mathbf{M}[\mathbf{1}] : v_{11}, v_{12}, \ldots, v_{1t} | \; v_{21}, v_{22}, \ldots, v_{2t} \; | \ldots | \; v_{d1}, v_{d2}, \ldots v_{dt}$$

The rules given for the binary tree in Figure 3.4 are converted into binary strings in the following way :

- If we know a feature f_i has to be equal to its jth value for that rule to hold, the bit for v_{ij} in $\mathbf{M}[\mathbf{0}]$ has to be set to 1 and the same bit in $\mathbf{M}[\mathbf{1}]$ will be set to 0. For example in Rule 1, the bits that stand for the value (inf-0.775) for the feature `same-srv-rate` will be set to 1 and 0 on $\mathbf{M_1}[\mathbf{0}]$ and $\mathbf{M_1}[\mathbf{1}]$, respectively. The bits for rest of the `same-srv-rate`'s values will be set to 0 on both $\mathbf{M_1}[\mathbf{0}]$ and $\mathbf{M_1}[\mathbf{1}]$, since we know the same feature cannot be equal to another value on the same path.

- If we know a feature f_i has to be **not** equal to a certain value v_{ij} for that rule to hold, the bits of v_{ij} in both $\mathbf{M}[\mathbf{0}]$ and $\mathbf{M}[\mathbf{1}]$ has to be set to 0. For example for Rule 2 to hold, the input's value for the feature `count` should **not** be equal to (1.5-7.5). As a result, the feature bit that corresponds to the value of (1.5-7.5) should be set to 0 in both $\mathbf{M_2}[\mathbf{0}]$ and $\mathbf{M_2}[\mathbf{1}]$. The remaining values for the `count` feature will be set to *don't care*, hence the value bits will be set to 1 in both $\mathbf{M_2}[\mathbf{0}]$ and $\mathbf{M_2}[\mathbf{1}]$.

- If a feature is not considered at all in a given rule, then all its value bits will be set to 1 for both $\mathbf{M}[\mathbf{0}]$ and $\mathbf{M}[\mathbf{1}]$. For example in Rule 2, the feature `src-bytes`'s value is not considered. Then, the bits that represent the values of `src-bytes` ($v_{\texttt{src-bytes}1}, v_{\texttt{src-bytes}2}, \ldots, v_{\texttt{src-bytes}t}$) have to be set to 1 in both $\mathbf{M_2}[\mathbf{0}]$ and $\mathbf{M_2}[\mathbf{1}]$.

3.4.3.2 Naive Bayes Classifier

The Naive Bayes classifier uses the Bayesian theorem in Equation 3.6 to estimate the probabilities of classes and select rules for classification.

$$P(c_j|r) = \frac{P(r|c_j)P(c_j)}{P(r)} \tag{3.6}$$

Given a new rule r, the probability that it belongs to a class c_j in the set of classes C is calculated. Then, the class with the highest conditional probability $P(c_j|r)$ is selected as the class of the given rule r. Since $P(r)$ does not depend on the class information, it can be removed from the equation, and the estimation can be used for the classification.

Since the Naive Bayes classifier assumes that the data consists of d number of independent features, the approximation given in Equation 3.7 is used to calculate the value of $P(r|c_j)$. In this equation, v_{ik_i} represents the value rule r takes for its ith feature.

$$P(r|c_j) = \prod_{i=1}^{d} P(v_{ik_i}|c_j) \tag{3.7}$$

A set S that contains all possible combinations of the distinct feature values is generated. Each $s_i \in S$ is a candidate rule r that is yet to be assigned to a class. To decide which class s_i belongs to, Equation 3.6 is calculated for each possible class c_j. The candidate rule s_i is added to the list of rules R_j of the class c_j that leads to the highest probability $P(c_j|s_i)$ among all other classes in C. Since only two classes are considered in this study (Benign and Malicious), we compare the “Malicious” rules with the test input to reach a classification decision.

Since the rules generated with the Naive Bayes technique are selected from a set of all feature combinations S, these rules contain an exact value for every possible feature. Hence, there are no *don’t care* values to consider for the rule encoding. In the binary string generation for the rules, if the feature f_j takes its ith value, the v_{ij}th bit will be set to 1 for $\mathbf{M_k}[\mathbf{0}]$ and it will be set to 0 for $\mathbf{M_k}[\mathbf{1}]$. All other value bits for feature f_j will be set to 0 for both $\mathbf{M_k}[\mathbf{0}]$ and $\mathbf{M_k}[\mathbf{1}]$.

This method requires the MO to generate all possible combinations for different signatures. Hence, as the number of features and the number of distinct values of a feature increase, it will be more difficult to generate the set of records in S. For that reason, in this thesis, minimized versions of the data sets are used for the evaluation of the Naive Bayes classifier.

The number of rules produced by the Naive Bayes algorithm is proportional to the number of unique malicious records in the training set. As the number of unique malicious records increases, the number of produced malicious rules is expected to increase as well. A high number of rules makes the timing performance of the classifier significantly drop when evaluated homomorphically. For this reason, in order to limit the number of unique malicious data points, a significantly smaller subsample of the original training set is used to generate the Naive Bayes rules.

3.4.3.3 Neural Network Classifier

The Neural Network classifier is implemented using the information-theoretic approach (Higgins & Goodman, 1991). The information-theoretic measure is used to extract a set of classification rules from the training data. However, this approach is modified so that the computational complexity of the classification can be decreased for homomorphic evaluation. The traditional information-theoretic approach and the rule-based one will be explained, respectively.

The information-theoretic measure is used to extract rules from the training set by measuring how informative a rule is. The informativeness of a rule can be measured by looking at the correlation between the feature values of an input and its label. This correlation is calculated using the J-measure of the training data points as shown in Formula 3.8 (Smyth & Goodman, 1992).

$$J(C;x_i) = P(x_i)\left(P(c_j \mid x_i)\log\frac{P(c_j \mid x_i)}{P(c_j)} + P(\bar{c_j} \mid x_i)\log\frac{P(\bar{c_j} \mid x_i)}{P(\bar{c_j})}\right) \tag{3.8}$$

The rules with the highest J-measure are considered to be the most informative rules. Here, C stands for all possible classes, c_j refers to the class of the input, and $\bar{c_j}$ stands for all classes except c_j. Since we use binary classification, $\bar{c_j}$ will refer to the complementing class. x_i stands for all feature values of the ith input, in other words, it is the input without the class information.

Once the J-measure of all training inputs is calculated, extracting the most informative rules from the training set R is the next step. The rule selection technique proposed by Higgins & Goodman (1991) is used to extract the rules. The extraction technique is as follows:

1. For each rule $r_i \in R$, r_i is selected for evaluation and referred to as the *parent rule*.

2. The J-measure of the parent rule is calculated.

3. A single feature is removed from the parent rule to generate a set of rules of order $d-1$, where d is the parent rule's feature size. These are referred to as *child rules*. J-measure of each child rule is calculated.

4. The rule with the greatest J-measure is selected among the parent rule and its child rules.

5. If two rules have the same J-measure, the one with the lowest order is selected.

6. If two rules have the same J-measure and the same order, one of the two rules is randomly selected.

7. If the selected rule is the parent rule, then we return the rule as an extracted rule. Otherwise, this algorithm repeats starting from Step 2, with the selected rule as the new parent rule.

Once a set of rules are extracted from the training set, a neural network is generated using these rules based on posterior probability calculations.

If the input's feature values match any of the feature values of an extracted rule, then this rule is referred to as a fired rule. A fired rule may belong to any of the classes. Once fired rules are identified, they are stored in different sets F_j, based on the class c_j they belong to. The logarithmic posterior probability of each class given the fired rules for that class F_j is calculated, which can be estimated using Equation 3.9 (Goodman et al., 1992):

$$\log P\left(c_j \mid x_1,\ldots,x_{|F_j|}\right) = \mathcal{C} + \log P\left(c_j\right) + \sum_{i=1}^{|F_j|} \frac{P(c_j|x_i)}{P(c_j)} \tag{3.9}$$

Here, $\mathcal{C}$ stands for a constant value that can be eliminated from the calculation with further approximation of the equation. For the explanation of which, see (Karaçay, 2019).

Finally, the class with the highest probability is chosen as the classification result.

The setback of this approach is that, in order to classify an input, one needs to perform the operations in Formula 3.9 homomorphically, which brings a significant overhead performance-wise.

Instead, a rule-based comparison approach is taken to estimate the neural network classification. A subset of malicious rules is generated for homomorphic classification by refining the original set of malicious rules R_M.

Multiple rules that have the same feature values could exist in our set of rules R. However, these rules might belong to different classes. Hence, we iterate over each malicious rule, and if the current malicious rule is equal to or the subset of one of the benign rules, then we add this rule to a set called fired rules F. Then at each iteration, we calculate the posterior probability of the rules in F. Similarly, the posterior probability for the current malicious rule $r_i \in R_M$ is calculated. If the sum of the posterior probability of the rules in F is higher than the posterior probability of r_i, then r_i is removed from R_M. On the other hand, if r_i's posterior probability is greater, then the rule remains in the set. Finally, this set of refined malicious rules is returned and used to classify test inputs.

The input is compared with the rules $r_i \in R_M$, and if the feature values of the input match r_i, then the input is classified as malicious.

4. IMPLEMENTATION RESULTS

In this chapter, the performance of the proposed algorithms and encoding schemes are investigated using their software implementations. First, the characteristics of the data sets used for each algorithm are introduced. Afterward, the classification performance of the algorithms is evaluated. Finally, the timing performances are included in the assessment.

To begin with, training sets are used to generate models using the proposed machine learning algorithms. The models are extracted and modified so that homomorphic classification can be applied to the instances in the test set. The training processes for the XGBoost and SVM algorithms are implemented on Python, using the scikit-learn machine learning library (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot & Duchesnay, 2011). On the other hand, the training parts for the Decision Tree, Neural Network and Naive Bayes algorithms are implemented on Java. The WEKA machine learning software is used for the Decision Tree training (Holmes et al., 1994). The model training is done on a machine with macOS operating system. The homomorphic evaluations are done on a machine with Linux (Ubuntu) operating system.

SEAL's default security level, which is 128 bits, was used for all classifier implementations, and the rest of the parameters were selected according to the specified security level. Some of the selected parameters provide a slightly higher level of security than specified; however, the level does not fall below 128 bits under any circumstance. For the XGBoost classifier, the ring size is selected as 2^{13}, the ciphertext modulus size is selected as 215 bits (with the exception of 218 bits for the BE version), and SEAL's default 20-bit value (for the given ring size) is used as the plaintext modulus. For the SVM classifier, the ring size is selected as 2^{12}, the ciphertext modulus size is selected as 109 bits, and SEAL's default 20-bit value (for the given ring size) is used as the plaintext modulus. For the ME and DE versions of the rule-based algorithms; the ring size is selected as 2^{12}, the ciphertext modulus size is selected as 109 bits, and the plaintext modulus is selected as 40961. For the

BE version of the rule-based algorithms, the ring size is selected as 2^{13}, the coefficient modulus is selected as 215 bits, and the SEAL's default 20-bit value (for the given ring size) is used as the plaintext modulus.

4.1 Data Set Information

The Decision Tree (DT), XGBoost, and SVM algorithms share the same training and test sets. These data sets are pre-processed as explained in Section 3.2. The information about the data sets is given in Table 4.1:

Data Set	**Train size**	**Test size**	**No. of attacks**	**No. of ft.**	**No. of bins**
ISCX	9460	4056	428	10	24
KDD'99	10534	4517	2569	20	11

Table 4.1 Data set information for XGBoost, SVM, and Decision Trees

A different subset of the KDD'99 data set is used to generate a larger test set that contains 1,059,932 records. We use this test set to measure the performance of the SVM algorithm, which outperforms the other algorithms in terms of timing performance. We do this to observe how the SVM algorithm scales for much larger data sets. The timing performance of the SVM algorithm using this data set will be referred to as SVM*.

The data sets are processed differently for the Neural Network (NN) and Naive Bayes (NB) algorithms. The Naive Bayes algorithm generates all combinations of the given features and picks the rule signatures from this set of combinations. Hence, as the number of features increases in a set, the number of signatures also increases and the rule generation time grows exponentially. Similarly, the number of selected rule signatures increases as the range of unique feature values for malicious samples increases in the training set. Since it becomes difficult to evaluate a high number of signatures homomorphically, a smaller data set is used for training.

Additionally, the rule signatures generated by the Neural Network algorithm also grow with the number of training records used. Furthermore, as the number of training records increases, it was observed that the classification performance of the Neural Network algorithm decreases and tends to shift significantly at each run of the algorithm. This shift is caused by the randomized rule selection part of the Neural Network algorithm. When the J-measures and the degrees of the candidate rules are equal, one of the rules is selected randomly to be included in the final set of signatures.

As a result, a much smaller training set is used, and the number of features and categories are minimized for both algorithms. The new feature, category, and training set sizes are selected so that the data sets yield a conservative number of rules while still preserving decent accuracy, precision, and recall rates.

The information about the data sets used for Neural Network and Naive Bayes algorithm is given in Table 4.2.

Data Set	**Train size**	**Test size**	**No. of attacks**	**No. of ft.**	**No. of bins**
ISCX	94	4056	428	7	4
KDD'99	51	4517	2569	7	4

Table 4.2 Data set information for Naive Bayes and Neural Network

For the rule-based classifiers, the number of rules generated for each data set is given in Table 4.3.

Data Set	**Decision Trees**	**Neural Network**	**Naive Bayes**
ISCX	7	2	48
KDD'99	15	2	48

Table 4.3 Number of rules generated per classifier

4.2 Results

The classification and timing performances of the algorithms under different encryption scenarios are discussed in this section.

4.2.1 Classification Results

The classification performances of the algorithms are evaluated using the following metrics: accuracy, precision, recall (detection rate), and false alarm rate (FAR) metrics. The accuracy, precision and recall metrics for the ISCX and KDD'99 data sets are given in Figure 4.1 and Figure 4.2, respectively. The false alarm rates for both data sets are shown in Figure 4.3. All classifiers were implemented in plaintext as well as homomorphically. It was observed that homomorphic evaluation did not alter the classification performance of the algorithms. Namely, the classification results remained the same for plaintext and homomorphic evaluations.

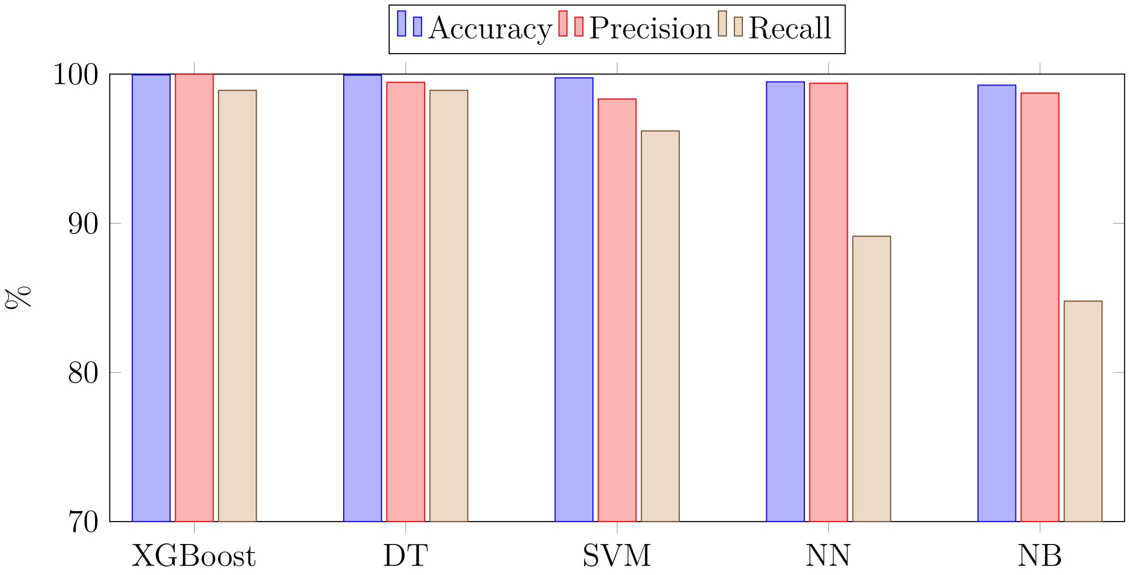

Figure 4.1 Comparison of the machine learning algorithms on the ISCX data set

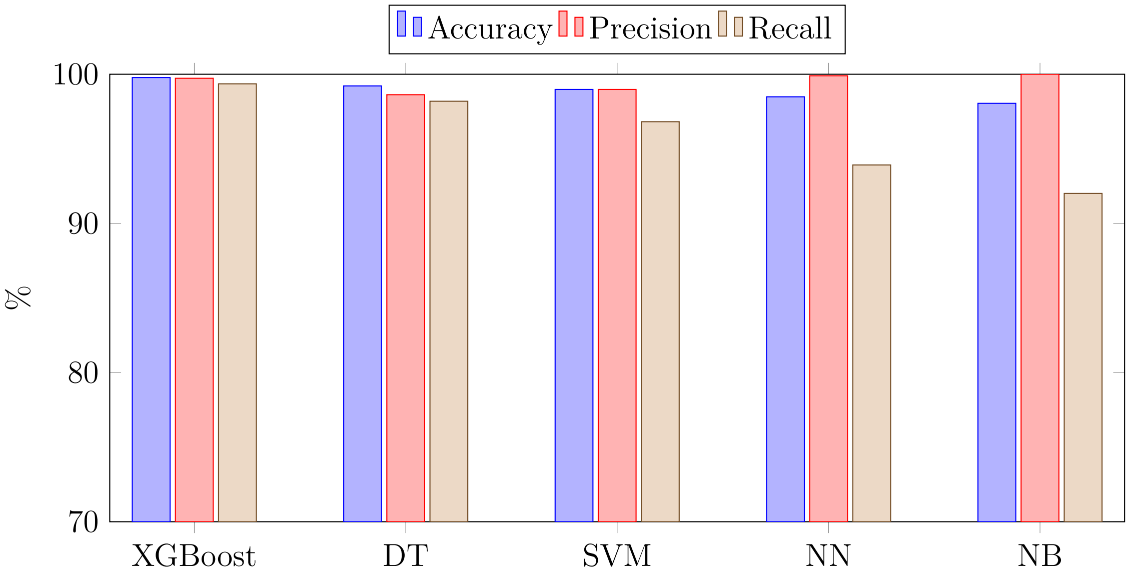

Figure 4.2 Comparison of the machine learning algorithms on the KDD'99 data set

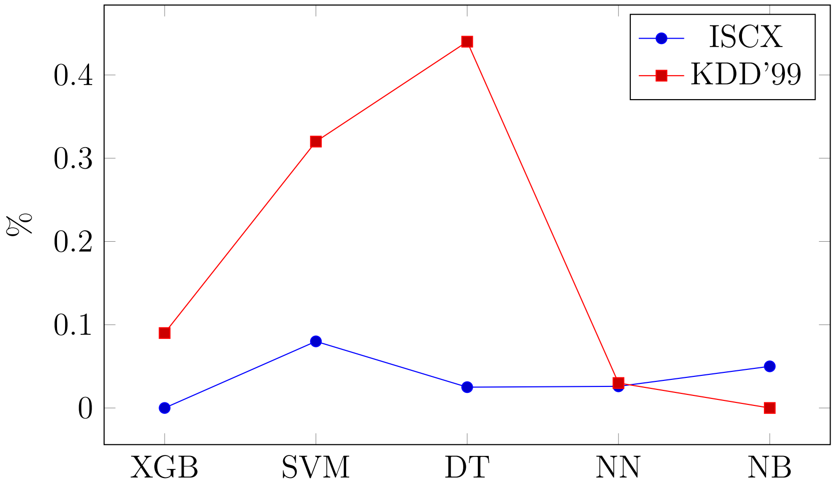

Figure 4.3 False alarm rates of the machine learning algorithms on the ISCX and KDD'99 data sets

The best classification performance is achieved with the XGBoost algorithm, followed by the Decision Tree and SVM algorithms.

While the Neural Network and Naive Bayes algorithms have a smaller false alarm rate, their detection rate (recall) is worse than the rest of the algorithms and they also tend to yield lower accuracies. This is likely due to the minimization of the training set, which significantly limits the unique data points used in training.

For intrusion detection, many organizations choose to decrease false negatives at the expense of increasing false positives (Scarfone et al., 2007). A false negative means an actual attack has been missed by the IDS, which can have serious consequences. On the other hand, while a false positive might be inconvenient, it is less likely to cause a significant issue for the users. As a result, a high detection rate is usually favored over a low false alarm rate in IDS applications.

4.2.2 Running Times

The single-threaded running time comparisons of the classifiers on the ISCX and the KDD data sets are shown in Figure 4.4 and Figure 4.5, respectively. The timings were taken on a machine with an Intel i9-7900X CPU (10 cores, 3.30 GHz, and 32GB RAM). As the comparison of the data sets clearly shows, the fastest algorithm is SVM and the slowest one is XGBoost. The presented running times are for the entire test data sets, namely 4056 test instances for the ISCX data set and 4517 test instances for the KDD'99 data set. The detailed running times (in seconds) can be found in Appendix A.

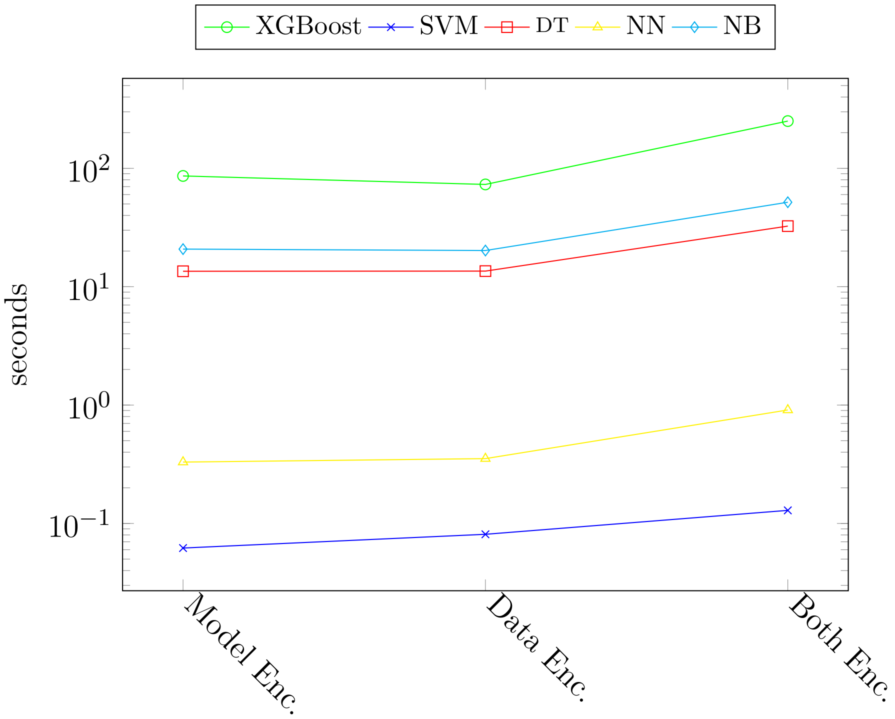

Figure 4.4 Single-threaded timings of ISCX data set

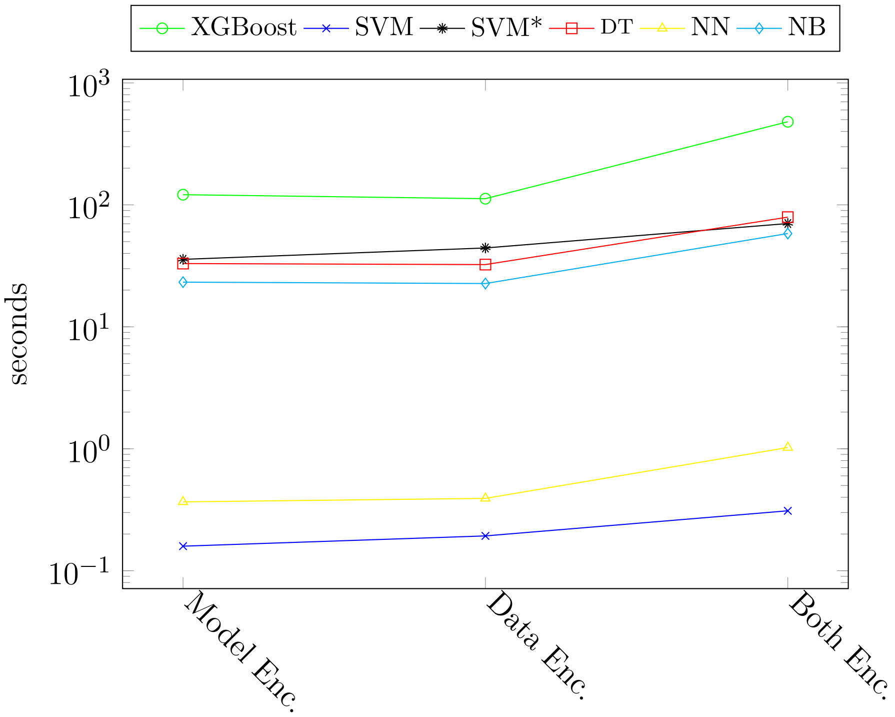

Figure 4.5 Single-threaded timings of KDD’99 data set

The multi-threaded timing results (using all 20 threads on 10 CPU cores of the underlying computing platform) of the ISCX and KDD’99 data sets are given in

Figure 4.6 and Figure 4.7 respectively. The parallelization has been applied for the encode/encrypt, evaluation, and decryption processes. Key generation is achieved with a single thread for all evaluations.

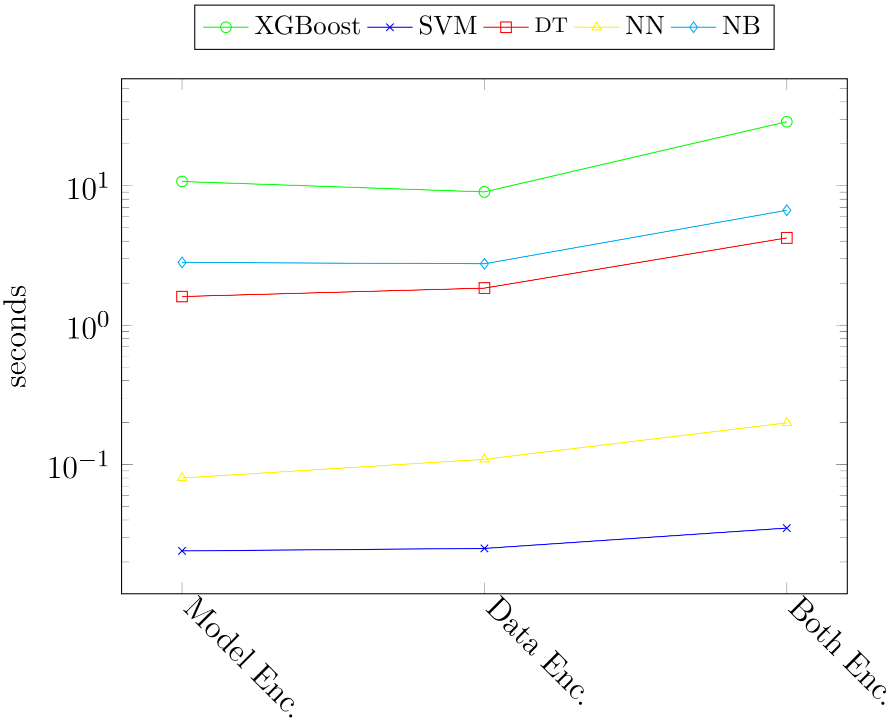

Figure 4.6 Multi-threaded timings of ISCX data det

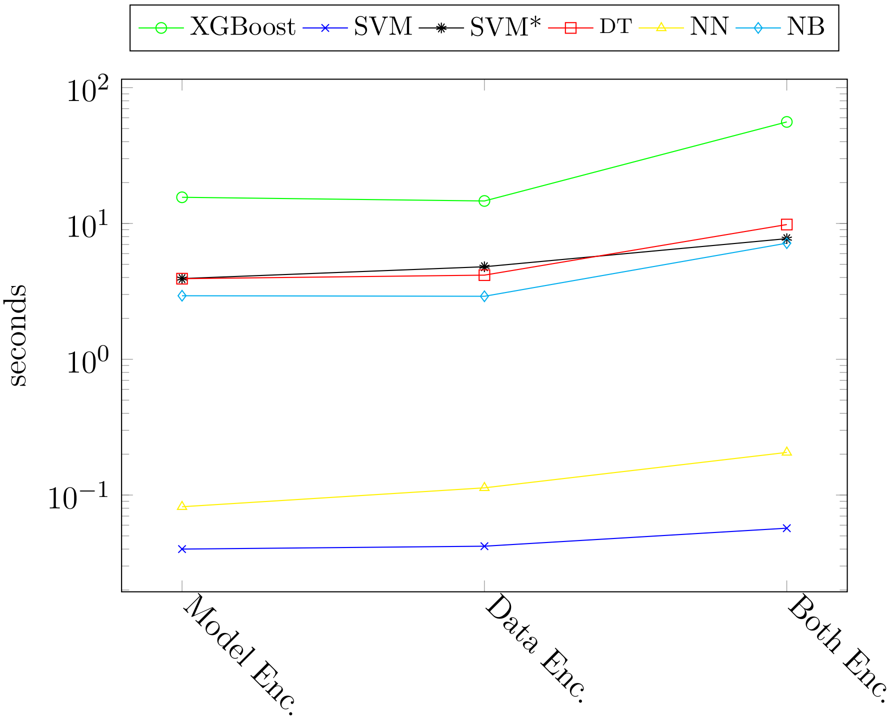

Figure 4.7 Multi-threaded timings of KDD’99 data set

In the following, we will provide more insights on the running times of different algorithms.

4.2.2.1 XGBoost Timing

The speed of XGBoost classification depends on the following elements:

- The number of trees in the XGBoost model
- The number of features
- The number of feature values
- The depth of the trees

For ISCX, the number of trees is selected to be 64, the number of bits used to represent the feature values, t, is 16, and the number of features is 10. On the other hand, for the KDD'99 data set, the number of trees is 128, the t is 8, and the number of features is 20.

We can see that the fastest encryption scenario is DE, which is when the user data is encrypted. On the other hand, the slowest scenario is BE, when both the data and the model are encrypted.

The BE version is significantly slower since both the input and the mask are encrypted during the masking operation. However, in the ME and DE versions, one of the operands in the homomorphic multiplication operation is a plaintext while the other one is ciphertext. The ciphertext-ciphertext multiplication followed by the relinearization operation leads to this overhead in the BE scenario.

We also observe a running time difference between the DE and ME versions. While the encoding and encrypting time of the DE version is higher than the ME version because of the large number of user data to be encrypted, the calculation time of ME is significantly higher than that of the DE scenario. The masking process causes this difference in the running times.

When the model is encrypted (the ME version), and the user is evaluating their data on the encrypted model, they do not know which features are used in the model. Hence, they multiply each feature value with its corresponding mask and add them to achieve the final result.

However, on the DE version, the evaluator (in this case, MO) knows which feature values are used in the model. Hence, MO does not perform the masking (multiplication) operation for the unused features. Thus, in the DE version, the number of masking multiplications drops significantly, decreasing the total execution time.

4.2.2.2 SVM Timing

The speed of the SVM algorithm depends on the number of features.

SVM is the fastest one amongst the classifiers. Since all inputs are encrypted, we observe that the slowest encryption scenario for SVM is also the BE scenario. However, this overhead is much lower than that of the other algorithms, making it almost negligible.

4.2.2.3 Rule-based Classifier Timing

The speed of the rule-based classifiers is directly proportional to the following factors:

- The number of rule signatures
- The number of features
- The number of feature values

The BE version's running times in the rule-based classifiers are also higher than the other versions. Since both parties share their encrypted data, the number of homomorphic operations, where both operands are ciphertexts, increases. As those operations take more time than plaintext-ciphertext operations, the noise budget must also be increased to accommodate this. The polynomial modulus that specifies the ring size R is set to 2^{13}, while for the DE and ME, it is set to 2^{12}.

5. SECURITY ANALYSIS

In this chapter, we go over the security of the BFV encryption scheme, followed by a security discussion on the implementations of proposed classifiers using different encryption scenarios. Furthermore, model extraction techniques are evaluated to observe how much information about the model can be learned by a malicious DO.

5.1 Security of the BFV Encryption Scheme

The BFV scheme provides IND-CPA (semantic) security based on the hardness assumption of the Ring Learning with Errors (RLWE) problem (Fan & Vercauteren, 2012). A system is said to be IND-CPA if an adversary cannot win the following game with a probability greater than 1/2 (Bellare & Rogaway, 2005):

- The challenger generates a public key pk and a secret key sk. The pk is shared with the adversary.
- The adversary generates two plaintext messages m_0 and m_1, and sends them to the challenger.
- The challenger picks a random integer $b \in \{0,1\}$, and sends the encryption of m_b to the adversary.
- The adversary is challenged to guess the value of b given the encryption of m_b. Note that since the adversary has access to pk, they can send queries to the challenge oracle and receive corresponding ciphertexts.

Two hard problems can be given for RLWE, these are *search* and *decision* problems. RLWE problems are known to be as hard as certain lattice-based problems, and they are quantum resistant (Lyubashevsky, Peikert & Regev, 2013). The decision RLWE problem can be defined as follows (Fan & Vercauteren, 2012):

Definition 1 (Decision RLWE)**.** Let $\mathcal{R}_q = \mathbb{Z}_q[x]/\phi(x)$, where $\phi(x)$ stands for a cyclotomic polynomial of degree n specified by the security parameter λ (in SEAL, $\phi(x)$ is selected as $x^n + 1$). Given $a, b \in \mathcal{R}_q$ (a is selected uniformly at random) and a distribution χ over $\mathcal{R}_q$, the decision RLWE problem is determining whether b is chosen uniformly at random from $\mathcal{R}_q$, or $b = a \cdot s + e$, where $s \in \mathcal{R}_q$ and $e \in \chi$.

As a result, each separate encryption of the same plaintext leads to a different element in the ciphertext space, which cannot be distinguished from a uniformly random element that exists in the same space.

5.2 Security of the Proposed Classifiers

In this thesis, we adopt the honest-but-curious assumption for both DO and MO, whereby they follow the protocol steps; although curious about the other party's private input, they do nothing malicious to force the leakage of the other party's input. Each of the three encryption scenarios might come with potential security issues when one of the interacting parties, MO or DO or both, acts maliciously and tries to learn each other's input by feeding malicious input to the other party. This chapter provides a preliminary security analysis for the classification algorithms. The obvious security drawbacks of the classifiers are examined, and several attack scenarios are introduced with possible solutions for remediation. But, apparently, the subject calls for an independent, in-depth study both for security analysis and effective and efficient countermeasures.

In the ME scenario, MO may send a malicious model to the DO, which is carefully crafted to gain information about DO's private input. Then, DO performs the computations homomorphically and sends the result back to MO. DO has no means to know that the model is malicious as it is encrypted with MO's public key. In this case, once MO decrypts the ciphertext, they might learn information about DO's input.

In the DE scenario, a malicious DO may encrypt and send a fake input to discover information about the model or form a new model with matching accuracy. MO will perform the computations homomorphically, and the result will be sent back to DO.

In the BE scenario, if the result is sent to DO, the security analysis becomes DE-based. Otherwise, if the result is sent to MO, the security analysis becomes the ME-based scenario.

In this chapter, we will provide a preliminary discussion about the privacy concerns of three different classifiers in three different encryption scenarios.

5.2.1 XGBoost Classifier

This section discusses the security of the XGBoost classifier under different encryption scenarios.

5.2.1.1 MO discovering DO's data

In the ME scenario, the MO encrypts the model and sends it to the DO to perform the evaluation. Once the DO evaluates their data on the model, they send the encrypted result back to the MO for decryption. Similarly, in the BE scenario, the MO encrypts the model and sends it to the evaluating third party. The third party later sends the evaluation result to the MO. In this case, an MO of malicious intent might send a false model to the DO (or the third party), and the result from this model might reveal information about the DO's data to the MO.

The MO might aim to learn the feature values of the DO's input. In the following example, we outline a simple attack by the DO to recover the value of a feature of DO data.

Example 4. Suppose that the malicious MO wants to learn the value of the feature f_1. For example, for the ISCX data set, each feature takes at most 24 values. We can, therefore, further suppose f_1 takes at most 24 values; i.e., $f_1 \in \{1, 2, \ldots, 24\}$. Recall that the predicates in the XGBoost trees are less-than comparisons. Suppose also that the implementation uses a 20-bit plaintext modulus, thus the final score cannot be larger than $2^{20} - 1$. Then, the malicious MO might construct a model similar to the one in Figure 5.1, where we depict the first two trees, in which the boxes representing leaves contain the scores used in XGBoost trees.

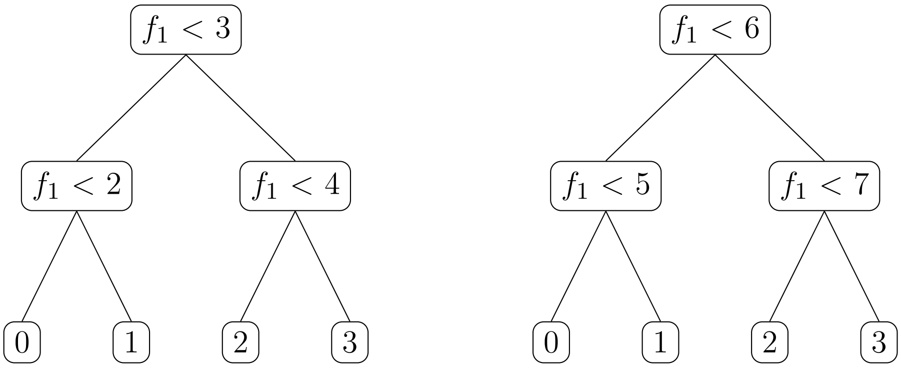

Figure 5.1 The first two trees of a malicious XGBoost model to extract f_1's value

If we construct the first eight trees in this way, where the boxes representing the leaf nodes include the corresponding scores[1], and set the leaf scores of the remaining trees to 0, then the malicious MO can recover the value of f_1.

Note that in the attack scenario, the least significant five bits of the score are used to recover the value of f_1, which can be at most 24. As we have a total of 20 bits in the plaintext modulus, we can use the other bits to encode scores for other features in the next XGBoost trees[2]. This way, we can recover the value of four features using 32 XGBoost trees.

Similarly, for the KDD'99 data set, the features can have 11 values at most. Hence, The first four trees of an XBoost model with a similar scoring method will suffice to learn a feature's value. As the malicious MO uses 4 bits to encode leaf scores for KDD'99 features, this method can be used to recover the values of up to five features by MO using 20 XGBoost trees.

It is difficult to ensure the security of the protocol in the presence of a malicious MO, which is feeding a crafted model to the protocol. DO needs a proof that what they receive as an encrypted model is not malicious. As DO does not hold the decryption key, however, it is not possible for them to check the authenticity of the model.

A simple remediation, perhaps, for DO, would be using a set of previously generated input/label pairs, which were made public by MO to promote the effectiveness of its model for classification. Therefore, we assume that before the evaluation process, the MO and DO agree on a set of open network data that are made known to be malicious or benign. At each evaluation, the DO chooses one of these public network data along with their own data to be evaluated. Once the MO shares the

[1] Note that the scores of the 8th tree will be 0, 1, 2, 0. Its leaf l_4 will never be reached since f_1 can never be bigger than 24, hence it is set to 0.

[2] The leaf scores for the next eight trees will be, then, 0, 32, 33, and 34.

model with DO/third-party, DO/third-party evaluates both these signatures and sends the results back to the MO. The MO, which cannot distinguish between the two results, is expected to share the classification results for both data items with the DO. In this case, if the MO sends a malicious model for evaluation, they will not get a classification result but the values for specific features in the ciphertext. Assuming that MO will not be able to correctly re-classify (or recognize from the recovered features) a public network data only given the leaked features, with a certain probability, the malicious MO may not be able to classify the public network data correctly. Since the DO compares the result of the public network data with its pre-determined label, they will be able to tell that the model is corrupted.

The effectiveness of this countermeasure is subject to further analysis as MO can recognize the published network data via recovered features. To prevent this, DO may randomize the scores without changing their sign as it is the indicator of the class of data. This can also be effective for protecting DO's features as well. But, this may require using a larger plaintext modulus, and one needs to be careful as a larger plaintext modulus and extra homomorphic multiplication result in performance degradation and other security concerns.

In order to test this approach, the countermeasure was applied to the ME version, which will be referred to as the "secure implementation". A random integer r is multiplied by the score to randomize it. To make sure that the sign is not changed after masking; if the score s is a positive number, then $0 < r \times s < p/2$ should hold; otherwise, $p/2 \leq r \times s < p$ should hold (p stands for the plaintext modulus). The default 20-bit plaintext modulus value provided by SEAL is used in this experiment. In the original implementation, the model generated for the ISCX data set contains 64 trees. However, as the number of trees increases, the upper bound for the absolute value of the score also increases. Similarly, the range of leaf scores affects the upper bound for the score value as well. Hence, the highest upper bound S is equal to $10^{l-1} \times m$, where l stands for the highest digit number of the leaf scores, and m stands for the number of trees. In order to limit the score value and increase the range of r (to increase the security this countermeasure provides), a new downsized model that consists of 16 trees is generated for the ISCX data set. The range of leaf scores (3 digits at most) was left the same as the original version. As a result of this, r can be equal to any integer in the range $[1, 32]$. The new downsized model's (16 trees) evaluation time is compared using the original and the secure implementations in Table 5.1 to observe the timing overhead this countermeasure poses. Additionally, the classification performances of the original (64 trees) and downsized (16 trees) models are compared in Table 5.2.

Implementation	**Original version**	**Secure version**
Single-thread	4785	4891
Multi-thread	73	74

Table 5.1 Evaluation time comparison of the original and secure XGBoost implementations (the results are given in ms, the downsized model is used for all evaluations)

Metrics	**Original model**	**Downsized model**
Accuracy	99.95%	99.70%
Recall	98.91%	93.48%
Precision	100%	100%
FAR	0%	0%

Table 5.2 Classification performance comparison of the original and downsized XGBoost models

Another countermeasure can be adding an extra protocol step, where the score, which is homomorphically masked by DO, is sent back to DO for unmasking. The masking can be performed using a modular addition by a random integer $r \in [0, p)$, where p is the plaintext modulus. In this method, there is no need for a larger plaintext modulus, and it is much more efficient and applicable as only one homomorphic addition operation per query is sufficient for masking. No overhead is observed as a result of this masking operation, as it replaces the masking operation for the intermediary values that is used in the original version. In the original version, the masking values for the final scores are set to 0, and the masking values for the intermediary values are selected as random integers $r \in [0, p)$. Instead of only masking the intermediary values, we also mask the final scores with this approach. Hence, the total number of homomorphic operations does not change. The security analysis of this solution is then reduced to that of the scenario where data is encrypted, which is treated next.

5.2.1.2 DO discovering the model

In the DE scenario, where the DO encrypts their data and shares it with the MO for evaluation, the DO might try to send all possible feature value combinations to reconstruct the model. While the evaluating party, MO, masks the intermediary calculation values, the resulting classification score will be known to DO, who owns the decryption key.

The data points for the ISCX data set consist of 10 features, and each feature has at most 24 categories. However, assuming that all features have 24 categories, the DO's signature might be one of $24^{10} \approx 2^{45.85}$ possibilities. Suppose the DO can evaluate all the combinations on MO's model and receive the classification result. In that case, they will be able to construct a faux model that can completely mimic the classification of the original one. Furthermore, since the DO can see the classification scores based on the changes in the overall score, the DO might be able to guess the leaf values and how specific feature values affect the overall score.

For the KDD'99 data set, the data points contain 20 features, each with at most 11 categories. Therefore assuming that all features have 11 categories, the number of total signature combinations is $11^{20} \approx 2^{69.19}$. As a result, working with a high number of features might increase the overall security as the number of combinations to evaluate on the model increases.

The scenario that requires DO to evaluate a prohibitively high volume of data using the model owned by MO seems unrealistic. MO will probably not provide this service to any DO this long. However, the model can be reconstructed with a fewer number of data items. For this, DO should have the expertise to construct a state-of-the-art model. Also, their data should contain sufficiently diverse feature values with correct labels to construct an effective model. We presume these conditions do not hold for typical DO, which lacks the expertise and the data.

A malicious DO can prepare specially crafted data to infer information about the XGBoost trees. Since that final score is the sum of the scores of all trees and where exactly (which tree and which branch) each feature value is used is not known, we assume DO can learn only a limited amount of information about the model, which may not be useful. The security concerns and what a malicious DO can learn about the model also depend on the data set to a large extent. Therefore, we recommend MO should investigate attack methods based on their models, and the data set they used for training before providing prediction services to DO.

5.2.2 SVM Classifier

The SVM model consists of a weight value w_i for each feature and an offset value b. The classification score of the algorithm R is obtained once the ciphertext is decrypted.

In the ME version, MO will be able to learn DO's input if they can solve for Equation 5.1. Since DO is expected to evaluate a different signature at each run, and the model is expected to stay the same, MO cannot learn more information about the user input with multiple evaluations.

$$(w_1 \times x_1) + (w_2 \times x_2) + \cdots + (w_d \times x_d) = R - b \tag{5.1}$$

Similar to the proposed attack scenario for the XGBoost model, MO can still learn a number of feature values from a single input. For the ISCX data set, the DO's feature values can take up to 24 values. Hence, a feature value is represented using 5 bits. In this case, MO can pick four of the weight values w as $1, 2^5, 2^{10}$ and 2^{15}, since the plaintext modulus is 20 bits. The DO's feature value will be shifted accordingly as a result of the multiplications. The rest of the weight values will be set to 0. As a result, the values of four specified features will be stored in the resulting ciphertext, where every 5 bits will represent the value of a single feature.

On the other hand, the entire model can easily be reconstructed by a malicious DO in the DE version. There, if DO wants to learn about the model, they have to discover the offset value b before solving for Equation 5.1. First, they can learn the offset value by sending an encrypted array $(0)^{1\times d}$ to the model owner for evaluation. The decryption of the resulting ciphertext will contain the offset value. MO is expected to receive multiple inputs from DO since an excessive number of network signatures are expected to be evaluated using the same model. A malicious DO might then send the following inputs of length d for evaluation to learn about the weights of the model:

$$\begin{array}{rl}
\texttt{Input 1:} & (1, 0, 0, \ldots, 0) \\
\texttt{Input 2:} & (0, 1, 0, \ldots, 0) \\
 & \ldots \\
\texttt{Input d:} & (0, 0, 0, \ldots, 1).
\end{array}$$

After $d+1$ iterations, the malicious user can learn everything about the model, which makes the current DE not suitable for privacy preservation.

Similar to one of the countermeasures taken for the XGBoost classifier, in both SVM scenarios, the resulting score can be homomorphically multiplied with a random integer r to mask the score. The crucial information for classification is the sign of the score. Hence, the evaluating party must make sure that the sign of the result is not changed by the masking. In order to provide a sufficient level of security, a large plaintext modulus might be required. For instance, suppose we know an upper bound for the absolute value of the score; i.e., $0 < |score| < S$. Then, if the random number $r > 0$ and $rS < p/2$, where p is the plaintext modulus, then the sign of the score will not be affected. On the other hand, even after multiplication, we cannot guarantee the resulting integer is uniformly distributed in the ranges $[0, p/2)$ for positive scores and $[p/2, p)$ for negative scores. Therefore, this countermeasure still needs further analysis to find out how much security it provides.

For our original SVM classifier and preprocessed data sets, it is not guaranteed that we will find a range for the random number r that meets the condition specified above. For example, in the ISCX data set, a feature can take 24 different values, and the data set contains 10 features. The normal vector elements w_i are signed integers that are at most 5 digits (Note that the number of digits can be adjusted since originally w_i are decimal numbers which are multiplied with a power of 10 to approximate an integer value for homomorphic evaluation). As a result, the upper bound for the absolute value of the score S would be equal to $24 \times (10^5 - 1) \times 10 = 23999760$, which by itself overly exceeds the 20-bit plaintext modulus we have specified. However, in reality, it is highly unlikely to achieve such a high final score. This is because the sign of each normal vector element w_i varies (products of the elements cancel each other out), and each w_i usually contains 3 to 4 digits in the generated models, causing the final score to fall in the ranges of the specified plaintext modulus. In the ISCX data set, the final scores are observed to be usually 5-digit numbers, which are well within the ranges of the selected 20-bit plaintext modulus. Nevertheless, this ambiguity prevents us from defining a range for the random number r, whose multiplication with the score is expected to not alter the sign value.

However, the normal vector elements, number of features, and number of bins could be minimized to ensure that rS does not exceed the ranges specified by the plaintext modulus. Since the downsizing of data results in information loss, the classification performance of the algorithm might suffer as a consequence of this. Nevertheless, to test the performance of this remediation approach, the ISCX data set was resized to contain 7 features in total, each containing 4 values. Furthermore, the normal vector elements w_i are approximated to be 2-digit integers at most. This minimization causes the highest possible score $|S|$ to be equal to 2772. Based on the 20-bit plaintext modulus we use in our implementation, r can be any integer in the range $[1, 186]$. The downsized data set is evaluated using the original implementation as well as the proposed secure implementation to observe their evaluation times and the overhead the secure version poses. The DE version is used for the timing comparison, which is given in Table 5.3. Moreover, the classification performances for the original and downsized data sets are shown in Table 5.4.

Implementation	**Original version**	**Secure version**
Single-thread	22	25
Multi-thread	10	13

Table 5.3 Evaluation time comparison of the original and secure SVM implementations (the results are given in ms)

Metrics	**Original version**	**Secure version**
Accuracy	99.75%	99.53%
Recall	96.19%	89.67%
Precision	98.33%	100%
FAR	0.08%	0%

Table 5.4 Classification performance comparison of the SVM models generated using the original and downsized ISCX data sets

Even though ME and DE approaches are both open to possible attacks, the ME version is observed to be more secure as only a certain number of DO's feature values could be extracted by a malicious MO. On the other hand, the DE version is much more vulnerable, putting the entire model at risk of being discovered by a malicious DO. As a result, the extra protocol countermeasure used for the XGBoost classifier could also be applied to the SVM classifier, so that the DE-based scenario could be converted into the ME-based scenario. The MO sums the final score S with a random integer $r \in [0, p)$ before sending it to the DO for decryption. The DO sends the decrypted result to the MO, for MO to subtract r from the masked score, and warn the DO if the score indicates an intrusion. As was the case for the XGBoost classifier, this masking operation does not cause a timing overhead for the homomorphic evaluation process. With this countermeasure, the existing addition operation used for intermediary value masking (in the original version) is used to mask the final scores as well. Only now, the slots that contain the actual score are also summed with a random integer r, instead of 0.

5.2.3 Rule-based Classifiers

In the rule-based classifiers, if the record signature and a rule signature match, then the score for that rule signature will be equal to 0. Otherwise, it will be equal to the sum of the indices of given in Formula 3.5, multiplied by a random integer $r \in [0, p)$. The index that stores the results of the comparisons is also picked randomly.

As a result, in the DE version, once DO decrypts the result, they will be able to see how many rules their input matched. However, since the order of the results is shuffled at each iteration, they will not be able to make a connection between their input and a specific rule. Since the only information DO can discover is the number of rules matched per record signature, it becomes difficult for them to predict the model's behavior.

In the ME version, since MO also cannot differentiate between the results from different rules, they can employ a different technique to differentiate the actual rules of the matching signatures. Malicious MO can send different instances of each unique rule signature to DO as if they were signatures of different rules. Each rule signature, in fact, might be checking for a different value of the same feature, and the values for the other features might be set to *don't care* (represented with *). For example, the MO wants to learn the input's value for feature f_i, for possible values $v_1, v_2, \ldots, v_t$ the feature takes. Then, they can generate the following rules as many times as the number of copies specified:

For v_1: 1, 0, . . . , 0 | *, *, . . . , * | . . . | *, *, . . . , * (1 copy)
For v_2: 0, 1, . . . , 0 | *, *, . . . , * | . . . | *, *, . . . , * (2 copies)
. . .
For v_t: 0, 0, . . . , 1 | *, *, . . . , * | . . . | *, *, . . . , * (t copies)

As a result, the number of 0's in the result array will tell which rule matched the DO's record. For example, if there were two 0's in the result array, that would mean the feature f_i takes the value v_2 in the DO's input. This is because the rules for other features will result in a non-zero value as they do not match the record signature. The record signature is expected to match only one unique rule signature for a single feature value. Hence, only one unique rule (and its copies) should give 0 as a result. Since each feature rule has a different number of copies, the number of zeroes is expected to tell which value of the feature the record signature matches.

However, compared to the shortcomings in specific versions of the other algorithms, the rule-based ones tend to leak the least amount of information. For example, in the ISCX data set, the malicious MO has to generate $24 \times 25/2 = 300$ rules with the proposed attack scenario to discover one feature's value. It would become infeasible to discover more feature values with a single evaluation. Additionally, as the number of feature values increases, the number of rules significantly increases as well, making evaluation more difficult.

5.3 Black-Box Attacks for Model Extraction

The models used for intrusion detection can be discovered to a certain extent by a malicious DO by using black-box attacks. This attack method, which is applied in the thesis of Mağara (2022) on XGBoost classifiers, is tested for the XGBoost and SVM models that are proposed in this thesis. The malicious party sends several queries to the victim model and gets the classification result for each query. Based on the response of the victim model, the malicious DO might try to steal and construct a new model that can mimic the behaviour of the original one.

Two different model extraction techniques are used in this thesis, namely the CNN and naive approaches:

- The CNN approach uses Python's Adversarial Robustness Toolbox (ART) v1.11 library for extraction attacks (Nicolae, Sinn, Tran, Buesser, Rawat, Wistuba, Zantedeschi, Baracaldo, Chen, Ludwig, Molloy & Edwards, 2018). The extraction methods evaluated by this library are Copycat CNN and KnockoffNets techniques, which generate CNN models that copy the behaviour of the victim model (Correia-Silva, Berriel, Badue, de Souza & Oliveira-Santos, 2018; Orekondy, Schiele & Fritz, 2019). These model extraction techniques are originally used to extract black-box CNN models. However in this thesis, the XGBoost and SVM models are queried to generate a stolen CNN model.
- In the naive approach, the malicious DO evaluates their data on the victim XGBoost/SVM model, and gets the corresponding classification result for each query. Using this data and the its classification results, a new XGBoost/SVM model is trained.

However, one of the limitations to such an attack scenario is the lack of real data a malicious DO can possess. The real data set owned by the malicious party alone might not be sufficiently abundant, diverse and representative to extract adequate information from the victim model to construct a new model as good as the victim model. To estimate the victim model to a better extent and cover possible edge-cases, synthetic data generation can be used. In this thesis, the CNN and naive approaches are evaluated using two different extraction data sets. In the first approach, the malicious DO uses only real data to extract the victim model. In the second approach, the malicious DO uses all the real data they own to generate a synthetic data set, and uses this data to extract the victim model. In this thesis, Python's DataSynthesizer library is used for synthetic data generation (Ping, Stoyanovich & Howe, 2017). The synthetic data generation is achieved using the *random mode*, which generates random values for each feature, taking into account the range and type (which is categorical in our case) of the values for the given feature.

The proposed attack scenarios are realized and tested using the XGBoost and SVM models that are generated for the ISCX data set, which consists of an original test set of size 4056. A part of the original test set is used for extraction purposes, while the remaining part is used to test the attack scenarios. The experiment settings for different extraction data sets are as specified below:

- **Real Data Only:** A subset of the original test set is used as the extraction data set (also referred to as the real data owned by the malicious DO). Once the extraction data set is removed, the rest of the original test set is used for testing purposes. The performance of the stolen model is observed using extraction sets of different sizes; i.e., $\gamma \in \{125, 250, 500, 1000, 2000, 2500\}$.

- **Synthetic Data Only:** A subset of the original test set (generation set) is used to generate a synthetic data set of size 10000. This synthetic data is used as the extraction data set. Once the generation set is removed from the original test set, the remaining queries are used for testing purposes. The classification performance of the stolen model is observed for different generation set sizes ($\gamma \in [100, 200, 300, 400, 500]$). It is difficult for a DO with no prior knowledge of the victim model to obtain any real data, and synthetic data generation is used when the number of real data is not sufficient to learn the victim model. As a result, in this approach, the size of the real data set γ is limited to be 500 or less.

To assess the performance of the stolen models, we use the four metrics which are also used to assess the performance of the victim models: Accuracy, Precision, Recall, and FAR.

5.3.1 XGBoost Classifier: Model Extraction Attack Results

Figure 5.2 shows the classification performance of the CNN extraction method for the XGBoost model using real data only. In the *Argmax* approach, the class with the highest predicted probability is selected as the label of the query and then used to train the stolen model. In the *Probabilistic* approach, the predicted class probabilities of the query are used to train the stolen model. The *Original Classifier* on the figures refers to the victim classifier.

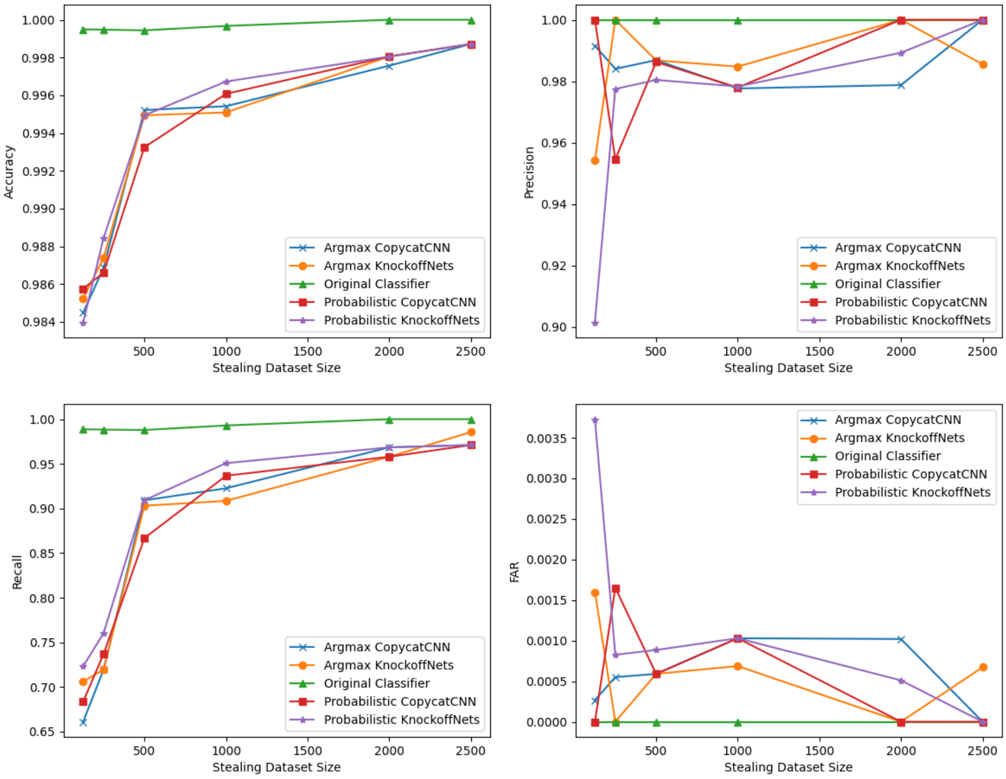

Figure 5.2 XGBoost model extraction using real data only (Stolen model is CNN).

Figure 5.3 shows the classification performance of the naive extraction method for the XGBoost model using only real data.

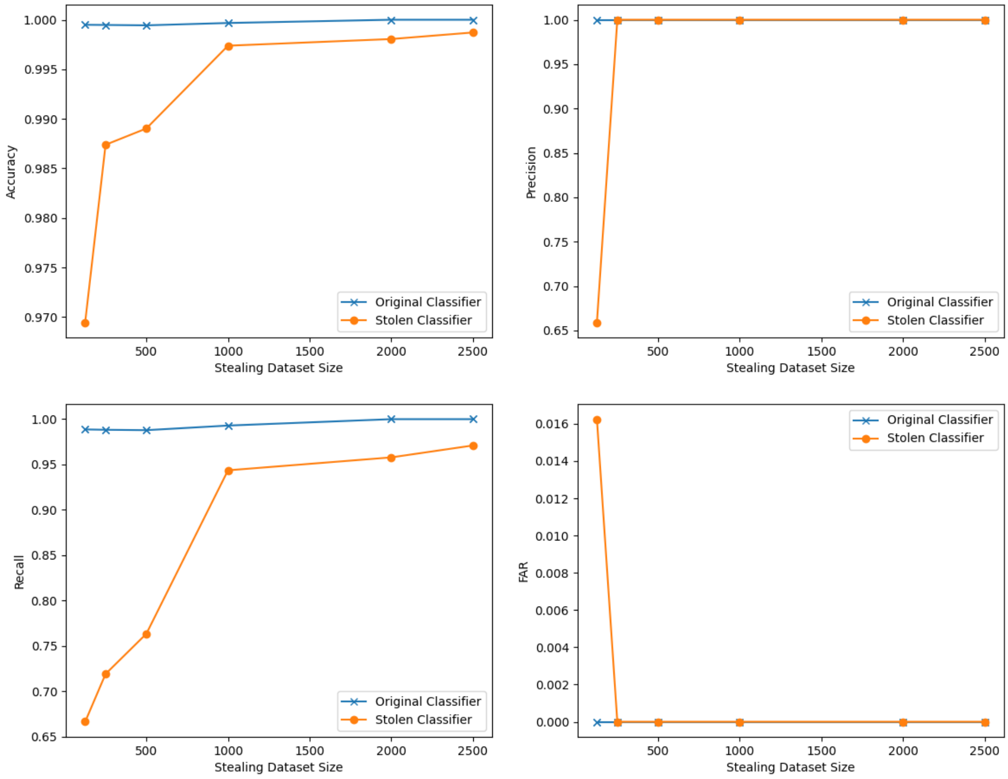

Figure 5.3 XGBoost model extraction using real data only (Stolen model is XGBoost).

Figure 5.4 shows the classification performance of the CNN extraction method for the XGBoost model using only synthetic data.

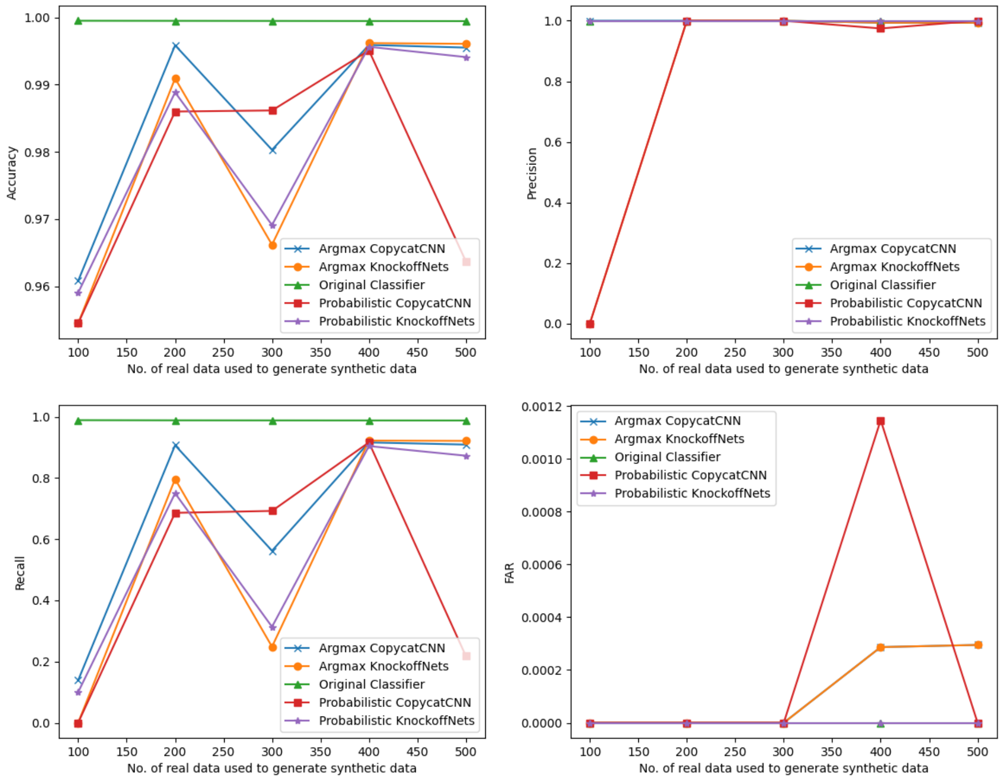

Figure 5.4 XGBoost model extraction using synthetic data only (Stolen model is CNN).

Figure 5.5 shows the classification performance of the naive extraction method for the XGBoost model using only synthetic data.

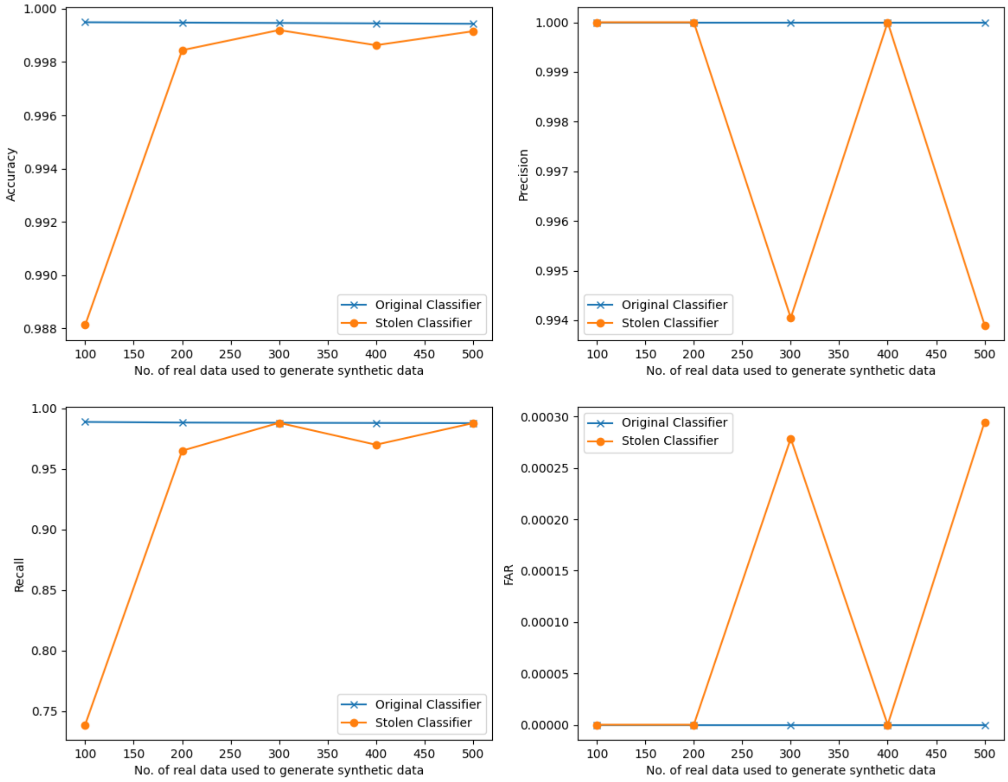

Figure 5.5 XGBoost model extraction using synthetic data only (Stolen model is XGBoost).

It was observed that the classification performance of the stolen models falls short compared to that of the victim XGBoost model provided by the MO. Additionally, as the number of real data owned by the malicious DO increases, the classification performance tends to improve. However, in terms of classification performance, synthetic data does not appear to provide a significant advantage over real data.

5.3.2 SVM: Model Extraction Attack Results

Figure 5.6 shows the classification performance of the CNN extraction method for the SVM model using only real data.

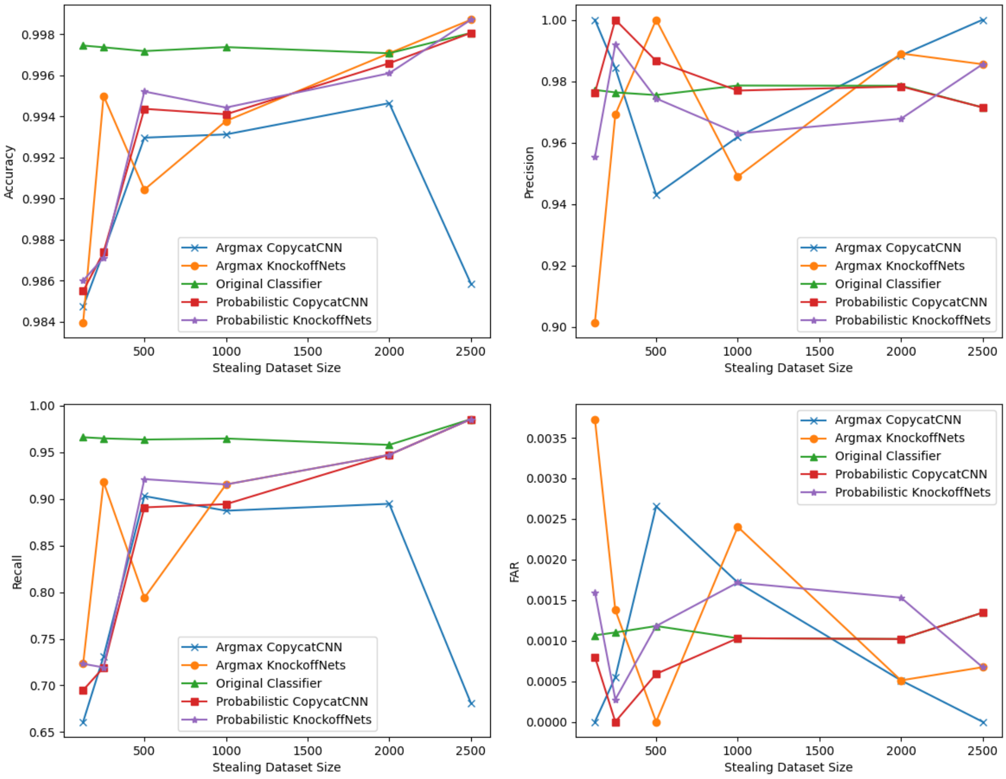

Figure 5.6 SVM model extraction using real data only (Stolen model is CNN).

Figure 5.7 shows the classification performance of the naive extraction method for the SVM model using only real data.

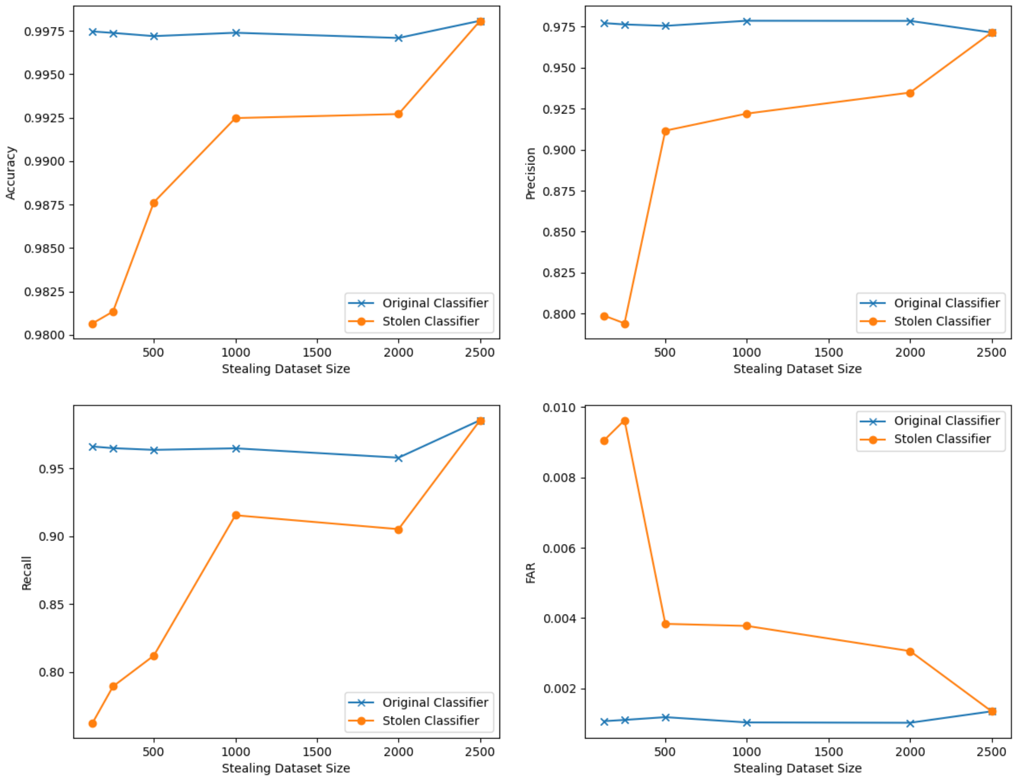

Figure 5.7 SVM model extraction using real data only (Stolen model is SVM).

Figure 5.8 shows the classification performance of the CNN extraction method for the SVM model using only synthetic data.

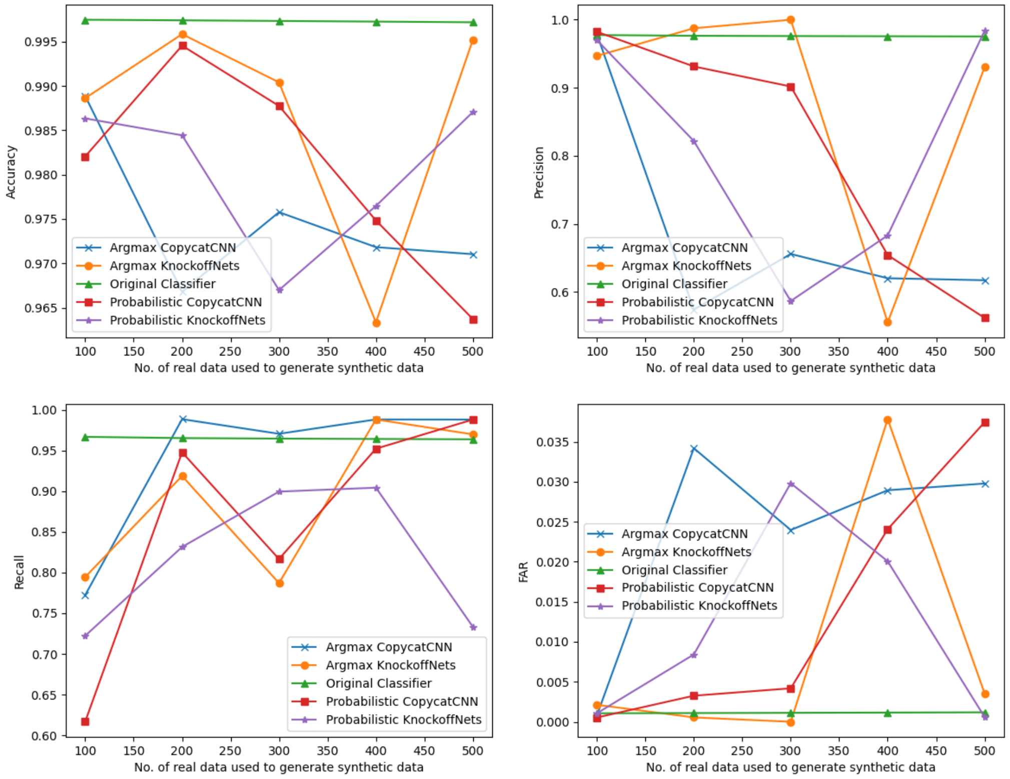

Figure 5.8 SVM model extraction using synthetic data only (Stolen model is CNN).

Figure 5.9 shows the classification performance of the naive extraction method for the SVM model using only synthetic data.

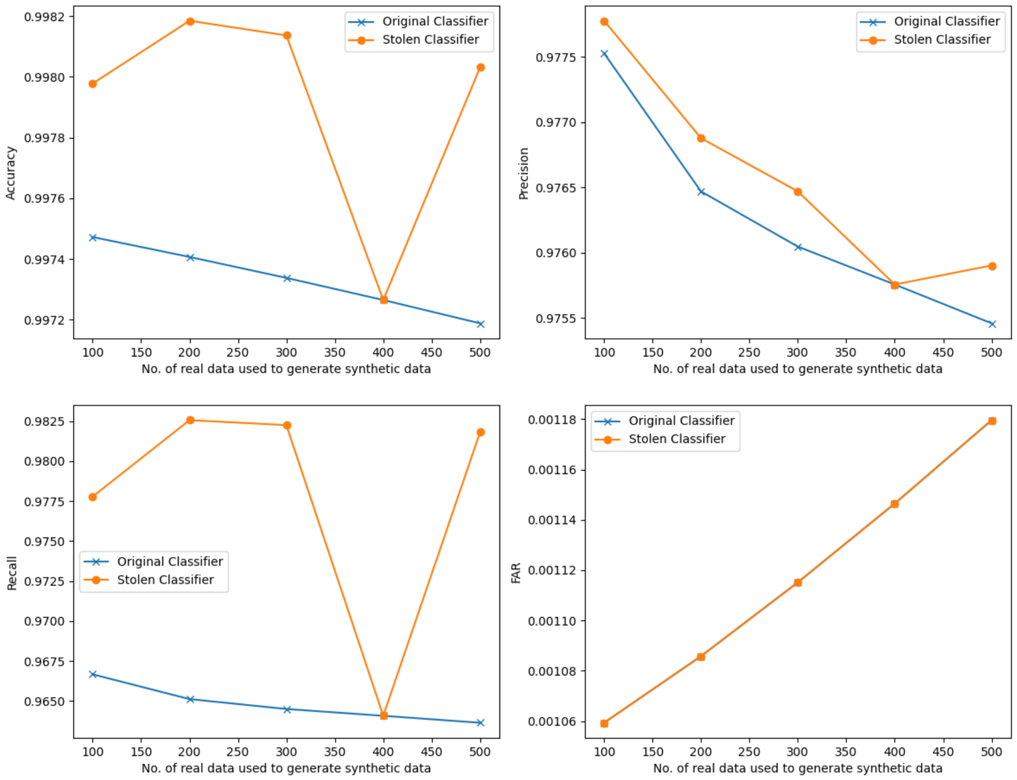

Figure 5.9 SVM model extraction using synthetic data only (Stolen model is SVM).

Overall, it was observed that the classification performance of the stolen models is worse than that of the victim model. However, the stolen model which uses the naive approach with synthetic data (shown in Figure 5.9) overperforms the original model. As a result, the SVM model is observed to be more vulnerable to black-box attacks, compared to the XGBoost model. For the naive approach, the usage of synthetic data is observed to boost the classification performance of the stolen model. However, for the CNN extraction method, the usage of synthetic data does not offer any significant improvements over the classification performance of the stolen model. Additionally, the overall performance of the stolen CNN models tends to be more unpredictable when synthetic data is used for extraction.

6. CONCLUSION

In this thesis, the performances of various machine learning based classifiers were investigated for network intrusion detection when homomorphic encryption technology is used for privacy. Different encryption scenarios are considered for different application needs: i) data encrypted by DO's public key, and homomorphic operation is performed by MO, ii) model encrypted by MO's public key and homomorphic operation is performed by DO, and iii) both data and model encrypted by either DO's or MO's public keys and homomorphic operation is performed by a third party.

The classification and timing performances of the proposed algorithms were compared, and their security concerns were discussed in the presence of malicious parties. Each proposed classification method has different strengths and weaknesses. Based on the purpose and needs of the intrusion detection system providers and users, one of the proposed methods can be selected for intrusion detection.

The XGBoost classifier proposed in this thesis delivers the best classification performance; however, it works slower than the other algorithms. On the other, the fastest algorithm is SVM. With the SVM classifier, more than a million input data can be classified almost at the same time the Decision Tree classifier can classify four thousand input data items. While its classification performance is worse than that of XGBoost or Decision Tree, SVM may be a preferable classifier for an organization that values fast intrusion detection. On the other hand, if an IDS that generates a low number of false alarms is preferred, the Naìve Bayes or Neural Network classifiers might be preferred.

These classifiers were implemented considering different encryption schemes. Each scenario's security and timing performance based on different classifiers was discussed. The most secure classification approach seems to be the rule-based classifiers, leaking the least amount of information in all encryption scenarios compared to the other classifiers. Hence, rule-based classifiers can be preferred for a highly secure detection system. However, the privacy-preserving properties of the other classifiers can also be significantly improved with the implementation of the proposed remediation techniques. As feature work, the proposed remediation techniques can be first analyzed in more detail and applied to increase the overall security of the algorithms. In addition, more classifiers based on other machine learning algorithms can also be investigated and compared with those discussed in this thesis.

BIBLIOGRAPHY

Alashqur, A. (2015). A novel methodology for constructing rule-based naïve bayesian classifiers. *International Journal of Computer Science & Information Technology*, *7*(1), 139.

Bace, R. & Mell, P. (2001). Nist special publication on intrusion detection systems. Technical report, Booz-allen and Hamilton Inc MCLEAN VA.

Belland, M., Xue, W., Kurdi, M., & Chu, W. (2017). Somewhat homomorphic encryption.

Bellare, M. & Rogaway, P. (2005). Introduction to modern cryptography. *Ucsd Cse*, *207*, 207.

Bost, R., Popa, R. A., Tu, S., & Goldwasser, S. (2014). Machine learning classification over encrypted data. *Cryptology ePrint Archive*.

Brakerski, Z., Gentry, C., & Halevi, S. (2013). Packed ciphertexts in lwe-based homomorphic encryption. In *International Workshop on Public Key Cryptography*, (pp. 1–13). Springer.

Chen, H., Dai, W., Kim, M., & Song, Y. (2019). Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In Cavallaro, L., Kinder, J., Wang, X., & Katz, J. (Eds.), *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, (pp. 395–412). ACM.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, (pp. 785–794).

Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In Takagi, T. & Peyrin, T. (Eds.), *Advances in Cryptology – ASIACRYPT 2017*, (pp. 409–437)., Cham. Springer International Publishing.

Correia-Silva, J. R., Berriel, R. F., Badue, C., de Souza, A. F., & Oliveira-Santos, T. (2018). Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In *2018 International Joint Conference on Neural Networks (IJCNN)*, (pp. 1–8). IEEE.

Cortes, C. & Vapnik, V. (1995). Support-vector networks machine learning vol. 20.

Cup, K. (1999). Dataset. *available at the following website http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html*, *72*.

Deforth, K., Desgroseilliers, M., Gama, N., Georgieva, M., Jetchev, D., & Vuille, M. (2021). Xorboost: Tree boosting in the multiparty computation setting. *Cryptology ePrint Archive*.

Fan, J. & Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, *2012*, 144.

Gentry, C. (2009). *A fully homomorphic encryption scheme*. PhD thesis, Stanford University. `crypto.stanford.edu/craig`.

Goodman, R. M., Higgins, C. M., Miller, J. W., & Smyth, P. (1992). Rule-based neural networks for classification and probability estimation. volume 4, (pp. 781–804). MIT Press.

Higgins, C. & Goodman, R. (1991). Incremental learning with rule-based neural networks.

Holmes, G., Donkin, A., & Witten, I. H. (1994). Weka: A machine learning workbench. In *Proceedings of ANZIIS'94-Australian New Zealnd Intelligent Information Systems Conference*, (pp. 357–361). IEEE.

Hsu, C.-W. & Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, *13*(2), 415–425.

Jiang, L. & Ju, L. (2022). Fhebench: Benchmarking fully homomorphic encryption schemes. *arXiv preprint arXiv:2203.00728*.

Karaçay, L. (2019). *Privacy-Preserving Intrusion Detection over Network Data*. PhD thesis, Sabanci University.

Kent, J. T. (1983). Information gain and a general measure of correlation. *Biometrika*, *70*(1), 163–173.

Kumar, G. (2014). Evaluation metrics for intrusion detection systems-a study. *Evaluation*, *2*(11), 11–7.

Laine, K. (2017). 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.

Laur, S., Lipmaa, H., & Mielikäinen, T. (2006). Cryptographically private support vector machines. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 618–624).

Leung, K. M. (2007). Naive bayesian classifier. *Polytechnic University Department of Computer Science/Finance and Risk Engineering*, *2007*, 123–156.

Lyubashevsky, V., Peikert, C., & Regev, O. (2013). On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, *60*(6), 1–35.

Mağara, S. S. (2022). Privacy-preserving xgboost inference with homomorphic encryption. Master's thesis, Sabanci University. (Unpublished master's thesis).

Meng, X. & Feigenbaum, J. (2020). Privacy-preserving xgboost inference. *arXiv preprint arXiv:2011.04789*.

Morris, L. (2013). Analysis of partially and fully homomorphic encryption. http://gauss.ececs.uc.edu/Courses/c5156/pdf/homo-outline.pdf. [Online; accessed 05-July-2022].

Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I., & Edwards, B. (2018). Adversarial robustness toolbox v1.2.0. *CoRR*, *1807.01069*.

Orekondy, T., Schiele, B., & Fritz, M. (2019). Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, (pp. 4954–4963).

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In Stern, J. (Ed.), *Advances in Cryptology — EUROCRYPT '99*, (pp. 223–238)., Berlin, Heidelberg. Springer Berlin Heidelberg.

PALISADE, . (2020). PALISADE Lattice Cryptography Library (release 1.9.2). https://palisade-crypto.org/.

Park, S., Lee, J., Cheon, J. H., Lee, J., Kim, J., & Byun, J. (2019). Security-preserving support vector machine with fully homomorphic encryption. In *SafeAI@ AAAI*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ping, H., Stoyanovich, J., & Howe, B. (2017). Datasynthesizer: Privacy-preserving synthetic datasets. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, (pp. 1–5).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Scarfone, K., Mell, P., et al. (2007). Guide to intrusion detection and prevention systems (idps). *NIST special publication*, *800*(2007), 94.

SEAL, . (2021). Microsoft SEAL (release 3.7). `https://github.com/Microsoft/SEAL`. Microsoft Research, Redmond, WA.

Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, *31*(3), 357–374.

Smart, N. P. & Vercauteren, F. (2014). Fully homomorphic simd operations. *Designs, codes and cryptography*, *71*(1), 57–81.

Smyth, P. & Goodman, R. M. (1992). An information theoretic approach to rule induction from databases. *IEEE transactions on Knowledge and data engineering*, *4*(4), 301–316.

Tait, K.-A., Khan, J. S., Alqahtani, F., Shah, A. A., Khan, F. A., Rehman, M. U., Boulila, W., & Ahmad, J. (2021). Intrusion detection using machine learning techniques: an experimental comparison. In *2021 International Congress of Advanced Technology and Engineering (ICOTEN)*, (pp. 1–10). IEEE.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, (pp. 1–6). Ieee.

Teo, S. G., Han, S., & Lee, V. C. (2013). Privacy preserving support vector machine using non-linear kernels on hadoop mahout. In *2013 IEEE 16th international conference on computational science and engineering*, (pp. 941–948). IEEE.

Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., & Lin, W.-Y. (2009). Intrusion detection by machine learning: A review. *expert systems with applications*, *36*(10), 11994–12000.

Xu, M., Li, X., Wang, Y., Luo, B., & Guo, J. (2021). Privacy-preserving multisource transfer learning in intrusion detection system. *Transactions on Emerging Telecommunications Technologies*, *32*(5), e3957.

APPENDIX A

Timing Results

The detailed timing results are provided in this section. The results are given in seconds for all the tables included.

ISCX Data Set: Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,150	0,154	0,151
Encode / Encrypt	0,596	14,180	14,232
Evaluation	85,093	58,388	235,859
Decryption	0,509	0,513	0,511
End-to-End	86,348	73,235	250,753

Table A.1 ISCX XGBoost Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,041	0,043	0,042
Encode / Encrypt	0,003	0,02	0,021
Evaluation	0,052	0,055	0,102
Decryption	0,007	0,006	0,006
End-to-End	0,103	0,124	0,171

Table A.2 ISCX SVM Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,040	0,040	0,150
Encode / Encrypt	0,015	0,255	0,292
Evaluation	13,402	13,206	32,067
Decryption	0,084	0,084	0,115
End-to-End	13,541	13,585	32,624

Table A.3 ISCX Decision Tree Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,041	0,039	0,150
Encode / Encrypt	0,005	0,033	0,045
Evaluation	0,315	0,310	0,850
Decryption	0,010	0,010	0,014
End-to-End	0,371	0,392	1,059

Table A.4 ISCX Neural Network Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,04	0,038	0,151
Encode / Encrypt	0,097	0,065	0,307
Evaluation	20,655	20,122	51,353
Decryption	0,022	0,021	0,030
End-to-End	20,814	20,246	51,841

Table A.5 ISCX Naive Bayes Serial Implementation

ISCX Data Set: Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,158	0,153	0,156
Encode / Encrypt	0,101	1,416	1,445
Evaluation	10,577	7,565	27,326
Decryption	0,060	0,061	0,063
End-to-End	10,896	9,195	28,990

Table A.6 ISCX XGBoost Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,044	0,044	0,043
Encode / Encrypt	0,004	0,007	0,011
Evaluation	0,018	0,016	0,023
Decryption	0,002	0,002	0,001
End-to-End	0,068	0,069	0,078

Table A.7 ISCX SVM Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,040	0,039	0,155
Encode / Encrypt	0,016	0,257	0,293
Evaluation	1,498	1,497	3,807
Decryption	0,091	0,090	0,129
End-to-End	1,645	1,883	4,384

Table A.8 ISCX Decision Tree Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,041	0,04	0,154
Encode / Encrypt	0,005	0,034	0,042
Evaluation	0,061	0,060	0,134
Decryption	0,014	0,015	0,023
End-to-End	0,121	0,149	0,353

Table A.9 ISCX Neural Network Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,043	0,040	0,155
Encode / Encrypt	0,098	0,071	0,308
Evaluation	2,691	2,661	6,330
Decryption	0,030	0,025	0,038
End-to-End	2,862	2,797	6,831

Table A.10 ISCX Naive Bayes Parallel Implementation

KDD Cup 1999 Data Set: Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,151	0,152	0,148
Encode / Encrypt	1,283	31,636	31,332
Evaluation	119,320	80,181	448,731
Decryption	0,570	0,573	0,562
End-to-End	121,324	112,542	480,773

Table A.11 KDD'99 XGBoost Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,041	0,041	0,041
Encode / Encrypt	0,004	0,041	0,045
Evaluation	0,142	0,139	0,252
Decryption	0,013	0,013	0,013
End-to-End	0,200	0,234	0,351

Table A.12 KDD'99 SVM Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,042	0,041	0,041
Encode / Encrypt	0,375	9,422	9,421
Evaluation	32,269	31,942	57,943
Decryption	3,101	2,956	2,951
End-to-End	35,787	44,361	70,356

Table A.13 KDD'99 SVM* Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,04	0,039	0,152
Encode / Encrypt	0,032	0,285	0,364
Evaluation	32,890	32,014	78,839
Decryption	0,093	0,093	0,129
End-to-End	33,055	32,431	79,484

Table A.14 KDD'99 Decision Tree Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,042	0,040	0,152
Encode / Encrypt	0,005	0,037	0,048
Evaluation	0,351	0,344	0,960
Decryption	0,011	0,011	0,016
End-to-End	0,409	0,432	1,176

Table A.15 KDD'99 Neural Network Serial Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,041	0,039	0,151
Encode / Encrypt	0,097	0,073	0,314
Evaluation	23,130	22,551	57,885
Decryption	0,024	0,024	0,034
End-to-End	23,292	22,687	58,384

Table A.16 KDD’99 Naive Bayes Serial Implementation

KDD Cup 1999 Data Set: Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,159	0,158	0,153
Encode / Encrypt	0,172	3,119	3,152
Evaluation	15,349	11,458	52,725
Decryption	0,063	0,068	0,068
End-to-End	15,743	14,803	56,098

Table A.17 KDD’99 XGBoost Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,044	0,044	0,043
Encode / Encrypt	0,005	0,014	0,018
Evaluation	0,032	0,026	0,037
Decryption	0,003	0,002	0,002
End-to-End	0,084	0,086	0,100

Table A.18 KDD’99 SVM Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,045	0,044	0,048
Encode / Encrypt	0,064	0,963	0,941
Evaluation	3,548	3,519	6,460
Decryption	0,316	0,318	0,329
End-to-End	3,973	4,844	7,778

Table A.19 KDD’99 SVM* Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,040	0,038	0,152
Encode / Encrypt	0,034	0,286	0,362
Evaluation	3,785	3,775	9,318
Decryption	0,101	0,102	0,138
End-to-End	3,960	4,201	9,970

Table A.20 KDD'99 Decision Tree Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,040	0,043	0,151
Encode / Encrypt	0,006	0,037	0,047
Evaluation	0,062	0,061	0,136
Decryption	0,014	0,015	0,023
End-to-End	0,122	0,156	0,357

Table A.21 KDD'99 Neural Network Parallel Implementation

Operations	Model Encrypted	Data Encrypted	Both Encrypted
Key Generation	0,040	0,040	0,151
Encode / Encrypt	0,101	0,077	0,316
Evaluation	2,804	2,797	6,803
Decryption	0,031	0,035	0,040
End-to-End	2,976	2,949	7,310

Table A.22 KDD'99 Naive Bayes Parallel Implementation