CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Homomorphic vs Functional Encryption in Intrusion Detection Systems for OT Networks

Master's thesis in Computer Science and Engineering

Jibbril Ndaw Berbres
Valdemar Stenhammar

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Homomorphic vs Functional Encryption in Intrusion Detection Systems for OT Networks

Jibbril Ndaw Berbres
Valdemar Stenhammar

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Homomorphic vs Functional Encryption in Intrusion Detection Systems for OT Networks

Jibbril Ndaw Berbres
Valdemar Stenhammar

Supervisor: Magnus Almgren, Chalmers Computer Science and Engineering
Advisor: Lisa Eskilson, Knowit Cybersecurity & Law AB
Mentor: Henrik Nero, Knowit Cybersecurity & Law AB
Examiner: Robin Adams, Chalmers Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Homomorphic vs Functional Encryption in Intrusion Detection Systems for OT Networks

Jibbril Ndaw Berbres
Valdemar Stenhammar
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As the number and complexity of cybersecurity threats against Operational Technology (OT) systems rise, the need for effective countermeasures becomes increasingly critical. One such countermeasure is the deployment of Intrusion Detection Systems (IDS) that monitor for signs of malicious activity. However, the implementation of IDSs can present several challenges. OT devices can be resource-constrained, and personnel with expertise in IDS can be difficult to find. For these reasons, IDS functionality is frequently outsourced to external parties. While effective, this solution raises concerns due to the large amount of potentially sensitive data that is shared with the external party. To address the privacy risks of data sharing, a surge in research has focused on cryptographic techniques that enable privacy-preserving external IDSs. While many interesting techniques have been presented, few studies include rigorous performance evaluations and technical comparisons between different approaches. This thesis aims to fill this gap by comparing two cryptographic techniques in an OT setting.

The first technique uses Cheon-Kim-Kim-Song (CKKS), an instance of homomorphic encryption (HE) that allows an IDS server to perform computations on encrypted data. The resulting values are then decrypted by the client to determine whether the system has been compromised. The second approach uses function-hiding inner product encryption (FHIPE), an instance of functional encryption (FE). It enables an external IDS server to calculate the distance between a client system's normal state and current state without revealing either. This distance is then used for intrusion detection.

The HE and FE techniques are evaluated on the Secure Water Treatment (SWaT) dataset using a neural network-based IDS. The results show that both methods achieve threat detection accuracy comparable to their unencrypted counterpart. The additional detection latency introduced by the systems is found to be less than 125 ms. The client-side memory demands are less than 4 MB for the HE approach and less than 82 kB for the FE approach. Lastly, while both schemes increase the average network payload size significantly, the overall bandwidth usage remains manageable for most modern systems.

Keywords: computer science, cyber security, operational technology, intrusion detection systems, cryptography, homomorphic encryption, CKKS, functional encryption, FHIPE.

Acknowledgements

We would like to thank our supervisor Magnus Almgren at Chalmers Computer Science and Engineering for his insights and guidance during the thesis process. We also want to extend our gratitude to our company supervisors Lisa Eskilson and Henrik Nero from Knowit Cybersecurity & Law for their continuous support and motivation. Lastly, we wish to show our appreciation to Alice Svensson, Lisa Larsson, and Robin Alfredsson for all the good times during the thesis process.

Jibbril Ndaw Berbres & Valdemar Stenhammar, Gothenburg, 2025-06-18

Contents

List of Figures

List of Tables

Glossary

AI (*Artificial Intelligence*): The field that makes computing machines perform tasks that have been traditionally associated with human intelligence.

AUC (*Area Under the Curve*): Metric for measuring the general accuracy performance of a classification model without a specific threshold.

CKKS (*Cheon-Kim-Kim-Song*): An **HE** scheme that enables approximate computation over real numbers.

CPU (*Central Processing Unit*): The circuit in a computer that interprets machine code instructions.

DL (*Deep Learning*): A subset of **ML** where models are intended to resemble the human brain, using artificial neural networks.

FE (*Functional Encryption*): A field of cryptography that encrypts a plaintext such that it is possible to derive certain functions from the ciphertext.

FHIPE (*Function-Hiding Inner Product Encryption*): An **FE** scheme that computes the inner product of two encrypted vectors.

HE (*Homomorphic Encryption*): A field of cryptography that encrypts a plaintext such that it is possible to perform certain operations on the ciphertext.

IDS (*Intrusion Detection System*): A defensive software security mechanism that alerts when an anomaly occurs in the monitored system.

L_2**-distance** : A way of calculating the distance between two vectors.

ML (*Machine Learning*): A subset of **AI** where the models learn and adjust from data.

OT (*Operational Technology*): Computing systems that directly monitor and/or control some physical process.

PPIDS (*Privacy-Preserving Intrusion Detection System*): An **IDS** that operates on encrypted data.

ROC (*Receiver Operating Characteristic*): A curve of the false-positive and true-positive rates for various thresholds of a classification model.

1

Introduction

In the modern industrial landscape, operational technology (OT) is integral to the functioning of critical infrastructure such as power grids, water distribution, and sewer systems [1]. As OT systems have become more prevalent, so has the number and severity of cyberattacks against them. For example, consider the Sandworm attack during 2022 [2], where a Russian cyber threat group exploited vulnerabilities in the widely used SCADA system. By compromising the system, they managed to cause unplanned power outages, which coincided with mass missile strikes targeting critical infrastructure across Ukraine. Another example is found in a 2021 attack against the Oldsmar Water Treatment System in Florida [3]. There, a hacker managed to gain remote access to the water treatment facility and attempted to increase the sodium hydroxide concentration to more than 100 times its normal level. At such levels, the water could cause severe gastrointestinal injuries and potentially be fatal if consumed. Fortunately, the attack was noticed by an operator, and the tainted water was never released to the public. However, such attacks highlight the need for a proactive approach to the security of OT systems.

In recent years, there has been a much research aimed at countering the rising level of cyberattacks against OT systems [3][4][5]. A key component in these defensive measures is intrusion detection systems (IDS), which monitor both individual components and system traffic to detect deviations and malicious behavior. By recording the standard state of a system and monitoring for deviations, a so-called anomaly-based IDS can detect and alert system administrators in real time, enabling swift defensive actions.

With the rise of machine-learning (ML) and artificial intelligence (AI), significant research has been devoted to integrating modern AI and ML models into IDSs [4][6]. While this constitutes a promising approach, it also introduces several complications. Firstly, AI and ML models typically require large amounts of data to train and run. Secondly, designing and training AI and ML models is often difficult, requiring specialized skills and technical knowledge not available at all companies. Consequently, it is common to outsource model development to external parties, a concept known as Software as a Service (SaaS) [7].

Importantly, using SaaS can introduce security concerns regarding privacy. The external party might be malicious, or could be compromised by a cyberattack themselves. Due to these concerns, the resource owner may be unwilling or legally prohibited from sharing the data. The resource owner then faces a dilemma: either

make substantial investments in building their own model, or risk a data leak by relying on an external party.

To help ameliorate the problem of data-sharing, researchers have tried integrating cryptographic schemes to develop privacy-preserving IDS (PPIDS) that enable IDS functionality on encrypted data [8][9]. This allows the resource owner to access external providers while maintaining privacy. Although significant progress has been made towards developing efficient PPIDS solutions, many questions remain, and a broad range of potential vulnerabilities continues being explored by both malicious and benevolent actors.

This thesis aims to contribute by evaluating the performance of two cryptographic approaches used in the construction of PPIDSs: homomorphic encryption (HE) and functional encryption (FE). In particular, this thesis considers how integrating these approaches affects IDSs in an OT environment.

1.1 Problem motivation

The emergence of privacy-preserving IDSs that use modern cryptographic methods is an exciting development. However, many such systems are research prototypes that are not yet ready for real-world application. Thus, a concerted effort is needed by both researchers and industry to bridge the gap between theory and practice. One step in this direction is to rigorously evaluate the strengths and weaknesses of different PPIDSs, and to compare them with each other as well as with unencrypted IDSs.

There have been many papers detailing novel approaches for constructing and implementing PPIDSs [8][10][11], and several summary studies that attempt to create a taxonomy of these approaches [4][12][13]. However, most summary papers only offer brief overviews and focus on qualitative comparisons, and most papers introducing new PPIDS techniques only provide basic performance comparisons to alternative solutions. As such, there is a need for thorough technical evaluations that systematically compare various PPIDS approaches to clearly delineate their respective strengths and weaknesses.

1.2 Thesis objectives

This thesis conducts a rigorous analysis of how introducing cryptographic schemes affects the performance of IDSs in OT environments. In particular, it compares two PPIDS approaches based on HE and FE. The HE and FE approaches are compared with each other and with a baseline IDS that is not privacy-preserving. This work is guided by the following central research question.

> How does an IDS that utilizes homomorphic or functional encryption for privacy preservation compare to an unencrypted IDS when applied to OT data?

To thoroughly answer this, the central research question is divided into a set of technical research questions. These are:

- How does the introduction of HE and FE affect the system's detection accuracy?
- How does the introduction of HE and FE affect the detection speed of the system?
- How does the introduction of HE and FE affect the network traffic in the system?
- How does the introduction of HE and FE affect the memory consumption of the system?
- How does the introduction of HE and FE affect the interpretability of results in the system?

1.3 Limitations

IDS are widely used in both OT and non-OT scenarios. Although their implementations and usage can be similar, this study is limited to the context of OT. This is because data in OT environments is often sensitive—consider power grids and industrial control systems—making such an environment an excellent real-world application for PPIDSs. The experiments in this thesis are only conducted on one dataset, the Secure Water Treatment (SWaT) [14]. This data is collected from the OT environment of a water treatment facility, which will serve as the running example throughout this thesis.

As presented by Hendaoui et al. [4], there are many ways of constructing PPIDSs. Although several promising avenues to perform computations on hidden data exist—such as differential privacy, data anonymization, autoencoders, and blockchain—this thesis focuses exclusively on evaluating HE and FE. This focus allows for a more thorough and meaningful evaluation within the limited time frame of a master thesis project. HE and FE were selected in particular since they are both cryptographic in nature, meaning that they have a similar theoretic foundation. As a result, they can be implemented in comparable scenarios, ensuring a focused and more fair comparison.

1.4 Related work

The growing demand for IDSs, combined with an increased dissemination of ML and AI techniques has produced a surge of research exploring the use of AI and ML methods for intrusion detection. A subset of this research has focused on privacy-preserving IDS using HE and FE.

Ryu et al. [8] present a method for computing L_p-distances over encrypted data using FE. Their scheme builds on the FHIPE system proposed by Kim et al. [15], and shows that it is possible to compute L_p-distances even for larger values of p as long

as p is even. They also present two practical applications of their findings, one of which is a PPIDS. By leveraging the privacy-preserving nature of FHIPE, they are able to perform anomaly detection while maintaining data confidentiality. Although they present a system architecture which serves as the foundation for one of the models used in this paper, they provide limited performance metrics and evaluation of the system's characteristics.

Ibrahem et al. [16] propose a method for combining FE and IDS to combat electricity theft in power grids. Using their system, they allow power system operators to compute bills following a dynamic pricing approach, monitor the total grid load, and use ML to detect fraudulent consumers, all without revealing the individual consumption readings. For the detection case in particular, they train a feedforward network (FFN) which takes encrypted power readings and classifies whether they are correct or fraudulent. The system is thoroughly described, and relevant comparisons to other systems in the literature are provided. However, most of the alternative models are somewhat outdated and do not represent state-of-the-art techniques such as the TenSEAL approach used in this thesis. Additionally, none of the alternative solutions leverage homomorphic encryption to protect data privacy.

Wu et al. [9] present p2Detect, a system that uses homomorphic encryption to detect electricity theft in power grids. It works by aligning a detection server, an edge server, and multiple smart meters to safely aggregate and evaluate data. The detection server then performs identification of fraudulent activity without seeing the electricity consumption readings. p2Detect is a major improvement on previous HE-based detection systems, but the authors provides limited performance evaluation. A brief comparison is made to a well-established non-cryptographic detection model, but no further analysis is provided.

Li et al. [10] propose an anomaly detection system for advanced metering infrastructure (AMI) that uses fully homomorphic encryption (FHE). Although FHE is known to be computationally demanding, the authors leverage precomputed lookup tables to significantly improve performance. By applying these techniques together with a private information retrieval (PIR) scheme, the authors achieve efficient anomaly detection while maintaining the secrecy of customer smart meter data. Although the authors provide graphs showcasing the model's properties, their performance analysis could be expanded by adding comparisons to alternative solutions.

Niu et al. [17] address the problem of outsourcing ML prediction services to third-party providers. To circumvent the problem of data-sharing, they propose a model named MVP that enables ML prediction in a verifiable and privacy-preserving manner. By leveraging polynomial decomposition and bilinear mathematical groups, the authors achieve oblivious detection evaluation that preserves data privacy. They also apply this model using support vector machines (SVM) to successfully perform spam detection on three different datasets. The authors present a mathematically rigorous approach to anomaly detection, but similar to many other studies in the area of PPIDS, they do not provide much in the way of comparisons to alternative solutions.

Zhang et al. [11] present a model that uses local density estimation (LDEM) for

anomaly detection. By estimating the density of new occurrences and comparing it to the average density of a training set, the LDEM model is able to efficiently perform anomaly detection. The authors extend the LDEM model by incorporating homomorphic encryption, resulting in a privacy-preserving variant (PPLDEM). A standout feature of the proposed model is that it can perform the density analysis with an $O(n)$ time complexity, making it efficient compared to alternative solutions.

Karaçay et al. [18] present another approach to PPIDS. They use a lattice-based approach to enable anomaly detection using somewhat homomorphic encryption (SHE). Their approach differs from many other PPIDS models in that, rather than encrypting the data, it encrypts the detection model itself. The encrypted model is then sent to the data owner (DO), who uses it for detection purposes. When evaluating the model using real datasets, they find no deterioration in accuracy compared to a similar well-established tool.

Chou et al. [19] introduce a lightweight pipeline that uses machine learning for privacy-preserving phishing detection. By using homomorphic encryption techniques, the model allows users to securely share the visual features of their browser with external providers. The external provider can then perform anomaly detection to differentiate authentic websites from phishing sites. The proposed model performs this visual classification with 90% accuracy and a computation time of less than a second. Although the visual approach showcases an interesting perspective on anomaly detection and the results are promising, the authors do not present direct comparisons to alternative methods.

In conclusion, while many studies have investigated the possibility of using HE and FE in an IDS settings, there is a significant lack of technical evaluations to quantify the performance of such systems. Most existing studies only provide basic performance comparisons, which makes it difficult for implementers to anticipate the impact of integrating such systems. This thesis aims to partially fill this gap by thoroughly evaluating two modern cryptographic approaches across a wide range of metrics.

1.5 Thesis outline

Chapter 1 provides an introduction to the problem investigated in this thesis. Various incidents are highlighted, demonstrating the need for developing privacy-preserving IDS solutions. This is followed by the thesis objectives, limitations, and a review of related work.

Chapter 2 provides a high-level introduction to subjects and research areas relevant to this thesis. Initially, the concept of OT is explained, followed by an overview of the basics in IDS, AI, and cryptography. Finally, a brief background is given regarding some of the Python libraries used in the construction of the PPIDSs.

Chapter 3 provides a technical low-level overview of the two major research areas addressed in this thesis: cryptography and neural networks. First, an overview of cryptography concepts is given, followed by a presentation of the concrete schemes

used. Next is an exploration of neural networks, which serve as the anomaly detection model in this thesis.

Chapter 4 provides an in-depth explanation of the experiments conducted for the comparison. Initially, an overview to cryptographic protocol scenarios is given. This is followed by a presentation of the constructed anomaly-detection model. Lastly, the chapter presents the metrics used in the comparison, along with a description of the experimental environment.

Chapter 5 presents the results, metric by metric. The results are then analyzed in Chapter 6, where the key findings are highlighted. This is followed by a discussion on ethics, sustainability, and future work. Finally, the thesis is concluded in Chapter 7.

2

Background

This thesis centers on three core theoretical domains: intrusion detection systems in OT environments, artificial intelligence, and cryptography. The interaction between these domains is of particular interest, as modern OT environments increasingly demand integrated solutions that draw upon all three. This chapter aims is to provide a basic foundation that situates the thesis within its broader academic context and supports the reader's understanding of its technical aspects. While the chapter does not aim to exhaustively explain each subject, references to the relevant literature are provided for the interested reader.

2.1 Operational technology

Operational technology (OT) is a counterpart to the more common domain of information technology (IT). While IT mainly considers software and communication between devices—such as clients and servers—OT encapsulates the interaction between a system and the physical environment [20]. Therefore, OT systems are integral to the operation of critical infrastructure such as power plants, water distribution, and sewer systems.

Much of modern OT is a result of automation and the integration of IT into existing physical systems [21]. For example, many manual systems have been replaced with embedded systems. The tasks of these embedded systems can be very sensitive and must be performed with great caution and accuracy: if the system were to fail, the consequences could in the worst case be catastrophic. Since OT systems are often prevalent in critical infrastructure, it is of utmost importance that these systems ensure reliability, safety and continuous operations [1].

Critical infrastructure is a natural target for malicious actors intending to disrupt a country's national safety. Consequently, the OT systems operating in these environments must be robust and able to withstand complex cyberattacks. OT systems typically have an "air gap", meaning a physical separation between the critical system and any untrusted network, such as the internet [21]. While air-gapped systems are not impenetrable[1], they are harder to breach. However, the emerging concept of Industrial Internet of Things (IIoT) with the introduction of big

[1]The Stuxnet malware breached air-gapped systems through USB flash drives.

data techniques have caused OT environments to become increasingly connected [1]. While this can be beneficial, it can also broaden the attack surface [21].

2.2 Intrusion detection systems

With a growing threat landscape comes a greater difficulty in keeping systems secure from malicious adversaries. Consider the Stuxnet attack that occurred in 2010 where malware spread to 14 industrial sites in Iran [22]. Among other victims, the malware successfully infiltrated a uranium-enrichment plant where it gained control of the centrifuges, spinning them to failure. Companies and organizations strive to avoid such disasters, but the cost of hiring knowledgeable IT-personnel is rapidly rising, creating bottlenecks across entire industries [23].

Intrusion detection systems (IDS) monitor and analyze a system's behavior for signs of potential security issues. Bace and Mell [24] define intrusions as "attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network" and claim that the purpose of an IDS is to automate this monitoring and analysis process.

There is a plethora of different intrusion detection techniques available, each suited for a specific type of system or data. The techniques can generally be categorized into either signature-based or anomaly-based detection [25]. In signature-based detection, the IDS compares network traffic against a database of well-known attacks. While simple to implement and highly efficient in detecting known attacks, a significant drawback of this method is that it is unable to detect zero-day exploits.

In anomaly-based detection, the IDS relies on the assumption that an intrusion will cause the system to exhibit some unusual behavior. Given this assumption, one can establish a model of the system's normal behavior, and then identify outliers as anomalies [26]. However, the challenge of anomaly-based detection lies in the difficulty of accurately specifying a system's normal behavior. Machine learning (ML) tools, especially deep learning (DL), has shown great promise in mitigating this issue [27]. While DL techniques is able to detect new types of attacks, it can also produce many false-positives [28].

Regardless of detection method, data-sharing between the monitored system and the IDS is required, which can be problematic if the IDS resides with an external service provider. A promising approach to mitigating data-sharing privacy concerns is integrating IDSs with cryptography [4]. By encrypting the data before transmitting it to a third-party, confidentiality is maintained while still retaining the IDS functionality.

2.3 Artificial intelligence

Artificial intelligence (AI) is a field combining math and computer science to solve tasks typically requiring human intelligence, or tasks that involve analyzing data at scales exceeding human ability [29].

With recent development in tools including ChatGPT [30] and Microsoft Copilot [31], AI has received widespread media coverage. However, there are many misconceptions about the capabilities of AI and many of the terms are used interchangeably, when they in fact refer to different concepts within AI. Figure 2.1 illustrates the proper hierarchy between some instances of the misinterpreted terminology. This section encompasses all of these areas and provides the necessary background on the relevant AI techniques and methodologies utilized in this thesis.

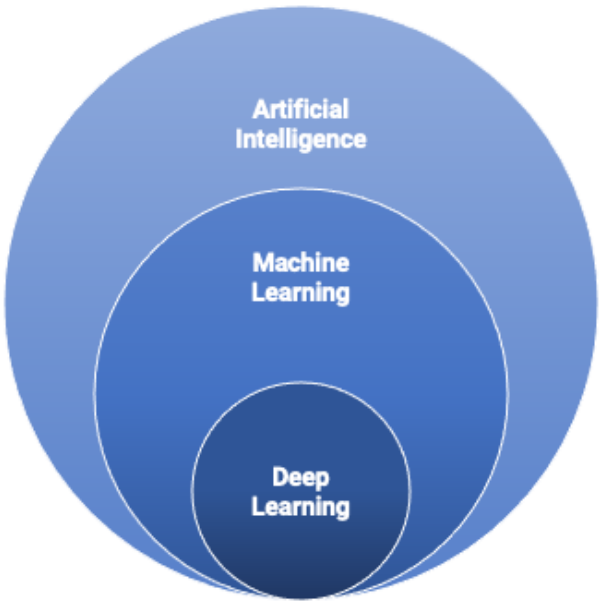


Figure 2.1: Venn diagram illustrating the relation between AI, ML and DL. Reproduced from [32].

2.3.1 Machine learning

Machine learning is a subfield of AI where models are designed to make predictions or decisions based on data, without being explicitly programmed for each particular task [33]. In this context, a model refers to a mathematical function that takes some aspects of reality as input, allowing it to create a description of reality to be used for making predictions. Instead of relying on predefined rules or instructions, ML algorithms build more general models that can be trained to recognize patterns and relationships in data, allowing the models to improve over time as they are exposed to more information. This data-driven approach allows the models to learn from historical data and adapt to new situations. By leveraging vast amounts of data, ML systems have the ability to tackle complex problems across a wide range of domains, including computer vision, finance, and computational biology [34].

There are three primary paradigms within ML: supervised learning, unsupervised learning and reinforcement learning [33]. However, while reinforcement learning is applicable in an IDS context, its methodology differs from the techniques used in this thesis. Thus, reinforcement learning is not considered in the scope of this

background.

Supervised learning is a subset of ML techniques where the model is trained on labeled data, meaning that each data point has a corresponding label [35]. Consider an email service with a spam filter that uses a supervised machine learning model. The filter can be trained on a labeled dataset where each data point is a pair of an email and its corresponding label, i.e. "Spam" or "Not spam". This allows the model to learn the relationship between the input (email) and the output (label), and it can use this knowledge to infer labels on new unseen emails. This is known as a classification problem and it is one of the main use cases of supervised learning [33].

Unsupervised learning, contrary to supervised learning, is a subset of ML techniques where the model is trained on unlabeled data [36]. This allows the model to find hidden patterns and discern specific groups in the data without any explicit instructions or guidance. Consider an email service that automatically categorizes incoming emails. When trained on unlabeled emails, the model creates clusters based on the similarity of their contents. Generally, the model cannot infer what each category represents (work, travel, newsletters), but it can create groups of emails that the user can then label. This is known as a clustering problem and is one of the main use cases for unsupervised learning [37].

2.3.2 Deep learning

As illustrated by Figure 2.1, deep learning is a branch of ML. The distinguishing feature of DL is not the problems it aims to solve, but the methodology it builds on. The mathematical concept behind DL is that of artificial neural networks. These are models whose architecture is inspired by the human brain, thus enabling machines to solve many problems they would otherwise struggle with [38]. This section will provide a brief introduction to DL and its applications, and neural networks will be covered in technical detail in Section 3.2.1.

At the core of DL models lies a layered architecture, consisting of interconnected neurons, each responsible for extracting a specific feature in the data through weights and activation functions [39]. These layers enable the model to capture increasingly abstract patterns as the data propagates forward through the layers. The structure of the network and the types of layers employed vary greatly depending on the purpose of the network. However, all networks adhere to the following abstract structure:

1. **Input layer**: Receives raw data as input.
2. **Hidden layers**: Extracts features from the data. Normally, this part consists of several layers, each focusing on a specific type of feature. There is no exact definition, but to be considered a "deep" network there should generally be at least more than two hidden layers, otherwise it can be referred to as a "shallow" network [40].
3. **Output layer**: Outputs the final prediction or result.

The hidden layers are the defining part of a network as this is where the network learns to represent the data in abstract and useful ways. These layers can be

optimized and tailored to the intention of each specific network. Convolutional layers in convolutional neural networks (CNN) provide a conceptual example of feature extraction. CNNs are commonly utilized in image recognition for their ability to extract visual features [38]. Consider a DL model tasked with interpreting handwritten digits. A visual feature in this case could be a small circle or a line located on a specific part of the image. By using several hidden layers, the CNN can extract multiple features from the same image and aggregate them to infer which number is written [41]. For instance, a horizontal line near the top and a diagonal line from the top right to the bottom left could be two extracted features resulting in the output seven.

The hidden layers are not the only factor that decides the performance of the network, other optimization vectors also require attention. The network learns and improves over time in the training phase by updating its internal weights, a process called backpropagation [42]. Conceptually, when the model provides an incorrect output it tries to improve itself to avoid making the same mistake again. To quantify what went wrong, the model calculates a loss function, which measures the error between the predicted and expected outputs. In turn, the gradient of the loss function is utilized in an optimization algorithm, which optimizes the weights of the network. The choice of loss function and optimization algorithm should be tailored to the specific purpose of the model.

2.4 Cryptography

Cryptography is sometimes defined as the art of sending messages over a public network without revealing any information to unauthorized parties. At its core, it provides mathematical tools for ensuring confidentiality and integrity, even in untrusted environments. This section gives relevant background knowledge on cryptography, beginning with the basic notation and finishing with the more advanced concept of Elliptic Curve Cryptography (ECC).

2.4.1 Introduction to cryptography

Cryptography often involves taking a plaintext message m and encrypting it using a function Enc to produce a ciphertext c. The ciphertext c can then be decrypted using a decryption function Dec to recover m. This constructs the basic cryptographic relationship

$$Enc(m) = c$$
$$Dec(c) = m$$

There are numerous cryptographic schemes that follow this pattern. The historically most widely used is called symmetric cryptography. In symmetric cryptography, a sender and a receiver share a key k used to encrypt m and decrypt c such that

$$Enc(k, m) = c$$
$$Dec(k, c) = m$$

Symmetric encryption has been widely used in protocols such as Transport Layer Security (TLS) [43], but it has several drawbacks, the most notable being the problem of initially sharing the key k.

For a sender and receiver to communicate using symmetric encryption, they must share the key k beforehand. If the participants are geographically close to each other, they can simply meet up and decide on a key. However, if they are far apart, or otherwise unable to meet, they first need a secure way of sharing k. This raises the question: if they have such a method, why not use it to send the actual messages as well?

The first open solution to the problem of key-sharing over a public network in symmetric cryptography was presented by Diffie and Hellman [44]. They suggested that a secure key exchange could be achieved by combining certain public and private values using exponentiation inside mathematical groups. This idea became the first iteration of what is now known as asymmetric cryptography, or public-key cryptography. Although the core ideas presented by Diffie and Hellman for cryptographic key exchange are still in use, many alternative techniques have gained popularity over the years. This thesis leverages one of these, called Elliptic Curve Cryptography.

2.4.2 Elliptic curve cryptography

Elliptic Curve Cryptography (ECC) is a field of cryptography in which the properties of elliptic curves are used to enable public-key cryptography. ECC can provide equivalent security properties to popular alternatives such as RSA [45], but at a lower computational cost. To accomplish this, ECC uses several sophisticated mathematical concepts that fall outside the scope of this background. As such, only an intuitive introduction to the subject is provided. Curious readers can refer to the excellent coursebook by Boneh and Shoup [46] that is freely available online.

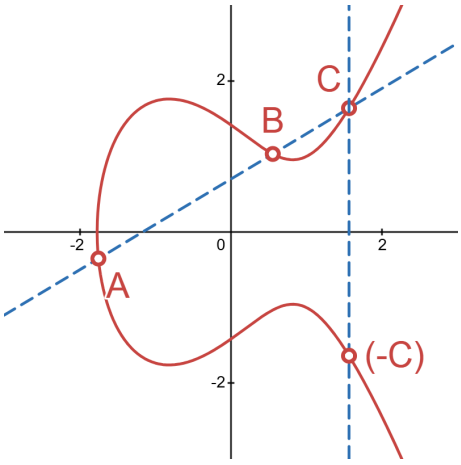


Figure 2.2: An elliptic curve where a line through the points A and B intersects the curve in the point C. C is then mirrored over the x-axis to obtain $-C$.

An elliptic curve is defined by the equation $y^2 = x^3 + ax + b$ where a and b are chosen such that the discriminant $\Delta = -16(4a^3 + 27b^2) \neq 0$. If this holds, the corresponding curve will be smooth and resemble the one seen in Figure 2.2, where the parameters (a, b) are set to $(-2, 2)$.

For the purposes of ECC, the key property of elliptic curves is how their structure interacts with the equation of the straight line. This is best explained using an example: Consider two points A and B on an elliptic curve. If a straight line is drawn through them, it will—under certain conditions on the selected points—intersect the curve at some third point C [46]. The point C can then be mirrored over the x-axis, as shown in Figure 2.2. In ECC, this operation of combining two points to find the third is captured in an operator denoted $\oplus$. Thus, Figure 2.2 shows the ECC operation of $A \oplus B = -C$. If one was to repeat this process, by computing $A \oplus (-C)$ for example, one would find a new point that could be mirrored over the x-axis. By continually using the $\oplus$ operation, it is possible to move across the curve in a seemingly erratic manner. This structured yet complex behavior forms the basis of ECC security, and is also the foundation on which the function-hiding inner product encryption scheme introduced in Section 3.1.3 is built.

2.5 Libraries

There are several publicly available libraries that can facilitate the task of constructing privacy-preserving intrusion detection models using homomorphic and functional encryption. Following is a presentation of the most important libraries employed in this thesis. Each library is given a brief introduction to its origins, followed by highlights of its contributions in the context of privacy-preserving IDSs.

2.5.1 PyTorch

PyTorch [47] is an open-source machine learning framework for Python. The framework provides a Python frontend with the GPU-accelerating features from the Torch library, enabling readable code and rapid prototyping [48]. For its ease of use and high performance, PyTorch is leveraged by many of the tech giants, including Microsoft, Apple, and IBM [49].

The PyTorch API provides the tensor data structure which is a multidimensional array that is optimized for deep learning purposes. The optimization includes parallel GPU-acceleration support and automatic differentiation. Using GPUs can greatly enhance the computation speed, as GPUs utilize parallel computation to a much greater degree than CPUs. Automatic differentiation refers to the ability of tensors to compute their own gradient efficiently. This is very useful for optimizing the parameters in a neural network, the backpropagation technique, which often relies on analyzing the gradients of each feature.

2.5.2 TenSEAL

TenSEAL [50] is an open-source Python library developed by OpenMined that facilitates HE operations on tensors. Built as a wrapper to Microsoft SEAL—a Python HE library supporting multiple schemes—TenSEAL provides a high-level interface that is similar to PyTorch, but that also enables encrypted tensor operations. The API includes support for operations such as matrix multiplication, reshaping and even 2-dimensional convolutions. These capabilities make it possible to construct neural networks that perform inference directly on encrypted data, while preserving privacy throughout the computation process. Therefore, TenSEAL is of particular interest for this thesis.

2.5.3 Charm

The Charm cryptographic library [51][52] is an open-source Python library designed to rapidly prototype advanced cryptosystems. It was first developed by researchers at Johns Hopkins University with academic use in mind. Charm provides a flexible interface for building cryptosystems, and relies on high-performance C libraries under the hood, making it suitable for theoretical exploration as well as proof-of-concept implementations.

Of particular interest to this thesis, Charm has comprehensive support for elliptic curve cryptography and bilinear groups. It enables the user to construct pairing-based protocols using pairing functions such as the Tate pairing, which are essential for constructing many types of cryptographic systems. In particular, the function-hiding inner product encryption scheme introduced in Section 3.1.3 uses these concepts as the foundation for its security.

3

Theory

While Chapter 2 provided an overview of the broader research area, this chapter details the technical frameworks, methods, and concepts that underpin the work presented in this thesis. This includes technical explanations of the HE and FE protocols, as well as the neural network-based IDS model. As such, this chapter serves as a bridge between the general background and the subsequent methodology chapter.

3.1 Cryptographic protocols

To enable secure and private processing of sensitive data, cryptographic techniques that allow computation over encrypted information are essential. While Section 2.4 introduced some basic background on the field of cryptography, this section aims to introduce two cryptographic paradigms that support privacy-preserving computations in different ways—homomorphic encryption and functional encryption. Homomorphic encryption supports direct computation on ciphertext data, while functional encryption enables certain fine-grained functions to be drawn from encrypted data. These protocols serve as the foundation for the privacy-preserving mechanisms discussed later in this thesis.

3.1.1 Homomorphic encryption

Homomorphic encryption (HE) is a field of cryptography that enables ciphertext-computation, i.e. computations that are performed directly on encrypted data. Specifically, it enables certain mathematical operations to be applied to ciphertexts in such a way that, once decrypted, the result matches what would have been obtained by directly applying the operations on the plaintext [53]. The primary goal of HE is to facilitate multi-party computation over sensitive data, allowing external service providers to operate on encrypted data without accessing the underlying information. This ensures the confidentiality of the data, even in insecure computing environments.

To provide a practical and intuitive example for the applicability of HE, consider an OT environment such as a water treatment facility. The facility wants to monitor for signs of intrusions or other anomalies in the data, without exposing the raw data to the detection system. This is particularly important if the intrusion system is

outsourced, or deployed as an untrusted component internally.

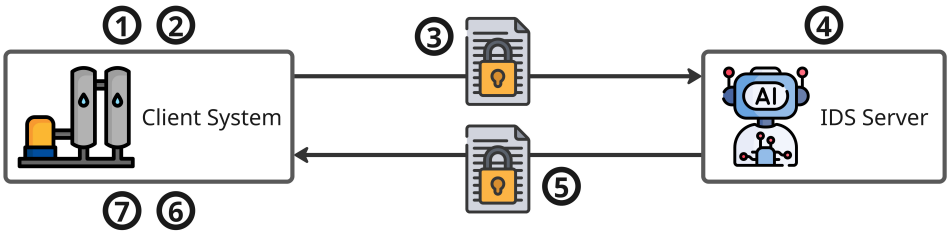
Suppose each device in a network produces data which is aggregated into a vector $\mathbf{x}$, containing information such as valve pressures, chemical levels, and flow rates. Normally, an IDS would receive $\mathbf{x}$ in plaintext, analyze it using some function $x' = f(\mathbf{x})$ and then report back whether an anomaly was detected. However, this exposes the data $\mathbf{x}$ to the IDS. Instead, by homomorphically encrypting the data, the IDS can compute and return an encrypted result:

$$y = f'(\text{Enc}(\mathbf{x}))$$

Note that the output y is encrypted and cannot be interpreted by the IDS. Additionally, f' achieves the same computation as f, but in a way that is compatible with the HE protocol. With this, only the party with the private keys is able to obtain the result

$$x' = \text{Dec}(y) \in \{0, 1\}$$

Thus, the facility has successfully received the same result x' as in the plaintext case, but computed by an untrusted IDS without exposing any of the sensitive internal data. A simplified visualization of this flow is given in Figure 3.1. Here, the client wants to calculate the sum s of two measurements a_1 and a_2, that is: $s = a_1 + a_2$.



HE addition flow

1. Client records plaintext data a_1 and a_2.
2. Client encrypts data as $x_1 = \text{Enc}(a_1)$ and $x_2 = \text{Enc}(a_2)$.
3. Client sends (x_1, x_2) to the IDS server.
4. IDS server computes $x' = x_1 \oplus x_2$
5. IDS server returns x' to the client.
6. Client decrypts x' as $s = \text{Dec}(x')$.
7. Note that $s = a_1 + a_2$.

Figure 3.1: Example of how HE addition is performed. Note that in step 4, a specific homomorphic addition operation $\oplus$ substitutes normal addition $+$.

The core ideas of encrypted computation reach back to the 1970's, but the first implementation of a partially[1] homomorphic encryption scheme was the RSA cryptographic system [45]. There have been many developments since, and of particular interest for this thesis is the CKKS scheme.

[1]Meaning that it only supports one type of operation.

3.1.2 Cheon-Kim-Kim-Song scheme

For this thesis, a specific approach to HE called Cheon-Kim-Kim-Song (CKKS) [54] is of particular interest. The scheme builds on lattice-based cryptography, and the hardness guarantee relies on the ring learning with error (RLWE) problem [54]. The scheme enables approximate addition or multiplication of two vectors in a privacy-preserving manner using these underlying techniques. This means that two scalars a and b can be encrypted under CKKS as

$$x = \texttt{CKKS.Encrypt}(a)$$
$$y = \texttt{CKKS.Encrypt}(b)$$

Given x and y, an external party can compute the sum z and product z':

$$z = \texttt{CKKS.Add}(x, y)$$
$$z' = \texttt{CKKS.Mult}(x, y)$$

Finally, when decrypted, the results match the normal mathematical operations of addition and multiplication:

$$a + b \approx \texttt{CKKS.Decrypt}(z)$$
$$a \cdot b \approx \texttt{CKKS.Decrypt}(z')$$

Thus, an external party has calculated the sum $a+b$ and product ab without learning anything about a or b. Note that the results are approximate, with a small error introduced by the noise inherent in the scheme. The error remains manageable as long as the computational depth is limited and the parameters are chosen carefully.

CKKS is a leveled homomorphic encryption scheme, meaning it allows for evaluation up to a pre-specified level L without requiring the computationally expensive bootstrapping procedure. Multiplication introduces significant noise to the ciphertext, and the specified level determines the number of multiplications possible before the ciphertext becomes indecipherable. The level is specified by producing a list of coefficient modulus, q. Each encrypted multiplication consumes one level of coefficient modulus, and the current level is denoted q_l. The scheme exposes five different functions, where a brief description of each follows. The curious reader is referred to the original paper by Cheon et al. [54], or the excellent article series by Daniel Huynh [55], for more detailed explanations.

CKKS.KeyGen

Given a security parameter λ describing the bit-level security, the `CKKS.KeyGen` function generates a public key pk, a secret key sk, and an evaluation key evk. The secret key sk is derived by sampling a small, sparse polynomial. The public key pk is constructed using the secret key along with Gaussian noise to ensure security. Lastly, the evaluation key evk is required to perform encrypted multiplication. It is generated by combining the square of the secret key with an integer P and some Gaussian noise. The function returns (sk, pk, evk).

CKKS.Encrypt

Given an encoded plaintext message m, the `CKKS.encrypt` function samples a small ternary vector v and Gaussian noise (e_0, e_1). The ciphertext is computed by combining the public key with the message and noise, to encrypt the message. The function returns the tuple $v \cdot pk + (m + e_0, e_1) \mod q_l$.

CKKS.Decrypt

Given an encrypted ciphertext $c = (b, a)$, the `CKKS.Decrypt` function decrypts the ciphertext using the secret key s. The function returns $b + a \cdot s$, which is an approximate reconstruction of the original plaintext.

CKKS.Add

Given two encrypted ciphertexts (c_1, c_2), the `CKKS.Add` function adds them. This is done component-wise and modulo the coefficient modulus. The function returns $c_1 + c_2 \mod q_l$.

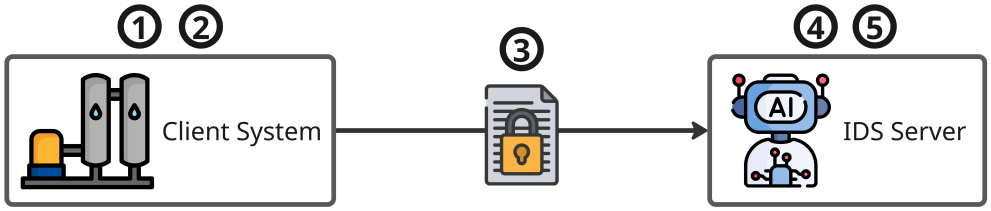
CKKS.Mult

Given two encrypted ciphertexts $c_1 = (b_1, a_1)$ and $c_2 = (b_2, a_2)$, the `CKKS.Mult` function first computes the intermediate terms $(d_0, d_1, d_2) = (b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \mod q_l$. Next, to reduce the amount of noise, the integer P from `CKKS.KeyGen` is used in conjunction with the evaluation key evk. The function returns $(d_0, d_1) + P^{-1} \cdot d_2 \cdot evk \mod q_l$.

3.1.3 Functional encryption

Functional encryption (FE) is a cryptographic paradigm that differs from conventional encryption. Similar to HE, it is a way of hiding information while enabling external parties to perform certain actions [56]. However, rather than encrypting and then sharing the data as in HE, FE encrypts the data in such a way that only a specific function of the data can be computed and revealed. This means that if a client has some data x that they do not want to share with an external party, they can define a function f and encrypt it using specific techniques to obtain a ciphertext $c = \text{Enc}(x)$. This c can then be shared, alongside a secret key k, to allow the external party to compute $x' = \text{Dec}(c, k) = f(x)$. If the function f is properly defined, x' can then enable useful applications while leaking minimal or no information about x itself. This flow is visualized in Figure 3.2.

To exemplify FE, consider the scenario of a water treatment facility. The facility has engineers, operators, auditors, and other roles that all need knowledge of various parts of the facility to perform their jobs. However, given the sensitive nature of an OT facility such as a water treatment plant, there is an incentive to limit the scope of information that each job group can access. For example, the operators might need to know whether a device has experienced any anomalies in the past 24 hours. To make this determination, the facility could give the operators access



FE function flow

1. Client records plaintext data x.
2. Client encrypts data as $c = \text{Enc}(x)$.
3. Client sends c and a decryption key k to the IDS server.
4. IDS server decrypts c as $x' = \text{Dec}(c, k)$.
5. Note that $x' = f(x)$, for some predefined function f.

Figure 3.2: Example of how FE allows the IDS server to compute a predefined function $f(x)$, without learning x itself.

to all the device data for the last 24 hours, denoted x, and let them investigate it for anomalies. However, this method would be disastrous if a malicious actor masqueraded as an operator, since they would gain full access to the device data, and could plan and execute attacks using it. A better alternative could be to define a function f which takes in the data x, evaluates it for anomalies, and then reports either 0 or 1 depending on whether an anomaly was found. If the result is denoted as

$$x' = f(x) \in \{0, 1\}$$

the operator could then fulfill their duties knowing only x' and not x. Thus, the facility protects its sensitive information while still benefiting from the operator's work. Although no technical specifications were mentioned in this minimal example, it succinctly captures the essence of FE: hiding real data and only exposing the result of some specific function.

The core ideas of FE were first introduced by Boneh et al. [56], but there have been many developments since. One of these is of particular interest to this thesis: function-hiding inner product encryption, which is discussed in the next section.

3.1.4 Function-hiding inner product encryption

For this thesis, a special form of FE called function-hiding inner product encryption (FHIPE) is of particular interest. Originally introduced by Kim et al. [15], the FHIPE scheme uses FE techniques to efficiently compute the inner product of two vectors in a privacy-preserving manner. This means that given two vectors $\mathbf{x}$ and $\mathbf{y}$, the scheme enables an external party to compute the inner product $z = \langle \mathbf{x}, \mathbf{y} \rangle$ without learning any information about $\mathbf{x}$ and $\mathbf{y}$ beyond the result.

The FHIPE scheme consists of four functions that all operate over the message space $\mathbb{Z}_q^n$, where n is the length of the vectors $\mathbf{x}$ and $\mathbf{y}$, and $q \in \mathbb{N}$ is a large prime. The functions are `IPE.Setup`, `IPE.KeyGen`, `IPE.Encrypt` and `IPE.Decrypt`. These will now be presented along with their associated properties.

IPE.Setup

Given a chosen security parameter λ signifying the desired bit security, along with a set $S \subseteq \mathbb{Z}_q$, where $|S| = \text{poly}(\lambda)$, indicating that the size of S grows polynomially with the security parameter, the `IPE.Setup` function uses a predefined asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, q, e)$ to perform the initial setup of the system. This consists of choosing two generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ along with a matrix $B \leftarrow \mathbb{GL}_n(\mathbb{Z}_q)$, where $\mathbb{GL}_n(\mathbb{Z}_q)$ signifies the general linear group of invertible matrices with size n and values sampled from $\mathbb{Z}_q$. The function then computes the matrix

$$B^* = \det(B) \cdot (B^{-1})^\top$$

and outputs the public parameters $pp = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, q, e, S)$ and the master secret key $msk = (pp, g_1, g_2, B, B^*)$.

IPE.KeyGen

Given the master secret key msk and a vector $\mathbf{y} \in \mathbb{Z}_q^n$, the key generation function starts by uniformly sampling an element $\alpha \leftarrow \mathbb{Z}_q$ and using it to compute

$$K_1 = g_1^{\alpha \cdot \det(B)}$$
$$K_2 = g_1^{\alpha \cdot \mathbf{y} \cdot B}$$

It then returns a secret key $sk = (K_1, K_2)$. Note that K_1 is a group element, and K_2 is a vector of group elements.

IPE.Encrypt

Given the master secret key msk and a vector $\mathbf{x} \in \mathbb{Z}_q^n$, the encryption algorithm begins by uniformly sampling some $\beta \leftarrow \mathbb{Z}_q$. It then computes

$$C_1 = g_2^{\beta}$$
$$C_2 = g_2^{\beta \cdot \mathbf{x} \cdot B^*}$$

and returns the ciphertext $ct = (C_1, C_2)$, where C_1 is a single group element and C_2 is a vector of group elements.

IPE.Decrypt

Given the public parameters pp, a secret key $sk = (K_1, K_2)$, and a ciphertext $ct = (C_1, C_2)$, the decryption function computes and returns the scalar product $\langle \mathbf{x}, \mathbf{y} \rangle$. It starts by using the pairing function e from the bilinear group to compute

$$D_1 = e(K_1, C_1)$$
$$D_2 = e(K_2, C_2)$$

If ct and sk have been created in accordance with the FHIPE system, and $\langle \mathbf{x}, \mathbf{y} \rangle \in S$, then there will exist a $z \in S$ such that

$$(D_1)^z = D_2$$
$$z = \langle \mathbf{x}, \mathbf{y} \rangle$$

Since $|S| = poly(\lambda)$, the discrete logarithm to find z can be computed efficiently. If it exists the decryption function returns it as output, otherwise it returns $\perp$. With that, the FHIPE scheme is complete.

3.2 Intrusion detection model

Intrusion detection systems are essential tools for monitoring a systems traffic and identifying potential security threats. This section provides an in-depth technical review of the underlying technology used in the IDS developed for this thesis. Specifically, it focuses on neural networks starting from a single neuron and finishing in the final layered architecture.

3.2.1 Neural networks

Artificial neural networks (hereafter referred to simply as neural networks) are mathematical structures inspired by the real neural networks found in human brains, and they are the essence of deep learning. As described in Section 2.3.2, a neural network is a layered architecture where each layer consists of a set of neurons with weights and activation functions. This section provides a technical review of neural networks, explaining key concepts including different layer types and activation functions. It begins with the initial artificial neuron: the perceptron.

The perceptron is a mathematical object inspired by neuron cells, and it is the fundamental building block of neural networks [57]. The concept was introduced by Frank Rosenblatt [58] in 1958, and the model remains largely unchanged even today. The idea of the perceptron is to produce a single output, which is derived from multiple inputs. Each input is assigned a weight describing its influence on the output, i.e. an input that is of great importance for the output has a larger weight. This is summarized in Equation 3.1, where all the inputs x_i are weighted with their respective weight w_i and then summed together. Lastly, a bias term b is applied to shift the activation function, which allows the perceptron to model data that is not centered at the origin. Intuitively, if the weights determine the slope of a line, the bias determines the intercept.

$$z = \sum_i x_i w_i + b \tag{3.1}$$

3.2.2 Activation functions

When the perceptron's value has been calculated as per Equation 3.1, that value is passed through an activation function which determines the output of the perceptron [59]. A rudimentary activation function is the Heaviside step function [60] defined in Equation 3.2. This activation function maps the output to either 0 or 1, based on whether the perceptron value z is greater than or less than 0.

$$H(z) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3.2}$$

The Heaviside step function can be generalized by introducing a threshold values θ that determines whether the output is 0 or 1, as presented in Equation 3.3.

$$H_\theta(z) = \begin{cases} 1, & x \geq \theta \\ 0, & x < \theta \end{cases} \tag{3.3}$$

While easy to understand, the downside of the Heaviside activation function is that it is not differentiable at $z = 0$ and constant everywhere else. This matters in the backpropagation step of neural networks, i.e. the process by which the model optimizes its internal parameters as described in Section 2.3.2. A common optimization algorithm is stochastic gradient descent (SGD) [61], which calculates the partial derivatives for each learnable parameter to minimize the loss function [59]. Consequently, SGD, and other gradient-based optimization algorithms, require meaningful gradients (derivatives) from the activation function in order to update the model parameters efficiently. Since the Heaviside activation function is constant almost everywhere, its gradient is zero in those regions, which prevents the model from learning during backpropagation.

An alternative activation function that is continuous and differentiable is the Sigmoid function, as defined in Equation 3.4. A key feature of the Sigmoid function is that its pre-image is $(-\infty, \infty)$ and its image is in the range $(0, 1)$. This means that it maps any real number $x \in \mathbb{R}$ to a real number $\sigma(x) \in (0, 1)$. Additionally, the Sigmoid function has the property of being monotonically increasing, i.e. greater inputs generate greater outputs [62]. This can be seen from the definition of the function itself, but it becomes especially clear when looking at the critical section of its graph, as shown in Figure 3.3. An important property of a monotonically increasing function f is that it preserves the order of inputs and outputs: if $x < y$ then $f(x) < f(y)$.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

Because the Sigmoid function is differentiable, monotonically increasing, and maps to the range $(0, 1)$, it is especially suitable for neural networks where such properties are crucial for effective learning [63]. For instance, its output can be interpreted as a probability-like confidence score in classification tasks, which makes it a natural fit for the output layers of binary classifiers.

One of the main purposes of an activation function is to introduce non-linearity to the model [64]. In a binary classification task, the data is generally not linearly separable, i.e. a single line does not suffice to label new unseen data. For example, consider the two cases presented in Figure 3.4. In Figure 3.4a, the data is easily separable through a single vertical line. However, in Figure 3.4b, the data follows a complex circular pattern which cannot be modeled linearly. To address this problem, there are several non-linear activation functions including Softmax, ReLU, and Tanh that are commonly used.

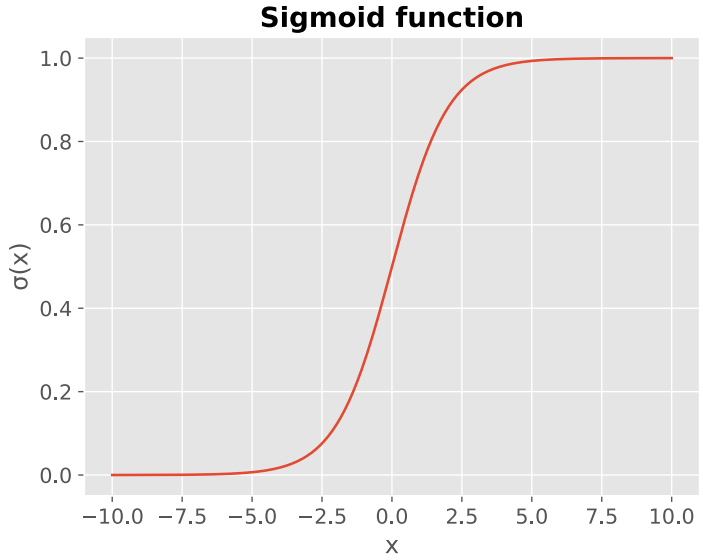


Figure 3.3: Sigmoid function over the range $[-10, 10]$. It is clear that $\sigma(x)$ approaches 1 as x approaches ∞. Dually, $\sigma(x)$ approaches 0 as x approaches $-\infty$

3.2.3 Layers

The final step to understanding neural networks is to grasp the layered architecture that combines their neurons. A single layer consists of a set of neurons, each with an associated weight and bias term, and an activation function. A layer takes the output from a previous layer as input (or the raw input data, in the case of the input-layer), and each neuron calculates its value before passing it through the activation function [59].

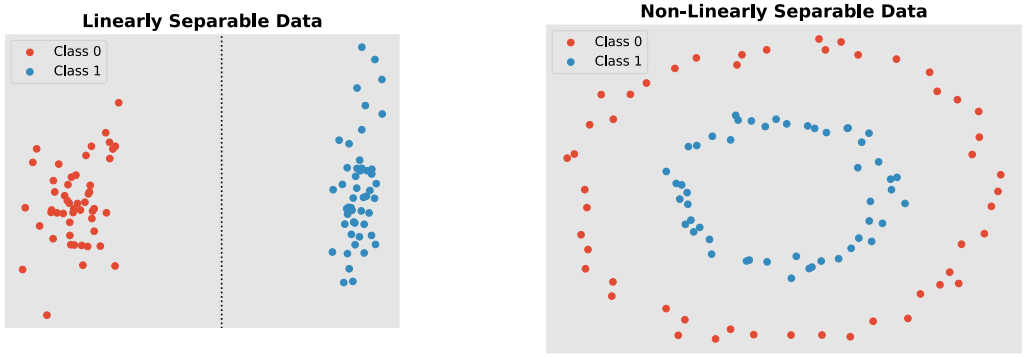
The simplest layer is a fully connected layer where all the neurons of the previous layer are connected to each of the neurons in the current layer. In mathematical terms, this means that the input to the layer, typically represented as a vector $\mathbf{x} \in \mathbb{R}^n$ of n features, is transformed using a weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ and a bias vector $\mathbf{b} \in \mathbb{R}^m$ where m is the number of neurons in the current layer. Each neuron computes a linear combination of the inputs as in Equation 3.5.

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{3.5}$$

This outputs a vector $\mathbf{z} \in \mathbb{R}^m$, which is then passed through an activation function ϕ applied element-wise, as shown in Equation 3.6

$$\mathbf{a} = \phi(\mathbf{z}) \tag{3.6}$$

Here, $\mathbf{a}$ represents the activated output of the layer, which is then passed to the next layer in the network. This linear transformation followed by a non-linear activation is what enables the network to learn and model complex patterns [59]. When multiple fully connected layers follow sequentially, the network can be viewed as a nested function, where each layer performs its transformation and feeds its result to the next



(a) Data can be classified as Class 0 or Class 1 by checking which side of the dotted line the point lies on.

(b) The data follows a more complex structure; a single straight line is not able to separate the data points.

Figure 3.4: Linearly separable data compared to non-linearly separable data.

layer. There are many other layers that are more complex than the fully connected layer. However, these are outside the scope of this thesis and will therefore not be reviewed in detail.

4

Experiment setup

Performing a thorough technical evaluation of the HE and FE encryption schemes in an OT IDS setting required a combination of practical and theoretical steps. This chapter details the practical minutiae of that process. It covers the experiment scenarios used, the experimental setup, the metrics evaluated, the choice and modification of data, and the hardware specifications of the test environment. The aim of this chapter is to ensure that the experiments can be reproduced and to provide clear descriptions of the methodology used throughout the thesis.

4.1 Experiment scenarios

To compare the HE- and FE-approaches in a systematic way, three experiment scenarios were designed. These scenarios represent how non-encrypted, HE-based, and FE-based IDS systems can be implemented in real-world situations. They also constitute the context in which data was gathered for the performance evaluation. This section introduces the scenarios and provides illustrations to give an intuitive understanding of how data flows through the systems.

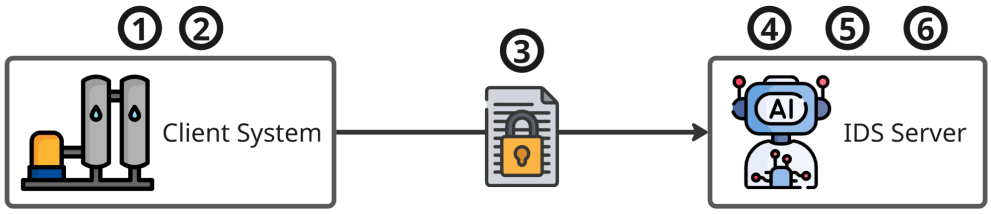
4.1.1 Baseline scenario

The first experiment scenario explores how an external IDS is typically implemented. It also serves as the baseline against which the upcoming HE- and FE-based scenarios are compared. In this scenario, a client OT system wants to procure the services of an external IDS provider. There can be various reasons for this. Often, it is because the client system is too computationally weak to run an IDS, or the client organization lacks the resources or personnel to implement one. Figure 4.1 presents the flow of data for this scenario.

The baseline scenario consists of six steps. The process begins with the client system monitoring and recording its runtime data. The data is then encrypted using a standard encryption scheme, typically the Advanced Encrypted Standard (AES) within the TLS protocol. The encrypted data is then sent to the IDS server.

Once the IDS server receives the encrypted data, it decrypts it to its plaintext form. The data is then evaluated, typically using a pre-trained detection model or algorithm. If an anomaly is detected, the IDS server can then execute subsequent defensive actions. With that, the baseline scenario is complete.

Importantly, although the data is encrypted during transit to protect it from other external parties, the IDS server itself has full access to the decrypted data. Although this might seem like an innocuous prerequisite to enable the external IDS, it introduces significant security vulnerabilities. First, the external IDS server may be a malicious actor masquerading as a trusted business partner. An unknowing client system could then be sharing highly sensitive data directly with a malicious actor, potentially enabling sophisticated and severe attacks. Second, the external IDS server might be honest, but could fall victim to an external breach. In such a case, sharing the client system's data could inadvertently result in a data leak. If the client system's data is sensitive, which is often the case with critical OT infrastructure such as water treatment facilities, this could enable highly sophisticated attacks causing large-scale societal damage. To prevent these issues, two alternative scenarios which enable IDS while maintaining data privacy are considered.



Baseline Scenario

1. Client System records data
2. Data is encrypted normally (AES or similar)
3. Encrypted data is sent to IDS Server
4. IDS Server decrypts data
5. IDS Server evaluates data
6. IDS Server triggers defensive measures in case of anomaly

Figure 4.1: Explanation of the data flow in the baseline scenario. Note that the data is decrypted and available to the IDS server in steps 5 and 6.

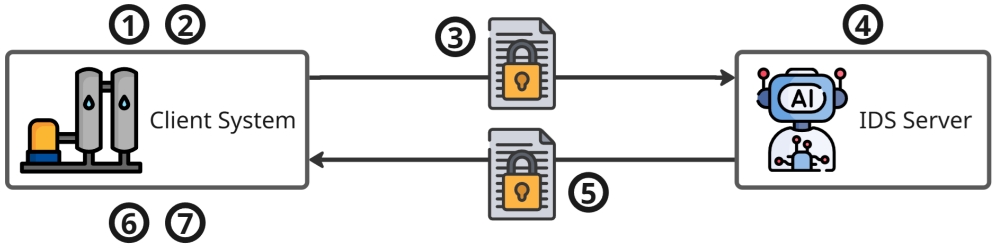
4.1.2 Homomorphic encryption scenario

The second scenario explored builds on the CKKS homomorphic encryption scheme described in Section 3.1.2. This scenario consists of seven steps, as depicted in Figure 4.2. The initial step involves the client system recording the data and encrypting it using `CKKS.Encrypt`. Next, the client system sends the homomorphically encrypted data to the IDS server for processing. Note that the IDS server does not have access to the decryption key, which ensures the confidentiality of the client system's data.

In the fourth step, the IDS server evaluates the data using an HE-compliant IDS. Since the IDS server is unable to decrypt the result of the model, the result must be returned to the client system, as per step five. The final steps consist of the client system decrypting the result using `CKKS.Decrypt` and triggering defensive measures

in the case of an anomaly. With this, the HE scenario is complete.

Note that the HE scenario is different from the baseline and the FE scenarios in that the result is interpreted by the client system. Depending on the context, this is not necessarily the optimal solution. However, this architecture represents the minimum required steps and does not preclude extensions. In fact, one can image an eighth step to Figure 4.2, where the decrypted results are sent to the IDS server for interpretation and potential response actions.



HE Scenario

1. Client System records data
2. Data is encrypted using HE
3. HE data is sent to IDS Server
4. IDS Server performs homomorphic computations (neural network evaluation)
5. IDS Server returns result of computation
6. Client System decrypts result
7. Client System triggers defensive measures in case of anomaly

Figure 4.2: Explanation of the data flow in the HE scenario.

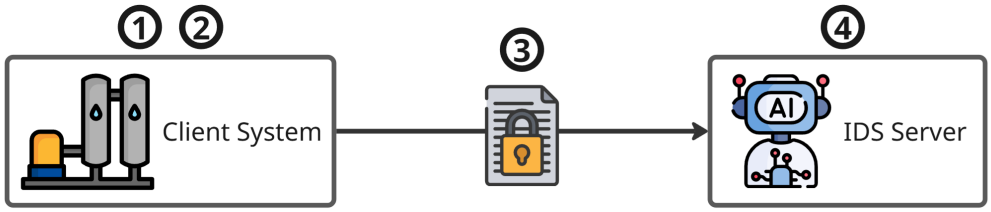
4.1.3 Functional encryption scenario

The third and final scenario uses the FHIPE system explained in Section 3.1.3, and consists of two phases: a setup phase and a runtime phase.

FE Scenario Setup

The FE scenario setup phase consists of four steps, as seen in Figure 4.3. To start, the client system must construct a vector where each element represents the "normal" state of a feature of the system. In the case of a water treatment facility, this could entail recording the amount of water in a container, the amount of time that a gate is opened or closed, etc. In practice the normal state is typically obtained by recording the system over a period of time and computing relevant averages.

The second step of the FE setup procedure commences once the client system has produced a normal-state vector. The vector is then encrypted using the `IPE.KeyGen` function and sent to the IDS server. Importantly, this encrypted normal-state vector reveals no information about the system to the IDS server. Once the IDS server



FE Scenario Setup

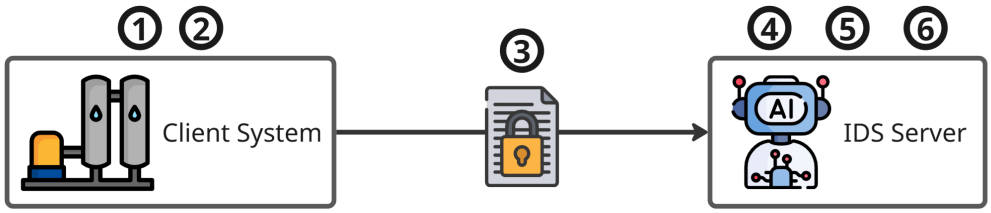
1. Client System records normal-state of monitored features
2. Client System encrypts normal-state vector using FE
3. Encrypted normal-state vector is sent to IDS Server
4. IDS Server persists encrypted normal-state vector

Figure 4.3: Explanation of the data flow in the FE scenario setup phase.

receives the encrypted vector, it stores it for use during runtime. With this the setup process is complete.

FE Scenario Runtime

The FE scenario runtime consist of six steps, as seen in Figure 4.4. The process begins with the client system monitoring and recording its runtime data. The data is then encrypted using the `IPE.Encrypt` function and sent to the IDS server. Note that this encrypted vector reveals no information about the current system status to the IDS server.



FE Scenario Runtime

1. Client System records data
2. Client System encrypts data using FE
3. FE data is sent to IDS Server
4. IDS Server computes distance between FE data and encrypted means
5. IDS Server evaluates distance
6. IDS Server triggers defensive measures in case of anomaly

Figure 4.4: Explanation of the data flow in the FE scenario runtime phase.

Once the IDS server receives a vector of encrypted values, it uses the `IPE.Decrypt` function. This function takes in the encrypted data and the encrypted normal-state

vector obtained during the setup phase, and computes the L_2-distance between them. If the distance is too great, the IDS classifies it as an anomaly, enabling the execution of subsequent defensive actions. Note that the IDS server is thus able to determine deviations from the normal state in the client system, without learning anything about the system itself other than the distance between the client system's current state and its normal state. With this, the FE scenario is complete.

4.2 Intrusion detection model implementation

To make a fair and reliable comparison between the HE and FE intrusion detection models, a non-privacy-preserving baseline model that operates strictly on plaintext data is required. The results from the baseline model serve as a benchmark from which the comparison can be made. While the neural network architecture used in the intrusion detection model is identical across the baseline, HE, and FE scenarios, their implementation and training differ in significant ways. All models are implemented in Python, using the libraries presented in Section 2.5.

4.2.1 Implementation of the baseline model

The underlying model of the IDS is a shallow neural network. The model consists of two layers with separate activation functions, as displayed in Figure 4.5. Only two layers were chosen to balance the model's accuracy with runtime performance, given the high computational cost of homomorphic encryption. The aim is not necessarily to create the best-performing PPIDS, but to compare HE- and FE-powered PPIDSs in an equal setting.

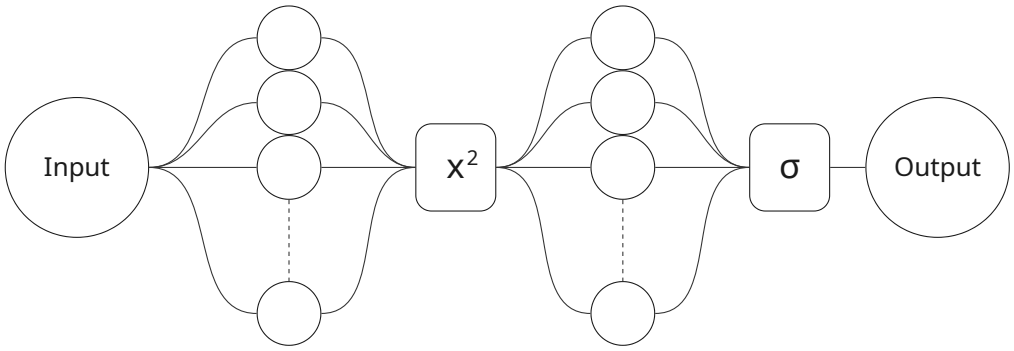


Figure 4.5: The neural network consists of two fully connected layers with a square activation between them. The final step is the Sigmoid activation function.

There are two criteria for the choice of activation functions: (*i*) the function needs to introduce non-linearity to the model, and (*ii*) the function must be compatible with the CKKS homomorphic encryption scheme, i.e. it must be representable as a polynomial. The input layer is a fully connected layer with an associated square

activation function, and since the square function is a polynomial of the second degree, the criteria are fulfilled.

Following the input layer is another fully connected layer, which is the final layer of the model. Its output is used to classify the data as either an attack or normal behavior. Therefore, the Sigmoid function is chosen as the activation since it compresses output values to the range $(0, 1)$. The label of a new data point p is contingent on the relation between the output a_p and some threshold t. If $a_p \geq t$, then p is labeled as an attack, otherwise, it is considered non-malicious. The threshold t is optimized by analyzing the ROC curve, as described in Section 4.3.2.

Apart from the architecture, there are other common traits of the models across the scenarios. Since the classification is binary, the loss of each epoch is calculated using binary cross-entropy (BCE). The models use stochastic gradient descent (SGD) with a learning rate of 0.004 to optimize the loss function and are trained for 4000 epochs. Lastly, the number of neurons in the input layer is equal to the number of features in the dataset. Following is a description of each of the models built for the encrypted experiments in this thesis.

4.2.2 Implementation of the homomorphic encryption model

The key difference between the HE model and the baseline model lies in the use of homomorphic encryption: the HE model leverages homomorphic tensor operations provided by the TenSEAL library, whereas the baseline model uses standard PyTorch tensors. Importantly, the HE model is constructed by extracting the trained weights and biases of the baseline model, allowing it to perform inference on encrypted data without retraining it. While it is technically possible to train the model directly on homomorphically encrypted data, this introduces significant overhead and is considerably more time-consuming. Therefore, for this thesis the model is trained on plaintext data and then translated to an HE-friendly model.

Another important distinction between the HE model and the baseline model is the implementation of the final activation, the Sigmoid function. In Section 4.2, two criteria for the activation function were presented, one being that the function needs to be representable as a polynomial. However, the Sigmoid function as presented in Equation 3.4 is not a polynomial. Chen et al. [65] solved this problem by using a polynomial approximation of the Sigmoid function. While this is possible, it introduces additional homomorphic multiplications, which in turn introduce more computational overhead. To avoid the additional computations, the architecture was changed slightly by moving the responsibility for the Sigmoid function to the client instead of the server. That is, the server returns the value after the final fully connected layer, but before the activation function. This setup is considered acceptable, as any client device capable of performing HE encryption is also capable of applying the Sigmoid function.

As mentioned in Section 3.1.2, the CKKS scheme requires several parameters to be set. Setting these parameters is difficult and there is no precise methodology to follow. However, the Homomorphic Encryption Standard [66] developed by a

consortium of industry, government and academia, provides the foundation for parameter selection. There are two criteria for the parameters: (i) they should allow for a certain computational depth, and (ii) the security level should be at least 128-bits.

Starting with criterion (i), the computational depth: the model contains two layers and only one activation function (Sigmoid is applied after decryption). As a result, each individual ciphertext is subject to three homomorphic operations. This means that three coefficient moduli are required, plus the additional two used for encryption and decryption, resulting in a total of five coefficient moduli.

Regarding criterion (ii), the level of security is dependent on the coefficient modulus q and the dimension n of the underlying ring. Higher dimension n provide better security, but it also makes the encrypted operations more expensive. To provide sufficient security and enough computational depth, the ring dimension $n = 8192$ was chosen, along with an encryption/decryption modulus of 40 bits and a coefficient modulus of 21 bits. This results in $q = 40 \cdot 2 + 21 \cdot 3 = 143$. According to the Homomorphic Encryption Standard, the pair (n, q) yields a security level of over 128 bits, satisfying both criteria.

4.2.3 Implementation of the functional encryption model

The FE detection model is similar to the baseline model, but differs in some crucial aspects. In the baseline model, the raw feature data is fed into the neural network for training and inference. However, due to the nature of the FHIPE system introduced in Section 3.1.4, the input data to the neural network in the FE scenario is not the raw data. Instead, it is the distance between the system's current state and its normal state. In this thesis, this distance is defined as the L_2-distance between a vector of the system's current state and the corresponding normal-state vector obtained during the FE setup stage, as described in Section 4.1.3.

In general, the FE model works by loading the raw feature data, extracting a normal-state vector, and comparing each row to the normal-state vector to compute an L_2-distance. The resulting matrix of distances is then fed into the neural network as the basis for training and inference. This means that, although the input is the same in all three scenarios, the detection model in the FE scenario uses the derived distances for both training and inference rather than the raw data. While the general structure of the data is preserved—since natural fluctuations in values are still reflected as deviations from the normal state—this represents a difference in how the models are constructed.

Importantly, the distance matrix that serves as input for the FE neural network can vary in size depending on how the distances are computed. For example, consider an $n \times m$ input data set A with n rows and m columns. Also consider the corresponding $1 \times m$ normal-state vector y consisting of the averages of each column in A. Each row r_i in A can be used to generate a singular distance by computing the L_2-distance between r_i and y. This distance is given by

$$L_2(r_i, y) = ||r_i - y||_2^2 = ||r_i||_2^2 + ||y||_2^2 - 2\langle r_i, y \rangle$$

where the $||\cdot||_2$ denotes the L_2-norm. This can be simplified by modifying r_i and y

$$\begin{aligned} r'_i &= \{||r_i||_2^2, -2r_{i1}, \ldots, -2r_{im}, 1\} \\ y' &= \{1, y_1, \ldots, y_m, ||y||_2^2\} \\ L_2(r_i, y) &= \langle r'_i, y' \rangle \end{aligned}$$

This means that the resulting distance matrix

$$D = \begin{bmatrix} \langle r'_1, y' \rangle \\ \vdots \\ \langle r'_n, y' \rangle \end{bmatrix}$$

only contains a single column with the same number of rows as the raw feature data. Note that each value in D represent the distance between a row in A and the normal-state vector y.

Another approach is to split the raw feature data A into multiple parts $A_1, \ldots, A_N$ where $N \in \mathbb{N}$, each containing a fixed number of columns. Each part A_j can then generate its own column of distances D_j using its own normal-state vector y_j. The resulting distance matrix D then has as many columns as there are subdivisions A_j. For example, assuming A is divided column-wise into three parts A_1, A_2 and A_3, this would mean that the resulting distance matrix D would be given by

$$D = \begin{bmatrix} D_1 & D_2 & D_3 \end{bmatrix} = \begin{bmatrix} \langle r'_{11}, y'_1 \rangle & \langle r'_{12}, y'_2 \rangle & \langle r'_{13}, y'_3 \rangle \\ \vdots & \vdots & \vdots \\ \langle r'_{n1}, y'_1 \rangle & \langle r'_{n2}, y'_2 \rangle & \langle r'_{n3}, y'_3 \rangle \end{bmatrix}$$

where r_{ij} is the i-th row of A_j and y_j is the normal-state vector corresponding to A_j.

This highlights one of the key properties of the FE approach: depending on the type of data monitored, one can "combine" raw feature columns into distance columns arbitrarily using the properties of the inner product operation. As presented in Chapter 5, the choice of how the distance matrix is computed can heavily impact the performance of the system.

This thesis considers three run configurations for generating the distance matrix used as input data to the FE detection model. As depicted in Figure 4.6, the first is referred to as the whole-row configuration. In it, no subdivisions of the dataset are made. Thus, the distance matrix consists of one column which represents the distance between each whole row r_i and the full normal-state vector y.

The second approach to generating the distance matrix is referred to as the modular run configuration. In it, the raw feature data is divided into six parts, each containing columns corresponding to the six modules of the SWaT dataset [14]. Because of this, the resulting distance matrix consists of six columns, each of which represents the distance between a module and its normal state.

The last approach to generating the distance matrix is referred to as the one-by-one run configuration. In it, the raw feature data is divided into parts that contain only one column each. This means that each column in the distance matrix represents

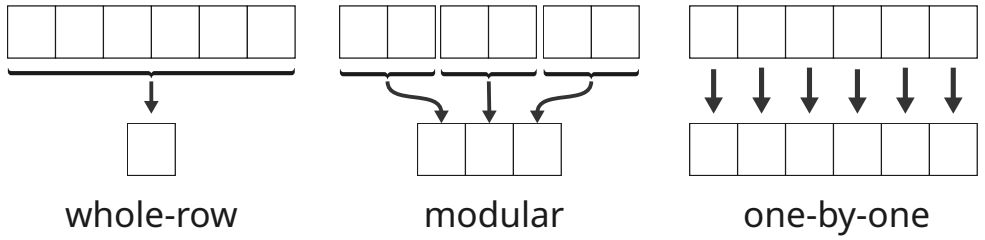


Figure 4.6: Visualization of the three run configurations used as input to the FE detection model.

the distance between the corresponding raw feature column and its column mean. Because each element in the original matrix is evaluated, the resulting distance matrix has the same dimensions as the original raw feature data.

Since the distance matrix has different dimensions depending on whether the whole-row, modular, or one-by-one run configurations is used, three neural network models are trained on the corresponding data. While the initial input size has differing shapes, all other parameters are identical to those of the baseline neural network.

4.3 Metrics

The overarching objective of this thesis is to investigate the effects of introducing HE- and FE-based techniques in an OT IDS setting. To accomplish this, the thesis uses quantitative and qualitative metrics to capture a broad picture of how the systems are affected in each scenario. This section describes these metrics and their application in the thesis, starting with speed, accuracy, network traffic, memory consumption, and finally, interpretability.

4.3.1 Speed

The time between an attack and the subsequent system alert can be the difference between a close call and a multi-million dollar breach. Consequently, the speed at which detection occurs is critical. For this reason, one of the key metrics evaluated in this thesis is how the addition of the cryptographic methods impacts detection speed.

There are several aspects to consider when measuring the speed of an IDS implementation. Consider the baseline scenario: the client needs time to gather and encrypt the data, the request from the client to the server takes time, and the server needs time to decrypt and evaluate the data. For the purposes of this thesis, the areas of interest are the ones that are impacted by the addition of the cryptographic protocols. More specifically, this thesis measures the time needed to encrypt, decrypt, process, and run the inference models on the data.

The timekeeping considers two aspects: the wall time and the CPU time. The wall time is captured using Python's built-in `time.time()` function, which measures the

wall time passed between two states in the program. The CPU time is captured in a similar way using the `Process.cpu_times()` function in the `psutil` library.

For each process measured, 1,000 recordings are collected and averaged using a randomly selected sample of 1,000 rows from the dataset. The only deviation from this process is the `IPE.Setup` function presented in Section 3.1.4, as its runtime is significantly longer than the other processes. For this reason, `IPE.Setup` is measured using only 50 recordings.

4.3.2 Accuracy

Encrypting data before it is sent to the IDS system is useful with regard to privacy, but it might degrade the detection ability of the model. This is particularly true for the FE approach since it, depending on feature selection and setup, might combine several features into one—possibly losing information. As such, one of the metrics considered is the accuracy of the system in all three scenarios.

The accuracy metric is broken down into three distinct methods. The first is the area under the curve (AUC) of the receiver operating characteristic (ROC). The second and third are extracted from the confusion matrix, which includes false-positive (FP), true-positive (TP), false-negative (FN), and true-negative (TN) outcomes observed during inference. These two methods are the model-accuracy and the F1-score. Model-accuracy is the inference results of the model that were labeled correctly, calculated as in Equation 4.1.

$$\text{Model-accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{4.1}$$

However, since model-accuracy can be misleading in binary classification tasks where one class heavily outweighs the other, the F1-score is also considered. The F1-score is the harmonic mean between precision and recall. Precision is the proportion of correct positive predictions, and recall is the proportion of actual positives that were correctly predicted. These are calculated as in Equation 4.2 and the F1-score is subsequently obtained as in Equation 4.3.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.2}$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.3}$$

The ROC is computed using the true positive rate (TPR) and the false positive rate (FPR) of the model, as given in Equation 4.4. While TPR reflects how well the model detects actual positives, FPR indicates how often the model mistakenly classifies negatives as positive [67]. The ROC curve is obtained by plotting the TPR on the y-axis and the FPR on the x-axis for a variety of thresholds. The perfect model that always labels data correctly would have a ROC curve passing through the point $(0, 1)$, corresponding to a FPR of zero and a TPR of one. The purpose of the ROC curve is to analyze different decision thresholds of a classification model; the threshold can be optimized to find a balance between high TPR and low FPR.

To summarize this performance trade-off into a metric independent of any specific threshold, the ROC curve can be integrated to find the area under the curve (AUC). A higher AUC indicates a better overall classification ability of the model.

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{4.4}$$

The confusion matrix is created by running the trained detection models on all rows in the dataset. These runs are executed with a decision threshold given by the point on the ROC curve that is closest to the optimal point $(0, 1)$, for the respective scenarios. The outcomes of the model predictions are then aggregated and compared to the dataset's ground truth labels, which indicate whether an attack is active. The results of these comparisons are then used to generate the confusion matrix.

4.3.3 Network traffic

Introducing the encryption schemes provides security benefits, but doing so may also affect the amount of traffic transmitted through the system. As such, this thesis considers a metric measuring the impact of the encryption schemes on data flow. In practical terms, the network traffic metric involves comparing the payload size of plaintext and encrypted data.

In the baseline scenario, a random sample of 1,000 rows is taken from the dataset. Each row is then serialized and its byte length is measured. The resulting sizes serve as the baseline amount of data that needs to be sent over the network.

To determine the additional network cost of using the HE and FE schemes, the sample of 1,000 rows is encrypted using HE and FE respectively. For FE, the encrypted data is serialized and the resulting size is recorded. The place of this measurement is shown in Figure 4.7 and it is identical to the baseline scenario. The process is similar for the HE-encrypted data, but as explained in Section 4.1.2, the HE scheme requires that the result of the IDS evaluation is passed back to the client for decryption, as depicted in Figure 4.8. Thus, the size of the returned data is also measured. Having acquired the plaintext data, encrypted FE data, pre-computation HE data, and post-computation HE data, the sizes are aggregated, averaged, and then compared.

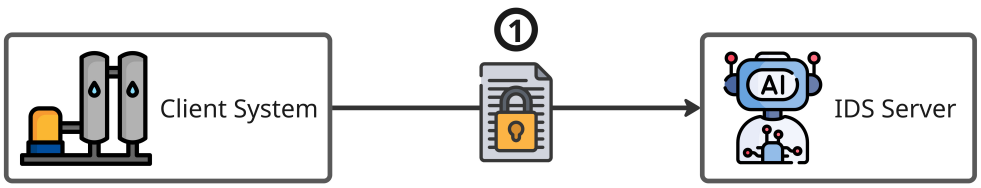


Figure 4.7: Visualization showing where the network traffic for the baseline and FE scenarios is measured.

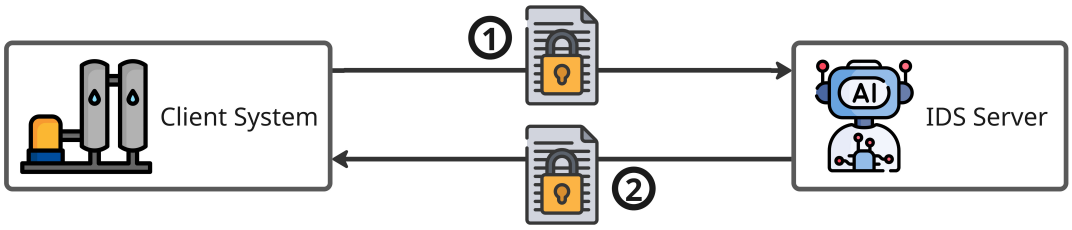


Figure 4.8: Visualization showing where the network traffic for the HE scenario is measured. The network traffic for HE is measured at 1 and 2 as the result of the IDS server must be returned to the client for decryption.

4.3.4 Memory consumption

Encrypting the data before detection improves security, but the encryption process also introduces additional memory cost. This cost can be detrimental for OT components, since they often have limited computational resources. To evaluate this aspect, this thesis uses a metric to measure the memory consumption in each experiment scenario.

When measuring the computational cost of the models, only the phases that involve some aspect of the cryptographic schemes are considered. This choice is based on the assumption that any system using an IDS will already require computational resources to run, log, and transmit data. Similarly, the IDS itself requires memory and computational power to function. Both of these aspects are relevant for a real implementation, but this thesis focuses on the costs and benefits of adding cryptographic schemes to IDS systems, not the IDS systems themselves. As such, the main interest lies where the cryptographic schemes add to the computational costs, not the overall system costs.

For the HE approach, the computational cost is measured during the inference phase and the two functions `CKKS.Encrypt` and `CKKS.Decrypt`, as these are the parts of the HE scenario that differ from the baseline scenario. Each of these processes are run on the same sample of 1,000 rows of the dataset.

For the FE approach, memory consumption and computational cost are measured during inference and during the four methods `IPE.Setup`, `IPE.KeyGen`, `IPE.Encrypt`, and `IPE.Decrypt`, as introduced in Section 3.1.4. To gather the measurements, each method is run on each row of a sample of 1,000 rows from the dataset. The only exception to this is the `IPE.Setup` function due to its significantly longer runtime. For this reason, `IPE.Setup` is measured using only 50 rows.

The memory consumption readings are captured using the `Process.memory_info()` function available in the `psutil` Python library. While the various methods are executing, the `memory_info` function is applied in a sampling-based approach to estimate peak memory consumption. An exact specification of the code used is

available in Appendix A.

4.3.5 Interpretability of results

In the case of a detected attack, the information provided by the IDS system might be crucial for subsequent defensive actions. For example, knowing where in the system an attack has occurred, or when it began, may significantly reduce response time. To evaluate such questions, this thesis analyzes the HE and FE models with regard to the interpretability of their results.

It is important to note that this metric, unlike the others, is qualitative in nature. Instead of measuring properties like magnitude or speed, the interpretability metric investigates the system's auxiliary characteristics. To structure this analysis, the models are evaluated against the following set of questions.

- In case of detection, can the deviating component of the system be identified?
- If the attack is previously known, can it be identified to predict subsequent effects?
- Can the system supply additional metadata to provide contextual information, which might be important for decision making?
- Can the system run when using computationally weak client side devices?

The analysis of the qualitative questions was performed separately from the performance measurements. To ensure the validity of the conclusions, the questions and written responses were submitted to the thesis supervisor, as well as industry partners, for review and feedback.

4.4 Dataset

There are many datasets that have been used for IDS purposes in the literature [4]. However, the majority of these are focused on network data and may not generalize well to OT scenarios. For this reason, this thesis uses the Secure Water Treatment (SWaT) dataset provided by iTrust, Centre for Research in Cyber Security, Singapore University of Technology and Design [14].

The SWaT dataset was collected from a comprehensive testbed that monitors the function of a scaled-down water treatment facility in Singapore [14]. The data was recorded over a period of 11 days, during which the testbed was running non-stop. There were no attacks executed during the first seven days; however, during the last four, a total of 36 attacks took place. The attacks varied in nature and lasted between a few minutes and an hour. In total, the dataset contains 946,722 samples over 51 attributes.

Preparing the SWaT dataset for analysis requires several steps. These are detailed in the list below.

- **Data-loading:** The raw SWaT data is provided as `CSV` files. These are loaded into a Jupyter Notebook-based Python environment using the Pandas library.
- **Column removal:** Once loaded into the Pandas DataFrame, the dataset is cleaned by removing constant columns, as they do not contribute to inference.
- **Decimal scaling:** Since the FHIPE system utilized in the FE approach only accepts integers, all columns are scaled and rounded in the FE scenario. The columns are scaled and rounded to only contain integers between 20 and 200. These bounds are somewhat arbitrary, but are selected to provide a large enough interval to allow nuanced detection of deviations, while still being small enough to not severely impact the computational time of the FE scheme. This is relevant since the computational difficulty of the discrete logarithm explained in Section 3.1.3 grows quadratically with the size of the largest value in the dataset.

4.5 Hardware

All experiments were carried out on a personal computer with the following specifications:

- **CPU:** Intel(R) Core(TM) i7-14700K 3.40 GHz
- **RAM:** 32,0 GB
- **GPU:** Nvidia GeForce RTX 4070 SUPER (ASUS)
- **SSD:** Samsung 990 PRO M.2 NVMe SSD 2TB
- **HDD:** Seagate BarraCuda Desktop 2TB 7200rpm 256MB

Windows Subsystem for Linux (WSL) was leveraged to enable Linux functionality on a Windows system. The operating systems used were the following:

- **Host OS:** Windows 11 Home
- **WSL:** WSL2
- **WSL Distribution:** Ubuntu 22.04.5 LTS
- **Python Version:** Python 3.13.2

In addition to the above, due to compatibility issues in the Charm cryptographic library, the FE system was run inside a docker image using the following specs:

- **Docker Engine:** 28.0.1
- **Linux Distribution:** Ubuntu 14.04.6 LTS
- **Python Version:** Python 3.4.3

5

Results

To accomplish the goal of thoroughly evaluating the HE and FE schemes in an OT IDS setting, several experiments were carried out. This chapter presents the outcomes of these experiments, and attempts to communicate them in a way that is both accessible and comprehensible. The results for the three scenarios described in Section 4.1 are presented metric by metric following the structure shown in Section 4.3: starting with the speed metric, then the accuracy, network, and memory metrics. Due to its qualitative nature, the interpretability metric is postponed to Section 6.5.

5.1 Speed

The complete outcome of the experiments with regard to the speed metric is presented in Table 5.1. For most processes, the wall time for all three scenarios is quite low, often at or below single digit milliseconds. The processes that deviate from this are generally the most computationally demanding parts of the system: generating large matrices and determinants in the FE Setup stage, solving bounded discrete logarithms in the FE decryption process, or performing inference on encrypted HE data.

With regard to CPU time, Table 5.1 shows that user time is greater than system time for all processes. This implies that most time is spent working on the active process rather than I/O or similar.

When comparing the wall time to the CPU time, there is a relatively large spread of outcomes. For most processes, the wall time is quite close to the user time, but there are also several cases when the CPU time is either larger than, or smaller than, the wall time. For example, the wall time for performing inference on the HE encrypted data is significantly smaller than the CPU time, implying that the process is heavily parallelized for efficiency. Another example is the FE setup process, where the wall time is significantly higher than the CPU time. This implies that the majority of the time was spent waiting for some external process. This is expected, as many of the computations in the setup process are performed in external C code which is not monitored.

Another observation is that the time demand rises with the input size for all processes involving FE. This increase is most pronounced in the setup process, where the

difference between the whole-row configuration containing one feature, and the one-by-one configuration containing 44 features, exceeds a factor of 200. However, for most of the FE-related processes, the difference is not as large. For example, the difference in inference time for the FE run configurations is less than three microseconds.

Table 5.1: Mean timing results for processes in all three experiment scenarios. User/System time refer to CPU time. All times are measured in milliseconds (ms).

Experiment	**Wall Time**	**User Time**	**System Time**
Baseline Inference	0.213	0.170	0.020
HE Decrypt	0.281	0.390	0.000
HE Encrypt	3.555	3.220	0.120
HE Inference	54.150	761.110	3.360
FE Inference (one-by-one)	0.210	0.210	0.010
FE Inference (modular)	0.209	0.340	0.010
FE Inference (whole-row)	0.181	0.240	0.000
FE Setup (one-by-one)	7718.324	145.667	48.000
FE Setup (modular)	237.633	27.667	1.000
FE Setup (whole-row)	37.607	27.000	1.333
FE KeyGen (one-by-one)	3.056	2.830	0.100
FE KeyGen (modular)	0.833	0.740	0.060
FE KeyGen (whole-row)	0.575	0.380	0.040
FE Encrypt (one-by-one)	18.788	17.387	0.470
FE Encrypt (modular)	9.319	3.120	0.130
FE Encrypt (whole-row)	1.617	1.330	0.110
FE Decrypt (one-by-one)	101.383	92.889	0.883
FE Decrypt (modular)	18.844	19.230	0.220
FE Decrypt (whole-row)	9.382	9.020	0.120

5.2 Accuracy

As described in Section 4.3.2, the three metrics that together quantify the performance of the models are the model-accuracy, the F1-score, and the AUC-score. The first two are gathered from the confusion matrix, which is generated by setting a classification threshold t. This threshold was selected by analyzing the ROC curve and choosing the point closest to the top-left corner (i.e., highest TPR and lowest FPR). Although highly contextual, these thresholds will hereby be referred to as optimal for each model. The ROC curves for all the models are displayed in Figure 5.1, along with their AUC-score and optimal threshold.

An immediate observation from Figure 5.1 is that the ROC curves and AUC-scores are very similar for the baseline, HE, and FE one-by-one modes. Furthermore, the modular and whole-row modes of the FE approach achieve notably lower scores than the other models.

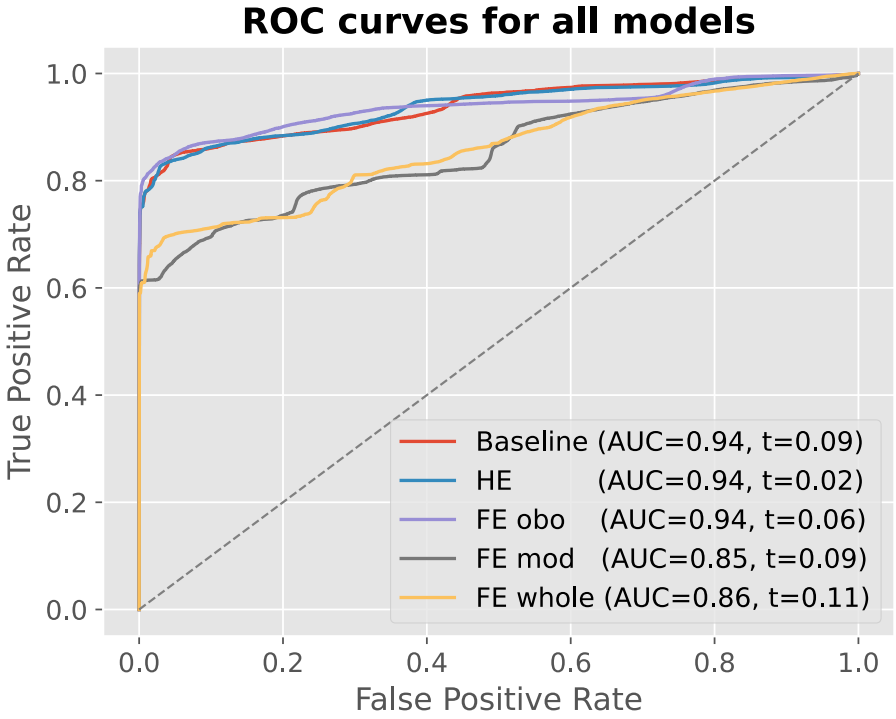


Figure 5.1: The ROC curves for all the models, along with their respective AUC and optimal threshold.

Following the ROC curves, the confusion matrices for the different models are presented. Figure 5.2 displays the results of the baseline model, Figure 5.3 presents the HE model, and Figure 5.4 includes all modes of the FE model. Lastly, the model-accuracy, F1-score, and AUC-score of each of the models are summarized in Table 5.2.

Table 5.2: The model-accuracy, F1-score, and AUC-score achieved by each model.

Model	**M-acc.**	**F1**	**AUC**
Baseline	0.931	0.75	0.94
HE	0.918	0.72	0.94
FE (one-by-one)	0.923	0.73	0.94
FE (modular)	0.861	0.56	0.85
FE (whole-row)	0.909	0.65	0.86

5.3 Network traffic

The network metric is based on analyzing the byte length of the data transported between the client system and the IDS server in each scenario. The full results of the experiments are displayed in Table 5.3. As explained in Section 4.1, the scenarios behind each model are slightly different. In the baseline, the data is only sent once

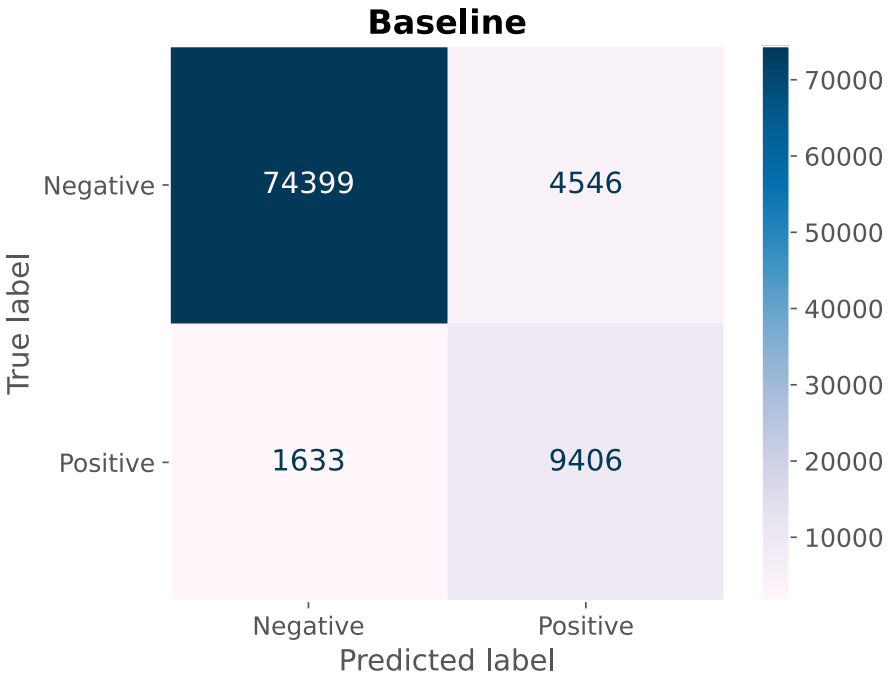


Figure 5.2: Confusion matrix containing the inference results of the baseline model.

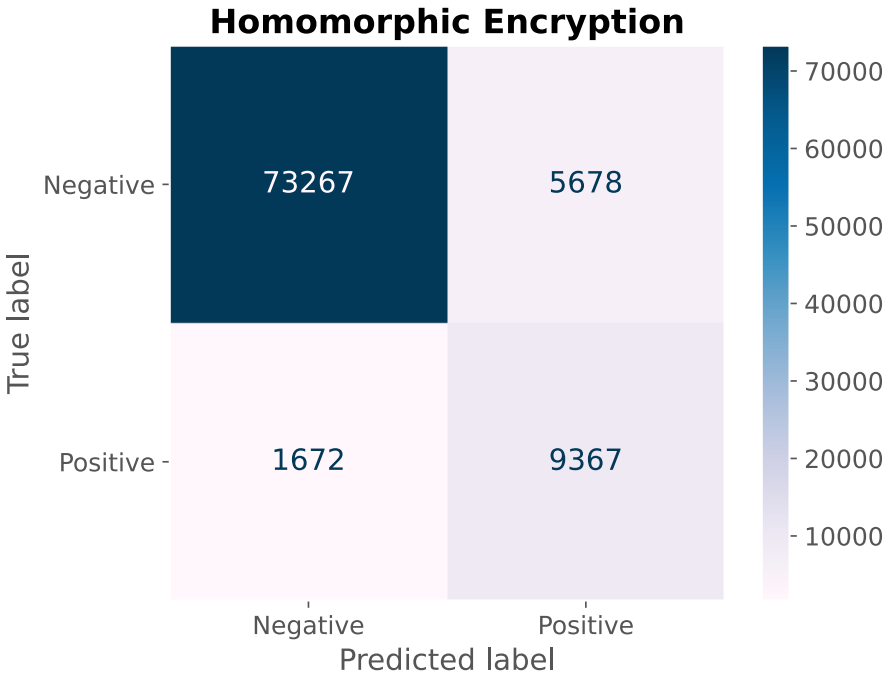


Figure 5.3: Confusion matrix describing the inference results of the homomorphic encryption model.

from the client system to the IDS server. However, in the HE scenario, the IDS server's result must be returned to the client for decryption. Therefore, both of these transfers are measured as to IDS server and from IDS server respectively.

The FE scenario is similar to the baseline in that there is only one step of data transportation. However, as introduced in Section 4.2.3, FE allows for flexibility in how the data is divided. Therefore, the experiments also include the whole-row, modular and one-by-one subdivisions.

The HE scenario has the most significant impact on network traffic, transmitting over 65 times more data to the IDS server compared to the second most bandwidth-intensive scenario, FE (44). Additionally, because the IDS result must be sent back to the client system for decryption, the total average network usage per data row reaches 365KB. This is nearly 100 times greater than that of the FE (44) scenario. The HE scenario is also the only experiment exhibiting a non-zero standard deviation in network traffic, indicating minor fluctuations in bandwidth usage over time. However, the variation is small and unlikely to have any significant impact on the overall performance.

Table 5.3: The average byte length and standard deviation for each time data is transported between the client system and the IDS server.

Experiment	**Avg. size**	**SD**
Baseline	176	0.0
HE (to IDS server)	270925	674.8
HE (from IDS server)	94355	42.8
FE (whole-row)	344	0.0
FE (modular)	774	0.0
FE (one-by-one)	4042	0.0

5.4 Memory consumption

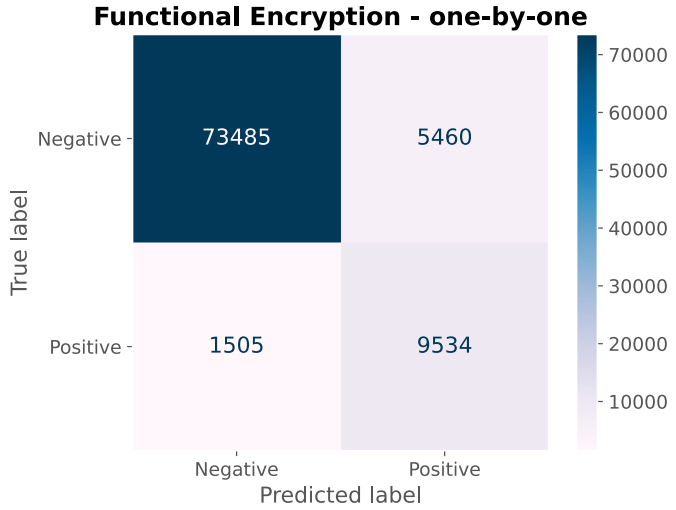
The outcome with regard to memory consumption is presented in Table 5.4. In general, most processes require very small amounts of memory. Most require less than 1 MB of memory, and all but one require less than 10 MB. Of particular note is the baseline inference which required 2.241 MB of memory which contrasts with the increased demands of 2.507 MB for the largest FE run configuration and 73.085 MB for the HE scheme.

While most processes have low memory consumption, one major outlier is the HE inference step. This result is plausible since the HE inference process involves complex linear algebra calculations that are heavily parallelized. It also points to the well-known idea that HE tends to be computationally demanding.

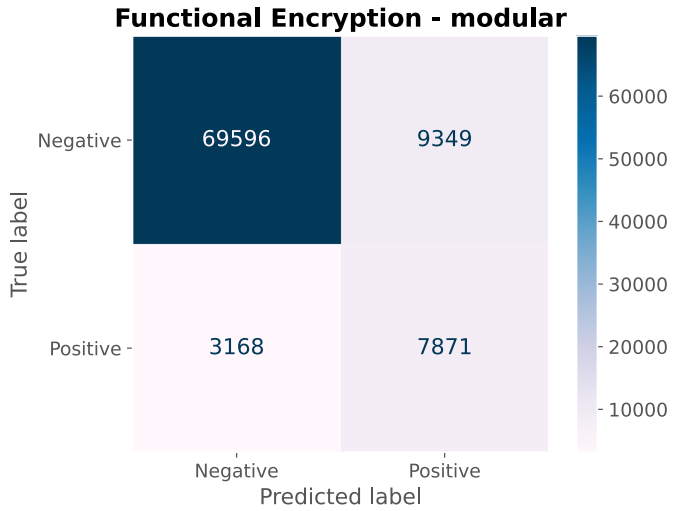
One thing to note is that the results for the FE Setup processes are somewhat incomplete. This is because, as explained in Section 5.1, the setup function leverages external C code which was not captured during the experiment stage.

Table 5.4: Peak memory usage for processes in all three experiment scenarios. All memory values are in megabytes (MB).

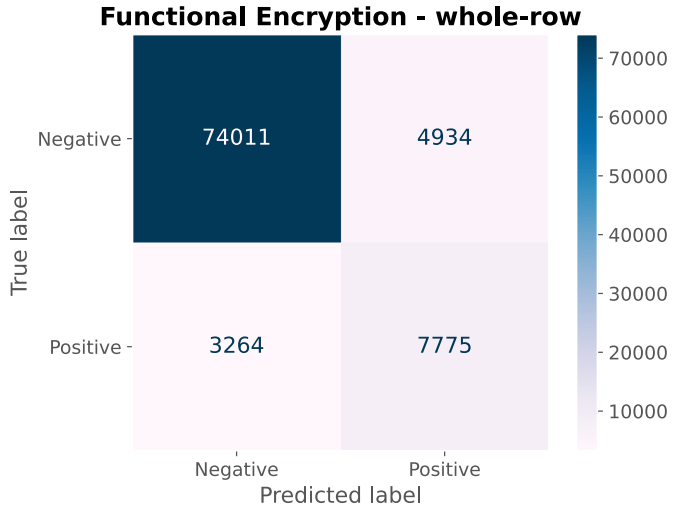
Experiment	**Memory (MB)**
Baseline Inference	2.241
HE Decrypt	0.541
HE Encrypt	3.953
HE Inference	73.085
FE Inference (one-by-one)	2.507
FE Inference (modular)	0.008
FE Inference (whole-row)	0.008
FE Setup (one-by-one)	2.146
FE Setup (modular)	1.671
FE Setup (whole-row)	1.491
FE KeyGen (one-by-one)	0.172
FE KeyGen (modular)	0.012
FE KeyGen (whole-row)	0.016
FE Encrypt (one-by-one)	0.078
FE Encrypt (modular)	0.082
FE Encrypt (whole-row)	0.082
FE Decrypt (one-by-one)	0.512
FE Decrypt (modular)	0.303
FE Decrypt (whole-row)	0.147



(a) Confusion matrix for the one-by-one mode.



(b) Confusion matrix for the modular mode.



(c) Confusion matrix for the whole-row mode.

Figure 5.4: The confusion matrices for the three FE modes.

6

Discussion

Chapter 5 presented the results of the thesis experiments. This chapter aims to analyze these results and provide insight into how introducing HE or FE affects an IDS operating in an OT environment. Initially, the results of the experiments are analyzed from the perspective of each metric presented in Section 4.3. After this, a summary analysis is given regarding the strengths and weaknesses of both HE and FE, as well as the possible ramifications of implementing either in a real-world scenario. The subsequent section discusses some ethical considerations relevant to this thesis. Lastly, several interesting avenues for further research are presented.

6.1 Speed

In general, the outcomes for both the HE and FE scenarios are surprisingly strong with regard to speed. For both approaches, the total wall time accrued from adding encryption, decryption, and in the case of HE, inference, is less than 125 ms. This points to a key finding: for many real-world systems, the time penalties of implementing either of the cryptographic schemes are quite small. There may be OT scenarios using automated defensive measures, such as automatic reactor shutdown in a nuclear plant, where an increase in detection delay of $\sim$125 ms could be catastrophic. However, many implementations rely on notifications that are monitored and manually handled by humans, and in such cases, the added milliseconds will be negligible from a crisis management perspective.

While the speeds of both cryptographic schemes are found to be comparable, that result is strongly connected to the choice of detection model. The shallow neural network presented in Section 4.2 was selected to enable fair comparisons across the full set of metrics considered. However, the model is very simple, and its accuracy could likely be improved by increasing the number of hidden layers and/or neurons. However, doing this would drastically impact the time required to perform the HE computations. A more complex model, which may offer better accuracy, could thus significantly increase the HE inference time. The same is not true for the FE scheme, as adding it does not impact the inference time in any significant way. Consequently, it is fair to say that the FE scheme has more flexibility with regard to the detection models it can interface with. The HE scheme, by its fundamental characteristics of requiring the client to decrypt the result, is more limited by the time demands of the homomorphic computations. If the detection model grows too complex, the HE

scheme may simply become too slow. This is discussed further in Section 6.8.

With regard to CPU time, several conclusions can be drawn from the data presented in Section 5.1. First, it is probable that most of the computational processes are not heavily parallelized. If they were, there would likely be significantly more CPU time than wall time, which is not the case for the overwhelming majority of processes. To exemplify this, one can consider the FE decryption process. While the decryption process consists of several steps, the majority of time is spent computing operations over the elliptic curve to find the exponent $z = \langle x, y \rangle$. Since this process is not easily parallelized, it is reasonable that the CPU time of 92.889 ms is close to the wall time of 101.383 ms. The one exception where parallelization is clearly visible is the HE inference which shows a wall time of 54.150 ms and a user CPU time of 761.110 ms. While this is a significant difference, it is not unreasonable since the process is heavy on matrix operations that are easily parallelizable. In conclusion, while performance improvements may be possible by optimizing both hardware and software, most processes do not appear to be easily parallelizable.

Another observation is that the computation time in the FE scenario rises with the length of input for all processes, but the speed at which the computation time grows differs greatly depending on the process. For example, we see in Figure 6.1 that the decryption wall time is 9.382 ms for the whole-row scenario, 18.844 for the modular scenario, and 101.383 ms for the one-by-one scenario. Given the corresponding length of input being 1, 6, and 44, this would suggest that the complexity is approximately linear, but more tests would be required to verify this. The FE setup process has wall times of 37.607 ms for the whole-row run configuration, 237.633 ms for the modular run configuration, and 7718.324 ms for the one-by-one configuration. Given the same input lengths of 1, 6, and 44, the steep increase in computational time suggests that this process exhibits superlinear or even exponential time complexity. Although this means that the setup process could be very slow if used in scenarios where the feature number is large, all processes active during runtime appear to scale reasonably as N increases.

The FE scenario was recorded in the docker environment described in Section 4.5. While this environment was hosted on the same system as the other experiments, basic testing showed that the docker environment was noticeably computationally slower than the regular WSL environment. This suggests that the FE scenario might perform better on a more modern system. However, further testing is needed to verify this.

6.2 Accuracy

Generally, it seems that adding the cryptographic schemes to the baseline model has a very limited negative effect on the accuracy of the IDS. At least this is true for HE and the one-by-one mode of FE, in which each feature is encrypted separately. As seen in the ROC curves displayed in Figure 5.1, it is apparent that the modular and whole-row modes of FE achieve considerably lower AUC-scores. While this was expected, the modular mode could likely be optimized by combining more heavily

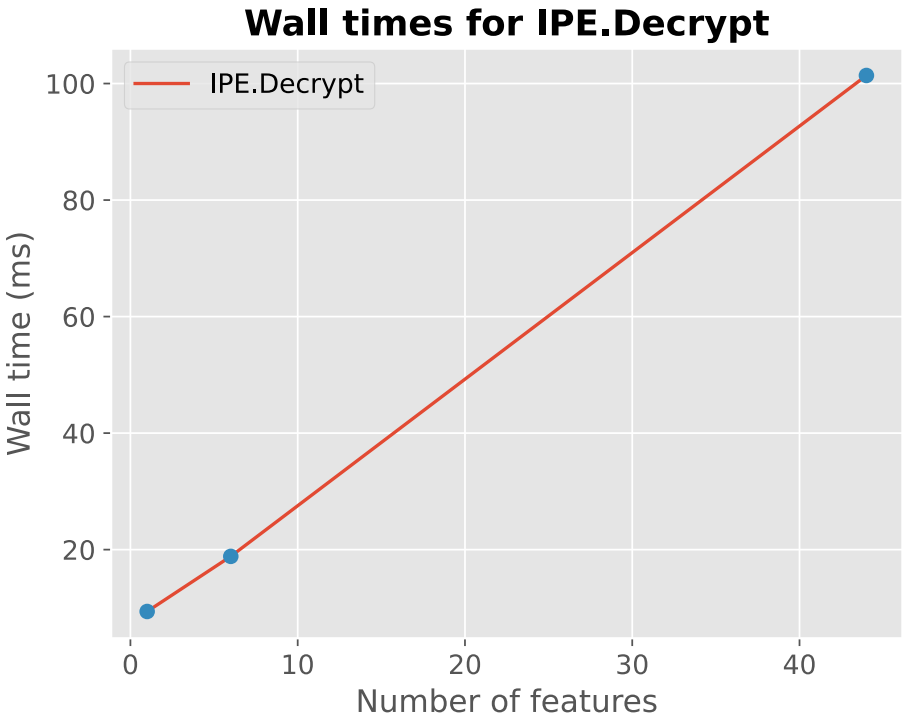


Figure 6.1: The ratio between the `IPE.Decrypt` wall time and the number of features appears to be approximately linear.

correlated features. However, this requires domain expertise and should be done in collaboration with specialists in the field.

The model-accuracy and F1-scores reported in Table 5.2 correspond to a specific threshold t. We chose t with the intention of maximizing the true-positive-rate and minimizing the false-positive-rate. This corresponds to the point on the ROC curve closest to the top-left corner, i.e. the point $(0, 1)$. This threshold was chosen to maintain a fair comparison between the models, but it is not necessarily the threshold corresponding to the greatest model-accuracy or F1-score. Depending on the context, and whether false positives or false negatives are more costly, the threshold can be adjusted to reduce either of these occurrences.

Analyzing the model-accuracies reported in Table 5.2, note that the HE and FE in the one-by-one mode achieve just slightly lower scores than the baseline. However, the AUC scores are identical at 0.94 for the three models. This indicates that their overall detection ability is equivalent, but for the chosen thresholds the baseline outperforms its encrypted counterparts slightly. This is also reasonable given the previous discussion of the threshold not being chosen to optimize model-accuracy.

Although a model-accuracy of 93% might seem impressive for such a simple model, note that 88% of the test-set is non-attack rows. This means that even the most rudimentary model, the one that labels everything as not an attack, obtains a model-accuracy of 88%. To avoid this pitfall, we instead investigate the F1-score, which is the harmonic mean between precision and recall, as explained in Section 4.3.2.

The F1-score of the three top models indicates that the performance is not quite as strong as the model-accuracy suggests. However, the similarity of F1-scores across the models supports the conclusion that introducing the cryptographic protocols only had a minor negative impact on performance. Furthermore, we note that the whole-row mode of FE achieved a suspiciously high model-accuracy, which is possibly a consequence of the non-attack labels constituting a large majority of the test set. This is indicated by the significantly lower F1-score and is verified by the ratio of predicted positives/negatives in Figure 5.4c.

The accuracies of all the models could be improved by introducing more complexity and depth to the models. However, HE introduces heavy constraints on the complexity of the model, given that it is only able to evaluate polynomial expressions. While there are methods to combat this limitation, such as using polynomial approximations to non-polynomial functions, this drastically reduces the ability to leverage available and scrutinized techniques. To make a fair comparison between HE and FE, the same set of limitations was applied to the FE model, even if the FE protocol by itself does not impose any restrictions on the complexity of the model. This speaks to the flexibility of FE. However, considering an OT domain, it appears that HE allows for sufficiently complex models to create a high performing PPIDS.

6.3 Network traffic

Regarding the network traffic metric, the results presented in Table 5.3 are very clear: HE produces significantly larger ciphertexts than FE. This places a much greater constraint on the bandwidth, which subsequently increases the risk of network congestion. Additionally, since the output of the IDS server must be returned to the client system, this places a total load of 365 kB that needs to be transferred for each new sensor reading. Depending on the frequency f of the measurements, the bandwidth-consumption is $365f$ kB/s. For the SWaT dataset, the frequency is only one measurement per second, meaning that the bandwidth-consumption is 365 kB/s. While this is much larger than for the baseline and FE scenarios, it is still easily manageable for modern network standards.

Note that HE is the only scenario with a non-zero standard deviation, meaning that the size of the payload is volatile. The fluctuation is inherent to the CKKS scheme, but a standard deviation of less than 1 kB is too small to have any significant impact on the performance of the network. While the network consumption is feasible in this scenario, another facility producing larger volumes or more complex data may require a more sophisticated model. Then, larger CKKS parameters are required which is directly correlated to the size of the ciphertext. In such a scenario, HE could impact the speed metric negatively, due to its higher bandwidth consumption and longer inference times.

6.4 Memory consumption

The results with regard to memory are quite positive for both the HE and FE approaches. Most importantly, the memory requirements for all processes active on the client devices are low, especially for the FE scenario. For the FE scenario, the most memory-intensive client side process is the encryption, which consumes 82 kB of memory at its peak. The most memory-intensive client-side process for the HE scheme is also encryption, but its peak memory consumption was 3.953 MB. This implies that the client-side memory requirements are significantly higher for the HE scheme than the FE scheme. The low memory demand of the FE scheme means that it could reasonably be run even on very weak OT devices and/or in embedded scenarios. The HE scheme does not have the same level of flexibility, but this does not rule out the HE scheme from OT scenarios as even the modest OT components typically have at least 4 MB of memory. With that said, very computationally weak devices may find the HE scheme challenging.

The data presented in Table 5.4 show that the majority of processes require less than one megabyte of memory, and all but one require less than 10 MB. The one outlier is the HE inference, which is performed on the IDS server and requires 73.085 MB of memory. This is a significant increase from the other processes, but is not unreasonable given the well-known complexity of homomorphic encryption operations.

One interesting result is that the modular and whole-row run configurations in the FE scenario require less memory during inference than the baseline model. This may seem counterintuitive at first, but there is a clear explanation. Since the modular and whole-row run configurations have input lengths of one and six, the corresponding neural networks receive a lower-dimension input data, which in turn speeds up the inference. For the one-by-one run configuration where the length of input is 44—as in the baseline case—inference is slightly slower.

6.5 Interpretability

We now present the qualitative answers related to the interpretability of results for both the HE and FE approaches.

In case of detection, can the deviating component of the system be identified?

For the baseline scenario, this is possible using explainable AI (XAI) techniques such as SHAP [68] or LIME [69]. Since the plaintext feature data is available to the external party, it is feasible to train a neural network which presents not only whether a deviation has occurred, but also which columns contribute most to that determination. This information could then be passed back to the client.

For the HE scenario, component identification is likely not possible using existing technology. Tools such as SHAP and LIME are fundamentally designed to model

predictions by perturbing input features and observing how these perturbations affect the output. This process inherently requires feature-level access to plaintext input data, which is not available under the CKKS scheme. In the HE scenario, the data remains encrypted throughout the entire detection pipeline and the server therefore cannot inspect any individual feature of the data. While this feature is the reason for encrypting the data in the first place, it is also the reason why deducible information is limited.

From the analysis of the FE scenario, the ability to identify a deviating component depends on which FE run configuration is used. For the whole-row configuration, this is impossible as the output of the FHIPE system is a single scalar. Trying to identify the attacked component is not possible since the external party cannot know what contribution a particular component made to the resulting inner product. However, if one uses a more fine-grained approach, such as the modular or one-by-one run configurations, some identification could be possible. In those cases, the IDS server would end up with an array of distances whose order corresponds to the order of the input data. The deviating column could then be identified similarly to the baseline scenario. With this information, the IDS server could pass the index of the deviating column back to the client, thus providing additional information about the attack.

If the attack is previously known, can it be identified to predict subsequent effects?

From the analysis, the answer for all three scenarios explored in this thesis is no. While there are many examples of IDSs identifying previously known attacks [70], the neural network-based approach used in this thesis does not lend itself well to that type of task. In all three scenarios, the models are quite successful at detecting attacks. However, identifying specific attacks and subsequently predicting upcoming effects would require a more complex detection model.

Can the system supply additional metadata to provide contextual information that might be important for decision making?

For the baseline scenario, this is possible. Since the plaintext feature data is available to the external party, any relevant metadata can simply be passed alongside it.

For the HE scenario, the answer is yes, but with limited options. Since the CKKS scheme only enables addition and multiplication on the ciphertext, the potential information that can be extracted from the metadata is greatly limited. However, there are other HE schemes, e.g., TFHE, that allow evaluation of ciphertext over arbitrary circuits of logical gates. In theory, a hybrid IDS could leverage TFHE for metadata, where logical operations such as comparisons might be required, while using CKKS to process the normal data. Such a system would provide more flexibility and precision, but it would also introduce additional computational overhead on the client system.

For the FE scenario, the answer is likely no. The FHIPE scheme is designed to compute scalar products of integer values. As such, passing additional arbitrary

metadata is not straightforward. There is a possibility of setting up additional scalar product operations—similar to the distance calculation used in detection—to generate other types of statistics, but at the time of writing, such use cases are not obvious. One could of course encrypt the metadata separately and pass it alongside the FHIPE payload, but this would mean that the IDS server gains full access to all metadata, which may not be desirable.

Can the systems run even when using computationally weak client side devices?

From the results presented in Tables 5.1, 5.2, 5.3, and 5.4, the answer for all three scenarios appear to be yes. While there are certain processes that require large amounts of memory or CPU time, these are all performed on the IDS server where one can assume that greater computational resources are available. When considering the client-side processes, they require at most 17.387 milliseconds of CPU time (FE one-by-one encryption), and at most 3.953 megabytes of memory (HE encryption). This implies that all three scenarios should be executable even on weak devices.

6.6 Summary analysis of homomorphic and functional encryption

With the experimental results presented and analyzed, this section provides a summary evaluation of the subjects investigated in this thesis. The research questions defined in Section 1.2 will serve as a template to structure the analysis.

How does the introduction of HE and FE affect the system's detection accuracy?

As seen in Table 5.2, when using the best run-configuration for the FE scenario, introducing both the HE and FE scenarios only produces minor degradations in model performance. The model-accuracy drops slightly from 0.931 for the baseline case to 0.918 and 0.923 for the HE and FE models respectively. The AUC remains unchanged, and the F1-score drops from 0.75 for the baseline model to 0.72 and 0.73 for the HE and FE models.

How does the introduction of HE and FE affect the detection speed of the system?

As stated in Section 6.1, the total worst-case added runtime from implementing the cryptographic schemes in the experiments was found to be less than 125 ms. This implies that the overall impact of introducing the encryption systems is relatively low for most applications.

How does the introduction of HE and FE affect the network traffic in the system?

As shown in Table 5.3, it is clear that both HE and FE introduce significant costs with regard to network traffic. While the memory footprint of FE is significantly smaller, its average network payload is still 23 times larger than that of the baseline scenario, assuming an input length of 44. The situation is even worse for the HE scheme: its data size is found to be approximately 1500 times that of the baseline scenario.

How does the introduction of HE and FE affect the memory consumption of the system?

As stated in Section 6.4, considering the memory consumption of adding HE or FE requires some nuance. When looking at the client-side processes, the peak memory usage of both HE and FE is relatively low, coming in at a maximum of around 4 MB for HE and 82 kB for FE. The server-side processes are significantly larger than this, requiring 73.085 MB for HE and 2.507 MB for FE. However, it is important to note that these requirements only exist on the server, where computational resources are more abundant.

How does the introduction of HE and FE affect the interpretability of results in the system?

While the answers to the qualitative interpretability questions defined in Section 4.3.5 vary somewhat between HE and FE, in general, this thesis finds that both HE and FE have a moderately negative impact on the interpretability of results. As described in Section 6.5, there are some scenarios where using the cryptographic schemes completely disables functionality that is available in the baseline model. Other times, the functionality is still available, but with additional restrictions or limitations. Lastly, there are also situations where the functionality remains unchanged even after introducing the cryptographic protocols. Because of this, the impact to the interpretability depends heavily on the structure of the original implementation, and which aspects of interpretability one considers.

How does an IDS that utilizes homomorphic or functional encryption for privacy preservation compare to an unencrypted IDS when applied to OT data?

As seen in the previous paragraphs, this thesis finds that the costs and benefits of introducing the cryptographic schemes vary depending on the specific metric and the cryptographic scheme used. As a result, how an HE- or FE-based IDS compares depends heavily on what aspects users find most important in a real-world implementation. If the IDS exists in a scenario where network access is limited and additional payload size incurs significant cost, or if sub-second time differences in detection speed are crucial, then implementing either system may

prove difficult. However, in scenarios where these factors are less critical, this thesis suggests that usage of HE or FE is indeed feasible. This approach enables the trustless use of external IDS services without compromising client data, requiring significant additional client-side computation, or incurring significant degradations in model-accuracy.

6.7 Ethics and sustainability

Many OT systems, such as power plants, smart grids, and water treatment facilities, are considered critical infrastructure. This classification implies that the data they produce must be tightly protected, as it may contain both personal information and sensitive operational details. If such data is exposed, it could provide adversaries with insights into system behavior, creating opportunities for exploitation or sabotage. Therefore, when outsourcing encrypted computation to cloud environments, it is crucial that the cryptographic protocols are not only theoretically sound, but also securely and correctly implemented. Failing to uphold these standards could not only compromise individual privacy, but also threaten the security and reliability of essential national infrastructure.

Most people trust the security of cryptographic protocols without having an understanding of the fundamental mathematical reasoning and proofs that support them. While this can be problematic, most broadly employed protocols are heavily scrutinized by cryptographers, which builds trust in the schemes. Traditionally, cryptography is mainly used for private communication and data storage. However, when cryptography begins to influence the decision-making process, as in the PPIDSs created in this thesis, it becomes much more critical to explain and challenge the results. This concern is amplified when combined with neural networks, which are already criticized for their lack of transparency. The result is an opaque black-box system that provides no justification for its claims. To address this, further research in explainable AI (XAI) and verifiable computation is crucial, in order to develop systems that both preserve privacy, and provide interpretable and trustworthy results.

Beyond interpretability, ethical deployments of such systems also require clearly defined terms of accountability. Even with explainable models, it is practically impossible to create the perfect intrusion detection system; misclassifications, either by failing to detect an attack or flagging benign activity as malicious, are inevitable. Therefore, it is important to delineate accountability for when errors do occur. Accomplishing this requires a comprehensive understanding of the system's architecture and behavior in order to detect, document, and address potential limitations.

While this thesis shows that operating a PPIDS using homomorphic or functional encryption is feasible in terms of computational costs, sustainability considerations must not be overlooked. Many OT facilities may already possess the computational resources to support expensive encryption operations. However, in more resource-constrained environments such as a smart grid with distributed low-power sensors, local encryption could pose a challenge. In such scenarios, upgrading edge devices may be necessary, leading to increased electronic waste and financial costs. Further-

more, although shallow neural networks reduces computational demand, performing expensive encrypted inference still requires significant amounts of energy. This has implications for both environmental and economic sustainability, particularly when scaled up.

6.8 Future work

Although this thesis covers significant ground with regard to the technical analysis of HE and FE in an OT IDS setting, several interesting avenues of further research remain. This section explores some of these avenues and gives ideas for future work.

One of the aspects considered in this thesis is the performance of the FE model across differenct run configurations. As presented in Section 4.2.3, these configurations are the one-by-one, modular, and whole-row configurations. While these represent three ways of subdividing the dataset for use in the FE scheme, they are by no means the only ones. Given how drastically the choice of run configuration impacts most metrics, an important area of research would be how to find the optimal subdivision of data. For example, combining columns as in the modular configuration reduces the computational burden, but it also degrades the detection accuracy as seen in Table 5.2. Perhaps a better subdivision exists that allows an implementation to get the benefits of fewer computations without losing accuracy? Maybe there is a general method for determining optimal subdivisions that extends beyond the SWaT dataset? Questions such as these provide ample opportunity for further research.

A current limitation of the FE scheme is the lack of open-source implementation support. While there is an open-source library created by Kim et al. [15] showcasing the techniques, it is old, not actively maintained, and not viable for production use. Even running the tests performed in this thesis required significant modifications and optimizations. As such, developing tools and libraries to make FHIPE implementation more accessible to researchers and developers is a large area for practical future work.

As stated in Section 6.1, some metrics presented in this thesis are influenced by the choice of detection model. In particular, the speed of the HE scheme is heavily reliant on the depth of the neural network. For a shallow neural network, such as the one used in this thesis, the homomorphic computations are manageable. However, as the model becomes more complex, the homomorphic computations become increasingly time-consuming. The rate at which speed decreases as model complexity increases remains unclear, making this an important area for future research into the strengths and weaknesses of the HE scheme.

Although the memory results discussed in Section 6.4 indicate that both the HE and FE schemes could be viable for many types of OT components, there is a lack of real-world implementations. As such, a clear area of future research relates to implementing the schemes on different kinds of OT devices to evaluate their practical functionality. This could improve the understanding of what types of devices could utilize each scheme. It may also help establish a taxonomy of possible real-world applications where the techniques explored in this thesis are applicable.

7

Conclusion

In this thesis, two cryptographic paradigms that enable ciphertext-computation were evaluated for their applicability in privacy-preserving IDSs in OT environments. The idea is to enable an external party to operate an IDS without access to the client's plaintext data. Three equivalent neural network-powered intrusion detection models were built and adapted to the respective paradigm, including an unencrypted non-privacy-preserving baseline. The privacy-preserving models were evaluated against each other across several metrics and then compared relative to the baseline model.

The results show that the introduction of the cryptographic paradigms, homomorphic and functional encryption, only gave a minor decrease in detection accuracy. However, the two scenarios introduce significant computational overhead, negatively affecting both detection speed and network traffic. While the shallow models built in this thesis remained computationally feasible for a modern computer, deploying a privacy-preserving IDS in a real-world OT setting introduces additional challenges. In particular, the volume and frequency of data measurements amplify the computational and communicational overhead introduced by the cryptographic operations. This can be problematic for rudimentary edge-devices that are common in OT networks. Additionally, the detection model would require greater complexity to obtain an acceptable level of accuracy.

In conclusion, this thesis has demonstrated that privacy-preserving intrusion detection using homomorphic and functional encryption is possible, and potentially practically viable under certain conditions. While trade-offs in performance and complexity remain, the benefits of enabling external IDS services without exposing sensitive data are significant. As threats to critical infrastructure continue to evolve, the defensive tools must evolve alongside them, with privacy remaining a central concern in that evolution. Future work could focus on optimizing the cryptographic implementations, building open-source libraries for functional encryption, and testing on real-world industrial sites to validate the practical applicability of the cryptographic paradigms.

Bibliography

[1] G. M. Makrakis, C. Kolias, G. Kambourakis, C. Rieger, and J. Benjamin, "Industrial and critical infrastructure security: Technical analysis of real-life security incidents," *IEEE Access*, volume 9, pages 165 295–165 325, 2021. DOI: 10.1109/ACCESS.2021.3133348.

[2] Google Cloud Threat Intelligence. "Sandworm disrupts power in Ukraine via operational technology." Accessed: 2025-04-04, Google Cloud. (2024), [Online]. Available: https://cloud.google.com/blog/topics/threat-intelligence/sandworm-disrupts-power-ukraine-operational-technology.

[3] H. H. Addeen, Y. Xiao, J. Li, and M. Guizani, "A survey of cyber-physical attacks and detection methods in smart water distribution systems," *IEEE Access*, volume 9, pages 99 905–99 921, 2021. DOI: 10.1109/ACCESS.2021.3095713.

[4] F. Hendaoui, A. Ferchichi, L. Trabelsi, R. Meddeb, R. Ahmed, and M. K. Khelifi, "Advances in deep learning intrusion detection over encrypted data with privacy preservation: A systematic review," *Cluster Computing*, volume 27, number 7, pages 8683–8724, 2024, ISSN: 1573-7543. DOI: 10.1007/s10586-024-04424-4. [Online]. Available: https://doi.org/10.1007/s10586-024-04424-4.

[5] G. Assenza, L. Faramondi, G. Oliva, and R. Setola, "Cyber threats for operational technologies," *International Journal of System of Systems Engineering*, volume 10, number 2, pages 128–142, 2020. DOI: 10.1504/IJSSE.2020.109127. eprint: https://www.inderscienceonline.com/doi/pdf/10.1504/IJSSE.2020.109127. [Online]. Available: https://www.inderscienceonline.com/doi/abs/10.1504/IJSSE.2020.109127.

[6] M. Shen, K. Ye, X. Liu, L. Zhu, J. Kang, S. Yu, Q. Li, and K. Xu, "Machine learning-powered encrypted network traffic analysis: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, volume 25, number 1, pages 791–824, 2023. DOI: 10.1109/COMST.2022.3208196.

[7] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, volume 34, number 1, pages 1–11, 2011, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2010.07.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804510001281.

[8] D.-H. Ryu, S.-Y. Jeon, J. Hong, and M.-K. Lee, "Efficient L_p Distance Computation Using Function-Hiding Inner Product Encryption for Privacy-Preserving Anomaly Detection," *Sensors*, volume 23, number 8, page 4169, 2023. DOI: 10.3390/s23084169. [Online]. Available: https://doi.org/10.3390/s23084169.

[9] L. Wu, H. Shi, S. Fu, Y. Luo, and M. Xu, "p2Detect: Electricity Theft Detection With Privacy Preservation for Both Data and Model in Smart Grid," *IEEE Transactions on Smart Grid*, volume 14, number 3, pages 2301–2312, 2023. DOI: 10.1109/TSG.2022.3214194.

[10] R. Li, S. Bhattacharjee, S. K. Das, and H. Yamana, "Look-up table based fhe system for privacy preserving anomaly detection in smart grids," in *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2022, pages 108–115. DOI: 10.1109/SMARTCOMP55677.2022.00030.

[11] C. K. Zhang, A. Yin, W. Zuo, and Y. Y. Chen, "Privacy preserving anomaly detection based on local density estimation," *Mathematical Biosciences and Engineering*, volume 17, number 4, pages 3478–3497, 2020. DOI: 10.3934/mbe.2020196. [Online]. Available: https://doi.org/10.3934/mbe.2020196.

[12] K. Kim and H. C. Tanuwidjaja, *Privacy-Preserving Deep Learning: A Comprehensive Survey*. Springer, 2021.

[13] S. Niksefat, P. Kaghazgaran, and B. Sadeghiyan, "Privacy issues in intrusion detection systems: A taxonomy, survey and future directions," *Computer Science Review*, volume 25, pages 69–78, 2017, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2017.07.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013716301204.

[14] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A dataset to support research in the design of secure water treatment systems," in *Critical Information Infrastructures Security*, G. Havarneanu, R. Setola, H. Nassopoulos, and S. Wolthusen, Eds., Cham: Springer International Publishing, 2017, pages 88–99, ISBN: 978-3-319-71368-7.

[15] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," in *Security and Cryptography for Networks*, D. Catalano and R. De Prisco, Eds., Cham: Springer International Publishing, 2018, pages 544–562.

[16] M. I. Ibrahem, M. Nabil, M. M. Fouda, M. M. E. A. Mahmoud, W. Alasmary, and F. Alsolami, "Efficient privacy-preserving electricity theft detection with dynamic billing and load monitoring for ami networks," *IEEE Internet of Things Journal*, volume 8, number 2, pages 1243–1258, 2021. DOI: 10.1109/JIOT.2020.3026692.

[17] C. Niu, F. Wu, S. Tang, S. Ma, and G. Chen, "Toward verifiable and privacy preserving machine learning prediction," *IEEE Transactions on Dependable and Secure Computing*, volume 19, number 3, pages 1703–1721, 2022. DOI: 10.1109/TDSC.2020.3035591.

[18] L. Karaçay, E. Savaş, and H. Alptekin, "Intrusion detection over encrypted network data," *The Computer Journal*, volume 63, number 1, pages 604–619, 2020. DOI: 10.1093/comjnl/bxz111.

[19] E. J. Chou, A. Gururajan, K. Laine, N. K. Goel, A. Bertiger, and J. W. Stokes, "Privacy-preserving phishing web page classification via fully homomorphic encryption," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pages 2792–2796. DOI: 10.1109/ICASSP40776.2020.9053729.

[20] J. T. Force, "Risk management framework for information systems and organizations," *NIST Special Publication*, volume 800, page 37, 2018.

[21] K. Stouffer, K. Stouffer, M. Pease, C. Tang, T. Zimmerman, V. Pillitteri, S. Lightman, A. Hahn, S. Saravia, A. Sherule, *et al.*, *Guide to operational technology (OT) security*. US Department of Commerce, National Institute of Standards and Technology, 2023.

[22] D. Kushner, "The real story of Stuxnet," *IEEE Spectrum*, volume 50, number 3, pages 48–53, 2013.

[23] L. Sgaglione, L. Coppolino, S. D'Antonio, G. Mazzeo, L. Romano, D. Cotroneo, and A. Scognamiglio, "Privacy preserving intrusion detection via homomorphic encryption," in *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, IEEE, 2019, pages 321–326.

[24] R. G. Bace, P. Mell, *et al.*, "Intrusion detection systems," *US Department of Commerce, Technology Administration, National Institute of Standards and Techonology*, 2001.

[25] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, volume 36, number 1, pages 16–24, 2013, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2012.09.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804512001944.

[26] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, volume 2, number 1, pages 1–22, 2019.

[27] M. Bhavsar, K. Roy, J. Kelly, and O. Olusola, "Anomaly-based intrusion detection system for IoT application," *Discover Internet of things*, volume 3, number 1, page 5, 2023.

[28] M. Kaouk, J.-M. Flaus, M.-L. Potet, and R. Groz, "A review of intrusion detection systems for industrial control systems," in *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, IEEE, 2019, pages 1699–1704.

[29] M. Haenlein and A. Kaplan, "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence," *California management review*, volume 61, number 4, pages 5–14, 2019.

[30] OpenAI, *Chatgpt*, https://chat.openai.com, Accessed: 2025-04-08, 2024.

[31] Microsoft, *Microsoft Copilot*, https://www.microsoft.com/en-us/microsoft-copilot, Accessed: 2025-04-08, 2024.

[32] Y. Chao-Ming. "Primer: Artificial Intelligence for Demand Forecasting." (2022), [Online]. Available: https://www.newhorizon.ai/blogs/primer-artificial-intelligence-for-demand-forecasting/ (visited on 04/08/2025).

[33] S. Rishabh. "Introduction to Machine Learning." (2024), [Online]. Available: https://medium.com/@RobuRishabh/introduction-to-machine-learning-555b0f1b62f5 (visited on 04/03/2025).

[34] I. El Naqa and M. J. Murphy, "What is machine learning?" In *Machine Learning in Radiation Oncology: Theory and Applications*, I. El Naqa, R. Li, and M. J. Murphy, Eds. Cham: Springer International Publishing, 2015, pages 3–11, ISBN:

978-3-319-18305-3. DOI: 10.1007/978-3-319-18305-3_1. [Online]. Available: https://doi.org/10.1007/978-3-319-18305-3_1.

[35] KodeinKGP. "Supervised Learning: A Comprehensive Guide." (2023), [Online]. Available: https://medium.com/@kodeinkgp/supervised-learning-a-comprehensive-guide-7032b34d5097 (visited on 04/04/2025).

[36] R. Sathya and A. Abraham, "Comparison of supervised and unsupervised learning algorithms for pattern classification," *International Journal of Advanced Research in Artificial Intelligence*, volume 2, number 2, pages 34–38, 2013.

[37] "What is unsupervised learning?" (2025), [Online]. Available: https://cloud.google.com/discover/what-is-unsupervised-learning (visited on 04/08/2025).

[38] P. Kashyap. "What is Deep Learning? A Beginner's Guide to Understanding Deep Learning Concepts." (2024), [Online]. Available: https://medium.com/@piyushkashyap045/what-is-deep-learning-a-beginners-guide-to-understanding-deep-learning-concepts-b2eef1370222 (visited on 04/09/2025).

[39] Google. "What is deep learning?" (2024), [Online]. Available: https://cloud.google.com/discover/what-is-deep-learning (visited on 04/09/2025).

[40] M. Kara. "How many layers are typically found in a neural network?" (2024), [Online]. Available: https://markmkara.medium.com/how-many-layers-are-typically-found-in-a-neural-network-a5e9af38ceda (visited on 04/09/2025).

[41] S. S. Kadam, A. C. Adamuthe, and A. B. Patil, "CNN model for image classification on MNIST and fashion-MNIST dataset," *Journal of scientific research*, volume 64, number 2, pages 374–384, 2020.

[42] N. Kang. "Introducing Deep Learning and Neural Networks — Deep Learning for Rookies (1)." (2017), [Online]. Available: https://medium.com/data-science/introducing-deep-learning-and-neural-networks-deep-learning-for-rookies-1-bd68f9cf5883 (visited on 04/09/2025).

[43] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force (IETF), Request for Comments 8446, Aug. 2018, Accessed: 2025-02-21. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446 (visited on 02/21/2025).

[44] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, volume 22, number 6, pages 644–654, 1976. DOI: 10.1109/TIT.1976.1055638.

[45] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, volume 21, number 2, pages 120–126, Feb. 1978, ISSN: 0001-0782. DOI: 10.1145/359340.359342. [Online]. Available: https://doi.org/10.1145/359340.359342.

[46] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*, Draft Version 0.6. 2023, Available at https://toc.cryptobook.us/. [Online]. Available: https://toc.cryptobook.us/ (visited on 02/21/2025).

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala,

"Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pages 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[48] "PyTorch." (2025), [Online]. Available: https://www.nvidia.com/en-us/glossary/pytorch/ (visited on 03/26/2025).

[49] "Companies that use PyTorch." (2025), [Online]. Available: https://theirstack.com/en/technology/pytorch (visited on 03/27/2025).

[50] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, *TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption*, 2021. arXiv: 2104.03152 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2104.03152.

[51] J. A. Akinyele, M. D. Green, and A. D. Rubin, *Charm: A framework for rapidly prototyping cryptosystems*, Cryptology ePrint Archive, Paper 2011/617, Accessed: 2025-05-28, 2011. [Online]. Available: https://eprint.iacr.org/2011/617.

[52] J. A. Akinyele, M. D. Green, and A. D. Rubin, *Charm: A framework for rapidly prototyping cryptosystems*, https://github.com/JHUISI/charm, Accessed: 2025-05-28.

[53] X. Yi, R. Paulet, and E. Bertino, "Homomorphic encryption," in *Homomorphic Encryption and Applications*. Cham: Springer International Publishing, 2014, pages 27–46, ISBN: 978-3-319-12229-8. DOI: 10.1007/978-3-319-12229-8_2. [Online]. Available: https://doi.org/10.1007/978-3-319-12229-8_2.

[54] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in cryptology–ASIACRYPT 2017: 23rd international conference on the theory and applications of cryptology and information security, Hong kong, China, December 3-7, 2017, proceedings, part i 23*, Springer, 2017, pages 409–437.

[55] D. Huynh. "CKKS explained." (2020), [Online]. Available: https://openmined.org/blog/ckks-explained-part-1-simple-encoding-and-decoding/?_gl=1*tzqcxm*_ga*NzY3NjYzMjQ3LjE3NDUyNTIzNjM.*_ga_X111NXDSGH*czE3NDg2MTg4NzUkbzEwJGcxJHQxNzQ4NjE5MTgxJGo0NyRsMCRoMA.. (visited on 05/30/2025).

[56] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography*, Y. Ishai, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pages 253–273, ISBN: 978-3-642-19571-6.

[57] S. I. Gallant *et al.*, "Perceptron-Based Learning Algorithms," *IEEE Transactions on neural networks*, volume 1, number 2, pages 179–191, 1990.

[58] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, volume 65, number 6, page 386, 1958.

[59] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[60] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*. Courier Corporation, 1965, volume 55.

[61] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pages 400–407, 1951.

[62] N. Kyurkchiev and S. Markov, "Sigmoid functions: Some approximation and modelling aspects," *LAP LAMBERT Academic Publishing, Saarbrucken*, volume 4, page 34, 2015.

[63] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA, 2015, volume 25.

[64] S. Rogers and M. Girolami, *A first course in machine learning*. Chapman and Hall/CRC, 2016.

[65] H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter, "Logistic regression over encrypted data from fully homomorphic encryption," *BMC medical genomics*, volume 11, pages 3–12, 2018.

[66] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., Nov. 2018.

[67] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, volume 27, number 8, pages 861–874, 2006.

[68] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pages 4765–4774. [Online]. Available: `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf`.

[69] M. T. Ribeiro, S. Singh, and C. Guestrin, *"why should i trust you?": Explaining the predictions of any classifier*, 2016. arXiv: `1602.04938 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1602.04938`.

[70] J. Díaz-Verdejo, J. Muñoz-Calle, A. Estepa Alonso, R. Estepa Alonso, and G. Madinabeitia, "On the detection capabilities of signature-based intrusion detection systems in the context of web attacks," *Applied Sciences*, volume 12, number 2, page 852, 2022. DOI: `10.3390/app12020852`. [Online]. Available: `https://doi.org/10.3390/app12020852`.

A

Memory sampler code

```
import time
import psutil
import os
import gc
from threading import Thread

class MemorySampler(Thread):
    def __init__(self, interval_ms=1):
        Thread.__init__(self)
        self.proc = psutil.Process(os.getpid())
        self.interval = interval_ms / 1000.0
        self.running = True
        self.peak_rss = 0

    def run(self):
        try:
            while self.running:
                rss = self.proc.memory_info().rss
                if rss > self.peak_rss:
                    self.peak_rss = rss
                time.sleep(self.interval)
        except psutil.NoSuchProcess:
            pass

    def stop(self):
        self.running = False
        self.join()
        return self.peak_rss

def monitor(func):
    def wrapper(*args, **kwargs):
        proc = psutil.Process(os.getpid())

        # Force garbage collection before measurement
        gc.collect()
```

```
        # Measure memory before function call
        rss_before = proc.memory_info().rss

        # Start sampler
        sampler = MemorySampler()
        sampler.start()

        start_wall = time.time()
        start_cpu = proc.cpu_times()

        result = func(*args, **kwargs)

        end_wall = time.time()
        end_cpu = proc.cpu_times()
        sampler_peak = sampler.stop()
        rss_after = proc.memory_info().rss

        wall_time = end_wall - start_wall
        user_cpu_time = end_cpu.user - start_cpu.user
        sys_cpu_time = end_cpu.system - start_cpu.system

        delta_rss = max(0, rss_after - rss_before)
        peak_rss = max(0, sampler_peak - rss_before, delta_rss)

        return result, (wall_time, user_cpu_time, sys_cpu_time, peak_rss)
    return wrapper
```