SECURE RULE MATCHING OF SIGNATURES IN DATA TRAFFIC WITH MACHINE LEARNING APPROACHES

by

Mohamadamin Nemati

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
December 2020

I dedicate this to my family, who helped me get where I am today.

Table of Contents

List of Tables

List of Figures

List of Abbreviations Used

Acronyms

ANN	Artificial Neural Network
DGK	Damgård, Gaisler, Krøigaard
IND-CPA	INDistinguishability under Chosen Plaintext Attack
LIME	Local Interpretable Model-agnostic Explanations
ML	Machine Learning
OPES	Order-Preserving Encryption System
RSA	Rivest, Shamir, Adleman
SHAP	SHapley Additive exPlanations

Functions

$\gcd(m, n)$	Greatest common divisor between m and n
$\lambda(p)$	Carmichael's function
$\mathrm{lcm}(m, n)$	Lowest common multiple between m and n
$\phi(n)$	Euler's totient function
$\phi_i(f, x)$	Shapely value for feature i in model f Having the set of input x
D	Decryption function
E	Encryption function
f	Machine learning model
$f_x(z)$	Machine learning model with implicit reverse mapping from simplified input to original input

$h(x)$	Mapping function from original input to simplified input
$h_x(z)$	Inverse mapping function from simplified input to original input

Sets

$\mathbb{R}$	Real Numbers
$\mathbb{Z}$	Integers
$\mathbb{Z}_n^*$	Integers from the set $\{0, 1, \cdots, n-1\}$ that are coprime to n
$\mathbb{Z}_n$	Integers from the set $\{0, 1, \cdots, n-1\}$
M	Set of all the features in the machine learning model
pk	Set of elements comprising public key
sk	Set of elements comprising secret/private key
x	Set of original input
z	Set of simplified input
x'	Subset of original input
z'	Subset of simplified input
$z' \setminus i$	Set of simplified input excluding feature i

Abstract

Due to its enormous capabilities, machine learning has been applied to various disciplines since its emergence. Cybersecurity is no exception in this regard. However, the effectiveness of machine learning classifiers comes at a cost.

Although these classifiers' complexity is the reason they are accurate, it is also the reason for their relative slowness. In this thesis, we try to propose a method that can be used to generate a Snort-like signature from a trained classifier. By doing this, we can get a signature (or usually multiple ones) that approximates the classifier and helps us do the filtering more efficiently. Apart from efficiency, this method can help us gain some insight into the inner-workings of the classifier.

After deriving the signatures from the classifier, we have to have an environment to evaluate our signatures. This environment should be able to simulate a real-life scenario to make the evaluation as veracious as possible.

For this evaluation, we have also proposed multiple methods that can be used for signature evaluation without divulging any confidential information inside the signature. These methods are then compared to see, which is the most suitable in different circumstances.

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Srini, who guided me throughout this project. I would also like to thank Dr. Manero, without whom this thesis would not be possible.

I wish to acknowledge the help and support that my labmates provided. I would also like to show my deep appreciation to my family, whose unwavering support helped me finalize this.

Chapter 1

Introduction

With the ever-growing reliance of human beings on the Internet, the need for privacy and security is becoming more and more critical. The concern about the safety of information stored on the Internet can hinder people from using it to its fullest potential. For defending Internet servers from different adversaries and intruders, we have many schemes in place. However, their constant need for reconfiguration to filter new malware varieties can be tedious and costly for companies implementing those schemes. Machine learning (ML) can act as a remedy for this problem. By training classifiers that are not susceptible to small modification in malware's codes, we can build network intrusion detection systems (NIDS) that are more robust than their signature-based counterparts.

Although these classifiers perform well as NIDS, they cannot match signature-based NIDS in process time. As a result, some companies may decide to avoid using such classifiers because of the resource-wise overhead they impose. In this thesis, we have introduced a new method for deriving signatures from trained classifiers. By deriving a signature that encompasses the algorithm that the classifier uses for prediction, we can have the best of both worlds.

ML interpretability can help us get to the desired signature from the trained ML model. However, due to the inherent trade-off between the accuracy and interpretability of a ML model, training a highly accurate model while maintaining interpretability is usually not feasible. Therefore, in some scenarios, data scientists may prefer a less accurate model that is partially interpretable over a black box that is more accurate. Nevertheless, there are some methods that can give us some insight into the classifier function. Using these methods, we may even be able to better understand a classifier without the need to reduce its complexity at the cost of accuracy. In this thesis, we have taken a look at some of such methods used in the scope of ML interpretability for finding a signature that mimics the ML model function.

After deriving the signature, we need to have a way of evaluating it. The degree to which the derived signature succeeds in mimicking the ML model function can be seen by testing it against the same data we used to evaluate the ML model. However, a model's accuracy can sometimes be deceiving. A highly accurate model can sometimes fail to perform that accurately in a real-life scenario. That is because the data that we encounter in real-life scenarios are much more diverse and unpredictable.

For this evaluation, to simulate real-life scenarios well enough, it would be better to use as much network traffic as possible. However, using network traffic from other sources can impose drastic overhead on the company's resources. An alternative is filtering the network traffic using the signature inside the third party company (i.e. the company that has provided the network traffic).

The methodology mentioned above can divulge some classified information about a company's network infrastructure. As a result, it is required that this filtering happens while the signature is encrypted. Hence, for the sake of confidentiality, we propose and compare different methods for performing this filtering in a secure manner.

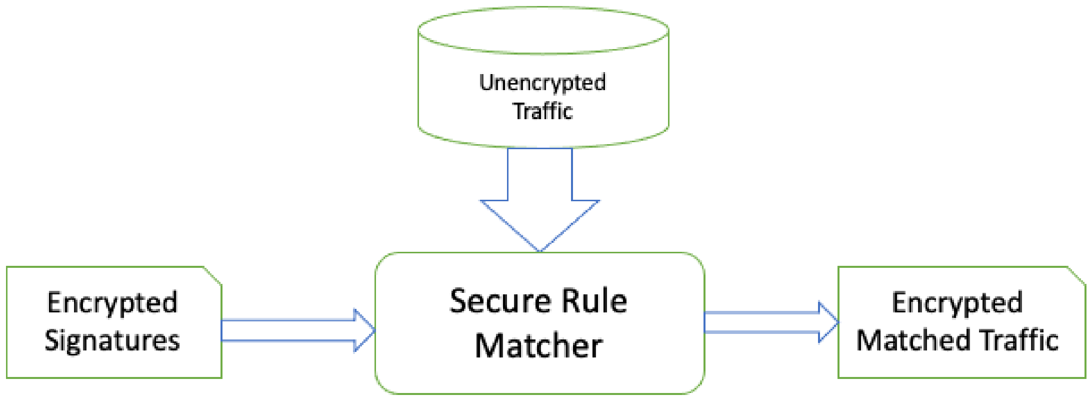

Figure 1.1: Outline of the challenge faced in secure matching

As shown in Figure 1.1, the main challenge is performing matching while the signatures are encrypted, and the network traffic isn't. The remedy for this situation is either encrypting the traffic in a specific way that can support matching or securely decrypting the signatures without violating confidentiality. Both of these solutions are explained in detail in Chapter 4.

1.1 Motivation

ML is a discipline that designs algorithms able to adapt to the system's data dynamically. A ML algorithm is a function with an input and an output

$$f(x) = y \tag{1.1}$$

This function will adapt its behaviour depending on the data used to train the algorithm. Traditional algorithms are functions too, but contrary to ML, their behaviour is defined in the code. In ML, the behaviour of the function is defined by the data used to train the model. The same algorithm, trained with different data, will behave differently.

On the other hand, signature-based NIDS are strictly defined and must be changed now and then to be able to capture malware with the same functionality but slightly different syntax. Adversaries can easily evade these kinds of NIDS by modifying some parts of malware's code. Therefore, the maintenance of these systems can be costly and time-consuming.

Due to the dynamicity mentioned above, ML models are more error-free and efficient in filtering malignant and benign network traffics. That is because ML models can capture the semantic definition of malware and do not restrict themselves to syntax. Hence, one cannot evade them by making small changes to an already renown malware. Therefore, these ML models have been applied as NIDS extensively before. Some of these applications are mentioned in the following paragraph.

Avatefipour [1] introduces an application of ML in in-vehicle network security in their work. Hui [2] introduces different ML techniques that are applicable in creating intrusion detection systems. Finally, Xin [3] introduces different cybersecurity schemes that utilize ML or deep learning in their survey paper.

These ML models can be useful in the detection and prevention of network intrusions [4]. Signature-based IDS can easily be evaded by introducing a small modification inside the malware code [5]. In this way, adversaries are constantly changing some small portion of their malware code to avoid detection. As a result, signature-based IDSs have to be updated so often to capture modified malware. This never-ending game can cost some companies too much money. ML classifiers can overcome this hurdle by learning the malware's inherent functionality rather than its syntax. In this way, these classifiers cannot easily be bypassed by only changing a small part of the malware's code.

Despite their effectiveness, these classifiers suffer from a serious issue. These classifiers' effectiveness relies on their complexity, and this complexity comes at the cost of resource-intensiveness. As mentioned before, all these classifiers impose drastic overhead on the system vulnerable to security threats to the point that their efficacy is sometimes ignored. For instance, none of these methods can be implemented on an implantable medical device because of their vast power consumption [6]. Although security threats to these devices can be life-threatening, we cannot implement these methods on them. It is because they have limited resources, and their batteries cannot regularly be replaced.

It would be great to determine these classifiers' knowledge to detect intrusions and implement them in more efficient ways. For doing so, we have decided to use different ML interpretability techniques. These methods can derive a signature-based counterpart IDS that is way more efficient and almost as effective. After deriving these signatures, we need to see how well they perform in detecting malicious network traffic.

The best way to evaluate the derived signatures is to test them against real network traffic and see if they are performing satisfactory enough or not. For doing so, we may want to use network traffic from different organizations since network traffic produced by one company may not be enough for evaluation. However, getting extra network traffic and evaluating at our end imposes serious overhead on our available resources. Sending signatures in plain text to third-party companies is not viable since it can divulge confidential information about the security infrastructure. To overcome these issues, we have decided to propose a way to filter network traffic from the encrypted signature.

The absence of a comprehensive method that can be used to interpret these classifiers and then evaluate signatures derived from those led us to research this thesis.

1.2 Contributions

This thesis makes two significant contributions to the field of cybersecurity;

1. Utilizing ML interpretability techniques to derive signatures from trained classifiers.
2. Introducing different methods for evaluation of the derived signatures and comparing them.

Although we have dedicated the bulk of the thesis to familiarizing readers with the required background knowledge, these two contributions form its cornerstones. Furthermore, in later chapters, we discuss the possibility of enhancing some of these algorithms further.

At first, we discuss some basic concepts about the interpretability of different ML models. We also introduce some basic techniques for ML interpretability and go over their pros and cons. One should consider different criteria like local or global interpretability and model-dependent or model-agnostic nature that the ML interpretability technique offers.

Following that, we discuss different types of encryption of signatures that make the filtering of network traffic possible without divulging confidential information in the signature [7]. For doing so, we have tested conventional public-key [8], homomorphic [9], and order-preserving cryptosystems [10]. We have used RSA [11] as an example for a conventional public-key cryptosystem. We have also considered using Ivan Damgård, Martin Gaisler, and Mikkel Krøigaard (DGK) [12] (i.e. a homomorphic cryptosystem named after its creators) and Paillier [13] for implementing homomorphic encryption in this context. Based on our problem's definition, we can use a simplified version of order-preserving encryption that meets our constraints.

Finally, we show how ML interpretability and these signature evaluation techniques relate to one another and give different alternatives for different scenarios based on each method's pros and cons.

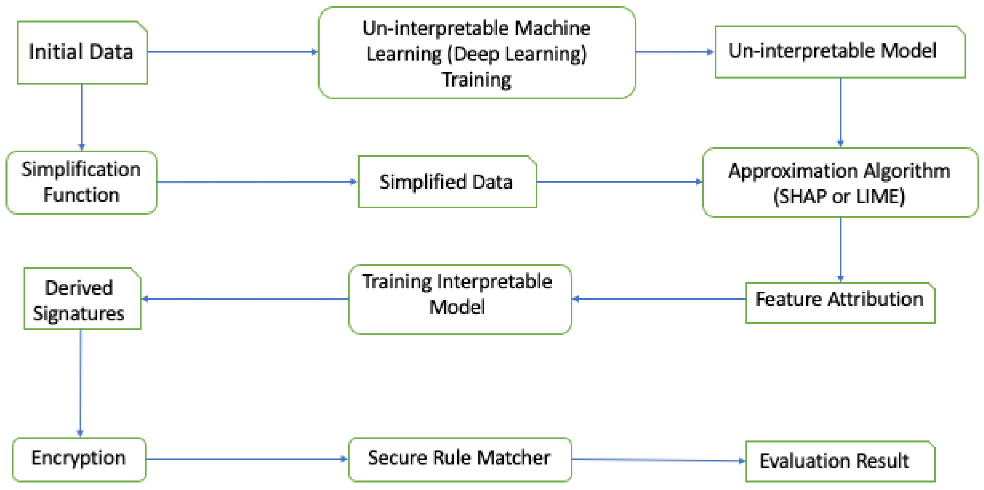

Figure 1.2: The overall proposed scheme in the thesis

In Figure 1.2, after training an uninterpretable model (e.g. Random forest or neural network), we use an approximation algorithm to find what features play the most crucial role in the model's classification. After finding those features, we use them for training another ML model that is inherently interpretable. Using the knowledge gained from the interpretable model, we derive one (or multiple) signatures that can mimic the initial uninterpretable model's functionality. In order to see how well the derived signatures perform in blocking malicious traffic, we use network traffic from other organizations. However, sharing the signatures in plaintext can divulge some confidential information. Therefore, we have to encrypt them without hindering the process of matching. Encrypted signatures are then sent to the organization whose traffic we want to use for evaluation. The result can then be interpreted to see the efficacy of the derived signatures.

1.3 Organization of the Thesis

The remainder of the thesis is organized as follows:

Chapter 2 delves into the background knowledge needed to understand the theses. First, we explain notions in ML interpretability and introduces different techniques for that. Then we discuss what attributes homomorphic cryptosystems should possess and what attributes mainly concern us. Finally, we explain the idea behind order-preserving cryptosystems and their application in our proposed signature evaluation method.

Chapter 3 explains our proposed method for signature generation and puts real-life scenarios in which the proposed methodology can work. It also clarifies each approach's strengths and weaknesses in our methodology and, given specific constraints, helps choose the best one.

Chapter 4 explains the details regarding implementing the proposed methodology for evaluation of derived signatures. This chapter talks about different cryptosystems used in implementation of different schemes for evaluation process. We also compare security and efficiency of each scheme. In comparing efficiency of these schemes, we have provided numerical values for their runtime. Doing so helps the reader better understands the resource-intensiveness of each approach.

Chapter 5 covers conclusions drawn from the thesis and takes a look at how the methodology can be further enhanced. It outlines how this thesis affects the cybersecurity discipline, and in what scenarios one can make improvements to this proposal.

Chapter 2

Background

This chapter explains the keywords and concepts that are important to know before we can propose our methodology. First, it defines ML interpretability and demonstrates its paramount importance in different fields. Then, we go over different methods for achieving this interpretability and compare their pros and cons. After that, we clarify the notion of secure computation and how that relates to our problem. Finally, we go over the idea behind different kinds of encryption (i.e. public-key, homomorphic, and order-preserving) that help us find an answer to our proposed problem.

2.1 Deep Learning Introduction

Deep learning is breaking performance barriers day after day in extraordinary feats that defeat our traditional beliefs on the limits of the computer algorithms. Understanding images, natural language, and generating art are some examples of its enormous capability. In many areas, we see results that challenge traditional approaches or overtake human skills. Deep learning has also been applied to DDoS filtering applications with success.

Deep learning methods, composed of multiple non-linear transformations, generate simpler representations of data combined to form complex hierarchical outputs by using deeper architectures. Deep learning is a subset of machine learning that can automatically represent the world using a hierarchy of simpler representations. Thus, it makes the feature extraction step automatic. Deep learning's significant contribution lies in its ability to extract features from complex data without an explicit feature representation phase using subject-area expertise. Deep learning follows the same general approach as ML, but in this case, the main algorithm is based on artificial neural networks organized in different structures and combinations.

Neural Networks come in many structures and with specific inner workings designed for different applications that show differentiated behaviours. From the simple shallow architectures to the complex multi-layered ones, there is a whole list of possible constructions used in practical applications.

Shallow architectures are architectures with a reduced set of layers and neurons. In contrast, deep architectures have many layers and neurons and can be constructed from simpler networks designed for specialist tasks. Neural networks are the building blocks of the complex Deep Learning architectures that can be trained over very large amounts of data and are responsible for most of the breakthroughs that have been attained by this discipline.

In these shallow architectures, each neuron is connected to all the neurons in its preceding and succeeding layers. Each of these connections has a weight, and we assign a number called bias to each neuron. We first randomly assign a value to each weight and bias in the architecture for training this structure. After that, we input our observation into the structure to what value is returned by structure. The act of inputting our observations values in the structure is called feed forward. We then compare the returned value with the value we expected to get (i.e. actual labels). After that, by using gradient descent or any other optimization algorithm that we see fit, we change the values of those weights and biases to make the structure's output as close as possible to the labels we have. The following figure helps in understanding the whole process:

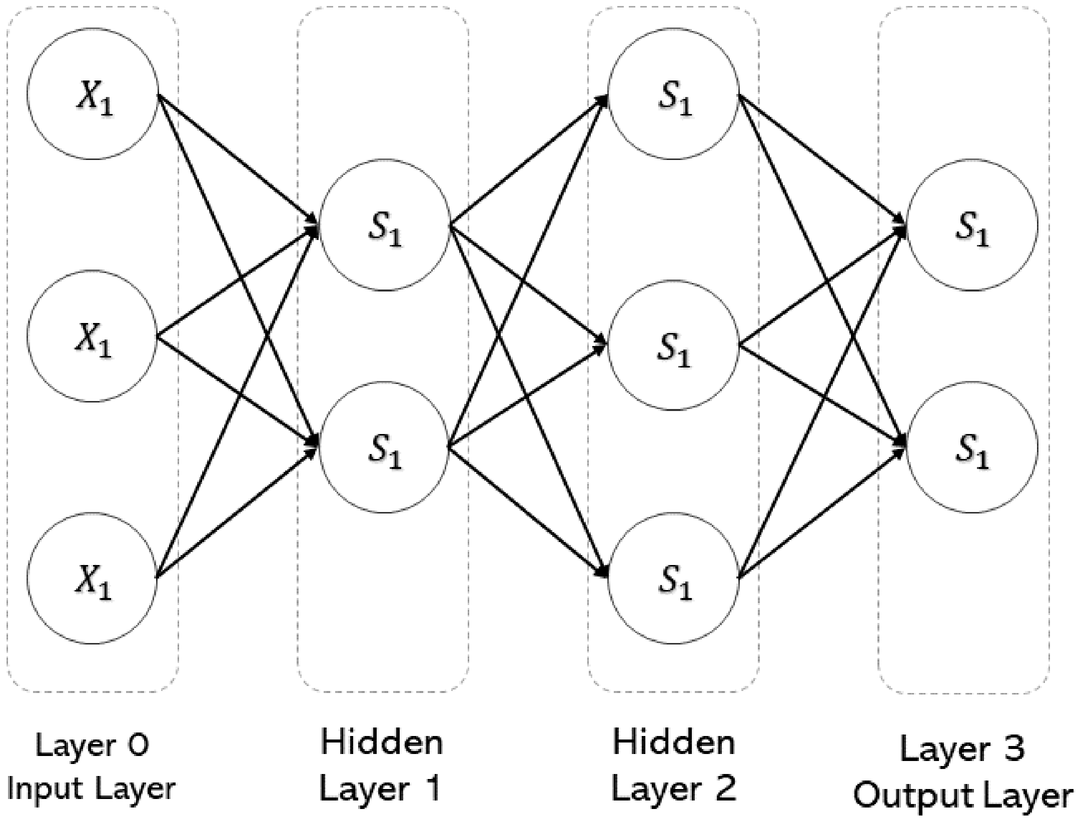

Figure 2.1: Overall structure of an artificial neural network(ANN)

In training an ANN, we start by equating the first layer variables $(x_1, x_2, x_3, ...)$ with the values we have for each feature in our dataset. we then observe the output value returned by the structure (i.e. y_1, y_2 in this case) and then update the weights and biases accordingly.

In deep learning, we will consider an architecture a combination of several simpler networks into a single structure. In this sense, when we describe an architecture, each of the components' structure must be analyzed in detail.

Deep learning can be used for finding an answer to a problem that asks for a mapping from an input vector $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$ to an output vector $\mathbf{y} = \{y_1, y_2, \cdots, y_n\}$. By increasing the size of the network with more layers, with more neurons or with new networks, the algorithm would be able to represent more complex functions and obtain the desired results.

2.2 Machine Learning Interpretability

Understanding ML models has always been imperative for engineers and scientists [14]. Based on the General Data Protection Regulation (GDPR) made by the European Parliament and Council of the European Union [15], an explanation of the model's decision or prediction is mandatory for some agencies. According to that regulation, every person affected by a model's prediction has the right to know why the model made such prediction.

Based on [16], ML interpretability is the ability to explain the inner-workings of a model to humans. Gilpin [17] also defines interpretability as "when you can no longer keep asking why." A number of methods have been proposed so far to achieve ML interpretability. We intend to introduce some of them in this chapter.

2.2.1 Tree Surrogates

In this method [18], we make an approximate flowchart of the algorithm that the ML model follows to make a prediction. A decision tree is trained using the initial data used to train the model and the model's predictions as labels. Since long decision trees themselves can be hard to interpret and generalize [19], one should avoid using unnecessary features. After training the decision tree, the interaction between variables is assumed to indicate the complex model's internal process. In this method, we surmise each variable's effect in prediction is related to the entropy [20], Gini index [21], or any other criterion based on which the splitting happens in the tree. Although there is almost no rigorous mathematical proof behind this method, it is shown that this method can be useful in some disciplines. One such discipline is convolutional neural networks (CNN) [22, 23]. Another such discipline is interpretability of an artificial neural network (ANN). The figure below shows a surrogate decision tree for an ANN classifier trying to find one of the most critical features in classifying network traffic.

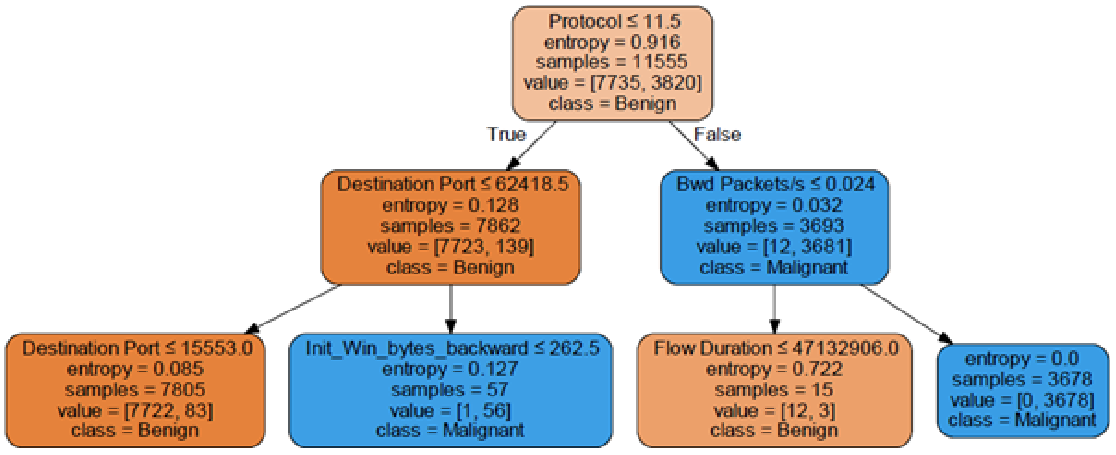

Figure 2.2: A decision tree surrogate visualization

Decision tree surrogate can act as a flowchart of the function learned by a more complex machine learning. It can be seen in Figure 2.2 that the feature Bwd Packets/s (i.e. backward packets per seconds) has the lowest entropy amongst all other features. Backward packets are notifications to the sender generated by a congested router. This indicates that it is the most distinguishing feature for deciding whether network traffic is malignant or benign.

2.2.2 Shapley Regression Values

In this method [24], we try to find each feature's importance based on its presence or absence. However, since the features may have subliminal effects on each other, this method proposes retraining models using all the subset of features with and without the specific feature. If we denote the set of all the features with F, subset of features with S, and the specified features as $\{i\}$, the comparison formula between two models becomes $f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)$. Therefore, since we have to consider all the subsets $S \subseteq F\setminus\{i\}$, the Shapely value as feature attribution is calculated as weighted average of the formula for all possible subsets:

$$\phi_i = \sum_{S\subseteq F\setminus\{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!}\left[f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)\right] \tag{2.1}$$

In this way, we can assign a value to each feature representing its contribution in helping the ML model make the decision.

2.2.3 Local Interpretable Model-Agnostic Explanations (LIME)

To get to know what this algorithm does, we must first define local and global interpretability. Local interpretation of a model refers to explaining the reason behind a model's prediction for a particular instance. However, global interpretability is concerned with explaining the overall approach that the model takes for prediction [25]. Local interpretability is used when we try to explain the model's behaviour for a specific decision. On the other hand, global interpretability is used to detect bias and compare two competing models [16].

The primary rationale behind this method is that a model's local interpretation (fidelity) can sometimes help us know its global interpretation (fidelity) [14]. In this manner, this method first finds an interpretable representation of the data. This representation is a binary vector whose content is dependent on the domain on which the model is being used. For instance, in text classification, this binary representation can indicate the presence or absence of a word, while the model itself may use something more complex like word embeddings. After that, it tries to train an explainer over interpretable representation. The explainer can then be used to see to what extent each feature contributes to the model's prediction. The detailed process is as follows:

1. We find an interpretable representation (binary vector) of the data denoted as x'.

2. We sample a random number of non zero elements of x' and call it z'. The number of draws and the act of drawing happens in a random uniform fashion.

3. We find z (the original space representation) from z' and use f(z) (the model prediction) as our explainer label.

4. We define π_x as an exponential kernel to measure locality of z to x and also weigh the model prediction on z.

5. We define Ω function that is used to return the complexity of the explainer. That is because there is an inherent trade-off between complexity and interpretability, as the authors suggest.

6. We then try to minimize the cost function:

$$\xi(x) = \underset{\mathrm{g} \in \mathrm{G}}{\operatorname{argmin}} \; \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{2.2}$$

The $\pi_x(z)$ is an exponential kernel defined on distance function D with width σ . The expression for that function is:

$$\pi_x(z) = exp(-D(x,z)^2/\sigma^2) \tag{2.3}$$

The f function is the model itself, and g is any linear model such that $g(z') = w_g.z'$, i.e. the explainer we are looking for. The loss function is also defined as follows:

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in \mathcal{Z}} \pi_x(z) \; (f(z) - g(z'))^2 \tag{2.4}$$

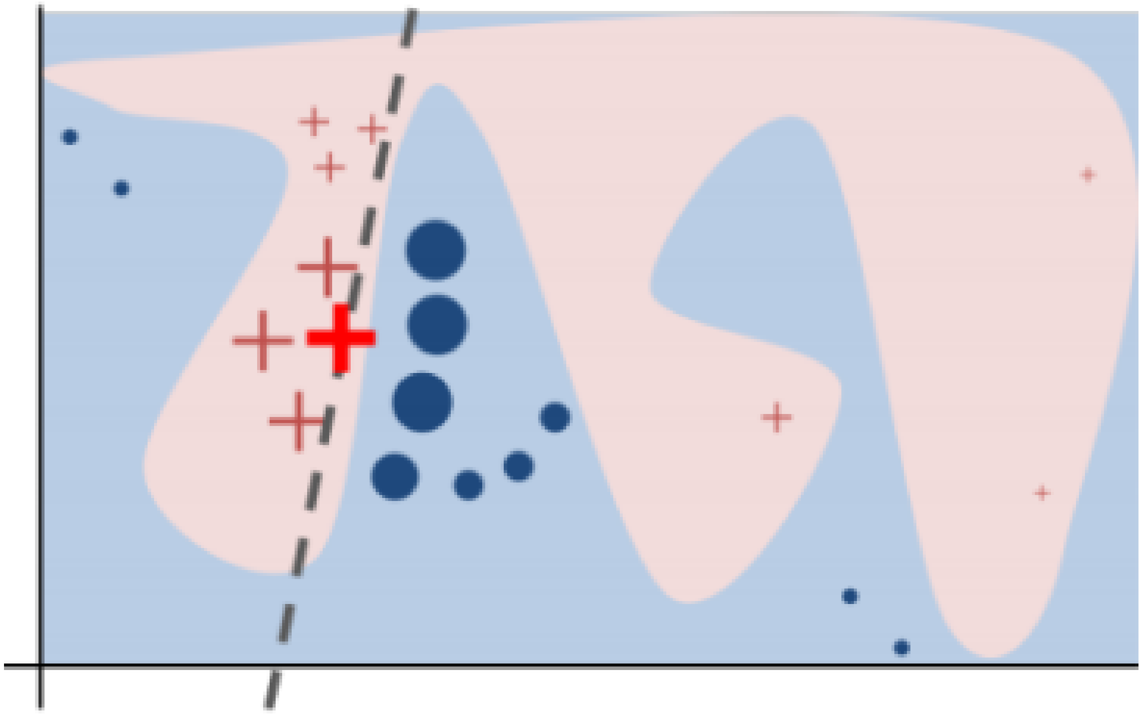

Figure 2.3: A simple visualization showing LIME intuition

In Figure 2.3, pink and blue areas represent the complex model. The explainer is the dashed line found by sampling around instances. The π_x function weighs the samples' predictions, and the size of the crosses and dots represents that weight in the figure. Figure adapted from [14].

In this manner, finding a useful explainer can itself be thought of as an optimization problem that can be solved by ML methods.

Although the function g found in this manner is a local linear explainer for a more complex machine learning model, it can sometimes give us insight into the model's algorithm.

2.2.4 SHAP (SHapley Additive exPlanation) Values

This method expands upon Additive feature attribution methods. First, we need to define what these methods are in order to be able to explain SHAP values. Additive feature attribution methods have the form of a linear regression function on binary variables.

As stated in (Lundberg and Lee, 2017) [26], the best explanation for a simple model is the model itself. However, when the model is complicated, we cannot use the model algorithm for interpreting it. In that case, we try to find a local linear approximation for that model and call it explainer. Authors of the paper (Lundberg and Lee, 2017) [26] argue that all the methods in the additive feature attribution class (e.g. DeepLIFT [27], LIME [14], Shapley regression values [24], ...) try to find a single unique solution in different ways. This unique solution has the following properties:

1. **Local accuracy** states that the explainer should match the complex model in some specific locality.

2. **Missingness** says that when a feature is absent in making the model prediction, no impact should be attributed to that feature (neither positive nor negative).

3. **Consistency** contends that if a feature's contribution to a model's prediction is higher than its contribution in another model, then its impact coefficient should be higher in the second model. This can be stated in the following mathematical statement:

   Let $f_x(z') = f(h_x(z'))$ and $z' \setminus i$ denote $z'_i = 0$. For any two models f and f', for all inputs $z' \in \{0,1\}^M$, if ${f_x}'(z') - {f_x}'(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i)$, then $\phi_i(f', x) \geq \phi_i(f, x)$. M is the number of simplified features and $x = h_x(x')$.

The explainer that conforms with the definition and properties given above should have the following form:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)] \tag{2.5}$$

$|z'|$ is the number of ones in z' and $z' \subseteq x'$ represents all the binary vectors z' whose ones are a subset of the ones in x'.

Following this line of reasoning, the paper's authors propose SHAP values as a unified measure for feature importance. First, they get the model's expected value when we do not know any input feature's value. After measuring this expected value, they add value from the dataset for a feature and see how much the expected value is changed. They do this process until there is no feature left. This process is done by going through all the possible ordering by which the features can be added. That is because the order by which the features are added matters in a nonlinear ML model. The SHAP value for each feature is then the average of all such ordering.

2.3 Snort-like Signatures

In this section, we demonstrate the general format of the snort-like signatures. These signatures perform traffic filtering using different network attributes. They also support defining the action that needs to be done if the signature matches network traffic. the following figure shows a simple snort-like signature with its different parts annotated.

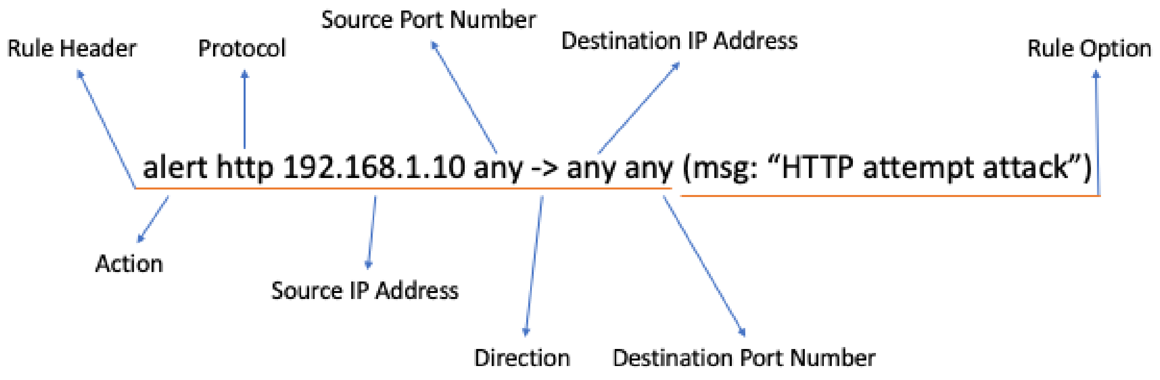

Figure 2.4: Example of a Snort-like signature

Figure 2.4 depicts an example of a simple snort signature. The section outside the parenthesis is called the rule header. It can contain different constraints filtering traffic based on various network attributes. in this example, the rule header filters traffic based on the protocol used and source and destination port numbers and IP addresses. We can also specify a more specific range for port numbers (e.g., source port numbers between 100 and 500). multiple IPs can also be filtered using subnet masking. The first keyword used in the rule header is the action that needs to be done when the signature matches traffic based on the given constraints inside its rule header. In this case, the signature shows an alert for each packet it filters. This alert's content is also defined in the rule option section (i.e., the section inside the parenthesis).

2.4 Homomorphic Encryption

The main idea of homomorphism comes from abstract algebra. In order to be able to explain it, first, we need to explain some definitions from that discipline. The first thing that we need to know about is the group.

Definition 1 (Group). *In abstract algebra, a group is a set of elements with a specific operator that satisfies group axioms.*

The group axioms are closure, associativity, identity, and invertibility. Any group must have all these properties. Each of them is explained below.

Property 1 (Closure). *The group set must be closed under its operator. For instance, the set of Integers $\mathbb{Z}$ are closed under addition. It means that if we add any two integers, the result is also an integer. Therefore, performing the operation (that is addition in this case) on two elements in the set (which are integers), we get an element already in the set.*

Property 2 (Associativity). *The order in which you apply the operator to three elements should not matter. If we denote the group operation as $*$, and the group elements as a, b, and c, this property can be shown in the following mathematical expression:*

$$(a * b) * c = a * (b * c)$$

Property 3 (Identity)**.** *The group set should have an element called the identity element. If combined with another element under the operator, this element gives the initial element as a result. This can be stated as follows:*

$$a * i = i * a = a$$

In the above expression, i indicates the identity element, and a indicates another element in the group (it can also be identity element). Although a group does not need to have the commutative property ($a * b = b * a$), its identity element should have this property.

Property 4 (Invertibility)**.** *There should be an inverse element for every element in the group set. Any element combined with its inverse under the operator produces the identity element. This element also should have commutative property even if the group does not have it. This property is shown below in the mathematical expression:*

$$a * a^{-1} = a^{-1} * a = i$$

The a superscripted by -1 indicates the inverse element.

Definition 2 (Homomorphism)**.** *A homomorphic function is a mapping from one group to another. This mapping should preserve group properties in both of them. If we denote two elements of the first group as a and b and we denote the function as f, $f(a)$ and $f(b)$ should be the second group's elements. We show the operator in the first group as $\cdot$, and the second group operation is shown as $*$. Based on the group properties mentioned above, we should have $x \cdot y = z$, and z is another element in the first group. We should also have $f(x) * f(y) = f(z)$. Since we had $x \cdot y = z$, we can substitute z and get $f(x) * f(y) = f(x \cdot y)$. Any function that preserves this equality is called a homomorphic function from the first group to the second one.*

If we think of plaintext space as a group set and ciphertext space as another group set, the homomorphic encryption is the homomorphism between these two groups. As a result, if we denote two plaintexts as P_1 and P_2, the encryption function as E, and their corresponding ciphertexts as C_1 and C_2, we can show that the homomorphism is only preserved if the following expression is satisfied:

$$C_1 * C_2 = E(P_1 \cdot P_2)$$

Since the four fundamental binary operations can be done using only addition and multiplication, we only consider those two operations as the group operation. Based on this, homomorphic cryptosystems fall under two major categories [28]. Fully homomorphic and partially homomorphic. Fully homomorphic cryptosystems support addition and multiplication at the same time. Therefore, if we perform the proper operation on two ciphertexts, we get the encrypted result in plaintext. The same is also true for multiplication. Gentry's circuits are the first and only example of fully homomorphic cryptosystems [29].

Partially homomorphic encryption, on the other hand, support only one of these two operations. They only support either addition or multiplication. DGK and Paillier are examples of homomorphic cryptosystems that only support addition (additive homomorphism). RSA and ElGamal are examples of multiplicative homomorphic cryptosystems.As we explain in Chapter 4, the problem of secure rule matching can be reduced to a secure comparison of two integers. As a result, we can only focus on additive homomorphic cryptosystems. The reason is that in the computer, all the four fundamental binary operations (addition, subtraction, multiplication, division) are done using addition [30]. Furthermore, they are not as resource-intensive as a fully homomorphic cryptosystem. Therefore, we only focus on the two additive homomorphic cryptosystems mentioned above (DGK and Paillier).

2.4.1 DGK Public-key Cryptosystem

Damgard, Geisler and Kroigaard proposed this cryptosystem in 2007 [12, 31]. They had to create a new cryptosystem for their proposed solution to secure comparison of two integers. This cryptosystem has lots of application from cloud computing [32] to smart metering systems [33].

Like any other cryptosystem, DGK also works with three main functions (i.e. Key generation, encryption, and decryption). We now explain the processes involved in each of them.

Key Generation: First, take three integers k, t, and ℓ such that $k > t > \ell$. Then, We need two t-bit primes, namely v_p and v_q and two other primes of length $k/2$ that we call p and q such that

$$v_p \mid p - 1$$

$$v_q \mid q-1$$

In this way, if we multiply p and q we generate a k-bit RSA modulus that we denote as n. After that, we find the smallest prime having more than $\ell+2$ bits (greater than $2^{\ell+2}$) and call it u. After that, we find two random numbers g and h from $\mathbb{Z}_n^*$. The multiplicative order of h and g modulo p and q should be v_pv_q and uv_pv_q, respectively. The multiplicative order of a number (denoted as b here) modulo m, is the smallest positive number a such that

$$b^a \equiv 1 \pmod m$$

It's obvious that b should be coprime to m. Otherwise, we cannot get a remainder of one no matter what the value of a is. The public key is

$$pk = (n, g, h, u)$$

And the secret key (private key) is

$$sk = (p, q, v_p, v_q)$$

In this way, the plaintext space becomes $\mathbb{Z}_u$ and the ciphertext space becomes $\mathbb{Z}_n^*$.
Encryption: Apart from the message itself, the encryption function also relies on a $2t$-bit integer random number. That is because we want the same key to produce different ciphertext every time it is applied to the same message using the same key. In this way, the secrecy of the cryptosystem is maintained. The formula for encryption is given below

$$E_{pk}(m,r) = g^m h^r \bmod n \tag{2.6}$$

Decryption: If we denote the ciphertext generate by encryption as $E_{pk}(m,r)$, for decryption we have to perform two steps:

1. First, we raise both sides of the equation 2.6 to the power of v_pv_q. Notice that both v_p and v_q are only known to the person that has access to the secret key. The equation then becomes:

   $$E_{pk}(m,r)^{v_pv_q} = (g^m h^r)^{v_pv_q} \bmod n$$

   Since the order of h is v_pv_q, we can replace $h^{v_pv_q}$ with 1 and get:

   $$E_{pk}(m,r)^{v_pv_q} = (g^{v_pv_q})^m \bmod n \tag{2.7}$$

Because the order of $g^{v_p v_q}$ is u, there is an injection from possible values of m to values of $(g^{v_p v_q})^m \bmod n$.

2. Then, by solving the equation 2.7, we can easily find m. We can also create a lookup table along with the key generation process to facilitate finding m. This lookup table should consist of all the values $(g^{v_p v_q})^i$ along with the i for $0 \leq i \leq u$.

Homomorphic Properties: As mentioned before, DGK is an example of an additive homomorphic cryptosystem. The homomorphic characteristics of this cryptosystem are as follows:

1. **Homomorphic Addition**: This property states that if we calculate the remainder of the product of two ciphertexts modulo n ($c_1 * c_2 \bmod n$), we get the encrypted summation of their corresponding value in plaintext. This is written below in mathematical expression:

$$E_{pk}(m, r) \cdot E_{pk}(m', r') \bmod n = E_{pk}(m + m' \bmod u, r + r') \tag{2.8}$$

2. **Homomorphic Multiplication of Plaintext**: Using this property, we can raise a ciphertext to the power of a plaintext and get their product value in the ciphertext. This is written below:

$$E_{pk}(m, r)^{m'} = E_{pk}(mm' \bmod u, rm') \tag{2.9}$$

   Notice that this property only support multiplication of a ciphertext and plaintext. It does not cover multiplication of two ciphertext. That kind of multiplication is only supported in fully or multiplicative homomorphic cryptosystems. The reason we have mod u in both 2.8 and 2.9 is because the plaintext space, as mentioned before, is $\mathbb{Z}_u$.

Although we have only given properties regarding addition or multiplication, we can easily tweak this cryptosystem to support subtraction as well. We are interested in subtraction because we want to use this cryptosystem to compare two integers directly. Doing so is only possible through their subtraction and observing the sign of the result. Even though subtracting a from b is equal to adding b to minus a, we cannot multiply a ciphertext with minus one to get its corresponding negative value. That is because, as mentioned before, the message space is $\mathbb{Z}_u$, and this space does not include minus 1. The remedy to this problem is postponed until Chapter 4.

2.4.2 Paillier Encryption

Pascal Paillier proposed this cryptosystem in 1999. Just like RSA, this cryptosystem works on the assumption that it is computationally infeasible to find n-th residue modulo of a number that is of unknown factorization. Although the modulo used in RSA is n and in Paillier is n^2, the idea is pretty much the same. First, we need to define some of the functions used in this system's key generation step before proceeding any further.

Definition 1 (Euler's totient function)**.** *For every positive integer n, this function returns the number of integers in the interval $[1, n]$ that are relatively prime (coprime) to n. This function is denoted by $\phi(n)$.*

This function has some useful properties that make its calculation easier for some specific numbers. These properties are as follows:

Property 1. *For every prime number p, $\phi(p) = p - 1$.*

This property is evident because every prime number has exactly two positive divisors; one and itself.

Property 2. *This property states that if we have $\gcd(m, n) = 1$, then $\phi(mn) = \phi(m)\phi(n)$. GCD stands for greatest common divisor (i.e. the smallest number that divides two or more numbers specified inside gcd operator).*

Proof. For proofing this property, we need to show that the number of elements in the set $\mathbb{Z}^*_{mn}$, is equal to the number of ordered pairs (x, y) where $x \in \mathbb{Z}^*_m$ and $y \in \mathbb{Z}^*_n$. This can be done by showing a bijection between these two sets. If we assume that the ordered pairs are of the form $(z \bmod m, z \bmod n)$, where $z \in \mathbb{Z}^*_{mn}$, finding the inverse mapping from these ordered pairs to z would be the same as solving the following system of the equations:

$$z \equiv x \pmod m$$

$$z \equiv y \pmod{n}$$

From the Chinese Remainder Theorem [34], we know that there is only one solution to this system $\mod mn$. We also know that the only way that a number is coprime to mn is by being coprime to both m and n at the same time. This proves the fact that $|\mathbb{Z}^*_{mn}| = |\mathbb{Z}^*_m| \cdot |\mathbb{Z}^*_n|$. □

Property 3. *This property contends that if p is prime and $k \geq 1$, then $\phi(p^k) = p^k - p^{k-1} = p^k(1 - \frac{1}{p})$.*

Proof. Since p is prime, the only way for $\gcd(p^k, m)$ to be any number other than one is for m to be multiple of p. We are only interested in the multiples of p that are less than or equal to p^k. These multiples are $p, 2p, 3p, \ldots, p^{k-1}p = p$. As you can see there are p^{k-1} of such multiples. Any number between 1 and p^k except these multiples is coprime to p^k. Hence, $\phi(p^k) = p^k - p^{k-1}$. □

Definition 2 (Carmichael's function)**.** *This function associate a positive integer m, to every positive integer n, such that*

$$a^m \equiv 1 \pmod{n}$$

for every integer a in the set $\mathbb{Z}^*_n$. Carmichael's function is denoted with λ.

Just like Euler's totient function, this function has some useful properties that facilitates calculation of its value for some integers (specially prime ones). One of these property is directly used in key generation phase of Paillier cryptosystem. These properties are as follows:

Property 1. *For every prime number p, the value of this function is:*

$$\lambda(p) = \phi(p) = p - 1$$

Property 2. *If we multiply two distinct prime numbers, namely p and q, the value of the Carmichael's function for their product is calculated using the following formula [35]:*

$$\lambda(pq) = \operatorname{lcm}(\phi(p), \phi(q)) = \operatorname{lcm}(p-1, q-1)$$

LCM stands for lowest common multiple. The operator lcm calculates this smallest common multiple between all the numbers passed to it.

Just like DGK, this cryptosystem consists of three major phases in its functionality (i.e., key generation, encryption, decryption). We now explain the steps that comprise each phase.

Key Generation: First, we choose two prime numbers, namely p and q. For these two numbers, the following equality should hold:

$$\gcd(pq,(p-1)(q-1))$$

This equality is always true for primes of equal length. After finding these two primes, we calculate their product and call it n. Then we calculate Carmichael's function value for n. As told before, the Carmichael's function value for multiplication of two prime numbers is equal to $\operatorname{lcm}(p-1,q-1)$. Then, we select a random integer from the set $\mathbb{Z}_n^*$. After that, we have to ensure that n divides the order of g. For doing that, first, we need to define a function in the following form:

$$L(x)=\frac{x-1}{n}$$

After that, we need to check the existence of a modular multiplicative inverse in the following form:

$$\mu = L(g^\lambda \bmod n^2)^{-1} \bmod n$$

The modular multiplicative inverse of a number is denoted by writing -1 as a superscript of that specified number. It should hold the following property:

$$aa^{-1} \equiv 1 \pmod n$$

If μ exists, it's guaranteed that n divides the order of g. After finding μ, the public and private keys are as follows:

$$pub = (n,g)$$

$$priv = (\lambda,\mu)$$

Encryption: Suppose we want to encrypt a message m (it should be less than n); first, we need to choose a random number r from the set $\mathbb{Z}_n^*$. After that, the ciphertext is calculated as follows:

$$c = g^m \cdot r^n \bmod n^2$$

Decryption: Given a valid ciphertext c (i.e. a number from the set $\mathbb{Z}^*_{n^2}$), the plaintext is calculated as follows:

$$m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$$

Homomorphic properties: This cryptosystem has additive homomorphic properties, just like DGK. Below we explain each of them. We denote the encryption and decryption function by E and D, respectively.

1. **Homomorphic Addition**: If we decrypt the product of two ciphertext modulo n^2, the result is equal to the addition of their corresponding plaintext modulo n. This is shown below in mathematical expression:

$$D(E(m_1, r_1) \cdot E(m_1, r_1) \bmod n^2) = (m_1 + m_2) \bmod n$$

2. **Homomorphic Multiplication of Plaintext**: Using this property, we can find out the result of the multiplication of ciphertext and plaintext. If we raise the ciphertext to the power of plaintext and calculate its residue modulo n^2, the resulting number after decryption is equal to the product of the two plaintext messages modulo n. The equivalent of this statement is written below using mathematical expressions:

$$D(E(m_1, r_1)^{m_2} \bmod n^2) = m_1 m_2 \bmod n$$

Like DGK, the homomorphic additive property of this cryptosystem can be used to perform a comparison of integers. One of the most important applications of these cryptosystems is secure integer comparison. In secure integer comparison, both parties involved in the comparison should not find anything about the other party's input. However, in our case scenario, only one party is untrustworthy. Therefore, we do not need to deal with all the complications regarding the implementation of secure integer comparison.

2.5 Order Preserving Encryption

As its name suggests, this cryptosystem focuses on maintaining the relative order of numbers even after encryption. If we have two plaintext messages m_1 and m_2 such that $m_1 \geq m_2$, and their corresponding ciphertext is c_1 and c_2, we can know for sure that $c_1 \geq c_2$ if the ciphertexts were calculated using order-preserving cryptosystem. It is evident that this cryptosystem only works on numeric data. Although finding a cryptosystem scheme that maintains this property may seem easy at first, it has its challenges.

First of all, no order-preserving cryptosystem can be asymmetric. To prove this claim, assume that we have such a cryptosystem. We should have a public key used for encryption and a private key used for decryption. In this scheme, just like any other asymmetric cryptosystem, it is fair to assume that an adversary can access public key and any ciphertext. Using the public key, the adversary cannot decrypt the ciphertext that he has. However, he can encrypt any number that he chooses. Suppose that the adversary encrypts a random number that he generated using the public key. Since the cryptosystem used in this scenario is order-preserving, he knows he can compare the ciphertext generated by him with the one he intercepted and determine the relationship between the two numbers. If the adversary keeps doing that, he can get to the actual plaint text with the time complexity of $O(\log n)$ using a binary search.

Secondly, there is no way that this cryptosystem is INDistinguishable under Chosen-Plaintext Attack (IND-CPA). In order to define what IND-CPA is, we need to lay the groundwork of a hypothetical scenario. Suppose that we have an adversary with two different plaintext messages on one side and a trustee with access to the cryptosystem's private key on the other side. The adversary sends both of the messages to the trustee and asks for their ciphertext. The trustee only sends one ciphertext. At this stage, the adversary cannot know for sure that what plaintext produced what ciphertext. However, if this process continues for some time, he can know for sure what ciphertext is for what plaintext. This is because of the inherent property of the order-preserving cryptosystem (i.e. maintaining order even after encryption). In any other cryptosystem that is IND-CPA, the adversary's guess is approximately as good as random guessing. That is because those cryptosystems randomize generated cyphertext. Hence, even using the same key and the same plaintext gives a different ciphertext each time.

Despite the problems that were just mentioned, the benefits of order-preserving encryption are too much to ignore. The main application of these cryptosystems is in databases. Even though the interfaces provided by the database management system (DBMS) usually provide an acceptable level of security, one may access the database files on the disk by bypassing those interfaces. If that happens, none of the security measures taken by the DBMS help in protecting the data. Conventional cryptosystems are also not suitable for encrypting database files since we have to decrypt them in retrieving information for each query.

So far, the only feasible solution for this problem is using an order-preserving cryptosystem. By maintaining the order, these cryptosystems make it possible for queries to be performed even on encrypted data. In that way, predicates containing MIN, MAX, range, and COUNT can be run without decryption. ORDER BY and GROUP BY operation are also applicable to the encrypted data. Furthermore, they provide an additional layer of security. As a result, even if one gains access to the actual database files on the disk, no information is divulged.

Following are different methods for implementing an order-preserving cryptosystem:

1. **Addition of random numbers [36]**: In this scheme, the encryption of any integer is done by adding that many random numbers. If we denote ciphertext with c, and the integer to be encrypted with p, the encryption formula becomes $c = \sum_{j=0}^{p} R_j$. The R_j in this formula denotes the jth number generated by the pseudorandom number generator R. Although easy, this scheme has two significant drawbacks. First, invoking the pseudorandom number generator p times for encryption and decryption is computationally-intensive. Second, since the ciphertext distribution resembles plaintext distribution, this scheme is prone to estimation exposure.

2. **Daisy-chained polynomial functions [37]**: For encryption using this scheme, we use multiple monotonic increasing polynomial functions, usually of first or second order. The key in this scheme is the file containing the coefficients of these polynomials. For encrypting a value, we give the number to be encrypted as input to the first function. Then, we give the output of the first function as the input of the second. This process continues until the last function. For decryption, we follow the same procedure in reverse order (i.e. starting from the

last function and ending in first). Although the distribution of ciphertext may not be similar to the distribution of plaintext in this scheme, the ciphertext distribution is dependent on the distribution of plaintext. Therefore, conclusions can be made about the distribution of plaintext by studying the distribution of ciphertext.

3. **Encryption using a lookup dictionary [38, 10]**: This scheme uses a dictionary using (key, value) pairs for encryption and decryption. In this scheme, the key is the plaintext, and the value is the ciphertext associated with that plaintext. Encryption is then achieved by finding the value in the lookup table for a given key, and decryption is finding the corresponding key for a given value. This lookup table can be created in a way that obfuscates the distribution of plaintext. Suppose that our plaintext space consists of n distinct integer values. First, we choose an arbitrary distribution. Then, we sample n distinct values from the chosen distribution. After that, we sort the sampled numbers and the numbers in plaintext space in two different lists. The lookup table is then created by putting these two lists next to each other. The ciphertext for the ith element in the first list is given by the ith element in the second list and vice versa. Despite its simplicity, this scheme has two major problems. First, the lookup table that acts as the encryption key can be large (it has twice as many unique values as there is in plaintext space). Second, if we need to add a new value to plaintext space, the whole table may need to be created from scratch.

As mentioned before, order-preserving cryptosystems mainly developed to be applied in databases. However, since we aim to use them for encrypting network signatures, some of the problems associated with the schemes explained above may not be as important as they are in database management systems (DBMS).

For instance, the problems mentioned for the scheme that uses a lookup dictionary are not that big of a deal in our context. That is because all the network packets' attributes have a predefined range. Hence, we may never need to recreate the lookup table from scratch. The table's size is also negligible due to the limited number of unique values used in network signatures. Therefore, we choose this scheme for encryption of network signatures. It is faster, easier to implement, and does not divulge any information about the plaintext distribution.

Chapter 3

Proposed Methodology for Signature Generation

3.1 Introduction

As mentioned before, we wanted to see if it is possible to derive a signature (or multiple signatures) that can approximately perform as well as a machine learning model in classifying network traffic. To do so, first, we need to have a reliable machine learning model. The process of training a model always starts with finding data. Therefore, we need to start looking for a dataset that contains benign and malignant network traffic along with different attributes associated with network packets.

The dataset that we have used is the dataset provided by the University of New Brunswick. We have used distributed denial of service (DDoS) attacks provided by (Sharafaldin et al., 2019) [39].

This dataset consisted of eleven different CSV files, each for a different kind of DDoS attack (e.g. DNS, UDP, MSSQL, . . .). Due to the limited resources available, we have decided to go with a portion of one of these CSV files. The file that we chose consists of DNS DDoS attacks and benign network traffic.

3.1.1 DNS DDoS Attacks

Denial of service (DoS) attacks are malicious attempts to disrupt a network infrastructure's normal function. These attacks aim to overwhelm the network to the point that it cannot serve the clients anymore. In some cases, these attacks do not originate from a single source. The adversary may use multiple compromised systems to initiate and carry on the attack on the specific network. In such a case, the attack is considered a distributed denial of service (DDoS). Sometimes an adversary uses DNS protocol to send numerous domain name resolution requests to the victim to overwhelm its network infrastructure. This kind of attack is called a DNS DDoS attack. The adversary may forge its DNS request packets' source IP address and change them to the victim's IP address. In that way, since the victim sees itself as the source asking for a response, send the response packet to itself and further increase its workload. This technique of changing source IP address to the victim's IP address is called Reflection, and it may be used with protocols other than DNS (e.g. MSSQL, NTP, TFTP, . . .).

3.2 General Methodology

Initially, the CSV file for DNS DDoS attack consisted of eighty-eight columns (eighty-seven features and one label column). This CSV file has 5074413 rows (instances) in total. 5071011 of these instances were DNS DDoS attacks, and the rest (3402) was benign traffic. We have sampled 100000 instances from malicious traffic to make the training process tractable. Since the data is heavily imbalanced ($\approx 96.71\%$ of the data is malicious and the rest is benign), we have decided to combine all the benign traffic from the other 10 CSV files and append it to the file that is being used for training. After this coalition, we end up with 84340 benign traffic. Hence, $\approx 54.24\%$ of the data becomes malicious, and the rest ($\approx 45.76\%$) becomes benign.

3.2.1 Data Preprocessing and Training the Classifier

After sampling data from designated dataset, we remove unnecessary columns. By unnecessary, we mean the columns that even if we remove, we will not see a significant reduction in the accuracy of the model trained on them. These columns are the ones that are highly correlated to other columns or have mostly zeroes as their values. Doing so reduces training time drastically, and as a result, we can spend more time on hyperparameter tuning of the model.

For this phase, we have used Pandas' profiling library. Using this library, we can detect features that have a high correlation. We can also find features that are mostly zero. We then keep only one of the columns that have a correlation coefficient of more than 0.9. Likewise, columns that are zero 98% of the time are removed. After this phase, the number of columns reduces to 32 from 88.

For the training process, we have decided to use Artificial Neural Network (ANN). The reason for that choice is the notoriety of these models in being uninterpretable. This model has one input layer, two hidden layers, and one output layer. Since these models are complex and can easily be over-fitted when the data's size is not so big, we do not make this model deep. The input layer has 31 units (i.e. the number of columns minus the label). Each hidden layer has ten rectified linear units (ReLU), and the function used in the output layer is sigmoid. Since we try to train a binary classifier, we only need one unit in the output layer. Below is the graph that shows the overall architecture of the classifier:

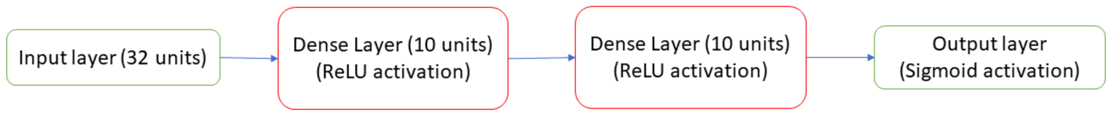
Input layer (32 units)
Dense Layer (10 units) (ReLU activation)
Dense Layer (10 units) (ReLU activation)
Output layer (Sigmoid activation)

Figure 3.1: Architecture of the ANN trained as NIDS

For training the model in Figure 3.1, we have used batch size 32 and 10 epochs. Twenty percent of the data is reserved for testing and the rest is used for training. For dividing the data into training and testing set, we have used stratified sampling (i.e. maintaining the relative frequency of each class in training and testing set).

3.2.2 Machine Learning Interpretability

After training the model, we use the LIME and SHAP algorithms discussed in Chapter 2 as our ANN classifier's interpretability technique. Using these two algorithms, we can find out the features that are playing the most important part in helping the classifier predict a network packet is malicious or benign. The overall procedure for this scheme is shown below:

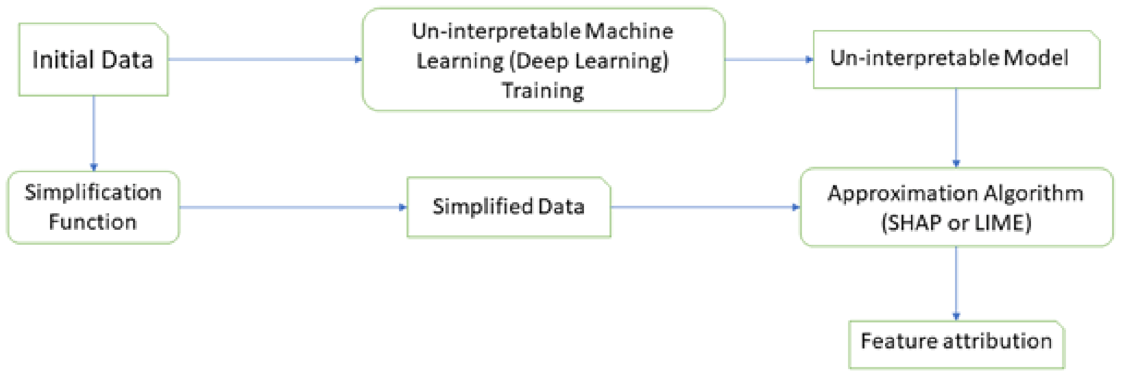
Initial Data
Un-interpretable Machine Learning (Deep Learning) Training
Un-interpretable Model
Simplification Function
Simplified Data
Approximation Algorithm (SHAP or LIME)
Feature attribution

Figure 3.2: Overall procedure for ML interpretability

As mentioned before, the simplification function is dependent on the specific discipline on which the machine learning is applied. Hence, its definition is almost always embedded as a preliminary step in the library implementing the approximation algorithm.

SHAP

First, we use SHAP as our approximation algorithm. The feature attribution graph that is returned by the SHAP algorithm is shown below:

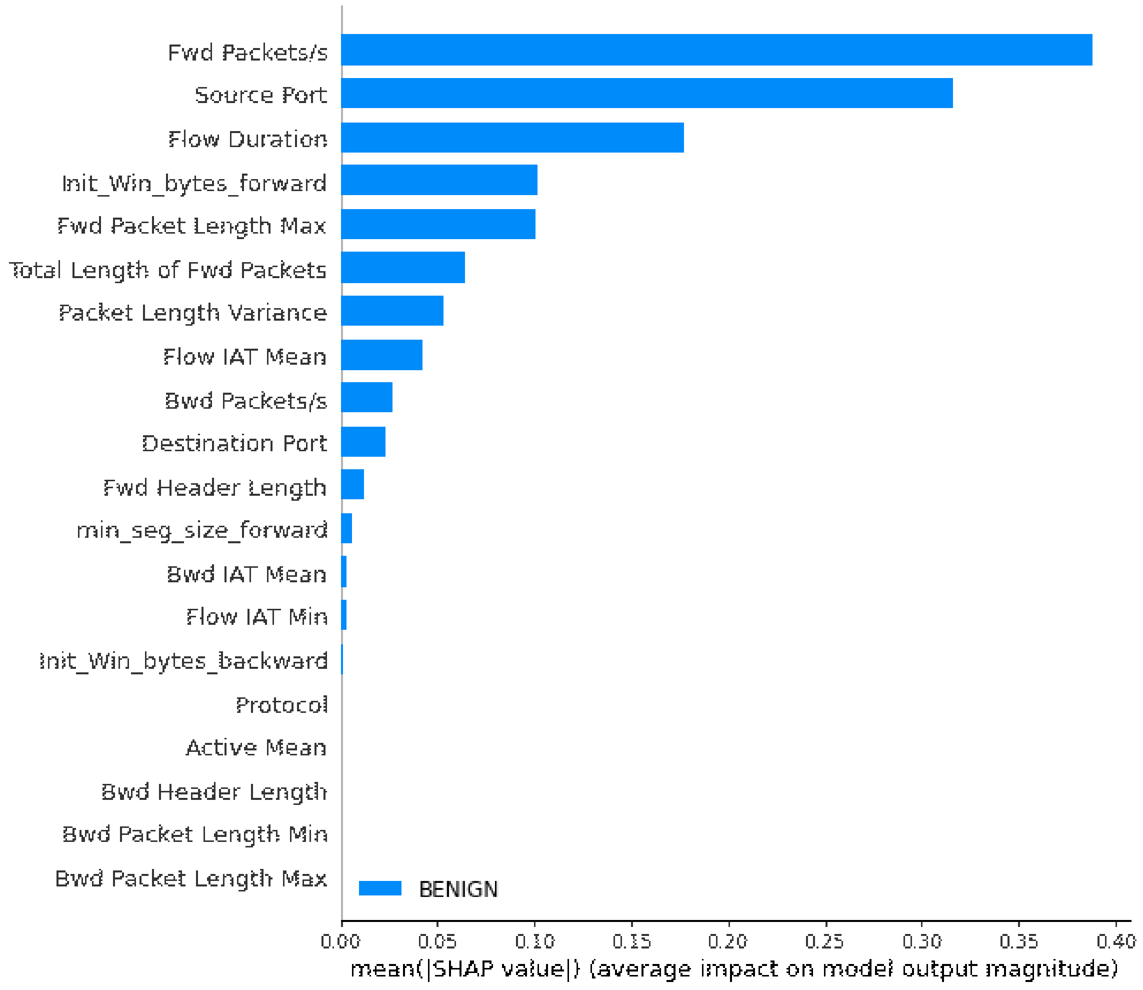

Figure 3.3: Feature attribution found by SHAP

In Figure 3.3, It can be seen that based on this algorithm, the three features Fwd Packets/s (i.e. number of packets received per second), Source Port, and Flow Duration, play the most important role in predicting if a packet is benign or malignant.

After finding out those distinguishing features, we can filter the dataset by different values of each feature to see what proportion of benign and malicious traffic is returned. Below we show graphs for two of these features showing the proportion of benign and malicious traffic returned after filtering the data based on different values for these two features.

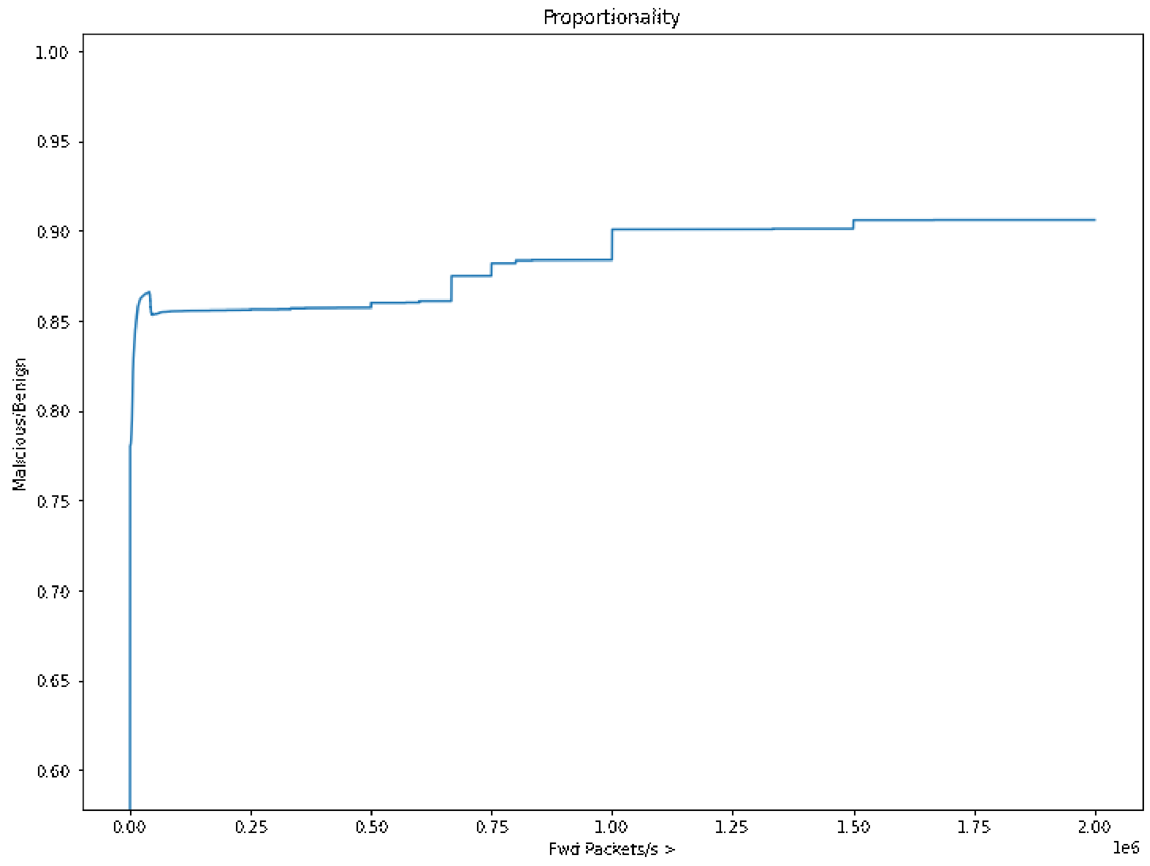

Figure 3.4: Filtering result based of different values of the feature “Fwd Packets/s”

Figure 3.4 shows that if we filter the data based on forwarding packets per second being more than a specific number, the proportionate of malicious traffic increases almost monotonically. The steps that are seen in the graph are for the wide range of forwarding packets per second (from 0 to 2000000). We had to increase the flow duration by 100 each time to make it feasible to cover all its possible values.

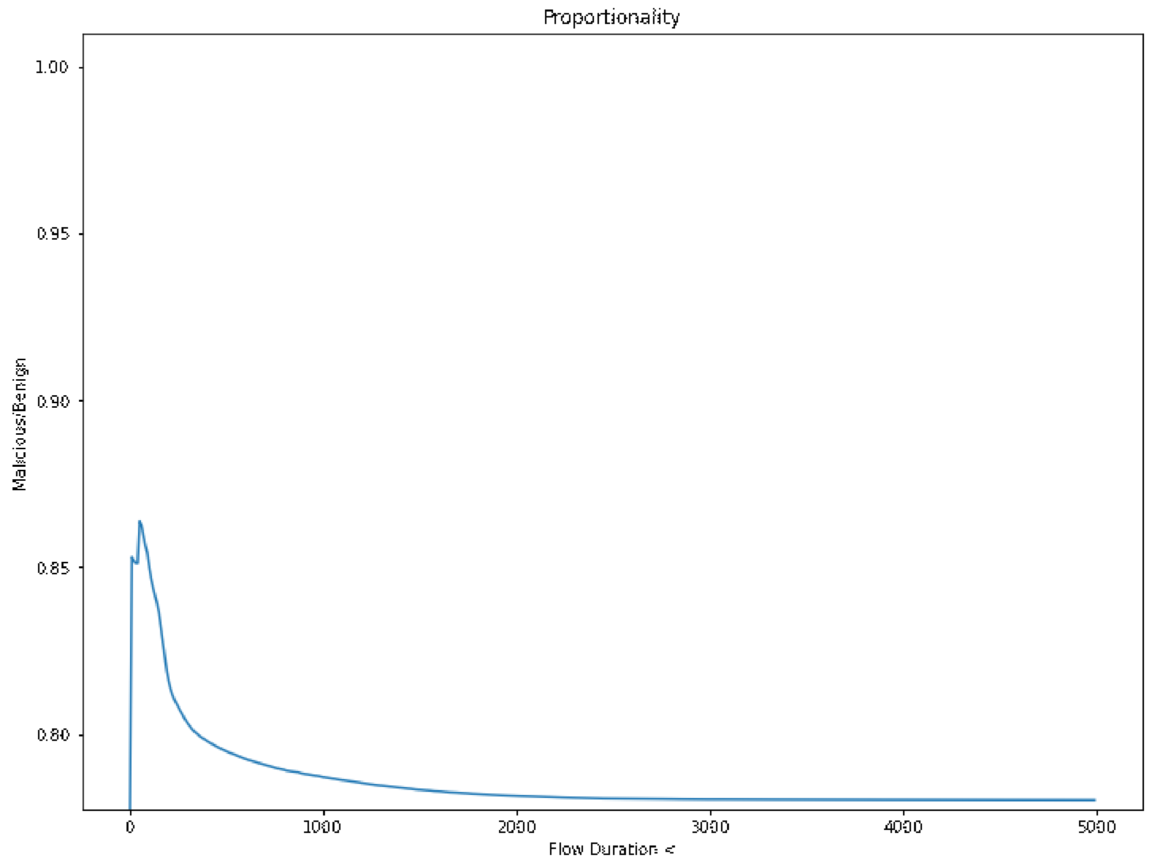

Figure 3.5: Filtering result based of different values of the feature “Flow Duration”

Figure 3.5 shows that if we filter data based on flow duration being less than a specific number, the proportion of returned data favours malicious traffic for that number’s small values. As we increase that number’s value, the proportion gets closer and closer to its initial value for unfiltered data.

Although SHAP recognized source port as the second most important feature in helping the model prediction, this feature cannot be used solely to filter the traffic. First, the variety of port numbers is not enough to create a smooth curve. Second, since the source port number is usually assigned randomly for the client, it cannot indicate malicious or benign traffic when used alone. Hence, a similar diagram for this feature does not give us useful insight.

LIME

As mentioned before, LIME is usually used to define the behaviour of a model for some specific prediction. This algorithm is useful for scenarios in which one wants to explain to someone affected by the model's decision about the rationale behind the decision (e.g. a financial advisor trying to explain to the client why he/she has been rejected an increase in the line of credit). In our context, we are interested in the general approach that the machine takes for classifying the network traffic rather than its methodology for a single instance. However, we have used LIME as well to see if it gives us an insight into the machine learning model.

Using LIME to perform feature attribution is itself a training process. We use the model that we have and a portion of the data used to train that model to generate a more interpretable approximation. After training that approximation, we get a weight for each feature and a general intercept (just like a linear regression model). After that, we can see how and to what degree each feature contributes to the model's prediction. Following is the graph showing the weights of some of those features:

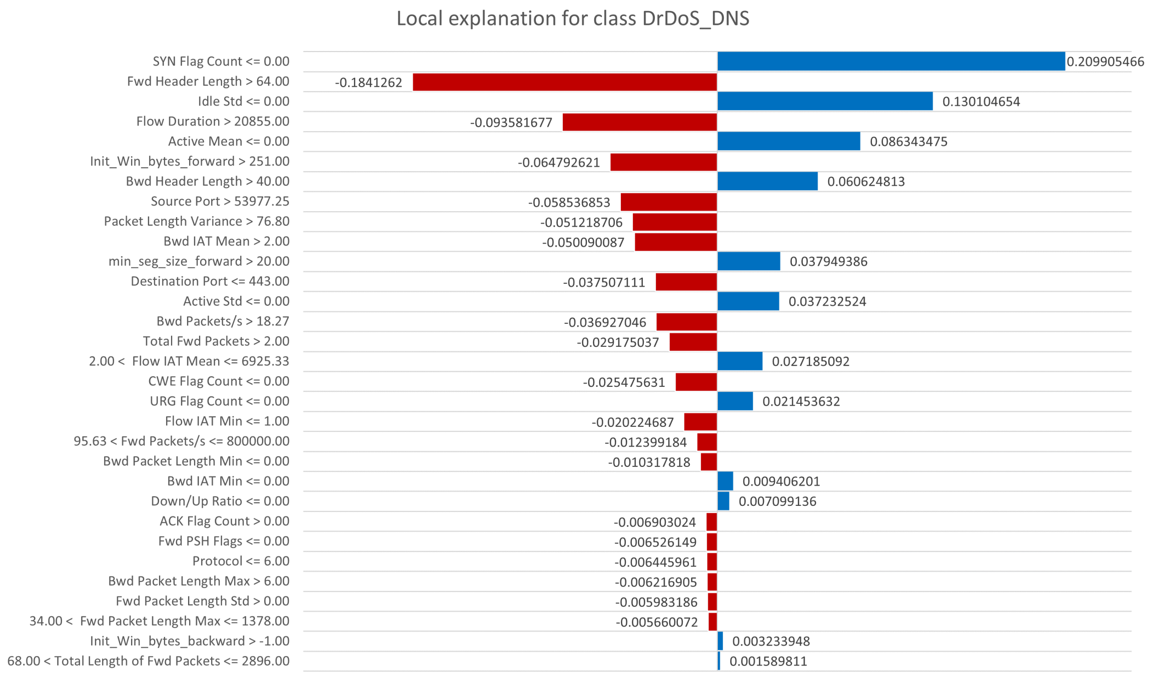

Figure 3.6: Feature attribution found by LIME

The blue bars in Figure 3.6 are attributes that work in favour of malicious traffic (i.e. helping us make sure they are malicious). The red ones are the features acting against that (i.e. an indication of malicious traffic). However, since this model is a binary classifier, each feature helps us understand class classification directly or by acting for the opposing class. The ranges that are shown for each feature on the left are for the sampling phase of LIME. Since LIME generates an approximation of the actual model, the above weights are valid in the specified ranges.

First, we will check the proportion in returned traffic if we filter data based on the "SYN Flag Count." This feature's only possible values are zero or one (a binary feature) in this dataset. If we set the value of that feature zero, the proportion in returned data is 0.7942 to 0.2058, favouring malicious traffic. If we set the value of that feature 1, we get 67.81% Benign traffic and 32.19% malicious traffic.

In the diagram shown below, we have tested the second most important feature found by LIME. Based on what is inferred from LIME's graph (Figure 3.6), forward header length should act against malicious traffic for values higher than 64 for that feature (i.e. traffic with forwarding heather length more than 64 tends to be benign). This graph can be used to check the validity of such a conclusion made by LIME.

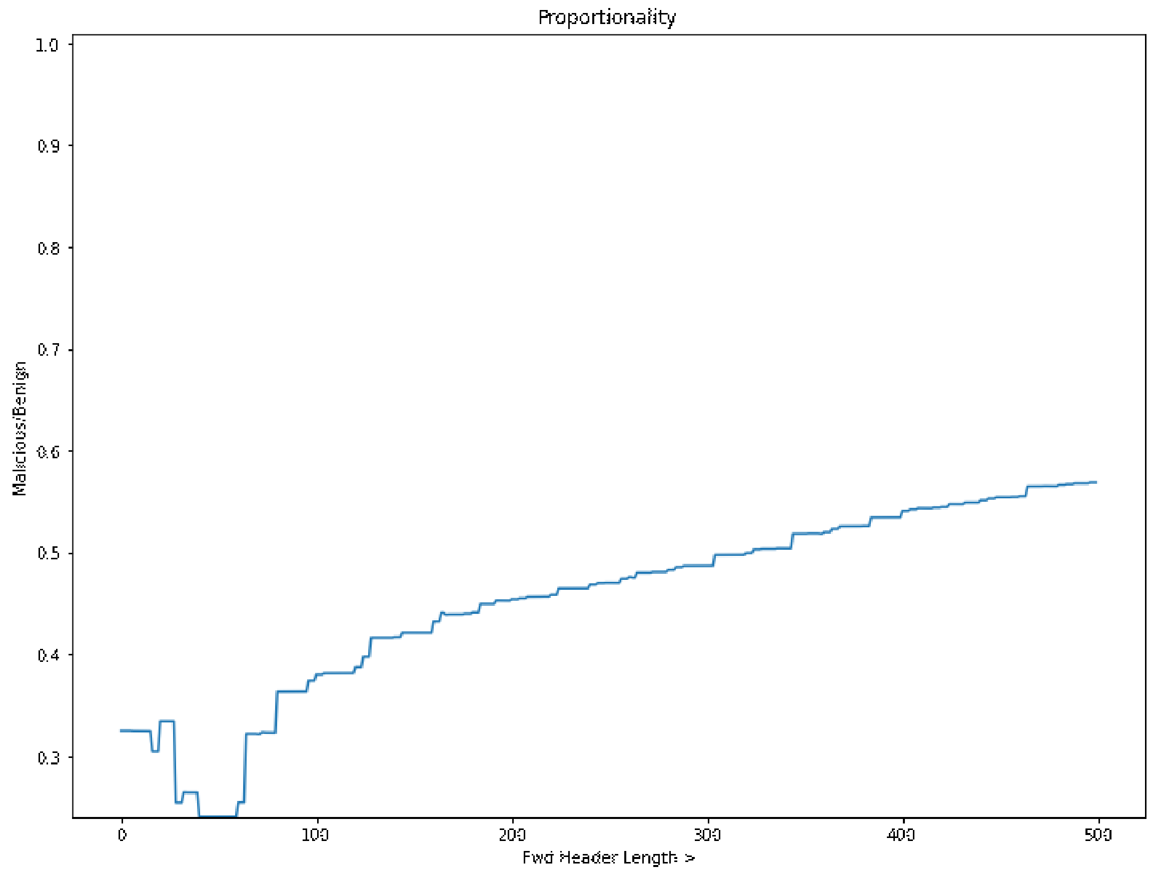

Figure 3.7: Filtering result based on the feature “Fwd Header Length” in large scale

The graph in Figure 3.7 was created for values between 0 and 500. This range does not cover all the possible values for this variable.

As you can see, although this conclusion may hold for values near 64, it is not the case far away from 64. Below we show the same graph with a much smaller scale to cover more values for this feature.

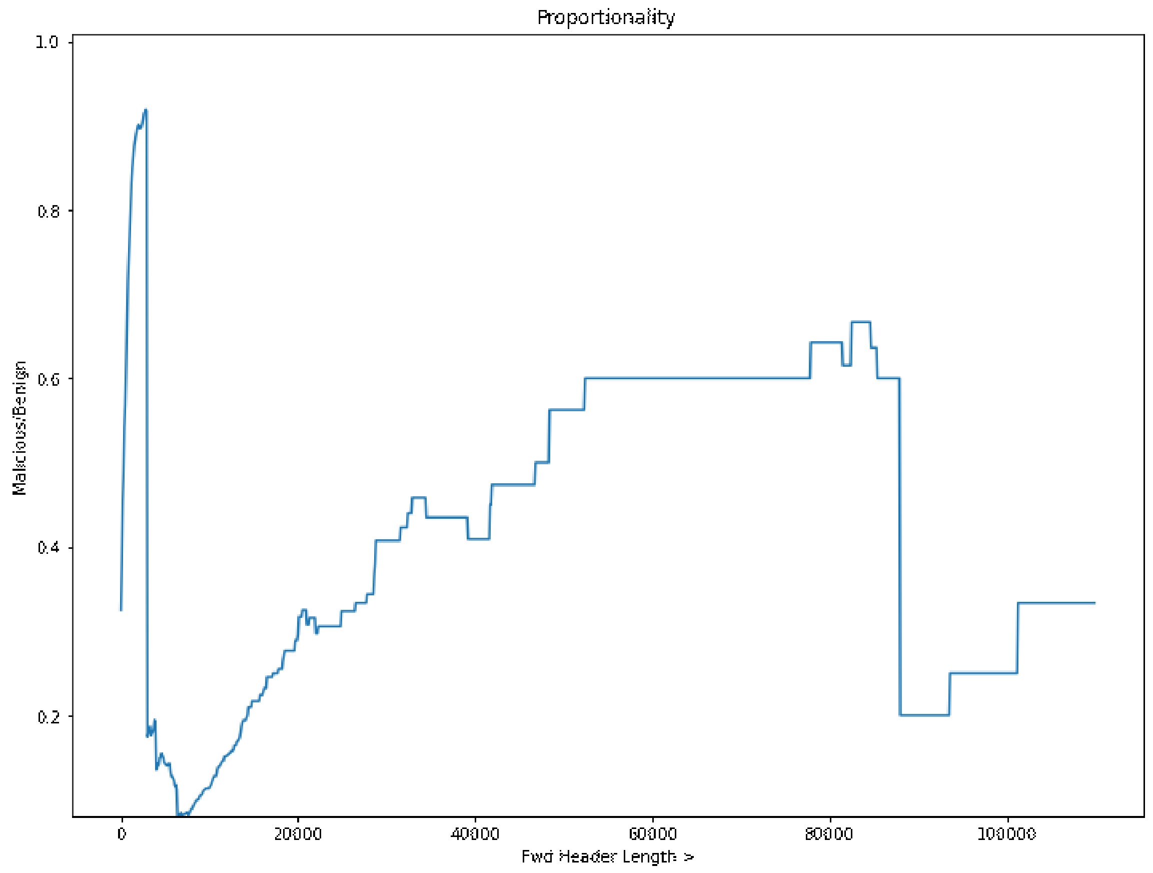

Figure 3.8: Filtering result based on the feature "Fwd Header Length" in small scale

To make the creation of the graph in Figure 3.8 feasible, we have incremented the value of the variable by 100 in each step.

In the Figure 3.8, we can see that the general trend suggested by LIME does not hold for all the values of this feature more than 64. However, for a specific value (2801 to be exact), we can get a high proportion of malicious traffic (up to 92%). This observation shows that even if the weights suggested by LIME may not be right for global interpretation, we can still use the feature found by that algorithm to understand the general importance of that feature for the model's prediction.

Another thing worth mentioning is that the graph created by LIME is more susceptible to change than the one created by SHAP. That is because LIME tries to approximate the model in the locality of an instance. This locality is prone to even small changes made to the data used for training. Based on the knowledge that we have from the importance of each feature from LIME and SHAP, we can use the features found by these algorithms to train a classifier that is inherently interpretable (something like logistic regression [40] or SVM [41]). We have tested this idea on all the combinations of the two most important features found by LIME and SHAP. The coefficients produced by the logistic regression model can then be used to produce a signature for blocking malicious traffic.

3.3 Deriving Snort-like Signatures

After using LIME and SHAP to find critical features in classification, we should see how well a combination of these features perform. For testing the combination of any two features found, we can draw a 3D graph. However, doing so does not help derive a signature. That is because the knowledge gained from 3D graphs is subject to perspective, and it is generally challenging to represent them properly in a 2D environment.

3.3.1 Deriving Snort-Like Signatures

For the reasons mentioned above, we have decided to use another machine learning model that is inherently interpretable to see how well those features perform in classification. One of such models is logistic regression. Using this model and the coefficients found for each feature in this model, we can directly find the signature that mimics this model's behaviour. We have decided to test this idea on the two features found by SHAP values (Fwd packets per second and flow duration). As mentioned before, the source port number is almost always chosen randomly b the client's machine. Therefore, we do not use it in training the logistic regression. After training the logistic regression, the model becomes a linear function of the form:

$$\ell = \ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \tag{3.1}$$

In the above formula, β_0 represents intercept of the line (i.e. bias), and β_1 and β_2 denote the coefficient for the features x_1 and x_2, respectively. p is also the probability of one instance being malicious. Hence, $1-p$ becomes the probability of the instance being benign.

For finding out what values of each feature should each instance have to be classified as a malicious traffic or benign, we must find when the value of this function is positive or negative. However, this is the case when we choose the threshold to be 0.5 (if a packet has a more than fifty percent chance of being malicious, then the model should classify it as malicious). For different threshold values, instead of checking for the sign of the function value, we must check for another positive or negative value that corresponds to that degree of the threshold.

3.3.2 Comparative Analysis

For training this function, we have used 75% of the data. Stratified sampling has also been used in this case. After testing the model on the remaining 25%, we check different criteria to see which combination of features returns the best logistic regression model.

First, we train the logistic regression model using SHAP's two features (i.e. Fwd packets per second and flow duration). After training this model and testing it on the test set, we get the following values for evaluation metrics shown below:

Classes	Precision	Recall	F1-Score	Accuracy
Benign	0.93	0.78	0.85	0.88
Malicious	0.84	0.95	0.89	
Total	0.89	0.87	0.87	

Table 3.1: Evaluation metrics for Logistic Regression model trained by features found by SHAP.

If we repeat the same scenario using the two features found by LIME (i.e. SYN flag count and forward Header Length), we get the following table:

Classes	Precision	Recall	F1-Score	Accuracy
Benign	0.68	0.97	0.80	0.77
Malicious	0.96	0.61	0.74	
Total	0.82	0.79	0.77	

Table 3.2: Evaluation metrics for Logistic Regression model trained by features found by LIME.

Although the recall for benign traffic is way more in this model than the previous one, we are interested in recall for malicious traffic, accuracy, and overall F1-score. Therefore, we have decided to go with the model trained on the features found by SHAP value.

After training, the formula for the first logistic regression model becomes as follows:

$$\ln\left(\frac{p}{1-p}\right) = (-2\text{e}{-9}) + (1.1\text{e}{-6})x_1 + (-6.51\text{e}{-5})x_2 \tag{3.2}$$

In the above equation, x_1 represents the value for forward packets per second and x_2 represents the value for flow duration. p still represents the probability of an instance being malicious.

After that, we can find a signature that can filter some portion of malicious traffic using the knowledge gained by interpreting the classifier. For finding the signature, we must find a combination of value for those two features that make the function positive. As said before, this is the case for threshold 0.5; for other values, we must look for other values of the function.

Based on the features' coefficients in the model's function, we have to look for traffic with low flow duration and high forward packets per second. We consider the traffic with a flow duration of less than 1000 and forward packets per second more than 1 million. After that, we try to see how well this signature is performing in detecting malicious traffic. For that, we test on the same dataset with which we trained our ANN classifier.

The derived signature returns 90% malicious traffic and 10% benign traffic. This retrieval includes 67% of all the malicious traffic in the dataset. When used on the dataset with MSQL DDoS attack, the same signature returns 98% malicious and 2% benign. At first glance it may seem that this high proportion may be the result of the dataset being imbalanced. However, this retrieval includes 78% of all the malicious traffic in the dataset. This observation attests to the efficacy of the derived signature.

Chapter 4

Secure Rule Matching

After deriving the signatures from the machine learning model, we have to evaluate them to see how well they are doing to block malicious traffic. Although testing the signature on the dataset that we already have can act as a preliminary evaluation, it would be great if we could perform this evaluation in a real-life scenario. The advantage of evaluation using real-time data over premade datasets is that these datasets often fail to convey an unbiased sample of the population of interest. Hence, the best way to test them is by deploying them in a system to see how well they perform. Furthermore, it is useful to not only rely on the network traffic from a single corporation. Diversifying our test space shows the true potentials of the signature and can help us make the evaluation process more rigorous.

Using network traffic from other corporations for the sake of evaluation can be done in two ways. First, we ask the corporation for their network traffic and test the signature using their traffic and our own resources. Doing so is the safest way since we do not share any information with the corporation whose network traffic is being used. Although safe, this solution is not entirely feasible. That is because we always have limited resources at our disposal, and it does not make sense to dedicate a high proportion of them to the evaluation of signatures. As a result, we again fail to diversify test space enough because the evaluation of signatures using network traffic from multiple corporations using our resources imposes considerable overhead on us.

An alternative way would be sending the signature to the designated organization so the evaluation can be done on their systems using their resources. This latter method is way more efficient than the former one. However, it has a significant drawback. The signatures that we derive from interpreting the machine learning model may sometimes contain sensitive data about the network infrastructure. As a result, these signatures cannot be passed around freely.

Another problem with this method is that even if we encrypt the signature itself and perform decryption in the end system right before the matching process, the signature's information can be reverse-engineered using the filtered traffic. To overcome these problems and get the best of both worlds, we have devised multiple methods for performing this matching process, each having its pros and cons.

4.1 Conventional Asymmetric Cryptosystem

In this method, we can use any asymmetric cryptosystem of our choice. First, we implement a rule matcher system with public and private key embedded in it. Then, we deploy that system in the corporation systems whose network traffic we are interested in. After that, we encrypt the signatures file using the public key that we have. Since the code for the system deployed in the corporation's unsafe environment is compiled, they have no way of accessing the private key (we call this compiled system secure rule matcher). Therefore, the decrypted signature file cannot be seen by an unauthorized user. After sending the encrypted signature file, it is decrypted using the private key in the rule matcher. The decrypted signature file is then used to filter traffic. The matched traffic that is returned by the signature needs to be encrypted as well. If the filtered traffic is sent back unencrypted, as mentioned before, it can divulge some confidential information. As a result, the filtered traffic is also encrypted using the private key in the rule matcher. The encrypted traffic is therefore sent back. If we are interested in evaluating multiple signatures simultaneously, we can assign a unique ID to each signature and the traffic it has matched. After receiving encrypted network traffic filtered using the signatures, we can decrypt them to see which one is benign or malicious. The overall procedure for this scheme is shown in Figure 4.1.

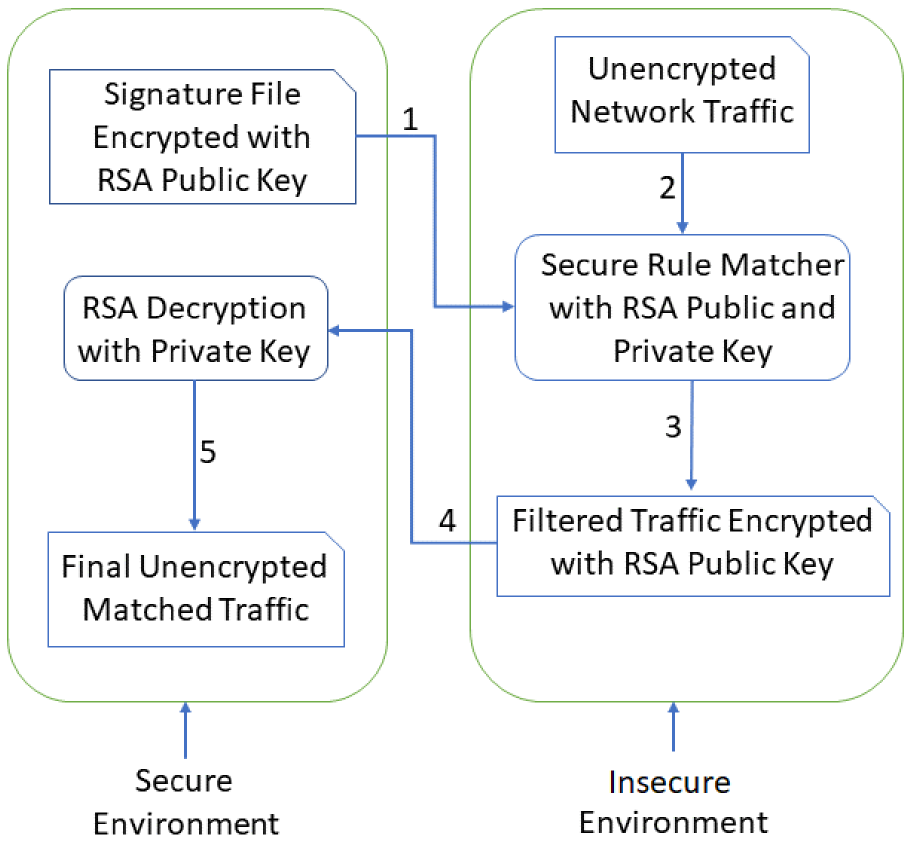
Signature File Encrypted with RSA Public Key
RSA Decryption with Private Key
Final Unencrypted Matched Traffic
Unencrypted Network Traffic
Secure Rule Matcher with RSA Public and Private Key
Filtered Traffic Encrypted with RSA Public Key
1
2
3
4
5
Secure Environment
Insecure Environment

Figure 4.1: RSA scheme for signature encryption

In this scheme, we have used RSA as our asymmetric cryptosystem. The overall scheme is shown in Figure 4.1. Choosing any other asymmetric cryptosystems does not contradict this methodology. Here, the secure environment refers to the corporation's infrastructure, whose signatures are pending evaluation. The insecure environment refers to the infrastructure of the corporation whose traffic is being used for evaluation.

As it is shown in Figure 4.1, the private key has to be embedded in the secure rule matcher in this scheme. Although doing so saves us much time in implementation, it has its own drawbacks security-wise. Embedding the private key inside a secure rule matcher may not be acceptable by corporations that demand a high level of security.

Apart from security, this scheme has some problems in scenarios when there is a large signature file, and the traffic that we want to filter with those signatures is relatively small. The problem is that RSA is only able to encrypt files as big as its key size. Therefore, for a key size of 2048 bit, the maximum file that it can encrypt is 256 bytes. For encrypting bigger files, usually, a symmetric cryptosystem is used along with that. For doing that, first, we encrypt the big file using the symmetric cryptosystem key (e.g. AES key). Then, we encrypt the symmetric cryptosystem key with RSA public key. In this way, the decryption process is done by first decrypting the symmetric cryptosystem key (using RSA private key) and then decrypting the actual file using that.

Doing the process mentioned above can be time-consuming for the corporation whose traffic we want to use if we have a large signature file. An alternative to that is converting signature files so that filtering can be done without decryption of the signature file. This conversion method is explained in the next section.

4.2 Order-preserving Cryptosystem

For implementing this scheme, we need an arbitrary hashing algorithm for converting the name of attributes in the signature file and the order-preserving cryptosystem for encrypting the values of those attributes. Since order-preserving cryptosystem works only on numeric data, we need an encoding method that maps categorical attributes in the signature files to its corresponding numeric value. As mentioned before, the order-preserving encryption that we use consists of a sorted list of numbers that act as the key for this cryptosystem. The cyphertext assigned to each number is defined by the relative magnitude of the number being encrypted. In this way, the order of numbers is maintained even after encryption.

The encoding method should also consider the comprehensiveness of the categorical attributes it is trying to encode. For example, since application-layer protocols like HTTP and DNS use TCP protocol to operate, filtering network traffic based on TCP should also return all the traffic using these application-layer protocols. Therefore, the encoding scheme must choose a number for TCP that is higher (or lower) than all the other numbers assigned for application-layer protocols that use it. The same scenario applies to UDP (or other keywords used in signatures like IP and . . .). To avoid overlap for these comprehensive keywords (UDP, TCP, IP, . . .), the encoding scheme should choose different ranges for each of them. After that, we can encrypt the numbers assigned to each of them using the order-preserving cryptosystem. After conversion, the signature file gets the following form:

$<$ Hashed value of the attribute's name $>$: $<$ The encrypted value $>$

The choice of the hashing algorithm is arbitrary, but this choice has to be projected in the secure rule matcher.

After that, the file used for encoding signatures (encoding scheme map) is encrypted using an RSA private key. The corresponding public key for this cryptosystem should also be embedded in the secure rule matcher later for decryption. These files are then sent over to the secure rule matcher.

The following figure depicts the overall procedure for this scheme:

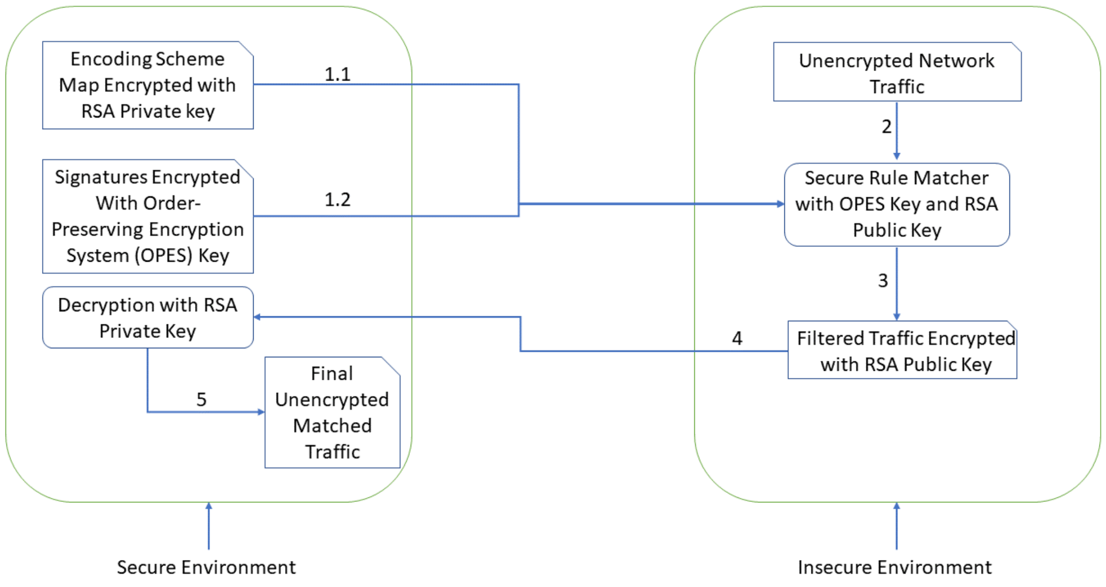
Encoding Scheme Map Encrypted with RSA Private key
1.1
Signatures Encrypted With Order-Preserving Encryption System (OPES) Key
1.2
Decryption with RSA Private Key
5
Final Unencrypted Matched Traffic
Unencrypted Network Traffic
2
Secure Rule Matcher with OPES Key and RSA Public Key
3
4
Filtered Traffic Encrypted with RSA Public Key
Secure Environment
Insecure Environment

Figure 4.2: Order-preserving scheme for signature encryption

Since order-preserving encryption is symmetric, in this scheme, the key should also be embedded in the secure rule matcher.

After sending the encoded and encrypted signature files, we have to the same thing on the network traffic. First, we encode the network traffic's categorical values; then, we encrypt each attribute value using OPES. In this way, filtering network traffic based on the signature boils down to a simple comparison of numeric values.

Filtered traffic is also encrypted using the embedded RSA public key to prevent unauthorized persons from reverse-engineering the signatures. The result is then sent back to the secure environment for decryption.

It is evident that the security in this scheme is not any better than the previous one. However, when the ratio of signature file size to the network traffic size is large, this scheme is expected to impose lower overhead to the infrastructure in the insecure environment. Generally speaking, if the time it takes to decrypt the signature files is more than the time it takes to encrypt network traffic using OPES, then this scheme is preferable over the first one. By using this scheme, we have reduced the amount of overhead that the insecure environment faces that the cost of increasing it in the secure environment.

In some cases, one corporation might choose to sacrifice efficiency in favour of security. In these circumstances, embedding the secure rule matcher with the private key (in the case of OPES, private and public keys are the same) is unacceptable. Therefore, we have to come up with another scheme in whose agenda security has the highest priority.

4.3 Homomorphic Cryptosystem

In this scheme, we again need an encoding method just like the one we used for the scheme mentioned above. However, in this scheme, the cryptosystem used to encrypt each attribute's value should be homomorphic (e.g. DGK, Paillier, . . .).

Just like the previous scheme, this one also starts by encoding categorical values in the signature file. After that, the signature file is converted in the same format mentioned above with a subtle difference. This time, instead of encrypting the numeric values using OPES, we use a homomorphic cryptosystem. We have chosen DGK as our homomorphic cryptosystem in this scheme.

After converting the signatures, the encoding scheme map is also encrypted using the DGK private key. The converted signature file is then sent over to the secure rule matcher along with the encrypted encoding file.

After that, the unencrypted network traffic is fed to the secure rule matcher. In the same way, encoding and encryption are performed on the unencrypted network traffic. Again the filtering act becomes as simple as comparing two numeric values. However, in this scheme, the order of numbers is not evident while they are encrypted. To perform this comparison, we decided to use the additive homomorphic property of this cryptosystem.

To make this comparison, we should first calculate the encrypted value for the negative of a number. That is because subtracting two numbers is the same as adding the first one with the negative of the second one. We cannot multiply the plaintext value by -1 and pass it to the formula 2.6 to get an encrypted value for that number's negative value. That is because in this cryptosystem (i.e. DGK), the plaintext space is $\mathbb{Z}_u$, and it does not include any negative number. However, there is a way around this limitation. Suppose that the plaintext value of the number whose negative value we want to encrypt is m. We are interested in finding the encrypted value of $-m$. Let's denote the encrypted value of m by c. We then calculate the encrypted value for the product of m and $u-1$ (u is given in public key and is accessible). Formula 2.9 can be used to calculate this encrypted value. In that formula, c is equal to $E_{pk}(m,r)$, and m' is equal to $u-1$. After putting these values inside formula 2.9, the right side of equality becomes $E_{pk}((mu-m) \bmod u, ru-r)$. We know that $(mu-m) \bmod u$ is equal to $-m$. Therefore, the right side of equality in formula 2.9 becomes $E_{pk}(-m, ru-r)$. This value is the encrypted value of $-m$ (i.e. the value we were looking for).

After that, we should first calculate the encrypted result of the two numbers' subtraction using the formula 2.8. We perform this operation every time we want to compare two numbers. The encrypted result of these subtractions is then stored in a file. The file is then sent back to the secure environment for decryption. However, when the result of the subtraction is positive, the decrypted value is still positive. This problem arises from the fact that this cryptosystem uses modular arithmetic. In modular arithmetic, the result of subtraction is never negative. For instance, $(2-3) \bmod 4$ is 3 (instead of -1).

To distinguish negative numbers from positive numbers after decryption, we have to know the subtraction result's possible range. Based on our application, we already know that. For instance, suppose that we want to compare two port numbers. Since a port number is an integer between 1 and 65535, we know that the two port numbers' subtraction result is between -65534 and 65534. By choosing our message space big enough (which, in this case, is insured by choosing the L parameter of this cryptosystem a number more than 16), we can solve the problem of distinguishing between positive and negative numbers. Following our example, in this manner, if we see a number more than 65534 after decryption, we can be sure that the result of subtraction was indeed a negative number. This line of reasoning can easily be extrapolated to cover the comparison for other attributes.

After finding out the sign for the subtractions' results, the file is sent back to the secure rule matcher. Secure rule matcher uses this file to perform filtering. After getting the filtered traffic, we encrypt it using the DGK public key. The result is then sent back to secure environment. It can then be decrypted using DGK private key. The overall process is shown below:

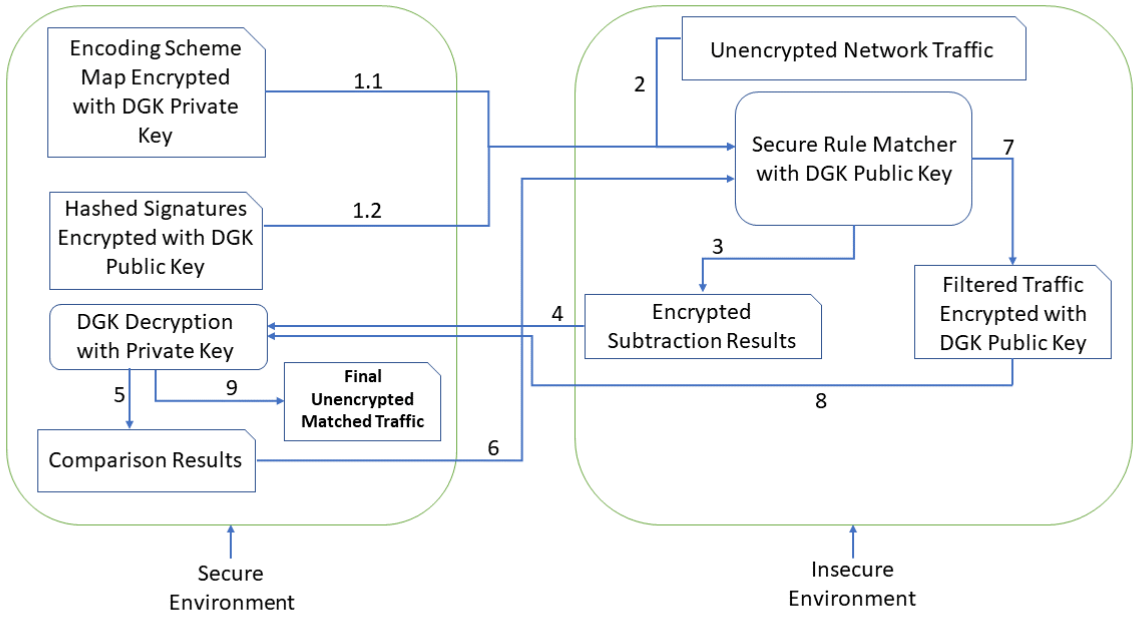
Encoding Scheme Map Encrypted with DGK Private Key
1.1
Hashed Signatures Encrypted with DGK Public Key
1.2
DGK Decryption with Private Key
5
9
Final Unencrypted Matched Traffic
Comparison Results
6
Unencrypted Network Traffic
2
Secure Rule Matcher with DGK Public Key
7
3
4
Encrypted Subtraction Results
Filtered Traffic Encrypted with DGK Public Key
8
Secure Environment
Insecure Environment

Figure 4.3: DGK scheme for signature encryption

Although the encoding scheme map is encrypted using DGK private key, it cannot be considered a vulnerability because it does not include any confidential information. Therefore, in the case of DGK public key exposure, the system's security is not compromised.

The main pitfall of this scheme is its efficiency. This scheme's overhead imposed on secure environment is almost equal to when we get the traffic from insecure environment and perform filtering using secure environment resources. We test this hypothesis by doing some experiments.

4.4 Comparison and Analysis

In this section, we try to compare the runtime for each of the schemes that were mentioned. For doing so, we have used a pcab file downloaded from the Wireshark website. This file includes one million network packets. We have filtered this network traffic using three different signatures in each of the schemes mentioned earlier. The runtime for filtering each signature in each scheme is given in the following table:

No. of Matched Packets	Elapsed Time	Signature	Scheme
2195	360451	Signature one	DGK
	379		OPES
	348		RSA
910883	358664	Signature two	DGK
	423		OPES
	295		RSA
849962	368130	Signature three	DGK
	414		OPES
	305		RSA

Table 4.1: Matching result for different schemes

In Table 4.1, the total number of packets in the pcab file was 1 million. The unit for elapsed time is millisecond.

Table 4.1 shows us interesting results from the efficiency of each of the schemes. Although at first glance, it may seem that the first scheme (RSA) is more efficient than the second one (OPES), it is not always true. As mentioned before, RSA can only be used to encrypt files as big as its key length. Hence, for encrypting large files, we have to either use another symmetric key along with it or break the file into smaller parts. Breaking up the file into smaller parts and encrypting each part with RSA is not an efficient solution. Therefore, what usually happens in this scenario is that we first use a symmetric cryptosystem (usually AES) to encrypt the large file. Then, we encrypt the symmetric cryptosystem key using RSA public key. When we experimented on our system, RSA decryption usually took 5ms to be done (for a file as big as 25 bytes). However, when we use AES and RSA for larger files, this runtime increases. For instance, decryption of a 2-MB file using this scheme takes us 32ms. Since decryption and filtering happen in the insecure environment, if we add this 32ms to the time it takes for the first scheme to filter the traffic, we see that the first scheme is less efficient compared to the second one when the signature file is big enough (even for 1 million network packets). Therefore, by using the second scheme, we reduce the overhead in insecure environments and increase it in the secure environment. That is because converting signatures is more time consuming that mere encryption by RSA (or RSA and AES for relatively big files).

It can also be seen that DGK, although safer, is not efficient at all. On average, on our system, DGK takes 56 microseconds for the decryption of a single number. If we assume that we need only one integer comparison for each network packet to filter it, we have to perform DGK decryption for every network packet at least once. If we multiply 1 million by 56 microseconds, we get 56 seconds. This runtime is almost 100 times more than what we have for the first and second schemes. Therefore, when the number of packets is relatively high (1 million in this case), asking for the network traffic and performing filtering on our end would more efficient than the third scheme. Furthermore, even when the number of packets is relatively small, usage of the third scheme is not justified because, in that case doing the filtering does not impose significant overhead. Therefore, the act of filtering can again be done in the secure environment.

We have also tested Paillier as our cryptosystem for the third scheme. For this application, it is even more time consuming when compared to DGK. The reason is that Paillier—unlike DGK—only takes one parameter (e.g. key length). To preserve security, the key length for this cryptosystem is often chosen to be 1024 bits. Although choosing this big of a key is necessary for maintaining security, it gives us a cyphertext space that is bigger than what we need. This big cyphertext space causes an increase in decryption and encryption runtime. When we tested Paillier's implementation of the third scheme on filtering nine network packets using a specific signature, it took 250 ms to return three packets as the answer. When implemented by DGK and tested on the same network packets with the same signature, the same scheme took only 4 ms to return the same answer.

One possible application of the schemes we proposed is in corporations with a high level of authority. These corporations that are very fastidious about their security and confidentiality can use the proposed methods to increase efficiency of their network filtering phase in their firewall by using signatures instead of ML classifiers.

For evaluating these signatures before deploying them, these corporations must have authority over the corporation whose traffic will be used for evaluation. In that case, they can enforce the deployment of the secure rule matcher on the insecure environment's infrastructure.

Chapter 5

Conclusion and Future Work

In this section, we briefly go through everything that we covered in this thesis and mention the contributions that this work can make to the field of cybersecurity. Furthermore, we explain what can be done to enhance further and optimize this work.

5.1 Conclusion

In this thesis, we tried different methods of machine learning interpretability to derive Snort-like signatures from classifiers that are trained to distinguish between malicious and benign data. As mentioned before, apart from the efficiency of these signatures compared to machine learning classifiers, by making the machine learning model interpretable, we can justify our claims better and defend our decision-making process in a corporation.

The methods we have chosen for ML interpretability are LIME and SHAP values. Since SHAP takes into consideration the overall shift in the model's expected output concerning each combination of variables—as opposed to LIME that tries to find a local approximation to the model—it is considered a more robust approach for interpretability. However, this does not mean that LIME is never worth testing for interpretability.

Although the interpretability methods discussed help understand how the model is classifying the instances, they do not remove expert insight. In using each of them, an expert insight helps figure out whether the features found by interpretability techniques play an important role in helping the model perform the classification. Each of the features found by these techniques can also be used individually for filtering the dataset to see how well they perform in classifying the data.

After deriving the signature from the model, we have to have a way to test its effectiveness. Although the derived signature can be evaluated using the available dataset that we already have, evaluating them using real-life network traffic is preferable because that data is the most unbiased data we can get our hands on.

For evaluating the signatures using real-life network traffic, we have two choices. First, ask other corporations for their traffic and testing the signatures using our own resources; Second, send the signatures over to the corporation whose network traffic we want to use for evaluation. However, each of these methods has its own problem. In the first one, a considerable overhead is imposed on our infrastructure; In the second one, the system's security might be compromised by the information that can be concluded from the shared signatures.

To reduce the overhead on our system and maintain our corporation's security, we devised three different approaches for performing the filtering in other corporations without divulging any information from the signatures (i.e. RSA, OPES, and DGK).

The filtering time for RSA is slightly lower than the filtering time for OPES. However, When the signature file is big, the RSA decryption is more time-consuming than when it is small. Hence, when the size of the signature file is relatively large, OPES might be preferable.

Working with DGK and Paillier is so resource-intensive for both parties involved that its usage is unjustifiable in almost all the cases.

5.2 Future Work

Right now, the process of making the ML classifier interpretable is not automatized. The features are extracted and chosen based on the knowledge of the expert. Then, the chosen features are used for trailing an inherently interpretable model. It would be great to find a way to make the system able to draw conclusions about the model as well as the expert.

Although our method for deriving the signature from the classifier is helpful, the signature that it provides is hardly ever optimal. One can work on possible methods for creating a list of signatures that can completely mimic the classifier's functionality.

For the evaluation phase, we do not have a completely secure and efficient method. The OPES and RSA schemes are efficient but not secure. DGK scheme, on the other hand, is secure but not efficient at all. Finding a methodology that has the best of both worlds is ideal.

Bibliography

[1] O. Avatefipour, A. S. Al-Sumaiti, A. M. El-Sherbeeny, E. M. Awwad, M. A. Elmeligy, M. A. Mohamed, and H. Malik, “An Intelligent Secured Framework for Cyberattack Detection in Electric Vehicles’ CAN Bus Using Machine Learning,” *IEEE Access*, vol. 7, pp. 127580–127592, 2019.

[2] L. Hui and C. Yonghui, “Research Intrusion Detection Techniques from the Perspective of Machine Learning,” in *2010 Second International Conference on Multimedia and Information Technology*, (Kaifeng, China), pp. 166–168, IEEE, 2010.

[3] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, “Machine Learning and Deep Learning Methods for Cybersecurity,” *IEEE Access*, vol. 6, pp. 35365–35381, 2018.

[4] C. Lee, Y. Su, Y. Lin, and S. Lee, “Machine learning based network intrusion detection,” in *2017 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA)*, pp. 79–83, 2017.

[5] Z. L. Li, C.-J. M. Liang, W. Bai, Q. Zheng, Y. Xiong, and G. Sun, “Accelerating rule-matching systems with learned rankers,” in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, (Renton, WA), pp. 1041–1048, USENIX Association, July 2019.

[6] A. Tabasum, Z. Safi, W. AlKhater, and A. Shikfa, “Cybersecurity Issues in Implanted Medical Devices,” in *2018 International Conference on Computer and Applications (ICCA)*, (Beirut), pp. 1–9, IEEE, Aug. 2018.

[7] M. G. Kaosar, R. Paulet, and X. Yi, “Fully homomorphic encryption based two-party association rule mining,” *Data & Knowledge Engineering*, vol. 76-78, pp. 1–15, June 2012.

[8] D. Lin and K. Sako, eds., *Public-Key Cryptography – PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part I*, vol. 11442 of *Lecture Notes in Computer Science*. Cham: Springer International Publishing, 2019.

[9] G. Traverso, D. Demirel, and J. Buchmann, *Homomorphic Signature Schemes*. SpringerBriefs in Computer Science, Cham: Springer International Publishing, 2016.

[10] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, “Order preserving encryption for numeric data,” in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data — SIGMOD ’04*, (Paris, France), p. 563, ACM Press, 2004.

[11] Xin Zhou and Xiaofei Tang, “Research and implementation of RSA algorithm for encryption and decryption,” in *Proceedings of 2011 6th International Forum on Strategic Technology*, (Harbin, Heilongjiang, China), pp. 1118–1121, IEEE, Aug. 2011.

[12] I. Damgård, M. Geisler, and M. Krøigaard, “Efficient and Secure Comparison for On-Line Auctions,” in *Information Security and Privacy* (J. Pieprzyk, H. Ghodosi, and E. Dawson, eds.), vol. 4586, pp. 416–430, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Series Title: Lecture Notes in Computer Science.

[13] P. Paillier, “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,” in *Advances in Cryptology — EUROCRYPT ’99* (J. Stern, ed.), vol. 1592, pp. 223–238, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. Series Title: Lecture Notes in Computer Science.

[14] M. T. Ribeiro, S. Singh, and C. Guestrin, “”Why Should I Trust You?”: Explaining the Predictions of Any Classifier,” *arXiv:1602.04938 [cs, stat]*, Aug. 2016. arXiv: 1602.04938.

[15] E. Commision, “2018 reform of EU data protection rules.”

[16] F. Doshi-Velez and B. Kim, “Towards A Rigorous Science of Interpretable Machine Learning,” *arXiv:1702.08608 [cs, stat]*, Mar. 2017. arXiv: 1702.08608.

[17] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, “Explaining Explanations: An Overview of Interpretability of Machine Learning,” *arXiv:1806.00069 [cs, stat]*, Feb. 2019. arXiv: 1806.00069.

[18] M. W. Craven and J. W. Shavlik, “Extracting tree-structured representations of trained networks,” in *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS’95, (Cambridge, MA, USA), p. 24–30, MIT Press, 1995.

[19] M. Mehta, J. Rissanen, and R. Agrawal, “Mdl-based decision tree pruning,” in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, KDD’95, p. 216–221, AAAI Press, 1995.

[20] J. R. Quinlan, “Improved use of continuous attributes in C4.5,” *J. Artif. Int. Res.*, vol. 4, p. 77–90, Mar. 1996.

[21] S. Sivagama Sundhari, “A knowledge discovery using decision tree by Gini coefficient,” in *2011 International Conference on Business, Engineering and Industrial Applications*, (Kuala Lumpur, Malaysia), pp. 232–235, IEEE, June 2011.

[22] N. Frosst and G. Hinton, “Distilling a Neural Network Into a Soft Decision Tree,” *arXiv:1711.09784 [cs, stat]*, Nov. 2017. arXiv: 1711.09784.

[23] S. Jia, P. Lin, Z. Li, J. Zhang, and S. Liu, "Visualizing surrogate decision trees of convolutional neural networks," *Journal of Visualization*, vol. 23, pp. 141–156, Feb. 2020.

[24] S. Lipovetsky and M. Conklin, "Analysis of regression in game theory approach," *Applied Stochastic Models in Business and Industry*, vol. 17, pp. 319 – 330, 10 2001.

[25] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, "Interpretable machine learning: definitions, methods, and applications," *Proceedings of the National Academy of Sciences*, vol. 116, pp. 22071–22080, Oct. 2019. arXiv: 1901.04592.

[26] 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.), pp. 4765–4774, Curran Associates, Inc., 2017.

[27] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje, "Not just a black box: Learning important features through propagating activation differences," 2017.

[28] L. Morris, "Analysis of partially and fully homomorphic encryption," *Rochester Institute of Technology*, pp. 1–5, 2013.

[29] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, (New York, NY, USA), p. 169–178, Association for Computing Machinery, 2009.

[30] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2017.

[31] I. Damgard, M. Geisler, and M. Kroigard, "A correction to 'efficient and secure comparison for on-line auctions'," *International Journal of Applied Cryptography*, vol. 1, no. 4, p. 323, 2009.

[32] A. B. Alexandru, K. Gatsis, Y. Shoukry, S. A. Seshia, P. Tabuada, and G. J. Pappas, "Cloud-based quadratic optimization with partially homomorphic encryption," 2019.

[33] M. Nateghizad, Z. Erkin, and R. L. Lagendijk, "An efficient privacy-preserving comparison protocol in smart metering systems," *EURASIP J. Inf. Secur.*, vol. 2016, Dec. 2016.

[34] V. Shoup, *A Computational Introduction to Number Theory and Algebra.* USA: Cambridge University Press, 2005.

[35] H. Riesel, ed., *Prime Numbers and Computer Methods for Factorization.* USA: Birkhauser Boston Inc., 1985.

[36] G. Bebek, “Anti-tamper database research: Inference control techniques,” 2002.

[37] G. Ozsoyoglu, D. A. Singer, and S. S. Chung, “Anti-Tamper Databases,” in *Data and Applications Security XVII* (S. De Capitani di Vimercati, I. Ray, and I. Ray, eds.), vol. 142, pp. 133–146, Boston, MA: Springer US, 2004. Series Title: IFIP International Federation for Information Processing.

[38] J. Feigenbaum and M. Merritt, eds., *Distributed Computing and Cryptography*, vol. 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* Providence, Rhode Island: American Mathematical Society, Apr. 1991.

[39] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, “Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy,” in *2019 International Carnahan Conference on Security Technology (ICCST)*, pp. 1–8, 2019.

[40] D. W. Hosmer and S. Lemeshow, *Applied logistic regression.* John Wiley and Sons, 2000.

[41] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 3rd ed., 2011.