DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Secure Anomaly Detection on Serverless Edge Computing

Astghik Hakobyan

DEPARTMENT OF INFORMATICS

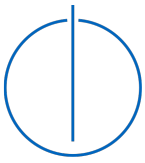
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Secure Anomaly Detection on Serverless Edge Computing

Sichere Erkennung von Anomalien im Serverless Edge Computing

Author:	Astghik Hakobyan
Supervisor:	Prof. Dr. Michael Gerndt
Advisor:	Anshul Jindal
Submission Date:	15.12.2021



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, Astghik Hakobyan

Abstract

Outlier or anomaly detection is the technique of finding patterns in a dataset that are inconsistent or considerably different from the rest of the data. The detection of anomalies often provides critical information in various application domains. Although there exist multiple techniques to solve this task, choosing the appropriate model proves to be tricky. Several aspects need to be considered to make the right decision. Whether the data contains sensitive information, is distributed, multi-dimensional, etc. plays an important role in the model choice.

The problem we address throughout this thesis is the secure anomaly detection on distributed data. We will deal with data containing sensitive information which can not be shared, thus we will need a technique to preserve privacy in the anomaly detection process. This data will not have a single source of information, but rather will be spread among multiple nodes with attached edge devices.

First, we present existing approaches for anomaly detection and privacy preservation tasks. Then, we introduce the necessary background for our model, focusing on isolation forests and cryptographic techniques. Afterwards, we present our model called **SECURE-SERVERLESS: SECURE anomaly detection on SERVERLESS edge computing** and discuss the improvements which were made to the original algorithm. We analyze the model's behavior on various datasets experimenting with different parameter values and propose improvements to overcome several challenging scenarios.

We show that we can get optimal results with the proposed model on several datasets. Nevertheless, in some cases we need to adjust the model to function better. We show that it might be beneficial to use synthetic data (e.g. when we have numerous nodes which have access only to a small chunk of data) or multiple split attributes (e.g. when data can not be split optimally with only horizontal and vertical lines). We also show a connection between parameter values and suggest being careful when choosing them as it is crucial for optimal performance. And last but not least, we propose an architecture which heavily uses the notions of serverless and edge computing.

Contents

1 Introduction

Outlier or anomaly detection is the technique of finding patterns in a dataset that are inconsistent or considerably different from the rest of the data. The detection of anomalies often provides critical information in various application domains [1]. For example, anomalies in credit card transactions could signify fraudulent use of credit cards. An anomalous spot in an astronomy image could indicate the discovery of a new star. An unusual computer network traffic pattern could stand for an unauthorized access. These applications demand anomaly detection algorithms with high detection and fast execution.

There exist several techniques to solve the anomaly detection task, but it is vital to choose an appropriate model which best suits the specific case. One important aspect to consider is whether the data we deal with contains sensitive information and privacy preservation is required. We need to understand what security preserving technique can be utilized that suits our needs and how it will affect the model. Another essential matter is the data distribution. Whether there is a single source of information or multiple nodes with small chunks of data plays in important role in the model choice and development. Once the model is chosen, the next step is setting the parameter values as they might greatly affect the accuracy, efficiency and performance of the algorithm.

Finally, it is important to design an architecture taking into consideration all the specifics of the model. Serverless computing can prove to be quite helpful in this problem [2]. Serverless computing, also known as Function-as-a-Service (FaaS), is an alternative cloud execution model, which has been quickly adopted by developers. It helps to abstract from infrastructure problems and focus on business logic. All the developer has to do is to submit the function code to a cloud provider, which scales automatically and requires payment only for the time the code is running. Utilizing state-of-the-art techniques like this can help develop an efficient architectural model for your problem.

1.1 Problem

The problem we address in the thesis is the secure anomaly detection on distributed data.

We will deal with data containing sensitive information which can not be shared, thus we will need a technique to preserve privacy in the anomaly detection process. This kind of data is very common in real life and can be encountered in various situations, e.g. when dealing with patient monitoring, home security, etc.

In addition to this, data will not have a single source, but rather will be spread among multiple nodes with attached edge devices. The first idea is naturally to take advantage of edge computing [3], which enables the processing of data at the edge of the network. Putting all the computing tasks on the cloud might be an efficient way for data processing, as the computing power on the cloud usually exceeds the capability of the things at the edge. But if we care about fast data transportation, short response time and reliability, we might need another solution. In edge computing, we put the computing at the proximity of data sources and gain several benefits compared to traditional cloud-based computing paradigm.

Anomaly detection in edge computing is relatively complicated, as the edge devices usually don't have the required resources to conduct this kind of analysis. And if we decide to send the data to the cloud, we might encounter severe privacy issues. Moreover, in case there is a very large number of nodes, efficient anomaly detection on small chunks of data might not produce optimal results. Thus, some kind of data sharing must be done securely.

1.2 Motivation

There exist various approaches for anomaly detection, such as statistical methods (distribution-based and depth-based) [4], clustering-based methods [5], reconstruction-based methods [6], etc. The major downsides of these methods is that they (i) are constrained to low dimensional and small data size, (ii) are not optimized to detect anomalous instances. An alternative approach is to focus on anomaly detection optimization, and an example of this method can be isolation forests [1].

Multiple different techniques can be used to achieve privacy guarantees. That can be done either via using synthetic data [7], differential privacy notion [8], cryptographic technique [9] or doing non-linear transformation of the data [10]. Choosing a suitable method is a crucial step in the development of the model.

Throughout this thesis we will experiment with a model for anomaly detection which will be based on the idea of isolation forests [1]. In order to ensure privacy, we will use a cryptographic technique. Finally, we will take advantage of edge and serverless computing to get high efficiency and optimal performance.

1.3 Contributions

In this thesis, we address the problem of secure anomaly detection on serverless computing. Following contributions are made.

- We develop a secure anomaly detection model based on the idea of isolation forests. The model is called **SECURE-SERVERLESS: SECURE anomaly detection on SERVERLESS edge computing**.

- We analyze the algorithm's behavior on various datasets, experimenting with different setups.
- We propose improvements to overcome several challenging scenarios.

1.4 Outline

In chapter 2, we present several widely known techniques to solve similar problems and discuss their pros and cons. In chapter 3, we introduce the necessary background information required for the model development. In chapter 4, we present the problem with all the details and describe the research questions the thesis will focus on. In chapter 5, we describe the proposed solution giving detailed information about the model, made improvements, general workflow and architecture. In chapter 6, we present the experimental setup which gives information about the used datasets, essential parameters, etc. In chapter 7, we describe the experiments and present the evaluation results with a short discussion. In chapter 8, we finalize our findings. In chapter 9, we summarize our contributions and suggest potential future improvements.

2 Related Work

In this chapter, we want to present the related work from several topics that are essential for this thesis. We start by presenting various privacy-preserving techniques in section 2.1. Afterwards, we dive deeper into anomaly detection algorithms in section 2.2. In subsequent sections, we explore Serverless Computing and Edge Computing. We finish the chapter by presenting findings from the topic of anomaly detection in edge computing.

2.1 Privacy-preserving Techniques

With the increasing interest in data analysis, as more and more sensitive data is being collected, analyzed and processed, it is becoming extremely important to tackle privacy issues. Most of the time we deal with multi-dimensional sensitive data containing personal information and ensuring total anonymization becomes hard to achieve. In very rare cases the anonymization of data might be performed by eliminating or changing sensitive information, but in many cases the re-identification of data can be easily achieved.

There exist different techniques which can be used to achieve at least some privacy guarantees. That can be done either via using synthetic data [7], differential privacy notion [8], cryptographic technique [9] or doing non-linear transformation of the data [10].

2.1.1 Synthetic Data Usage

The idea of synthetic data [7] is to produce a dataset containing records that are similar to the original ones and preserve the high-level relationships within the real data, without actually revealing the single data points. With the help of this method, we can still conduct our analysis and the results will be close to the ones that can be achieved through the real data. Of course, one downside of this approach is that although the global structure of the data is preserved, we still have a loss in data utility. So, it is important to be careful when deciding whether this can be an acceptable approach for a specific scenario. To generate synthetic data, a preprocessing step can be configured, which would have access to the real data, learn the pattern and generate similar data which will be later used for the training. For this purpose, models having different levels of complexity can be used, ones that learn independent probability density functions for each attribute or more advanced models which try to capture also correlations between attribute values.

It is possible to make the data either fully or partially synthetic [7]. Depending on the case, it might make sense to replace only those attribute values which contain sensitive information or even choose a group of samples which must be replaced. If full synthetic data is being used, the possibility that the intruder will be able to link the generated data to the original one having no information is very low. Research has been conducted to get insights about how effective this method is for more advanced attacks, e.g., when the intruder has attribute information, etc. [7].

Utilizing synthetic methods in the anomaly detection setting can be tricky. The issue is that this method generates data which has the general pattern similar to the real data, and it doesn't care about single data points. As a result, this new generated data won't have points similar to the outliers and the anomaly detection might produce weird results. Several studies have analyzed this peculiar case to find out whether this method can be used in this setting. These studies show that not all anomaly detection algorithms/techniques give promising results but in specific settings, models which were trained on the synthetic data might have similar effectiveness as when trained on original data [7].

2.1.2 Non-linear Data Transformations

Several studies have considered linear data transformations (additive, multiplicative or a combination of both) in order to prepare the sensitive data to be analyzed [11]. The idea of additive methods is to perturb data by adding noise to it, but it was shown that this can be detected using spectral filtering techniques. Afterwards, multiplicative perturbation techniques have been offered, one of which is to multiply a random number generated from a Gaussian distribution of mean one and small variance to each data point. But this method appeared to have vulnerabilities against specific attacks [11]. Another group of techniques focuses on projecting data into a random subspace using orthogonal or pseudo-random matrices [11]. In these approaches the distance between elements is preserved, thus algorithms depending on that can be easily used. In another study, a combination of the above-mentioned techniques is being used, but this approach has vulnerabilities against attackers who know a set of input-output pairs.

Another interesting approach conducted recently considered the case of non-linear transformations which turned out to be quite effective [10]. The highlight of this approach is that it allows the user to control the amount of privacy by varying the level of nonlinearity. This privacy technique essentially uses a random non-linear map to transform the input data. The mapping fulfills two properties: (1) for all points in the normal operating region, the mapping approximately keeps the distance between them in the transformed space, and (2) it maps all outliers to a finite set of discrete values. Consequently, the outliers will remain such after the transformation. It is shown that if this transformation is non-invertible, then it is nearly impossible to break it and reveal the initial data. This technique can be used for preserving

privacy in detecting fraud across financial institutions, finding anomalies in medical records, etc.

2.1.3 Differential Privacy

The current state-of-the-art technique for privacy preservation is differential privacy. A database privatization mechanism satisfies differential privacy if the addition or removal of a single data point does not alter the probability of any outcome of the privatization mechanism by more than some small amount [8]. The definition is intended to capture the notion that "distributional information is not private"—we may reveal that smoking correlates to lung cancer, but not that any individual has lung cancer. Individuals may submit their personal information to the database secure in the knowledge that (almost) nothing can be discovered from the database with their information that could not have been discovered without their information [12].

Using differential privacy in anomaly detection algorithms might be challenging, especially for the scenarios where we use distance-based detectors. The problem of outlier detection is to find a few instances that are significantly distant from the other instances. On the other hand, the objective of differential privacy is to conceal the presence (or absence) of any particular instance. Outlier detection and privacy protection are thus intrinsically conflicting tasks. Still this approach can be used to count the number of outliers or discover the subspaces with a lot of anomalous elements. [8]

2.1.4 Cryptographic Techniques

Encryption is the standard means for ensuring a private communication. The sender encodes each message before transferring it to the receiver. The receiver (but no unauthorized person) knows the appropriate decoding function to apply to the received message to obtain the original message. An attacker who hears the transmitted message hears only "garbage" (the ciphertext) which makes no sense to him since he does not know how to decrypt it [13].

If Bob wants to send Alice a message M in a public-key cryptosystem, the procedure goes as follows. First, he retrieves Alice's public key from the public file (or the public keys can be previously exchanged). Then he sends her the enciphered message. Alice deciphers the message using her private key. As only Alice has the required private key, no one else can decipher that text [13].

There exist several encryption schemes having interesting properties. One example can be El Gamal's scheme, which enjoys a multiplicative homomorphic property, by which one can easily obtain an encryption of m1*m2 by multiplying encryptions of m1 and m2. This is a nice property, but for many real applications it is preferable to have the additive homomorphic property, so that we can obtain the encryption of m1+m2 simply by combining

the ciphertexts. An example of a scheme which has this property is Paillier's scheme, based on which later Cramer and Shoup proposed a new scheme which allowed a double decryption mechanism [9], which will be used in this thesis.

2.2 Anomaly Detection Algorithms

While there is no single formal definition of an outlier, Hawkins' definition captures the main idea: "an outlier is an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [14]. Outlier or anomaly detection is the technique of finding patterns in a dataset that are inconsistent or considerably different from the rest of the data. The detection of anomalies often provides critical information in various application domains. For example, anomalies in credit card transactions could signify fraudulent use of credit cards. An anomalous spot in an astronomy image could indicate the discovery of a new star. An unusual computer network traffic pattern could stand for unauthorized access. These applications demand anomaly detection algorithms with high detection performance and fast execution. [1]

There are a lot of suggested methods for anomaly detection, such as statistical methods [4] (distribution-based and depth-based), distance-based methods [15], clustering-based methods [5], density-based methods [4], reconstruction-based methods [6] etc. The major drawback of most of these methods is that they are only applicable to low dimensional data and small datasets. Most of them construct a profile of normal instances, then identify instances that do not conform to the normal profile as anomalies [16]. Other approaches for anomaly detection which might be better suited for several scenarios are based on the concept of isolation, etc.

2.2.1 Statistical Methods

Most of the previous studies on anomaly detection were conducted in the field of statistics. These studies can be classified into two main categories.

The first category is distribution-based, where a standard distribution is used to fit the data best. A data point is classified as an outlier if the probability of it being generated from that distribution is below a certain threshold. The advantage of such models is that the decision of whether an object is anomalous is based on the calculated probability, which is objective and theoretically justifiable [16]. Over one hundred tests of this category, called discordancy tests, have been developed for different scenarios [4]. The main drawback of the distribution-based method is that the underlying distribution is usually unknown and is not a standard distribution for many practical applications [4].

The second category of outlier detection methods in statistics is depth-based. Each data object is represented as a point in a k-dimensional space and is assigned a depth. With respect to anomaly detection, outliers are more likely to be data objects with smaller depths. There

are many definitions of depth that have been proposed. In theory, depth-based approaches could work for large values of k. However, in practice, while there exist efficient algorithms for k = 2, 3, depth-based approaches become inefficient for large datasets for $k \geq 4$. [4]

2.2.2 Distance-based Methods

The distance-based methods identify outliers by computing distances among all data items. An element is considered as an outlier when it has d_0 distance away from p_0 percentage of items in the dataset [15]. In [17], the distance among objects is calculated in feature subspace through projections for high dimensional data sets. The problem of these methods is that the local outliers are usually misclassified for the data set with multiple clusters. To detect the local outliers, a top-n k-th nearest neighbor distance is proposed in [18], in which the distance from an object to its k-th nearest neighbor indicates outlierness of the object.

Depth-based outliers would be more applicable than distance-based outliers to situations where no reasonable metric distance function can be used. However, for numerous applications, defining a distance function is not hard.

2.2.3 Clustering-based Methods

Anomaly detection and clustering analysis are two interconnected tasks. Clustering finds patterns in a data set and organizes the data accordingly, whereas anomaly detection captures those exceptional cases that differ significantly. So the idea of these methods is to detect anomalies in the process of finding clusters: the samples that don't fit into any of the clusters are considered outliers. However, since the main objective of a clustering algorithm is to find clusters, they are developed to optimize clustering, and not to optimize outlier detection. The exceptions (called "noise" in the context of clustering) are typically just tolerated or ignored when producing the clustering result. Even if the outliers are not ignored, the notions of outliers are essentially binary, and there is no quantification as to how outlying an object is. [4]

2.2.4 Density-based Methods

In density-based methods, an outlier is detected when its local density differs from its neighborhood. Different density estimation methods can be applied to measure the density. The outlierness score can be based on the distance of the object from its local reachable neighborhood, the relative distance from an object to its neighbors (reverse neighbors might also be considered), etc. [4]

2.2.5 Reconstruction-based Methods

In reconstruction-based techniques, the anomalies are estimated based on reconstruction error. In this technique every normal sample is reconstructed accurately using a limited set of basis functions whereas abnormal data is observed to have larger reconstruction loss. Depending on the model type, different loss functions and basis functions might be used. Some methods use PCA, K-means, GANs, etc. [6]

Generative adversarial networks (GANs) are a class of models that have been successfully used to model complex and high-dimensional distributions. Intuitively, a GAN that has been well-trained to fit the distribution of normal samples should be able to reconstruct such a normal sample from a certain latent representation and classify the sample as coming from the true data distribution. However, as GANs only implicitly model the data distribution, using them for anomaly detection requires a costly optimization procedure to recover the latent representation of a given input example, making this an impractical approach for large datasets or real-time applications. To overcome this challenge, in one of the studies, researchers leverage GAN methods that simultaneously learn an encoder during training to develop an anomaly detection method that is efficient at test time [19].

Recently, a deep neural network DeepOC was developed which can simultaneously train a classifier and learn compact feature representations. This framework uses the reconstruction error between the ground truth and predicted future frame to detect anomalous events. [6]

2.2.6 Isolation Forests

Recently, a new approach has been proposed, which offers a different type of model-based method that explicitly isolates anomalies rather than profiles normal instances. To achieve this, the proposed method uses two main properties of anomalous samples: i) they are the minority consisting of fewer instances and ii) they have attribute-values that are very different from those of normal instances. In other words, anomalies are 'few and different', which makes them more susceptible to isolation than normal points. It is shown that a tree structure can be constructed effectively to isolate every single instance. Because of their susceptibility to isolation, anomalies are isolated closer to the root of the tree, whereas normal points are isolated at the deeper end of the tree. This isolation characteristic of tree forms the basis of this method to detect anomalies. [16]

2.3 Serverless Computing and FaaS platforms

Serverless computing is based on the idea that computing resources, along with their configuration and management, are dynamically provisioned at run-time by the cloud service provider. This is in contrast to traditional methods, in which the needed resources are planned

during application design and provided during the deployment phase. The name serverless emphasizes the fact that the application design does not include servers and developers can focus only on the application code.

Function-as-a-Service (FaaS), a key enabler of serverless computing, allows an application to be decomposed into simple, standalone functions that are executed on a FaaS platform [20]. This is the key difference between FaaS and the other cloud service models, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Instead of deploying a whole platform or an application in the cloud servers, by using FaaS just functions are required, as components of complex applications. Such functions can be loaded as virtual containers when needed, on demand, and possibly in parallel, without any need for controlling application deployment processes at the operating-system level. This means that containers are dynamically scheduled in the hardware infrastructure maintained by cloud providers. This approach allows service designers and users to only focus on the application logic, without having to deal with server management and autoscaling functions, which are intrinsically included in the FaaS service [21]. These functions are stateless and can be invoked by a user's HTTP request or by another type of event created within the FaaS platform. The pricing is charged based on the number of requests to the functions and the duration, the time it takes for the function code to execute [20].

The FaaS platform is responsible for provisioning resources for function invocations and performs automatic scaling [20]. FaaS platforms implementations are based on starting containers for function invocations on top of a container orchestration platform, e.g. Kubernetes. There is a wide range of both commercial fully managed FaaS platforms and non-commercial open-source FaaS platforms. Examples of fully managed offers include AWS Lambda [1], Google Cloud Functions, Azure Functions, and IBM Cloud Functions. Prominent candidates of open source FaaS platform implementations are OpenWhisk, OpenFaaS, Fission and Knative [2].

Apache OpenWhisk[2] is a serverless open-source cloud platform that was originally developed by a research group at IBM [20]. Functions in OpenWhisk are called actions, and the execution of an action is called an invocation. Created actions can be invoked either manually or by event triggers originating from timers, databases, message queues, etc. Invokers set up a new docker container for each action, inject the code into them, execute the code, obtain the results, and then destroy it. These containers are run inside Kubernetes pods. There can be an invoker per kubernetes worker node or an invoker can be responsible for managing multiple kubernetes worker nodes.

OpenFaaS[3] is another widely popular open-source serverless cloud platform. It implements FaaS on top of Kubernetes as the container orchestration platform, and you can interact with

[1] AWS Lambda, `https://docs.aws.amazon.com/lambda/latest/dg/welcome.html`
[2] OpenWhisk documentation, `https://openwhisk.apache.org/documentation.html`
[3] OpenFaaS documentation, `https://docs.openfaas.com/`

OpenFaaS resources directly through kubectl, the command line interface for Kubernetes. It utilizes Prometheus and its AlertManager to continuously expose metrics. Functions in OpenFaaS can be written in any language, and in contrast to OpenWhisk, you don't have to create custom runtimes to make it work. A prebuilt docker image of the function can be supplied to it. When a function is created, its code is pulled from the docker registry and executed inside a container [20].

Google Cloud Functions (GCF) is a serverless execution environment for building and connecting services in a cloud-based application offered by Google Compute Platform (GCP)[4]. With GCF, developers do not need to care about the infrastructure or worry about managing any servers, the whole environment including infrastructure, operating systems, and runtime environments are managed by Google. Each Cloud Function runs in its own isolated secure execution context, scales automatically, and has a lifecycle independent of other functions. Cloud Functions handles incoming requests by assigning them to instances of function. Depending on the volume of requests, as well as the number of existing function instances, Cloud Functions may assign a request to an existing instance or create a new one [20].

2.4 Edge Computing

Putting all the computing tasks on the cloud is an efficient way for data processing in multiple scenarios, since the computing power on the cloud outclasses the capability of the things at the edge. However, in many scenarios where the data transformation speed might be a possible bottleneck, and it is essential to have short response time as well as smaller network pressure, allowing data consumption and processing at the edge side proves to be effective.

Edge computing technologies allow computation to be performed at the edge of the network on behalf of cloud services. "Edge" is defined as any computing and network resource along the path between data sources and cloud data centers. [22]

The concept of Edge Computing is related to the so-called Fog Computing [23]. While the boundary between the two technologies is not clearly defined, edge computing is typically regarded as a service implemented either on the devices to which the sensors are attached or very close to it. Fog computing means a service that runs in a local network, close to sensors. Thus, running applications in edge computing means deploying them close enough to the end users in order to save transmission resources by processing data locally. The main features of Edge Computing include location awareness, dense geographical distribution, and limited amount of computing and storage resources available [21].

[4]GCF documentation, `https://cloud.google.com/functions/docs/concepts/overview`

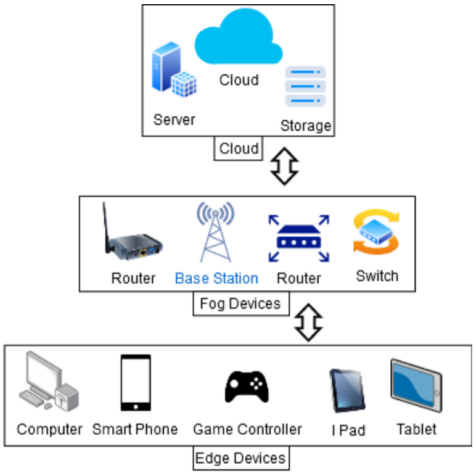


Figure 2.1: Architectural overview of edge computing [6]

2.5 Anomaly Detection in Edge Computing

In this section, we will present examples, where anomaly detection models on edge computing are proposed. We will see that in many critical situations edge computing is the best choice as it makes real time analysis possible.

2.5.1 Anomaly Detection in Underground Mining

The purpose of data anomaly detection in underground mining is to determine whether there is an abnormality in the current construction environment and to provide timely early warning. But the analysis of the multi-sensor data in this setting turns out to be a challenging task. The reason is that most of the existing methods process sensor data on a cloud server, which brings several problems: firstly, a lot of invalid and redundant data transmission wastes a limited network resources; secondly, some sensor data have real-time requirements for anomaly detection, which can not be satisfied if we depend on the cloud server. To resolve these issues a multi-sensors data anomaly detection method based on edge computing is proposed [24].

In the underground construction environment, the data are collected and sent by sensors, processed by the sink node, and then forwarded to the cloud. Because the cloud is usually deployed on a faraway server center, this will lead to a long data transmission time. But if an abnormality is found, then an early warning should be given in order to prevent fatalities, which might not be possible if we rely on the cloud server. If the sink node had enough capabilities, such as CPU, memory, etc. then it could process the data, but this is not usually

the case. So a new model should be adjusted, which will give possibility to analyze time-series data collected by multiple sensors across different locations and do all that within a given period of time [24].

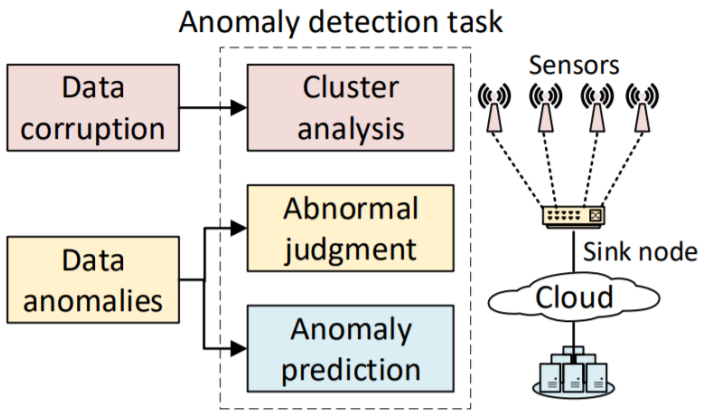


Figure 2.2: Edge computing model in underground mining [24]

In order to use the benefits of edge computing, the task of data anomaly detection needs to be divided into several parts and migrated to different edge devices for execution. In underground mining, there are two types of edge devices: sink node and sensor node. A sensor node will take care of data collection and data preprocessing, and a sink node is there for data aggregation, fusion, and forwarding. In this problem, the data anomaly detection task will be reasonably migrated to the sensor node and the sink node. Anomaly detection task is implemented in three parts: 1) cluster analysis carried out by the sensors to determine whether data is damaged due to factors, such as equipment failure 2) abnormal judgement executed by the sink node, which determines whether the environment is abnormal based on the data sent by multiple sensors 3) anomaly prediction performed by the cloud [24].

3 Background

In this chapter, we present the necessary background for this work. In section 3.1, we introduce a public-key cryptosystem, which will be utilized in the scope of this thesis in order to ensure privacy guarantees. In subsequent sections, we present anomaly detection techniques which serve as the basis for our model. We start by describing Isolation Forests in section 3.2 and then dive deeper into Secure Isolation Forests in section 3.3.

3.1 BCP Cryptosystem

3.1.1 Motivation

In order to preserve privacy in the anomaly detection process, we will first utilize a simple public-key cryptosystem. The idea of a public key cryptosystem is to find an encryption function E, which is easy to compute but hard to invert unless some secret information, the *trapdoor* is known [25]. We will focus on a public key cryptosystem with a double trapdoor decryption mechanism (we will refer to it as BCP cryptosystem from now on) [9]. It achieves the most important notion of cryptosystems which is the semantic security (a.k.a ciphertext indistinguishability) as well as offers useful properties. One of these properties is providing a double decryption mechanism, which allows decryption not only via the secret key (private key) by the single recipient, but also via a unique master key. This is especially useful when we have a hierarchy in the group, and we want to allow the authority to decrypt all the information while the others shouldn't have access to each other's data. Another interesting characteristic that this cryptosystem has is the additive homomorphic property, which allows encrypted data to be combined without revealing them explicitly.

In the next subsections, we will dive deeper into the inner workings of this cryptosystem and look at the implementation details.

3.1.2 Implementation Details

First, let's define Z_N and Z_N^* [1]. Z_N is the set of non-negative integers less than n. Z_N^* is a subset of this, which is the multiplicative group for Z_N modulo N. The set Z_N^* is the set of

[1] Mult. group modulo n, `https://en.wikipedia.org/wiki/Multiplicative_group_of_integers_modulo_n`

integers between 1 and N that are relatively prime to N (they do not share any factors). If N is prime, then Z_N^* includes values up to N - 1.

BCP cryptosystem is based on Cramer and Schoup's methodology [26], which is an improved version of Paillier's original scheme [27]. The basic idea of Paillier's scheme is that to encrypt a message, $m \in Z_N$ one selects a random value $y \in Z_N^*$ and sets the ciphertext as $g^m y^N mod N^2$ (where g is an element whose order is a multiple of N in $Z_{N^2}^*$). The semantic security of the scheme is proved with respect to the decisional N-th residuosity assumption: given a random value $x \in Z_N^*$ it is computationally infeasible to decide if there exists another element $z \in Z_{N^2}^*$ such that $x \equiv z^N mod N^2$. But if the factorization of the modulus N is known, this problem becomes easily tractable. As no adaptive chosen ciphertext attack recovering the factorization of the modulus is known, Paillier's proposal is considered to be the best solution among additively homomorphic cryptosystems. Later, Cramer and Schoup proposed an adjustment to Paillier's scheme which strengthened the security properties and allowed a double decryption mechanism.

Let N be a product of two safe primes p and q [2]. The message to be encrypted is in Z_N. Here are the main functional units of the cryptosystem.

- **Key Generation** - Choose a random element $\alpha \in Z_{N^2}^*$, a random value $a \in [1, ord(G)]$ and set $g = \alpha^2 \mod N^2$ and $h = g^a \mod N^2$. The public key is given by the triplet, (N, g, h) while the corresponding secret key is a.
- **Encrypt** - Given a message $m \in Z_N$, a random pad r is chosen uniformly and at random in Z_{N^2}. The ciphertext (A, B) is computed as

$$A = g^r \mod N^2 \tag{3.1}$$

$$B = h^r(1 + mN) \mod N^2 \tag{3.2}$$

- **First Decryption Procedure** - Knowing a, one can compute m as follows

$$m = \frac{B/(A^a) - 1 \mod N^2}{N} \tag{3.3}$$

- **Alternate Decryption Procedure** - If the factorization of the modulus is provided, one can compute $a \mod N$ and $r \mod N$ [9]. Let $ar \mod ord(G) = \gamma_1 + \gamma_2 N$, thus $\gamma_1 = ar \mod N$ is efficiently computable. Note that

[2] Safe prime, https://en.wikipedia.org/wiki/Safe_and_Sophie_Germain_primes/

$$D = (\frac{B}{g^{\gamma_1}})^{\lambda(N)} = \frac{(g^{ar}(1+mN))^{\lambda(N)}}{g^{\gamma_1\lambda(N)}} = 1 + m\lambda(N)N \mod N^2 \tag{3.4}$$

Denoting by π the inverse of $\lambda(N)$ in Z_N^*, one can compute m as

$$m = \frac{D - 1 mod N^2}{N} \pi(mod N) \tag{3.5}$$

As we can see from the defined procedure, knowing the factorization of the modulus (p and q values) one is able to decrypt ciphertexts generated with respect to any public key. Another observation to keep in mind is that if a ciphertext is not correctly generated, this can be revealed when decrypting using the private key, but faults can't be detected when using the alternate decryption mechanism [9].

In order to provide insight into why privacy is guaranteed in this cryptosystem, two problems were introduced, which are the Partial Discrete Logarithm problem proposed by Paillier and the Decisional Diffie-Hellman problem [9]. The investigation of these problems proves that the cryptosystem is well protected against several attacks and that these problems can be efficiently solved if the modulus is known. Thus, without knowing the two safe primes which form N, the system does not have severe vulnerabilities.

3.2 Isolation Forests

3.2.1 Motivation

In order to detect anomalies in the data, we will use an approach which drastically differs from many existing methods. As discussed in the related work review, there exist numerous approaches which construct a profile of normal instances and then detect outliers. These detectors are not optimized to identify anomalies, which is a big drawback. Another disadvantage of these methods is that they are mainly used for small datasets or datasets with few properties. The approach which we will leverage uses the idea of isolation and specifically tries to detect anomalies. The model is called Isolation Forest [1]. It also works well in high-dimensional problems or situations where the dataset doesn't contain anomalies. What is also worth mentioning this method is that it doesn't depend on distance measuring between instances, which could be quite costly.

In the final section we will have a look at a model which utilizes the encryption scheme discussed before in the isolation forests, and this will be the main model we will be relying on throughout this thesis.

3.2.2 Implementation Details

The main idea of the proposed model is that anomalies in the data are 'few and different', which makes them easier to identify via isolation. The model effectively constructs a tree structure (iTree) in which every single element is isolated. Because of the susceptibility to isolation, anomalous samples are isolated closer to the root while the normal instances are isolated deeper in the trees [3].

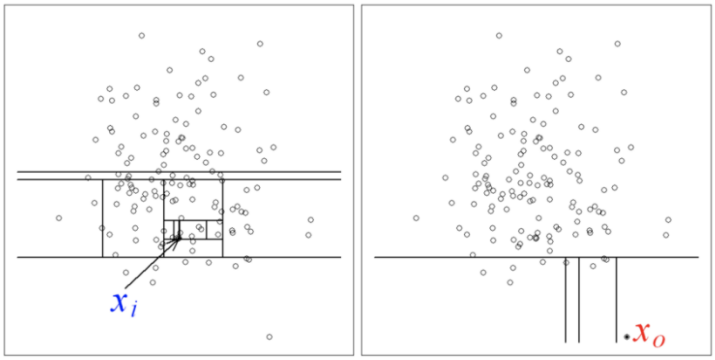


Figure 3.1: Identifying normal vs anomalous instances

An ensemble of isolation trees forming an isolation forest (iForest) is built, and the instances which have the shortest average path length are classified as anomalies. The variables in this method are the number of trees, tree height limit and subsample size for each tree construction. Choosing them even not so big, this approach ensures high detection performance and high efficiency.

Anomaly detection is done in two stages. In the training stage, isolation trees are built using subsamples of the training set data. In the testing stage, instances go through the trees and the anomaly scores are calculated.

Given a chunk of data X = $\{x_1, ..., x_n\}$, to build an isolation tree, we recursively divide X by randomly selecting an attribute q and a split value p, until either: (i) the tree reaches a height limit, (ii) $|X| = 1$ or (iii) all data in X have the same values. Each instance is being isolated to an external node and can be assigned an anomaly score. For the anomaly score calculation, we need to define the notion of path length. Path length $h(x)$ of an instance is the number of edges that must be traversed until an external node is encountered. Given a dataset of n elements, the anomaly score for the instance x is calculated using the following formula.

[3] `Isolationforest,https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e`

$$s(x,n) = 2^{\frac{-E(h(x))}{c(n)}} \tag{3.6}$$

$$c(n) = 2H(n-1) - (2(n-1)/n) \tag{3.7}$$

where $H(i)$ is the harmonic number [4], $h(x)$ is the path length, $E(h(x))$ is the average path length among all the trees.

Using the anomaly score s, we will assess the possibility of an instance to be anomalous in the following way:

- if instances return s very close to 1, then they are definitely anomalies
- if instances have s much smaller than 0.5, then they are quite safe to be regarded as normal instances, and
- if all the instances return s > 0.5, then the entire sample does not really have any distinct anomaly.

3.2.3 Observations

Some interesting observations to keep in mind when working with iForests are the following:

- Contrary to many other methods, a large sampling size is not desirable in this case, because normal instances might interfere with the process and prevent it from clearly finding the anomalies. Once a specific sub-sampling size is reached, there is no benefit in increasing it as the performance is not improving anymore.
- When normal instances are too close to outliers or there are too many outliers close to each other, identifying them might be problematic for many methods. As iForest uses multiple trees to build partial models and each tree does a sub-sampling, the possibility that these problems will become less disturbing increases.
- Wisely choosing iForest parameters, the model can still achieve high accuracy when dealing with data which consists only of normal instances.
- iForests give promising results when working with high-dimensional data, and it was noted that in this case the detection performance improves when the subspace size comes closer to the original number of attributes.

[4] Harmonic number, `https://en.wikipedia.org/wiki/Harmonic_number`

3.3 Secure Isolation Forest

3.3.1 Motivation

Imagine we have a distributed system, where data is generated by different sources and has to be kept private. In order to make anomaly detection possible in such scenarios, we have to improve the Isolation Forest model, so that analysis will be possible despite the fact that the data is to be kept safe.

In paper [28], an adjusted model is proposed (Secure Isolation Forest, from now on will be referred to as SIF) which is based on Isolation Forests and utilizes the BCP cryptosystem in order to keep sensitive information private. This solution can handle inputs encrypted by different independent public keys and, as shown in the paper, proves to be more efficient compared to many other models.

3.3.2 Implementation Details

SIF consists of multiple secure isolation trees (from now on SIT), which are full binary trees consisting of $2^{h+1} - 1$ nodes, in which all leaves are at the same depth h. The building process of a SIT is a recursive algorithm which works by splitting the dataset into two parts at each node until max depth is reached. The algorithm randomly selects a dimension q and selects the mid-point of q $((max(dataset_q) + min(dataset_q))/2)$ dividing the dataset into two parts (greater and less than q). Each node has an encrypted message which indicates the number of instances that node contains ($Node.Size$).

Score(x, T) measures the anomaly score of an instance x in a SIT. It will search from the root of an SIT until a terminal node is found in order to find the partition of x. This function then returns the anomaly score of x in the given tree using the following formula.

$$Score(x, T) = Node.Size \times 2^{Node.depth} \tag{3.8}$$

Here, Node depth is the depth of the terminal node which contains Node.Size instances. The final anomaly score of x is the average of scores obtained from each SIT in the forest:

The main issue here is that the $Node.Size$ values are encrypted in an SIT and the trees are encrypted by different public keys. Thus, combining the anomaly score values becomes difficult. Now we will present the process that helps us overcome these problems. So we have multiple clients which use different public keys for size value encryption. And these clients exchange their encrypted trees and need a way to combine the anomaly scores. The properties that come to rescue us in this scenario are that the BCP cryptosystem offers a double decryption mechanism and has the additive homomorphic property. So the client, that wants to combine anomaly scores for a given instance in different trees, will transform

these encrypted values to be encrypted by a single public key (which will be the product of all the public keys used). Once the scores are encrypted by the same public key, they can be easily combined (based on the additive homomorphic property). Finally, to get the real score, we can decrypt the value using the master key.

4 Problem Statement

In this chapter, we present the problem addressed throughout the thesis. We define the desired outcome of our work and specify the research questions, which should be answered as part of the solution.

4.1 Problem Introduction

Suppose we have multiple sensors, which reside far from each other and produce similar data containing sensitive information, which can not be shared. Edge devices are placed next to the sensors and receive the partial data generated by a single sensor. It is important to note that these devices do not have high computational power, so the operations they can run are strictly limited. Still, some computations are possible to conduct with low latency if the power of these devices is utilized.

Our task is to enable secure anomaly detection on the produced data in order to detect inconsistent patterns, which could be used to raise alarms in critical situations. In order to achieve high performance and efficiency, we will utilize the computational power of the edge devices, thus highly relying on the advantages of the Edge/Fog Computing technologies.

4.2 Desired Outcome

Our solution should provide an efficient, secure anomaly detection algorithm.

- We will strive for an efficient and accurate anomaly detection algorithm
- We will conduct the analysis on all the data produced by multiple sensors
- No sensitive and private information will be leaked in the analysis process
- The computational power of the edge devices will be utilized in order to achieve high performance

4.2.1 Research Questions

The following research questions are addressed in the thesis.

- How to ensure data security in the anomaly detection process?
- What are the optimal parameter values for the defined algorithm?
- How can we deal with data that doesn't have a typical structure?
- How to deal with the situation where we have a very large number of sensors, which produce a very small portion of the data (e.g. only 10 instances)?
- Can we detect the contamination percentage of the data without beforehand analysis and labeling?

In the subsequent chapters, we will present the solution model and will show the results from multiple experiments which were conducted to find answers to these questions. Some results are not limited to the use of the given algorithm, and can also be beneficial in the context of other methodologies. For example, automatic detection of the contamination percentage or ensuring sensitive data protection problems are common to many other algorithms.

5 Proposed Solution

In this chapter, we give detailed information about the proposed solution. In the first section we present the solution concept developed in the scope of this thesis. In the next section we discuss the solution's architecture and infrastructure.

5.1 Solution Concept

5.1.1 General Idea

As mentioned in the problem statement, we have multiple sensors which generate data and edge devices attached to these sensors. The edge devices can't send the original sensor data to other devices due to privacy issues, and neither can they detect anomalies based only on their chunk of data. In order to overcome this challenge, we will use them to create secure isolation trees (SIT, for more information please refer to §3.3) based on the partial data that they have and then combine the outcomes from these trees on a central computation unit side. At the end, this central unit (anomaly detector) can conduct the analysis and identify anomalies in the data. From now on, the solution model we present in this thesis will be referred to as **SECURE-SERVERLESS: SECURE anomaly detection on SERVERLESS edge computing**.

5.1.2 Components

There are four conceptual components present in this setting:

- Sensor, which
  - Produces data for a single edge device (we will simulate data generation using already existing datasets and will split it beforehand among edge devices)
- Edge Device/Node, which
  - Analyzes the data produced by a single sensor
  - Constructs several secure isolation trees
  - Sends these trees to a storage accessible to the anomaly detector

- Cryptosystem Regulator (CSR), which
  - Runs the cryptosystem initialization unit
  - Makes the information needed for decryption accessible only to the anomaly detector
  - Makes the encryption functionality available to all the nodes
- Anomaly Detector (AD), which
  - Has access to the information needed for value decryption
  - Analyzes secure isolation trees generated by multiple nodes
  - Detects outliers by calculating anomaly scores for the instances

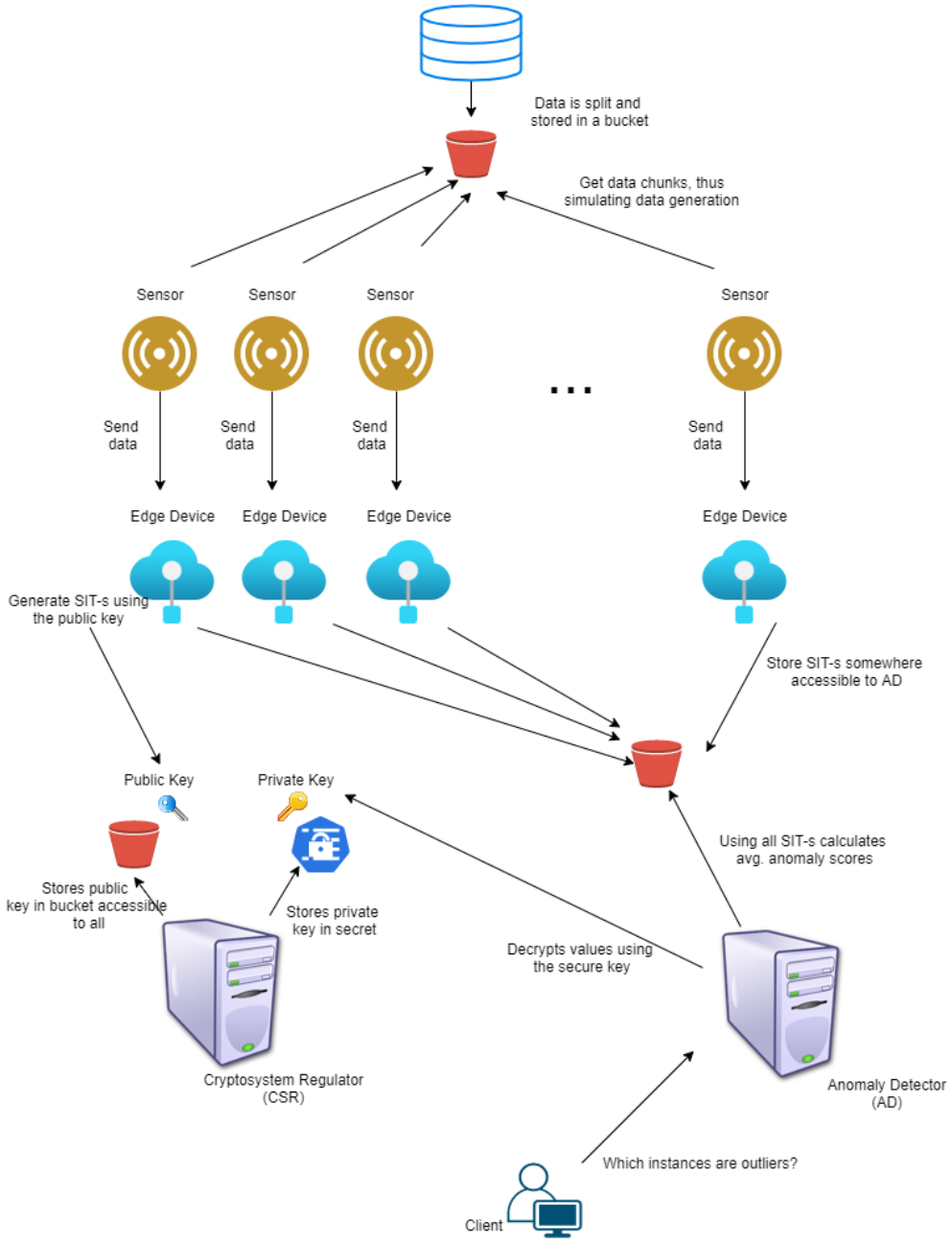


Figure 5.1: Workflow diagram

5.1.3 Details

Below, we present the detailed steps of the flow implemented as part of the solution concept.

- As the first step, the chosen cryptosystem's initialization unit is called, which generates the necessary keys. The keys/parameters used by all the nodes in the process of secure isolation tree generation are kept in a publicly available storage. As none of the nodes has access to the information needed for decryption, the encrypted values can not be decrypted by them. This information is securely kept in a dedicated secret manager and is accessible only to the anomaly detector. The format and contents of the keys/parameters depend on the cryptosystem which is being used.

  It was crucial to choose an appropriate cryptosystem, which would be easy to use and efficient. Initially, the BCP cryptosystem (§3.1) was utilized in the method. As there is no publicly available implementation for this method, we implemented it from scratch. Although our method which utilized this cryptosystem was producing optimal results in terms of accuracy and other evaluation metrics, the performance was quite poor. This was mainly because of manipulations and calculations with big numbers, which needed to be optimized. Another challenge we were facing in the development process of the cryptosystem was the problem of safe prime generation [1], which was needed for the private key. Although this problem was solvable, we didn't want to spend a lot of time on it. Moreover, as our setting differed from the one we see in the Secure Isolation Forest model (§3.3), it was not necessary for us to use a cryptosystem which had specific properties (e.g. additive homomorphism). Thus, we had the possibility to experiment with different cryptosystems. As a result, we started using the Fernet[2] module of the cryptography package in order to have enough time to focus on the core algorithm development and improvement. More information about Fernet can be found in the specification[3].

- In real life, data would be gathered by the sensors and sent to the attached edge devices. But as this would require specialist knowledge to prepare and label the data, in our experiments we simulate data generation using publicly available labeled data sources which already contain a number of anomalies [4]. Before the training stage, we take the dataset, split it into a number of chunks (depending on how many nodes we have) and send them to a storage. Each edge device will query this storage and get its chunk of data for the training stage. For more information about the publicly available datasources you can have a look at the section (§6.1).

[1] Safe prime, `https://en.wikipedia.org/wiki/Safe_and_Sophie_Germain_primes/`
[2] Fernet, `https://cryptography.io/en/latest/fernet/`
[3] Fernet spec, `https://github.com/fernet/spec/blob/master/Spec.md`
[4] Publicly available datasets, `http://odds.cs.stonybrook.edu/`

- Edge devices, receiving their chunk of data, construct iForests and send them to a storage accessible to the anomaly detector. Each tree in this forest is constructed using a subset or all of the data chunk (depending on the experiment). As mentioned in the original paper, isolation forests work best with small subsets of the data, and it was interesting to get convinced in that through experiments. Each tree node contains the split attribute, split value and the encrypted subtree size value. Split attributes and split values are chosen randomly in the tree construction process. The number of trees in the iForest and the maximum tree height are variables, optimality of which is investigated via experiments. What is worthwhile to mention here is that through experiments we found an interesting correlation between subsampling size and number of trees (more details in §7 chapter).

  As an improvement to the main algorithm, we further experimented with multiple split attributes in order to better deal with data which doesn't follow a specific pattern (vertical and horizontal split lines do not provide sufficient results) [29]. We chose split attributes by generating vectors of column number length, which had elements either in $\{0, 1\}$ with different ratios (e.g. 0.9% 0s and 0.1% 1s, 0.5% 0s and 0.5% 1s, etc.) or in range [0, 1].

  As mentioned in one of the articles ([30]), one important fault in the Secure Isolation Forest model (§3.3) is that the nodes construct trees based only on their chunk of data and ignore the general properties of the data. Thus, having numerous nodes results in a situation, when trees are constructed from e.g. 10 instances which will obviously not provide accurate results. So in order to get higher accuracy in similar cases, we generated synthetic data and shared it among nodes, so that each node had an idea about the general data structure. We had to be careful not to allow re-engineering of the received data, so the initial sensitive data would be kept safe. The edge devices would construct the trees based on their chunk of data merged with the synthetic data. We will see in the experiments that this gives much higher accuracy in case we have multiple nodes, which process only a very small portion of data. To the best of our knowledge, synthetic data hasn't yet been used in this manner in the anomaly detection process.

- Once the trees are collected, the training stage is over and the model is ready to identify anomalies. In order to calculate the score for a single instance, the instance is being passed through all the trees and the terminal nodes are found. At the terminal nodes the encrypted size values are decrypted, anomaly scores are calculated (using 3.8) and the average of these scores is taken.

- Instances are sorted by the anomaly scores and the higher the score, the lower is the possibility for that instance to be an outlier. Having the percentage of expected outliers, we return that many instances having the lowest scores as anomalies.

One of the challenges we were facing was that the number of outliers (contamination percentage) had to be specified. If the proportion of the anomalous instances is known at the beginning of the analysis (as it happens to be the case in many publicly available datasets), this should not be a problem. Otherwise, the proportion of outliers should be previously determined in a preprocessing step. But in several situations this might be problematic, as detecting this percentage is somewhat equivalent to understanding which are the outliers, but this is the goal of the anomaly detection process. So manual labeling and specialist knowledge might be needed. In order to automatically detect this value, it might be needed to have a deeper look at the anomaly scores, which we did in our experiments.

5.1.4 Real Time Analysis

Our solution can be easily customized to be used in real time setting. In order to implement that, the edge devices will construct the trees every time they receive a new chunk of data and send them to the storage. So the AD will start detecting outliers on a larger number of SIT-s and as a result will consider also the new data. Thus, gradually the anomaly detection will be adjusted to the data changes.

5.2 System Architecture and Infrastructure

In order to analyze and evaluate the described solution, it was deployed on AWS [5] and the overall infrastructure diagram is shown in Figure 5.2. Main AWS services used for the deployment were Lambda, IAM, S3, Secrets Manager and API Gateway. Further, in this section we present in more detail, how these services served us.

Several Function-as-a-Service functions were deployed on AWS Lambda.

1. prepare_data
   - Reads the .mat file of the dataset
   - Stores the loaded and processed data (X and labels) in an S3 bucket
   - Splits the dataset into chunks (the number must be provided in the call to the lambda) and saves them in an S3 bucket
2. create_and_send_sit_to_cloud
   - Loads a data chunk (depending on which node it is)
   - Creates multiple secure isolation trees (parameters are specified in the call)

[5] AWS, https://aws.amazon.com/

- Sends these trees to the S3 bucket

3. detect_outliers
   - Loads the isolation trees from the S3 bucket. It is interesting to note that it doesn't have an idea how many trees to accept or which nodes have finished the creation of the trees. The lambda just loads the saved trees and conducts the analysis based on the given trees.
   - For each instance in the dataset, passes it through the trees and calculates the anomaly score
   - Returns the indexes of the anomalies
4. encrypt
   - Returns the encrypted ciphertext
   - Encryption is done using the cryptosystem initialized and saved in a file
5. decrypt
   - Decrypts the given ciphertext
6. client
   - Invokes the lambdas with different values for the memory parameter
   - Returns information about how long each conceptual part lasts

Several S3 buckets were created.

1. datasets-for-analysis: stores the .mat files of the publicly available datasets which would be used for the analysis (needs to be filled manually)
2. data-chunks: stores the whole training data and the data chunks for each node (will be filled by the prepare_data lambda)
3. secure-isolation-trees: stores the isolation trees created by the nodes

Two API Gateways were created: for encryption and decryption. The API call for encryption is available to everyone and doesn't require any authorization. In contrast, the decryption API call requires the API key for authorization. This API key is saved in the Secrets Manager and is accessible only to the detect_outliers lambda.

To give necessary access and permissions, several policies were created.

1. allow-secret-access: allows the detect_outliers lambda access to the secret API key

2. allow-sit-creation-lambda-invocation: allows client lambda to invoke the create_and_send_sit_to_cloud lambda
3. allow-anomaly-detection-lambda-invocation: allows client lambda invoke the detect_outliers lambda

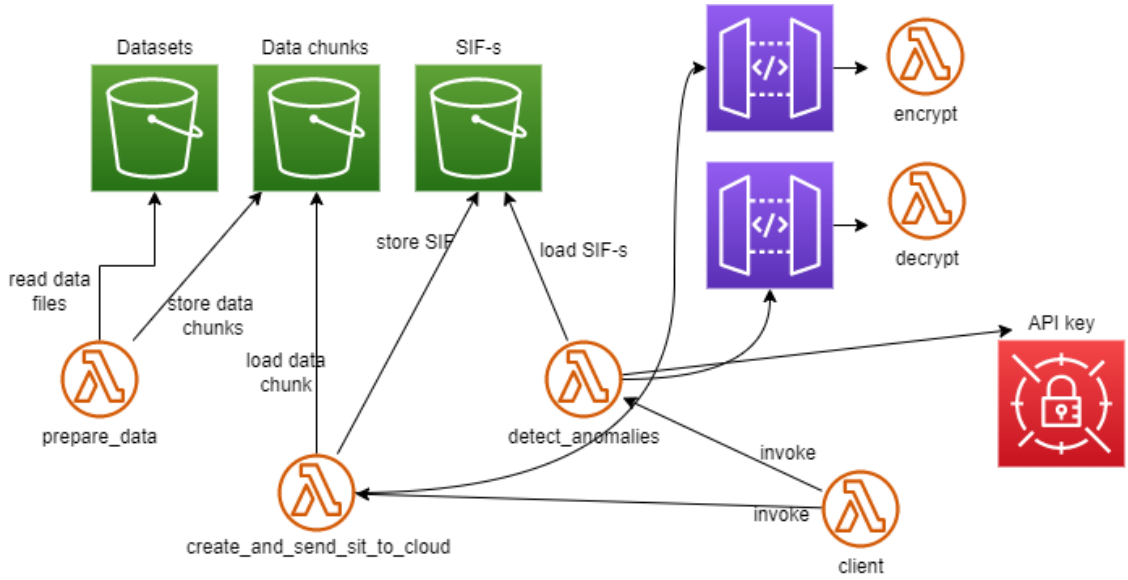


Figure 5.2: Infrastructure

The described infrastructure can easily be replicated on other platforms as well.

6 Experimental Setup

In this chapter, we present details about the experimental setup. We start the chapter by giving information about the datasets which will be used in the experiments. Afterwards, we present the parameters which will be considered during the process.

6.1 Datasets

The experiments will be conducted on publicly available datasets[1]. We will focus on multi-dimensional point datasets, where there is one record per data point, and each record contains several attributes. We will choose several datasets with interesting properties to fully observe the model behavior.

Our final set should contain datasets from the following categories:

- **M** - Medium dataset that has a normal number of anomalies. As anomalies are considered to be few and different, we expect that a normal dataset shouldn't have numerous outliers. These sets will be the primary test sets for the analysis.
- **L** - Large dataset to analyze the model behavior on big data. The model our solution is based on (isolation forest) tends to work very well with large and high-dimensional datasets. So, it is interesting to observe how our model behaves in these cases.
- **HD** - High-dimensional dataset

Table 6.1 shows more information about the chosen datasets.

Dataset	#points	#dim	#outliers(%)	Category
Pima	768	8	268 (35%)	M
Breastw	683	9	239 (35%)	M
Ionosphere	351	33	126 (36%)	M
Annthyroid	7200	6	534 (7.42%)	L
Arrhythmia	452	274	66 (15%)	HD

Table 6.1: Datasets

[1]Datasets, http://odds.cs.stonybrook.edu

6.2 Parameters

In this section, we will introduce the parameters of our model, which will be considered in the experiments and play an important role in the results. These parameters can be split into 3 categories: CRYPTO, IF and INFRA. The CRYPTO parameters will be the parameters of the chosen cryptosystem which is used to make the isolation forest secure. These will affect achieved privacy guarantees, and it is essential to find optimal values because CPU/memory usage heavily depend on them. IF parameters will affect anomaly detector efficiency and of course are important to consider in order to achieve high performance. And finally, INFRA group consists of the parameters which come from the defined infrastructure.

CRYPTO (BCP)	IF	INFRA
Modulus of N	Number of Nodes	Memory
Generated parameter range	Node data size	
	Number of trees	
	Tree generation data size	
	Number of features	

Table 6.2: Parameters

6.2.1 CRYPTO parameters

These parameters vary depending on the chosen cryptosystem. When experimenting with BCP cryptosystem, we should consider the following parameters:

- **Modulus of N (private key)**: p and q safe primes such that p*q=N
- **Several randomly generated values**: particularly a and r which are used in key generation

The built-in Fernet module, which was further used, does not require any parameters to be provided.

6.2.2 IF parameters

The anomaly detector algorithm which is used in the proposed solution has several parameters, which heavily affect the performance and accuracy of the algorithm. So, in order to find the optimal values for them, we will experiment with various numbers. The parameters that play an essential role are the following:

- **Number of nodes (a.k.a. edge devices)**. This will be the same as the number of sensors, as in our setting each sensor has an edge device connected to it.
- **Dataset size for each node**. As we imitate data generation by sensors, we have to split the existing dataset into parts as if each part is generated by a single sensor. We will randomly split data without caring about how many anomalies are included in each chunk. A further step could be to distribute anomalies uniformly, or gather the anomalies in 2-3 data chunks and see how the model reacts.
- **Number of trees generated by each node**. We can use the same value for all the nodes or vary them across the nodes.
- **Dataset size for a single tree generation**. As shown in the original paper, it is important to consider this parameter, as isolation trees work best when the data size is not too large.
- **Number of features used for a single tree generation**. This is the same as the maximum height of the secure isolation trees.

In the scope of the thesis we will focus on the following parameters: #nodes, #trees, #features, data size for a single tree construction. There are some suggested values mentioned in the original paper of isolation forests, which might prove to be helpful in finding the optimal parameter values for our model.

6.2.3 INFRA parameters

These parameters will highly affect the performance of the model and have to be considered. The parameter that we will focus on is the memory provided to the serverless functions.

7 Experimental Results

We start this chapter by presenting the evaluation metrics used for result analysis. Afterwards, we describe the experiments which were conducted to answer the research questions raised in the problem statement.

7.1 Evaluation Metrics

In order to evaluate the anomaly detection algorithm, we will use metrics, which are widely used in classification problems. These metrics are based on the elements of the confusion matrix defined in Table 7.1, and Table 7.2.

	Actual Positive Class	**Actual Negative Class**
Predicted Positive Class	True positive (tp)	False negative (fn)
Predicted Negative Class	False positive (fp)	True negative (tn)

Table 7.1: Confusion matrix [31]

Metric	Formula	Evaluation Focus
Accuracy (acc)	$\frac{tp+tn}{tp+fp+tn+fn}$	In general, the accuracy metric measures the ratio of correct predictions over the total number of instances evaluated.
Error Rate (err)	$\frac{fp+fn}{tp+fp+tn+fn}$	Misclassification error measures the ratio of incorrect predictions over the total number of instances evaluated.
Sensitivity (sn)	$\frac{tp}{tp+fn}$	The metrics is used to measure the fraction of positive patterns that are correctly classified.
Specificity (sp)	$\frac{tn}{tn+fp}$	This metric is used to measure the fraction of negative patterns that are correctly classified.
Precision (p)	$\frac{tp}{tp+fp}$	Precision is used to measure the positive patterns that are correctly predicted from the total predicted patterns in a positive class.
Recall (r)	$\frac{tp}{tp+tn}$	Recall is used to measure the fraction of positive patterns that are correctly classified.
F-Measure (FM/F1)	$\frac{2*p*r}{p+r}$	This metric represents the harmonic mean between recall and precision values.

Table 7.2: Evaluation metrics [31]

We will mainly focus on accuracy, precision, recall and F-Measure/F1 score. Another metric that we will also consider is the AUC score [31]. The AUC value reflects the overall ranking performance of a classifier. For two-class problems, the AUC value can be calculated as below

$$AUC = \frac{S_p - n_p(n_n+1)/2}{n_p n_n} \tag{7.1}$$

where, S_p is the sum of the all positive examples ranked, while n_p and n_n denote the number of positive and negative examples, respectively. The AUC was proven to be theoretically and empirically better than the accuracy metric for evaluating the classifier performance and discriminating an optimal solution during the classification training [32].

7.2 Experiments

7.2.1 Algorithm Analysis

We start this section by conducting basic experiments to estimate the algorithm's accuracy, efficiency, etc. We will also analyze the algorithm's performance in comparison to the case, when encryption is disabled, to estimate the impact of the cryptosystem integration into the model. For these experiments we suppose having a single node and for the construction of each tree use 256 instances (as this is the optimal subsampling size mentioned in the original paper [1]). For these experiments, we ran our model and the Isolation forest model provided by the sklearn library. On each graph you can see the results we got using the following outlier sets: true outliers coming from the labeled data (True_outliers), outliers produced by our model (SIF_v2) and outliers produced by IF from sklearn (IF_sklearn).

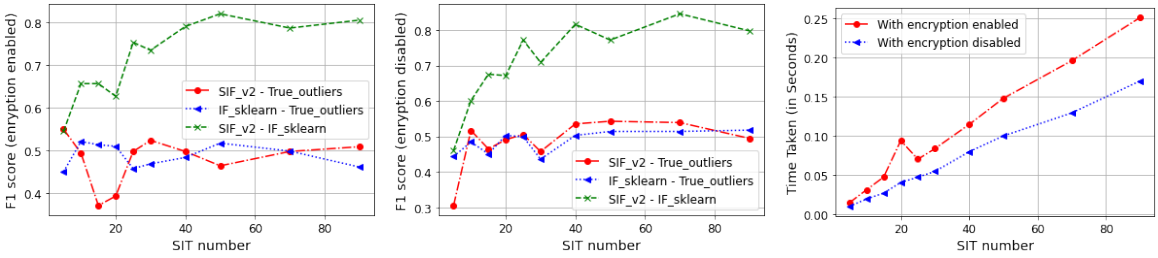


Figure 7.1: SIT number experiments on Pima dataset

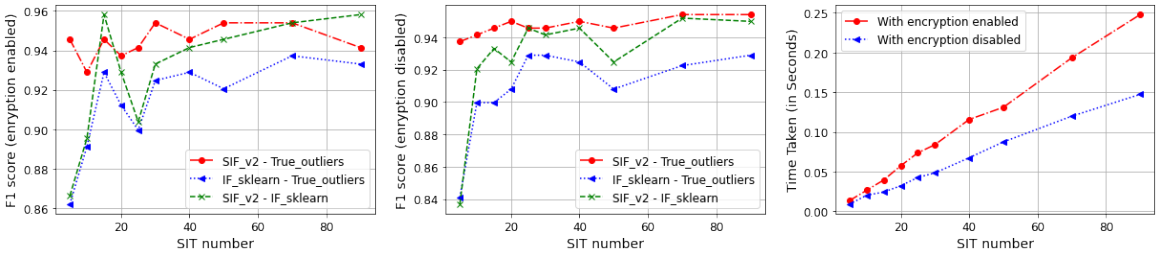


Figure 7.2: SIT number experiments on Breastw dataset

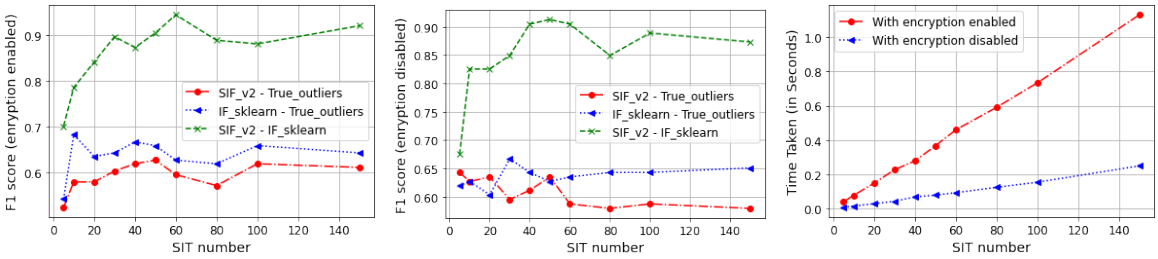


Figure 7.3: SIT number experiments on Ionosphere dataset

We can make the following observations from the produced results.

- Our model produces nearly the same results as the built-in isolation forest model from sklearn library.
- The cryptosystem integration does not worsen the metric values.
- Encryption and decryption procedures do not consume a lot of time. The time difference between the secure model and the version without encryption is not significant.

7.2.2 Optimal Parameter Values

It is essential to find the optimal parameter values for the secure anomaly detection algorithm. In this section, we will mainly focus on 3 parameters, which heavily affect the algorithm's accuracy and performance. These parameters are SIT number, maximum tree height and subsampling size for a single tree construction.

In the original model [1], the optimal values for these parameters were suggested as shown in Table 7.3.

Parameter	Optimal value
SIT number	100
Tree height	$\log_2$ data_size
Subsampling size	256 (for big datasets 128 and 512 are mentioned)

Table 7.3: Optimal parameter values from the original paper [1]

SIT Number

Investigating the SIT number parameter through multiple experiments, we observed the following:

- The optimal value for this parameter is not necessarily 100. The algorithm provides optimal results also for smaller values.
- We can conclude that for medium-size datasets (~1000 elements), 30 is usually the best choice for the algorithm.
- Increasing this parameter further does not provide significant improvement in the metric values.

	SIT-s	Height	AUC	Acc	P	R	F1
0	5	3	0.65	0.68	0.55	0.55	0.55
1	10	3	0.61	0.65	0.49	0.50	0.49
2	15	3	0.52	0.56	0.37	0.37	0.37
3	20	3	0.53	0.58	0.39	0.40	0.39
4	25	3	0.61	0.65	0.50	0.50	0.50
5	30	3	0.63	0.67	0.52	0.53	0.52
6	40	3	0.61	0.65	0.50	0.50	0.50
7	50	3	0.59	0.62	0.46	0.47	0.46
8	70	3	0.61	0.65	0.50	0.50	0.50
9	90	3	0.62	0.66	0.51	0.51	0.51

Table 7.4: Pima

	SIT-s	Height	AUC	Acc	P	R	F1
0	5	3	0.96	0.96	0.95	0.95	0.95
1	10	3	0.95	0.95	0.93	0.93	0.93
2	15	3	0.96	0.96	0.95	0.95	0.95
3	20	3	0.95	0.96	0.94	0.94	0.94
4	25	3	0.95	0.96	0.94	0.94	0.94
5	30	3	0.96	0.97	0.95	0.95	0.95
6	40	3	0.96	0.96	0.95	0.95	0.95
7	50	3	0.96	0.97	0.95	0.95	0.95
8	70	3	0.96	0.97	0.95	0.95	0.95
9	90	3	0.95	0.96	0.94	0.94	0.94

Table 7.5: Breastw

	SIT-s	Height	AUC	Acc	P	R	F1
0	5	6	0.63	0.66	0.52	0.52	0.52
1	10	6	0.67	0.70	0.58	0.58	0.58
2	20	6	0.67	0.70	0.58	0.58	0.58
3	30	6	0.69	0.72	0.60	0.60	0.60
4	40	6	0.70	0.73	0.62	0.62	0.62
5	50	6	0.71	0.73	0.63	0.63	0.63
6	60	6	0.68	0.71	0.60	0.60	0.60
7	80	6	0.67	0.69	0.57	0.57	0.57
8	100	6	0.70	0.73	0.62	0.62	0.62
9	150	6	0.70	0.72	0.61	0.61	0.61

Table 7.6: Ionosphere

Table 7.7: Experimental results on SIT number

Tree Height

Further, we experimented with different tree height values to find the optimal one.

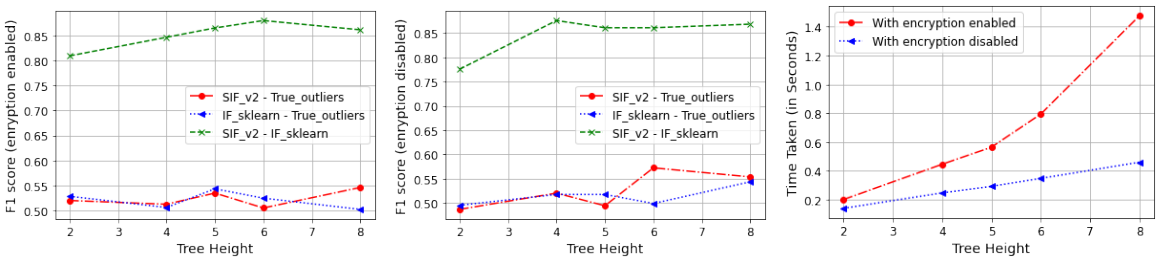


Figure 7.4: Tree height experiments on Pima dataset

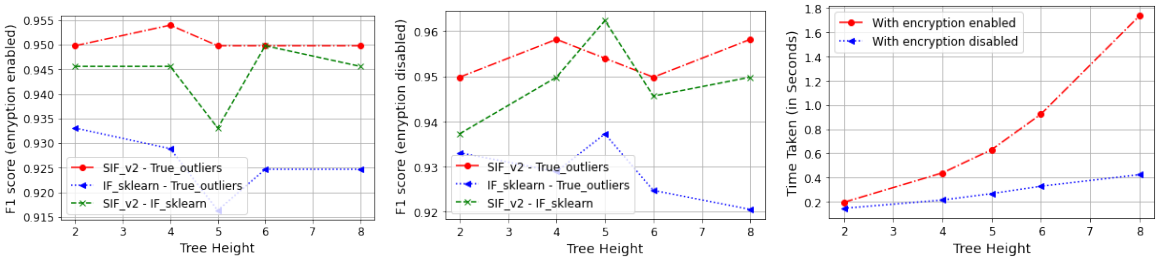


Figure 7.5: Tree height experiments on Breastw dataset

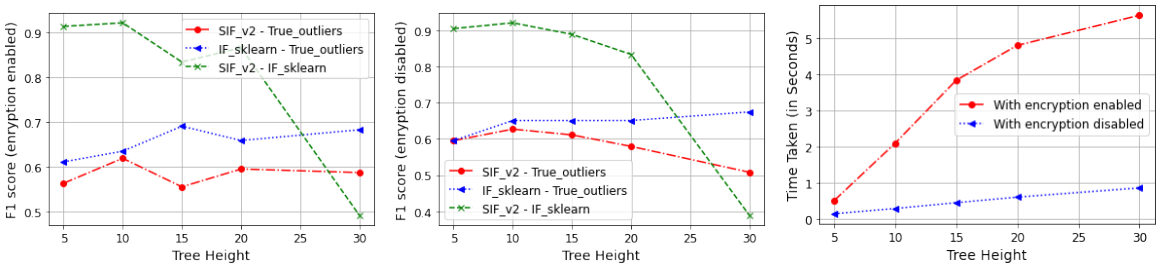


Figure 7.6: Tree height experiments on Ionosphere dataset

Investigating the tree height parameter through multiple experiments, we observed the following.

- We do not notice a significant difference among the results produced using different tree height values.
- The performance difference becomes much more noticeable compared to the experiments conducted on SIT number parameter. So increasing the SIT number has less effect on processing time than increasing the tree height for all the trees.
- As the optimal value for this parameter was suggested to be $\log_2 len(data)$, we can leave it like this, though keeping in mind that it can be reduced if needed.

	SIT-s	Height	AUC	Acc	P	R	F1
0	100	2	0.63	0.66	0.52	0.52	0.52
1	100	4	0.63	0.66	0.51	0.51	0.51
2	100	5	0.64	0.67	0.53	0.54	0.54
3	100	6	0.62	0.65	0.50	0.51	0.51
4	100	8	0.65	0.68	0.54	0.55	0.55

Table 7.8: Pima

	SIT-s	Height	AUC	Acc	P	R	F1
0	100	2	0.96	0.96	0.95	0.95	0.95
1	100	4	0.96	0.97	0.95	0.95	0.95
2	100	5	0.96	0.96	0.95	0.95	0.95
3	100	6	0.96	0.96	0.95	0.95	0.95
4	100	8	0.96	0.96	0.95	0.95	0.95

Table 7.9: Breastw

	SIT-s	Height	AUC	Acc	P	R	F1
0	100	5	0.66	0.69	0.56	0.56	0.56
1	100	10	0.70	0.73	0.62	0.62	0.62
2	100	15	0.65	0.68	0.56	0.56	0.56
3	100	20	0.68	0.71	0.60	0.60	0.60
4	100	30	0.68	0.70	0.59	0.59	0.59

Table 7.10: Ionosphere

Table 7.11: Experimental results on tree height

Tree Data Size

Another interesting set of experiments proved that the number of constructed trees and the data size for each tree construction are connected with each other. It was logical to think that if we increase the data which is considered for a single tree construction, then we would need less trees to achieve the same results and vice versa.

	Dataset	Data size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	500	50	3	0.65	0.68	0.55	0.55	0.55
1	pima	250	100	3	0.63	0.66	0.52	0.52	0.52
2	pima	100	250	3	0.65	0.68	0.54	0.55	0.55
3	pima	50	500	3	0.65	0.68	0.54	0.54	0.54
4	pima	20	650	3	0.65	0.68	0.54	0.54	0.54
5	breastw	500	50	3	0.95	0.96	0.94	0.94	0.94
6	breastw	250	100	3	0.96	0.97	0.95	0.95	0.95
7	breastw	100	250	3	0.96	0.96	0.95	0.95	0.95
8	breastw	50	500	3	0.96	0.97	0.95	0.95	0.95
9	breastw	20	650	3	0.96	0.96	0.95	0.95	0.95
10	ionosphere	250	100	10	0.68	0.70	0.59	0.59	0.59
11	ionosphere	100	250	10	0.66	0.69	0.56	0.56	0.56
12	ionosphere	50	500	10	0.66	0.69	0.56	0.56	0.56
13	ionosphere	20	650	3	0.67	0.69	0.57	0.57	0.57

Table 7.12: Dependency between SIT number and subsampling size

These experiments showed that as long as the data sampling size is not too high and not too low, choosing the right number of trees, we can achieve the same optimal results. The suggestion made in the original paper [1] to work with 100 trees and 256 sampling size might not be the best solution for your case. Depending on the data, it might be beneficial to adjust these parameter values, and it might result in better performance.

7.2.3 Multiple Split Attributes

Several experiments were conducted to examine whether the algorithm can be improved using multiple split attributes. In order to choose multiple split attributes, we generate vectors of columns length and randomly choose the vector values. We experimented with vectors which were drawn from Dirichlet distribution [1], with vectors the elements of which were randomly chosen from set 0, 1 with some probability. Through all these experiments, the other parameter values were fixed. We used 256 as subsampling size, 100 SIT-s, and max tree height as $\log_2 data\_size$.

	Dataset	Data size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	256	100	3	0.61	0.64	0.49	0.49	0.49
1	breastw	256	100	3	0.96	0.97	0.95	0.95	0.95
2	ionosphere	256	100	10	0.70	0.73	0.62	0.62	0.62
3	arrhythmia	256	100	10	0.60	0.80	0.32	0.32	0.32
4	annthyroid	256	100	3	0.52	0.87	0.12	0.12	0.12

Table 7.13: Split attribute vectors chosen from Dirichlet distribution

	Dataset	Data size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	256	100	3	0.60	0.64	0.48	0.49	0.48
1	breastw	256	100	3	0.96	0.96	0.95	0.95	0.95
2	ionosphere	256	100	10	0.70	0.72	0.61	0.61	0.61
3	arrhythmia	256	100	10	0.69	0.85	0.47	0.47	0.47
4	annthyroid	256	100	3	0.59	0.89	0.25	0.25	0.25

Table 7.14: 10% 1s

	Dataset	Data size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	256	100	3	0.63	0.66	0.51	0.51	0.51
1	breastw	256	100	3	0.95	0.96	0.94	0.94	0.94
2	ionosphere	256	100	10	0.73	0.75	0.65	0.65	0.65
3	arrhythmia	256	100	10	0.68	0.84	0.45	0.45	0.45
4	annthyroid	256	100	3	0.62	0.90	0.30	0.30	0.30

Table 7.15: 50% 1s

	Dataset	Data size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	256	100	3	0.59	0.62	0.46	0.47	0.46
1	breastw	256	100	3	0.96	0.97	0.95	0.95	0.95
2	ionosphere	256	100	10	0.67	0.70	0.58	0.58	0.58
3	arrhythmia	256	100	10	0.58	0.79	0.29	0.29	0.29
4	annthyroid	256	100	3	0.50	0.86	0.07	0.07	0.07

Table 7.16: 90% 1s

Table 7.17: Split attribute vectors chosen randomly from 0, 1 with some probability

The results do not show significant improvements over the initial model for most of the datasets. The reason for this could be the data structure, which should be previously examined. It might happen that the data can be easily split using vertical and horizontal

[1] Dirichlet distribution, https://en.wikipedia.org/wiki/Dirichlet_distribution

splitting lines, but instead we use diagonal ones, which complicate the process without significant benefits. Nevertheless, we notice that for high-dimensional and large datasets, this change resulted in much better F1 score.

7.2.4 Contamination Parameter

One major challenge in the models based on the isolation forest is that we need to provide the contamination parameter, which denotes the percentage of outliers in the data. Having only the anomaly scores, we can not tell whether an instance is anomalous or not, as we don't know the threshold to discriminate between normal and anomalous instances. This problem might be relevant in other models as well, which rely on the contamination parameter. In order to find this value, we might need specialists to manually explore the data and based on their knowledge predict the percentage of outliers. But obviously this might be costly, not efficient and not accurate. Thus, we will try to find the number of outliers based on the anomaly scores assigned to the instances during the training stage.

We conducted several experiments in order to find a way to detect the contamination parameter. The experiments were carried out on the primary medium category datasets.

The first thought was to observe the anomaly scores assigned to the instances. The idea was that there should be a big gap between the anomalous and normal scores. As the anomaly scores are based on the terminal node's subtree size and height, we expected the outliers to have very low scores compared to the normal instances, and this difference should be clearly visible in the score graphs. But surprisingly, this was not the case for all the datasets, as you can see below.

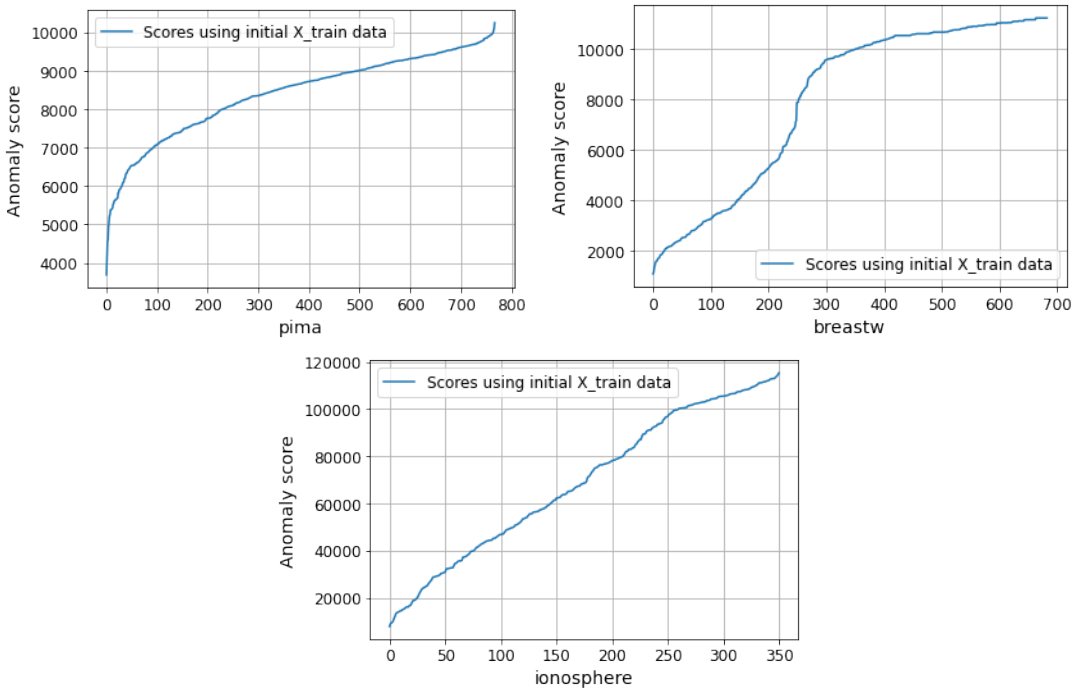


Figure 7.7: Anomaly scores

For one of the datasets (breastw) we can clearly see that starting from some point (after ~230), the anomaly scores start increasing very quickly and this can be an indicator of how many outliers there are. This was not the case for all the datasets, and one reason for this could be the data structure. What is also interesting to note is that for the dataset for which we can clearly see the threshold, the F1 score acquired through our model is quite high (~95%) compared to the results on other datasets (~55%).

	Dataset	Data Size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	256	100	3	0.68	0.68	0.54	0.54	0.54
1	breastw	256	100	3	0.96	0.96	0.95	0.95	0.95
2	ionosphere	256	100	5	0.71	0.71	0.60	0.60	0.60

Table 7.18: Experimental results for contamination parameter evaluation

As the next step, we tried to improve the algorithm so that the gap between the scores of anomalous and normal instances would be better visible. The flow of the adjusted algorithm is the following:

1. Run the anomaly detector on the initial dataset.
2. Separate ~30% of the data having the highest anomaly scores. These instances will definitely be normal instances, as we suppose anomalies are rare. If more than half the data is anomalous, then there is a problem with the data itself, and we do not consider this case.
3. Generate synthetic data of the training dataset length using the normal instances from the previous step.
4. Generate isolation trees using the initial data merged with the synthetic data.
5. Repeat steps 3-4 several times (in our experiment 10).
6. Run anomaly detector using all the generated trees.

This way, we want to emphasize where the normal instances are, thus increasing the scores for non-anomalous instances. The scores of the outliers should remain the same. Consequently, we thought that this way the difference between the scores would get more noticeable and would help us in the graph analysis.

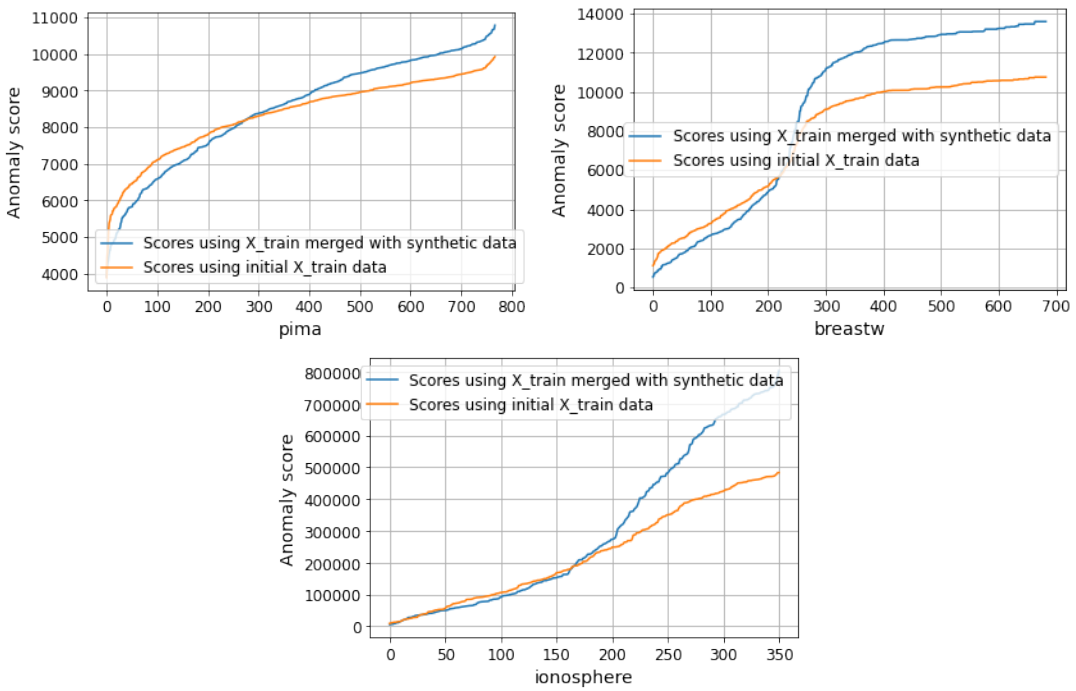


Figure 7.8: Anomaly scores for detecting contamination parameter

This worked out quite well, as looking at the graphs we can detect the contamination parameter. We can see that starting from some instance (which turns out to be the number of anomalies), there is a big gap in the anomaly scores generated from these 2 experiments.

Below, you can also see the evaluation metrics for the case when training data was merged with the synthetic data.

	Dataset	Data Size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	256	100	3	0.68	0.68	0.54	0.55	0.55
1	breastw	256	100	3	0.96	0.96	0.95	0.95	0.95
2	ionosphere	256	100	8	0.71	0.71	0.60	0.60	0.60

Table 7.19: Experimental results for contamination parameter evaluation (initial data merged with synthetic data)

We can conclude that in order to detect the number of outliers, we need to find the point where the gap between these 2 graphs becomes significantly big. One graph should be formed from the anomaly scores which we got from training the model on the data we had initially, and the second formed from the adjusted model. It is important to mention that for some datasets the results are not very consistent, but this can be caused by the data structure. And this can also be the reason why we do not get e.g. high F1 score. This theory still needs to be checked on large or high-dimensional datasets to make sure it can be used universally in order to find the contamination parameter.

7.2.5 Synthetic Data Usage

It was interesting to observe the model's behavior in the case when we had many nodes. The first experiment was conducted with 200 nodes on the primary medium datasets. Thus, each node would construct isolation trees based on very few instances and as expected, it produced non-optimal results. As trees considered only a small set of instances and not the general data properties, obviously the model could not produce optimal results.

	Dataset	Data Size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	256	20	3	0.39	0.39	0.13	0.13	0.13
1	breastw	256	20	3	0.95	0.95	0.92	0.92	0.92
2	ionosphere	256	20	5	0.61	0.61	0.46	0.46	0.46

Table 7.20: Experimental results with 200 nodes

To improve the algorithm's behavior in this case, we decided to use synthetic data. Initially, synthetic data (~10% of dataset length) was generated using all the data and then shared with all the nodes. So, the nodes generated the isolation trees using their chunk of data merged with the synthetic one. And as we can see from the experiment, we had a big improvement in the evaluation scores.

	Dataset	Data Size	SIT-s	Height	AUC	Acc	P	R	F1
0	pima	256	20	3	0.67	0.67	0.52	0.53	0.52
1	breastw	256	20	3	0.96	0.96	0.95	0.95	0.95
2	ionosphere	256	20	5	0.68	0.68	0.56	0.56	0.56

Table 7.21: Experimental results with 200 nodes using synthetic data

One interesting aspect to consider here is that in a real life setting, we would not have all the training data in one place to generate the synthetic data. So the question is how or when to produce it. One possible solution could be that the nodes having their small chunks of data construct synthetic data, encrypt and send to the anomaly detector (we can also have another component for this). Then the anomaly detector would decrypt data, merge together and generate synthetic data. And as there is no security risk involved here anymore, this data will then be directly shared with the nodes.

7.2.6 Large and high-dim Datasets

We observed that the previously mentioned optimal parameter values were the same also for high-dimensional or large datasets. Although with slightly increasing the SIT number, we sometimes got better AUC score, the difference is ~0.05%. But it is important to note that several other metric values are quite low, especially the F1 score. This is also the case when using the original isolation forest model from sklearn.

	Dataset	Data Size	SIT-s	Height	AUC	Acc	P	R	F1
0	annthyroid	256	100	3	0.87	0.87	0.15	0.15	0.15
1	arrhythmia	256	100	10	0.77	0.77	0.21	0.21	0.21

Table 7.22: Experimental results on large and high-dim datasets

7.2.7 Running Time

As described in the proposed solution (§5), we deployed the solution on AWS. The goal was to create the whole working flow and analyze the running times when varying the memory parameter. The memory configuration for encryption and decryption lambdas was fixed at 1028mb. You can find the results below.

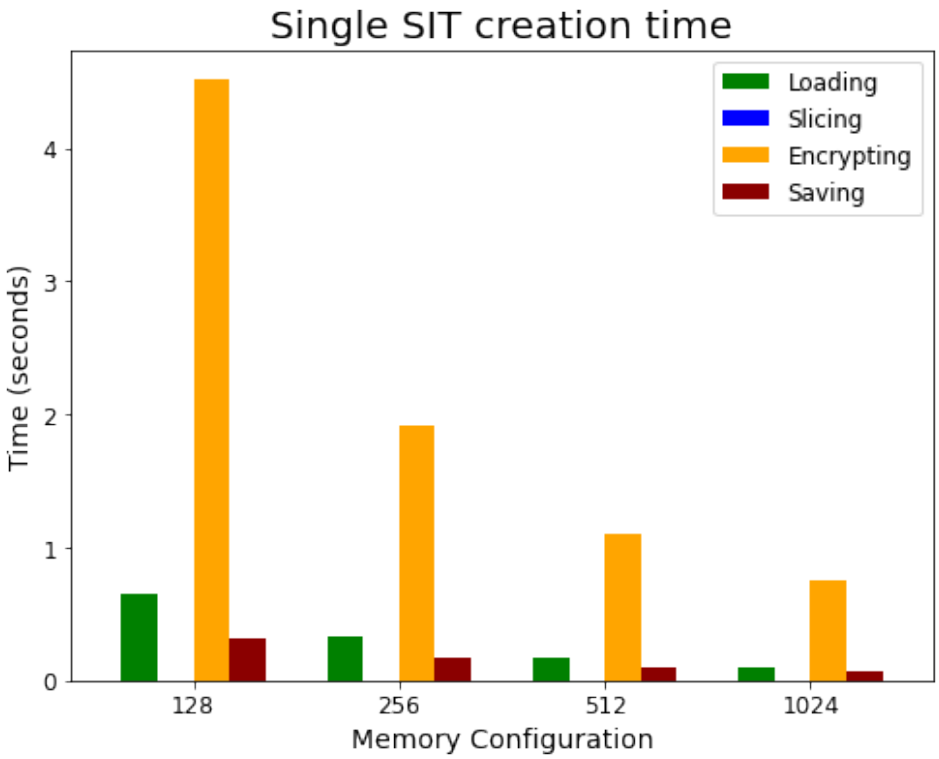


Figure 7.9: Single tree creation running time on AWS

Loading time is the time spent on reading the data chunk from the S3 bucket. Slicing time is the period spent on choosing a random subset of the data chunk which would be used to construct a single tree. Encrypting time is the overall time spent on the encryption during the secure tree creation. Saving is the time spent on pickling the SIF and saving it into the S3 bucket.

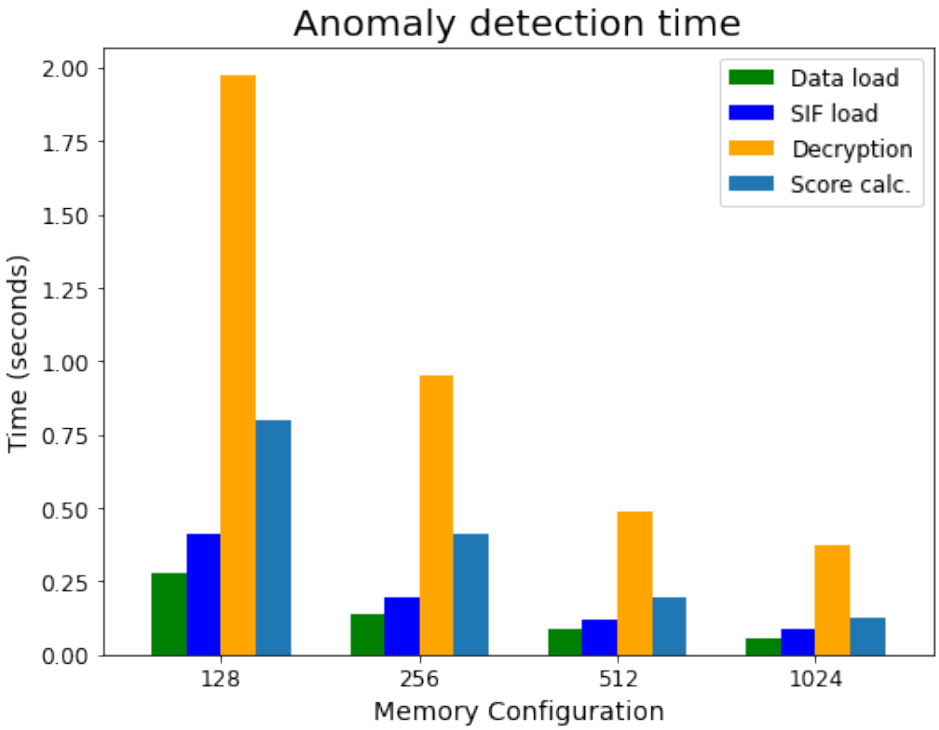


Figure 7.10: Anomaly detection time on AWS using one SIT

Data load is the time spent to load all the instances from S3 bucket. SIF load is the time spent to load saved trees from the S3 bucket. Decryption time is the total time spent on value decryption needed for score calculation. Score calculation is the time spent on score calculation without the decryption part.

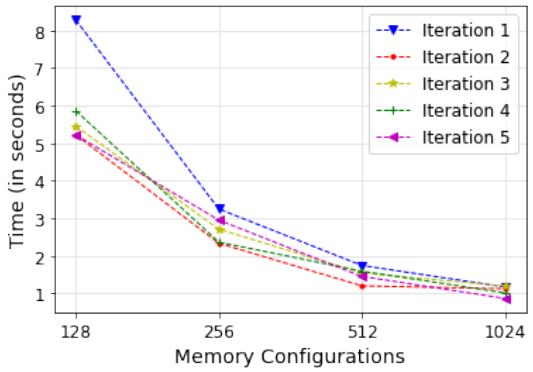


Figure 7.11: Single SIT creation

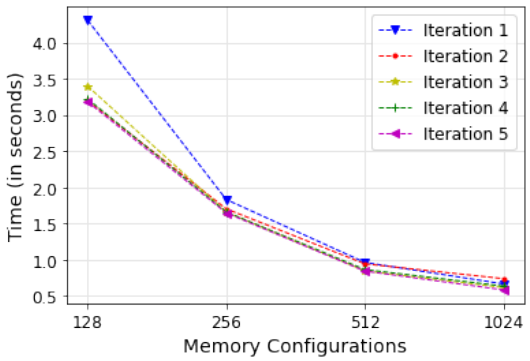


Figure 7.12: Anomaly detection

Figure 7.13: Running time for 5 iterations (tests for cold start)

As we can see, most time is spent on the encryption and decryption processes. This is mainly because of the API calls. One way to improve this situation, is to invoke the encryption and decryption methods directly from the lambdas (e.g. using Boto3) or make a bulk call to encrypt/decrypt multiple values in one call and reduce traffic time. We can see from the metrics provided by AWS that the average execution duration for the encrypt/decrypt function is 200 milliseconds. We can also notice that it is important to consider cold starts to improve the performance. Even only for 5 iterations, the improvement is quite noticable.

7.2.8 BCP cryptosystem

As mentioned before, initially we were using the BCP cryptosystem in order to ensure privacy preservation. Below, you can find the experimental results on a specific dataset. Again, the results are provided using the outlier sets from the labeled data (true outliers), outliers which we got using the IF library from sklearn (lib outliers) and the outliers from our model (encr outliers). As we can see, the cryptosystem didn't affect the accuracy of the algorithm, but the running time was rather high (it would take several minutes). That was the reason that we switched to a built-in cryptography module, which would provide us the efficiency we needed.

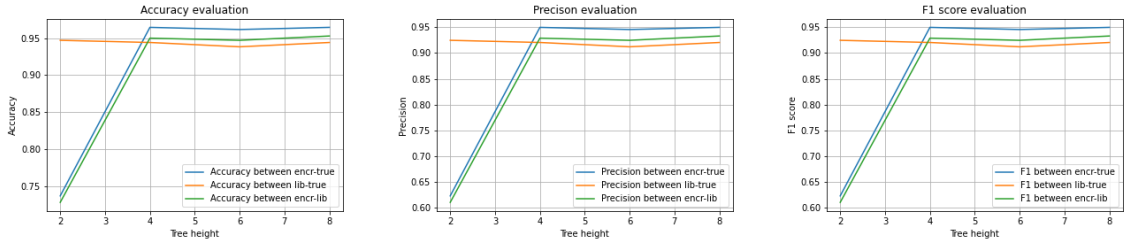


Figure 7.14: Tree height experimental results with BCP crypto for Breastw dataset

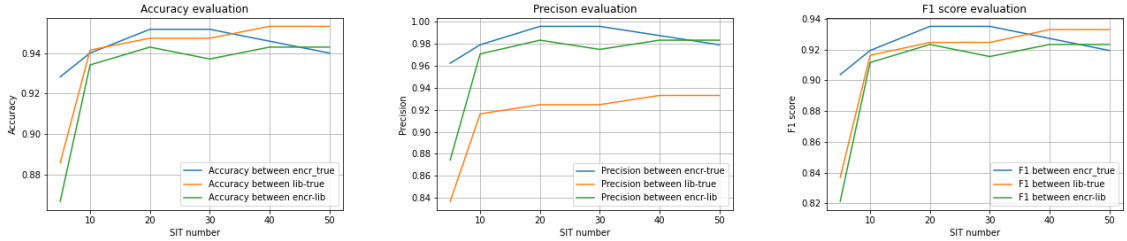


Figure 7.15: SIT number experimental results with BCP crypto for Breastw dataset

8 Discussion

8.1 Solution Optimality

From the experimental results, we can conclude that the secure anomaly detection model developed in this work is successful when configured properly. One important step in the model preparation process is the optimal parameter choice. We have specified the optimal values for several parameters based on experiments conducted on different datasets and believe that they will be optimal also on other data which has similar characteristics (e.g. approximately the same percentage of outliers, number of features, etc.). Yet, there is no exact equation for finding the optimal parameters for a dataset with completely different properties.

We have seen that the initial model reacts well to several improvements, but might not show positive change to others. For example, using multiple split attributes in order to improve accuracy sounds reasonable, but the experiments showed that with the chosen datasets there is no noticeable change. This can be caused by the data structure, which might be important to investigate before the analysis. Another improvement, which turned to be a success, was using synthetic data to overcome the scenario when we have numerous nodes and trees are constructed using small chunks of data. The experiments showed that synthetic data usage proves to be helpful. It is worth noting, that as a result of this improvement, general information about the data is shared in the process. In our setting, we consider this not to be problematic, otherwise we would have to come up with a different approach.

A valuable achievement of this work is the automatic detection of the contamination parameter. One problem is that the experiments do not provide consistent results, thus can not be reliable. Still, this can be a good starting point to understanding how to tackle this problem. If later a consistent process is defined, it will mean that no expert knowledge or previous labeling will be required beforehand.

Another important observation is that this solution can be used in real-time setting as well. The anomaly detector does not depend on the number of nodes or trees. So if constantly new trees are being created based on the changing data, sooner or later the detection algorithm will start considering these changes and adapt to them.

A key challenge mentioned in the problem statement was the privacy preservation during the anomaly detection process. In order to achieve that, we utilized a cryptosystem which ensured sensitive data protection. In our main model, we used the Fernet module of the

cryptography library, which is an implementation of symmetric authenticated cryptography.

8.2 Deployment

In §5 we described how the solution is deployed. We experimented with AWS and took advantage of the FaaS platform. As we used mostly standard services provided by AWS, they should exist on other cloud providers as well, thus replicating this should not be problematic. One major aspect to keep in mind when deploying the model is the issue of permissions and roles. It is vital to restrict access to important resources and follow the principle of the least privilege, otherwise we might encounter privacy issues.

8.3 Performance

A single secure isolation tree can be generated in about 2-6 seconds (might vary because of cold start and memory configuration). This seems to be efficient, especially if we suppose that each node will have to construct not a huge number of trees (depends on the data, number of nodes, etc.). Still, we might want to reduce the running time if possible. We notice that the biggest proportion of the time is spent on encrypting. One reason for this is that in the process of SIT creation, we make several calls to the encryption API. This results in a lot of traffic and can be improved by using bulk calls to the API. Another possible workaround would be the direct invocation of the required functionality (e.g. invoking the lambda during the SIT creation process), thus reducing the traffic time. Similar improvements can be made for the anomaly detection process, which depends on the calls made to the decryption API.

9 Conclusion

Analyzing the results, we can conclude that the proposed model is suitable for solving the secure anomaly detection task on distributed sensitive data. It achieves optimal results in the anomaly detection process as well as does not leak private information. The proposed improvements focus on specific disadvantages of the base algorithm and suggest ways of overcoming those. Several results we got are not bound only to this model, but can prove to be helpful in many other scenarios as well.

9.1 Current Use Cases

The proposed model can be used in multiple scenarios, where we have distributed data sources and privacy preservation is required. The cryptosystem used in the model can be changed to satisfy certain needs and preferences, which makes the model very flexible. As the model is based on the idea of isolation trees, it holds the benefits of the base model, such as the ability to work well with high-dimensional or large datasets. Thus, this solution is worthy to consider when dealing with secure anomaly detection task.

9.2 Further Work

9.2.1 Reliable Detection of Contamination Percentage

As mentioned in the discussion, one important achievement of this work is the automatic detection of the contamination parameter. But as the results are not consistent, it might be important to further investigate the reasons and improve the defined process. One idea would be to experiment with other anomaly score calculation equations. We used a single fixed formula for this, but it might happen that slightly changing this formula we will get consistent results to reliably find the contamination percentage.

9.2.2 Use Cases for Multiple Split Attributes

Using multiple split attributes for a secure isolation tree construction seems to be a reasonable improvement over the initial model, but we need to understand when exactly it makes sense to do so. A new preprocessing step might be useful to setup, which will investigate the data

structure and determine whether it will be beneficial to include this improvement in a specific case.

9.2.3 Privacy Preserving Techniques

It will be interesting to observe the model behavior when other cryptosystems are used. Besides cryptosystems, we might want to also experiment with other privacy-preserving techniques mentioned in the related work. It might prove to be beneficial to construct a guide map to suggest a privacy-preserving technique considering all the specifics of a given scenario. Whether the model should use a cryptosystem or data transformations or synthetic data, entirely depends on the given problem. Isolation forest is a flexible and straightforward model and can be combined with different security preserving techniques. And the main question is how to choose the right methodology.

9.2.4 F1 score improvement

We notice from the experiments, that although the model provides optimal values for several metrics, such as Accuracy, AUC score, etc., the F1 score is quite low sometimes. It is important to investigate the reasons and conduct necessary improvements. One observation that might be a good starting point for the investigation is noticing that using multiple split attributes for large and high-dimensional datasets improved the F1 score. Thus, analyzing the data structure and properties beforehand is a possible step to consider.

List of Figures

List of Tables

Bibliography

[1] F. T. Liu, K. M. Ting, and Z.-H. Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[2] J. Kuhlenkamp, S. Werner, and S. Tai. "The Ifs and Buts of Less is More: A Serverless Computing Reality Check". In: *2020 IEEE International Conference on Cloud Engineering (IC2E)*. 2020, pp. 154–161. DOI: 10.1109/IC2E48712.2020.00023.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[4] B. Tang and H. He. *A Local Density-Based Approach for Local Outlier Detection*. 2016. arXiv: 1606.08538 [cs.AI].

[5] Z. He, X. Xu, and S. Deng. "Discovering cluster-based local outliers". In: *Pattern Recognition Letters* 24.9 (2003), pp. 1641–1650. ISSN: 0167-8655. DOI: https://doi.org/10.1016/S0167-8655(03)00003-5. URL: https://www.sciencedirect.com/science/article/pii/S0167865503000035.

[6] D. R. Patrikar and M. R. Parate. *Anomaly Detection using Edge Computing in Video Surveillance System: Review*. 2021. arXiv: 2107.02778 [cs.CV].

[7] R. Mayer, M. Hittmeir, and A. Ekelhart. "Privacy-Preserving Anomaly Detection Using Synthetic Data". In: *Data and Applications Security and Privacy XXXIV*. Ed. by A. Singhal and J. Vaidya. Cham: Springer International Publishing, 2020, pp. 195–207. ISBN: 978-3-030-49669-2.

[8] R. Okada, K. Fukuchi, K. Kakizaki, and J. Sakuma. *Differentially Private Analysis of Outliers*. 2015. arXiv: 1507.06763 [stat.ML].

[9] E. Bresson, D. Catalano, and D. Pointcheval. "A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications". In: *Advances in Cryptology - ASIACRYPT 2003*. Ed. by C.-S. Laih. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 37–54. ISBN: 978-3-540-40061-5.

[10] K. Bhaduri, M. D. Stefanski, and A. N. Srivastava. "Privacy-Preserving Outlier Detection Through Random Nonlinear Data Distortion". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 41.1 (2011), pp. 260–272. DOI: 10.1109/TSMCB.2010.2051540.

[11] K. Liu, H. Kargupta, and J. Ryan. "Random projection-based multiplicative data perturbation for privacy preserving distributed data mining". In: *IEEE Transactions on Knowledge and Data Engineering* 18.1 (2006), pp. 92–106. DOI: 10.1109/TKDE.2006.14.

[12] A. Blum, K. Ligett, and A. Roth. *A Learning Theory Approach to Non-Interactive Database Privacy*. 2011. arXiv: 1109.2229 [cs.DS].

[13] L. A. R. L. Rivest A. Shamir. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[14] D. Hawkins. *Identification of Outliers*.

[15] R. T. N. Edwin M. Knox. *Algorithms for Mining Distance-Based Outliers in Large Datasets*.

[16] C. C. Aggarwal. *Outlier analysis*.

[17] C. C. Aggarwal and P. S. Yu. "Outlier Detection for High Dimensional Data". In: *SIGMOD Rec.* 30.2 (May 2001), pp. 37–46. ISSN: 0163-5808. DOI: 10.1145/376284.375668. URL: https://doi.org/10.1145/376284.375668.

[18] S. Ramaswamy, R. Rastogi, and K. Shim. "Efficient Algorithms for Mining Outliers from Large Data Sets". In: *SIGMOD Rec.* 29.2 (May 2000), pp. 427–438. ISSN: 0163-5808. DOI: 10.1145/335191.335437. URL: https://doi.org/10.1145/335191.335437.

[19] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar. *Efficient GAN-Based Anomaly Detection*. 2019. arXiv: 1802.06222 [cs.LG].

[20] A. Jindal, M. Gerndt, M. Chadha, V. Podolskiy, and P. Chen. "Function delivery network: Extending serverless computing for heterogeneous platforms". In: *Software: Practice and Experience* n/a.n/a (). DOI: https://doi.org/10.1002/spe.2966. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2966. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2966.

[21] P. Benedetti, M. Femminella, G. Reali, and K. Steenhaut. "Experimental Analysis of the Application of Serverless Computing to IoT Platforms". In: *Sensors* 21.3 (2021). ISSN: 1424-8220. DOI: 10.3390/s21030928. URL: https://www.mdpi.com/1424-8220/21/3/928.

[22] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[23] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner. "Resource Provisioning for IoT Services in the Fog". In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. 2016, pp. 32–39. DOI: 10.1109/SOCA.2016.10.

[24] C. Liu, X. Su, and C. Li. "Edge Computing for Data Anomaly Detection of Multi-Sensors in Underground Mining". In: *Electronics* 10.3 (2021), p. 302.

[25] S. Goldwasser and S. Micali. "Probabilistic encryption". In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 270–299. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(84)90070-9. URL: https://www.sciencedirect.com/science/article/pii/0022000084900709.

[26] R. Cramer and V. Shoup. "Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption". In: *Advances in Cryptology — EUROCRYPT 2002*. Ed. by L. R. Knudsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 45–64. ISBN: 978-3-540-46035-0.

[27] P. Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology — EUROCRYPT '99*. Ed. by J. Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238. ISBN: 978-3-540-48910-8.

[28] C. Zhang, H. Liu, Y. Li, A. Yin, Z. L. Jiang, Q. Liao, and X. Wang. "A novel privacy-preserving distributed anomaly detection method". In: *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*. 2017, pp. 463–468. DOI: 10.1109/SPAC.2017.8304323.

[29] S. Hariri, M. C. Kind, and R. J. Brunner. "Extended Isolation Forest". In: *IEEE Transactions on Knowledge and Data Engineering* 33.4 (Apr. 2021), pp. 1479–1489. ISSN: 2326-3865. DOI: 10.1109/tkde.2019.2947676. URL: http://dx.doi.org/10.1109/TKDE.2019.2947676.

[30] G. Lu, C. Duan, G. Zhou, X. Ding, and Y. Liu. "Privacy-Preserving Outlier Detection with High Efficiency over Distributed Datasets". In: *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 2021, pp. 1–10. DOI: 10.1109/INFOCOM42981.2021.9488710.

[31] M. Hossin and S. M.N. "A Review on Evaluation Metrics for Data Classification Evaluations". In: *International Journal of Data Mining & Knowledge Management Process* 5 (Mar. 2015), pp. 01–11. DOI: 10.5121/ijdkp.2015.5201.

[32] J. Huang and C. Ling. "Using AUC and accuracy in evaluating learning algorithms". In: *IEEE Transactions on Knowledge and Data Engineering* 17.3 (2005), pp. 299–310. DOI: 10.1109/TKDE.2005.50.