DFASC: Distributed Framework for Analytics Security in the Cloud

Mamadou H. Diallo, Christopher T. Graves, Michael August,
Kevin Groarke, Michael Holstrom, and Megan Kline

Naval Information Warfare Center Pacific, San Diego, CA USA
U.S. Department of Defense
Email: {mamadou.h.diallo, christopher.t.graves, michael.august,
kevin.groarke, michael.holstrom, megan.kline}@navy.mil

***Abstract*—Processing big data requires advanced technologies that can extract useful information from large scale data to support decision making. These advanced technologies are currently being offered in the form of analytic tools hosted in the cloud, and are being developed using different techniques such as artificial intelligence, machine learning, data mining, and statistical analysis. However, these tools are not very secure since the data they operate on must be in plaintext in the cloud, thereby leaving the data vulnerable to both insider and outsider attacks. To address these security issues when running data analytics in the cloud, we propose DFASC, a Distributed Framework for Analytics Security in the Cloud. At the core of the framework is homomorphic encryption (HE), which enables operations to be performed directly on encrypted data without using the private decryption key. Using HE, DFASC can distribute homomorphically encrypted data and analytics into the nodes of a distributed system and allow the analytics to operate on the encrypted data in each node. As a framework, DFASC provides mechanisms to enable the incorporation of HE libraries and data processing algorithms into the framework, which can than be used to implement analytic tools. A fundamental challenge with HE is its performance overhead due to the computationally intensive HE operations. This challenge of accelerating individual HE operations needs to be solved before secure big data processing in the cloud can be made practical. The distribution of the analytics not only improves the performance of the underlying analytic algorithms, it also helps to speed up the underlying HE operations. To enable the sharing of the encrypted data between parties in the cloud, DFASC incorporates a cryptographic key management infrastructure. To analyze feasibility of the framework, it was extended to implement a system that classifies images using a Neural Network algorithm. The experimental results show performance improvement of the system, including in HE operations, as the number of nodes in the cluster is increased.**

***Index Terms*—*Homomorphic Encryption*; *Cloud Computing*; *Privacy*; *Data Analytics*; *Data Sharing*.**

I. Introduction

Recent advances in communications and networking technology have revolutionized the way information systems are being developed and used. Cloud computing technology is a result of these advances and provides a computing paradigm with a large amount of computing power and storage resources. Advances in cryptography are also enabling the development of stronger security protocols for cloud-based systems [1]. The cloud computing market is growing at a rapid pace, and includes top cloud providers such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform. According to the Fortune Business Insight magazine, ”The global cloud computing market size stood at USD 199.01 billion in 2019 and is projected to reach USD 760.98 billion by 2027” [2]. These resources are being used to develop information systems for individuals and organizations such as social media platforms, collaborative environments, and Internet of Things (IoT) based systems. These information systems are generating increasingly larger amounts of data as they are being used by individuals and organizations, resulting in a tremendous amount of data. For instance, the number of Internet users was estimated to be 2.4 billion in 2014, 3.4 billion in 2016, 3.7 billion in 2017, and 4.4 billion in June 2019. These high numbers of Internet users are reflected in the amount of data being generated on the Internet. For example, in 2019, Google reported 300,000 billion searches conducted worldwide daily and FaceBook over 4.3 billion messages posted daily.

Performing data analytics in the cloud is becoming increasingly significant for organizations of all types and sizes. These analytics are based on techniques such as artificial intelligence, data mining, machine learning or statistical methods [3], [4], [5]. Organizations are taking advantage of these analytic tools to gain powerful insights out of the ever-growing pools of organizational data. These cloud based data analytic tools are being developed for various application domains [6], [7], [8], [9]. However, there is also a growing concern about data security and privacy in cloud-based systems and applications that provide analytic tools [10], [11], [12], [13]. In particular, cybersecurity attackers are becoming more sophisticated, and attacks on data in large organizations are occurring more frequently [14].

The current cybersecurity vulnerabilities in the cloud stem from the fact that data is not protected while being manipulated by analytic tools in the cloud, which is inherited from the shortcomings of current cryptographic techniques for securing data. The recommended randomized encryption schemes, such as the Advanced Encryption Standard (AES) and Blowfish, provide strong protection of data in transit and at rest, but do not protect data in processing. This means that data needs to be decrypted in memory before processing of the data can take place, which leaves the data vulnerable to attacks from both internal and external attackers.

There are many examples of both insider and outsider attacks on large organizations' information systems. The personal data of 77 million Sony users was leaked in 2011. Information from 38 million Adobe accounts was stolen in 2013. A major example of an insider attack was Edward Snowden's leakage of data from within the NSA in 2013. Data from 110 million Target customers was also hijacked in 2013. In 2015, US Office of Personal Management records for more than 21.5 million people were stolen by an outsider. Credit information on 143 million American, Canadian and British customers was stolen from Equifax in 2017. These attacks occurred because of a vulnerable attack surface within the information systems of these organizations.

Another challenge is how the data can be shared securely among parties in the cloud. For organizations outsourcing their data in the cloud, sharing the data is an important benefit. For instance, a medical system that manages patient health records can be deployed in the cloud and hospitals can collaboratively use the system to share the health records between them. This issue of data sharing in the cloud has been significantly studied and various techniques have been introduced. However, most of these techniques are based on PKI, which requires a Certificate Authority (CA) to manage the cryptographic keys. The CA is a third party entity and as such increases the complexity of building cloud systems. Deployment and maintenance of a PKI is complex and expensive, and certificate management can be challenging.

To address the shortcomings of existing standard cryptographic schemes, Homomorphic Encryption (HE) has been proposed [15]. HE schemes have revolutionized data security, as they enable computation to be performed directly on the encrypted data without needing the private decryption keys. Given ciphertexts as input, HE allows computation to be performed directly on the ciphertexts to generate encrypted results. When these encrypted results are then decrypted, they yield the correct plaintext answer for the computation as if it were performed entirely in plaintext. However, HE, while significantly improving data security in untrusted environments, comes with significant computation and storage overhead [16]. In general, the computational complexity of HE is orders of magnitude higher than that of standard operations on plaintext. A given ciphertext encoding is also much larger than its corresponding plaintext. Acceleration of HE is an active area of research, and great strides have been made to speed up the underlying HE operations. Note that more progress still needs to be made for HE to be practical for performing big data analysis. Even with its significant computational overhead, HE can still be practical today for certain types of application domains such as interactive applications [17].

In this paper, we introduce DFASC, a distributed framework for analytics security in the cloud. DFASC leverages HE to provide data security for cloud analytics not only in transit and at rest, but most importantly, when being processed. The framework is modularized and extensible to enable the incorporation of different types of HE schemes. The framework also provides mechanisms for incorporating data analytic tools that use HE schemes across the nodes of the distributed framework. This enables data analytic tools to operate directly on the encrypted data. To enable data sharing, the framework includes a cryptographic key management infrastructure based on the approach introduced in [18]. Using this approach, an organization can analyze data in the cloud and share the results with other organizations. Distributing analytic tool execution across the nodes of the framework speeds up the expensive operations of the HE schemes to improve the overall performance of the tools. The framework enables tool developers using various machine learning and data mining algorithms to use the framework to build analytic tools, in addition to enabling system developers to leverage these analytic tools within their applications. The secure systems developed based on the framework can then be made available to end-users to analyze their data securely and privately in the cloud. Having access to different analytics will enable end-users to trade off between the quality of the results of the data analysis and the time it takes to perform the analysis. Furthermore, end-users will have the ability to share their data with other parties securely and privately.

The paper is organized as follows. In Section II, we describe the challenges in processing data and our proposed solution. In Section III we describe our overall approach. In Section IV we present our approach for data sharing within the framework. In Section V we outline the data flow through the system. In Section VI we discuss our implementation of the framework and sample application. In Section VII we present the results of experiments performed on the system. In Section VIII we contrast our paper with related works. We end the paper with a conclusion and future work in Section IX.

II. Background

In this section, we take a look at how organizations make use of data analytics in the cloud and give an overview of HE, which can be used to provide data security in the cloud.

A. Data Analysis in the Cloud

Data analytics in the cloud, or cloud analytics, is a service model where data analytics are pushed to the cloud to take advantage of the cloud's resources. A hybrid model can also be used as a to allow analytics to be implemented partially on the client side and connected to cloud side to form an integral system, which can be scaled in the cloud as the need arises. In this way, a hybrid approach can be used to split the data analysis between the client and the server. These analytics are developed using techniques such as artificial intelligence, machine learning, data mining, and statistical modeling and analysis.

Due to the benefits they are providing to organizations of all types and sizes, cloud analytics are gaining popularity. They are steadily making their way into enterprise applications in the cloud in various areas, such as customer support, fraud detection, and business intelligence [6]. As organizations are becoming aware of these cloud analytics, the need for them is increasing. The major cloud service providers are responding to this need for tools that provide data analysis and business intelligence capabilities within the cloud by adding these features to their cloud services [19]. Thus, the trend of organizations outsourcing their Information Technology operations to the cloud, combined with the trend of cloud service providers adding more intelligence to their cloud services, indicates that increasingly organizations will make use of the cloud to analyze their large and potentially sensitive data sets.

However, the cloud is vulnerable to cyberattacks from both internal and external attackers, and running analytics in the cloud on organizationally sensitive data can result in loss of data security. In [20], an analysis of the security threats in big data analysis using MapReduce and Hadoop revealed the complexity of securing cloud analytics. One security challenge is related to the large amount of data to be processed, which makes current security techniques impractical because they are too slow to be effective [21]. Another security challenge is related to the various sources of data to be combined and processed in the cloud [22]. To address these vulnerabilities of the cloud, we use HE to ensure the confidentiality of the data that are collected, stored, and processed in the cloud.

B. Homomorphic Encryption Schemes

Homomorphic Encryption (HE) is a cryptographic scheme that enables operations to be performed directly on encrypted data without using decryption keys. The HE computations are represented as either Boolean or arithmetic circuits, which are characterized by their depth. There are different types of HE schemes based on the types of operations they support. *Partially Homomorphic Encryption (PHE)* schemes support only one type of operation, addition or multiplication, while *Fully Homomorphic Encryption (FHE)* schemes allow arbitrary computation on ciphertexts. Between these two extreme cases, there are schemes that are *Somewhat Homomorphic Encryption (SHE)*, which support both addition and multiplication but are limited in the number of operations (the depth of the circuit) that can be executed on ciphertexts.

The existence of fully homomorphic encryption was theorized in 1978 by Rivest [23]. However, it was only in 2009 that the first working *Fully Homomorphic Encryption* (FHE) scheme was constructed by Gentry [15]. Gentry's approach involves taking a *Somewhat Homomorphic Encryption* scheme and "squashing" the decryption circuit to reduce the noise in a process called "bootstrapping" to get the *Fully Homomorphic Encryption*. The security of his scheme assumes the hardness of two problems: certain worst-case problems over ideal lattices, and the sparse subset sum problem. However, this process was impractical due to the required computation time. Since then, a number of more practical FHE schemes have been proposed.

a) FHE Scheme Based on Ring Learning With Errors: Most of the FHE schemes that have been proposed base their security on the hardness of the (Ring) Learning With Errors (RLWE) problem [24]. The RLWE problem has been proven to provide a strong security guarantee while supporting more practical FHE schemes. Before describing some of these FHE schemes, let us first define the RLWE problem.

Definition of RLWE: let $n = 2^k$ and choose a prime modulus q such that $q \equiv 1 \mod 2n$. Let the ring $R_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, represent the set of all the polynomials over the finite field $\mathbb{Z}_q$ for which $x^n \equiv -1$. Given samples of the form $(\mathbf{a}, \mathbf{b} = \mathbf{a} \times \mathbf{s} + \mathbf{e}) \in R_q \times R_q$ where $\mathbf{s} \in R_q$ is a fixed secret vector, an element $\mathbf{a} \in R_q$ is chosen uniformly, and $\mathbf{e}$ is chosen randomly from an error distribution in R_q. Given this definition of the RLWE problem, finding s is infeasible.

Using the RLWE problem, a message $m \in R_q$ can be encrypted by using the b element above as a one-time pad encryption scheme [25]. The ciphertext can be represented by $\mathbf{c} = \mathbf{b} + \mathbf{m}$, where $\mathbf{c} \in R_q$.

BGV Scheme Brakerski, Gentry, and Vaikuntanathan proposed a Leveled FHE scheme based on the RLWE problem and referred to the scheme as BGV [26]. It is referred to as "leveled" due to the fact that its parameters depend (polynomially) on the depth of the circuits that it is capable of evaluating. "Leveled" FHE means that the size of the public key is linear in the depth of the circuits that the scheme can evaluate, that is, its size is not constant. The key operation in the scheme is the *REFRESH* procedure, which switches the moduli of the lattice structure and switches the key.

C. Homomorphic Encryption Libraries

Following the constructions of the FHE schemes, software libraries implementing the schemes are being developed. The first of such libraries to be implemented is HElib, first released in 2012 [27]. This first version of HElib implemented only the Brakerski-Gentry-Vaikuntanathan (BGV) scheme, but the latest version now includes the approximate FHE scheme proposed by Cheon, Kim, Kim and Song (CKKS) [28], the newest FHE scheme. Following HElib, a number of other FHE development efforts have been launched as presented in the survey of the current FHE libraries in [29]. Here we limit our discussion to PALISADE [30] and Microsoft's Simple Encrypted Arithmetic Library (SEAL) [31].

The PALISADE library is being developed under an open-source project that provides efficient implementations of lattice cryptography building blocks and leading homomorphic encryption schemes. PALISADE is designed for usability, providing simpler APIs, modularity and cross-platform support. The current version of PALISADE supports the BGV, Brakerski-Fan-Vercauteren (BFV) [32], CKKS, and FHEW schemes and a more secure variant of the TFHE scheme, including bootstrapping [30]. The Microsoft SEAL is also an open-source library that provides an efficient implementation of lattice cryptography using leading homomorphic encryption schemes. The current version of SEAL also supports the BGV, BFV, and CKKS schemes. The proposed DFASC framework integrates these two libraries: PALISADE and SEAL.

III. Framework

In this section, we describe the architecture of the proposed DFASC framework, the integration of HE libraries and data processing algorithms into the framework, the data sharing protocol used to enable clients to share encrypted data, and the threat model of the framework.

A. Architecture

The DFASC framework is designed using a hybrid client-server/distributed model, where clients send requests to a remote server, and the server forwards the client's requests to a distributed system for processing. The results of the data processing are returned to the remote server for storage and to be made available to the clients. The high-level design of the framework is presented in Figure 1. The architecture is composed of two main components: *Trusted Client* and *Untrusted Cloud Environment*.

The *Trusted Client* comprises three main sub components, *Client Manager*, *HE Manager*, and *Configurations Manager*. The *Client Manager* coordinates the activities of the client and manages the interactions with the server. The *HE Manager* provides support for HE operations including generation and storage of public and private keys, encryption and decryption of data, and keys revocation. The *Configurations Manager* keeps track of the cloud resources for the clients, which change dynamically as the system is being used. Note that system developers will need to extend the framework to build concrete systems for specific application domains. In addition to the above core components, system developers need to implement a user interface for end-users to interact with the system.

The *Untrusted Cloud Environment* is composed of an *Untrusted Server* and an *Untrusted Distributed System*. All the data sets sent by the clients to the *Untrusted Cloud Environment* will remain encrypted at all times. The sub-components of the *Untrusted Server* include a *Service Engine* for coordinating all the activities related to distributing data and operations into the *Untrusted Distributed System*; an *HE Manager* for managing HE libraries stored in the *HE Libraries* storage; an *Analytics Manager* for managing the analytic algorithms persisted in the *Libraries* storage; a *Sharing Manager* for sharing encrypted data between the clients; and a *Configurations* storage for storing various cloud configurations and metadata. The *Service Engine* communicates with the *Untrusted Distributed System* to coordinate its activities, including sending workloads and partitioning the nodes within the cluster.

The *Untrusted Distributed System* provides the infrastructure for distributing analytics algorithms. The inputs to the *Untrusted Distributed System* include the set of data to be processed and the software program to be executed on the nodes of the distributed system that will process the data. At the core of the distributed system is a *Distribution Manager*, which provides the mechanisms for generating the clusters of distributed nodes. The nodes are generated by the *Distribution Manager* on demand based on the configurations provided by developers. In addition, the *Untrusted Distributed System* provides an interface to enable interaction with other distributed systems.

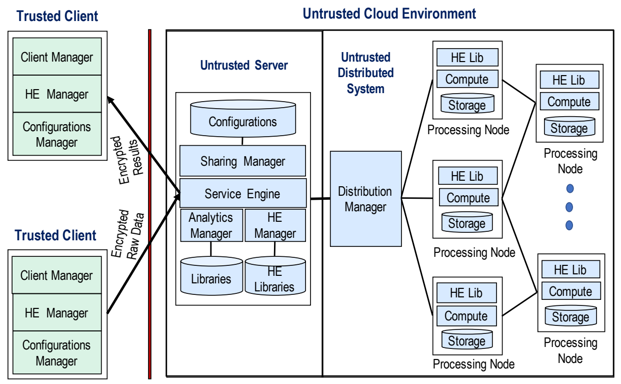

Fig. 1: Distributed Framework Architecture

B. HE Library Integration

At the core of DFASC is the mechanism for incorporating HE libraries into the framework. Like the standard cryptographic algorithms, homomorphic encryption algorithms have a well-defined set of operations. These operations include key generation, encryption, decryption, and ciphertext operations. To accommodate various FHE libraries, DFASC abstracts out the common core operations of HE libraries and builds those operations into the framework. It adopts a parameterization approach to enable each library to provide all necessary parameters to execute the operations. At a low-level of the implementation, a binary operation takes as inputs two integers A and B, and returns the result as an integer C. These operations are abstracted out into an interface that can then be used to integrate a given HE library. As part of the framework, we integrated the PALISADE and SEAL libraries.

C. Data Processing Algorithm Integration

DFASC provides an extensible interface to enable developers to extend or customize DFASC to add new machine learning and data mining algorithms into the framework. Considering the complexity of using existing HE libraries, the first machine learning algorithm we considered for the DFASC framework is the linear Support Vector Machine (SVM). The second machine learning algorithm we implemented within the framework was an artificial neural network. In the future, we plan on adding more machine learning algorithms into the framework.

1) Support Vector Machines: SVMs are supervised learning models that can be used to analyze data based on classification and regression analysis. The SVM serves as a non-probabilistic binary linear classifier.

Consider a set S of sample data elements, and two subsets S_A and S_B of S, where $S_A \cup S_B = S$, and each element of S ($S_1 \in S$) is annotated as belonging to S_A or S_B. The SVM training algorithm generates a mathematical model that can be used to categorize new elements of S as belonging to S_A or S_B.

First, we are given a labeled training dataset of n points of the form $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$. This training dataset contains both the inputs and the desired outputs. Given the training dataset, we then compute the SVM model to be used for classification. This model then separates the elements of S into two classes, S_A and S_B, based on the classifier that was generated from the training data. The internal operations of the linear SVM include the dot product of vectors, addition, and subtraction. To demonstrate the utility of the DFASC framework, we implemented an SVM classifier on top of our distributed framework using the PALISADE library.

2) Neural Networks: Neural networks (NN) are a learning mechanism that model the biological brain. They consist of a set of transformations of a signal vector throughout a graph of nodes. Each node, called a neuron, is connected to the next layer of neurons via edges, called links. Each link has a weight associated with it. Each neuron processes its input signal as a linear combination of the weights of its input neurons according to an activation function and produces an output signal that gets forwarded to neurons in the next layer of the network. The training phase constructs a model by updating the weights associated with each neuron in the network. After being constructed, the model can be represented by a mathematical function and used to classify real world inputs to the neural network. In this way, neural networks are considered a black box machine learning approach. Deep neural networks typically have many layers and utilize specialized neural network architectures. Over the past few years there has been a resurgence in neural network research due to the success of deep neural networks when applied to certain application domains such as object detection and image classification. To demonstrate the applicability of the DFASC framework to parallelize an encrypted image classification task, we implemented a feedforward neural network classifier on top of our distributed framework using a homomorphic encryption library.

D. Security Model

In this section, we describe the threat model and security properties of the framework.

1) Threat Model: We adopt the *honest-but-curious* adversarial model. We assume that the client-side is trusted while the cloud environment is untrusted. We assume that Cloud Service Providers (CSP) as well as users can act as adversaries. When users send data into the cloud, the CSP has the ability to store the data in different locations and make use of it without the user's knowledge. We assume that the CSP will not deliberately tamper with users' data by inserting, deleting, modifying, and truncating parts of the data. An adversarial CSP may not provide false answers in response to user queries. We assume that the adversarial CSP cannot obtain a user's secret keys.

2) Security Properties: In the DFASC framework all users are required to register in the system to get credentials for authentication. Only users verified through authentication can gain access to the system. In addition to authentication, the framework employs a policy-based authorization service to provide access control to data. Using this service, users can decide who can get access to what parts of their encrypted data in the cloud. Homomorphic encryption schemes have been proven to be very secure. All the private keys for decrypting the data remain with clients, and only public keys are sent to the cloud.

IV. Data Sharing

One challenge in sharing data securely in the cloud is how to enable recipients to access the shared data. Many different techniques and approaches have been proposed in the literature to address this challenge [33], [34], [35]. Most of these approaches are based on the Public Key Infrastructure (PKI) technology, which is used to authenticate users and devices against information systems. PKI relies on a CA, which acts as a trusted third party responsible for managing and certifying public keys ownership. The CA associates a given user ID with a public key by generating a signature referred to as a certificate. In this approach, all users of a system can exchange their public keys for the purpose of data sharing. In this case, a $Sender$ wanting to share data with a $Recipient$ would use the public key of the $Recipient$ to encrypt the data. The $Recipient$ would use the corresponding secret key to decrypt the data. Through the use of digital signatures, the CA can guarantee that public keys will not be subject to impersonation, where a malicious party could replace the public key of a legitimate party with a compromised one.

One drawback of the PKI based data sharing is that it requires complex computations for data encryption and decryption, which can slow down systems with extensive data sharing. Another drawback is the dependency on the trusted third party CA, which increases the communication overhead in a system. To address these challenges, various techniques that combine public key and symmetric key cryptography have been proposed. For these approaches, the symmetric keys are used to encrypt and decrypt data and the PKI system is used to share the symmetric keys between the users. All sharing approaches based on PKI are exposed to the security vulnerabilities associated with a CA. If a CA is breached, the certificates can be compromised resulting in sending the data to the wrong users.

In this paper, we adopt a simple approach proposed in [18] for data sharing in untrusted environments, which does not require a CA. The approach makes use of a key management system based on PKI to provide clients with mechanisms to generate, store, distribute, and revoke public/private keys in a distributed system. Instead of using a CA, our approach is to exchange the private keys using an email infrastructure, where each client is equipped with a built-in email server.

1) Access Control to Encrypted Data: There are various identity and access management (IdAM) systems that can be used to restrict access to resources in a system. Among other functionalities, these systems manage, identify, authenticate, and authorize individuals to ensure appropriate access to resources.

2) Data Partitioning: To facilitate data sharing, each client needs to partition its data based on its sharing policies. Each partition will be encrypted using a different public/secret key pair to restrict access to the data. The sharing policies define how the data can be partitioned in such a way that the number of keys required to encrypt the data is minimized. Let us denote $\{c_1, c_2, \ldots, c_m\}$, the set of all clients in the system. Let us also denote $\{d_1^{c_i}, d_2^{c_i}, \ldots, d_n^{c_i}\}$, the set of data partitions for a given client c_i. Then, for each data partition $d_j^{c_i}$, a public/secret key pair, $(pk_j^{c_i}, sk_j^{c_i})$, will be generated to encrypt $d_j^{c_i}$. This will give the client a flexible approach for sharing their data in the cloud at a fine-grained level of access control.

3) Exchanging Public Keys: For the purpose of sharing encryption keys, each client will create a sharing public/secret key pair (sk_{c_i}, pk_{c_i}). The first time two clients, c_i and c_j, interact in the distributed system, they exchange their public keys as follows. The client c_i sends a message to c_j containing the tuple (Id_{c_i}, pk_{c_i}), where Id_{c_i} represents the unique indentier of c_i, and the client c_j replies with a message containing the tuple (Id_{c_j}, pk_{c_j}).

4) Sharing Data: When a sender c_i wants to share a data partition $d_j^{c_i}$ with a receiver c_j in the distributed system, the sender needs to provide the receiver with the secret key $sk_j^{c_i}$ corresponding to $pk_j^{c_i}$ used to encrypt the data partition $d_j^{c_i}$ in order to decrypt it. To protect the secret key, the sender encrypts it using the receiver's sharing public key. The sender replies with the following message containing the tuple $(Id_{c_i}, Enc(sk_j^{c_i}, pk_{c_j}))$, where $Enc(sk_j^{c_i}, pk_{c_j})$ means that the $sk_j^{c_i}$ is encrypted using the pk_{c_j}. This will guarantee that only the intended receiver can decrypt the message containing the secret key.

Figure 2 shows an example where two clients, c_1 and c_2, have generated public/secret key pairs, (pk_{c_1}, sk_{c_1}) and (pk_{c_2}, sk_{c_2}), for sharing secret encryption keys, partitioned their data, and generated different public/secret key pairs to encrypt each partition separately. The sharing keys for both clients are published in the cloud through the *Sharing Manager* and the homomorphically encrypted data stored in the cloud through the *Storage Manager*. The example also shows the partition $d_1^{c_1}$ and its corresponding secret key $sk_1^{c_1}$, to be used to decrypt it, received by the client c_2 and shared by the client c_1.

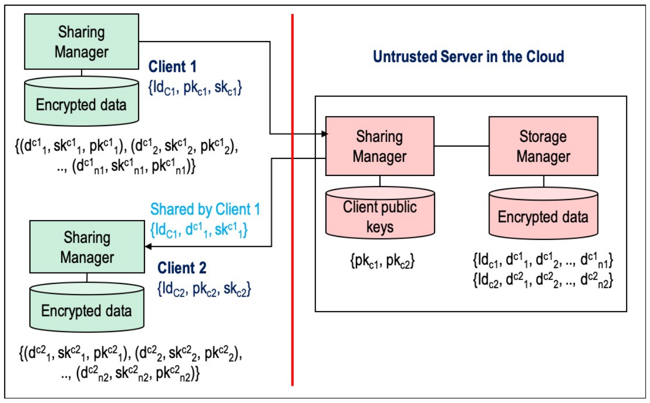
Sharing Manager
Encrypted data
Client 1
{Id_C1, pk_c1, sk_c1}
{(d^c1_1, sk^c1_1, pk^c1_1), (d^c1_2, sk^c1_2, pk^c1_2), .., (d^c1_n1, sk^c1_n1, pk^c1_n1)}
Shared by Client 1
{Id_C1, d^c1_1, sk^c1_1}
Sharing Manager
Encrypted data
Client 2
{Id_C2, pk_c2, sk_c2}
{(d^c2_1, sk^c2_1, pk^c2_1), (d^c2_2, sk^c2_2, pk^c2_2), .., (d^c2_n2, sk^c2_n2, pk^c2_n2)}
Untrusted Server in the Cloud
Sharing Manager
Storage Manager
Client public keys
{pk_c1, pk_c2}
Encrypted data
{Id_c1, d^c1_1, d^c1_2, .., d^c1_n1}
{Id_c2, d^c2_1, d^c2_2, .., d^c2_n2}

Fig. 2: Data Sharing Protocol

V. DFASC Operational Flows

The architecture of the DFASC framework comprises a number of components that interact to support the functionalities of the framework from the perspective of both developers and end-users. It abstracts out the complexity related to building a web-based client-server application, building a cloud-based distributed system, and connecting the two entities. In the following sections, we describe the operational flows of the framework, focusing particularly on how developers can extend the core components of the framework and instantiate it to build concrete systems, and then discuss how end-users can use those concrete systems.

A. Extending the Framework

For developers extending the framework, there are two main features: adding a new HE library, and adding a new data processing algorithm based on machine learning or data mining techniques. At the design level, the framework employs a modular design to isolate the HE libraries and data processing algorithms. At the implementation level, the framework uses containers to enable each HE library and each data processing algorithm to be self-contained. To add an HE library, the developer needs to deploy the HE library in a container and expose an API to enable the HE manager to make use of it. Similarly, a new data processing algorithm needs to be implemented and made available to the analytics manager, which will distribute it to the nodes at runtime. Both SVM and Neural Network implementations are included in the framework to serve as a guideline for developers to incorporate their own algorithms into the framework.

B. Instantiating the Framework to Build A Concrete System

The framework provides building blocks that can be used to build concrete distributed systems where analytic tools can be run in the encrypted domain. The application domain will determine the specific analytic tools to be applied using one of the available HE-enabled machine learning or data mining algorithms. For instance, in the application we built to evaluate the framework, both an SVM and a Neural Network were determined to be suitable for an image classification task. During the analysis, each data point falls in one of ten classes. The application domain dictates the type of data and how it needs to be encoded appropriately to ensure compatibility with the data format of the underlying HE library. Recall that the current HE libraries support only low level operations, such as addition or multiplication of numbers. It is the task of the developer to figure out how the specific data types of the application domain can be transformed in such a way that these low level operations of HE can be applied to the data.

C. Using the Concrete System

Once the system is completed, then it can be made available to end users. There are two main workflows of the system for the end user: 1) analyzing data using an analytic tool, and 2) sharing data with other users. At a high-level, the following operational workflow depicts the process for analyzing data in the distributed system.

- The User opens the Client web-based GUI.
- From the Client GUI, the user uploads the raw data to the Client local storage.
- The User selects the analytic tool to be used to process the raw data.
- The User requests the data to be encrypted.
- The Client Engine selects the appropriate HE library, and uses it to encrypt the data.
- The Client Engine sends the encrypted data along with the user parameters to the Untrusted Server.
- The Untrusted Server selects the number of nodes to use in the distributed system.
- The Untrusted Server partitions the data according to the parameters selected by the user and pushes it to the nodes.
- The Untrusted Server notifies the User after the data has been distributed.
- The User requests data to be processed and forwarded to the Untrusted Server.
- The Untrusted Server delegates the workload to the Distribution Manager.
- The Distribution Manager initiates the data processing throughout the Untrusted Distributed System.
- After the execution is completed, the Untrusted Server gathers the results from the Distribution Manager, and sends them to the User.
- The Client Engine decrypts the results and displays them on the GUI.

The following operational workflow summarizes the process for sharing data in the distributed system. The Sharing Manager on the Untrusted Server is responsible for sharing encrypted data and encrypted secret keys between parties sharing data with each other. If the recipient does not already have the secret key to decrypt the data, then the Sharing Manager will request the secret key from the sender, and the sender will encrypt the secret key using the recipient's sharing public key and send it to the Sharing Manager, which serves as the proxy between sender and receiver. We assume that the user possesses a public/secret key pair to be used by the underlying sharing protocol. We assume that each party has the sharing public key of the receiver. We also assume the data to be shared is stored with the Storage Manager component.

- From the Client GUI, the sender selects the set of data to be shared, the recipients and their sharing public keys.
- The User sends the request to share the data to the Untrusted Server.
- The Sharing Manager on the Untrusted Server passes a message to the recipient containing a reference to the stored encrypted data.
- The Sharing Manager notifies the recipients about the availability of the data.
- The Recipients retrieve the shared data and use their secret keys to decrypt the data.

VI. Implementation

In this section, we describe the implementation of the DFASC framework including all the core components. We also describe the implementation of a use case system that instantiates the framework, referred to as Image Classifier. The Image Classifier system includes two data analytics for classifying images. The first analytic tool is implemented using SVM while the second analytic is implemented using NN. As described previously, both SVM and NN are integrated into the framework. This Image Classifier is used to evaluate the feasibility of the framework. We leveraged a number of open-source projects for the implementation including the Django web framework [36], Apache Hadoop [37], Apache Spark [38], and Xen hypervisor [39].

A. Framework Implementation

The implementation of the framework is broken down into three main subsystems: Client, Cloud, and Distributed Computation.

The Client subsystem is implemented as a web service, which includes a template for the web-based client interface, a web server for managing all the client services, and a database for storing encryption/decryption keys, plaintext data, ciphertext, and cloud configuration. We used the Django web framework to implement this Client subsystem to connect all the client modules. The Django Rest Framework allows for quick development of web based REST APIs.

The Cloud subsystem implements various modules corresponding to the core component of the framework to support its various services in the cloud. These services include managing HE libraries, data processing algorithms, analytics, cloud configurations, and data sharing among users. REST APIs allow developers to extend DFASC to build concrete applications, such as adding new HE libraries and data processing algorithms.

We used Apache Spark as the basis to implement the distributed system. Spark is highly modularized, which simplifies its integration with other systems. Spark is an ideal distribution framework for DFASC, as it enables the distribution of data as well as programs for execution on the cloud nodes.

As mentioned previously, we selected the PALISADE and SEAL HE libraries as the first libraries to be integrated with the DFASC framework. Both PALISADE and SEAL are implemented using C++ and provide a simple interface to access their basic functionality. The integration of these HE libraries into our framework required building a C++ wrapper to interact with the Django web server written in Python as well as the Spark interfaces used for the distribution.

We used the Xen hypervisor to deploy a local instance of a cloud infrastructure as a service (IaaS). This local cloud serves as the testbed to generate and manage virtual machines for the distributed system. We used this local cloud instance to deploy and test our distributed framework.

B. Use Case: Image Classification

To analyze the feasibility and performance of the overall DFASC framework, we designed and implemented an image classification system using the framework. Image classification deals with labeling of images into predefined classes and training a classifier to classify a given image in one of those classes. Various machine learning classifiers such as SVM, K-Nearest Neighbors, and Decision Tree, or deep learning classifiers like Convolutional Neural Networks and Artificial Neural Networks, can be used to classify images. Image classification is useful in various application domains including autonomous driving, labeling x-ray images, and recognizing human faces for security purposes.

For our image classification system, we used the two available machine learning algorithms in the DFASC framework, SVM and NN, to implement two classifiers. As mentioned previously, DFASC includes two versions for each of the machine learning algorithms, corresponding to PALISADE and SEAL. In the following section, we describe the implementation of NN. To learn how we implemented SVM, refer to our prior publication [1].

1) NN Implementation: The specific NN we implemented is the Feedforward Neural Network, which we trained on the *sklearn Digits Dataset* [40], a reduced version of the MNIST handwritten digit dataset. Each input to the network is a 64 value array representing one of the 8 by 8 images in the dataset. The network was trained on plaintext data and then adapted to predict on encrypted data. The neural network consists of 4 layers. The input layer contains 64 neurons, with each neuron taking in one value from the 64-value input array. The second and third layers each consist of 128 neurons. This value was selected because it provided accurate prediction results on plaintext data but can be changed to optimize the calculation speed and accuracy of encrypted prediction results. The fourth and final layer is our output layer. The neuron with the highest activation in this layer is the Neural Network's final prediction. The sigmoid activation function was chosen since it can be adapted to operate on encrypted functions by representing it as a polynomial function. The following is the approximation function used:

$$f(x) = 0.500781 + 0.14670403x + 0.001198x^2 - 0.001006x^3$$

This function is described in [41]. We use cross-entropy to measure the distance between the predicted and actual probability distributions, which is then used in back- propagation to adjust the network's weights and biases. Once the network is trained, encrypted data can be fed through the network using the sigmoid approximation function described above. The output can then be decrypted to see the network's prediction. In order to speed up the calculations, multiple inputs are distributed across the Spark cluster such that each node performs one prediction and returns the results.

2) Implementation of Image Classification Application: Reusing the Software Defined Radios Link Analyzer system we introduced in [1], we implemented the Image Classification system with two clients, User Client and Administrator Client. Through the Administrator Client GUI, among other functionalities, the administrator can create nodes (VMs) and list the resources available on the distributed system. Likewise, through the User Client GUI, users can upload data, encrypt and decrypt data, and send encrypted data to the cloud for processing. During the operation of the application, after uploading the data, the user will have a set of standard machine learning algorithms to choose from to process the data. Currently, the framework provides two algorithms, SVM and NN. Once selected, the distributed machine learning algorithm with the HE implementation will be run on the distributed system, which will then return the answer in encrypted form to be decrypted when needed.

VII. Experiments

In this section, we describe the experiments we performed to analyze the performance of the overall framework. The first part of the experiments focused on analyzing how the distributed system can improve the performance of the homomorphic encryption operations. The second part of the experiments looked at the trade-offs between computation overheads of cloud analytics and the accuracy of their estimations.

A. Experiments Setup

As described in the implementation section, the current version of the DFASC framework includes two HE libraries, PALISADE and SEAL. Each of these two libraries implements a number of HE schemes including BGV, CKKS, and The Brakerski-Fan-Vercauteren (BFV) [32]. For these experiments, we used the BGV scheme as it is the most matured HE scheme. In addition, we incorporated two machine learning algorithms, SVM and NN, using both of the HE libraries, PALISADE and SEAL. To analyze the feasibility of the framework, we implemented an image classification system, which classifies images using both SVM and NN. We use this image classification system to perform the experiments.

For the experiments, we selected the *sklearn Digits Dataset* and used it as inputs for the two versions of the image classification tool, one for SVM and one for NN. The *sklearn Digits Dataset* is a popular standard dataset for classification written by scikit-learn [40]. It is made up of 1797 8x8 images, where each image is of a hand-written digit. In order to use this dataset in our experiments, we first transformed it into a feature vector with length 64, which defines its dimensionality. Overall, the dataset includes 10 classes, 180 samples per class with a total of 1797 images. The relevant features are integers from 0 to 16.

This dataset was uploaded into the Image Classification system through its GUI. We then encrypted the data homomorphically, and sent it to the distributed system for processing. For both algorithms, SVM and NN, we performed multiple runs by varying the dataset sizes and the numbers of nodes used in the distributed system.

B. Results of Experiments

Based on the above setup, we performed multiple experiments to analyze the performance of the DFASC framework in running the SVM and NN based analytic tools against the encrypted dataset. Specifically, we looked at the overhead incurred by the framework due to the expensive HE operations for both PALISADE and SEAL. During the experiment, the data was grouped into varying numbers of clusters as follows: 300, 600, 900, 1200, 1500, 1800. The distributed system was configured with varying numbers of nodes as follows: 1, 16, 32, 48, 64. Then, we ran the two analytic tools with each cluster size on each node configuration. Note that the times reported are execution times for the classifier and do not include the time taken to train the model.

1) Performance Analysis of the Distributed Framework: In this experiment, we analyzed the performance of the distributed framework as we increase the number of nodes for a given workload. The following four tables and figures summarize the results of the experiments.

Image Classification using SVM Implemented with PALISADE In this experiment, we ran the Image Classifier tool using SVM implemented with PALISADE. Table I shows the average running time of the classifier as we vary both the number of images and the number of nodes in the distributed system. The data from Table I is plotted in Figure 3. Note that, when running on a single node, the range of the running time is between 5 and 45 seconds for all six clusters of images. Figure 3 shows that, for a given cluster of images, the running time of the Image Classifier is decreasing exponentially as we increase the number of nodes in the distributed system (from 1 to 64). The largest gain in performance is when going from one node to thirty two nodes. Beyond thirty two nodes, the running time does not decrease further.

TABLE I: Image Classifier using SVM, Implemented with PALISADE (data in seconds)

Images	1	16	32	48	64	Total
300	7.57	1.30	1.20	1.20	1.60	12.87
600	15.10	1.80	1.80	1.90	2.10	22.70
900	22.78	2.30	2.00	2.50	2.80	32.38
1200	30.58	3.00	2.60	2.60	3.30	42.08
1500	37.57	3.80	3.40	3.10	3.60	51.47
1800	45.42	4.40	3.60	4.00	4.10	61.52
Total	159.02	16.60	14.60	15.30	17.50	223.02

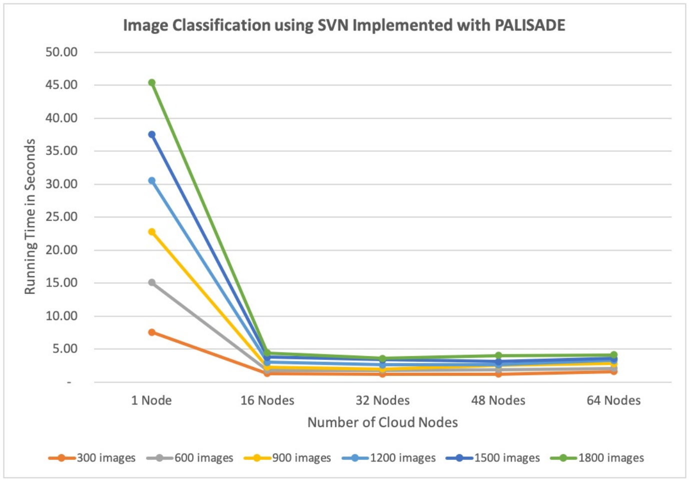

Fig. 3: Image Classifier using SVM, Implemented with PALISADE

Image Classification using NN Implemented with PALISADE In this experiment, we ran the Image Classifier tool using NN implemented with PALISADE. Table II shows the average running time of the classifier as we vary both the number of images and the number of nodes in the distributed system. The data from Table II is plotted in Figure 4. Note that, when running on a single node, the range of the running time is between 15 and 100 seconds for all six clusters of images. Figure 4 shows that, for a given cluster of images, the running time of the Image Classifier is decreasing exponentially as we increase the number of nodes in the distributed system (from 1 to 64).

Image Classification using SVM Implemented with SEAL In this experiment, we ran the Image Classifier tool using SVM implemented with SEAL. Table III shows the av-

TABLE II: Image Classifier using NN, Implemented with PALISADE (data in seconds)

Images	1	16	32	48	64	Total
300	16.08	2.10	1.60	1.70	1.70	23.18
600	31.87	3.00	2.70	2.70	3.10	43.37
900	47.95	3.90	2.80	3.60	3.60	61.85
1200	64.03	4.80	4.20	3.50	4.50	81.03
1500	79.80	6.30	5.30	4.60	5.50	101.50
1800	96.18	8.10	5.00	6.00	5.90	121.18
Total	335.92	28.20	21.60	22.10	24.30	432.12

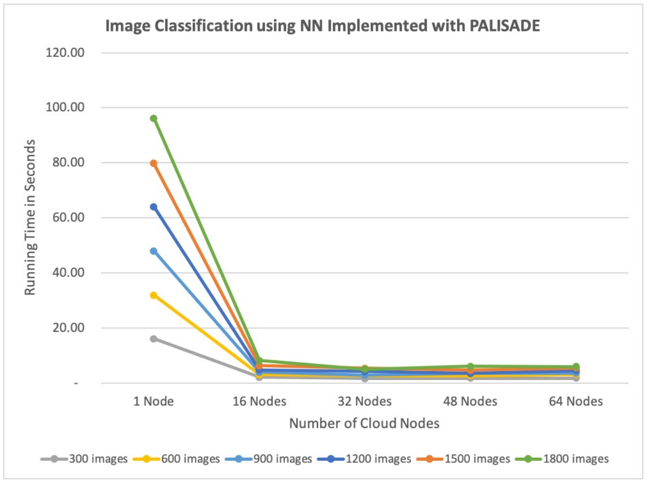

Fig. 4: Image Classifier using NN Implemented with PALISADE

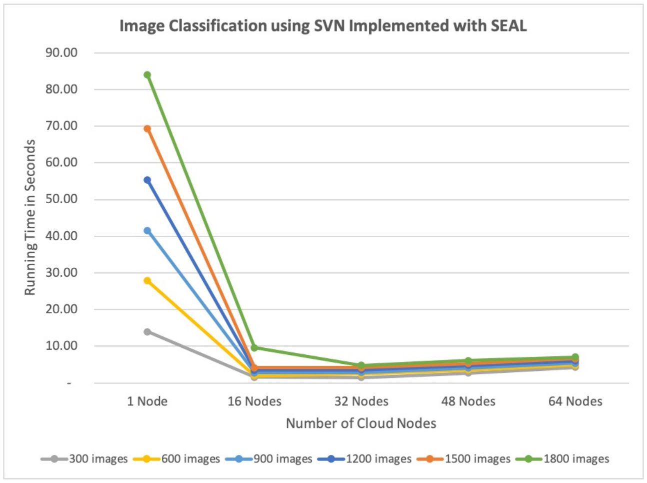

Fig. 5: Image Classifier using SVM, Implemented with SEAL

TABLE III: Image Classifier using SVM, Implemented with SEAL (data in seconds)

Images	1	16	32	48	64	Total
300	13.93	1.60	1.50	2.70	4.30	24.03
600	27.83	2.00	2.40	3.60	4.90	40.73
900	41.62	2.80	2.90	4.10	5.40	56.82
1200	55.30	3.50	3.60	4.80	6.00	73.20
1500	69.37	4.20	4.20	5.30	6.60	89.67
1800	83.98	9.60	4.80	6.10	7.10	111.58
Total	292.03	23.70	19.40	26.60	34.30	396.03

TABLE IV: Image Classifier using NN, Implemented with SEAL (data in seconds)

Images	1	16	32	48	64	Total
300	29.05	2.50	2.40	3.90	6.20	44.05
600	57.48	2.80	3.40	4.70	6.50	74.88
900	86.15	3.90	3.80	5.30	7.20	106.35
1200	115.22	4.60	4.90	5.80	8.00	138.52
1500	144.13	5.80	5.80	6.90	8.80	171.43
1800	172.47	11.00	6.20	7.40	8.90	205.97
Total	604.50	30.60	26.50	34.00	45.60	741.20

erage running time of the classifier as we vary both the number of images and the number of nodes in the distributed system. The data from Table III is plotted in Figure 5. Note that, when running on a single node, the range of the running time is between 15 and 85 seconds for all six clusters of images. Figure 5 shows that, for a given cluster of images, the running time of the Image Classifier is decreasing exponentially as we increase the number of nodes in the distributed system (from 1 to 64).

Image Classification using NN Implemented with SEAL In this experiment, we ran the Image Classifier tool using NN implemented with SEAL. Table IV shows the average running time of the classifier as we vary both the number of images and the number of nodes in the distributed system. The data from Table IV is plotted in Figure 6. Note that, when running on a single node, the range of the running time is between 30 and 175 seconds for all six clusters of images. Figure 6 shows that, for a given cluster of images, the running time of the Image Classifier is decreasing exponentially as we increase the number of nodes in the distributed system (from 1 to 64).

For a given cluster of images, as the number of nodes is increased, the time taken to execute the classifier is reduced. However, the optimal value is between 32 and 48 nodes. As the number of nodes is increased beyond 48, the time taken increases again. This is true for both classification algorithms and both HE libraries. One possible explanation for this is that, since all of the nodes were implemented as virtual machines on one physical machine, the overhead associated with each running virtual machine competed with the speedup gained from parallelizing the algorithm across multiple virtual machines. As the number of virtual machines was increased and then exceeded the number of physical cores on the machine, significant context switching overhead

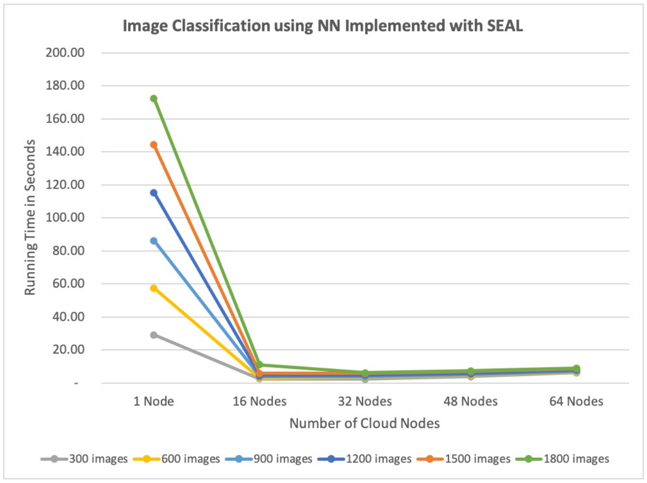

Fig. 6: Image Classifier using NN, Implemented with SEAL

was introduced, which countered any performance gains from increasing parallelism.

2) Trade-off Between Computation Overhead and Accuracy: In this experiment, we compared the performance of SVM and NN in classifying the images in the dataset. We used the combination of the selected six clusters of images (300, 600, 900, 1200, 1500, 1800) and five clusters of cloud nodes (1, 16, 32, 48, 64). This combination resulted in 30 different sets of data to be used as inputs for the two analytics. The results of running the two analytics against the 30 datasets are summarized in the previous four tables. Figure 7 shows a graph of a snapshot of these experiments, where the number of nodes is fixed to 16 and the number of datasets varied through all six clusters. As can be seen, the SVM based analytic ran exponentially faster than the NN based analytic. On the other side, the SVM based analytic was less accurate than the NN based analytic. The average accuracy after running all experiments was 85.44% for SVM based analytic and 94.65% for NN based analytic. From these results, we can observe that there is a tradeoff between the computation overhead and the accuracy of the analytics.

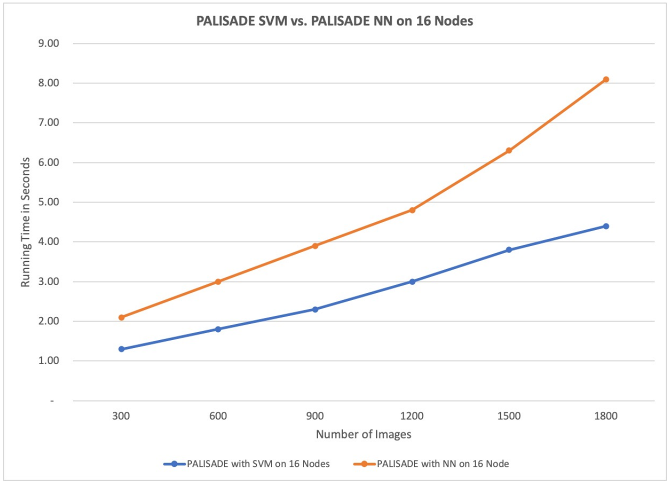

Fig. 7: SEAL and SVM based Image Classifier with 16 Nodes

Storage Overhead This experiment focused on analyzing the storage overhead associated with the ciphertext. The size of each image in the dataset is 780 bytes. After the image is encrypted homomorphically, the size of the cipher image expands to 29,367,027 bytes for PALISADE and 467,807 bytes for SEAL. The difference between PALISADE and SEAL resides in the techniques used by each library to encode the ciphertext.

3) Performance of Sharing: As described earlier, we use an email-based protocol for exchanging private keys to enable secure data sharing between users of the system. The email server provides an external channel to be used to securely share decryption keys. For this experiment we analyze the performance of the sharing protocol. The protocol includes exchanging public keys for sharing, sharing decryption keys, retrieving the shared encrypted data from the cloud, and decrypting the data. We executed this sharing protocol multiple times for both, SVM and NN, and computed the average execution time. We observed that the average time for both (0.90 seconds for SVM and 0.81 for NN) is less than a second. For data sharing purposes, this delay is not very noticeable to the user.

VIII. Related Work

Different techniques have been proposed for securing cloud analytics to increase their adoption [42]. These techniques in general use a combination of deterministic and randomized encryption, where a trade-off between the two is employed. Deterministic encryption supports a limited form of queries to be performed on the encrypted data, but is less secure, while randomized encryption is very secure, but does not support any operation on the encrypted data. Provably secure searchable encryption is an example of such techniques.

Provably secure searchable encryption (SE), including searchable symmetric encryption (SSE) and public key encryption with keyword search (PEKS), enables limited queries to be performed on encrypted data using encrypted keywords [43], [44]. SE techniques are in general expensive in terms of their computation and storage overhead. These techniques include trade-offs between security, efficiency, and functionality [45]. However, as observed in [43], regardless of SE schemes efficiency drawbacks, there is a noticeable lack of query expressiveness that hinders deployment in practice. With homomorphic encryption based security techniques, there is no limit on the number or type of operations that can be performed on the encrypted data.

Using HE to enable machine learning algorithms, including deep learning, to process data securely has gained attention in the research community in recent years [46], [47], [48], [49]. Many of the proposed approaches focus on using a given HE scheme to implement a specific machine learning algorithm. In [47], the authors show that it is possible to use a SHE scheme to implement a linear SVM to classify images for facial recognition. They extended Gentry's SHE scheme to work with low-degree polynomial functions, which are not limited by Hamming distance or linear projection. In [50], the authors went further by proposing an approach for implementing a non-linear SVM for classifying images in general using a SHE scheme. CryptoNets [51] uses the Microsoft SEAL HE library to implement deep learning algorithms. HE parallelization is limited to SIMD operations provided by the HE scheme. Faster CryptoNets [52] improves the performance of CryptoNets by leveraging the sparse representations throughout the neural network to optimize the HE operations and improve their performance. MSCryptoNet [53], based on multi-scheme FHE, protects the evaluation of the classifier, where the inputs can be encrypted with different encryption schemes and different keys. More information on current trends in using HE to process big data can be found in [54]. Unlike the above approaches, we are proposing a general framework for securing cloud analytics. We focus particularly on improving the performance of analytic tools, implemented with HE, by distributing their executions in the cloud and providing a key management infrastructure for sharing the encrypted data.

HE schemes have also been considered as a means for securing statistical computations [55]. In [56], the authors demonstrate the feasibility of using HE in approximating conventional statistical regression methods. This approach takes advantage of the fact that estimation and prediction can both be performed in the encrypted domain; bootstrapping can be avoided even for moderately large problems; and scales linearly with the number of predictors. In [57], HE is used to develop a secure system that protects both the training and prediction data in logistic regression. Despite the non-linearity of both the training and prediction in logistic regression, this paper showed that it is feasible to use HE since only the addition operation is needed, which significantly improves performance compared to FHE. Our approach differs in that it provides a framework to enable developers to use a variety of analytic tools, which can be based on statistical analysis or other analytic techniques.

Privacy-preserving data splitting is another approach proposed to preserve data privacy in the cloud. In this approach, sensitive data is split in such a way that any partition by itself is not sensitive, and is stored separately. However, the techniques proposed are not very secure as they either don't support encryption or they support only limited operations to take place in the encrypted domain [58], [59]. Furthermore, these techniques are focusing more on preserving privacy of the data at rest rather than in processing.

Other proposed techniques for securing machine learning algorithms are based on multiparty computation (MPC) [60]. Fundamentally, MPC requires interactive communications among the different nodes to perform the computations, whereas our approach using HE allows computations to be performed independently by the nodes. In addition, since HE enables the computations to be performed without any key exchange, there is no overhead of secret sharing as in MPC. HE allows for empowering a single party to take advantage of the cloud to securely analyze and share their data with other parties.

IX. Conclusion and Future Work

In this paper, we propose DFASC, a distributed framework for secure computing in the cloud, to enable the development of secure distributed systems. Secure distributed systems developed using this framework allow for analytic tools to be implemented using HE and distributed throughout the nodes of the distributed system. These systems will provide a high level of data security for the analytic tools since data will remain encrypted during transit to and from the cloud, and during storage and processing in the cloud. In addition, the framework provides a simple but flexible technique for sharing encrypted data among users. This approach of using HE to provide data security during data processing addresses the shortcomings of standard cryptographic schemes such as the Advanced Encryption Standards and Blowfish, and addresses some of the vulnerabilities of outsourcing data to the cloud. This approach will enable organizations of all types and sizes to take advantage of large pools of computing resources available in the cloud without giving up the privacy of their data.

The challenge with the existing HE schemes resides in the computation and storage overheads they incur. We addressed the computation overhead by distributing the HE computations across multiple nodes to reduce the computation time. For future work, we plan on combining the high level distribution of HE libraries and the low level parallelization of the HE operations themselves proposed in the literature. For instance, one proposed technique is to use General-Purpose Graphics Processing Units (GPGPUs) to speed up the underlying operations of the HE libraries [61]. Combining these two approaches has the potential to significantly speed up the HE operations executed within the DFASC framework.

Currently, our framework includes two HE libraries, which both implement the BGV and CKKS HE scheme. To improve the validation of the framework, we plan to incorporate additional HE libraries with additional HE schemes into the framework. We will extend the framework to facilitate a trade-offs analysis of these libraries and schemes.

References

[1] M. Diallo, C. Graves, M. August, V. Rana, and K. Groarke, "DFSCC: A distributed framework for secure computation and sharing in the cloud," in Proceedings of the Sixth International Conference on Big Data, Small Data, Linked Data and Open Data (ALLDATA) 2020. IARIA XPS, 2020, pp. 27–33.

[2] A. B. Cummings, D. Eftekhary, and F. G. House, "Cloud computing market," Fortune Business Insights, 2019.

[3] S. Kumar, F. Morstatter, and H. Liu, Twitter data analytics. Springer, 2014.

[4] A. Alexandrov et al., "The stratosphere platform for big data analytics," The VLDB Journal—The International Journal on Very Large Data Bases, vol. 23, no. 6, 2014, pp. 939–964.

[5] F. Zulkernine et al., "Towards cloud-based analytics-as-a-service (claaas) for big data analytics in the cloud," in 2013 IEEE International Congress on Big Data. IEEE, 2013, pp. 62–69.

[6] D. Talia, "Clouds for scalable big data analytics," Computer, vol. 46, no. 5, 2013, pp. 98–101.

[7] S. K. Sharma and X. Wang, "Live data analytics with collaborative edge and cloud processing in wireless iot networks," IEEE Access, vol. 5, 2017, pp. 4621–4635.

[8] R. Ranjan, "Streaming big data processing in datacenter clouds," IEEE Cloud Computing, vol. 1, no. 1, 2014, pp. 78–83.

[9] J. L. Asenjo et al., "Industrial data analytics in a cloud platform," Sep. 6 2016, US Patent 9,438,648.

[10] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," IEEE Internet Computing, vol. 16, no. 1, 2012, pp. 69–73.

[11] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," Future Generation computer systems, vol. 28, no. 3, 2012, pp. 583–592.

[12] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in 2009 Fifth International Joint Conference on INC, IMS and IDC. IEEE, 2009, pp. 44–51.

[13] H. Takabi, J. B. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," IEEE Security & Privacy, vol. 8, no. 6, 2010, pp. 24–31.

[14] N. Gruschka and M. Jensen, "Attack surfaces: A taxonomy for attacks on cloud services," in 2010 IEEE 3rd international conference on cloud computing. IEEE, 2010, pp. 276–279.

[15] C. Gentry, "Fully homomorphic encryption using ideal lattices," in Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing, ser. STOC '09. New York, USA: ACM, 2009, pp. 169–178.
[16] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," CoRR, vol. abs/1704.03578, 2017.
[17] M. H. Diallo, M. August, R. Hallman, M. Kline, H. Au, and V. Beach, "Callforfire: A mission-critical cloud-based application built using the nomad framework," in Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers, ser. Lecture Notes in Computer Science, J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. S. Wallach, M. Brenner, and K. Rohloff, Eds., vol. 9604. Springer, 2016, pp. 319–327. [Online]. Available: https://doi.org/10.1007/978-3-662-53357-4_21
[18] M. H. Diallo, B. Hore, E. Chang, S. Mehrotra, and N. Venkatasubramanian, "CloudProtect: Managing data privacy in cloud applications," in 2012 IEEE Fifth International Conference on Cloud Computing, June 2012, pp. 303–310.
[19] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of "big data" on cloud computing: Review and open research issues," Information systems, vol. 47, 2015, pp. 98–115.
[20] V. N. Inukollu, S. Arsi, and S. R. Ravuri, "Security issues associated with big data in cloud computing," International Journal of Network Security & Its Applications, vol. 6, no. 3, 2014, p. 45.
[21] R. Toshniwal, K. G. Dastidar, and A. Nath, "Big data security issues and challenges," International Journal of Innovative Research in Advanced Engineering (IJIRAE), vol. 2, no. 2, 2015.
[22] Y. Gahi, M. Guennoun, and H. T. Mouftah, "Big data analytics: Security and privacy challenges," in 2016 IEEE Symposium on Computers and Communication (ISCC). IEEE, 2016, pp. 952–957.
[23] R. L. Rivest, L. Adleman, M. L. Dertouzos et al., "On data banks and privacy homomorphisms," Foundations of secure computation, vol. 4, no. 11, 1978, pp. 169–180.
[24] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2010, pp. 1–23.
[25] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in Proceedings of the 31st Annual Conference on Advances in Cryptology, ser. CRYPTO'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 505–524.
[26] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," ACM Transactions on Computation Theory (TOCT), vol. 6, no. 3, 2014, pp. 1–36.
[27] S. Halevi and V. Shoup, "Algorithms in helib," in Annual Cryptology Conference. Springer, 2014, pp. 554–571.
[28] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in International Conference on the Theory and Application of Cryptology and Information Security. Springer, 2017, pp. 409–437.
[29] S. S. Sathya, P. Vepakomma, R. Raskar, R. Ramachandra, and S. Bhattacharya, "A review of homomorphic encryption libraries for secure computation," arXiv preprint arXiv:1812.02428, 2018.
[30] K. Rohloff and G. Ryan, "The palisade lattice cryptography library," 2017, retrieved: 01, 2020.
[31] S. S. Sathya, P. Vepakomma, R. Raskar, R. Ramachandra, and S. Bhattacharya, "A review of homomorphic encryption libraries for secure computation," CoRR, vol. abs/1812.02428, 2018.
[32] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." IACR Cryptol. ePrint Arch., vol. 2012, 2012, p. 144.
[33] G. Wang, Q. Liu, J. Wu, and M. Guo, "Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers," computers & security, vol. 30, no. 5, 2011, pp. 320–331.
[34] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," Information sciences, vol. 258, 2014, pp. 355–370.
[35] C.-C. Lee, P.-S. Chung, and M.-S. Hwang, "A survey on attribute-based encryption schemes of access control in cloud environments." IJ Network Security, vol. 15, no. 4, 2013, pp. 231–240.
[36] A. Holovaty and J. Kaplan-Moss, The definitive guide to Django: Web development done right. Apress, 2009.
[37] Apache Software Foundation, "Apache hadoop," https://hadoop.apache.org, last accessed on 12/08/20.
[38] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin et al., "Apache spark: a unified engine for big data processing," Communications of the ACM, vol. 59, no. 11, 2016, pp. 56–65.
[39] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim, "Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones," in 5th IEEE Consumer Communications and Networking Conference. IEEE, 2008, pp. 257–261.
[40] D. Dua and C. Graff, "UCI Machine Learning Repository," University of California, Irvine, School of Information and Computer Sciences, http://archive.ics.uci.edu/ml, last accessed on 12/08/20.
[41] Q. Liu, X. Lu, F. Luo, S. Zhou, J. He, and K. Wang, "Securebp from homomorphic encryption," Security and Communication Networks, vol. 2020, 06 2020, pp. 1–9.
[42] S. Sobati Moghadam and A. Fayoumi, "Toward securing cloud-based data analytics: A discussion on current solutions and open issues," IEEE Access, vol. 7, 2019, pp. 45 632–45 650.
[43] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," ACM Computing Surveys (CSUR), vol. 47, no. 2, 2014, pp. 1–51.
[44] F. Han, J. Qin, and J. Hu, "Secure searches in the cloud: A survey," Future Generation Computer Systems, vol. 62, 2016, pp. 66–75.
[45] B. Hore, E.-C. Chang, M. H. Diallo, and S. Mehrotra, "Indexing encrypted documents for supporting efficient keyword search," in Workshop on Secure Data Management. Springer, 2012, pp. 93–110.
[46] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," in International Conference on Information Security and Cryptology. Springer, 2012, pp. 1–21.
[47] J. R. Troncoso-Pastoriza, D. González-Jiménez, and F. Pérez-González, "Fully private noninteractive face verification," IEEE Transactions on Information Forensics and Security, vol. 8, no. 7, 2013, pp. 1101–1114.
[48] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, "Secure logistic regression based on homomorphic encryption: Design and evaluation," JMIR medical informatics, vol. 6, no. 2, 2018, p. e19.
[49] Y. Aono, T. Hayashi, L. Wang, S. Moriai et al., "Privacy-preserving deep learning via additively homomorphic encryption," IEEE Transactions on Information Forensics and Security, vol. 13, no. 5, 2017, pp. 1333–1345.
[50] A. Barnett et al., "Image classification using non-linear support vector machines on encrypted data." IACR Cryptology ePrint Archive, vol. 2017, 2017, p. 857.
[51] R. Gilad-Bachrach et al., "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in International Conference on Machine Learning, 2016, pp. 201–210.
[52] E. Chou and Others, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," arXiv preprint arXiv:1811.09953, 2018.
[53] P. Li et al., "Multi-key privacy-preserving deep learning in cloud computing," Future Generation Computer Systems, vol. 74, 2017, pp. 76–85.
[54] R. A. Hallman, M. H. Diallo, M. A. August, and C. T. Graves, "Homomorphic encryption for secure computation on big data," in IoTBDS, 2018.
[55] L. J. Aslett, P. M. Esperança, and C. C. Holmes, "A review of homomorphic encryption and software tools for encrypted statistical machine learning," arXiv preprint arXiv:1508.06574, 2015.
[56] P. M. Esperança, L. J. Aslett, and C. C. Holmes, "Encrypted accelerated least squares regression," 2017.
[57] Y. Aono, T. Hayashi, L. Trieu Phong, and L. Wang, "Scalable and secure logistic regression via homomorphic encryption," in Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, ser. CODASPY '16. New York, NY, USA: ACM, 2016, pp. 142–144.
[58] D. Sánchez and M. Batet, "Privacy-preserving data outsourcing in the cloud via semantic data splitting," Computer Communications, vol. 110, 2017, pp. 187–201.
[59] N. Kaaniche and M. Laurent, "Data security and privacy preservation in cloud storage environments based on cryptographic mechanisms," Computer Communications, vol. 111, 2017, pp. 120–141.
[60] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in Proceedings of the 55th Annual Design Automation Conference. ACM, 2018, p. 2.
[61] M. H. Diallo, M. August, R. Hallman, M. Kline, H. Au, and S. M. Slayback, "Nomad: a framework for ensuring data confidentiality in mission-critical cloud-based applications," Data Security in Cloud Computing, Security, 2017, pp. 19–44.