DFSCC: A Distributed Framework for Secure Computation in the Cloud

Mamadou H. Diallo, Christopher T. Graves, Michael August, Verinder Rana, and Kevin Groarke

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

***Abstract*—Currently, various advanced data analytic tools based on machine learning and data mining techniques are available for performing data analysis in the cloud. However, these tools are not very secure since the data they operate on must be in plaintext, thereby leaving the data vulnerable to both insider and outsider attacks. In this paper, we take a different approach and propose the Distributed Framework for Secure Computation in the Cloud (DFSCC), a flexible framework for building secure, distributed computation and sharing systems. The framework takes advantage of Homomorphic Encryption (HE) techniques to enable data analytics to be performed directly on the encrypted data stored within the nodes of the distributed system. An advantage of distributing data analytics into the nodes of the framework is enhanced performance of HE-based computation. In addition, the framework incorporates a cryptographic key management infrastructure to enable secure data sharing. To evaluate the framework, we extended it to implement a system that analyzes link quality between software defined radios using a machine learning algorithm. Experiments performed on the system show performance improvement of the system as the number of nodes in the cluster is increased.**

Keywords–Homomorphic Encryption, Secure Computing, Privacy, Machine Learning, Distributed Systems

I. INTRODUCTION

Performing data analytics in the cloud is becoming increasingly significant for organizations of all types and sizes. The cloud provides the scalable infrastructure and resources needed to efficiently run the analytic tools. Organizations are taking advantage of these analytic tools to gain powerful insights out of the ever-growing pools of organizational data. These analytics are in general based on techniques such as machine learning, data mining, and statistical analysis [1]–[3]. These cloud based data analytic tools are being developed for various application domains [4]–[7]. However, there is also a growing concern about data security and privacy in cloud-based systems and applications that provide analytic tools [8]–[11]. In particular, cybersecurity attackers are becoming more sophisticated, and attacks on data in large organizations are occurring more frequently [12].

The main issue is how the data is 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.

To address this shortcoming of existing standard cryptographic schemes, Homomorphic Encryption (HE) has been proposed [13]. 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 overheads [14]. 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.

In this paper, we introduce the Distributed Framework for Secure Computation in the Cloud (DFSCC), a distributed framework that enables the development of secure computation and sharing systems using HE schemes. HE schemes provide data security not only in transit and at rest, but most importantly, during processing. 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 into 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 [15]. Using this approach, an organization can analyze data in the cloud and share the results with other organizations. Distributing analytic tool execution into 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

 ISBN: 978-1-61208-775-7

Section IV we outline the data flow through the system. In Section V we discuss our implementation of the framework and sample application. In Section VI we present the results of experiments performed on the system. In Section VII we contrast our paper with related works. We end the paper with a conclusion and future work in Section VIII.

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 Homomorphic Encryption, which can be used to address data security in the cloud.

A. Data Analysis in the Cloud

Machine learning, data mining, and statistical modeling and analysis techniques are steadily making their way into enterprise applications in areas, such as customer support, fraud detection, and business intelligence [4]. 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 [16]. 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 cyber attacks from both internal and external attackers. 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

Craig Gentry [13] introduced the first working *Fully Homomorphic Encryption* (FHE) scheme in his 2009 PhD dissertation by taking a *Somewhat Homomorphic Encryption* (SHE) scheme and "squashing" the decryption circuit to reduce the noise in a process called "bootstrapping". However, this process was impractical due to the required computation time. A more practical approach explored within the FHE research community has been the *Learning With Errors* (LWE) problem and its variants, particularly the *Ring Learning With Errors* (RLWE) problem. Below, we describe these two approaches.

1) Gentry's FHE Implementation with Ideal Lattices: A *Point Lattice*, or *Lattice* $\mathcal{L}$, is the set of all integer linear combinations of a set of linearly independent vectors $\mathcal{B} \subseteq \mathbb{R}^m$. For constants $c_i \in \mathbb{Z}$ and $\mathbf{b}_i \in \mathbb{R}^m$,

$$\mathcal{L} = \sum_{i=1}^{n} c_i \mathbf{b}_i.$$

Gentry's FHE scheme encrypts a plaintext by placing it in the *fundamental region* of a lattice with "noise" generated by several classic, hard lattice problems. After a number of additions and multiplications, the ciphertext noise risks becoming so great that the ciphertext is moved outside of the fundamental region. Therefore, Gentry "squashes" the decryption circuit to give bootstrapability. An encryption scheme is *bootstrappable* if it can homomorphically evaluate its own decryption circuit. The process of bootstrapping involves providing the secret key that has been re-encrypted with a new key. In order to keep the noise at a manageable level, the bootstrapping process is done before the noise reaches the threshold where decryption is no longer possible. Because of this limitation, bootstrapping may be every other operation. In later implementations of Gentry's FHE scheme, the performance of the bootstrapping process has been optimized.

2) FHE Based on Ring Learning With Errors: To describe 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 described above, a message $m \in R_q$ can be encrypted by using the b element above as a one-time pad encryption scheme [17]. The ciphertext can be represented by $\mathbf{c} = \mathbf{b} + \mathbf{m}$, where $\mathbf{c} \in R_q$. FHE schemes based on the infeasible RLWE Problem have been shown to be cryptographically secure given an appropriate security level.

One major effort in FHE using this approach is Microsoft's Simple Encrypted Arithmetic Library (SEAL), which utilizes the BFV FHE scheme based on RLWE [18]. The BFV scheme allows for modular arithmetic to be performed on encrypted integers. The SEAL library also implements the CKKS FHE scheme, which supports approximate arithmetic over complex and real numbers [19]. Another major FHE library which implements these schemes is the PALISADE library [20].

III. FRAMEWORK

In this section, we describe the architecture of the proposed framework, how HE schemes and data processing algorithms are integrated into the architecture, and a mechanism for data sharing.

A. Architecture

The DFSCC framework is designed using a hybrid client-server/distributed model, where clients send requests to the server, and the server sends the client's requests to the distributed system for processing. 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*. We adopt the *honest-but-curious* adversarial model. We assume that the client-side is trusted while the cloud environment is untrusted. Therefore, all the private keys for decrypting the data remain with clients, and only public keys are sent to the cloud.

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.

 ISBN: 978-1-61208-775-7

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.

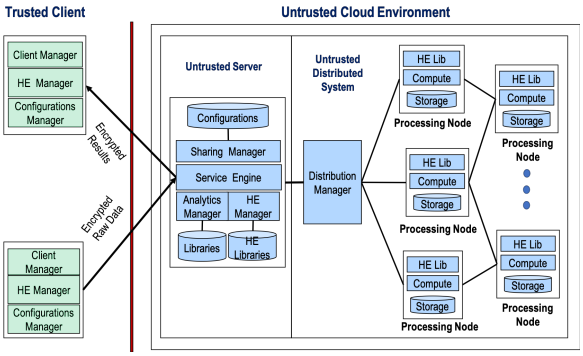

Figure 1. Distributed Framework Architecture

B. HE Scheme Integration

At the core of DFSCC are the mechanisms for incorporating HE schemes with distributed data processing algorithms. DFSCC also provides a key management infrastructure to enable sharing of data processed by the distributed system. Additionally, by abstracting out the core functionality that is commonly found in HE schemes, DFSCC is designed to facilitate the incorporation of different HE libraries. These common operations include key generation, encryption, decryption, and parameter selection. These operations are abstracted out into an interface that can then be used to integrate a given HE library.

C. Data Processing Integration

Machine learning algorithms are being used in the cloud to analyze data stored in the cloud and to enhance the cloud's capabilities. DFSCC provides an extensible interface to enable developers to extend or customize DFSCC 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 DFSCC framework is the linear Support Vector Machine (SVM). 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 DFSCC framework, we implemented an SVM classifier on top of our distributed framework using the PALISADE library.

D. Key Management Infrastructure

We use a key management system based on Public Key Infrastructure (PKI) to provide clients with mechanisms to generate, store, distribute, and revoke public/private keys in the distributed system. Overall, the key management system is based on the simple approach proposed in [15], which doesn't require a central authority for managing the keys. The approach is to exchange private keys using an email infrastructure, where each client is equipped with a built-in email server.

The protocol for exchanging public keys is as follows.

1) Exchanging Public Keys: 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}) and the client c_j replies with a message containing the tuple (Id_{c_j}, pk_{c_j}).

2) Data Partitioning: To facilitate data sharing, each client needs to partition their data based on sharing policies. Each partition will be encrypted using a different public/secret key pair. For instance, let's assume the user data d is divided into a set of partitions $\{d_1, d_2, \ldots, d_n\}$. Then, for each d_i, a public/secret key pair, (pk_i, sk_i) will be generated to encrypt d_i. This will give the client a flexible approach for sharing their data in the cloud at a fine-grained level.

3) Sharing Data: When a sender wants to share a piece of data d_i with a receiver in the distributed system, the sender needs to provide the receiver with the secret key sk_i corresponding to pk_i used to encrypt the data d_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 $(Id_{receiver}, Enc(sk_{sender}, pk_{receiver}))$, where

 ISBN: 978-1-61208-775-7

$Enc(sk_{sender}, pk_{receiver})$ means that the sk_{sender} is encrypted using the $pk_{receiver}$. This will guarantee that only the intended receiver can decrypt the message containing the secret key. Note that, in this approach of data sharing, we assume that the distributed system includes an access control enforcement mechanism to give access to data based on sharing policies defined by the users. The description of the access control mechanism is beyond the scope of this work.

IV. DFSCC Operational Flows

The architecture of the DFSCC 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. 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. In addition to the SVM implementation, other data processing algorithms will be 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, SVM was determined to be suitable to implement a tool to analyze radio data to optimize the link quality between Software Defined Radios (SDRs). Analyzing radio link quality requires classifying the data into two classes, high quality and low quality. During the analysis, each data point falls in one of those two classes. The application domain will also dictate the type of data that 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 the basic operations of HE can be applied on 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 doesn't 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 Distribution 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.

 ISBN: 978-1-61208-775-7

- 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.

V. IMPLEMENTATION

We implemented the overall DFSCC framework and the Software Defined Radio link quality analysis application to evaluate the framework. We leveraged a number of open-source projects for the implementation including the Django web framework, PALISADE HE library [20], Apache Hadoop, Apache Spark, and Xen hypervisor.

A. Framework

The implementation is broken down into three main subsystems: client, server, and distributed system. We use the Django web framework to develop a web-based system to connect the client and server subsystems and to provide web service capability to DFSCC.

As mentioned previously, we selected the PALISADE HE library as the first library to be integrated with the DFSCC framework. PALISADE is implemented using C++ and provides a simple interface to access its basic functionality. The integration of this HE library 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 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 DFSCC, as it enables the distribution of data as well as programs for execution on the cloud nodes. REST APIs allow developers to extend DFSCC to build concrete applications, such as adding new HE libraries and data processing algorithms.

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 virtual machines for the distributed system. We used this local cloud instance to deploy and test our distributed framework.

B. Framework Use Case: SDR Link Quality Analysis

For the test application, we implemented two Graphical User Interfaces (GUI), one for the administrator, and another for the user. Through the admin GUI, among other functionalities, the admin can create nodes (VMs) and list the resources available on the distributed system. Likewise, through the user GUI, users can upload data, encrypt and decrypt data, and send encrypted data to the cloud for processing. Once the data has been uploaded, there is a library of standard machine learning algorithms that the user can select to run on the uploaded data. Once selected, the distributed machine learning algorithm with the HE implementation will be run on the distributed system that will then return the answer in encrypted form to be decrypted when needed. We used this process to analyze the quality of Software Defined Radio signals to determine the best way to tune the radios. This was done using a simple SVM on the data to classify good versus bad link quality of the radios. Note that, in this paper, we focus only on analyzing the performance and overhead of the underlying HE operations.

VI. EXPERIMENTS

As part of the experimental setup, we deployed two Software Defined Radios, a sender and a receiver, and established a connection between the two. Then, we initiated a video stream from the sender to the receiver and extracted the data packets in the stream using the network packet capture feature of Snort. The most relevant features in this dataset are the bit error ratio, signal level, noise level, distortion level, and signal-plus-noise-plus-distortion to noise-plus-distortion ratio. Due to the limited number of features in this dataset, we generated synthetic radio data to analyze the performance of the system with a larger number of features. In the synthetic dataset, the number of features ranges from 8 to 256. We also varied the number of nodes in the distributed system from 4 to 64.

Based on the above setup, we performed a number of experiments to analyze the performance of the DFSCC framework in running the SVM based analytic tool against the encrypted dataset. Specifically, we looked at the overhead incurred by the framework due to the expensive HE operations. During the experiment, the data was grouped into varying numbers of features as follows: 8, 16, 32, 64, 128, 256. The distributed system was configured with varying numbers of nodes as follows: 4, 8, 16, 32, 64. Note that both of these scales are logarithmic as represented by the y-axis in each of the figures below. Then, we ran the analytic tool with each feature size on each node configuration.

A. Features Comparison

In this experiment, we ran all the feature sizes on 1-node, 8-node, and 16-node configurations, and recorded the running time of the computations. In both, Figure 2 and Figure 3, each of the points represents a single run of the SVM algorithm for a feature size specified on the y-axis. The x-axis represents the running time of the algorithm, where the blue circle represents a computation distributed across 8 or 16 nodes, and the red triangle represents running on a single node.

As can be seen in Figure 2, with low numbers of features, there isn't a significant difference between the two setups. However, as the number of features gets larger, the speed improvement becomes clear. Note that the outlier seen in each of the runs was an initialization of the distributed system that was exacerbated in the distributed setting but can also be seen in the single node setting. This outlier is only on the first run of the algorithm so will be less significant over many runs of the algorithm.

B. Nodes Comparison

In this experiment, we look at the comparison of running the tool with 64 and 256 features while varying the number of nodes as described earlier. We can see in Figure 4 and Figure 5 that by increasing the number of nodes there is a performance improvement. Given this, we can determine the optimal number of nodes needed for a dataset, for a given workload on the system. There is still a significant performance improvement resulting from distribution across multiple nodes, but care should be taken to balance the workload evenly throughout the distributed system in order to minimize the amount of downtime waiting for dependent computations to complete before the results of the overall computation can be delivered.

 ISBN: 978-1-61208-775-7

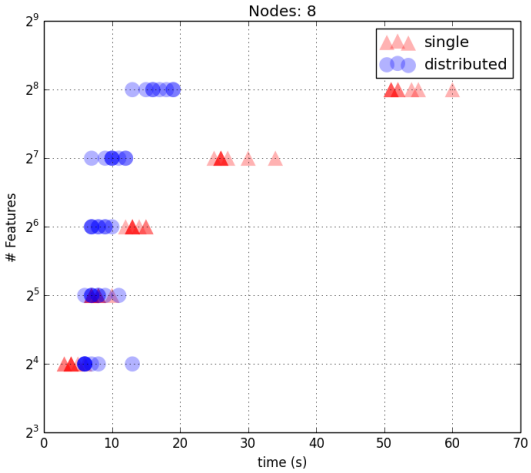

Figure 2. Overhead for 8 Nodes Setup

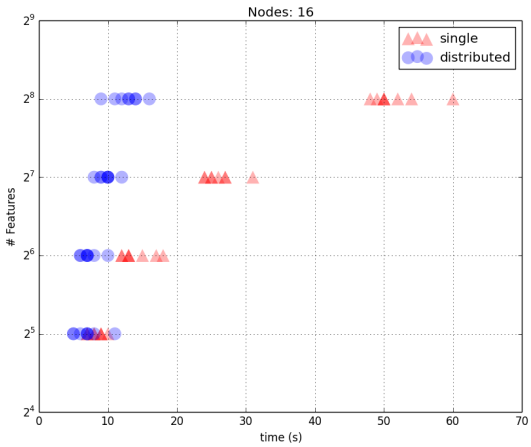

Figure 3. Overhead for 16 Nodes Setup

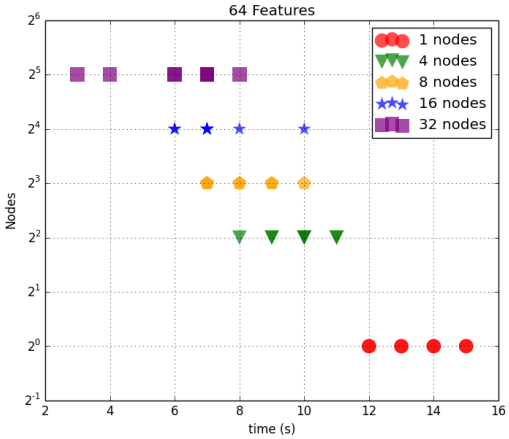

Figure 4. Overhead of 64 Features Comparison

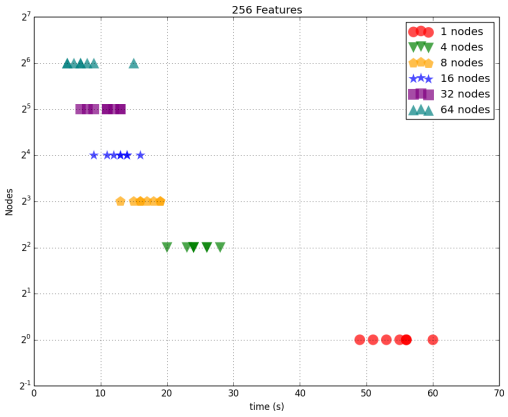

Figure 5. Overhead of 256 Features Comparison

VII. Related Work

Using HE to enable machine learning algorithms, including deep learning, to process data securely has gained attention in the research community in recent years. Many of the proposed approaches focus on using a given HE scheme to implement a specific machine learning algorithm. In [21], 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 [22], the authors went further by proposing an approach for implementing a non-linear SVM for classifying images in general using a SHE scheme. CryptoNets [23] 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 [24] improves the performance of CryptoNets by leveraging the sparse representations throughout the neural network to optimize the HE operations and improve their performance. MSCryptoNet [25], based on multi-scheme FHE, protects the evaluation of the classifier, where the inputs can be encrypted with different encryption schemes and different keys. Unlike the above approaches, we are proposing a general framework for secure machine learning in the cloud. Furthermore, we are using distributed processing to improve the performance of machine learning computations and HE based computations.

HE schemes have also been considered as a means for securing statistical computations. In [26], 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 [27], 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,

 ISBN: 978-1-61208-775-7

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 [28], [29]. 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 MPC [30]. 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.

VIII. Conclusion and Future Work

In this paper, we propose DFSCC, a distributed framework for secure computing in the cloud, to enable the development of secure distributed systems. A secure distributed system developed using this framework allows for analytic tools to be implemented using HE and distributed throughout the nodes of the distributed system. Systems developed using the framework 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, and addresses some of the vulnerabilities of outsourcing data to the cloud. This 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 GPGPUs to speed up the underlying operations of the libraries [31]. Combining these two approaches has the potential to significantly speed up the HE operations executed within the DFSCC framework.

References

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

[2] 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.

[3] 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.

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

[5] 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.

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

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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, NY, USA: ACM, 2009, pp. 169–178.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] J. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," 11 2017, pp. 409–437.

[20] K. Rohloff and G. Ryan, "The palisade lattice cryptography library," 2017, retrieved: 01, 2020.

[21] 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.

[22] A. Barnett et al., "Image classification using non-linear support vector machines on encrypted data." IACR Cryptology ePrint Archive, vol. 2017, 2017, p. 857.

[23] 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.

[24] E. Chou and Others, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," arXiv preprint arXiv:1811.09953, 2018.

[25] P. Li et al., "Multi-key privacy-preserving deep learning in cloud computing," Future Generation Computer Systems, vol. 74, 2017, pp. 76–85.

[26] P. M. Esperança, L. J. Aslett, and C. C. Holmes, "Encrypted accelerated least squares regression," 2017.

[27] 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.

[28] D. Sánchez and M. Batet, "Privacy-preserving data outsourcing in the cloud via semantic data splitting," Computer Communications, vol. 110, 2017, pp. 187–201.

[29] 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.

[30] 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.

[31] M. Diallo, M. August, R. Hallman, M. Kline, H. Au, and S. Slayback, Nomad: a framework for ensuring data confidentiality in mission-critical cloud-based applications, 10 2017, p. 19–44.

 ISBN: 978-1-61208-775-7