Received 12 October 2023, accepted 5 November 2023, date of publication 13 November 2023, date of current version 17 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3332222

RESEARCH ARTICLE

Privacy-Preserving Machine Learning on Apache Spark

CLÁUDIA V. BRITO[1,2], PEDRO G. FERREIRA[1,3], BERNARDO L. PORTELA[1,3], RUI C. OLIVEIRA[1,2], AND JOÃO T. PAULO[1,2]

[1]INESC TEC, 4200-465 Porto, Portugal
[2]Department of Informatics, University of Minho, 4710-057 Braga, Portugal
[3]Faculty of Sciences, University of Porto, 4099-002 Porto, Portugal

Corresponding author: Cláudia V. Brito (claudia.v.brito@inesctec.pt)

This work was supported by FCT - Portuguese Foundation for Science and Technology through the Ph.D. grant DFA/BD/146528/2018 and realized within the scope of the project LA/P/0063/2020.

ABSTRACT The adoption of third-party machine learning (ML) cloud services is highly dependent on the security guarantees and the performance penalty they incur on workloads for model training and inference. This paper explores security/performance trade-offs for the distributed Apache Spark framework and its ML library. Concretely, we build upon a key insight: in specific deployment settings, one can reveal carefully chosen non-sensitive operations (*e.g.* statistical calculations). This allows us to considerably improve the performance of privacy-preserving solutions without exposing the protocol to pervasive ML attacks. In more detail, we propose Soteria, a system for distributed privacy-preserving ML that leverages Trusted Execution Environments (*e.g.* Intel SGX) to run computations over sensitive information in isolated containers (enclaves). Unlike previous work, where all ML-related computation is performed at trusted enclaves, we introduce a hybrid scheme, combining computation done inside and outside these enclaves. The experimental evaluation validates that our approach reduces the runtime of ML algorithms by up to 41% when compared to previous related work. Our protocol is accompanied by a security proof and a discussion regarding resilience against a wide spectrum of ML attacks.

INDEX TERMS Privacy-preserving, machine learning, distributed systems, apache spark, trusted execution environments, Intel SGX.

I. INTRODUCTION

The ubiquitous environment provided by cloud computing providers offers a scalable, reliable, and performant environment to deploy compute-intensive Machine Learning (ML) workloads. However, many of these workloads operate over users' sensitive information (*e.g.*, medical records, financial information). Regulations like HIPAA and GDPR enforce strong security policies when processing or storing sensitive data at untrusted third-party infrastructures [1], [2]. As such, outsourcing ML data storage and computation to third-party services leave users vulnerable to attacks that may compromise the integrity and confidentiality of their data [3], [4]. Indeed, the ML pipeline encompasses several stages, both for model training and inference, in which users' data is known to be susceptible to different attacks such as *adversarial attacks*, *model extraction*, and *inversion*, and *reconstruction attacks* [5], [6], [7].

Recent works have addressed these attacks with solutions based on homomorphic encryption or secure multi-party computation schemes [8], [9]. However, these cryptographic schemes impose a significant performance toll that restricts their applicability to practical scenarios [10]. To circumvent this performance penalty, another line of research is that of exploring hardware technologies enabling Trusted Execution Environments (TEEs), such as Intel SGX [11]. These technologies allow the execution of code within isolated processing environments (*i.e.*, enclaves) where data can be securely handled in its original form (*i.e.*, plaintext) at untrusted servers.

The associate editor coordinating the review of this manuscript and approving it for publication was Peter Langendoerfer.

The latter approach typically deploys full ML workloads inside TEEs [12], [13], [14]. However, as the amount of computational and I/O operations performed at the enclaves increases, the performance of ML training and inference is noticeably affected by hardware limitations, limiting the design's applicability in practice [15].

This paper builds upon the idea that ML runtime performance could be improved by reducing the number of operations done at enclaves. In fact, this insight is backed up by previous work [16], [17], [18] exploring the partitioning of computation across trusted and untrusted environments, but in contexts (*e.g.*, SQL processing, MapReduce, distributed coordination) with different security requirements and processing logic than the ones found for ML workloads.

Therefore, the key challenge addressed by this paper is to understand and define the set of ML operations to run inside/outside TEEs. Ideally, these operations should significantly reduce the enclaves' overall computational and I/O load for different ML workloads, and doing so should not leak critical sensitive information during the execution of ML workloads.

Our reasoning is twofold: *i.)* the majority of current attacks on the ML pipeline are only successful if the attacker has some knowledge about the datasets or the models being used [6], [19]; and *ii.)* previous work shows that such knowledge cannot be inferred from the information leaked by statistical operations, such as the calculation of confidence results, table summaries, ROC/AUC curves, and probability distributions for classes, without additional knowledge regarding the dataset or the model [20]. As a result, these operations are ideal candidates to be offloaded from enclaves. We support these claims by analyzing the security and performance implications of different ML workloads and attacks.

A. SOLUTION

Thus, we propose Soteria,[1] an open-source system for distributed privacy-preserving ML that leverages the scalability and reliability of Apache Spark and its ML library (MLlib) [21]. Unlike previous solutions [22], [23], Soteria supports a wide variety of ML algorithms without changing how users build and run these within Spark. Further, it ensures that critical operations, which enable existing attacks to reveal sensitive information from ML datasets and models, are exclusively performed in secure enclaves. This means that the sensitive data being processed only exists in plaintext inside the enclave and is encrypted in the remainder of the dataflow (*e.g.*, network, storage). This solution enables robust security guarantees, ensuring data privacy during ML training and inference.

[1]This is an extended version of the work published in [21]. We improved our earlier publication by refining the paper's structure, contributions, and background work, adding a new discussion about Soteria's design goals, introducing the analysis of our secure mechanisms for a concrete ML algorithm, providing new experiments with more ML workloads and their analysis, improving the comparison of Soteria against related work, and adding a formal security proof for Soteria.

Soteria introduces a new computation partitioning scheme for Apache Spark's MLlib, Soteria-P, that offloads non-critical statistical operations from the trusted enclaves to untrusted environments. Soteria-P is accompanied by a formal security proof for how data remains private during ML workloads and an analysis of how this guarantee ensures resilience against various ML attacks. Furthermore, Soteria offers a baseline scheme, Soteria-B, where all ML operations are done inside trusted enclaves without a fine-grained differentiation between critical and non-critical operations. Soteria-B provides a performance and security baseline for comparison against our new partitioned scheme.

We compare experimentally both approaches with a non-secure deployment of Apache Spark and a state-of-the-art solution, namely SGX-Spark [22]. Our experiments, resorting to the HiBench benchmark [24] and including seven different ML algorithms, show that Soteria-P, while considering a more significant subset of ML attacks, reduces training time by up to 41% for Gradient Boosted Trees workloads and up to 4.3 hours for Linear Regression workloads, when compared to SGX-Spark. Also, compared to Soteria-B, Soteria-P reduces execution time by up to 37% for the Gradient Boosted Trees workloads and up to 3.3 hours for the Linear Regression workloads.

B. CONTRIBUTIONS

Our contributions are summarized as follows:

- A new system tailored for Apache Spark, which leverages Intel SGX for the secure execution of a wide range of ML algorithms while not requiring any changes to how users typically implement and run their workloads.
- A novel computation partitioning scheme, named Soteria-P, builds upon the insight that the computation over non-critical information can be offloaded from trusted enclaves to reduce the performance impact of these secure mechanisms on ML processing.
- An open-source prototype, implemented by resorting to the Gramine Library OS, overcoming the complexity of supporting both Java and Scala programming languages in Intel's SGX while offering additional security guarantees (*i.e.*, security isolation, host platform compatibility).
- An extensive evaluation, including seven ML algorithms, demonstrates the feasibility and usability of our system. The experimental evaluation validates that our approach reduces the runtime of ML algorithms by up to 41% compared to previous related work.

C. OUTLINE

The remainder of this paper is organized as follows. Section II gives brief background information on Apache Spark, MLlib, Intel SGX, and Gramine. Then, Section III presents the threat model of Soteria and a description of current ML attacks. In Section IV, we concretely define the design goals and architecture of Soteria. Section V discusses the conducted experimental testbed, the main observations, and their analysis and discussion. Next, Section VI presents the current state-of-the-art solution, emphasizing their main differences with Soteria. Finally, Section VII concludes and highlights the main contributions proposed in this paper.

II. BACKGROUND

Next, we provide background on the key technologies that work as building blocks for our solution. Namely, in Subsection II-A, we overview the core concepts of Apache Spark and MLlib. Finally, Subsection II-B exposes key concepts of Intel SGX, while Subsection II-C describes Gramine, the chosen LibOS to implement Soteria.

A. APACHE SPARK

Apache Spark is a distributed cluster computing framework that supports Extract, Transform, and Load (ETL), analytical, ML, and graph processing workloads over large volumes of data. Spark can be deployed on a cluster of servers, at a private infrastructure, or in the cloud and supports different data sources (*e.g.*, HBase, HDFS) for reading the data to be processed and storing the corresponding output and logs. The framework performs most of the computation in-memory, thus promoting better performance for data-intensive applications when compared to Hadoop's MapReduce [25].

Spark follows a master-worker distributed architecture. Users interact with the master node by submitting jobs (*i.e.*, processing workloads) and collecting the corresponding outputs. The master is then responsible for submitting jobs to the worker nodes by resorting to a job scheduler and resource negotiator named Spark Driver. This driver splits the jobs into tasks (*i.e.*, map phase) and defines which workers will compute a given set of tasks. Finally, each worker performs the designed task(s) and reports the resulting output to the master, which then aggregates (*i.e.*, reduce phase) and forwards the response back to the user. Further details about Spark's flow are discussed in Section IV-B.

Apache Spark's ML library (*MLlib*) [26] enables Spark users to build end-to-end ML jobs. Its workflow is similar to the one found in other ML solutions, with the addition of an initial data treatment stage. The library also provides a set of tools and utilities for feature extraction and model persistence.

Figure 1 shows the typical ML workflow for data engineers and scientists. The first stage is typically known as the process of ETL, where data is collected and extracted in a pre-determined format, treated accordingly, and loaded in the needed format (*e.g.*, CSV, Parquet, Text File). In the second stage, data is split into train and test datasets, and a given ML algorithm is chosen. The third stage is the training stage, where data is iterated to deliver an optimized trained model

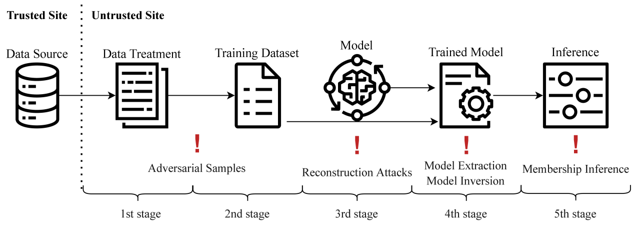


FIGURE 1. Machine Learning process flow.

at the fourth stage. In the fifth stage, the trained model can be saved (persisted) and loaded (accessed) for inference.

B. INTEL SOFTWARE GUARD EXTENSIONS

Intel SGX provides a set of new instructions available on Intel processors that applications can use to create trusted memory regions. These regions (enclaves) are isolated from any other code on the host system, preventing other processes, including those with higher privilege levels (such as the host OS, hypervisor, and BIOS), from accessing their content [11], [27].

Since SGX protects code and data from privileged access, sensitive plaintext data can be processed at the enclave without compromising its privacy. Thus, TEEs outperform typical traditional cryptographic computational techniques (*e.g.*, searchable encryption, homomorphic encryption) [27]. Nonetheless, even though the second generation of SGX has improved the size of the protected memory region, it still defines the Enclave Page Cache (EPC) to 128MB per CPU [28]. Memory swapping occurs when such a limitation is met, which is a performance-costing mechanism [15]. Thus, SGX-based solutions must balance the number of I/O operations, the amount of data handled by enclaves, and the Trusted Computing Base (TCB) to optimize performance.

In this paper, we chose Intel SGX over other TEE's (*e.g.*, ARM TrustZone [29]) due to its general use in academia [14], [16], [17] and industry [30], availability, as well as its security guarantees and computing reliability. However, our solution is generic and can be applied to other TEE technology that follows similar design principles to SGX.

C. GRAMINE

Porting existing source code into TEEs is a non-trivial and error-prone task. Initially, applications' source code needed to be rewritten in C/C++ to run inside enclaves, thus requiring manual intervention. This inefficient approach led to the proposal of solutions to port unmodified applications to run inside TEEs automatically.

In this paper, we resort to Gramine [31] (previously named *Graphene-SGX*), an open-source library OS designed to run unmodified applications inside Intel SGX enclaves. Gramine's architecture (Figure 2) is based on Drawbridge's *picoprocess* [32], which provides an isolated address space. Its architecture contains three main components: *i)* the unmodified binary application and corresponding libraries, to be ported into SGX; *ii)* the LibOS, a custom operating system implemented by Gramine; and *iii)* the Enclave Platform Adaptation Layer (PAL).

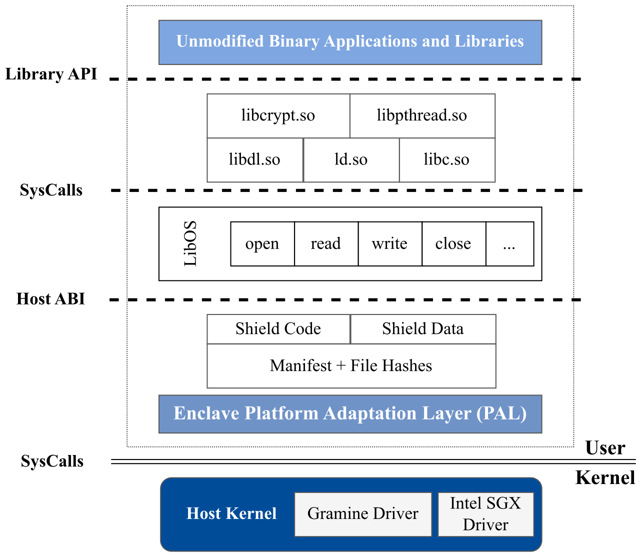


FIGURE 2. Gramine's architecture based on [31]. The enclave includes an OS shield, a library OS, `libc`, and other user binaries.

a: PAL

This is a core component of Gramine, which acts as an intermediary between the LibOS and the underlying hardware platform to provide the necessary abstractions and interfaces that enable Gramine to run efficiently [33]. In detail, this layer:

- Abstracts the low-level hardware details, allowing Gramine to be portable across different hardware platforms without requiring extensive modifications to its core codebase.
- Handles the initialization and management of the SGX enclaves and sets up the enclave memory layout.
- Facilitates communication between the Gramine enclave and the host system.
- Provides mechanisms for securely passing data and invoking system calls from the enclave to the host and vice versa.
- Manages the resources allocated to the enclave, ensuring it has access to the necessary memory, CPU, and other resources.

b: MANIFEST FILE

Through this configuration file, developers specify the application libraries to be deployed inside a secure enclave, the paths for files that the enclave will need to access securely, and trusted system libraries. It also contains the environment's specifications (*e.g.*, the CPU and memory resources for each enclave) [34].

In sum, the Gramine library works similarly to a paravirtualization environment, taking advantage of the standard virtualization benefits such as security isolation, host platform compatibility, and migration while mapping high-level APIs onto a few paravirtual interfaces to the host kernel [31].

By resorting to this framework, one can take full advantage of the application's native performance while potentiating the use of trusted execution environments. Moreover, with a growing community focused on confidential computing with Intel SGX, Gramine is supported by Intel and several universities [35].

III. THREAT MODEL AND ATTACKS

This section provides an overview of Soteria's threat model, where we define its deployment setting (§III-A) and present the ML attacks covered by our solution (§III-B).

A. SOTERIA *THREAT MODEL*

Soteria enables the secure outsourcing of ML training and inference workloads. These are scenarios where the data owner holds sensitive information (a private dataset and/or model) and wants to perform some ML workload on it using an external cloud provider.

This is a typical setting in health-related contexts, such as rare disease identification [36]. Hospitals and clinics are unwilling to (or outright unable to) offload patient data to a potentially malicious cloud service, even if such service provides computational resources to process large quantities of data more efficiently. Alternatively, insurance companies also collect large quantities of data and leverage it to forecast traffic accidents [37]. These companies would also significantly benefit from using the resources of third-party cloud infrastructures. However, risking the leakage of said predictive model entails losing valuable market advantage.

Our deployment model is depicted in Figure 3 and is as follows. The client (data owner) will be trusted and will provide input for ML tasks. Then, a Spark master node and N worker nodes will be deployed in an untrusted environment (cloud provider) equipped with Intel SGX technology. Externally, we also consider a distributed data storage backend. The protocol assumes an implicit setup where the client securely stores its input data within this backend, which is also considered untrusted throughout the protocol execution.

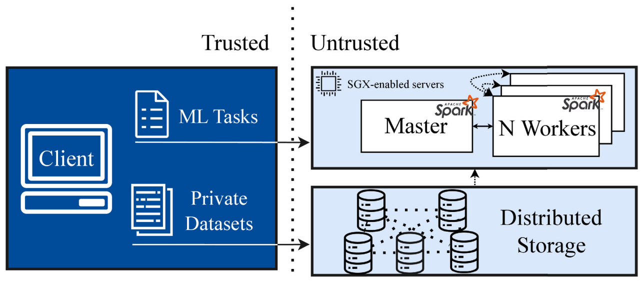


FIGURE 3. Soteria deployment model.

We consider semi-honest adversaries, meaning that security is defined according to a threat that attempts to break the confidentiality of data and model but will not actively deviate from the protocol specification. This is a good fit for cloud-based systems, where data breaches are common and malicious entities can read internal processing data temporarily [3]. In brief, our security goal is to allow clients to provide input data for training and inference in a way that is not vulnerable to confidentiality breaches.

B. ML WORKFLOW ATTACKS

Throughout the paper, we consider the black-box setting proposed by [38], which is as follows. When an adversary is given *black-box* access to a model, it means that it can query any input x and receive the predicted class probabilities $P(y|x)$ for all classes y. This allows the adversary to interact with the trained model without retrieving additional information, *e.g.*, computing the gradients. Ensuring security against attacks under this threat model entails including countermeasures against a wide array of attack vectors. Given this context, we now further detail pervasive attacks on the ML pipeline and establish their adversarial assumptions. A scheme summarizing these attacks is depicted in Figure 1.

1) ADVERSARIAL ATTACKS

These attacks are characterized by injecting malicious data samples to manipulate the model and disclose information about the original data used for training or inference purposes. Successful attacks in the literature require the attacker to have direct access to the training dataset (data poisoning, transfer-based, and gradient-based attacks), the model and gradients (gradient-based attacks), or the full results (*i.e.,* the output of inference) and class probabilities (score-based attacks) [19], [39].

2) MODEL EXTRACTION

These attacks aim at learning a close approximation to an objective function of the trained model. This approximation is based on the exact confidence values and response labels obtained by inference. To obtain the desired result, the attacker must know the dimension of the original training dataset (equation-solving attacks), the dimension of the decision trees, the data features, and the final confidence values (path-finding attacks) or hold actual samples from the training dataset (class-only attacks and data-free knowledge distillation (DFKD)) [5], [20], [40].

3) MODEL INVERSION AND MEMBERSHIP INFERENCE

These attacks target the recovery of values from the training dataset. Both consider an adversary that queries the ML system in a black-box fashion, and both are currently based on ML services, which publicly disclose their trained models and confidence values. In model inversion attacks, the adversary must have partial knowledge of the training dataset's features to infer and query the model with specific queries [5], [6]. Membership inference aims to test whether a specific data point d was used as training data and requires the adversary to know a subset of samples used to train the model (that does not contain d) [41].

4) RECONSTRUCTION ATTACKS

The goal of this attack is similar to that of membership inference. However, instead of testing for the existence of a specific data point, the adversary intends to reconstruct raw data used for training the model. To be successful, some attacks require the adversary to have model-specific information, namely feature vectors (*e.g.,* Support Vector Machines or K-Nearest Neighbor) [42], others only require black-box access to the model [43]. Nonetheless, in this setting, the attacker needs to have access to another dataset with the same distribution as the original training dataset (*i.e.,* local dataset and training dataset are subsets from a larger dataset).

TABLE 1. Comparison between state-of-the-art solutions and Soteria regarding the safety against ML attacks.

Attacks		Systems [12]	[13]	[23]	[22]*	SOTERIA
Adversarial	*Gradient-based*	✗	✗	✓	✗	✓
	Score-based	✗	✗	✓	✗	✓
	Transfer-based	✗	✗	✓	✗	✓
	Decision-based	✗	✗	✓	✗	✓
Model Extraction	*Equation-solving*	✓	✓	✗	✓	✓
	Path-finding	✓	✓	✗	✗	✓
	Class-only	✓	✓	✗	✗	✓
	DFKD	✓	✓	✗	✓	✓
	Model Inversion	✓	✓	?	✓	✓
	Reconstruction Attacks	✓	✓	✓	✓	✓
	Membership Inference	✗	✗	✗	✗	✓

*Data encryption is not provided on the open-source version.
✓ - Protected; ✗ - Non-protected; ? - Not disclosed.

5) SUMMARY

Unlike previous works [12], [13], [22], [23], which typically consider a small subset of ML attacks, our proposal aims at providing mechanisms that cover the full range of the exploits as mentioned above. Table 1 presents relevant state-of-the-art solutions, the security attacks covered by these, and the attacks addressed by Soteria. Intuitively, the resilience of our system is the result of combining several mechanisms, which are only partially ensured by other systems: *i)* authenticity verification of inputs excludes injections necessary for *adversarial attacks*; *ii)* isolation guarantees of our protocol ensure that malicious workers gather no additional information other than statistical data, an essential aspect for preventing most attacks, and *iii)* query input via a secure channel prevents the adversary from performing arbitrary queries to our system, which is also a central requirement for *model inversion* or *reconstruction attacks*. This resiliency is analyzed in detail in Section IV-F.

TEE-related security issues such as *side-channel* and *memory access pattern* attacks are considered orthogonal and complementary to our design goals. Indeed, mechanisms such as ObliviousRAM [44] can be layered over Soteria to address these at the cost of additional performance overhead [45].

IV. SOTERIA

In this section, we present the design goals of Soteria (§IV-A) and describe its architecture and flow of requests (§IV-B to §IV-D). The novel partitioned scheme is presented in detail in Subsection IV-E, while Subsection IV-F provides Soteria's security analysis. Finally, we overview the prototype of our solution (§IV-G).

A. DESIGN GOALS

Soteria is a distributed privacy-preserving machine learning system that avoids changing the architecture and processing flow of Apache Spark and MLlib, retaining its usability, scalability, and fault tolerance properties. It is built under the assumption that ML runtime performance can be improved if one can diminish the number of operations done inside secure enclaves. Thus, Soteria proposes a partitioning scheme to split the computation to be performed inside and outside these.

Soteria builds upon four core principles:

a: GENERAL APPLICABILITY FOR ML WORKLOADS

Soteria aims to offer an encompassing solution for several ML algorithms by relying on Apache Spark's MLlib.

b: PRIVACY-BY-DESIGN

In Soteria, sensitive data is only on plaintext inside the enclaves, being encrypted in the remaining workflow. This is achieved by resorting to trusted execution environments and encryption mechanisms that safeguard data privacy.

c: BALANCED OVERHEAD

Soteria offers a partitioning scheme that balances the imposed performance overhead of the privacy measures and the leakage of such a solution.

d: LOW INTRUSIVENESS

Both the processing flow of Apache Spark and the user's interaction with the system remain unchanged or require minor changes.

B. ARCHITECTURE AND FLOW

As depicted in Figure 4, Apache Spark's operational flow is as follows. Before submitting ML tasks (*e.g.*, model training, and/or inference operations) to the Spark cluster, users must load their local datasets and models to a distributed storage backend. Users can then submit ML processing tasks, specified as ML task scripts, to the Spark client, which is responsible for forwarding these scripts to the master node. At the master node, tasks are forwarded to the Spark Driver, which generates a Spark Context that then distributes the tasks to a set of worker nodes.

As workers may be executing different steps of a given task, they need to be able to transfer information among each other through the network. For instance, in a distributed ML training task, this information can contain model parameters that must be exchanged across workers. After finishing the desired computational steps, workers return their outputs to the master node, which merges the outputs and replies to the client.

Similar to the regular flow of Apache Spark, Soteria can be divided into two main environments or sides: the Soteria Client, trusted side, and the Soteria Cluster, untrusted side, (*e.g.,* cloud environment). Next, we describe the main modifications required by Soteria to the original Apache Spark's design, depicted in Figure 4 by the white dashed and solid line boxes.

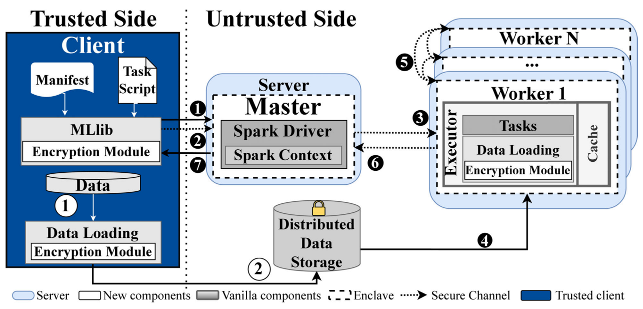


FIGURE 4. Soteria architecture and flow of operations.

C. CLIENT

Users use Soteria's client module for three main operations: i) loading data into the distributed storage backend, ii) sending ML training tasks to the Spark cluster, and iii) sending ML inference tasks to the Spark cluster. Soteria does not change how users typically specify and perform the previous operations. The only exception is that users need to provide additional information in a *Manifest* configuration file, as described next.

a: DATA LOADING

First, the user must specify the data to be loaded to the storage backend for the first operation. However, such data has to be encrypted before leaving the trusted user premises. This step is done by extending Spark's data-loading component with a transparent encryption module (Figure 4-①). This module encrypts the data being loaded into the distributed storage backend with a symmetric-key encryption scheme (Figure 4-②).

b: TASKS SUBMISSION

ML training and inference operations include two main files: the ML task script and the *Manifest* file. The transparent encryption module, also integrated within MLlib, is used to encrypt the ML task script (Figure 4-❶), which contains sensitive arguments (*i.e.,* model parameters) and the ML's workload processing logic, and to decrypt the outputs (*e.g.*, trained model or inference result) returned by Spark's master node to the client.

The *Manifest* file contains the libraries to be used by the ML task script, as well as the path at the storage backend where the training or inference data for that specific task is kept (Figure 4-❷). Briefly, and as explained in the following sections, this file ensures that different Spark components can attest to the integrity of libraries and data being used/read by them and cannot access other libraries or data that these are not supposed to.

The encryption module is in charge of securely exchanging the *Manifest file* and the user's symmetric encryption key with the SGX enclave on the master node (Figure 4-❶❷). This is done once, at the ML task's bootstrapping phase, and requires establishing a secure channel between the client and master's enclave. This channel guarantees the security and integrity of the user's encryption key and the *Manifest file*. At the same time, the encrypted ML task scripts can be safely sent via an unprotected channel.

With the previous design, sensitive data is only accessed in plaintext format at trusted user premises or inside trusted enclaves. This includes users' encryption keys, the information in the *Manifest file* and ML task scripts, as well as the final output.

D. CLUSTER

Training and inference ML task scripts are sent encrypted to Spark's master node to avoid revealing sensitive information. However, the node requires access to the plaintext information contained in these cryptograms to distribute the required computational load across workers. So, the Spark Driver and Context modules must be deployed in a secure SGX enclave where the cryptograms can be decrypted, and the plaintext information can be securely accessed. The cryptograms, however, can only be decrypted if the secure enclave has access to the user's encryption key, thus explaining why the key must be sent through a secure channel established between the client module and the enclave.

For inference operations, the master node also needs to access the distributed storage backend to retrieve the stored ML model. The user's encryption key is necessary, so the encrypted model is only decrypted and processed at the secure enclave. The *Manifest* file ensures that only the storage locations specified in the file are accessible to the master Node (Figure 4-❷).

After processing the ML task scripts, the master's enclave establishes secure channels with the enclaves of a set of workers to send the necessary computational instructions[2] along with the user's encryption key and *Manifest file* (Figure 4-❸). The user's encryption key is needed at the worker nodes so that these can read encrypted data (*e.g.,* train dataset or data to be inferred) from the storage backend while decrypting and processing it in a secure enclave environment (Figure 4-❹). Once again, the *Manifest file* prevents unwanted access to stored data. Furthermore, the enclaves at the worker nodes establish secure channels between themselves to transfer sensitive metadata information such as model training parameters (Figure 4-❺).

[2]The same metadata sent by a vanilla Spark deployment so that workers know the computational operations to perform.

Finally, after completing the desired computational tasks, the workers send the corresponding inference or training outputs to the master node through the established secure channel (Figure 4-❻). The master node then merges the partial outputs into the final result, which is done inside a trusted enclave, and sends it encrypted, with the user's encryption key, to the trusted client module (Figure 4-❼). At the latter, the result (*i.e.,* trained model or inference output) is decrypted by the transparent encryption module and returned to the user in plaintext.

E. PARTITIONED DESIGN

Soteria proposes a novel partitioning scheme, Soteria-P, that does fine-grained partitioning, of which operations execute inside and outside secure enclaves. Note that this partitioning is only done for ML operations executed at Spark worker nodes. The remaining operations performed at other Spark components (*i.e.*, master) are always executed inside trusted enclaves.

To better understand the novelty of our partitioning scheme, we first introduce a common state-of-the-art approach, Soteria-B, which is also supported by our system and is used in this paper as a security and performance baseline.

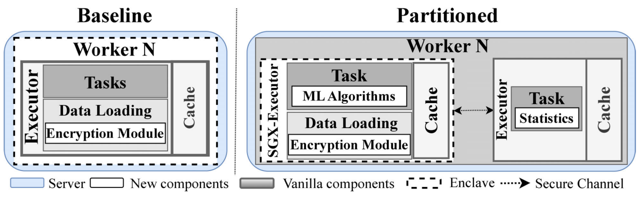


FIGURE 5. Comparison between Soteria-B and Soteria-P schemes.

a: SOTERIA BASELINE (SOTERIA-B)

In Soteria-B, all computation done by Spark workers is included in a trusted environment. Namely, the executor processes launched by each worker node are deployed inside an enclave, as depicted in Figure 5. Outside the enclave, data is always encrypted in an authenticated fashion, allowing the worker to decrypt and validate data integrity within the enclave.

b: SOTERIA PARTITIONING SCHEME (SOTERIA-P)

Our novel scheme is based on the observation that ML workloads are composed of different computational steps. Some must operate directly over sensitive plaintext information (*e.g.,* train and inference data and model), while others do not require access to this type of data and are just calculating and collecting general statistics about the operations being made. For instance, in a multiclass ML task, where the user may want to predict multiple classes, the evaluation of such an algorithm would need to measure the precision and the probability of each individual class. These measurements can be performed independently of other operations over sensitive information.

Therefore, Soteria-P decouples statistical processing, used for assessing the performance of inference and training tasks, from the actual computation of the ML algorithms done over sensitive plaintext information. This decoupling builds directly upon MLlib and refactors its implementation without requiring changes to how users submit ML tasks. As depicted in Figure 5, statistical processing is done by executor processes in the untrusted environment, while the remaining processing endeavors are done by another set of executors inside a trusted enclave (*SGX-executors*).

This decoupled scheme leads Soteria-P to reveal the following statistical information during the execution of ML workloads: the calculation of confidence results, which encompass the loss values and calculations of accuracy, precision, recall, and F1-scores, table summaries, ROC/AUC curves, and probability distributions for classes.

To provide a more concrete example, Algorithm 1 depicts the pseudo-code of a Linear Regression algorithm under the Soteria-P scheme and behaves as follows. The goal is to minimize the loss function, in this particular example, the *Root Mean Squared Error*. First, in the SGX-executor, an instance of Spark loads the dataset, creating a dataframe (X, y). This dataframe is further split into train and test data, $(X_{train}, y_{train}, X_{test}, y_{test})$. After this first pre-processing, an instance of a Linear Regression algorithm (lrM) is trained with the training data, and with the testing data, the first predicted values are inferred (P).

With these values, the Root Mean Squared Error (RMSE) is calculated at the non-secure executor (*rmse*). This computation is depicted in Algorithm 2, which intends to find the minimum error value. If no initial error is available, the algorithm returns the calculated RMSE. Otherwise, it returns the newly calculated RMSE (*newRMSE*). This is the only part of the computation within Soteria-P that is done outside trusted hardware, and thus, it is highlighted in orange.

After receiving the result, the SGX-executor continues the model training according to the number of maximum iterations (*maxIter*) defined by the user. In each iteration, it trains a new model (lrM') and predicts new values (P'), which are iteratively used to calculate a new RMSE. If the new RMSE is lower than the previous, the worker keeps the newly trained model. The master node then collects the results from the workers, maps and reduces the model parameters, and returns the encrypted final model to the client.

The statistical information that can be computed outside the secure enclave may differ and must be decided on a per-algorithm basis. For instance, Principal Component Analysis (PCA), a dimensionality reduction algorithm, performs its computation mostly on top of raw data. However, this algorithm performs an initial validation based on the number of features and principal components to understand if the remaining computation can be done. This validation is done by a statistical function defined in MLlib as *memorycost*, which in Soteria is done outside the enclave.

As with other examples, loss algorithms such as Root Mean Squared Error (RMSE) or Absolute Error (L1), used in Linear Regression, Alternating Least Squares, or Gradient Boosted Trees, are also offloaded from the secure enclaves. Similarly, the optimization algorithms, such as Expectation-Maximization (EM), implemented in Latent Dirichlet Allocation, the cost functions to evaluate the centroids of K-means, and the accuracy calculations in Naive Bayes are decoupled from the main algorithms to run outside of enclaves.

Algorithm 1 LinearRegression(DS, prms, lrM):
```
1: $(X, y)$ = Spark.load(DS)
2: $(X_{\text{train}}, y_{\text{train}}), (X_{\text{test}}, y_{\text{test}}) = (X, y)$.split()
3: $lrM$.fit($X_{\text{train}}, y_{\text{train}}$, prms)
4: $P \leftarrow lrM$.transform($X_{\text{test}}, y_{\text{test}}$)
5: rmse $\leftarrow$ Untrusted($y_{\text{test}}, P, \epsilon$)
6: for $i = 1$ to $maxIter$ do
7:     $lrM' \leftarrow lrM$.fit($X_{\text{train}}, y_{\text{train}}$, prms)
8:     $P' \leftarrow lrM'$.transform($X_{\text{test}}, y_{\text{test}}$)
9:     rmse$'$ $\leftarrow$ Untrusted($y_{\text{test}}, P'$, rmse)
10:    if rmse$'$ < rmse then
11:        $lrM \leftarrow lrM'$
12:        rmse $\leftarrow$ rmse$'$
13:    end if
14: end for
15: return $lrM$, rmse
```

Algorithm 2 Untrusted($y_{\text{test}}, y_{\text{new}}$, rmse):
```
1: $n \leftarrow$ len($y_{\text{test}}$)
2: newRMSE $\leftarrow \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{\text{test}_i} - y_{\text{new}_i})^2}$
3: if rmse = $\epsilon$ then
4:     return newRMSE
5: else
6:     return min(rmse, newRMSE)
7: end if
```

F. SECURITY
Our security goal is formally defined using the real-versus-ideal world paradigm, similar to the Universal Composability framework [46]. Succinctly, we prove that Soteria is indistinguishable from an idealized service for running ML scripts in an arbitrary external environment that can collude with a malicious insider adversary. We then use that abstraction to demonstrate how Soteria is resilient to real-world ML attacks. This idealized service is specified as a functionality parameterized with the input data, which executes the tasks described in the ML task script and returns the output to the client via a secure channel.

The full proof of Soteria-P and Soteria-B schemes can be found in Appendix A. The outline is as follows. The role played by the master node can be seen as an extension of the client, establishing secure channels, providing storage encryption keys, and receiving outputs. We follow the reasoning of [47] and replace the master node with a reactive functionality performing the same tasks. Similarly, each Soteria worker behaves simultaneously as a processing node and as a client node, providing inputs to the computation of other workers (*e.g.*, model training parameters). This enables us to do a hybrid argument, where worker nodes are sequentially replaced by idealized reactive functionalities executing their roles in the task script.

Finally, all processing is done in ideal functionalities, and all access to external storage is fixed by the ML task script and the Manifest file, so we can refactor the functionalities to process over hard-coded client data and replace the secure data storage with dummy encryptions. We have now reached the ideal world, where all ML computation is done in an isolated service, and all other protocol interactions are simulated, given the ML task script and Manifest files. Our analysis refers to Soteria-B and thus establishes the baseline security result when no computation is done outside the enclave (no leakage). The reasoning for Soteria-P is identical, with the caveat that statistical data is explicitly revealed as leakage in the ideal world.

1) SECURITY IMPLICATIONS OF STATISTICAL LEAKAGE

To show that our system is resilient against ML attacks, we must consider a common prerequisite for such attacks to be successful: the adversary must have black-box access to the model (as per definition on Section III-B). Our result implies that adversaries cannot infer internal data from the workers, and the secure channel between the client and master prevents adversaries from injecting queries into the system. This would intuitively suggest that our adversary is unable to perform queries to the model in a black-box fashion. However, Soteria-P has the aforementioned additional leakage of statistical information.

As such, a crucial security question to answer is: *how does statistical information relate to black-box model access, i.e., does the first imply the second in any way?* Specifically, our argument is by reduction: if an attack based on black-box access to the model occurs in Soteria-P, then the adversary must have been able to extract such access from the statistical information revealed. Indeed, a recent work by Chandrasekaran et al. [20] shows that, by only basing the attack on statistical information not directly associated with the raw data (i.e., training and validation dataset, ML model), one cannot perform attacks that disclose sensitive information from such raw information.

Given how statistical data depends on the underlying ML script, consider the concrete example provided in Algorithm 1. Here, the leakage can be defined as the sum of all data revealed to the untrusted execution, namely the set of predictive labels y_{test}, and the results of *maxIter* number of predictions after $lrM.\mathsf{fit}(X_{\text{train}}, y_{\text{train}}, \mathsf{prms})$. Concretely, the leakage l of an execution of Algorithm 1 can be defined as:

$$l = (y_{\mathsf{test}}, \sum^{maxIter} lrM.\mathsf{transform}(X_{\text{test}}, y_{\text{test}})) \qquad (1)$$

Equation 1 quantifies the amount of information explicitly revealed to the adversary of Soteria. As such, attacks requiring black-box access to the model can only occur if there exists an efficient algorithm that can take l and produce a sufficient approximation to lrM for black-box attacks to be conducted.

For the general case, extracting model access from statistical data is an ongoing area of research. However, current attacks suggest one is unable to do this in any successful way [20]. This supports our thesis that statistical values *are not sensitive information*, so their leakage does not expose our system to these types of attacks. From this, it follows that Soteria-P *scheme is resilient to any attack requiring black-box access to the model to succeed.*

2) RELATION TO ML ATTACKS

We now overview the four types of attacks referred to in Section III-B on a case-by-case basis. Appendix B contains a more in-depth analysis of these attacks.

Soteria achieves resistance against input forgery through authenticated data encryption. This means that the input dataset is authenticated by the data owner and explicitly defined in the *Manifest file*, allowing the SGX-executors to check the authenticity of all input data. Thus, no forged data is accepted for processing, which is necessary for performing any *adversarial attack*.

The secure channels between the TEE at the master node and the client ensure that an external adversary cannot observe legitimate query input/outputs and cannot submit arbitrary queries to Soteria. This query privacy feature is crucial to block illegitimate model access, which allows us to protect against *model extraction*, *model inversion*, *membership inference* as well as instances of *reconstruction attacks* that require black-box access to the model.

Finally, *reconstruction attacks* require additional knowledge about internal ML model data. Our security result shows that Soteria is indistinguishable from an idealized ML service, which does not reveal the trained model. This includes the critical feature vectors required for this attack, which cannot be inferred from confidence values and class probabilities alone. Alternatively, *reconstruction attacks* requiring black-box access to the model are strictly stronger, but this, as we have argued, is not possible only with knowledge of confidence values, class probabilities, ROC/AUC curves, and table summaries (the explicit leakage of Soteria-P, as defined in Appendix A).

G. IMPLEMENTATION

Soteria's prototype is built on top of Apache Spark 2.3.0 and implemented using both Java and Scala. Spark's data loading library was extended to include Soteria's transparent encryption module. The latter uses the AES-GCM-128 authenticated encryption cipher mode, providing data privacy and integrity guarantees.

Our prototype supports both Soteria-B and Soteria-P schemes. For Soteria-P's implementation, Spark's MLlib implementation was decoupled into two sub-libraries, one with the statistical processing (to be executed outside SGX) and another with the remaining ML computational logic (to be executed inside SGX).

Gramine 1.0 was used for the overall management of Intel SGX enclaves' life cycle, for specifying the computation (*i.e.*, internal Spark and MLlib libraries) to run at each enclave, and for establishing secure channels (*i.e.*, with the TLS-PSK protocol) between the enclaves at the master and worker nodes [31]. Soteria's *Manifest* file was also provided by Gramine.

V. EVALUATION

Our evaluation answers three main questions:

i) How does Soteria impact the execution time of ML workloads?
ii) How does the Soteria-P scheme compares, in terms of performance, with state-of-the-art approaches (*i.e.*, Soteria-B and SGX-Spark)?
iii) Can Soteria efficiently handle different algorithms and dataset sizes?

We split our evaluation into three different stages to present our results more clearly. In detail, Subsection V-A presents the methodology used for the testbed, Subsection V-B summarizes the main evaluation observations, and Subsection V-C analyzes these observations and provides key insights.

A. METHODOLOGY

1) ENVIRONMENT

The experiments use a cluster with eight servers, a 6-core 3.00 GHz Intel Core i5-9500 CPU, 16 GB RAM, and a 256GB NVMe. The host OS is Ubuntu 18.04.4 LTS, with Linux kernel 4.15.0. Each machine uses a 10Gbps Ethernet card connected to a dedicated local network. We use Apache Spark 2.3.0 and version 2.6 of the Intel SGX Linux SDK (driver 1.8). The client and Spark Master run on one server, while Spark Workers are deployed on the remaining seven servers. Moreover, due to hardware limitations, the memory of the enclaves was defined as 4GB.

2) WORKLOADS

As depicted in Table 2, we resort to the HiBench benchmark [24] for evaluating seven ML algorithms that are broadly used and natively implemented on top of MLlib. Further, the benchmark suite offers different workload sizes for each algorithm ranging from *Tiny* to *Gigantic* configurations. Next, we describe each of the algorithms in more detail.

a: ALTERNATING LEAST SQUARES (ALS)

ALS is an algorithm mainly used in recommendation systems, such as the ones used by Amazon and Facebook, among others. The algorithm is usually formulated as the factorization of an m x n matrix R, where m can be seen as the users and n the items. In this setting, the items' ratings given by the users are not all filled, and the ALS algorithm is used to fill those blank spaces, recommending new items to the users. According to [48], the time complexity of this algorithm is dependent on m, n, and hidden k dimension, which results in a complexity of $O((m+n)k^3 + mnk^2)$.

b: LINEAR REGRESSION (LR)

Linear Regression is broadly used in statistics to understand the correlation between one dependent variable and one or more independent variables. The outcome of this algorithm is supposed to follow a Gaussian distribution. The computation of a Linear Regression algorithm is based on matrix multiplications and inversions in a matrix of m x n, where m is the number of samples and n is the number of features, the time complexity equals $O(m * n^2 + n^3)$ [49].

c: K-MEANS CLUSTERING (K-MEANS)

K-means is an algorithm used in clustering tasks (unsupervised learning tasks). Giving a dataset R $(x_1, \ldots, x_m)$, data should be grouped into clusters. For each data point x_i, a feature vector is given with no labels y_i. Here, for each data point, the goal is to predict k centroids and the label y_i. For this matter, K-means has a quadratic time complexity, $O(n^2)$, with n being the size of the input dataset [50].

d: GRADIENT BOOSTED TREES (GBT)

With a fundamental basis in decision trees, GBT merges decision trees with gradient boosting. Overall, an ensemble of trees is used for predicting the target label. A first model is created and followed by a second model that should surpass the first one's results. Combining the best model with the previous models should minimize the error [51]. For each tree, there is a time complexity dependent on the number of training samples, n, and the number of labels, y, $O(nyn_{trees})$ [52].

e: LATENT DIRICHLET ALLOCATION (LDA)

As a dimensionality reduction algorithm, the goal of LDA is to lower the high-dimensional space, recurring to eigenvalue decomposition, increasing the usability of data and the feasibility of other algorithms. The complexity of this algorithm is given as $O(mnt + t^3)$, where m is the number of samples, n is the number of features, and $t = min(m, n)$ [53].

f: PRINCIPAL COMPONENTS ANALYSIS (PCA)

Being a classic method for dimensionality reduction, PCA, in a dataset of n observations and m variables, tries to

TABLE 2. Representation of each ML algorithm's tasks, time complexity, and data sizes for different workloads.

Algorithms	Tasks	Time Complexity	Workloads			
			Tiny	Large	Huge	Gigantic
ALS	RS	$O((m+n)k^3 + mnk^2)$ [48]	193KB	345MB	2GB	4GB
PCA	DR	$O(nm * min(n,m) + m^3)$ [55]	256KB	92MB	550MB	688MB
GBT	P	$O(n * y * n_{trees})$ [52]	36KB	46MB	92MB	183MB
LR	C + P	$O(m * n^2 + n^3)$ [49]	11GB	134GB	335GB	894GB
Naive Bayes	MC	$O(nm)$ [56]	-	-	5GB	-
LDA	DR	$O(mnt + t^3)$ [53]	-	-	2GB	-
K-means	Cl	$O(n^2)$ [50]	-	-	56GB	-

RS: Recommendation Systems **DR**: Dimensionality Reduction **P**: Prediction **C**: Classification **MC**: Multi-class Classification **Cl**: Clustering.

compute a target dimensionality y based on the eigenvalue decomposition of its covariance matrix [54]. The time complexity of PCA, as similar to other algorithms, depends on the size of matrix nxm, $O(nm * min(n, m) + m^3)$ [55].

g: SPARSE NAIVE BAYES (NAIVE BAYES)

Used for multi-class classification tasks, Naive Bayes is a highly efficient algorithm that only needs to compute one time over the dataset where, given a matrix of n training examples and m attributes, the time complexity is $O(nm)$ [56].

3) SETUPS AND METRICS

To validate Soteria's performance and the benefits of fine-grained differentiation of secure ML operations, we compare the implementations of our system with the Soteria-B and Soteria-P schemes. These setups are compared with a deployment of Apache Spark that does not offer privacy guarantees (Vanilla).

Moreover, we test SGX-Spark [22], a state-of-the-art SGX-based solution that protects both analytical and ML computation done with Apache Spark. It is designed to process sensitive information inside SGX enclaves, so it can be considered the most similar system to Soteria. However, SGX-Spark can only guarantee that *User Defined Functions (UDFs)* are processed in secure enclaves. This decision leaves a large codebase of Spark outside the protected memory region and, consequently, limits the users to only being able to execute privacy-preserving ML algorithms based on UDFs.[3]

For each experiment discussed in the next section, we include the average algorithm execution time and standard deviation for three independent runs. The *dstat* monitoring tool was used to collect the CPU, RAM, and network consumption at each cluster node.

B. PERFORMANCE OVERVIEW

Figure 6 shows the execution time of all the setups for the seven algorithms when using a huge-sized workload configuration. Moreover, Figures 7b, 7c, 7a and 7d present the performance evaluation for *PCA*, *GBT*, *ALS* and *LR* algorithms for different workload sizes. Next, we list our main observations to aid in the characterization of these results. Unless stated otherwise, the performance overhead values discussed in this section correspond to the number of times that the algorithm's execution time increases for a given setup when compared to the Vanilla Spark deployment results. *Observations 1* to *8* correspond to Figure 6, whilst *Observations 9* to *12* refer to Figure 7.

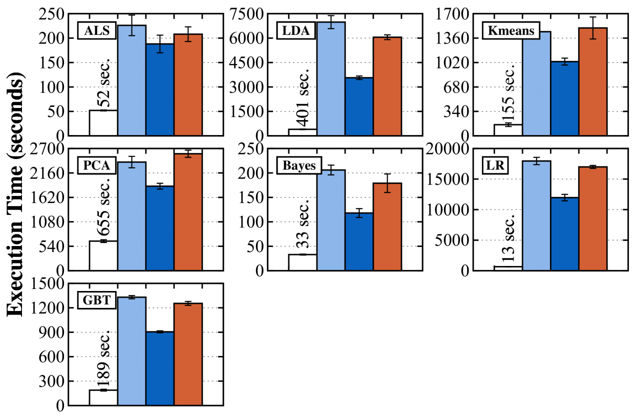


FIGURE 6. Execution time for each algorithm with *Huge* workload. The legend is as follows: □ Vanilla Spark; ▨ Soteria-B; ■ Soteria-P; 🟥 SGX-Spark.

a: OBSERVATION 1

Vanilla Spark's execution times for ALS, PCA, LR, and GBT algorithms are 55, 655, 657, and 189 seconds.

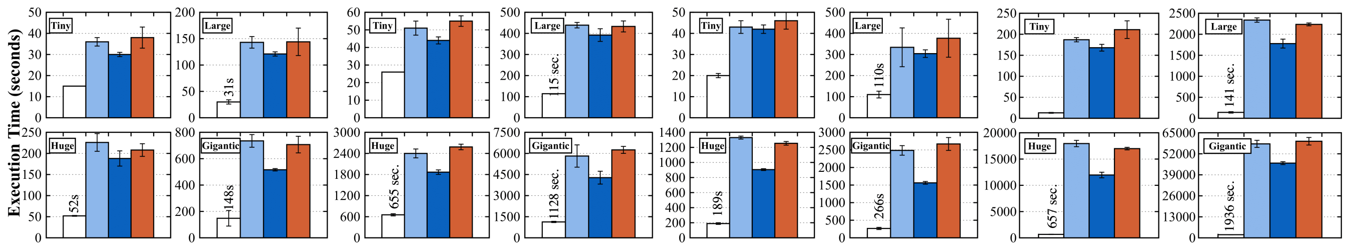
b: OBSERVATION 2

The execution time for ALS increases by 3.62x and 4.35x for Soteria-P and Soteria-B, respectively. SGX-Spark incurs an execution overhead of 4x. Thus, the three setups have similar results, requiring approximately 150 seconds more processing time than the vanilla deployment. Nevertheless, Soteria-P performs slightly better than the other two approaches.

c: OBSERVATION 3

For PCA, Soteria-B and Soteria-P have an execution overhead of 3.67x and 2.85x, while SGX-Spark increases

[3]Another similar solution to Soteria is Uranus [17]. Although Uranus is open-source, it currently does not provide enough information on how Apache Spark and all the tested ML algorithms could be deployed using this system, limiting a possible comparison against Soteria.



(a) ALS. (b) PCA. (c) GBT. (d) LR.

FIGURE 7. Runtime execution for PCA, GBT, ALS, and Linear Regression for *Tiny*, *Large*, *Huge* and *Gigantic* workloads. The legend is as follows: ☐ Vanilla Spark; 🟦 Soteria-B; 🟦 Soteria-P; 🟥 SGX-Spark.

the computational time by 3.95x. When compared to SGX-Spark, Soteria-P decreases the execution time by 12 minutes (27.8%).

d: OBSERVATION 4

For LR, Soteria-B and SGX-Spark exhibit an overhead of 27.31x, while Soteria-P reduces this value to 18.2x. This reduction of 29.6% allows Soteria-P to complete this workload 1.4 hours earlier.

e: OBSERVATION 5

With the GBT algorithm, Soteria-B shows similar execution times compared to SGX-Spark, with a 7.04x and 6.64x increase, respectively. Soteria-P outperforms both approaches, with an overhead of 4.79x, 27.8% less than SGX-Spark.

f: OBSERVATION 6

The LDA algorithm exhibits higher execution overhead of 17.40x, 8.89x, and 15.08x for Soteria-B, Soteria-P, and SGX-Spark setups, respectively. Soteria-P outperforms SGX-Spark by a difference of 41.5 minutes (*i.e.*, reduces execution time by 41%).

g: OBSERVATION 7

When compared with the vanilla deployment, Soteria-B increases execution time of KMeans by 9.37x and Soteria-P by 6.68x. SGX-Spark has an overhead of 9.7x, which means that, in comparison with Soteria-P, it requires an additional 468 seconds (7.8 minutes) to execute, *i.e.*, Soteria-P is 31% faster.

h: OBSERVATION 8

With *Huge* workload and Naive Bayes, Soteria exhibits an overhead of 6.24x for Soteria-B, which is higher than the 5.33x observed for SGX-Spark. Also, Soteria-P presents a lower overhead (3.58x) compared to SGX-Spark. The absolute difference in execution time between Soteria-P and Soteria-B is 88 seconds, while, with SGX-Spark, Soteria-P decreases execution time by 61 seconds (34%).

i: OBSERVATION 9

For *Tiny* and *Large* workloads with the PCA algorithm, Soteria performs similarly for our two schemes while outperforming SGX-Spark. With larger workload sizes, the overhead imposed by our solutions increases. However, Soteria continues to show better performance than SGX-Spark. Soteria-B has an overhead of 1.96x to 5.15x for *Tiny* and *Gigantic* workloads, whilst Soteria-P incurs an overhead of 1.72x to 3.79x. When compared with SGX-Spark, the results show an absolute difference of 4 seconds and 7 minutes (7%) for Soteria-B and 7 seconds and 33 minutes (19% and 31%), respectively, for Soteria-P.

j: OBSERVATION 10

Regarding the GBT algorithm, and the *Tiny* workload, the overhead of Soteria-B, Soteria-P, and SGX-Spark are similar. However, the difference between the three approaches is more visible when increasing the workload size. Soteria-P (*Tiny-2.13x and Gigantic-5.88x*) outperforms both approaches, while Soteria-B (*Tiny-2.18x, Gigantic-9.35x*) and SGX-Spark (*Tiny-2.3x, Gigantic-10.34x*) have similar results. Soteria-P surpasses SGX-Spark's execution time in the *Gigantic* workload by up to 41%.

k: OBSERVATION 11

With ALS, Soteria-P shows an execution time overhead of 2.04x and 3.28x, for the *Tiny* and *Gigantic* workloads, respectively. Soteria-P achieves lower overhead than Soteria-B and SGX-Spark for all dataset sizes, with the execution time decreasing by 8 seconds (9%) for the *Tiny* and 191 seconds (27%) for the *Gigantic* workloads.

l: OBSERVATION 12

For LR, with the *Tiny* workload, Soteria-B and Soteria-P increase execution time by 14.39x and 12.95x, respectively. As for the *Gigantic* workload, Soteria-B incurs an overhead of 30.04x and Soteria-P of 23.89x. Compared to SGX-Spark, Soteria-P decreases the execution time by 43 seconds for the *Tiny* workload and by 4.31 hours for the *Gigantic* workload (22.6%).

TABLE 3. Mean CPU usage (in %) for ALS, PCA, GBT, LR, Naive Bayes, LDA, and Kmeans algorithms with the *Huge* workload.

	CPU (%)			
Algorithms	Vanilla	SOTERIA-B	SOTERIA-P	SGX-Spark
ALS	19.7	23.8	22.6	23.1
PCA	19.1	25.5	24.5	27.1
GBT	23.0	28.3	27.5	26.7
LR	20.7	29.7	29.1	33.1
Naive Bayes	18.3	24.6	22.3	23.9
LDA	20.0	28.3	23.3	26.4
K-Means	18.6	26.9	24.5	29.0

m: OBSERVATION 13

The overall CPU usage for all the experiments is similar for Vanilla, Soteria-B, Soteria-P, and SGX-Spark. In more detail, Soteria-B with LR presents the upper-bound limit for CPU, showing an increase of 9% compared with Vanilla (20,7% CPU usage) as seen in Table 3. The standard deviation is never above 10%.

TABLE 4. Mean Memory usage (in %) for ALS, PCA, GBT, LR, Naive Bayes, LDA, and Kmeans algorithms with the *Huge* workload.

	Memory (GB)			
Algorithms	Vanilla	SOTERIA-B	SOTERIA-P	SGX-Spark
ALS	3.2	3.8	3.5	3.6
PCA	4.8	5.5	5.1	5.5
GBT	4.2	4.8	4.5	4.5
LR	5.4	6.1	5.9	7.7
Naive Bayes	4.0	4.5	4.3	4.7
LDA	3.6	4.5	4.3	4.6
K-Means	5.5	6.1	5.9	6.6

n: OBSERVATION 14

The mean memory consumption for each algorithm is similar across the different setups. As depicted in Table 4, Soteria-B presents an increase of ± 20% when compared to the Vanilla setup, which is explained by the extra memory needed when fetching data to the enclaves. The standard deviation is never above 9%.

TABLE 5. Mean Network usage (in MB/s) for ALS, PCA, GBT, LR, Naive Bayes, LDA, and Kmeans algorithms with the *Huge* workload.

	Network (MB/s)			
Algorithms	Vanilla	SOTERIA-B	SOTERIA-P	SGX-Spark
ALS	60.3	65.3	62.3	63.3
PCA	133.0	147.7	139.0	142.3
GBT	48.0	52.7	50.0	53.7
LR	121.7	130.7	133.7	130.0
Naive Bayes	95.0	100.7	103.3	101.3
LDA	80.7	87.7	89.3	93.0
K-Means	106.7	115.0	115.3	116.3

o: OBSERVATION 15

The network shows an upper-bound increase of 10% in Soteria-B with PCA due to extra encrypted data paddings being sent between Spark workers, as seen in Table 5. Vanilla Spark shows an upper-bound network usage of 133MB/s for the PCA algorithm. Interestingly, it is possible to see a slight increase (± 3 MB/s) in the network bandwidth usage on the Soteria-P scheme when compared to vanilla and Soteria-B, which is explained by the extra communication between SGX-executors and non-secure executors. The standard deviation is never above 10%.

p: OBSERVATION 16

Soteria does not impact the accuracy of ML workloads. For all experiments, we measured the corresponding accuracy metrics (*e.g.*, accuracy, root mean square error, or ROC). The results corroborate that both Soteria-B and Soteria-P show accuracy values similar to the vanilla Spark setup.

C. ANALYSIS

We now further analyze the experimental observations according to three topics, *i)* dataset size; *ii)* algorithm complexity; *iii)* size of trusted computing base (TCB).

1) DATASET SIZE

Figure 7 shows the performance degradation for the PCA, GBT, ALS, and Linear Regression algorithms with increasing dataset sizes. Results show that, for PCA, GBT, and ALS workloads with smaller datasets, Soteria-B and Soteria-P perform similarly. However, as the size of the datasets increases, more operations and data must be transferred to the SGX enclave, thus having a more noticeable toll on the overall performance. Indeed, the page swapping mechanism of SGX, which occurs due to its memory limitations, incurs a significant performance penalty [15], [57]. For example, when compared to the vanilla setup, the PCA algorithm overhead for Soteria-B varies between 1.96x for *Tiny* workload and 5.15x for *Gigantic* workload. While for Soteria-P, the execution time increases by 1.78x in the *Tiny* workload and 3.79x in the *Gigantic* workload. Linear Regression is the most expensive algorithm in terms of performance as it processes more data for the distinct workload sizes (Table 2). Compared with SGX-Spark, Soteria-P deals better with the data volume increase. Indeed, as seen in *Observations 9-12*, we reduce the execution time from 9% up to 41% when compared to SGX-Spark. Also, compared with our baseline, Soteria-P achieves up to 33% less execution time.

2) ALGORITHM COMPLEXITY

The execution times of ALS and LDA algorithms are very different even though their dataset size is similar. The computational complexity of each algorithm explains these results. For ALS, the synthetic workload data generated by the benchmark has a low hidden k dimension with a low ranking of 10, simplifying the required computation and decreasing execution time. For the LDA algorithm, the computational complexity, and consequently the execution time, are increased due to the higher number of dependencies between values at the generated synthetic workload data.

Observations 2 and 6 emphasize the performance of these two algorithms for a similar workload size. Like LDA, *Observations 3 and 9* show that PCA complexity and performance overhead increase with the processed data volume. Commonly classified as regression and classification algorithms, Bayes and GBT have similar performance, as seen in *Observation 8* and *Observation 5*. The data sizes of these two algorithms are entirely different, where GBT uses 91.7MB, and Bayes has 5.2GB. However, the Bayes algorithm iterates over the data only once, while GBT iterates over several decision trees to find its best model. Kmeans' performance is highly dependent on the chosen dataset size. This is also true for the Linear Regression algorithm (*Observations 4 and 12*).

3) SIZE OF TCB

The results discussed in Section V-B show that SGX-Spark outperforms Soteria-B for some of the evaluated algorithms (*Observation 2, 4-6, 8*). As SGX-Spark only protects UDFs, the performance overhead imposed by our solution's larger trusted computing base is naturally higher. Nevertheless, when compared to SGX-Spark, Soteria-B covers a wider range of machine learning attacks while keeping performance overhead below 1.59x. Indeed, for algorithms such as PCA and Kmeans, Soteria-B has a similar or slightly inferior execution time (*Observations 3 and 7*). This happens because, for these algorithms, both SGX-Spark and Soteria-B perform similar computations at the secure enclaves, while the UDF mechanism is not the most optimized choice for running some of these workloads.

Finally, due to the TCB reduction by our second scheme, Soteria-P consistently outperforms SGX-Spark and Soteria-B (*Observations 2-12*). The results show that this solution can reduce the training time by up to 41%, namely for the LDA algorithm with the *Huge* workload (*Observation 6*). Although the statistical information outsourced from secure enclaves differs for each algorithm, in general, Soteria-P outperforms Soteria-B from 1.1x to 1.9x (*Observation 6 and 10*). In detail, for GBT with the *Tiny* workload, the gains of Soteria-P over Soteria-B are low, 1.1x. The small difference in gain is due to the dataset size but also because the computation offloaded from the enclave (*i.e.*, Absolute Error for calculating the model's loss) is only executed once after the entire tree is constructed (*Observation 10*).

Conversely, when running LDA with both Soteria-B and Soteria-P with workload *Huge*, we observe a gain in performance by Soteria-P of 1.9x. In this particular case, the optimization of LDA, which resorts to the EM algorithm and is done outside of the trusted enclave, is performed in several stages of the model training process. This further shows the performance impact of offloading different amounts of statistical computation from secure enclaves.

4) DISCUSSION

The results show that Soteria-P outperforms other state-of-the-art approaches, namely SGX-Spark, for all the considered ML algorithms. Also, Soteria-P achieves better performance than the Soteria-B setup while offering similar security guarantees when considering distinct ML attacks (Section IV-F). This is made possible by filtering key operations to be done outside enclaves.

In detail, when compared to Soteria-B, Soteria-P reduces ML workloads' execution time by up to 37%. When compared with SGX-Spark, the execution time is reduced by up to 41%. Interestingly, for the LR algorithm using a *Gigantic* workload (*894GB*), Soteria-P decreases computation time by 4.3 hours and 3.3 hours when compared with SGX-Spark and Soteria-B, respectively. The performance overhead of Soteria-P for the four different algorithms ranges from 1.7x to 23.8x when compared to Vanilla Spark.

VI. RELATED WORK

This section is split into two subjects. First, we overview solutions for deploying generic applications on trusted execution environments (§VI-A). Then, we discuss solutions targeting privacy-preserving ML and analytics on TEEs (§VI-B).

A. TRUSTED EXECUTION ENVIRONMENTS

The challenges associated with deploying applications on TEEs have been studied since this technology became available. Here, we expose two alternative approaches that are the basis of Soteria generic architecture: *i)* full deployment of an unmodified application on TEEs (§VI-A1) and *ii)* partitioning of computation between trusted and untrusted environments (§VI-A2).

1) GENERIC APPLICATIONS ON TEES

The full deployment of applications inside a TEE is a challenging task, and solutions commonly resort to LibOses and similar approaches that shield the application on the TEE. While Panoply [62], SCONE [63], and Ryoan [64] ease the process of porting applications into secure enclaves, these solutions are still intrusive as they require changing the code base of targeted applications.

On the other hand, Gramine, SGX-LKL [65] and Occlum [66] allow users to run unmodified applications on SGX by specifying the application libraries that should be deployed on trusted enclaves. Enarx [67] and Veracruz [68] emerged as alternative solutions that provide users with a sandboxing environment to run their applications. Both Veracruz and Occlum are based on Apache Teaclave's confidential computing platform [69]

Soteria differs from these approaches because it does not aim to offer a generic solution to deploy applications at enclaves. Namely, our system is focused on ML workloads running on top of Apache Spark, which, as shown in Section IV, enables specific optimizations in terms of computation partitioning that allow for reducing performance overhead.

TABLE 6. Taxonomy of Related Work systems.

Systems	Cryptographic Primitives	Distribution	Algorithms	Spark-enabled	Availability
Chiron [12]	○	*C*	*DL, ML*	○	○
Myelin [13]	○,◑	*C*	*DL, ML*	○	○
SGX-BigMatrix [23]	○, ◐	*C*	*ML*	○	○
SGX-Spark [22]	○	*D*	*ML, A*	●	●
Uranus [17]	○	*D*	*A (mostly)*	◐	●
Opaque [16]	○,◐	*C*	*A*	○	●
Slalom [58]	○	*C*	*DL*	○	●
TensorScone [59]	○	*C*	*DL*	○	○
Privado [60]	○	*C*	*DL*	○	○
Occlumency [61]	○	*C, H*	*DL*	○	○
SOTERIA [21]	○	*D*	*ML*	●	●

Cryptographic Primitives	Distribution	Algorithms	Spark-enabled	Availability
○ - TEE	*C* - Centralized	*ML* - Machine Learning Algorithms	● - Yes	● - YES
◐ - OP	*D* - Distributed	*DL* - Neural Networks	○ - No	○ - No
◑ - DP	*H* - Collaborative	*A* - Analytics (SQL queries, etc)	◐ - Partially	

2) COMPUTATION PARTITIONING ON TEES

Recently, a different group of solutions have emerged to address the challenge of partitioning and selecting specific code from applications that should run in trusted execution environments.

Glamdring [70] proposes a static analysis tool that infers a partition between trusted and untrusted code in an application. It tries to achieve a balanced distribution of partitions to minimize the number of edges crossing between components. Another approach, Civet [71], focuses solely on partitioning Java applications. It provides an annotation-based approach for partitioning Java applications and ensures inter-object communication and consistent garbage collection across the partitioned components. Finally, Uranus [17] and Montsalvat [72] propose two automatic partitioning tools. Uranus [17] addresses the challenges of automatic partitioning between trusted and untrusted code in Intel SGX enclaves. However, unlike Glamdring, Uranus tries to enforce the trusted and untrusted code partition at runtime. Conversely, Montsalvat [72] provides an automatic partitioning tool for GraalVM images that automatically annotates trusted and untrusted code to be computed inside trusted execution environments.

Soteria is different from these solutions as it does not aim at proposing a generic partitioning tool. Namely, Soteria proposes a specific partitioning scheme optimized to balance the security and performance trade-offs for workloads using Apache Spark's MLlib.

B. PRIVACY-PRESERVING MACHINE LEARNING AND ANALYTICS ON TEES

Privacy-preserving ML solutions can be classified into four main groups based on the techniques being used: *i)* encryption-based [8], [9], [10], *ii)* secure multi-party computation [73], [74], *iii)* differential privacy [75], [76] and, *iv)* trusted execution environments (TEEs) [12], [13], [27], [58]. This paper is included in group *iv)*.

1) PRIVACY-PRESERVING ML WITH TEES

Chiron [12] enables training ML models on a cloud service without revealing information about the training dataset. Also, once the model is trained, only the data owners can query it. It supports launching multiple enclaves while each operates on different shards of training data, keeping the enclaves synchronized via a parameter server to ensure they all collaboratively converge to the same model. As such, this framework, besides providing a privacy-preserving approach, also provides data parallelism based on a parameter server for training the model faster. *Myelin* [13] offers a similar solution to *Chiron* while adding differential privacy (DP) and data oblivious protocols to the algorithms to mitigate the exploits from *side-channels* and the information leaked by the model parameters. Soteria differs from these works as it can cover both the training and inference phases while providing additional protection against *adversarial samples*, *reconstruction*, and *membership inference* attacks (Table 1).

In [27], five ML algorithms are re-implemented with data oblivious protocols.[4] Combined with TEEs, these protocols ensure strong privacy guarantees while preventing exploiting *side-channel* attacks that observe memory, disk, and network access patterns to infer private information. Unlike this solution, Soteria aims to support all ML algorithms built with MLlib transparently.

2) PRIVACY-PRESERVING ANALYTICS WITH TEES

TEEs have also been used to ensure privacy-preserving computation for general-purpose analytical frameworks [23]. In comparison to SGX-Spark [22], detailed in Section V-A, Soteria supports a broader set of algorithms (*i.e.*, any algorithm that can be built with the MLlib API), while protecting users from a more complete set of ML attacks (Table 1).

[4]A data oblivious algorithm guarantees that its control flow and memory access patterns are non-dependent of the input data [77].

Opaque [16], and Uranus [17] resort to SGX to provide secure general-purpose analytical operations while only supporting a restricted set of ML algorithms. Opaque combines SGX with oblivious protocols (OP) and requires the re-implementation and rewriting of the default Apache Spark UDF operators. This solution tries to mitigate the access pattern leakage by sorting and shuffling the entirety of the database, which represents an overhead of 1.6-46x when adding obliviousness to the system. Uranus is also based on porting UDF processing to SGX enclaves but includes a single ML workload. Differently, Soteria is targeted at ML and is not limited by UDF-based algorithms that, compared with MLlib-based ones, exhibit lower performance for some ML workloads [78]. Therefore, the design, implementation, and security requirements are distinct compared to Soteria.

Like the above solutions, SGX-Big Matrix [23] combines SGX and oblivious protocols (OP) to deliver a framework to process large encrypted datasets while hiding their access patterns. This solution proposed a new processing framework based on a generic language optimized for data analytic tasks. Unlike this work, Soteria relies on a widely-used framework, namely Apache Spark, maintaining its easy-to-use interface while increasing its privacy guarantees.

3) PRIVACY-PRESERVING DEEP LEARNING WITH TEES

TEEs have also been applied to the training and inference of deep neural networks [58], [59], [60], [61]. However, there is a substantial difference between the internals of ML and DL frameworks and algorithms, thus requiring significantly different privacy-preserving designs for each scenario. Since MLlib does not natively support DL workloads, the focus of Soteria is solely on ML algorithms.

4) SUMMARY

As shown in Table 1, current solutions related to Soteria (*i.e.*, addressing privacy-preserving ML) cover a smaller spectrum of ML exploits. The table does not include Opaque [16] and Uranus [17] since the first only covers SQL queries, while the second only supports a single ML workload. The works in this section are summarized in Table 6, categorized by cryptographic primitive, distribution, workload, applicability to Apache Spark, and availability. The type of distribution here refers to centralized, distributed, and collaborative computation. For instance, multi-party computation is depicted as collaborative computation, while outsourcing the computation to the cloud or other third-party infrastructure to perform the intended computation in a single server is named centralized, and approaches similar to the deployment setup of Apache Spark are depicted as distributed.

In detail, the works proposed by [12], [22], and [13] aim at protecting sensitive data's privacy during the model training stage while being resilient to model extraction attacks. However, these are vulnerable to adversarial and membership inference attacks because the dataset used for training is not encrypted at rest. Moreover, [22] is vulnerable to path-finding and class-only model extraction attacks because only UDF processing is offloaded to trusted enclaves.

Conversely, [23] supports encryption at rest for the dataset, as in Soteria, but it does not provide encryption for the trained model, thus being susceptible to model extraction and membership inference attacks.

To the best of our knowledge, Soteria is the first privacy-preserving ML framework that proposes an alternative TEE-based scheme (Soteria-P), which can improve the performance of training and inference workloads by reducing the number of operations done at secure enclaves. As such, Soteria is the first solution that covers a large spectrum of ML exploits and supports various ML algorithms while not changing how users build and run their algorithms within Spark MLlib.

Moreover, while orthogonal to this work, we acknowledge the increasing focus of the community on privacy-preserving DL-based solutions [58], [59], [60], [61]. To apply Soteria to these algorithms, one would need to profile which operations could be offloaded to run outside the secure enclaves. Also, one would need to support third-party APIs, as Apache Spark (v2.3.4) does not natively support DL workloads. We defer such tasks to future work.

VII. CONCLUSION

We propose Soteria, a system for distributed privacy-preserving ML. Our solution builds upon the combination of Apache Spark and TEEs to protect sensitive information being processed at third-party infrastructures during the ML training and inference phases.

The innovation of Soteria stems from a novel partitioning scheme (Soteria-P) that allows specific ML operations to be deployed outside trusted enclaves. Namely, we show that it is possible to offload non-sensitive operations (*i.e.*, statistical calculations) from enclaves while still covering a larger spectrum of ML attacks than in previous related work. Also, this decision enables Soteria to perform better than existing solutions, such as SGX-Spark, while reducing ML workloads execution time by up to 41%.

APPENDIX A
SOTERIA SECURITY PROOF

We now discuss the privacy-preserving security of the Soteria protocol. The goal is to reduce the security of our system to the security of the underlying security mechanisms, namely the isolation guarantees of Intel SGX and the bootstrapped secure channels, and the indistinguishability properties of encryption.

The security goal consists in demonstrating that Soteria ensures privacy-preserving machine learning. Concretely, this means that the real-world behavior displayed by Soteria is indistinguishable from the one displayed by an idealized functionality in the ideal-world, which simply computes over the task script and provides an output via a secure channel.

The only information revealed during this process is the length of I/O, the number of computation steps, and the access patterns to the external storage where data is kept.

Formally, this security goal is defined using the real-versus-ideal world paradigm, similar to the Universal Composability [46] framework.

We begin with a more formal description of our security model. Then, we present an intermediate result for ensuring the security of enclaves relying on external storage. We can finally specify the behavior of the Client, Master and Workers, and present the full proof.

A. FORMAL SECURITY MODEL

Our model considers external environment $\mathcal{Z}$ and internal adversary $\mathcal{A}$. Π denotes the protocol running in the real world, and $\mathcal{S}$ and $\mathcal{F}$ denote the simulator and functionality, respectively, running in the ideal-world. The real-world considers a Client C, a Master node M, and 2 Worker nodes W_1 and W_2. This is for simplicity, as the definition and proof can be easily generalized to consider any number of Worker nodes. We also consider global storage G, which is initialized by $\mathcal{Z}$ before starting the protocol. The Ideal functionality is parametrised by this external storage $\mathcal{F}$<G>, and will reveal the access patterns via leakage function $\mathcal{L}$.[5]

In the real-world, $\mathcal{Z}$ begins by providing public inputs to C in the form of (s, m), where s is the task script and m is the manifest detailing data in G to be retrieved.[6] The Client will then execute protocol Π, sending messages to M, W_1 and W_2. When the script is concluded, the output is provided to C, finally being returned to $\mathcal{Z}$. $\mathcal{A}$ can observe all communication between C, M, W_1, W_2 and G.

In the ideal world, (s, m) are provided to dummy Client C, which in turn forwards them to $\mathcal{F}$<G>. The functionality will simply run the protocol and forward the output to C, which in turn is returned to $\mathcal{Z}$. All the communication observed by $\mathcal{A}$ must be emulated by simulator $\mathcal{S}$, which receives (s, m), leakage $\mathcal{L}$ produced from the functionality interaction with storage G, and the output size.

Security is predicated on ensuring that $\mathcal{S}$ does not require any sensitive information (contained in G) to emulate the communication to $\mathcal{A}$. Given that we consider a semi-honest adversary, we can simplify the interaction with the system and instead discuss equality of views, as $\mathcal{Z}$ and $\mathcal{A}$ are unable to deviate the system from its expected execution. This is captured by the following definition.

Definition 1: Let Real denote the view of $\mathcal{Z}$ in the real-world, and let Ideal denote the view of $\mathcal{Z}$ in the ideal-world. Protocol Π securely realises $\mathcal{F}$ for storage G if, for all environments $\mathcal{Z}$ and all adversaries $\mathcal{A}$,

$$\mathsf{Real}_{\mathcal{Z},\mathcal{A},\Pi}(G) \approx \mathsf{Ideal}_{\mathcal{Z},\mathcal{A},\mathcal{S},\mathcal{F}}(G)$$

B. INTERMEDIATE RESULT

For convenience, Soteria does not require the Client to provide input data at the time of the ML processing, and instead, the Workers are given access to external storage from which they retrieve the data. When discussing the security in the context of secure outsourced computation for SGX, this is functionally equivalent to classical scenarios where the Client provides these inputs via a secure channel (Theorem 3 in [79]). The reasoning is simply that if a protocol securely realizes a functionality with a given input provided via a secure channel, then the same functionality can be securely realized with the same input fixed in an external storage, securely accessed by the enclave.

```
Algorithm Setup(i, m)<G>:
k ←$ Θ.Gen()
c ←$ Θ.Enc(k, i)
G[m] ← c
Return (m, k)
```

FIGURE 8. Secure external storage setup.

Consider a protocol Π_1 that securely realises some functionality $\mathcal{F}$ with simulator $\mathcal{S}_1$ according to Theorem 3 of [46]. We construct protocol Π_2 built on top of this secure protocol Π_1, where input data is pre-established and provided to the enclave via an initial Setup stage where inputs are stored in an encrypted fashion (Figure 8 describes a simplified version of the process for a single entry). Inputs to Π_2 are exactly the same as those for Π_1, but instead of being transmitted via the secure channel established with Attested Computation, they are retrieved from storage using a key sent via the same channel. The Client-server communication increases by a constant (the key length), which can be trivially simulated, and the rest of the input can be simulated in a similar way using the IND-CPA properties of Θ. This protocol behavior will be key for all Soteria Workers. Our theorem is as follows.

Theorem 1: Let Π_1 be a protocol that securely realises functionality $\mathcal{F}$ according to Theorem 3 in [79]. Then Π_2, constructed as discussed above, securely realises $\mathcal{F}$ according to Definition 1.

Proof. To demonstrate this result, we construct simulator $\mathcal{S}_2$ using $\mathcal{S}_1$, then argue that, given that $\mathcal{S}_1$ is a valid simulator for the view of Π_1, then the simulated view must be indistinguishable from the one of the real world of Π_2.

We begin by deconstructing $\mathcal{S}_1$ in two parts: $\mathcal{S}_1.\mathsf{AC}()$ will produce the view for establishing a secure channel, while $\mathcal{S}_1.\mathsf{Send}(l)$ will produce a simulated view of Client inputs, given their length. In turn, our simulator will share the same functions, but also include a third $\mathcal{S}_2.\mathsf{Get}(l)$ to simulate information being retrieved from G, given its length. Our simulator is depicted in Figure 9.

[5]Reasoning for the security of Soteria-P instead would only require this function to also reveal statistical data to the simulator, which we consider to be non-sensitive.

[6]Soteria Clients are trusted. As such, we assume (s, m) to both be *valid*, in the sense that they are correct ML scripts and data sets in G, and thus can be interpreted by ideal functionality $\mathcal{F}$.

```
Algorithm AC()
k ←$ Θ.Gen()
Return $\mathcal{S}_1$.AC()

Algorithm Send($l$)
Return $\mathcal{S}_1$.Send($l$)

Algorithm Get($l$)
$i \leftarrow \{0\}^l$
c ←$ Θ.Enc(k, $i$)
Return c
```

FIGURE 9. Simulator for Π_2.

The view presented to $\mathcal{A}$ is composed of three different types of messages:

- Messages exchanged during the secure channel establishment are exactly the same as in Π_1. Thus they remain indistinguishable from Π_2.
- Outputs received via the secure channel follow the exact same simulation strategy than Π_1, and thus are indistinguishable from Π_2.
- Messages produced from G in Π_2 are encryption of data in $G[m]$, while the values presented by $\mathcal{S}_2$ are dummy encryptions with the same length. We can thus reduce the advantage of $\mathcal{A}$ to distinguish these views to the advantage of the same adversary to attack the IND-CPA guarantees of encryption scheme Θ, which is negligible.

As such, if $\mathcal{S}_1$ is a valid simulator for Π_1 to $\mathcal{A}$, then the view presented by $\mathcal{S}_2$ must also be indistinguishable for Π_2 to $\mathcal{A}$.

Let

$$\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) \tag{2}$$

$$= \Pr[\mathsf{Real}^{G}_{\mathcal{Z},\mathcal{A},\Pi_2} \Rightarrow \mathsf{T}] - \Pr[\mathsf{Ideal}^{G}_{\mathcal{Z},\mathcal{A},\mathcal{S}_2,\mathcal{F}} \Rightarrow \mathsf{T}] \tag{3}$$

To conclude, we have that, for negligible function μ,

$$\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_2,\mathcal{S}_2,\mathcal{F}}(G) = \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_1,\mathcal{S}_1,\mathcal{F}}() + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\Theta,\mathcal{A}}() \tag{4}$$

$$\leq \mu() \tag{5}$$

and Theorem 1 follows.

C. SOTERIA *CLIENT, MASTER AND WORKERS*

The Soteria components follow standard methodologies for ensuring secure outsourced computation using SGX. As such, and given the complexity of ML tasks described in the script, we consider the following set of functions.

Secure channels are established with enclaves. We define $\mathsf{init}(P)$ as the bootstrapping process, establishing a channel with participant P. This produces an object that can be used to send and receive data via send and $\mathsf{receive}$. Untrusted storage is not protected with secure channels and can be accessed using the call $\mathsf{uGet}(G, m)$, which retrieves data from G considering manifest file m. Concretely, this is achieved using the open-source library Gramine, which we assume to implement this mechanism correctly. Finally, the script s defines the actual computation that must be performed by the system and will be executed collaboratively with both Workers. As such, we define s as a stateful object with the main method $\mathsf{Run}(\mathsf{id}, i_1, i_2)$, where input id is the identifier of the Worker, i_1 is input from storage and i_2 is intermediate input (*e.g.*, model parameters), returning (o_1, o_2), where o_1 is the (possibly) final output, and o_2 is the (optional) intermediate output for dissemination. For simplicity, we also define method $\mathsf{Complete}$ that returns T if the task is complete, or F otherwise.

The Soteria components can be analyzed in Figure 10 and are as follows. The Client C (left of Figure 10) simply establishes the channel with M, sends the parameters (manifest file, task script, and storage key), and awaits computation output. Observe that we assume that the key k has been previously initialized and that the actual data has been previously encrypted in G using it. The Master M (middle of Figure 10) will receive the parameters from C and establish channels with W_1 and W_2, forwarding them the same parameters and awaiting computation output. When it arrives, it is forwarded to the Client.[7] Worker W_1 (right of Figure 10) receives the parameters from M and starts processing the script: retrieves encrypted data from G, decrypts, processes and exchanges intermediate results with the other Worker. When the script is concluded, it returns its output to M. The behavior of W_2 is the same, but the connection is established instead with W_1.

```
Algorithm $C(m, s, k)$
sc ← init($M$)
sc.send($m, s, k$)
$o$ ← sc.receive()
Return $o$

Algorithm $M()$
sc$_c$ ← init($C$)
($m, s, k$) ←
sc$_c$.receive()
sc$_1$ ← init($W_1$)
sc$_1$.send($m, s, k$)
sc$_2$ ← init($W_2$)
sc$_2$.send($m, s, k$)
$o_1$ ← sc$_1$.receive()
$o_2$ ← sc$_2$.receive()
sc$_c$.send(($o_1, o_2$))

Algorithm $W_1()$
$m \leftarrow \epsilon$
sc$_m$ ← init($M$)
($m, s, k$) ← sc$_m$.receive()
sc$_w$ ← init($W_2$)
While !$s$.Complete:
  $c$ ← uGet($G, m$)
  $i$ ← Θ.Dec($k, c$)
  ($o, m$) ←$ $s$.Run($W_1, i, \epsilon$)
  sc$_w$.send($m$)
  $m$ ← sc$_w$.receive()
sc$_m$.send($o$)
```

FIGURE 10. Soteria Components. Client *C* (left), Master node *M* (middle), and Worker node 1 *W* (right).

D. *FULL PROOF*

Given the description of Soteria components in Figure 10, the Soteria protocol Π_{xyz} is straightforward to describe. Considering a pre-encrypted storage G, the Client C, Master M, and Workers W_1, W_2 execute following their respective specifications. Our theorem for the security of Soteria is as follows.

Theorem 2: Π_{xyz}, assuming the setup of Figure 8 and constructed as discussed above, securely realises $\mathcal{F}$ according to Definition 1.

The proof is presented as a sequence of four games. We begin in the real-world, and sequentially adapt our setting until we arrive in the ideal world. We then argue that all steps up to that point are of negligible advantage to $\mathcal{A}$, and thus the views must be indistinguishable to $\mathcal{Z}$.

The first is a simplification step, where, instead of using a single storage G, we slice the storage to consider G_1 and G_2. Figure 11 represents this change. This enables us to split the execution environment of W_1 and W_2 seamlessly and can be done trivially since manifest file m by construction will

[7] In the actual protocol, the Master has additional steps to process the output. We describe it like this for simplicity, as it does not change the proof.

```
Algorithm $W_1()$
$m \leftarrow \epsilon$
$\mathsf{sc}_m \leftarrow \mathsf{init}(M)$
$(m, s, k) \leftarrow \mathsf{sc}_m.\mathsf{receive}()$
$\mathsf{sc}_w \leftarrow \mathsf{init}(W_2)$
While $!s$.Complete:
    $c \leftarrow \mathsf{uGet}(G_1, m)$
    $i \leftarrow \Theta.\mathsf{Dec}(k, c)$
    $(o, m) \leftarrow\$ s.\mathsf{Run}(W_1, i, \epsilon)$
    $\mathsf{sc}_w.\mathsf{send}(m)$
    $m \leftarrow \mathsf{sc}_w.\mathsf{receive}()$
$\mathsf{sc}_m.\mathsf{send}(o)$
```

```
Algorithm $W_2()$
$m \leftarrow \epsilon$
$\mathsf{sc}_m \leftarrow \mathsf{init}(M)$
$(m, s, k) \leftarrow \mathsf{sc}_m.\mathsf{receive}()$
$\mathsf{sc}_w \leftarrow \mathsf{init}(W_1)$
While $!s$.Complete:
    $c \leftarrow \mathsf{uGet}(G_2, m)$
    $i \leftarrow \Theta.\mathsf{Dec}(k, c)$
    $(o, m) \leftarrow\$ s.\mathsf{Run}(W_1, i, \epsilon)$
    $\mathsf{sc}_w.\mathsf{send}(m)$
    $m \leftarrow \mathsf{sc}_w.\mathsf{receive}()$
$\mathsf{sc}_m.\mathsf{send}(o)$
```

FIGURE 11. Soteria Workers with split storage.

never require different Workers to access the same parts of G. Since these two games are functionally equal, the adversarial advantage is exactly 0.

The second step is a hybrid argument, where we sequentially replace both Workers by ideal functionalities performing partial steps of the ML script. Concretely, we argue as follows. Replace W_1 with a functionality for its part of the ML script $\mathcal{F}_{W1}$, according to Definition 1. From Theorem 1, we can establish that this adaptation entails negligible advantage to $\mathcal{A}$ provided that the protocol without external access realizes the same functionality. However, this is necessarily the case, as it follows the exact structure as the constructions in [79]. We can repeat this process for W_2.[8] As such, using the intermediate result, we can thus upper bound the advantage adversary to distinguish these two scenarios by applying twice the result of Theorem 1.

The third step replaces the Master with an ideal functionality $\mathcal{F}_M$ that simply forwards requests to the Worker functionalities. This one follows the same logic as the previous one, without requiring external storage, as the protocol also follows the exact structure as the constructions in [79].

In the final step, we have 3 functionalities ($\mathcal{F}_M$, $\mathcal{F}_{W1}$, $\mathcal{F}_{W2}$) playing the roles of (M, W_1, W_2), respectively. We finalize by combining them into a single functionality $\mathcal{F}$ for ML script processing. This can be done by constructing a big simulator S that builds upon the simulators for the individual components (S_M, S_{W1}, S_{W2}). The simulator S behaves as follows:

- Run S_M to construct the communication trace that emulates the first part of $\mathcal{F}$.
- Run the initial step of S_{W1} and S_{W2} to construct the communication trace for establishing secure channels between Workers and Master.
- Call leakage function $\mathcal{L}$ to retrieve the access patterns to G. Use the result to infer which part of the storage is being accessed, and run S_{W1} or S_{W2} to emulate the computation stage.

Given that the view produced by S is exactly the same as the one provided by the combination of S_M, S_{W1}, and S_{W2}, the adversarial advantage is exactly 0.

[8]Again, this technique extends for an arbitrary number of Workers. N number of Workers would just require us to adapt the multiplication factor in the final formula, which would still be negligible.

We are now exactly in the ideal world specified for Definition 1.

Let

$$\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) \tag{6}$$

$$= \Pr[\mathsf{Real}^{G}_{\mathcal{Z},\mathcal{A},\Pi_{\mathsf{xyz}}} \Rightarrow \mathsf{T}] - \Pr[\mathsf{Ideal}^{G}_{\mathcal{Z},\mathcal{A},\mathcal{S},\mathcal{F}} \Rightarrow \mathsf{T}] \tag{7}$$

To conclude, we have that, for the negligible function μ,

$$\mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi,\mathcal{S},\mathcal{F}}(G) = 2 \cdot \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_{W1},\mathcal{S}_{W1},\mathcal{F}_{W1}}(G) \tag{8}$$

$$+ \mathsf{Adv}^{\mathsf{Dist}}_{\mathcal{Z},\mathcal{A},\Pi_M,\mathcal{S}_M,\mathcal{F}_M}() \tag{9}$$

$$\leq \mu() \tag{10}$$

and Theorem 2 follows.

APPENDIX B
ML WORKFLOW ATTACKS

This section presents the attacks in Section III-B in further detail and argues in which circumstances Soteria is secure against each attack. First, we will describe a general adversarial model against Soteria that follows the security restrictions justified in Appendix A. Then, we will present an experiment that captures what constitutes a valid attack under each definition, as described in Section III-A. For each attack, we consider our protocol to be secure if we can demonstrate that one cannot rely on a valid adversary against the experiment of said attack under the constraints of Soteria. In some instances, this will depend on known attack limitations, which we detail case-by-case.

E. ATTACKER AGAINST SOTERIA

Our goal is to present a model that details the conditions in which these attacks are possible. As such, it must be both generic to capture the multiple success conditions of attacks, as well as expressive, so that it can be easy to relate to each specific attack.

In this definition, we will also consider an adversary that can play the role of an honest client and thus will have black-box access to the produced model. We stress that, in practice, this will not be the case in many circumstances. In those scenarios, since queries to the model are made via a secure channel, an external adversary is unable to arbitrarily request queries to the model without causing it to abort. This means that any attack that requires black-box access to the model is not possible if Soteria assumes external adversaries.

```
Game $\mathsf{AdvXYZ}_{\mathcal{A},\Pi_{\mathsf{xyz}}}(G, s)$:
$(G') \leftarrow\$ \mathcal{A}_1(G, s)$
$(m, l) \leftarrow \Pi_{\mathsf{xyz}}(G', s)$
$r \leftarrow\$ \mathcal{A}_2^{m}(l)$
Return $\mathsf{Success}(G, s, m, r)$
```

FIGURE 12. Adversary interacting with Soteria.

Let Π_{xyz} denote the full training protocol of Soteria. It receives external storage G and task script s as inputs, and produces a model m, which can then be queried. Based on the security result of Appendix A, the interaction of an adversary with our system can be described in Figure 12. The adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ can first try and manipulate the input dataset G to G'. This is then used for Π_{xyz}, which will produce the model m and the additional leakage l (Soteria-B has no additional leakage, so $l = \epsilon$). Finally, the adversary can interact black-box with the model until a conclusion r is produced. This will be provided to a $\mathsf{Success}$ predicate, which will state if the attack was successful. This predicate is specific to the attack and allows us to generally describe attacks such as *adversarial samples*, where the goal is to make the resulting model deviate, as well as *membership inference*, where the goal is to retrieve information from the original dataset.

Remark Observe that $\mathcal{A}_1$ and $\mathcal{A}_2$ do not share state. This is because they play different roles within this experiment: the first influence the system by attempting to manipulate the training dataset G, while the second interacts with the model m and leakage l to try and extract information. Indeed, our first step will be to show that $\mathcal{A}_1$ is unable to rely on G' to meaningfully convey any additional knowledge gained by observing (G, s).

F. DATASET MANIPULATION

Dataset manipulation attacks are defined by an adversary with the capability of inserting, removing or manipulating dataset information. These align with the setting considered for attacks via *adversarial samples*. Figure 13 is an experiment that describes what constitutes a successful attack for dataset manipulation. The adversary $\mathcal{A}$ is given full knowledge of G,[9] and must produce an alternative input dataset G'. We then train the model (protocol Π) over that data to produce model m, and the adversary is successful if said model satisfies some attack success criteria $\mathsf{T}/\mathsf{F} \leftarrow \mathsf{Success}$.

```
Game DSetMan_{A,Π}(G, s):
  $G' \leftarrow\$ \mathcal{A}(G, s)$
  $m \leftarrow \Pi(G', s)$
  Return $\mathsf{Success}(G, s, m)$
```

FIGURE 13. Model for dataset manipulation attack.

We now argue that the integrity guarantees of the authenticated encryption used by our external storage G ensure that these attacks do not occur for Π_{xyz}. We do this by showing that any adversary that performs an *adversarial samples* attack on Π_{xyz} can be used to construct a successful attack on the security of the authenticated encryption scheme. First, observe that no attack can be successful if the adversary makes no changes on the input dataset, so if $G = G'$, then $\mathsf{F} \leftarrow \mathsf{Success}$. Furthermore, if Π_{xyz} aborts, then no model is produced, so it naturally follows that the attack is unsuccessful $\mathsf{F} \leftarrow \mathsf{Success}$.[10]

As such, the only cases in which $\mathsf{T} \leftarrow \mathsf{Success}$ are those in which $G' \neq G$ and Π_{xyz} do not abort. But this means that the adversary was able to forge an input that correctly decrypts, breaking the integrity of the underlying encryption scheme. Since the security guarantees of authenticated encryption ensure that the probability of existing such an adversary is negligible, the probability of such an attacker in Soteria will also be negligible.

G. BLACK-BOX ATTACKS

All the remaining attacks, with the exception of some *reconstruction attacks*, follow a similar setting, where the adversary leverages a black-box access to the trained model, depicted in the experiment of Figure 14. We begin by running Π to produce our model and leakage and then run an additional procedure $\mathsf{Extract}$ to obtain additional information from the original dataset, which cannot be retrieved by simply querying the model. This procedure captures whatever knowledge regarding the underlying ML training might be necessary for the attack to be successful (e.g., information about data features). We then provide this additional information to the adversary and give it black-box access to the model. The success criteria depends on the specific attack and is validated with respect to the original dataset, model, and the task script being run. E.g., for *model extraction* attack, the goal might be to present a model m' that is similar to m, evaluated by the $\mathsf{Success}$ predicate.

For simplicity, we first exclude all attacks for an external adversary, which does not have black-box access to the model of Soteria. This is true if we can show that one cannot emulate black-box access to the model using confidence values and class probabilities. Albeit an interesting research topic, current attacks are still unable to do this in an efficient way [20]. We now go case-by-case, assuming an adversary can play the role of a genuine client in our system.

Our arguments for Π_{xyz} depend on being able to rely on a successful adversary $\mathcal{A}$ of Figure 14 to perform the same attack in Figure 12. As such, the security of our system will depend on the amount of additional information z, on how it can be extracted from the view of the adversary of Soteria.

- For *membership inference*, *reconstruction attacks* based on black-box access to the model, *model inversion*, and *model extraction* via *data-free knowledge distillation*, no additional information z is required. This means that any successful adversary in Figure 14 will also be successful in Figure 12, meaning that both for Soteria-B and Soteria-P are vulnerable. Preventing these attacks requires restricting access to the model to untrusted participants.
- *Class-only* attacks for *model extraction* require additional knowledge from the dataset. Specifically, z must

[9] Realistically, an attacker would have less information, but for our purposes we can go for the worst case and give him all the information regarding the computation and its input.

[10] The only circumstance in which this could be considered a successful attack was if the goal was to perform a denial-of-service attack, which we consider to fall outside the scope of an adversarial sample attack.

TABLE 7. Summary of attacks against Soteria. ✓ means Soteria is resilient to the attacks, ✗ means Soteria is vulnerable to the attacks, and {X} means Soteria is secure if argument {X} is also true.

		External		Client	
		SOTERIA-B	SOTERIA-P	SOTERIA-B	SOTERIA-P
Adversarial Samples		✓	✓	✓	✓
Reconstruction	*WB*	✓	{1a}	{1b}	{1c}
	BB	✓	{2}	✗	✗
Membership Inference		✓	{2}	✗	✗
Model Inversion		✓	{2}	✗	✗
Model Extraction	*Equation*	✓	{2}	{3a}	{3b}
	Path	✓	{2}	{4a}	{4b}
	Class	✓	{2}	{5a}	{5b}
	DFKD	✓	{2}	✗	✗

contain concrete training dataset samples. This means that to leverage such an adversary $\mathcal{A}$, one must first be able to use $\mathcal{A}_2^m(l)$ to extract such a z. This exactly matches the setting of *model inversion* attacks. This means that Soteria is vulnerable to *class-only* attacks for *model extraction* in Soteria-B or Soteria-P if there is also an efficient attack for *model inversion* in Soteria-B or Soteria-P, respectively.

- *Equation-solving model extraction* requires knowledge of the dimension of the training dataset G. This is additional information z that is not revealed by querying the model, which means no adversary $\mathcal{A}_2^m(\epsilon)$ can retrieve z, and thus Soteria-B is secure against said attacks. However, combining public data with confidence values might allow for $\mathcal{A}_2^m(l)$ to extract a sufficient z to perform the attack, which makes Soteria-P vulnerable to *equation-solving model extraction*.
- *Path-finding model extraction* attacks require information regarding leaf count, tree depth and leaf ID. As such, all this must be encapsulated in z. Reference [5] suggests that such information is not retrievable from only black-box access to the model [5], which means no adversary $\mathcal{A}_2^m(\epsilon)$ can produce z and thus Soteria-B is secure. However, this is information that can be extracted from confidence values, which suggests that an efficient adversary $z \leftarrow\$ \mathcal{A}_2^m(l)$ is likely to exist, and thus Soteria-P is vulnerable to such attacks under these assumptions.

We can generalize the security of our system to these types of attacks as follows. If no additional information z is required, then Π_{xyz} is vulnerable to an adversary that can play the role of an honest client. If z can be extracted from black-box access to the model, then we can still rely on said adversary to attack Π_{xyz}. Otherwise, Soteria-B is secure, as no additional information is leaked. Furthermore, the security of Soteria-P will depend on whether one can infer z from l and from the black-box access to the model. Concretely, if we can show that no (efficient) function F exists, such that $z \leftarrow\$ \mathsf{F}^m(l)$, then Π_{xyz} for leakage l is secure against attacks requiring additional data z.

```
Game BlackBox$_{\mathcal{A},\Pi_{\mathsf{xyz}}}(G, s)$:
$m \leftarrow \Pi(G, s)$
$z \leftarrow\$ \mathsf{Extract}(G, s)$
$r \leftarrow\$ \mathcal{A}^m(z)$
Return $\mathsf{Success}(G, s, m, r)$
```

FIGURE 14. Model for black-box attacks.

H. WHITE-BOX ATTACKS

White-box attacks capture a scenario where an adversary requires white-box access to the model. These align with the setting of *reconstruction attacks* that explicitly require white-box access to the model. Figure 15 is an experiment that describes what constitutes a successful *reconstruction attack* in this context. We begin by training the model (protocol Π) over the original dataset to produce the model. We then provide the trained model directly to the adversary, which will reconstruct raw data r. Finally, the success of the attack is validated with respect to the original dataset.

```
Game WhiteBox$_{\mathcal{A},\Pi_{\mathsf{xyz}}}(G, s)$:
$m \leftarrow \Pi(G, s)$
$r \leftarrow\$ \mathcal{A}(m)$
Return $\mathsf{Success}(G, s, m, r)$
```

FIGURE 15. Model for white-box attacks.

We now argue that these attacks do not occur for Π_{xyz}, as long as it is not possible to extract the model from the confidence values and from black-box access to the model. This is because the attacker of Figure 15 receives explicitly the model m, whereas the adversary $\mathcal{A}_2$ in Figure 12 receives the confidence values in l, and black-box access to the model. To rely on such an attacker, $\mathcal{A}_2$ must therefore be able to produce input m from its own view of the system. As such, relying on such an adversary implies there is an efficient way $m \leftarrow\$ \mathcal{A}_{\mathsf{bb}}^m(l)$ to retrieve the model m from confidence values l and black-box access to the model m, which is exactly the setting of *model extraction* attacks in the previous section. This adversary $\mathcal{A}_{\mathsf{bb}}$ can then be called by $\mathcal{A}_2$ to produce input m, which is then forwarded to the adversary of Figure 15 to produce a successful attack r. As such, Soteria is vulnerable to white-box *reconstruction attacks* if there exists an efficient adversary $\mathcal{A}'$ that successfully wins the experiment of Figure 14 for the *model extraction* attack.

I. SUMMARY

Table 7 summarizes the attacks discussed. These are divided between all the identified classes of attacks, as well as whether the adversary is external or if it can query the model as a client. In many instances, the security of our system hinges on another argument over specific restrictions assumed for the adversary.
We now list the arguments that propose the security of our system in different contexts.

{1}: an adversary is unable to retrieve the (white-box) model from confidence values {1a}, black-box access to the model {1b}, or both {1c}.

{2}: an adversary is unable to emulate black-box access to the model from confidence values.

{3}: an adversary is unable to retrieve the dimension of the dataset from black-box access to the model {3a} and confidence values {3b}.

{4}: an adversary is unable to retrieve information of leaf count, depth, and ID from black-box access to the model {4a} and confidence values {4b}.

{5}: no *model inversion* attack exists for retrieving dataset samples for Soteria-B {5a} and Soteria-P {5b}.

White-box-based attacks explicitly require additional information, such as feature vectors, over black-box access to the model [43]. This is something that supports {1b} directly, and since confidence values are not computed from feature vectors, so would be {1a} and {1c}. Extracting model access from only confidence values is an active area of research, but current attacks [20] are still unable to do this in an efficient way {2}. Typically, one cannot infer dimension from simply querying the model, which suggests {3a} is true, but this is unclear for confidence values, and thus one might consider {3b} is false. Reference [5] suggests {4a} is true, but {4b} is not. {5} will fundamentally depend on the application [5], [20].

REFERENCES

[1] U.S. Department of Heatlh and Human Services. *HIPAA for Professionals | HHS.Gov*. Accessed: Feb. 21, 2021. [Online]. Available: https://www.hhs.gov/hipaa/for-professionals/index.html

[2] P. Voigt and A. Von dem Bussche, "The Eu general data protection regulation (GDPR)," in *A Practical Guide*, 1st ed. Cham, Switzerland: Springer, 2017.

[3] S. Iqbal, M. L. M. Kiah, B. Dhaghighi, M. Hussain, S. Khan, M. K. Khan, and K.-K. R. Choo, "On cloud security attacks: A taxonomy and intrusion detection and prevention as a service," *J. Netw. Comput. Appl.*, vol. 74, pp. 98–120, Oct. 2016.

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

[5] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction $*APIs*$," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 601–618.

[6] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1322–1333.

[7] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 36–52.

[8] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, p. 4325.

[9] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.

[10] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, May 2018.

[11] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardw. Architectural Support Secur. Privacy*, Jun. 2013, pp. 1–8.

[12] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," 2018, *arXiv:1803.05961*.

[13] N. Hynes, R. Cheng, and D. Song, "Efficient deep learning on multi-source private data," 2018, *arXiv:1807.06689*.

[14] D. Le Quoc, F. Gregor, J. Singh, and C. Fetzer, "SGX-PySpark: Secure distributed data analytics," in *Proc. World Wide Web Conf.*, May 2019, pp. 3563–3564.

[15] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything you should know about Intel SGX performance on virtualized systems," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 47, no. 1, pp. 77–78, Dec. 2019.

[16] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 283–298.

[17] J. Jiang, X. Chen, T. Li, C. Wang, T. Shen, S. Zhao, H. Cui, C.-L. Wang, and F. Zhang, "Uranus: Simple, efficient SGX programming and its applications," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 826–840.

[18] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, "SecureKeeper: Confidential ZooKeeper using Intel SGX," in *Proc. 17th Int. Middleware Conf.*, Nov. 2016, pp. 1–13.

[19] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," in *Proc. 6th Int. Conf. Learn. Represent.*, 2018, pp. 77–149.

[20] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan, "Exploring connections between active learning and model extraction," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, 2020, pp. 1309–1326.

[21] C. Brito, P. Ferreira, B. Portela, R. Oliveira, and J. Paulo, "SOTERIA: Preserving privacy in distributed machine learning," in *Proc. 38th ACM/SIGAPP Symp. Appl. Comput.*, Mar. 2023, pp. 135–142.

[22] L.-Sustainable Development Strategy International Group. *SGX-Spark*. Accessed: Oct. 22, 2022. [Online]. Available: https://github.com/lsds/sgx-spark

[23] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan, "SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1211–1228.

[24] Intel. *Hibench is a Big Data Benchmark Suite*. Accessed: Oct. 22, 2022. [Online]. Available: https://github.com/Intel-bigdata/HiBench

[25] M. Zaharia, "Apache Spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Nov. 2016.

[26] X. Meng, "MLlib: Machine learning in apache spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[27] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious $*multi − party*$ machine learning on trusted processors," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 619–636.

[28] M. El-Hindi, T. Ziegler, M. Heinrich, and A. Lutsch, "Benchmarking the second generation of Intel SGX hardware," in *Proc. 18th Int. Workshop Data Manag. New Hardw.*, 2022, pp. 1–8.

[29] T. Alves, "TrustZone: Integrated hardware and software security," *Inf. Quart.*, vol. 3, no. 4, pp. 18–24, 2004.

[30] M. Azure. *Azure Confidential Computing*. Accessed: Oct. 22, 2022. [Online]. Available: https://azure.microsoft.com/en-us/solutions/confidential-compute/

[31] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 645–658.

[32] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, "Rethinking the library OS from the top down," in *Proc. 16th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2011, pp. 291–304.

[33] Gramine. *Pal Host ABI*. Accessed: Aug. 1, 2023. [Online]. Available: https://gramine.readthedocs.io/en/stable/pal/host-abi.html

[34] Gramine. *Manifest Syntax*. Accessed: Aug. 1, 2023. [Online]. Available: https://gramine.readthedocs.io/en/stable/manifest-syntax.html

[35] Gramine. *Gramine Library OS With Intel SGX Support*. Accessed: Aug. 1, 2023. [Online]. Available: https://github.com/gramineproject/gramine

[36] J. Jia, R. Wang, Z. An, Y. Guo, X. Ni, and T. Shi, "RDAD: A machine learning system to support phenotype-based rare disease diagnosis," *Frontiers Genet.*, vol. 9, p. 587, Dec. 2018.

[37] S. Rawat, A. Rawat, D. Kumar, and A. S. Sabitha, "Application of machine learning and data visualization techniques for decision support in the insurance sector," *Int. J. Inf. Manage. Data Insights*, vol. 1, no. 2, Nov. 2021, Art. no. 100012.

[38] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, Nov. 2017, pp. 15–26.

[39] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *Proc. Int. Conf. Learn. Represent.*, 2017. [Online]. Available: https://openreview.net/forum?id=BJm4T4Kgx

[40] J.-B. Truong, P. Maini, R. J. Walls, and N. Papernot, "Data-free model extraction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 4769–4778.

[41] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 3–18.

[42] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions," *IEEE Secur. Privacy*, vol. 17, no. 2, pp. 49–58, Mar. 2019.

[43] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and Y. Zhang, "$*updates − leak*$: Data set inference and reconstruction attacks in online learning," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, 2020, pp. 1291–1308.

[44] E. Stefanov, M. v. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An extremely simple oblivious ram protocol," *J. ACM (JACM)*, vol. 65, no. 4, pp. 1–26, 2018.

[45] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer, "Varys: Protecting SGX enclaves from practical side-channel attacks," in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 1–14.

[46] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Oct. 2001, pp. 136–145.

[47] R. Bahmani, M. Barbosa, F. Brasser, and B. Portela, "Secure multiparty computation from SGX," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2017, pp. 477–497.

[48] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua, "Fast matrix factorization for online recommendation with implicit feedback," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jul. 2016, pp. 549–558.

[49] J. Dowling. (Nov. 2017). *Distributed ML and Linear Regression*. Accessed: Dec. 1, 2020. [Online]. Available: https://www.kth.se/social/files/5a040fe156be5be5f93667e9/ID2223-02-ml-pipelines-linear-regression.pdf

[50] M. K. Pakhira, "A linear time-complexity k-Means algorithm using cluster shifting," in *Proc. Int. Conf. Comput. Intell. Commun. Netw.*, Nov. 2014, pp. 1047–1051.

[51] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.

[52] S. Si, H. Zhang, S. Keerthi, D. Mahajan, I. Dhillon, and C.-J. Hsieh, "Gradient boosted decision trees for high dimensional sparse output," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3182–3190.

[53] D. Cai, X. He, and J. Han, "Training linear discriminant analysis in linear time," in *Proc. IEEE 24th Int. Conf. Data Eng.*, Apr. 2008, pp. 209–217.

[54] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *J. Comput. Graph. Statist.*, vol. 15, no. 2, pp. 265–286, 2004.

[55] T. Elgamal and M. Hefeeda, "Analysis of PCA algorithms in distributed environments," 2015, *arXiv:1503.05214*.

[56] C. Elkan, "Boosting and naive Bayesian learning," in *Proc. Int. Conf. Knowl. Discovery Data Mining*, 1997. [Online]. Available: https://pages.cs.wisc.edu/~dyer/cs540/handouts/elkan97boosting.pdf

[57] C. Zhao, D. Saifuding, H. Tian, Y. Zhang, and C. Xing, "On the performance of Intel SGX," in *Proc. 13th Web Inf. Syst. Appl. Conf. (WISA)*, Sep. 2016, pp. 184–187.

[58] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," 2018, *arXiv:1806.03287*.

[59] R. Kunkel, D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "TensorSCONE: A secure TensorFlow framework using Intel SGX," 2019, *arXiv:1902.04413*.

[60] K. Grover, S. Tople, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and secure DNN inference with enclaves," 2018, *arXiv:1810.00602*.

[61] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using SGX," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2019, pp. 1–17.

[62] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "PANOPLY: Low-TCB Linux applications with SGX enclaves," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–15.

[63] S. Arnautov et al., "{SCONE}: Secure linux containers with intel {SGX}," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 689–703.

[64] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," *ACM Trans. Comput. Syst.*, vol. 35, no. 4, pp. 1–32, Nov. 2017.

[65] C. Priebe, D. Muthukumaran, J. Lind, H. Zhu, S. Cui, V. A. Sartakov, and P. Pietzuch, "SGX-LKL: Securing the host OS interface for trusted execution," 2019, *arXiv:1908.11143*.

[66] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan, "Occlum: Secure and efficient multitasking inside a single enclave of Intel SGX," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.* New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 955–970, doi: 10.1145/3373376.3378469.

[67] Enarx Provides. (2022). *Enarx: Webassembly + Confidential Computing*. Accessed: Sep. 20, 2023. [Online]. Available: https://enarx.dev/

[68] M. Brossard, G. Bryant, B. E. Gaabouri, X. Fan, A. Ferreira, E. Grimley-Evans, C. Haster, E. Johnson, D. Miller, F. Mo, D. P. Mulligan, N. Spinale, E. van Hensbergen, H. J. M. Vincent, and S. Xiong, "Private delegated computations using strong isolation," *IEEE Trans. Emerg. Topics Comput.*, early access, Jun. 5, 2023, doi: 10.1109/TETC.2023.3281738.

[69] *Apache Teaclave*. Accessed: Nov. 21, 2020. [Online]. Available: https://teaclave.apache.org/

[70] J. Lind, C. Priebe, D. Muthukumaran, D. O'Keeffe, P.-L. Aublin, F. Kelbert, T. Reiher, D. Goltzsche, and D. Eyers, R. Kapitza, "Glamdring: Automatic application partitioning for Intel $*SGX*$," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2017, pp. 285–298.

[71] C.-C. Tsai, J. Son, B. Jain, J. McAvey, R. A. Popa, and D. E. Porter, "Civet: An efficient Java partitioning framework for hardware enclaves," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, 2020, pp. 505–522.

[72] P. Yuhala, J. Ménétrey, P. Felber, V. Schiavoni, A. Tchana, G. Thomas, H. Guiroux, and J.-P. Lozi, "Montsalvat: Intel SGX shielding for GraalVM native images," in *Proc. 22nd Int. Middleware Conf.*, Dec. 2021, pp. 352–364.

[73] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.

[74] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 619–631.

[75] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2015, pp. 909–910.

[76] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.

[77] J. C. Mitchell and J. Zimmerman, "Data-oblivious data structures," in *Proc. 31st Int. Symp. Theor. Aspects Comput. Sci. (STACS)*, 2014, pp. 554–565.
[78] Databricks. *Optimizing Apache Spark UDFS*. Accessed: Oct. 27, 2022 [Online]. Available: https://www.databricks.com/session_eu20/optimizing-apache-spark-udfs
[79] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from SGX," in *Financial Cryptography Data Security*, A. Kiayias, Ed. Cham, Switzerland: Springer, 2017, pp. 477–497.

CLÁUDIA V. BRITO is currently pursuing the Ph.D. degree with the Doctoral Program in Informatics, University of Minho. She is a Researcher with HASLab, one of the research units of INESC TEC and the University of Minho. Her current research interests include the intersection of machine learning and privacy-preserving techniques, as well as the application of such solutions in the biomedical field. For more information, please visit: https://www.inesctec.pt/en/people/claudia-vanessa-brito.

PEDRO G. FERREIRA received the Ph.D. degree in artificial intelligence from the University of Minho, in 2007. He was a Postdoctoral Fellow with the Centre for Genomic Regulation, Barcelona, from 2008 to 2012, and the University of Geneva, from 2012 to 2014. He is currently an Assistant Professor with the Department of Computer Science, Faculty of Sciences, University of Porto. He is also a Researcher with INESC TEC-LIADD and i3s/Ipatimup. His main research interest includes genomic data science. In particular, he is interested in unraveling the role of genomics on human health and disease. For more information, please visit: https://www.inesctec.pt/en/people/pedro-gabriel-ferreira.

BERNARDO L. PORTELA received the Ph.D. degree from the Universities of Minho, Aveiro, and Porto, in 2018, under the MAP-i Doctoral Program. He is currently an Assistant Professor with the Department of Computer Science, Faculty of Sciences, University of Porto. He is also a Researcher with HASLab, one of the research units of INESC TEC. His research interests include cryptography and information security, more specifically regarding secure multiparty computation protocols relying on trusted hardware. For more information, please visit: https://www.inesctec.pt/en/people/bernardo-luis-portela.

RUI C. OLIVEIRA received the Ph.D. degree from École Polytechnique Fédérale de Lausanne, in 2000. He is currently an Associate Professor with Habilitation at the Informatics Department, University of Minho, a Member of the Board of Directors of INESC TEC, the Director of the Minho Advanced Computing Centre, and the Co-Director of the UT Austin Portugal Program. His research contributions have been in the field of fault-tolerant distributed agreement, epidemic communication algorithms, and the conception, development, and assessment of dependable database systems. For more information, please visit: https://www.inesctec.pt/en/people/rui-carlos-oliveira.

JOÃO T. PAULO received the Ph.D. degree from the Universities of Minho, Aveiro, and Porto, in 2015, under the MAP-i Doctoral Program. He is currently an Assistant Professor with the University of Minho and a Senior Researcher with HASLab, one of the research units of INESC TEC and the University of Minho. His research interests include distributed and operating systems with an emphasis on storage and database solutions' scalability, performance, and dependability. For more information, please visit: https://www.inesctec.pt/en/people/joao-tiago-paulo.