Securing Big Data Scientific Workflows via Trusted Heterogeneous Environments

Saeid Mofrad^, Ishtiaq Ahmed^, Fengwei Zhang*, Shiyong Lu, Ping Yang, Heming Cui

Index Terms—trusted computing, Intel SGX, AMD SEV, big data workflow, heterogeneous cloud

Abstract—Big data workflow management systems (BDWMS)s have recently emerged as popular data analytics platforms to conduct large-scale data analytics in the cloud. However, the protection of data confidentiality and secure execution of workflow applications remains an important and challenging problem. Although a few data analytics systems, such as VC3 and Opaque, were developed to address security problems, they are limited to specific domains such as Map-Reduce-style and SQL query workflows. A generic secure framework for BDWMSs is still missing. In this paper, we propose SecDATAVIEW, a distributed BDWMS that employs heterogeneous workers, such as Intel SGX and AMD SEV, to protect both workflow and workflow data execution, addressing three major security challenges: (1) Reducing the TCB size of the big data workflow management system in the untrusted cloud by leveraging the hardware-assisted TEE and software attestation; (2) Supporting Java-written workflow tasks to overcome the limitation of SGX's lack of support for Java programs; and (3) Reducing the adverse impact of SGX enclave memory paging overhead through a "Hybrid" workflow task scheduling system that selectively deploys sensitive tasks to a mix of SGX and SEV worker nodes. Our experimental results show that SecDATAVIEW imposes moderate overhead on the workflow execution time.

1 Introduction

Today, technology advances provide an opportunity to collect and store a large amount of data (referred to as *big data*) from different data sources, such as Event logs, the Internet, Smartphones, Databases, Sensors, IoT devices, etc [1]. The study and comparison of these collected data provide useful knowledge that is often used in decision-making processes. For example, in the business context, big data is used to forecast market trends. Analyzing the collected data allows policymakers to take prescriptive action for the benefit of society in healthcare and government. Cloud providers, with their vast and elastic processing, storage and networking infrastructure, offer exciting potential for large-scale data analytics, which is often compute and resource intensive. For example, the Hadoop framework [2] is a big data analytics platform that processes Map-Reduce-style workflows and is often deployed in a cloud environment. Nonetheless, the notion of cloud is based on resource sharing abstraction, and cloud hardware and software resources are typically shared among different users or organizations through isolation techniques such as virtual machines or containers. The characteristics of resource sharing and the large size of cloud system software make the cloud vulnerable to different classes of attacks [3]–[8]. Scientific workflows running on clouds or virtualized data centers rely on the integrity of the OS and hypervisor code to operate correctly, which introduces a large trusted computing base (TCB). For instance, Linux kernel has about 35.5 million lines of code and the latest Xen hypervisor contains 586 thousands of lines of code [9]. This large TCB inevitably creates vulnerabilities that could be exploited by attackers. The National Vulnerability Database shows that there are 21 vulnerabilities in Xen that have been reported between 2017 to 2019 and 21 vulnerabilities in the latest Linux kernel version 5.x.x [10]. Attacks may stem inside the cloud provider (e.g., dishonest administrator) or outsiders. External attackers may exploit such vulnerabilities to gain access to computers on which scientific workflows execute to access or modify data and workflow tasks. For example, Ristenpart *et al.* [8] showed that an outside adversary could extract unauthorized information in AWS EC2 instances. One of the fundamental and realistic security-enhancing strategies is to isolate the execution of workflows at runtime, where workflow data is processed in clear text to ensure data processing efficiency. Hardware-assisted TEE is a promising solution for protecting the execution of big data workflows in the cloud. To create hardware-assisted security solutions, hardware vendors introduced several hardware-assisted TEEs, such as Intel Software Guard eXtensions (SGX) [11]–[13] and AMD Secure Encrypted Virtualization (SEV) [14]. Intel SGX and AMD SEV technologies are designed to be general-purpose hardware-assisted TEEs in the x86 architecture, which help to reduce the runtime attacks in cloud environments. Some researchers [15] have explored the pros and cons of each approach with a side-by-side comparison. There have been several prior efforts to protect

- *S. Mofrad^ and I. Ahmed^ are entitled the co-first authors with equal contribution; F. Zhang* is the corresponding author.*
- *S. Mofrad is with COMPASS Lab and Big Data Research Lab, Department of Computer Science, Wayne State University, Detroit, MI, USA. E-mail: saeid.mofrad@wayne.edu.*
- *I. Ahmed is with Big data Research Lab, Department of Computer Science, Wayne State University, Detroit, MI, USA. E-mail: ishtiaq@wayne.edu.*
- *F. Zhang is with COMPASS Lab, Department of Computer Science and Engineering, Southern University of Science and Technology, China. E-mail: zhangfw@sustech.edu.cn.*
- *S. Lu is with Big data Research Lab, Department of Computer Science, Wayne State University, Detroit, MI, USA. E-mail: shiyong@wayne.edu.*
- *Ping Yang is with the Department of Computer Science, State University of New York at Binghamton, Binghamton, NY, USA. E-mail: pyang@binghamton.edu.*
- *H. Cui is with the HKU Systems Software Group, Department of Computer Science, University of Hong Kong, Hong Kong. E-mail: heming@cs.hku.hk.*

big data analytics in the cloud with Intel SGX. For example, Shuster *et al*. [16] proposed VC3, a system that leverages SGX to protect unmodified Map-Reduce tasks written in C/C++. Rafael Pires *et al*. [17] proposed a lightweight, Map-Reduce framework with Lua [18], a high-level language that interprets the Map-Reduce Lua scripts in Intel SGX. In another effort, Zheng *et al*. proposed Opaque [19] to enhance the security of the Spark SQL with SGX. Although these systems are the pioneers in using hardware-assisted TEEs for big data analytics, they are limited to specific domains. For example, both VC3 and Rafael Pires *et al* [17] systems support only Map-Reduce-style workflows consisting of a Map task and a Reduce task, but not workflows of a more flexible structure. The Opaque application is limited to the use of relational algebra based tasks with Spark SQL. Existing systems do not support workflow tasks with well-defined input and output ports of complex data types, such as lists, maps, and arrays. In this paper, we present SecDATAVIEW, a new distributed BDWFMS that leverages Intel SGX and AMD SEV to develop a TEE for the secure execution of big data workflows. SecDATAVIEW protects against attacks that mainly happen on cloud providers and data centers, including attacks that are launched by a dishonest cloud administrator, malicious cloud software, or a compromised virtual machine. SecDATAVIEW is transparent to the users and the application-level workflow tasks. Our research and development focus on addressing the following challenges. Firstly, to address the above-mentioned clould's security vulnerabilities, SecDATAVIEW reduces the size of the system's TCB by isolating the security-sensitive modules of the system in the SGX-protected enclaves or SEV-protected VMs and by keeping the high-privileged cloud system software outside of the TCB. Secondly, SGX applications are bounded by a limited set of C/C++ libraries. This is due to the system call restriction in the SGX enclave that limits the availability of C/C++ libraries inside an SGX enclave. However, many workflow tasks are written in Java and may use several third-party Java libraries which are not directly supported by SGX. To address this challenge, SecDATAVIEW uses the *SGX-Shield* approach [20] and specifically incorporates the SGX-LKL library OS [21] to execute workflow tasks written in Java inside SGX enclaves while maintaining a small TCB of the BDWFMS. Thirdly, big data workflow tasks are often memory-intensive. For example, 75% of the execution time of the Broadband workflow [22] is consumed by workflow tasks that require over 1GB memory. Running the DATAVIEW [23] Kernel itself also requires over 500MB memory. As a result, SGX memory paging could significantly increase the execution time of the DATAVIEW server and workflow tasks. To address this concern, SecDATAVIEW uses AMD SEV, instead of SGX enclaves, to support a larger amount of secure memory. Our contributions are summarized as follows:

1) We propose SecDATAVIEW, a heterogeneous big data workflow management system that leverages Intel SGX and AMD SEV for the secure execution of big data workflows. We propose a secure architecture and the WCPAC (Workflow Code Provisioning and Communication) protocol that uses real-time Intel remote attestation along with in-enclave VPN connection to provision and attest secure worker nodes, securely provision the code for the *Task Executor* and workflow tasks on each participating worker node for a workflow, establish secure communication between the master node and worker nodes, and ensure secure file transfers among worker nodes. We leverage the SGX-LKL library OS to execute workflow tasks written in Java to overcome the limitation of SGX's lack of support for Java programs.
2) To support memory-intensive workflows and reduce the overall performance overhead incurred by SGX enclaves EPC memory paging, SecDATAVIEW introduces the notion of "Hybrid" operation that enables users to selectively assign confidential tasks into SGX and SEV worker nodes. Our previous work [15] reported that SEV performs faster than SGX for workloads that require a larger amount of secure memory. However, SGX offers better security than SEV due to its smaller TCB size, enclave abstraction, and memory integrity protection. In SecDATAVIEW, users can assign memory-intensive confidential tasks (e.g., tasks that do not require enhanced-degree of security but require a large amount of secure memory) in SEV worker nodes while assigning security-sensitive confidential tasks (e.g., tasks that need enhanced-degree of security) to SGX worker nodes.
3) We have implemented and evaluated SecDATAVIEW with a comprehensive set of real-world workflows, including a Diagnosis Recommendation workflow [24], a Distributed K-means workflow, a Neural Network workflow, a Map-Reduce workflow [25], and MONTAGE [26] workflow, to demonstrate the feasibility and usability of the proposed system. Our experimental results show that SecDATAVIEW imposes a moderate overhead on the execution times of various workflows.

This paper is an extended version of our prior work [27] that is awarded with the "Artifacts Evaluated - Functional" badge. In this work, we developed a more complete and more secure system. The source code is available in the SecDATAVIEW GitHub[1]. The main differences between the two versions are summarized as follows:

1) We provided a more secure and more practical edition of SecDATAVIEW as part of the extensions. We proposed and integrated a stronger security measurement that we assumed their existence in our prior work [27]; the implemented security measure is to prevent attacks that fake the presence of TEE (i.e., SGX or SEV) in the cloud provider environments. We added this protection through enforcing a real-time cloud's hardware and enclave binary attestation for every single worker node during the worker launch. The TEE remote attestation sufficiently prevents attacks that fake the presence of TEE and reveals any modifications to the enclave's

1. https://github.com/shiyonglu/SecDATAVIEW

binary in the TEE [28]. Additionally, we used in-enclave VPN tunneling methods in the SGX-LKL for secure delivering SGX TEE's secret after the successful remote attestation. Moreover, we enforced the real-time TEE's storage clean-up before terminating the worker nodes, which prevents attacks that may happen after workflow execution is finished (i.e., accessing workflow residual files in the disk image).

2) We modified the Workflow Code Provisioning and Communication (WCPAC) protocol, the SecDATAVIEW system architecture, Cloud Resource Management, Workflow Executor and Task Executor to enforce the proposed security in SecDATAVIEW. To facilitate further research and development, we disclosed detail information describing above-mentioned modules and also other subsystems of SecDATAVIEW that have not been discussed in ACSAC 19 conference paper. Also, all WCPAC steps are mapped at source code level helping interested researchers to understand the logic of source codes and their interaction in the SecDATAVIEW GitHub.
3) We added a set of new experimental validations (e.g., Deep Learning and MONTAGE workflows) targeted to observe the SecDATAVIEW system performance with different configuration settings and workflow designs; all discussed in Section §4. For example the deep learning algorithms such as Neural Network (NN) families often depends upon "loop" construct whose representation is limited in form of DAG diagram. We showed that the NN algorithm could be assigned as an independent task in the workflow DAG. Such deep-learning experiments prove the usability and flexibility of the SecDATAVIEW system and show how proposed system handles a challenging case where a complex algorithm cannot be distributed as part of different tasks in the DAG. In addition, the MONTAGE workflow represents a workflows with many input and output data channels that go through encryption/decryption and secure data transfer during the secure workflow execution. Our experiment on the MONTAGE workflow reveals the behavior of SecDATAVIEW in heavily-connected DAG scenario.

The rest of the paper is organized as follows. Section §2 provides an overview of big data workflows, the DATAVIEW workflow management system, the Intel SGX, the AMD SEV, and the adversary model. Section §3 describes the design and implementation of SecDATAVIEW. Section §4 presents our experimental results, security analysis, and comparison studies. Section §5 presents the related work and Section §6 concludes the paper.

2 Background & Adversary Model

Big data: Big data refers to a collection of large datasets and records containing the raw information related to the data collector sensors and data sources [29]–[31]. Business experts, understand the power of data and how data could empower them to compete with their business rivals, provide a better customer experience, and gain revenue advantages. In fact, companies are very active with collecting data whenever and wherever they can and big data analytics becomes essential to many modern companies. While big data provides invaluable information for the decision-making process, big data analysis poses various challenges in storage, transfer, processing, and management due to the following big data characteristics [1], [31]: (1)-*Volume* that represents the size of big data records that range between terabytes to petabytes; (2)-*Variety* that demonstrates the format of data records, which can be structured (such as student records), unstructured (such as videos and images), or semi-structured in which data records do not typically follow a particular data schema; (3)-*Velocity* that represents the data arrival speed, which can be very high in real-time applications that require rapid and on-the-fly processing of data. (4)-*Value* that speaks for the results that could be extracted from big data records and is categorized as statistical, hidden, and unknown. Also, integrating and correlating data records originating from different sources unlocks useful information that might not be obtainable by processing just one data source. (5)-*Veracity* that embodies the trustworthiness and consistency of big data.

Big data workflows: A big data workflow is a computerized model for automating a data analytics process, which consists of a set of computational tasks and their data dependencies, to process and analyze data of ever increasing in scale, complexity, and rate of acquisition [23], [32]. A big data workflow management system (BDWFMS) is a system that completely defines, modifies, manages, monitors, and executes scientific workflows on the cloud in the order that is driven by the workflow logic [23], [32]. An example workflow is given in Figure 4. SecDATAVIEW was developed based on the DATAVIEW scientific workflow management system [23]. We chose DATAVIEW as the baseline for the development of our secure BDWFMS because of the following reasons. Firstly, DATAVIEW represents the state-of-the-art big data workflow management system and has a strong user base (over 700 registered users worldwide). Secondly, DATAVIEW has been used in various data analytics applications, including diagnosis recommendation [24], predicting the efficacy of therapeutic services for autism spectrum disorder [33], analysis of vehicle data to assess driver's driving behavior [23], medical image processing [34], biological simulation data analysis [35], and brain fiber connectivity analysis [36]. The architecture of DATAVIEW is given in Figure 1. DATAVIEW system consists of three layers: the *Presentation & Visualization* Layer, the *Workflow Management* Layer, and the *Task Management* Layer. The *Presentation & Visualization* layer is responsible for the presentation of workflows and the visualization of different data products, and also provenance metadata. The *Workflow design & configuration* module implements a GUI front-end environment for end-users to design and configure workflows. The *Workflow Engine* is the module that manages the execution of workflows. The *Workflow Monitoring* module keeps track of the status of workflow execution. The *Data Product Management* module stores and manages all data products used in workflows. The *Provenance Management* module is responsible for generating workflow provenance. The *Task Management* module is responsible for the execution of workflow tasks that are executed in the cloud. The *Cloud Resource Management* module interacts with clouds for provisioning and de-provisioning

virtual machines (e.g., Amazon EC2 instances).

Intel SGX: Intel SGX is a recent hardware innovation that enables users to instantiate a secure container, called *enclave*, to protect the execution of code from being altered by malicious code or external attackers. SGX protects the integrity of the enclave code and data, even when the high-privileged system software is compromised [37]. SGX also protects against the physical memory access class of attacks [20]. With SGX, the trusted computing base (TCB) contains only the processor and the code running inside the enclave. SGX reserves a limited size of the encrypted memory region called *Enclave Page Cache (EPC)*, where enclaves are created within this region. In the current SGX release, the size of EPC is 32MB, 64MB, or 128MB [37], [38]. Although a larger memory size can be supported through the paging mechanism, it incurs up to $1,000X$ performance overhead [37]. To speed up the execution performance of parallel applications, SGX supports multi-threads inside the enclave.

AMD Secure Encrypted Virtualization (SEV): AMD SEV is a security feature that is created on top of the AMD Secure Memory Encryption (SME) [14] technology and provides the protection against attacks that usually occur in cloud system software such as high-privileged hypervisors by encrypting the memory space of VM instances. SEV protects a VM's memory space with an encryption key that is protected from the hypervisor, cloud management software or other parts of the system [14]. SEV protection is transparent to the user applications that are running inside SEV-protected instances. Protected applications are unaware of underlying memory encryption. AMD's Memory Encryption Engine is capable of using different encryption keys to protect different SEV-protected VM's memory spaces on the same platform.

Adversary Model: The adversary model for SecDATAVIEW is similar to that for VC3 [16]. We assume that an attacker may control the whole software stack in remote servers, including their system software. An attacker may also have access to network packets and capture, replay, and modify them. In addition, an attacker may access or change data after the data leaves the processor with hardware-tapping or probing techniques. An attacker can also access any process running on a worker node. The adversary could fake the presence of the TEE or be a dishonest administrator who can tap into a worker node to read user data, or an attacker who can exploit a vulnerability in the worker node host's system software and access user data that is located in the unprotected memory, in the network buffer, or on the physical storage medium. We assume the attacker is not capable of modifying SGX-enabled CPU package or AMD SEV SoC that resides in the remote location. Other attacks, including network traffic-analysis [39], denial-of-service, access pattern leakage [40], side-channel attacks [41], and fault injections [42], are out of the scope of this paper.

3 Design of SecDATAVIEW

We identify the following security-related requirements for SecDATAVIEW:

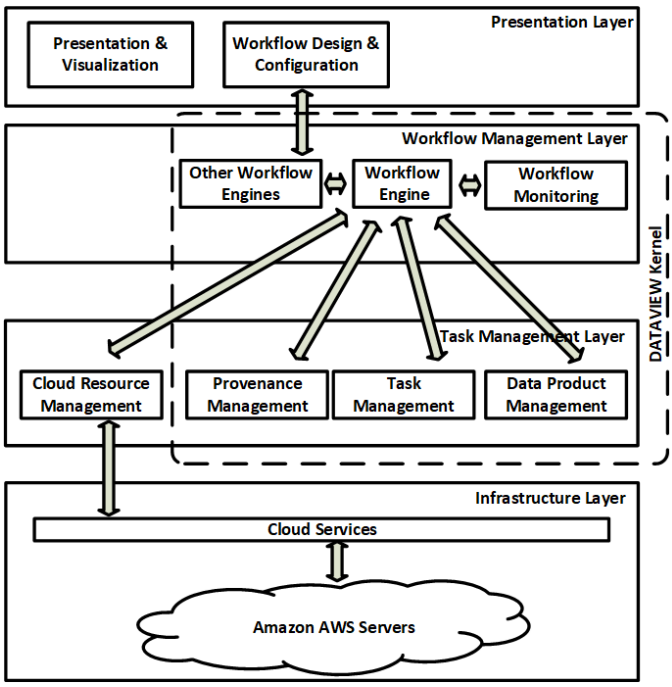


Fig. 1: Architecture of the *DATAVIEW (unsecured) system* [32].

- **R1:** Providing the confidentiality and integrity of code and data for workflows running on public untrusted clouds.
- **R2:** Evaluating the authenticity of hardware resources and validating the worker VMs that are provided by an untrusted cloud provider.
- **R3:** Minimizing the TCB size for SecDATAVIEW.
- **R4:** Enabling the trade-off between security and performance for workflows with different user requirements.
- **R5:** Supporting the execution of Java-based workflow tasks in SGX nodes without tedious code refactoring.

Figure 2 depicts the architecture of SecDATAVIEW, that uses a heterogeneous computing environment including both SGX and SEV worker nodes. We propose the Workflow Code Provisioning and Communication (WCPAC) protocol that guarantees the integrity and confidentiality protection of a workflow execution (Requirement R1). The WCPAC protocol in SecDATAVIEW leverages a real-time TEE attestation mechanism that is provided by the hardware vendors (e.g., Intel SGX Attestation Service via SGX-LKL) to evaluate the trustworthiness of cloud's hardware resources and enclave's binary during the worker launch process (Requirement R2). SecDATAVIEW's architecture leverages hardware-assisted TEEs in the cloud providers and only provisions and executes the security-sensitive modules and data inside TEE enclaves and trusted VMs, which significantly decreases the hardware and software TCB of the SecDATAVIEW system (Requirement R3). SecDATAVIEW provides a "Hybrid" operation mode by leveraging a heterogeneous computing environment (i.e., Intel SGX and AMD SEV). The "Hybrid" operation provides the trade-off between the performance and the degree of security (Re-

quirement R4). Based on the previous study [15], SGX offers better security than SEV due to its smaller TCB size, enclave abstraction, and memory integrity protection. However, SGX may impose high performance overhead on memory-intensive applications due to its limited enclave memory size. While SEV offers better performance for memory-intensive applications and the assurance of confidentiality, it comes with the limitations of a larger TCB size (i.e., entrusting the entire VM) and lack of memory integrity protection, which decreases its degree of security assurance. SecDATAVIEW benefits greatly from our proposed "Hybrid" operation and the heterogeneous computing environment that includes both SGX and SEV worker nodes. Security-sensitive workflow tasks (e.g., tasks that process confidential data) are executed on SGX nodes and memory-intensive tasks with lower security requirement (e.g., tasks that do not process confidential data) are executed on SEV nodes. In this way, SecDATAVIEW achieves the degree of security with low performance overhead. Moreover, SecDATAVIEW leverages the *SGX-Shield* approach that was initially proposed in [20] and a SGX-supported Linux kernel library that is provided by SGX-LKL [21] to execute JVM and Java tasks in the SGX worker nodes (Requirement R5).

3.1 SecDATAVIEW Architecture

To address the security-related requirements of SecDATAVIEW, we first identify the components in DATAVIEW that process confidential data. In DATAVIEW [23], the *Workflow Engine* and the *Task Management* module are security-sensitive components as they interact with workflow tasks that may process confidential data. These components need to be distributed inside SGX/SEV TEEs or a trusted on-premises server (Requirement R3). Also, DATAVIEW was not designed with security in mind and all communications between two different modules were passed through an unencrypted channel. Although the input and output data channels were transferred through secure FTP (sftp), they were stored in the plaintext format. To address this security flaw and securing DATAVIEW, we develop the WCPAC (Workflow Code Provisioning And Communication) protocol to provision and attest secure worker nodes (Requirement R2). Via the WCPAC protocol, the system securely provisions the code for the *Task Executor* and workflow tasks on each participating worker node, and establish the secure communication and file transfers between the master node and worker nodes, and among worker nodes. As a result, the confidentiality and integrity of intermediate workflow data products are protected during their transfer from one workflow task to another (Requirement R1).

To integrate the WCPAC protocol into DATAVIEW, we redesigned the *Cloud Resource Management* module to initialize and attest SGX/SEV worker nodes, and added two security-related subsystems – *Code Provisioner* and *Code Provisioning Attestation* – to the *Task Management* and to the *Workflow Engine* modules, respectively. Figure 2(a) gives the secure system architecture for SecDATAVIEW in the cloud and the zoom-in view of its two components: the *Workflow Engine* and the *Task Management*. Figure 2(d) provides the deployment architecture of SecDATAVIEW, which consists of two parts: the master node running in a secure on-premises server and worker nodes running in a public cloud. The gray components in the figure represent the redesigned components in SecDATAVIEW. In SecDATAVIEW, the *Code Provisioner* and *Task Executor* are executed inside SGX enclaves or SEV-protected VMs.

3.1.1 Executing workflows inside SGX enclaves

SGX-based applications are implemented with Intel SGX SDK that uses low-level C/C++ to accomplish SGX primitives and introduces the notion of enclave abstraction into the programming model. The enclave abstraction divides every SGX application into trusted and untrusted runtime that should be designed carefully by the developers. We identify two common SGX-based application design. One approach is called the *Specialized-Enclave*, in which the developer follows all the SGX rules, such as code partitioning in trusted and untrusted parts, defining Ecalls and Ocalls [43], and configuring the Enclave Definition Language [43], to develop applications. SecureKeeper [38] uses the *Specialized-Enclave* approach. In the *Specialized-Enclave* approach, the size of the TCB is small because the size of code running inside the enclave is minimal. The *Specialized-Enclave* approach works well if the system depends on only the static components that are usually created by skillful developers. However, the DATAVIEW system uses dynamic and third-party proprietary tasks and libraries that are not created or used by the DATAVIEW system developers. Applying the *Specialized-Enclave* approach would dramatically decrease the usability and the security of the DATAVIEW system due to the burden of learning low-level SGX-based programming on the shoulder of its end-users. Besides, C/C++ is not a type-safe language and user-created SGX workflow tasks may unintentionally expose low-level vulnerabilities that result in the leak of sensitive information from the enclave and the compromise of the system runtime environment. Another approach is the *SGX-Shield* approach that was initially proposed in [20]. The *SGX-Shield* approach executes an unmodified application in the SGX runtime. In this approach, the unmodified application along with its execution environment (such as JVM) and codes that belong to the library operating system (LibOS) entirety is executed inside the enclave. On one hand, the *SGX-Shield* approach introduces a larger TCB as it puts more code inside the enclave and may significantly decrease the memory access performance of the enclave [38] when the enclave memory size exceeds $96MB$ due to the EPC memory paging overhead. On the other hand, the *SGX-Shield* approach substantially increases the usability of the SGX-based system by supporting the execution of unmodified applications. In addition, the *SGX-Shield* approach enables end-users to execute code written in type-safe languages such as Java, which mitigates unintended memory leakage in the program and is suitable for security-sensitive scientific workflow applications.

Considering above-mentioned benefits, we developed SecDATAVIEW using the *SGX-Shield* approach. HAVEN [20], Graphene-SGX [44], SCONE [37], and the SGX-LKL library OS [21] use the *SGX-Shield* approach to run unmodified applications in enclaves. Among them, SCONE and SGX-LKL support Java. Because SGX-LKL is open-source, SecDATAVIEW uses SGX-LKL to execute workflow tasks written in Java inside SGX enclaves. One limitation

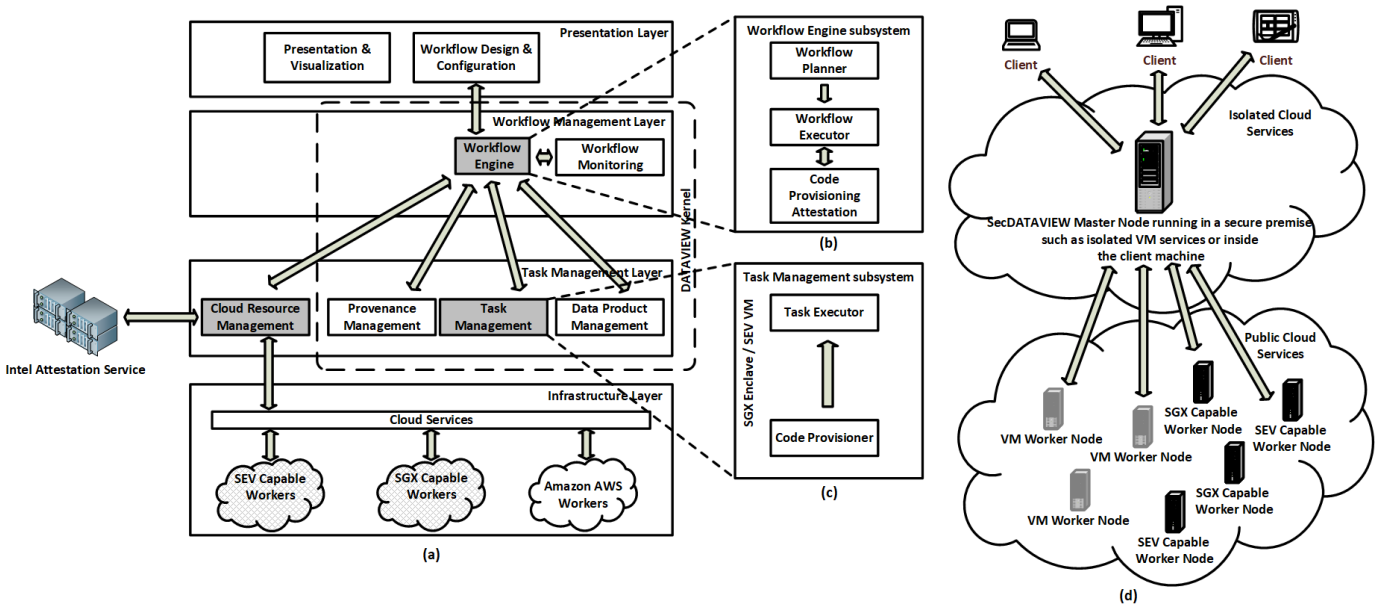


Fig. 2: (a) The system architecture of SecDATAVIEW and zoom-in views of its two components: (b) Workflow Engine and (c) Task Management. (d) The all-in-cloud deployment architecture of SecDATAVIEW.

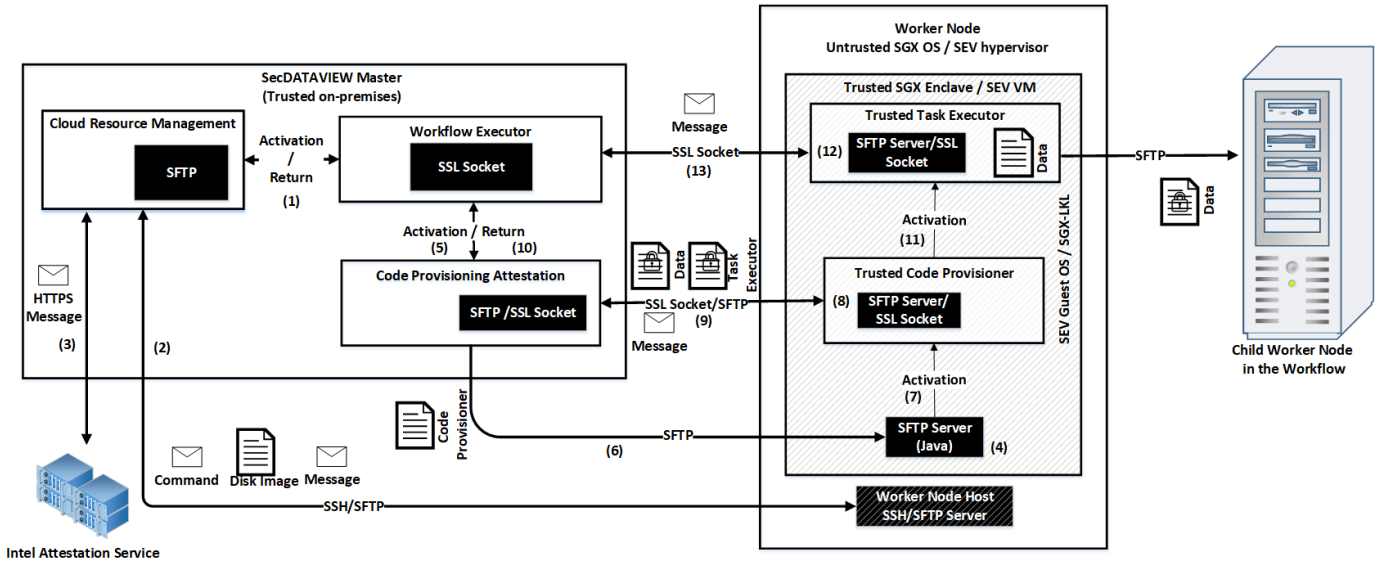


Fig. 3: The WCPAC Protocol for securing the communication between *Workflow Executor*, *Cloud Resource Management*, *Code Provisioning Attestation*, *Code Provisioner*, and *Task Executor*.

of SGX-LKL is that SGX-LKL supports only the execution of a single process inside the SGX enclave. However, complex modules in SecDATAVIEW such as *Code Provisioner* and *Task Executor* are often run as multiple processes (e.g., SSL socket and sftp server). To tackle the above limitation, we developed a Java-written sftp server which is included in the SGX-LKL encrypted disk image and is sent to the SGX worker node. When an SGX-LKL enclave is launched, the Java sftp server starts executing inside the enclave. The sftp server leverages Java multi-threading, class loader, and reflection to dynamically activate the *Code Provisioner* module upon its arrival and as part of its single running process inside the enclave. In the same way, the *Code Provisioner* module is enabled to activate the *Task Executor* inside the enclave.

3.1.2 *Executing workflows inside SEV-protected VMs*

AMD SEV is designed for cloud applications and protects unmodified applications by shielding the SEV VM instances from other parts of the system [14]. SEV does not protect the integrity of the memory content but imposes lower performance overhead than SGX. To reduce the performance overhead, SecDATAVIEW introduces a "Hybrid" operation mode in which memory-intensive workflow tasks that do not require enhanced-degree of protection are executed inside SEV-protected VMs. The workflow designer decides whether a workflow task should be executed on an SEV or on an SGX worker node. SecDATAVIEW contains a pre-

created SEV disk image. This SEV disk image is used at runtime to provision a customized VM on a SEV worker node with an execution environment that includes the guest OS, the Java virtual machine and other necessary components (e.g., the stand-alone Java sftp server) for secure workflow execution.

3.2 The WCPAC Protocol

We developed a Workflow Code Provisioning And Communication (WCPAC) protocol for securing the execution of workflow tasks in remote worker nodes. The main functionalities of WCPAC include (1) provisioning and attesting secure worker nodes, (2) securely provisioning the code for the *Task Executor* and workflow tasks on each participating worker node, (3) establishing a secure communication channel between the master node and each worker node, and (4) establishing secure communication channels among worker nodes for secure data transfer.

Every SGX worker node is configured to execute the SGX-LKL library. AMD servers are used to execute SEV instances. When a worker node is launched, the *Cloud Resource Management* subsystem uses an SGX-LKL-based Intel remote attestation similar to [45] to verify the trustworthiness of Intel SGX CPU and SGX-LKL enclave and to send the application configuration (i.e., disk cryptography key and JVM setting) remotely inside the trusted SGX enclave. Besides, AMD guest attestation [14] should be used to launch and verify the SEV instances. Note that due to the 1) similarity of the AMD guest attestation idea with SGX remote attestation, 2) existing security concerns regarding AMD guest attestation [46] and 3) the fact that SecDATAVIEW, uses SEV workers mainly for executing less security-sensitive tasks in the workflow, we leave the implementation of AMD guest attestation for future work. Nevertheless, the WCPAC protocol assumes that such a protocol is incorporated, and the SEV workers would pass the guest attestation upon the request of the *Cloud Resource Management* module. Besides, the WCPAC protocol assumes that the approaches used by TEE hardware vendors (i.e., SGX-LKL and AMD SEV) to launch, attest and access the disk images are secure.

The SecDATAVIEW master node is deployed on a trusted on-premises server whose security is ensured. SecDATAVIEW will provision as many worker nodes as necessary from a given heterogeneous computing environment to execute a particular workflow. The user can determine workflow scheduling during the workflow design, or a workflow scheduling algorithm, called SEED [47], can be used to schedule efficient workflow execution on the provisioned worker nodes. During the execution, SecDATAVIEW dynamically deploys a *Code Provisioner* and a *Task Executor* on each worker node using the WCPAC protocol. The remaining components of SecDATAVIEW will run on the trusted on-premises server. Figure 3 shows the communication diagram of the WCPAC protocol. The detailed sequence diagram of the WCPAC protocol is provided in GitHub [2].

Firstly, the *Workflow Executor* activates the *Cloud Resource Management* module with a request containing the machine type (i.e., SGX or SEV) to initialize the worker nodes – **Step (1)** in Figure 3. If a worker node is SGX node, then the *Cloud Resource Management* module sends the SGX-LKL encrypted disk image to the worker node and activates SGX-LKL over ssh, which initializes the worker SGX enclave for remote activation – **Step (2)** in Figure 3.

Intel SGX worker remote attestation: *Cloud Resource Management* follows the remote attestation and remote control steps provided by SGX-LKL group [45]. We have modified the SGX-LKL remote attestation source code and use it to provide attestation service in SecDATAVIEW. *Cloud Resource Management* makes Intel Attestation Service (IAS) queries (**Step (3)** in Figure 3) and compares enclave measurement (e.g., (MRENCLAVE) and (MRSIGNER)) with expected values to evaluate the cloud provider's SGX hardware and the enclave binary that is executed on the SGX worker node. Upon receiving the successful attestation report from the IAS, a public VPN key of in-enclave's VPN server for the attested SGX worker is received by the *Cloud Resource Management* module from the worker. Upon VPN's public key arrival, the SecDATAVIEW master node adds the worker enclave's VPN endpoint as its VPN peer. Using the secure VPN channel, the decryption key of the disk image along with stand-alone sftp server's application configuration and JVM environments are sent inside the trustworthy enclave. At this moment, the enclave executes the stand-alone sftp server and the SGX worker node is ready. If a worker node is an SEV-protected VM, then the *Cloud Resource Management* module sends the SEV disk image to the worker node, launches SEV-protected VM over ssh, and runs the stand-alone sftp server inside the SEV-protected VM – **Step (2)** in Figure 3.

Upon successful initialization, all worker nodes have active Java sftp server – **Step (4)** in Figure 3. At this step, the *Cloud Resource Management* module returns the control back to the *Workflow Executor*. The *Workflow Executor* then activates the *Code Provisioning Attestation* module, which computes the SHA256 digest of the *Code Provisioner* file and stores the digest in its memory – **Step (5)** in Figure 3. In addition, the *Code Provisioning Attestation* module randomly generates an encryption key and stores the key in its memory. The *Code Provisioning Attestation* module then encrypts the *Task Executor* with the generated key and sends the *Code Provisioner*, the SSL certificates of the *Code Provisioner*, and the encrypted *Task Executor* to the SGX enclave or the SEV instance through sftp – **Step (6)** in Figure 3. The stand-alone sftp server process dynamically activates the *Code Provisioner* through Java reflection and class loader, transfers the control to the *Code Provisioner*, and terminates the sftp server. The *Code Provisioner* then computes the SHA256 digest on its file (self-integrity inspection), initiates a new sftp server as part of a new running thread for the secure file transfer, opens a new SSL socket to communicate with the *Code Provisioning Attestation* module, and sends its SHA256 digest to the *Code Provisioning Attestation* module through the SSL socket – **Steps (7) and (8)** in Figure 3.

After the *Code Provisioning Attestation* module receives the *Code Provisioner*'s SHA256 digest, the *Code Provisioning Attestation* module compares the SHA256 digest against the digest stored in its memory to ensure that *Code Provisioner* is not altered. If the SHA256 digests do not match, the application is terminated; otherwise, the *Code Provisioning Attestation* module sends the *Task Executor*'s decryption key

2. https://github.com/shiyonglu/SecDATAVIEW/blob/master/WCPAC/WCPAC.png

to the *Code Provisioner*. In addition, the *Code Provisioning Attestation* module sends the encrypted workflow's input data, the *Task Executor*'s configuration, and the *Task Executor*'s SSL certificate to the *Code Provisioner* and through sftp. After the success of attestation and file transfer, the control is returned to the *Workflow Executor* from the *Code Provisioning Attestation* module – **Steps (9) and (10)** in Figure 3.

Upon receiving the decryption key of the *Task Executor* and all the dependency files, the *Code Provisioner* module decrypts the *Task Executor* and dynamically activates the *Task Executor* using the Java reflection and class loader. The *Code Provisioner* then terminates and the control is transferred to the *Task Executor* – **Step (11)** in Figure 3.

The *Task Executor* is initialized and a new SSL socket with its SSL certificate is started as part of the *Task Executor* running thread. At this moment, the communication between the *Workflow Executor* and the *Task Executor* is secured and the *Task Executor* completes all assigned tasks based on the local workflow schedule it receives from the *Workflow Executor*. The results are sent through sftp to the children worker nodes in the workflow or send back to the user in the encrypted form, and the *Task Executor* terminates – **Steps (12) and (13)** in Figure 3. It is noteworthy that the workflow's data cryptography key is carried with the *Task Executor* and is used for the encryption and decryption purpose throughout the workflow execution. The data owner generates and encrypts the input files with a provided cryptography tool, and the secret key is compiled as part of the *Task Executor* and is securely transferred to and decrypted in the trustworthy worker nodes. Also, all trustworthy worker nodes share the same cryptography key, so the data received from parent nodes could be decrypted in the children nodes in the workflow and vice versa.

Algorithm 1: Code Provisioning Attestation

Input : A GlobalSchedule $gsch = [lsch_1, \cdots, lsch_n]$
Output : Status

$password \longleftarrow$ random six characters;
$codeProvisionerJarSHA256 \longleftarrow$ Generate SHA256 digest for $CodeProvisioner.jar$;
Encrypt the $TaskExecutor.jar$ to $TaskExecutor.enc$ by the generated password;
forall *LocalSchedule* $lsch_i \in gsch$ ***in parallel*** **do**
  send $CodeProvisioner.jar$, $CodeProvisioner.jks$, $TaskExecutor.enc$ from workflow lib directory to remote machine associated with $lsch_i$ IP
end
forall *LocalSchedule* $lsch_i \in gsch$ **do**
  Set up a SSL/TLS socket connection to each *Code Provisioner* inside each worker node's SGX/SEV TEE;
end
$authenticationValidated \longleftarrow True$;
$totalSuccess \longleftarrow 0$;
forall *LocalSchedule* $lsch_i \in gsch$ ***in parallel*** **do**
  send signal $initialization$ to the remote machines associated with $lsch_i$;
  while *SSL/TLS socket connection is active and authenticationValidated* **do**
    $message \longleftarrow$ response from remote machines associated with $lsch_i$;
    if $message_{firstPart} = codeProvisionJarSHA256Value$ **then**
      if $message_{secondPart} = codeProvisionerJarSHA256$ **then**
        send all the input data files from workflow data directory to remote machine associated with $lsch_i$ IP;
        send $signal\ decryptTaskExecutor$ along with $password$ to remote machines associated with $lsch_i$;
      end
      else
        $authenticationValidated \longleftarrow False$;
      end
    end
    if $message_{firstPart} = decryptionComplete$ **then**
      $totalSuccess \longleftarrow totalSuccess + 1$;
    end
    if $totalSuccess =$ *total local schedules* **then**
      terminate all remote machines by sending signal $terminate$;
    end
  end
end
return $authenticationValidated$;

3.3 SecDATAVIEW Integration

Below, we describe the integration of the WCPAC protocol and modified modules in SecDATAVIEW.

Cloud Resource Management: This module initializes SGX and SEV worker nodes upon receiving the request from the *Workflow Executor*. It implements machine-specific commands to send pre-configured encrypted SGX-LKL (or the SEV disk image) to each worker node and communicates with the worker node's hypervisor using an *ssh* bash session to launch the AMD SEV-protected instance or Intel SGX-LKL enclave. After successfully initializing the worker node, it sends a TEE's hardware and enclave attestation query to the TEE's vendor remote attestation service. Upon a successful remote attestation, it sends the disk image decryption key into the SGX/SEV TEE and JVM's application configuration to run the stand-alone sftp server inside the TEE, then it returns the control to the *Workflow Executor*.

Workflow Engine: The *Workflow Engine* is the heart of the SecDATAVIEW system. This component is responsible for communicating with other components for the successful execution of a workflow. The *Workflow Engine* is depicted in Figure 2(b). We divide the workflow engine into three subsystems: a) *Workflow Planners*, b) *Workflow Executors*, and c) *Code provisioning Attestation*. A user may choose a particular workflow planner for the execution of a workflow.

a) Workflow Planner: The *Workflow Engine* starts with the *Workflow Planner*. Given a workflow and a resource provider, a workflow planner will produce a workflow schedule, which specifies the types of resources, the number of resources for each type, and the mapping of workflow tasks to the resources. Users have the flexibility to choose different workflow planners from the workflow planner pool in SecDATAVIEW. Each workflow planner implements a different workflow scheduling algorithm. A workflow schedule consists of three levels: i)-Task Schedule, ii)-Local Schedule, and iii)-Global Schedule.

i) Task Schedule: A task schedule maps a particular task to one resource, and estimates the start time and finish time of the task on that resource. A task schedule also contains the information of a task's incoming and outgoing data channels to facilitate data movement during workflow execution. The incoming data channels are the incoming edges from its parent tasks and the outgoing data channels are the outgoing edges from the current task to children tasks.

Algorithm 2: Code Provisioner

```
Input  : Input files
Output : Transfer control to TaskExecutor
actualSHA256OfSshd ⟵ calculate SHA256 digest of TEE's SSHD.jar;
if actualSHA256OfSshd ≠ expectedSHA256ValueSshd then
    return;
end
start SSHD Server in a separate thread;
start SSLServerSocket;
accept CodeProvisioningAttestation SSL/TLS connection request;
while SSL/TLS socket is active do
    message ⟵ message from CodeProvisioningAttestation;
    if message_firstPart = initialization then
        SHA_value ⟵ calculate SHA256 digest of TEE's CodeProvisioner.jar;
        send SHA_value with message key codeProvisionerJarSHA256Value to CodeProvisioningAttestation;
    end
    if message_firstPart = decryptTaskExecutor then
        password ⟵ message_secondPart;
        decrypt TaskExecutor.enc to TaskExecutor.jar with password;
    end
    if message_firstPart = terminate then
        close the socket connection;
        initialize the TaskExecutor.jar;
    end
end
```

ii) Local Schedule: A local schedule contains a list of all tasks scheduled for a particular resource. A local schedule prescribes how a sequence of tasks will be executed on a particular resource. However, some incoming data channels may come from another local schedule that is located at a different resource. Similarly, outgoing data channels may reach local schedule on other machines. Local schedule also contains the IP address of the worker node that the local schedule is mapped to.

iii) Global Schedule: A global schedule is the collection of all local schedules for a workflow. When all the local schedules are created, a *global schedule* combines them and then passes the combined schedule to the *Workflow Executor* for execution.

b) Workflow Executor: It is the main subsystem of the *Workflow Engine*. The *Workflow Executor* executes on the master node in a trusted on-premises server and communicates with *Task Executor*s that are executing in remote workers' SGX/SEV TEE. At first, it receives the global schedule from *Workflow Planner* and the location of files that need to be sent to each of the worker nodes. Then it provisions the number of machines with the help of *Cloud Resource Management* module according to the global schedule and assigns an IP address to each of local schedule. Afterward, it transfers the control to the *Code Provisioning Attestation* module to securely send and execute *Task Executor* and workflow data in each worker's SGX/SEV TEE. Once the code provisioning is successful, it securely communicates with *Task Executor*s in the remote worker nodes to complete the workflow job.

c) Code Provisioning Attestation: It is a subsystem of the *Workflow Engine*. The *Code Provisioning Attestation* module is executed on the trusted master node and, provisions the *Task Executor* with the help of *Code Provisioner*. It uses a SHA256 digest message to verify the integrity of the *Code Provisioner* executed inside a remote worker's SGX/SEV TEE. When the integrity of the *Code Provisioner* is verified, the *Code Provisioning Attestation* module sends the *Task Executor*'s decryption key, the workflow's input data, and the Task Executor's SSL certificate to the *Code Provisioner* module to facilitate the *Task Executor* initialization, and returns the control to the *Workflow Executor*. Otherwise, the *Code Provisioning Attestation* terminates the workflow execution due to the code attestation failure. Listing 1 shows the steps in this subsystem.

Task Management: The *Task management* component is responsible for executing the tasks in remote workers. It receives a *local schedule* from the *Workflow Executor* and performs operation accordingly. It contains two subsystems: a) *Code Provisioner* and b) *Task Executor*.

a) Code Provisioner: This is the first subsystem of *Task Management* component by which the *Task Management* layer is initiated. It communicates with the *Code Provisioning Attestation* module and is started by the signal that is received from the *Code Provisioning Attestation* module. After proving its authenticity to the *Code Provisioning Attestation* module, it receives all the necessary files for running the workflow with sftp channel. Through an SSL socket, the *Task Executor*'s decryption key is sent to this subsystem. After a successful decryption of *Task Executor*, it activates the *Task Executor*. Listing 2 shows the steps in *Code Provisioner*.

b) Task Executor: *Task Executor*s are the core subsystem of the *Task Management*. Each *Task Executor* packages all necessary code and libraries used by workflow tasks, executes workflow tasks inside the worker node's SGX/SEV TEE, and communicates with other worker nodes' SGX/SEV TEE. This module actively interacts with the *Workflow Executor* and carries the secret key for cryptography of workflow data and results. In addition, AEAD AES-GCM 256 symmetric cryptography [48], [49] scheme, SSL socket, and sftp channel are used to protect the communication and file transfer between worker nodes. It receives all the required files before starting its procedure and is activated as soon as it receives the starting signal from the *Workflow Executor*. Initially, it gathers information about the confidentiality of each task through a configuration file that is received from the *Code Provisioning Attestation* module. If a task name is in the confidential list, its incoming and outgoing data channels are encrypted and decrypted. In the beginning, all scheduled tasks to a particular *Task Executor* are started at the same time with the help of multi-threading. Then, a particular task evaluates whether all incoming data channels are ready and begins the task execution when all incoming data channels are ready. After a particular task is finished, *Task Executor* prepares its outgoing data channels, applies encryption on results if the task name enlisted in the confidential list and transfers the results to the destination worker. A "job finish" signal is sent to the children workers and the *Workflow Executor*. Finally upon receiving the "terminate" signal from the *Workflow Executor*, and before terminating its thread, the *Task Executor* executes the "clean-up" phase in which every workflow related and intermediate residual files except the encrypted workflow

TABLE 1: Testbed Configuration.

Testbed Machine	SecDATAVIEW Master	Intel SGX	AMD SEV
CPU Model	Intel Xeon E3-1275 v5	Intel Xeon E3-1275 v5	EPYC 7251
CPU Core	4	4	8
CPU Thread	8	8	16
CPU Base Clock	3.6GHz	3.6GHz	2.1GHz
CPU Boost Clock	4GHz	4GHz	2.9GHz
Cache Type	Smart Cache	Smart Cache	L3
Cache Size	8MB	8MB	32MB
Motherboard	Intel FOG	Intel FOG	GIGABYTE MZ31-AR0
Memory	32GB DDR4 Non-ECC	32GB DDR4 Non-ECC	32GB ECC
Storage	NVME SSD	NVME SSD	SATA SSD
Hypervisor/OS	Ubuntu 16.04 LTS	Ubuntu 16.04 LTS	Ubuntu 18.04 LTS
Kernel Version	4.15.0-74-generic-x64	4.15.0-74-generic-x64	4.20.0-sev-x64
SGX SDK Version	N/A	Ver 2.0	N/A
SGX-LKL	N/A	Hardware Mode	N/A
SGX-LKL Memory	N/A	2GB (Encrypted)	N/A
SGX-LKL Storage	N/A	2GB (Encrypted Disk Image)	N/A
SEV VM Kernel	N/A	N/A	5.3.0.29-generic-x64
SEV VM Memory	N/A	N/A	4GB (Encrypted)
SEV VM Storage	N/A	N/A	32GB (Disk Image)

data and results in the the SGX/SEV storage medium are deleted.

4 Evaluation

This section presents the evaluation results of SecDATAVIEW. Specifically, we aim to answer three research questions: (1) What is the performance overhead of running workflows inside SecDATAVIEW? (2) Does SecDATAVIEW preserve its security properties? (3) How is SecDATAVIEW compared with other systems? We used an Intel-based processor machine as the SecDATAVIEW master node, two Intel SGX machines, and two SEV-protected VMs that are running on one AMD EPYC server to conduct experiments. Table 1 shows the configuration of the hardware and software settings for the master and worker nodes. We have also installed Java OpenJDK 1.8 on both the SEV and SGX-LKL disk image. JVM in the SEV worker is allowed to allocate up to $4GB$ of heap memory. For SGX workers, we have compiled the latest SGX-LKL in hardware mode. Each SGX-LKL enclave is set to allocate $2GB$ of heap memory. Also, the JVM runtime in the SGX-LKL enclave is allowed to allocate up to $1GB$ of heap memory. All machines were connected with a 100Mb LAN interface, forming a heterogeneous cluster of five nodes. The source code for all of the experimental workflows is available in the SecDATAVIEW GitHub repository.

4.1 Workflow Performance Evaluation

We measured the performance overhead incurred by SGX/SEV in terms of the execution time and the memory usage for each workflow in ten different configurations that are depicted in Table 2. In the table, "Data Cryptography Active" refers to the scenario when both task code and data are encrypted during the workflow execution and file transfer, and then decrypted before their usage. "Data Cryptography Inactive" refers to the scenario when the task code is encrypted and decrypted, but the data is not encrypted during the workflow execution and file transfer.

4.1.1 The Diagnosis Recommendation Workflow

This experiment deals with a real-life diagnosis recommendation workflow [24] involving machine learning methods and raw textual dataset that provides the prescription for a group of patients. Since invoking machine learning models requires extensive computation both for the training and the testing datasets, we examine how the overall execution time and memory footprint are affected due to running

TABLE 2: SecDATAVIEW settings for experimental workflows.

SecDATAVIEW Setting	TEE Setting	Data Cryptography
SGX inactive without data encryption	Inactive	Inactive
SGX inactive with data encryption	Inactive	Active
SGX active without data encryption	Active	Inactive
SGX active with data encryption	Active	Active
SEV inactive without data encryption	Inactive	Inactive
SEV inactive with data encryption	Inactive	Active
SEV active without data encryption	Active	Inactive
SEV active with data encryption	Active	Active
Hybrid TEE active without data encryption	Active	Inactive
Hybrid TEE active with data encryption	Active	Active

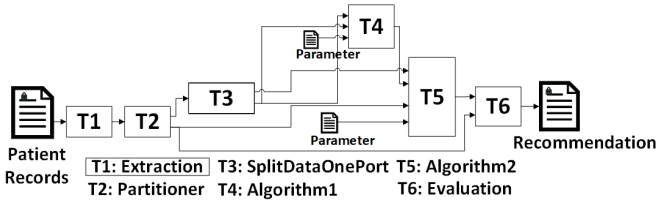
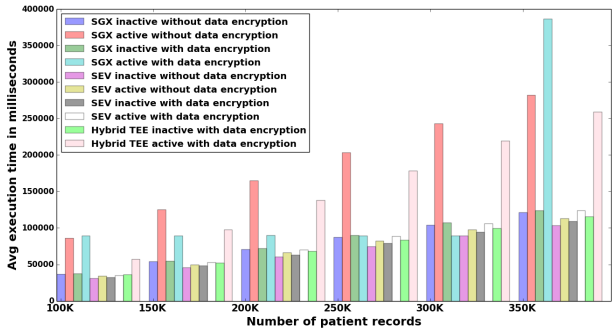


Fig. 4: The Diagnosis Recommendation Workflow [24].

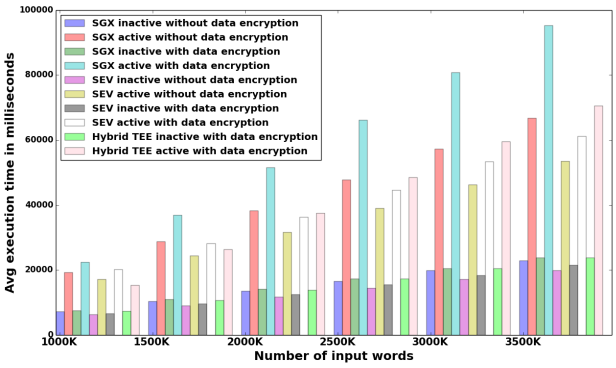
every task of the workflow in Figure 4 inside a worker's TEE. Here, we synthetically create patient records in the size varying from $100k$ to $350k$ patients with an average length of 150 characters for an individual prescription. We conducted the experiments with 10 different SecDATAVIEW settings mentioned earlier. For each scenario, we used 75% of the random dataset for training, and the rest were for testing. Figure 5 (a) shows the average workflow execution time of ten trials in milliseconds. We observe that the training and testing of the machine learning models for relatively bigger datasets demand relatively a long period of time and as a result, secure execution demands more time span. For the $350K$ dataset and with TEE and encryption active setting, the results show that the SGX, SEV, and the "Hybrid" setting with two SGX and two SEV workers impose $3.11X$, $1.13X$, and $2.24X$ performance overhead, respectively. Also, the cryptography overhead inside TEE shows that SGX imposes $1.37X$ and SEV imposes $1.09X$ performance overhead compared to the baselines where encryption was not used in the TEE. We also conducted experiment that captures the total allocated memory and the total number of active processing threads in the workflow. The results show that up to $459MB$ heap memory and 29 active threads were used in the diagnosis recommendation workflow. Table 3 shows the total memory usage for the workflow execution inside a worker node. Figure 7 depicted the detailed distribution time span of TEE and cryptography overhead for the experimental workflows.

4.1.2 Word Count (Map-Reduce) Workflow

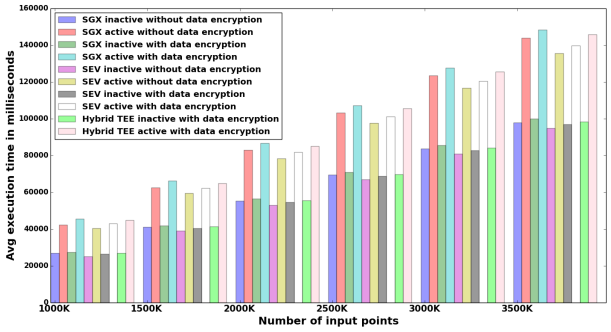
In this experiment, we consider a well-known word-count example for the Map-Reduce [25] operation to investigate the execution timespan and memory footprint with 10 different SecDATAVIEW settings mentioned earlier. We create a workflow involving 16 tasks including one task for input processing, six (three Splitting + three Mapping) tasks for map operation, eight tasks (four Shuffling + four Reducing) for the reduce operation, and one task for the final output organization. At first, we randomly generate words with a length of two characters in the size varying from $1,000K$ to $3,500K$. In the first task, the inputs are equally distributed into three different Splitting tasks. Figure 5 (b)



(a) Diagnosis Recommendation Workflow.



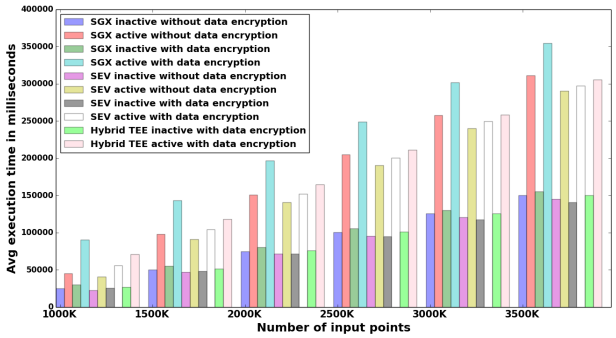
(b) Word Count (Map-Reduce) Workflow.



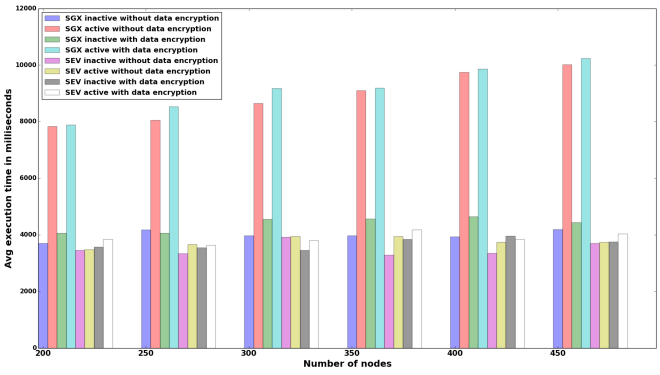
(c) Distributed K-means Workflow.

Fig. 5: Execution times of running different workflows in different configurations for different input datasets.

depicts the overall timespan comparison for secure and the baseline executions with different settings. It is observed that even though the number of words has been increased, the timespan is slightly enlarged. For the $3,500K$ words and with TEE and encryption active setting, the results show that SGX, SEV, and the "Hybrid" setting with two SGX and two SEV workers impose $4X$, $2.85X$, and $2.96X$ performance overhead, respectively. Also, the cryptography overhead inside TEE shows that SGX imposes $1.42X$ and SEV imposes $1.14X$ performance overhead compared to the baselines where encryption was not used in the TEE. We also conducted an experiment that captures the total allocated memory and the total number of active processing threads in the workflow. The results show that up to $556MB$ heap memory and 31 active threads were used in the word-count



(a) MONTAGE Workflow.



(b) Neural Network Workflow.

Fig. 6: Execution times of running different workflows in different configurations for different input datasets.

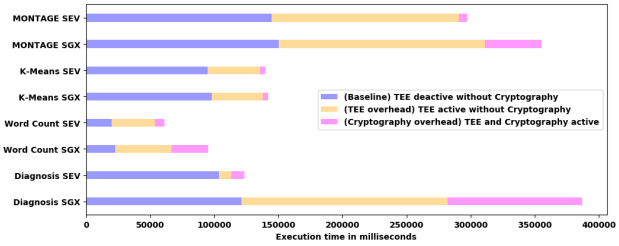


Fig. 7: The distribution of the TEE and cryptography overhead in workflow executions.

Map-Reduce workflow. Table 3 and 4 show the total memory usage and the cryptography overhead for the workflow execution inside a worker node.

4.1.3 *The Distributed K-means Workflow*

We measured the execution time and memory usage of SecDATAVIEW using a Distributed K-means workflow [3], where several clusters and the number of splits of datasets are designed dynamically. In this experiment, we randomly generated $1000K$ to $3500K$ points, each of which has an x and a y coordinate. Figure 5 (c) shows the execution time of SecDATAVIEW with 10 different settings. We observe that SGX, SEV, and the "Hybrid" approach with two SGX and two SEV workers impose $1.48X$, $1.44X$ and $1.48X$ overhead on the largest dataset ($3500K$ points), respectively. Also, results showed that running the distributed K-means

3. https://www.flickr.com/photos/waynestateise/47529826741/

TABLE 3: Memory footprints of experimental workflows.

Workflow	Max Heap	Max Threads
Diagnosis	$459MB$	29
Map-Reduce	$556MB$	31
K-Means	$476MB$	32
MONTAGE	$359MB$	27
Neural Network	$496MB$	22

TABLE 4: Cryptography overhead inside the TEE.

Workflow	SGX	SEV
Diagnosis	$1.37X$	$1.09X$
Map-Reduce	$1.42X$	$1.14X$
K-means	$1.03X$	$1.03X$
MONTAGE	$1.14X$	$1.02X$
Neural Network	$1.02X$	$1.07X$

workflow used $476MB$ heap and 32 active processing threads, that is represented in Table 3. Table 4 shows the cryptography overhead for this workflow.

4.1.4 The MONTAGE Workflow

To evaluate how SecDATAVIEW performs when a workflow construct that uses a heavily-connected DAG (i.e., the graph edge to node ratio is a large number), we consider a MONTAGE workflow [26]. The MONTAGE workflow was created with ten different tasks. The number of randomly generated integers ranges between $1,000K$ to $3,500K$. Each of the tasks is responsible for sorting the given number that is randomly generated through a completely uniform distribution of the numbers. When the numbers are populated, a merge sort is performed among the given numbers so that we can effectively measure the performance overhead of the SecDATAVIEW system for the sorting and extra memories to complete the procedure. Figure 6 (a) shows the execution time of SecDATAVIEW with 10 different settings. We observe that SGX, SEV, and the "Hybrid" approach with two SGX and two SEV workers impose $2.28X$, $1.98X$ and $2.03X$ overhead on the largest dataset ($3,500K$), respectively. Also, running the MONTAGE workflow uses $359MB$ heap and 27 active processing threads, which is represented in Table 3. Table 4 shows the cryptography overhead for this workflow.

4.1.5 The Neural Network Workflow

Figures 6 (b) shows the execution time of SecDATAVIEW with 8 different settings. In this workflow, we consider a two-layer feed-forward Neural Network (NN) for our experiment. Since SecDATAVIEW considers the directed acyclic graph only, we can not consider the traditional deep learning workflow, i.e., splitting the tasks into different pieces with a circular graph. As a result, we examine everything in a single task for representing the whole scenario. This NN recognizes the X-OR operation with the help of heavy matrix calculation that slows down the overall execution when it comes to processing large data sets. We want to investigate how SecDATAVIEW performs this heavyweight matrix computation. The core concept could be scaled for any number of layers and nodes. In this experiment, we want to examine the performance overhead of different number of input nodes. We observe that SGX and SEV impose $2.3X$ and $1.07X$ performance overhead, respectively. "Hybrid" approach was not used for the NN workflow since the neural network algorithm is implemented as a single workflow task to support iterative processing necessary for training an NN model. Finally, the NN workflow with the largest dataset used $496MB$ heap memory and 22 active processing threads as depicted in Table 3. Table 4 shows the cryptography overhead for this workflow.

4.2 Security Analysis

The SecDATAVIEW architecture and TCB: The SecDATAVIEW architecture provides small software and hardware TCB for deploying a big data workflow management system in the cloud. For SGX workers, the software components of TCB are the LibOS, the JVM, the Code Provisioner, and the Task Executor. For SEV workers, the software components of TCB are the guest OS, the JVM, the Code Provisioner, and the Task Executor. The hardware components of the TCB are the CPU package for the SGX workers and are AMD SoC and AMD secure processor for the SEV worker. The SecDATAVIEW architecture excludes all the underlying and high-privileged cloud system software (i.e., hypervisor and cloud management software) from the TCB. Besides, SecDATAVIEW is protected against memory corruption vulnerabilities (e.g., buffer overflow) since memory access is protected by type-safe Java language and JVM.

Workflow code and data confidentiality and integrity: The SecDATAVIEW architecture protects the confidentiality and integrity of the workflow's code and data at the booting time and runtime with the help of TEEs. TEEs are attested through the hardware attestation method that is provided by TEE hardware vendors (i.e., Intel and AMD). Besides, SecDATAVIEW uses different security primitives such as AEAD scheme, one-way hash function, SSL, and SFTP channels. Specifically, SecDATAVIEW uses authenticated encryption with associated data (AEAD). The associated data is validated, but not combined in the ciphertext. However, the Initialization Vector (IV) that is used to generate the AEAD is implicitly integrated within the ciphertext. We assume that AEAD is secure [50].

SecDATAVIEW cloud's SGX hardware and enclave attestation: SecDATAVIEW uses the TEE attestation mechanism provided in SGX-LKL, which is an Intel-based attestation approach, for remote attestation. The Intel-based attestation approach uses the Enhanced Privacy ID (EPID) scheme [51] to ensure the anonymity of the SGX platform, which uses a group signature to allow the SGX platform to generate a signature without leaking an identity. The anonymity is provided as each EPID group contains many SGX platforms [51]. Although the Intel-based attestation is universally used for SGX TEE, the Intel-base attestation requires the direct involvement of Intel during the attestation process and hence Intel may learn who requested the Attestation (i.e., SecDATAVIEW owner) through the unique registered Service Provider ID (SPID) and SecDATAVIEW IP address that may reveal the possible location of the SecDATAVIEW server when the SecDATAVIEW master node connects to the Intel Attestation Servers. Also, Intel may learn what enclave is being attested and who signed the enclave through the enclave measurement primitives (i.e., MRENCLAVE and MR-

SIGNER values) that are sent to Intel during the attestation process. One possible mitigation for above-mentioned identity leakage is to leverage third-party attestation. We identify two third-party attestation approaches: Intel-provided on-premises third-party attestation for data centers [52] and OPERA [28] which is the Internet-based third-party Attestation. The third-party attestation platform should be owned and operated by the SecDATAVIEW and data owner when the security of the platform is the utmost important factor.

The WCPAC protocol: SecDATAVIEW uses the WCPAC protocol to 1) provision and attest worker nodes, 2) provision the code for the *Task Executor* and workflow tasks on each participating worker node, 3) establish the secure communication and file transfers between the master node and worker nodes, and 4) ensure the secure file transfers among worker nodes. The WCPAC protocol protects the SecDATAVIEW network connectivity by establishing an SSL socket connection for messaging and the SFTP for file transferring between active workers. WCPAC is protected against eavesdropping, the man-in-the-middle attack, and the replay attack.

Attacks against network channel: Assume that an adversary actively eavesdrops on the communication among different workers. The adversary may learn the source, the destination, the number of transmitting packets, the time when the message was sent, and the total size of the transferred message. Conversely, the adversary cannot know the content carried by the packet's payload due to our multi-layer protection mechanisms. First, the communication is protected with the SSL protection. Even if the adversary breaks the SSL cryptography protection, the payload is protected with the AEAD encryption and the adversary needs to break the second layer of cryptography protections, which decreases the chance of successful attacks.

Access pattern leakage attack: SecDATAVIEW could be vulnerable to access pattern leakage attack when it executes workflows whose DAG construct is well-known and predictable to the adversary. For example, in a Map-Reduce workflow, all values with the same key are sent to the same reducer. If an adversary can infer or count the total number of pairs received by a reducer node from other mappers, it can leak some information about the result. In Map-Reduce workflow, the chance of the successful information leakage is increased when the number of keys in the key-value pairs processed by reducer is small (e.g., processing vote between two presidential candidates). However, if the number of reducers is high, the distribution of values to each reducer (key) could leak negligible amount of information. The access pattern leakage attack is a common vulnerability in most Map-Reduce frameworks and even in a secure Map-Reduce framework such as VC3 [16]. In SecDATAVIEW, the mitigation solution is workflow-specific and should be addressed during the workflow design by the workflow owner. To provide the general-purpose characteristics and support workflows with different requirements, the SecDATAVIEW engine is not confined to specific workflow data structures (e.g., SQL query, Map-Reduce, etc). Having a general built-in solution at the system level that mitigates access pattern leakage attacks for all workflows with different data structures and data stream models is still an open research challenge. Currently, for a Map-Reduce workflow, the workflow designer could assign more than one reducer task to each worker node that hides the actual distribution of values to each reducer or include additional workflow tasks similar to the proposal in [40], [53] to suppress the access pattern leakage attacks. In the same way, for a SQL query workflow, the workflow owner and designer could adapt the data stream obliviousness techniques discussed in [19] during designing SQL query workflow.

The denial of service (DoS) attack: SecDATAVIEW is vulnerable to the DoS attacks, but this attack is also present in all SGX and SEV TEEs. For SGX, the DoS attack is mainly caused by a malicious host that refuses to launch the enclave or services the enclave requests. In SEV, it could be caused by a malicious hypervisor that refuses to start the SEV-protected VM or by attackers who modify the SEV-protected memory image and due to the lack of the SEV memory integrity protection, causing the VM to crash or exhibit unexpected behavior. Another DoS attack vector that presents only in the SGX server happens when a malicious enclave application implements the Rowhammer attack [54] on the enclave protected memory region and modifies data in that protected region. Violating the integrity of the enclave memory causes the memory integrity protection policy in the Intel SGX initiated, which puts the CPU in system-wide lockdown that can only be fixed via a hard cold reboot. The permanent countermeasure for the DoS attack that is caused by the Rowhammer attack depends upon the availability of Rowhammer-free DRAM that thoughtfully discussed in [54]. Permanent countermeasure for system software level DoS attack can only be addressed via hardware vendors by removing the dependency of TEE from unprotected system software. One possible solution is to use a dedicated, trustworthy, and isolated integrity protected System on Chip (SoC) to handle requests that relate only to the TEEs. However, how to develop such a system is still an open research challenge. DoS attack on SecDATAVIEW does not leak any sensitive information and only affects the progress of the workflow execution, which can be easily detected by the user. The user can relaunch the workflow on a different cloud or use a different worker node to counter that attack.

The side-channel attack: SecDATAVIEW is vulnerable to the side-channel attack that is present in every SGX [41], [55] and SEV [15] TEE.

4.3 Comparison with Existing Big Data Systems

Table 5 compares SecDATAVIEW against several representative big data systems including VC3 [16], Opaque [19], and the lightweight Lua Map-Reduce system [17].

Functionality: SecDATAVIEW has two main advantages compared to the existing systems: 1) it is compatible with many forms of data structures/formats, and 2) it is capable of executing workflows by leveraging a heterogeneous computing setting (i.e., SGX and SEV). VC3, the lightweight Lua Map-Reduce, and Opaque are limited to Map-Reduce

TABLE 5: A comparison with existing TEE-based big data systems.

Feature	SecDATAVIEW	VC3 [16]	Opaque [19]	Lua Map/Reduce [17]
Data confidentiality	AES-GCM-256	AES-GCM-128	AES-GCM-128	AES-CTR-128
Data integrity	Authenticated Encryption	Authenticated Encryption	Authenticated Encryption	No
Intel SGX	Yes	Yes	Yes	Yes
AMD SEV	Yes	No	No	No
Data structure compatibility	All types of workflow	Map-Reduce	SQL query	Map-Reduce
Job integrity verification	No	Yes	Yes	No
Access pattern leakage protection	No	No	Yes	No
Access pattern leakage overhead	N/A	N/A	1.6X-46X (oblivious mode)	N/A
Job performance overhead	1.48X-2.96X (Hybrid mode)	1.04X-1.08X (base-encrypted mode)	0.52X-3.3X (encrypted mode)	1.3X-2X (encrypted mode)

and SQL query workflows, respectively. Besides, they only support SGX TEE.

Security: SecDATAVIEW and the lightweight Lua Map-Reduce use the managed code (Java/Lua) that is protected against memory corruption vulnerabilities (e.g., buffer overflow). VC3 uses C/C++ and offers an execution mode in which the integrity of the enclave memory region is evaluated. However, when this feature is activated, the performance overhead is increased to $1.27X$. Among the compared systems, Opaque and VC3 offer job execution verification. In SecDATAVIEW, since the structure of workflows and the size of input files do not need to follow a pre-defined data structure (i.e., Map-Reduce or query), having a general verification model to be applied in many forms of workflow is an open research challenge. Among the compared systems, only Opaque provides the protection against access pattern leakage attack. However, it is based on the oblivious computation, which imposes up to $46X$ overhead on the job execution time.

Performance: SecDATAVIEW imposes moderate overhead, a range between $1.48X$-$2.96X$ in the "Hybrid" operation with different workflow data structures. Among compared systems, VC3 is fastest when it operates without enclave memory region checking. However, when VC3 activates the enclave memory region checking, its performance is competitive with SecDATAVIEW (i.e., VC3 imposes about $1.27X$ overhead and SecDATAVIEW imposes about $1.48X$ overhead when a well-optimized task scheduling algorithm is used). Additionally, SecDATAVIEW outperforms Opaque ($2.96X$ vs $3.3X$ overhead) and has higher overhead than the lightweight Lua Map-Reduce ($2X$ overhead).

5 Related Work

Bertino *et al.* [1] and Ye *et al.* [56] provided comprehensive studies on data security and privacy requirements as well as existing research challenges for providing security and privacy in the big data context. Qui *et al.* [57] presented a comprehensive study on existing research advances and open research challenges on machine learning for big data. Brenner *et al.* proposed Securekeeper [38] that uses Intel SGX to protect the confidentiality of ZooKeeper coordination service. Considering the enclave programming spectrum, the Securekeeper used the *Specialized-Enclave* with Java JNI approach to call the SGX primitives in native C/C++, which helped it to maintain a small size of TCB. SecureKeeper imposes 32.18% overhead compared to the base ZooKeeper. In another work, researchers proposed SGX-Spark [58] that used SGX-LKL to run unmodified Java applications in the SGX enclaves. Reported results in [59] show that SGX-Spark imposes about $4X$ - $5X$ performance overhead using $32MB$ of a medical dataset and with vanilla Spark. Still, SecDATAVIEW offers the flexibility of leveraging heterogeneous cloud (i.e., AMD SEV and Intel SGX) and supports different types of workflows. Recently Jiang *et al.* proposed URANUS [60] as an SGX-aware JVM to run Java applications in the SGX TEE. URANUS decreases the TCB of Java execution environments via porting only essential JVM components inside the SGX (i.e., GC, dynamic code loader, JIT, and exception handler). In addition, URANUS introduced two new compiler annotations (i.e., JECall and JOCall) to the Java programming model that developers should use to identify sensitive parts of the code that need to be executed inside the SGX enclaves. URANUS has been tested with ZooKeeper and Spark. Experimental results show that ZooKeeper-URANUS imposes 19.4% performance overhead compared to the native (insecure) ZooKeeper. Also, URANUS-Spark imposes $1.2X$ - $7.6X$ performance overhead compared to native Spark. URANUS requires Java code refactoring. Compared with URANUS, SecDATAVIEW is compatible with heterogeneous cloud (i.e., AMD SEV and Intel SGX), and supports unmodified Java application that is portable to every Java runtime environment. Recently Tsai *et al.* proposed Civet [61] that uses a modified JVM, a Java class partitioning tool, dynamic taint-tracking, and Graphene-SGX for partitioning a Java application into trusted and untrusted classes. Trusted classes execute inside enclaves, and untrusted classes run outside enclaves. Civet results show a performance overhead of about 16% – 22% without and 70% – 80% with taint-tracking for partitioning and shielding a Hadoop mapper (RegexMapper) in a Hadoop regular expression parser that used for searching a regular expression inside the 1GB dataset. Compared with SecDATAVIEW that supports heterogeneous clouds (i.e., AMD SEV and Intel SGX), Civet is only bound to the Intel SGX platform. Schuster *et al.* proposed VC3 [16] that works with unmodified Hadoop and uses Intel SGX to protect Map-Reduce code and job execution. In VC3, all Map-Reduce jobs run inside the enclave with one executing thread (i.e., no multi-threading is used). Additionally, all data traffic of intermediate Map-Reduce results is kept encrypted during the job execution. Experimental results show that VC3 imposes 4.3% - 24.5% performance overhead when the enclave self-integrity checking mode is used. Pires *et al.* [17] proposed an SGX-based lightweight and secure Map-Reduce framework. The system is integrated with a lightweight virtual machine for the Lua language [18], which is a high-level language that interprets the Map-Reduce Lua scripts, and a Secure Content Based Routing System, which is a secure publish/subscribe system for the message passing and data distribution between the client and worker nodes in the distributed system. In this system, three main entities – client, SCBR, and worker nodes – collaborate to execute

a Map-Reduce workflow. All message routing as well as the execution of the map and reduce Lua scripts occurs inside the secure enclave. Their experimental results show that their system imposes up to $2X$ performance overhead. Zheng *et al.* [19] proposed Opaque that enhances the security of the Spark SQL with SGX. One execution mode, called the encryption mode, provides the confidentiality protection on the data and results. In this mode, the Opaque's code at the client side is transferred to the enclave and with the help of the Intel attestation protocol, the code is verified and the secret keys are distributed inside the enclave. Their experimental results show that the Opaque's encryption mode imposes $3.3X$ performance overhead. Moreover, Opaque uses the oblivious mode and the oblivious pad mode to provide protection against the access pattern leakage and the size leakage with the help of oblivious computations. Opaque's experimental results showed that the oblivious mode imposes $1.60X$ to $46X$ performance overhead. Intel recently announced Trusted Domain eXtensions (TDX) [62] as its next-generation TEE for cloud applications. Intel TDX, similar to AMD SEV, designed to provide hardware-isolated VM with a large amount of secure memory and processing resources. Upon its availability, Intel TDX can also be integrated into SecDATAVIEW with proper engineering effort and as another secure cloud TEE for big data workflow execution.

6 Conclusions

In this paper, we present SecDATAVIEW, an efficient and secure big data scientific workflow management system to protect the confidentiality and integrity of Java-written tasks and data in the workflow with the help of Intel SGX and AMD SEV TEEs. SecDATAVIEW significantly reduces the TCB size of the worker node to the shielded code that belongs to the *Task Executor*, individual workflow tasks, and their execution environment running inside the SGX/SEV TEE. Our experimental results with different types of workflows show the usability of SecDATAVIEW with acceptable performance overhead, while securing confidential task execution at runtime. In the future, we plan to develop a trusted execution environment (TEE) to secure general-purpose GPU computing (GPGPU) in big data context, which would help secure big data management systems that leverage the enormous computing power of GPU accelerators in untrusted cloud environments.

7 Acknowledgement

We would like to thank the anonymous reviewers for their insightful comments that helped improve this paper. Ahmed's current affiliation is with Google; his contributions to this work were primarily made while he was a student at Wayne State University. This project is supported in part by the National Science Foundation under grant NSF OAC-1738929. Heming Cui is supported by the research grants from the HKU-SCF FinTech Academy R&D Funding Scheme, HK RGC GRF (17202318, 17207117), and a Croucher Innovation Award.

References

[1] Bertino, "Big data - security and privacy," *Big Data Congress*, 2015.
[2] T. White, *Hadoop: The definitive guide*. "O'Reilly Media, Inc.", 2012.
[3] S. Bugiel, S. Nürnberger, T. Pöppelmann, A.-R. Sadeghi, and T. Schneider, "AmazonIA: when elasticity snaps back," in *CCS*, 2011.
[4] D. Perez-Botero, J. Szefer, and R. B. Lee, "Characterizing hypervisor vulnerabilities in cloud computing servers," in *Proceedings of the international workshop on Security in cloud computing*, 2013.
[5] K. Kortchinsky, "Cloudburst: A vmware guest to host escape story," *Black Hat USA*, vol. 19, 2009.
[6] R. Wojtczuk, J. Rutkowska, and A. Tereshkin, "Xen 0wning trilogy," *Invisible Things Lab*, 2008.
[7] F. Rocha and M. Correia, "Lucy in the sky without diamonds: Stealing confidential data in the cloud," in *DSN Workshops*, 2011.
[8] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *CCS*, 2009.
[9] i. Black Duck Software, "Black duck open hub," https://www.openhub.net/p?query=xen&sort=relevance, Last accessed on 11/02/2021.
[10] "National institute of standards, national vulnerability database," https://nvd.nist.gov/, Last accessed on 11/02/2021.
[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 *HASP@ ISCA*, 2013.
[12] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *HASP@ ISCA*, 2013.
[13] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions." in *HASP@ ISCA*, 2013.
[14] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption," *White paper*, 2016.
[15] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of intel sgx and amd memory encryption technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP@ ISCA)*, 2018.
[16] A. Baumann, M. Peinado, and G. Hunt, "VC3: Trustworthy data analytics in the cloud using SGX," in *IEEE S&P*, 2015.
[17] R. Pires, D. Gavril, P. Felber, E. Onica, and M. Pasin, "A lightweight MapReduce framework for secure processing with SGX," in *CCGRID*. IEEE, 2017.
[18] A. Hirschi, "Traveling light, the lua way," 2007.
[19] 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 *NSDI*, 2017.
[20] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *TOCS*, 2015.
[21] 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," *arXiv preprint*, 2019.
[22] R. W. Graves and A. Pitarka, "Broadband ground-motion simulation using a hybrid approach," *Bulletin of the Seismological Society of America*, 2010.
[23] A. Kashlev and S. Lu, "A system architecture for running big data workflows in the cloud," in *IEEE SCC*, 2014.
[24] I. Ahmed, S. Lu, C. Bai, and F. A. Bhuyan, "Diagnosis recommendation using machine learning scientific workflows," in *Big Data Congress*, 2018.
[25] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, 2008.
[26] R. Sakellariou, H. Zhao, and E. Deelman, "Mapping workflows on grid resources: experiments with the montage workflow," in *Grids, P2P and services computing*, 2010.
[27] S. Mofrad, I. Ahmed, S. Lu, P. Yang, H. Cui, and F. Zhang, "Secdataview: A secure big data workflow management system for heterogeneous computing environments," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019.
[28] G. Chen, Y. Zhang, and T.-H. Lai, "Opera: Open remote attestation for intel's secure enclaves," in *CCS*, 2019.
[29] Saraladevi *et al.*, "Big data and hadoop-a study in security perspective," *Procedia computer science*, vol. 50, pp. 596–601, 2015.
[30] Nelson *et al.*, "Security and privacy for big data: A systematic literature review," in *IEEE International Conference on Big Data*, 2016.

[31] Terzi *et al.*, "A survey on security and privacy issues in big data," in *International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015.

[32] A. Kashlev, S. Lu, and A. Mohan, "Big data workflows: a reference architecture and the DATAVIEW system," *STBD*, 2017.

[33] F. Bhuyan, S. Lu, I. Ahmed, and J. Zhang, "Predicting efficacy of therapeutic services for autism spectrum disorder using scientific workflows," in *International Conference on Big Data*, 2017.

[34] H. Hamidian, S. Lu, S. Rana, F. Fotouhi, and H. Soltanian-Zadeh, "Adapting medical image processing tasks to a scalable scientific workflow system," in *IEEE World Congress on Services*, 2014.

[35] X. Fei and S. Lu, "A dataflow-based scientific workflow composition framework," *IEEE Transactions on Services Computing*, 2010.

[36] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua, "A reference architecture for scientific workflow management systems and the view soa solution," *IEEE Transactions on Services Computing*, 2009.

[37] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, and M. L. Stillwell, "SCONE: Secure linux containers with Intel SGX," in *OSDI*, 2016.

[38] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, "SecureKeeper: Confidential ZooKeeper using intel SGX," in *Middleware*, 2016.

[39] J.-F. Raymond, "Traffic analysis: Protocols, attacks, design issues, and open problems," in *Designing Privacy Enhancing Technologies*, 2001.

[40] T. T. A. Dinh, P. Saxena, E.-C. Chang, B. C. Ooi, and C. Zhang, "M2R: Enabling stronger privacy in MapReduce computation." in *USENIX Security*, 2015.

[41] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *CCS*, 2017.

[42] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, 2012.

[43] Intel, "SGX-SDK developer guide," https://software.intel.com/en-us/documentation/sgx-developer-guide, 2017.

[44] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: A practical library os for unmodified applications on sgx," in *USENIX ATC*, 2017.

[45] LSDS, "Sgx-lkl,remote attestation," https://github.com/lsds/sgx-lkl/wiki/Remote-Attestation-and-Remote-Control, 2018.

[46] R. Buhren, C. Werling, and J.-P. Seifert, "Insecure until proven updated: Analyzing amd sev's remote attestation," in *CCS*, 2019.

[47] I. Ahmed, S. Mofrad, S. Lu, C. Bai, F. Zhang, and D. Che, "Seed: Confidential big data workflow scheduling with intel sgx under deadline constraints," in *2020 IEEE International Conference on Services Computing (SCC)*. IEEE, 2020.

[48] P. Rogaway, "Authenticated-encryption with associated-data," in *CCS*, 2002.

[49] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[50] P. D'Avilar, J. D'Errico, K. Berends, and M. Peck, "Reading guide 3: Authenticated encryption," 2004.

[51] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel® software guard extensions: Epid provisioning and attestation services," *White Paper*, 2016.

[52] V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski, "Supporting third party attestation for intel® sgx with intel® data center attestation primitives," *White Paper*, 2018.

[53] Ohrimenko *et al.*, "Observing and preventing leakage in MapReduce," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[54] Jang *et al.*, "Sgx-bomb: Locking down the processor via rowhammer attack," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, 2017.

[55] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yuval, B. Sunar, D. Gruss, and F. Piessens, "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection," in *(IEEE S&P)*, 2020.

[56] H. Ye, X. Cheng, M. Yuan, L. Xu, J. Gao, and C. Cheng, "A survey of security and privacy in big data," in *2016 16th international symposium on communications and information technologies (iscit)*. IEEE, 2016.

[57] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, pp. 1–16, 2016.

[58] LSDS, "Lsds sgx-spark github," https://github.com/lsds/sgx-spark, 2019.

[59] C. Segarra, R. Delgado-Gonzalo, M. Lemay, P.-L. Aublin, P. Pietzuch, and V. Schiavoni, "Using trusted execution environments for secure stream processing of medical data," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2019.

[60] 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 *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020.

[61] 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 *29th USENIX Security Symposium*, 2020.

[62] Intel, "Trust domain extensions," https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf, 2020.

Saeid Mofrad is a Ph.D. candidate in the Department of Computer Science at Wayne State University, USA. He received his M.S. in Computer Science in 2016 from Eastern Michigan University, USA, where he has been awarded the National Scholars Program (NSP) scholarship for two consecutive years. Currently, he is a member of the Computer and System Security Laboratory (COMPASS) under the supervision of Dr. Fengwei Zhang and Big Data Research Laboratory under the supervision of Prof. Shiyong Lu. He is also a co-reviewer in the IEEE Big Data and IEEE Access venues. His current research focuses on systems security, hardware-assisted TEE in x86 architecture, and big data scientific workflows security and privacy.

Ishtiaq Ahmed received his Ph.D. degree from the Department of Computer Science at Wayne State University, USA, in 2019. He received his M.S. degree in 2014 from the Department of Computer Engineering, Kyung Hee University, South Korea, where he has been awarded the South Korea Global IT scholarship for two consecutive years. He is a member of the Big Data Research Laboratory under the supervision of Prof. Shiyong Lu. He received the best student paper award in IEEE Big Data Congress 2018. He is also a co-reviewer in IEEE Big Data, IEEE Big Data Congress and, IEEE SCC venues. His current research includes big data scientific workflows, workflow scheduling algorithms, big data security and privacy, machine learning and, data science.

Fengwei Zhang received the Ph.D. degree in computer science from George Mason University in 2015. He is currently an Associate Professor and the Director of the Computer and Systems Security Laboratory, Department of Computer Science and Engineering, Southern University of Science and Technology. His research interests are in the areas of systems security, with a focus on trustworthy execution, transparent malware debugging, hardware-supported security, and plausible deniability encryption. He received the Distinguished Paper Award from ACSAC in 2017.

Shiyong Lu Ph.D., is a Professor in the Department of Computer Science at Wayne State University, and the director of the Big Data Research Laboratory. Dr. Lu received his Ph.D. in computer science from Stony Brook University in 2002. Dr. Lu's current research interests focus on scientific workflows, big data security, services computing, and provenance management. Dr. Lu is an author of two books and over 120 articles published in various international journals and conferences. He is the founding chair of the IEEE International Workshop on Scientific Workflows (SWF) and a founding editorial board member of International Journal of Big Data. He is a senior member of the IEEE.

Ping Yang is currently an Associate Professor in the Department of Computer Science at State University of New York at Binghamton. She got her Ph.D. in Computer Science from State University of New York at Stony Brook in 2006, M.E. from Chinese Academy of Sciences in 1999, and B.S. from Sun Yat-sen University in 1996. Her research interests include cybersecurity, privacy, blockchain, access control, formal method, and cloud computing.

Heming Cui received the Ph.D. degree from Columbia University, New York City, NY, USA, in 2014. He is currently an Associate Professor with the Department of Computer Science, The University of Hong Kong. His research interests include operating systems, programming languages, distributed systems, and cloud computing, with a particular focus on building software infrastructures and tools to improve reliability and security of real-world software.