SecDATAVIEW: A Secure Big Data Workflow Management System for Heterogeneous Computing Environments

Saeid Mofrad
Department of Computer Science
Wayne State University
saeid.mofrad@wayne.edu

Ishtiaq Ahmed
Department of Computer Science
Wayne State University
ishtiaq@wayne.edu

Shiyong Lu
Department of Computer Science
Wayne State University
shiyong@wayne.edu

Ping Yang
Department of Computer Science
State University of New York at Binghamton
pyang@binghamton.edu

Heming Cui
Department of Computer Science
The University of Hong Kong
heming@hcu.hk

Fengwei Zhang*
Department of Computer Science
Wayne State University
fengwei@wayne.edu

ABSTRACT

Big data workflow management systems (BDWFMSs) have recently emerged as popular platforms to perform 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 were developed to address this problem, they are limited to specific structures such as Map-Reduce-style workflows and SQL queries. This paper proposes SecDATAVIEW, a BDWFMS that leverages Intel Software Guard eXtensions (SGX) and AMD Secure Encrypted Virtualization (SEV) to develop a heterogeneous trusted execution environment for workflows. SecDATAVIEW aims to (1) provide the confidentiality and integrity of code and data for workflows running on public untrusted clouds, (2) minimize the TCB size for a BDWFMS, (3) enable the trade-off between security and performance for workflows, and (4) support the execution of Java-based workflow tasks in SGX. Our experimental results show that SecDATAVIEW imposes $1.69x$ to $2.62x$ overhead on workflow execution time on SGX worker nodes, $1.04x$ to $1.29x$ overhead on SEV worker nodes, and $1.20x$ to $1.43x$ overhead on a heterogeneous setting in which both SGX and SEV worker nodes are used.

CCS CONCEPTS

• **Security and privacy**;

* The corresponding author. The work was done at WSU and he is currently affiliated with SUSTech.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACSAC '19, December 9–13, 2019, San Juan, PR, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7628-0/19/12…$15.00
https://doi.org/10.1145/3359789.3359845

KEYWORDS

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

ACM Reference Format:
Saeid Mofrad, Ishtiaq Ahmed, Shiyong Lu, Ping Yang, Heming Cui, and Fengwei Zhang. 2019. SecDATAVIEW: A Secure Big Data Workflow Management System for Heterogeneous Computing Environments. In *2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3359789.3359845

1 INTRODUCTION

Recently, scientific workflows increasingly use cloud computing to provision on-demand scalable resources in computation and storage for large-scale big data analytics [30, 37]. Cloud computing has the advantage of being able to provision practically an unlimited number of resources to a workflow application based on a pay-as-you-go pricing model, enabling scientific workflows to solve larger scientific problems and to address the big data challenges (volume, velocity, and variety) more efficiently. This leads to the notion of big data scientific workflows [30, 33], or *big data workflows* for short.

However, cloud's hardware resources are usually shared among different users or organizations through isolation techniques such as virtual machines or containers. The resource sharing characteristics and the large size of the cloud's system software makes the cloud vulnerable to various attacks [2, 14, 34, 45, 50, 57]. For example, Ristenpart *et al.* [49] showed that an outside adversary could extract unauthorized information in AWS EC2 instances. In addition, big data 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) - the quintessential software and hardware portion of the system whose security is critical and must be ensured for the sound behavior of the rest of the system. The vulnerabilities introduced by a large TCB may be exploited by adversaries. Attacks could also be originated from a malicious insider, such as a dishonest administrator or high-privileged malicious cloud software, where the big data analytics

platform is deployed. Hardware-assisted trusted execution environments (TEEs) is a promising solution to protect the execution of big data workflows and workflow data. Intel Software Guard eXtensions (SGX) [5, 24, 42] and AMD Secure Encrypted Virtualization (SEV) [29] are two widely used general purpose hardware-assisted TEEs developed for the x86 architecture. The pros and cons of Intel SGX and AMD SEV technologies are discussed in [43]. Intel SGX has been used for protecting big data analytics in the cloud. For example, Shuster *et al.* [9] proposed VC3, a system that leverages SGX to protect unmodified Map-Reduce tasks written in C/C++. Pires *et al.* [46] proposed a lightweight, Map-Reduce framework with Lua [23], a high-level language that interprets the Map-Reduce Lua scripts in Intel SGX. Zheng *et al.* proposed Opaque [61] that enhanced the security of the Spark SQL with SGX. Although these systems are the pioneers in using hardware-assisted security technologies for big data analytics, they are limited to specific domains. For example, the systems proposed by Shuster *et al.* [9] and Pires *et al.* [46] support only Map-Reduce-style workflows consisting of a Map task and a Reduce task, but not workflows with more complex structure [3, 33]. The Opaque application is limited to the use of relational algebra based tasks with Spark SQL. Although Opaque provides techniques for protecting the secure execution of relational algebra expressions, the proposed system does not support tasks whose source code is not available. In addition, each task in Opaque is a relational operator with relations as their inputs and outputs. Finally, existing systems do not support tasks with well-defined input and output ports of complex data types. In this paper, we propose SecDATAVIEW, a secure big data workflow management system that leverages Intel SGX and AMD SEV to provide a heterogeneous trusted execution environment (TEE) for big data workflows. SecDATAVIEW is transparent to users and application-level workflow tasks, and addresses the following three challenges.

Firstly, 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 572 thousands of lines of code [11]. This large TCB inevitably creates vulnerabilities that can be easily exploited by attackers. The National Vulnerability Database shows that there are 307 vulnerabilities in Xen and 6 vulnerabilities in the latest Linux kernel version 5.1.6 [1]. External attackers may exploit such vulnerabilities to gain access to computers on which scientific workflows execute to access or modify data and workflow tasks. To address this issue, 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 instances and by keeping the high-privileged cloud system software outside of the TCB.

Secondly, SGX is compatible with only a limited set of C/C++ libraries. However, many workflow tasks are written in Java and use third-party Java libraries, which is not directly supported by SGX. To address this issue, SecDATAVIEW uses the shielding approach proposed in [8] and the SGX-LKL library OS [15, 40] to execute workflow tasks written in Java in the secure enclaves of the worker nodes. Alternatively, SecDATAVIEW uses AMD Secure Encrypted Virtualization (SEV) to provide the protected worker nodes during the workflow runtime.

Thirdly, big data workflow tasks are often memory-intensive. For example, 75% of the execution time of the Broadband workflow [20] is consumed by workflow tasks that require over 1GB memory. Running the kernel of a workflow management system such as DATAVIEW [30] itself also requires over 500MB memory. As a result, in the SGX-protected workflows, when tasks require a large amount of secure memory, the SGX enclave page cache (EPC) memory paging could significantly increase the execution time of a workflow. SecDATAVIEW addresses this issue with the help of SEV-protected instances that provide a larger amount of secure memory than SGX enclaves. Our contributions are summarized as follows:

- We propose SecDATAVIEW, a secure 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 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-shielding approach and 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.
- To support memory-intensive workflows and reduce the overall performance overhead incurred by SGX enclaves EPC memory paging, SecDATAVIEW enables users to selectively assign confidential tasks into SGX or SEV worker nodes. Researchers in [43] 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 may run the 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 the security-sensitive confidential tasks (e.g., tasks that need enhanced-degree of security) to the SGX worker nodes.
- We have implemented SecDATAVIEW and conducted experiments on a real-world diagnosis recommendation workflow [3], Map-Reduce workflow [16], and a distributed K-means 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. The source code of the SecDATAVIEW system is available on GitHub[1] for further research and improvement.

The rest of the paper is organized as follows. Section §2 provides an overview of big data, the DATAVIEW workflow management system, the Intel SGX and AMD SEV technologies, and adversary

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

model. Section §3 presents the architecture design and communication protocol implementation of SecDATAVIEW. Section §4 presents our experimental results. Related work is given in Section §5 and Section §6 concludes the paper.

2 BACKGROUND AND ADVERSARY MODEL

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 inter-dependencies, to process and analyze data of ever increasing in scale, complexity, and rate of acquisition [30, 33]. 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 [30, 33]. An example of workflow is shown in Figure 6, which is a well-known word count (Map-Reduce) workflow.

SecDATAVIEW was developed based on the DATAVIEW scientific workflow management system [30]. The architecture of DATAVIEW is given in Figure 1. The reasons we chose DATAVIEW as our BDWFMS are as follows: DATAVIEW represents the state-of-the-art big data workflow management system, and it has a strong user base – over 700 registered worldwide. DATAVIEW has been used in various data analytics applications, including diagnosis recommendation [3], predicting the efficacy of therapeutic services for autism spectrum disorder [10], analysis of vehicle data to assess driver's driving behavior [31], medical image processing [22], biological simulation data analysis [19], and brain fiber connectivity analysis [36]. DATAVIEW consists of three layers: the *Presentation & Visualization Layer*, the *Workflow Management Layer*, and the *Task Management Layer*. The *Presentation & Visualization* module is responsible for the presentation of workflows and the visualization of various data products and provenance metadata. The *Workflow design & configuration* module provides intuitive GUI for users to design and configure workflows. The *Workflow Engine* is a central module that orchestrates 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 storing, browsing, and querying workflow provenance. The *Task Management* module enables the execution of heterogeneous atomic tasks, including web services and scripts, and tasks that are executed on VMs in the cloud. The *Cloud Resource Management* module interacts with virtual resources in clouds and data centers. Using DATAVIEW, a user can not only easily share data and workflows with peer collaborators, but also design and run big data scientific workflows in the cloud, including commercial Amazon EC2 and academic clouds.[2]

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 [6]. SGX also protects against the physical memory access class of attacks [8]. With SGX, TCB contains only the processor and the code running inside the enclave. SGX reserves a limited size

[2]https://portal.futuresystems.org/

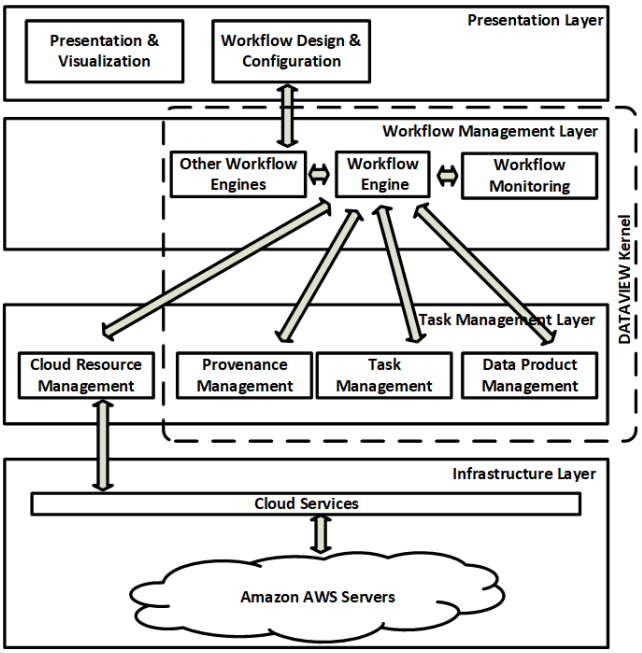


Figure 1: Architecture of the *base DATAVIEW* [33].

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 [6, 13]. Although a larger memory size can be supported through the paging mechanism, it incurs up to $1,000x$ performance overhead [6]. To speed up the execution performance of parallel applications, SGX supports multi-threads inside of 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) [29] technology and provides the protection against attacks that usually happen in cloud system software such as high-privileged hypervisor 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 [4, 28, 29]. 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 [9]. The attackers may control the whole software stack in remote servers, including their system software. The attackers may also have access to network packets and capture, replay, and modify them. An adversary may also access or change data after it left the processor with hardware-tapping or probing techniques. Attacker can access any process running on a worker node. The adversary could 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 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 that resides in the remote location. Other attacks, including network traffic-analysis [48], denial-of-service, access pattern leakage [17], side-channel attacks [56], and fault injections [7], are out of the scope in this paper.

3 DESIGN AND IMPLEMENTATION

We identify the following security-related requirements for SecDATAVIEW:

- **R1:** Providing the confidentiality and integrity of code and data for workflows running on public untrusted clouds.
- **R2:** Minimizing the TCB size for SecDATAVIEW.
- **R3:** Enabling the trade-off between security and performance for workflows with different user requirements.
- **R4:** Supporting the execution of Java-based workflow tasks in SGX nodes without tedious code refactoring.

Figure 2 gives the architecture of SecDATAVIEW, which uses a heterogeneous computing environment including both SGX and SEV worker nodes. This environment provides the flexibility of trade-off between performance and the degree of security (Requirement R3). Based on the previous study [43], 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., trusting the entire VM) and no support for memory integrity protection, which decreases its degree of security assurance. SecDATAVIEW benefits greatly from our proposed 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.

3.1 SecDATAVIEW Architecture

To protect data and prevent the execution of DATAVIEW from being altered by attackers, one approach is to execute the whole DATAVIEW system inside an SGX enclave or SEV-protected VM. While this approach may work for SEV-protected VM, it is not practical for SGX due to the following reasons. First, the code and data that reside in a protected physical memory region inside an enclave are called the *enclave page cache (EPC)*. The size of the EPC is up to $128MB$. To support applications that use a large amount of memory, SGX provides a memory paging mechanism to swap memory pages between EPC and the memory outside the enclave; memory pages swapped out are encrypted. Enclave memory paging is expensive and imposes performance overhead. For example, experimental results in [6] show that, when the accessed memory is beyond the size of the EPC, the triggered page faults may impose an overhead of $1,000x$. Scientific workflow tasks are often memory-intensive. For example, 75% of the execution time of the Broadband workflow [20] is consumed by workflow tasks that require over $1GB$ memory. Running the DATAVIEW kernel itself also requires over $500MB$ memory. As a result, memory paging could significantly increase the execution time of the DATAVIEW server and workflow tasks, which may reduce the user's willingness to use the proposed trusted execution environment. Secondly, SGX is compatible with only a limited set of C/C++ libraries. Many workflow tasks in DATAVIEW are written in Java that use third-party Java libraries, which are not directly supported by SGX. Although SGX-LKL library OS supports the execution of Java code, it supports only the execution of a single process inside the enclave (system call *fork* is not supported). Moreover, putting all DATAVIEW modules inside the enclave or SEV-protected VM increases the size of TCB, which in turn decreases the security of the system.

To address the above challenges, we identify the components in DATAVIEW that process confidential data and execute only such components inside SGX enclaves; other components are executed on the trusted premises such as private cloud computing platforms or the user side premise as usual (Requirement R2). As different components in DATAVIEW interact with each other, we develop the WCPAC (Workflow Code Provisioning And Communication) protocol to provision and attest secure worker nodes, securely provision 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 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 workflow task.

In DATAVIEW [32], the *Workflow Engine* and the *Task Management* modules are security-sensitive components as they interact with workflow tasks that may process confidential data. DATAVIEW was not designed with security in mind and all communications between two different modules are passed through an unencrypted channel. In addition, although the input and output data are transferred through secure FTP (sftp) channel, they were stored in plaintext format. SecDATAVIEW aims to protect the confidentiality and integrity of the workflow's code and data. To do so, we redesigned the *Cloud Resource Management* to initialize SGX/SEV worker nodes, and added two security-related modules – *Code Provisioner* and *Code Provisioning Attestation* – to the *Task Management* and *Workflow Engine* sub-systems, respectively.

Figure 2(a) visualizes 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 premise 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 instances.

The *Workflow Executor* executes on the master node in a secure premise. *Task Executors* runs on each worker nodes with TEE support. AEAD AES-256 GCM symmetric cryptography [51, 52] and SSL secure socket are used to secure the communication between the *Workflow Executor* and the *Task Executors*. When a workflow is initialized and before the workflow's code is decrypted in worker nodes, a code attestation protocol is executed.

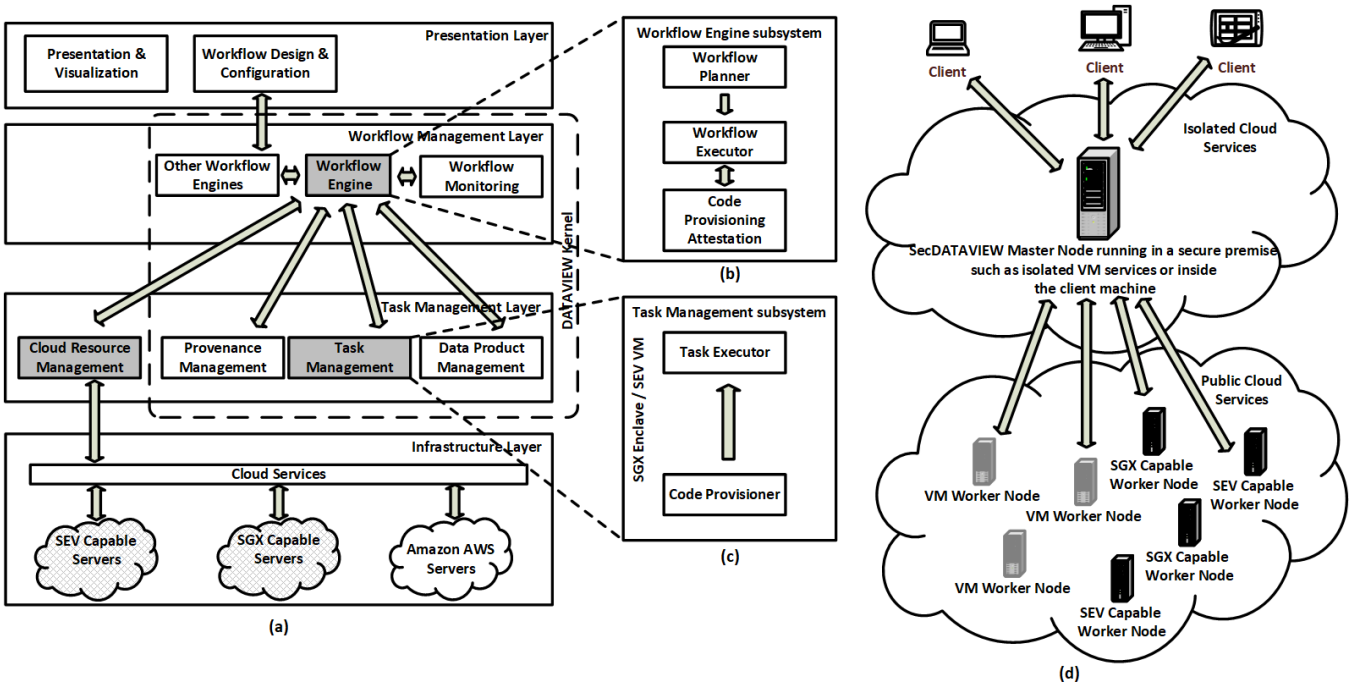


Figure 2: SecDATAVIEW Architecture. (a) visualizes the secure system architecture for SecDATAVIEW and zoom-in views of its two components: (b) Workflow Engine, and (c) Task Management. (d) provides the all-in-cloud deployment architecture of SecDATAVIEW.

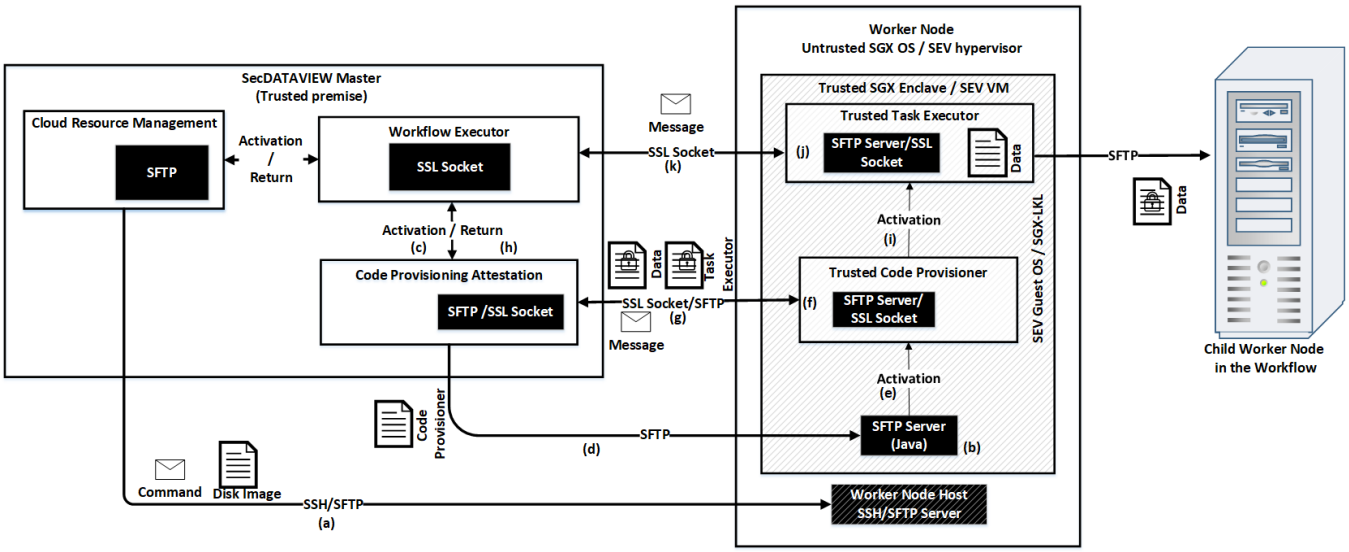


Figure 3: The WCPAC Protocol for securing the communication between *Workflow Executor, Cloud Resource Management, Code Provisioning Attestation, Code Provisioner,* and *Task Executor.*

The *Code Provisioning Attestation* module running on the master node provisions *Task Executor* with the help of *Code Provisioner* and uses a SHA256 digest message to verify the integrity of the *Code Provisioner* executed at a remote worker node. If the integrity of *Code Provisioner* is verified, then 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, and returns the control to the *Workflow Executor*. Otherwise, the *Code Provisioning Attestation* module terminates the workflow execution and informs the user about the code attestation failure.

The *Code Provisioner* module then decrypts the *Task Executor* and workflow code on the SGX/SEV worker node using the received decryption key.

The *Cloud Resource Management* module in SecDATAVIEW 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 and communicates with the worker's hypervisor using a *ssh* bash session to launch the AMD SEV-protected instance or Intel SGX-LKL enclave. After successfully initializing the worker node, it returns the control to the *Workflow Executor*.

The *Task Executor* packages all necessary code and libraries used by workflow tasks, executes workflow tasks inside the worker node's TEE, and communicates with other worker nodes. This module also interacts with the *Workflow Executor* and carries the secret key for encryption and decryption of workflow data and results. In addition, AEAD AES-256 GCM symmetric cryptography [51, 52], SSL socket, and sftp are used to protect the communication between worker nodes.

3.1.1 Executing workflows inside SGX enclaves. SGX-based applications were 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, Ocalls [26], and configuring the Enclave Definition Language [25]. The *specialized-enclave* approach has the advantage of the minimum TCB size since the amount of code in the enclave runtime is minimum. The *specialized-enclave* is suitable when the system depends on static components that are usually created by skillful developers. For example, secureKeeper [13] uses the *specialized-enclave* approach. 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 security of 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 defects that leak sensitive information from enclaves and compromise the system runtime environment.

Another approach is the *SGX Shield* approach that 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 operation system (LibOS) entirety is executed inside the enclave. On one hand, *SGX Shield* introduces a larger TCB as it puts more code inside the enclave and may significantly decrease the memory access performance of the enclave [13] when the enclave memory size exceeds $96MB$ due to memory paging overhead. On the other hand, *SGX Shield* substantially increases the usability of the SGX-based system by removing the requirement of SGX-expert knowledge and making it possible to execute unmodified applications in enclaves. In addition, *SGX Shield* 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. HAVEN [8], Graphene-SGX [55], SCONE [6], and SGX-LKL library OS [39] 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 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. The Java sftp server is started as the only active process inside the SGX-LKL enclave. The sftp server is capable to dynamically activate the *Code Provisioner* module upon its arrival inside the enclave. The sftp server leverages Java multi-threading, class loader, and reflection, and dynamically extends its active process to complex *Code Provisioner* at enclave runtime. 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 VM. AMD SEV is designed for cloud applications and protects unmodified applications by shielding the SEV VM instances from other parts of the system [29]. Unlike SGX, SEV does not provide memory integrity protection but imposes lower performance overhead than SGX. To minimize the performance overhead, SecDATAVIEW executes memory-intensive workflow tasks that do not require enhanced-degree of protection inside SEV-protected VMs. The end-user can decide what task in a confidential workflow is assigned to the SEV or SGX worker nodes. SecDATAVIEW contains a pre-created SEV disk image. This SEV disk image is used during runtime to provision a custom 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) to provision and attest secure worker nodes, (2) to securely provision the code for the *Task Executor* and workflow tasks on each participating worker node, (3) to establish the secure communication and file transfers between the master node and the worker nodes, and (4) to establish secure file transfers among worker nodes.

Every SGX worker node is configured to execute the SGX-LKL library. AMD servers are used to execute SEV instances. During the worker's launch process, an SGX-LKL-based Intel remote attestation similar to [38] is used by the *Cloud Resource Management* subsystem to verify the trustworthiness of Intel SGX CPU and SGX-LKL enclave and to send the application configuration (i.e., disk encryption key) remotely. Besides, AMD guest attestation [29] should be used to launch and verify the SEV instances. Note that at

the time we wrote this paper, the Intel remote attestation feature was not fully integrated into the release of SGX-LKL [54], and we leave the implementation of remote attestation for future work. Nevertheless, the WCPAC protocol assumes that such a protocol is incorporated, and the SGX and SEV worker nodes would use and pass the remote attestation upon the request of the *Cloud Resource Management* subsystem. Besides, the WCPAC protocol assumes that the approaches used by TEE hardware vendors (i.e., SGX-LKL and AMD SEV) to launch 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. 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 the Appendix 7. Firstly, the *Workflow Executor* activates the *Cloud Resource Management* module with a request specifying the machine type (i.e. SGX/SEV) to initialize the worker nodes – **Step (a)** in Figure 3. When a worker node is an SGX node, then the *Cloud Resource Management* module sends the SGX-LKL disk image to the worker node and activates SGX-LKL over ssh, which runs the stand-alone sftp server inside an SGX-LKL enclave, with the Intel remote attestation protocol. When a worker node is a SEV, then the *Cloud Resource Management* module sends the SEV disk image to the worker node, activates the SEV over ssh, and runs the stand-alone sftp server inside the SEV-protected VM with the AMD guest attestation protocol. Upon successful initialization, all worker nodes have active sftp server – **Step (b)** in Figure 3. At this step, the *Cloud Resource Management* module returns the control 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 (c)** in Figure 3. Besides, 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*, as well as the encrypted *Task Executor* to the SGX enclave or SEV instance through sftp – **Step (d)** 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 (e) and (f)** 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. With this technique we enforce another check on the *Code Provisioner* to capture any possible network or other flaws during code transferring event. If the digest does not match, the job is terminated; otherwise, the *Code Provisioning Attestation* module sends the *Task Executor*'s decryption key 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 through sftp. After the success of attestation and file transfer, the control is returned to the *Workflow Executor* from the *Code Provisioning Attestation* – **Steps (g) and (h)** in Figure 3.

Table 1: Testbeds Configuration.

Testbed Machine	SecDATAVIEW Master	Intel SGX	AMD SEV
CPU Model	Intel Core i7-6700T	Intel Xeon E3-1275 v5	EPYC 7251
CPU Core	4	4	8
CPU Thread	8	8	16
CPU Base Clock	2.8GHz	3.6GHz	2.1GHz
CPU Boost Clock	3.6GHz	4GHz	2.9GHz
Cache Type	Smart Cache	Smart Cache	L3
Cache Size	8MB	8MB	32MB
Motherboard	Dell Inspiron 24-5459	Intel FOG	GIGABYTE MZ31-AR0
Memory	12GB DDR4 Non-ECC	32GB DDR4 Non-ECC	32GB ECC
Storage	Conventional HDD	NVME SSD	SATA SSD
Hypervisor/OS	Ubuntu 16.04 LTS	Ubuntu 16.04 LTS	Ubuntu 18.04 LTS
Kernel Version	4.15.0-50-generic-x64	4.15.0-50-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	4.18.20-generic-x64
SEV VM Memory	N/A	N/A	4GB (Encrypted)
SEV VM Storage	N/A	N/A	32GB (Disk Image)

Upon receiving the *Task Executor*'s decryption key 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* terminates and the control is transferred to the *Task Executor* – **Step (i)** 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 threads. At this moment, the communication between *Workflow Executor* and *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 output 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 (j) and (k)** 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.

4 EVALUATION

This section presents the evaluation results of SecDATAVIEW. Specifically, we aim to answer three research questions:

Sect. 4.1: What is the performance overhead of running workflows inside SecDATAVIEW?

Sect. 4.2: Does SecDATAVIEW preserve its security properties?
Sect. 4.3: 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.

- The SecDATAVIEW master node has Intel Core i7-6700T 3.6GHz CPU with 4 physical cores and 8 logical threads, 8MB of smart cache, 12GB of DDR4 non-ECC RAM, and a conventional HDD storage, running Ubuntu 16.04 LTS with the kernel version 4.15.0-50-generic-x64.
- The SGX machines have Intel Xeon E3-1275 v5 4GHz CPU with 4 physical cores and 8 logical threads, 8MB smart cache, 32GB DDR4 non-ECC RAM, and an NVME SSD storage. Ubuntu 16.04 LTS with kernel version 4.15.0-50-generic x64 and SGX SDK version 2.0 were installed on SGX machines.
- The AMD machine has an AMD EPYC 7251 2.9GHz CPU with 8 physical cores and 16 logical threads, 32MB L3 cache, 32GB of DDR4 RAM, and 512GB SATA SSD, running Ubuntu 18.04 LTS with kernel 4.20.0-sev. Two SEV-protected VMs were used in our experiments. Each VM was assigned 4GB of memory, 4 CPU cores, and 32GB storage. Ubuntu 18.04 LTS with 4.18.20-generic-x64 kernel was installed in each SEV-protected VM.
- We have also installed Java version 1.8 on all machines and compiled the latest SGX-LKL software in hardware mode on SGX machines. All machines are connected with a 100Mb LAN interface thus making a heterogeneous cluster of five nodes.

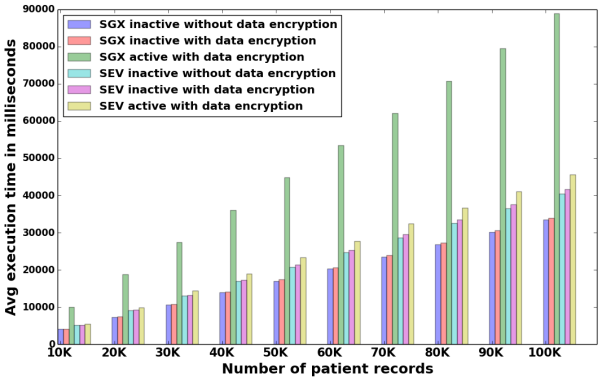
4.1 Workflow Performance Evaluation

We used three different types of workflows in our experiments: the Diagnosis Recommendation workflow [3], the Word Count workflow, a.k.a. the Map-Reduce workflow [16], and the Distributed K-means workflow [27]. We measured the performance overhead incurred by SGX/SEV in terms of the execution time and the memory usage for each workflow in eight different configurations:

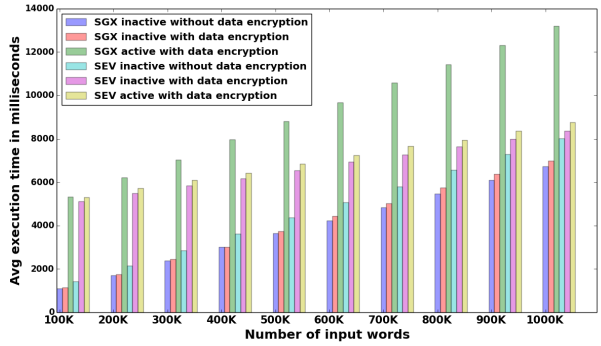
- **SGX inactive without data encryption**: in this configuration, all workflow tasks are running outside of enclaves on SGX worker nodes. Task code is encrypted and data is not encrypted.
- **SGX inactive with data encryption**: in this configuration, all workflow tasks are running outside of enclaves on SGX worker nodes. Task code and data are encrypted during network transfer and then decrypted before their usage.
- **SGX active with data encryption**: in this configuration, all workflow tasks are running inside of enclaves on SGX worker nodes. Both task code and data are encrypted during network transfer and then decrypted before their usage.
- **SEV inactive without data encryption**: in this configuration, all workflow tasks are running on SEV worker nodes but the SEV feature is not used. Task code is encrypted and data is not encrypted.
- **SEV inactive with data encryption**: in this configuration, all workflow tasks are running on SEV worker nodes but the SEV feature is not used. Both task code and data are encrypted during network transfer and then decrypted before their usage.
- **SEV active with data encryption**: in this configuration, all workflow tasks are running on SEV worker nodes with the SEV feature used. Both task code and data are encrypted during network transfer and then decrypted before their usage.
- **Hybrid inactive with data encryption**: in this configuration, workflow tasks are running on a mixed setting of SGX and SEV worker nodes without using the features of SGX and SEV. Both task code and data are encrypted during network transfer and then decrypted before their usage.
- **Hybrid active with data encryption**: in this configuration, workflow tasks are running on a mixed setting of SGX and SEV worker nodes with the use of the features of SGX and SEV. Both task code and data are encrypted during network transfer and then decrypted before their usage.

All experiments were conducted with 2GB SGX heap and 4GB SEV RAM and 1GB JVM heap memory space. Our experiments target an extreme scenario in which all workflow tasks are scheduled on the same SGX/SEV worker node. These experiments are used to measure the maximum possible overhead of SecDATAVIEW. In our experiments, the memory footprint and heap size are larger than the SGX EPC memory and communication between two workflow tasks are encrypted. To measure the execution time, we have started a timer in the *Workflow Executor* module that is initiated before the activation of *Task Executor* and ends when the workflow finishes execution. The reported results represent the aggregated performance and overhead incurred by SGX-LKL/SEV instance runtime, secure network stack, read/write access inside the SGX-LKL and SEV disk image, file encryption/decryption, file transfer between worker nodes, and secure execution of workflow tasks.

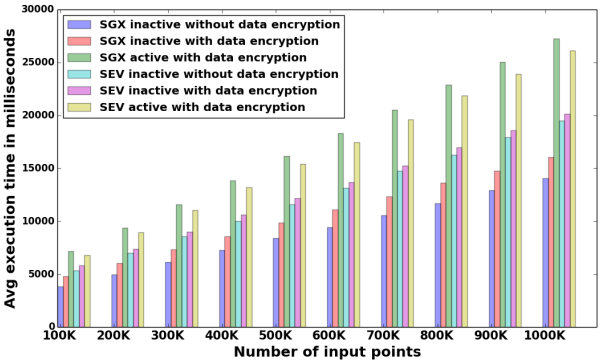
4.1.1 The Diagnosis Recommendation workflow. The diagnosis recommendation workflow is a real-life diagnosis workflow [3] that uses machine learning methods and raw textual dataset as a prescription for a group of patients. We synthetically created $10,000 - 100,000$ patient records with an average length of 150 characters for an individual prescription. In our first experiment, all workflow tasks are associated with SGX worker nodes with three different settings: (a) *SGX inactive without data encryption*, (b) *SGX inactive with data encryption*, and (c) *SGX active with data encryption*. In our second experiment, all workflow tasks are associated with SEV nodes with three settings: (a) *SEV inactive without data encryption*, (b) *SEV inactive with data encryption*, (c) *SEV active with data encryption*. Figure 4(a) shows that, when the size of the patient records increases, the execution time also increases. This is because when the size of the patient records increases, the time spent in encrypting and decrypting the records increases and it takes a longer time for the training and testing machine learning models in the workflow to process larger records. Our experimental results show that SGX and SEV impose $2.62x$ and $1.29x$ overhead on the largest dataset containing $100,000$ patient records, respectively. It is also depicted that *SGX active with data encryption* mode has higher execution time than the *SEV active with data encryption*. However, while experimenting outside of SGX and SEV TEEs, the execution time of *SGX inactive with data encryption* is lower than *SEV inactive with data encryption*. In the hybrid approach that uses two SGX nodes and two SEV nodes for the largest of datasets, the overhead is $1.20x$ as it is depicted in Figure 5. Since Algorithm1 and Algorithm2 from Diagnosis Recommendation workflow [3] are resource-intensive tasks and are not highly security-sensitive, we assigned them to



(a) Diagnosis Recommendation Workflow.



(b) Word Count Workflow.



(c) Distributed K-means Workflow.

Figure 4: Execution time of running different workflows in different configurations for different input datasets.

two SEV instances whereas the rest of the tasks are assigned to two SGX enclaves. We have also experimented to obtain the memory usage for the largest data sets. It is depicted that the SEV instances are faster than SGX enclaves. Our experimental results also show that $1GB$ heap memory and $33MB$ non-heap memory is consumed when executing the diagnosis recommendation workflow, as shown in Table 2. Table 3 depicts the encryption and decryption overhead in hybrid active settings is $1.04x$.

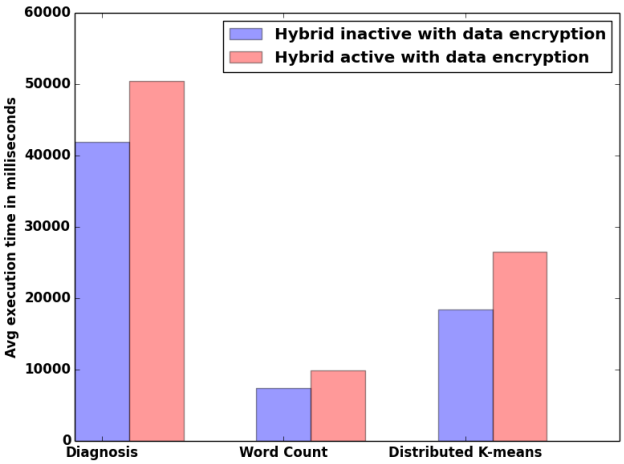


Figure 5: Execution time of running Diagnosis Recommendation, Word Count and Distributed K-means workflows in Hybrid inactive/active with data encryption mode for the largest datasets.

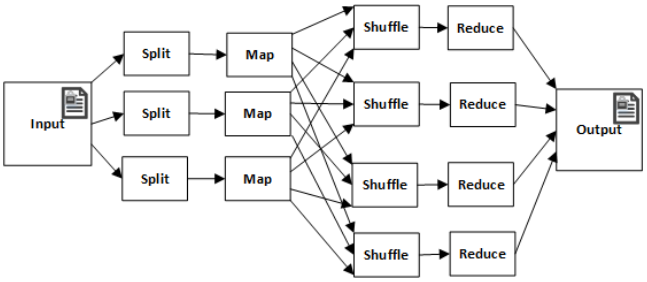


Figure 6: The word count (Map-Reduce) workflow.

4.1.2 The Word Count workflow. Figure 6 depicts a Map-Reduce workflow, a well-known word-count example for the Map-Reduce [16] operation to measure the execution times and memory overhead of SecDATAVIEW. To conduct this experiment, we created a workflow with 16 tasks including one task for input processing, 3 Splitting and 3 Mapping tasks for map operation, 4 Shuffling and 4 Reducing tasks for reduce operation, and one task for the final output organization. In our experiment, we randomly generated $100,000 - 1,000,000$ words containing 2 characters as input. Figure 4(b) compares the execution time of the word count workflow with and without SGX/SEV. Each data point in the figure is an average of 5 workflow iterations. The figure shows that when the number of words increases from $100,000$ to $1,000,000$, the execution time increases linearly. SGX and SEV impose $1.89x$ and $1.04x$ overhead for the largest dataset (i.e., the dataset containing $1,000,000$ words), respectively. This shows that SEV imposes little overhead on the execution time and SGX imposes higher overhead than SEV. In our hybrid approach in which workflow tasks are executed inside both SGX (2 nodes) and SEV (2 nodes) machines at the same time by randomly distributing the tasks to 4 SGX/SEV nodes, the performance overhead is $1.33x$ as depcited in Figure 5.

Table 2: Memory usage of experimental workflows.

Workflow	Max heap	Max non-heap	Total
Diagnosis Recommendation	$1GB$	$33MB$	$1,057MB$
Word Count	$1GB$	$31MB$	$1,055MB$
Distributed K-means	$1GB$	$30MB$	$1,054MB$

Table 3: Encryption decryption performance overhead of Hybrid active.

Workflow	Including	Excluding	Overhead
Diagnosis Recommendation	$50,481ms$	$48,132ms$	$1.04X$
Word Count	$99,59ms$	$96,11ms$	$1.03X$
Distributed K-means	$26,551ms$	$24,591ms$	$1.07X$

We have also measured the memory usage of SecDATAVIEW. Our experimental results show that SecDATAVIEW uses $1GB$ heap and $31MB$ non-heap memory, as depicted in Table 2. Table 3 shows the encryption and decryption imposes $1.03x$ overhead in hybrid active settings.

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 $100,000$ to $1,000,000$ points, each of which has an x and a y coordinate. Figures 4(c) and 5 give the execution time of SecDATAVIEW, which shows that SGX, SEV, and our hybrid approach impose $1.69x$, $1.29x$ and $1.43x$ overhead on the largest dataset ($1,000,000$ points), respectively. Also, running the distributed K-means workflow uses $1GB$ heap and $30MB$ non-heap memory, which is represented in Table 2.

4.2 Security Analysis

SecDATAVIEW architecture and TCB: The SecDATAVIEW architecture provides the smallest software and hardware TCB for deploying big data workflow management system in the cloud. For SGX nodes, the software component of TCB is the LibOS, the JVM, the Code Provisioner, and the Task Executor. For SEV worker nodes, the software component of TCB is the guest OS, the JVM, the Code Provisioner, and the Task Executor. The hardware component of the TCB is the CPU package for the SGX workers and is AMD SoC and AMD secure processor for the SEV worker nodes. 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 languages like Java and JVM.

Workflow code and data confidentiality and integrity: 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 with the help of hardware attestation method that is provided by TEE hardware vendors (i.e., Intel Attestation Server and AMD Guest Attestation). Besides TEEs, SecDATAVIEW security protection is guaranteed with different security primitives such as AEAD scheme, one-way hash functions, SSL/TLS, and SFTP network connectivity. 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 [18], implying that without the secret key, and given the ciphertexts for any elected plaintexts and associated data, it is computationally infeasible to form another pair of ciphertext and associated data to decrypt the ciphertext.

WCPAC protocol: SecDATAVIEW uses the WCPAC protocol to 1) provision and attest worker nodes, 2) provision the code for the *Task Executor* and workflows 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 nodes. WCPAC is protected against eavesdropping, the man-in-the-middle attack, and the replay attack.

Attacks against network channel: Assume an adversary actively eavesdrop on the communication among different nodes. The adversary can learn the source, the destination, the number of transmitting packets, the time of message 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/TLS protection. Even if the adversary breaks the SSL/TLS 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 attack.

The denial of service (DoS) attack: SecDATAVIEW is vulnerable to the DoS attack, but this attack also presents in SGX and SEV TEE. For SGX, it is mainly caused by a malicious host that refuses to launch the enclave. 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 due to the lack of the SEV memory integrity protection, causing the VM to crash or exhibit unexpected behavior.

The side-channel attack: SecDATAVIEW is vulnerable to the side-channel attack that is present in every SGX [12, 21, 35, 44, 53, 56, 58, 59] and SEV [43] TEE.

4.3 Comparison with Existing Systems

Table 4 compares SecDATAVIEW against several representative big data systems including VC3 [9], Opaque [61], and the lightweight Lua Map-Reduce system [46].

Functionality: SecDATAVIEW has two main advantages compared to the existing systems: 1) it is compatible with many forms of data structures/formats and is also capable of executing a variety type of 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 and SQL query workflows, respectively. Besides, they only support SGX TEE to protect their

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

Table 4: Comparisons with existing TEE-based big data systems.

Feature	SecDATAVIEW	VC3 [9]	Opaque [61]	Lua Map/Reduce [46]
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 compability	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	1.2X-1.43X (hybrid mode)	1.04X-1.08X (base-encrypted mode)	0.52X-3.3X (encrypted mode)	1.3X-2X (encrypted mode)

computation.

Security: SecDATAVIEW and the lightweight Lua Map-Reduce use the managed code (Java/Lua) that are 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 up 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 in Opaque.

Performance: SecDATAVIEW imposes moderate overhead, a range between $1.20x$-$1.43x$ in a hybrid approach. Among compared systems, VC3 is fastest when it works with its fastest mode and without enclave memory region checking. However, when VC3 activates the enclave memory region checking, its performance is competitive with SecDATAVIEW (i.e., VC3 imposes up to $1.27x$ overhead and SecDATAVIEW imposes up to $1.43x$ overhead). Additionally, SecDATAVIEW ($1.43x$) outperforms Opaque ($3.3x$ overhead) and the lightweight Lua Map-Reduce ($2x$ overhead).

5 RELATED WORK

In this section, we survey the state-of-the-art solutions regarding big data security. Brenner *et al.* proposed Securekeeper [13] 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++ that helped it to maintain a small size of TCB. SecureKeeper imposes 32.18% overhead on the base ZooKeeper. Schuster *et al.* proposed VC3 [9] 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 (No multi-threading applied). Additionally, all data traffic of intermediate Map-Reduce results is kept encrypted during the job execution. VC3 results show a performance overhead between 4.3% to 24.5% when the enclave self-integrity checking mode is used. Pires *et al.* [46] proposed a lightweight, secure Map-Reduce framework that leveraged Intel SGX. Their system is integrated with a lightweight virtual machine for Lua language [23], which is a high-level language that interprets the Map-Reduce Lua scripts, and a Secure Content Based Routing System (SCBR) [47], which is a secure publish/subscribe system for the message passing and data distribution in the distributed system between the client and worker nodes. 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 map and reduce Lua scripts execution happens inside the secure enclave. The reported results showed up to $2x$ performance overhead. Zheng *et al.* [61] proposed Opaque that enhanced the security of the Spark SQL with SGX. One of the execution modes referred to as the encryption mode provides the confidentiality protection on the data and the results. In this mode, the Opaque's code at the client side is transferred to the enclave and with the help of 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 on the execution time. 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 $46x$ overhead on the execution time.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we present SecDATAVIEW, an efficient and secure big data workflow management system that protects the confidentiality and integrity of Java-written tasks and data in the workflow with the help of SGX/SEV worker nodes. SecDATAVIEW significantly reduces the TCB size of the worker node, and protects the *Task Executor* and individual workflow tasks by executing them inside the SGX enclave or the SEV-protected instance. Our experiments with different types of workflows show the usability of the system with a low-performance overhead while securing the confidential task execution at SGX enclave/SEV instance runtime. We plan to investigate the security issues of collaborative scientific workflows [41, 60] in the future, in which multiple users design and execute a workflow in the cloud collaboratively.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful comments that helped improve this paper. This work is supported by National Science Foundation under grant NSF OAC-1738929.

REFERENCES

[1] [n.d.]. National institute of standards, national vulnerability database. https://nvd.nist.gov/.

[2] Secunia Advisory. 2013. Xen pv kernel decompression multiple vulnerabilities.
[3] Ishtiaq Ahmed, Shiyong Lu, Changxin Bai, and Fahima Amin Bhuyan. 2018. Diagnosis Recommendation using Machine Learning Scientific Workflows. In *Big Data Congress, 2018 IEEE International Conference on*. IEEE.
[4] AMD. 2018. Secure Encrypted Virtualization API Version 0.16. https://support.amd.com/en-us/search/tech-docs.
[5] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13.
[6] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, and Mark L Stillwell. 2016. SCONE: Secure Linux Containers with Intel SGX. In *OSDI*, Vol. 16. 689–703.
[7] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. 2012. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proc. IEEE* 100, 11 (2012), 3056–3076.
[8] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)* 33, 3 (2015), 8.
[9] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *IEEE Symposium on Security and Privacy (SP), 2015*. IEEE, 38–54.
[10] Fahima Bhuyan, Shiyong Lu, Ishtiaq Ahmed, and Jia Zhang. 2017. Predicting efficacy of therapeutic services for autism spectrum disorder using scientific workflows. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 3847–3856.
[11] inc Black Duck Software. [n.d.]. Black Duck Open Hub. https://www.openhub.net/p?query=xen&sort=relevance.
[12] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. *arXiv preprint arXiv:1702.07521* (2017), 33.
[13] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. 2016. SecureKeeper: Confidential ZooKeeper using Intel SGX. In *Middleware*. 14.
[14] Sven Bugiel, Stefan Nürnberger, Thomas Pöppelmann, Ahmad-Reza Sadeghi, and Thomas Schneider. 2011. AmazonIA: when elasticity snaps back. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 389–400.
[15] Jon Crowcroft. 2018. Description of SGX-LKL by Peter Pietzuch - Imperial College London. https://www.cl.cam.ac.uk/~jac22/talks/ox-strachey-6.3.2018.pptx.
[16] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
[17] Tien Tuan Anh Dinh, Prateek Saxena, Ee-Chien Chang, Beng Chin Ooi, and Chunwang Zhang. 2015. M2R: Enabling Stronger Privacy in MapReduce Computation.. In *USENIX Security Symposium*. 447–462.
[18] Paul DâĂŹAvilar, Jeremy DâĂŹErrico, Ken Berends, and Michael Peck. 2004. Reading Guide 3: Authenticated Encryption. (2004).
[19] Xubo Fei and Shiyong Lu. 2010. A dataflow-based scientific workflow composition framework. *IEEE Transactions on Services Computing* 5, 1 (2010), 45–58.
[20] Robert W Graves and Arben Pitarka. 2010. Broadband ground-motion simulation using a hybrid approach. *Bulletin of the Seismological Society of America* 100, 5A (2010), 2095–2123.
[21] Marcus Hähnel, Weidong Cui, and Marcus Peinado. 2017. High-resolution Side Channels for Untrusted Operating Systems. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '17)*. USENIX Association, Berkeley, CA, USA, 299–312. http://dl.acm.org/citation.cfm?id=3154690.3154719
[22] Hajar Hamidian, Shiyong Lu, Satyendra Rana, Farshad Fotouhi, and Hamid Soltanian-Zadeh. 2014. Adapting Medical Image Processing Tasks to a Scalable Scientific Workflow System. In *2014 IEEE World Congress on Services*. IEEE, 385–392.
[23] Ashwin Hirschi. 2007. Traveling light, the Lua way. *IEEE software* 24, 5 (2007).
[24] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions.. In *HASP@ ISCA*. 11.
[25] Intel. 2018. Intel Software Guard Extensions SDK (EDL). https://software.intel.com/en-us/sgx-sdk-dev-reference.
[26] Intel. 2019. Intel Software Guard Extensions SDK (ECALL-OCALL Functions). https://software.intel.com/en-us/node/702973.
[27] Geetha Jagannathan and Rebecca N Wright. 2005. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 593–599.
[28] David Kaplan. 2016. AMD x86 Memory Encryption Technologies. USENIX Association, Austin, TX.
[29] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper, Apr* (2016).
[30] Kashlev et al. 2014. A system architecture for running big data workflows in the cloud. In *Proc. of the 2014 IEEE International Conference on Services Computing (SCC)*. IEEE, 51–58.
[31] Andrey Kashlev and Shiyong Lu. 2014. A system architecture for running big data workflows in the cloud. In *Services Computing (SCC), 2014 IEEE International Conference on*. IEEE, 51–58.
[32] Andrey Kashlev and Shiyong Lu. 2014. A system architecture for running big data workflows in the cloud. In *Services Computing (SCC), 2014 IEEE International Conference on*. IEEE, 51–58.
[33] Andrey Kashlev, Shiyong Lu, and Aravind Mohan. 2017. Big Data Workflows: a Reference Architecture and the DATAVIEW System. *Services Transactions on Big Data (STBD)* 4, 1 (2017), 1–19.
[34] Kostya Kortchinsky. 2009. Cloudburst: A VMware guest to host escape story. *Black Hat USA* (2009), 19.
[35] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium, USENIX Security*. 16–18.
[36] Cui Lin, Shiyong Lu, Xubo Fei, Artem Chebotko, Darshan Pai, Zhaoqiang Lai, Farshad Fotouhi, and Jing Hua. 2009. A reference architecture for scientific workflow management systems and the VIEW SOA solution. *IEEE Transactions on Services Computing* 2, 1 (2009), 79–92.
[37] Xiao Liu, Dong Yuan, Gaofeng Zhang, Wenhao Li, Dahai Cao, Qiang He, Jinjun Chen, and Yun Yang. 2011. *The design of cloud workflow systems*. Springer Science & Business Media.
[38] LSDS. 2018. SGX-LKL,Remote Attestation. https://github.com/lsds/sgx-lkl/wiki/Remote-Attestation-and-Remote-Control.
[39] LSDS. 2019. The Allan Turing Institute SGX-LKL Library. https://www.turing.ac.uk/research/publications/sgx-lkl-library-os-running-java-applications-intel-sgx-enclaves.
[40] LSDS. 2019. LSDS SGX-LKL Library. https://github.com/lsds/sgx-lkl.
[41] Shiyong Lu and Jia Zhang. 2009. Collaborative scientific workflows. In *2009 IEEE International Conference on Web Services*. IEEE, 527–534.
[42] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution.. In *HASP@ISCA*. 10.
[43] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. 2018. 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 '18)*. ACM, New York, NY, USA, Article 9, 8 pages. https://doi.org/10.1145/3214292.3214301
[44] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. Cachezoom: How SGX amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 69–90.
[45] Diego Perez-Botero, Jakub Szefer, and Ruby B Lee. 2013. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 international workshop on Security in cloud computing*. ACM, 3–10.
[46] Rafael Pires, Daniel Gavril, Pascal Felber, Emanuel Onica, and Marcelo Pasin. 2017. A lightweight MapReduce framework for secure processing with SGX. In *Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on*. IEEE, 1100–1107.
[47] Rafael Pires, Marcelo Pasin, Pascal Felber, and Christof Fetzer. 2016. Secure content-based routing using Intel Software Guard Extensions. In *Proceedings of the 17th International Middleware Conference*. ACM, 10.
[48] Jean-François Raymond. 2001. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies*. Springer, 10–29.
[49] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 199–212.
[50] Francisco Rocha and Miguel Correia. 2011. Lucy in the sky without diamonds: Stealing confidential data in the cloud. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*. IEEE, 129–134.
[51] Phillip Rogaway. 2002. Authenticated-encryption with Associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02)*. ACM, New York, NY, USA, 98–107. https://doi.org/10.1145/586110.586125
[52] Bruce Schneier. 2007. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons.
[53] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.
[54] Cloud Research Security. 2018. SGX-LKL,SCONE,Graphene-SGX-Remote Attestation status. https://github.com/lsds/sgx-lkl/issues/13.

[55] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC)*.

[56] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2421–2434.

[57] Rafal Wojtczuk, Joanna Rutkowska, and Alexander Tereshkin. 2008. Xen 0wning trilogy. *Invisible Things Lab* (2008).

[58] Yuan Xiao, Mengyuan Li, Sanchuan Chen, and Yinqian Zhang. 2017. Stacco: Differentially analyzing side-channel traces for detecting SSL/TLS vulnerabilities in secure enclaves. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 859–874.

[59] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 640–656.

[60] Jia Zhang, Daniel Kuc, and Shiyong Lu. 2012. Confucius: A tool supporting collaborative scientific workflow composition. *IEEE Transactions on Services Computing* 7, 1 (2012), 2–17.

[61] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform.. In *NSDI*. 283–298.

7 APPENDIX

The WCPAC protocol's main functionality includes: 1) to provision and attest secure worker nodes, 2) to provision securely the code for the *Task Executor* and workflow tasks on each participating worker node, 3) to establish the secure communication and file transfers between the master node and worker nodes, and 4) to ensure secure file transfers among worker nodes. Figure 7 shows the steps of WCPAC and relations between different entities in the SecDATAVIEW. In the following, we define SecDATAVIEW naming conventions for the sequence diagram.

machineLaunchRequest(machineType, IP): This function is to call Cloud Resource Management for launching the remote worker node (SEV or SGX machines). It accepts machineType and the IP address from the workflow executor and initializes the remote machines based on the given parameters, machineTypes are categorized as "AMD" or "SGX".

send(file): This function is responsible for sending a corresponding file from source to destination. Sending file is implemented by SFTP.

message(content): This function is responsible for sending message through SSL socket from source to destination. This message is used for sending various signals. The content type is a string.

sha256(file): This function is responsible for generating the SHA256 digest of a given file.

keyGen(): This function is responsible for generating a random password.

encrypt(key, AD, file): This function AEAD encrypts the input file based on the given secret key and AD (Associated Data).

decrypt(key, AD, file): This function AEAD decrypts the provided encrypted file based on the given secret key and AD.

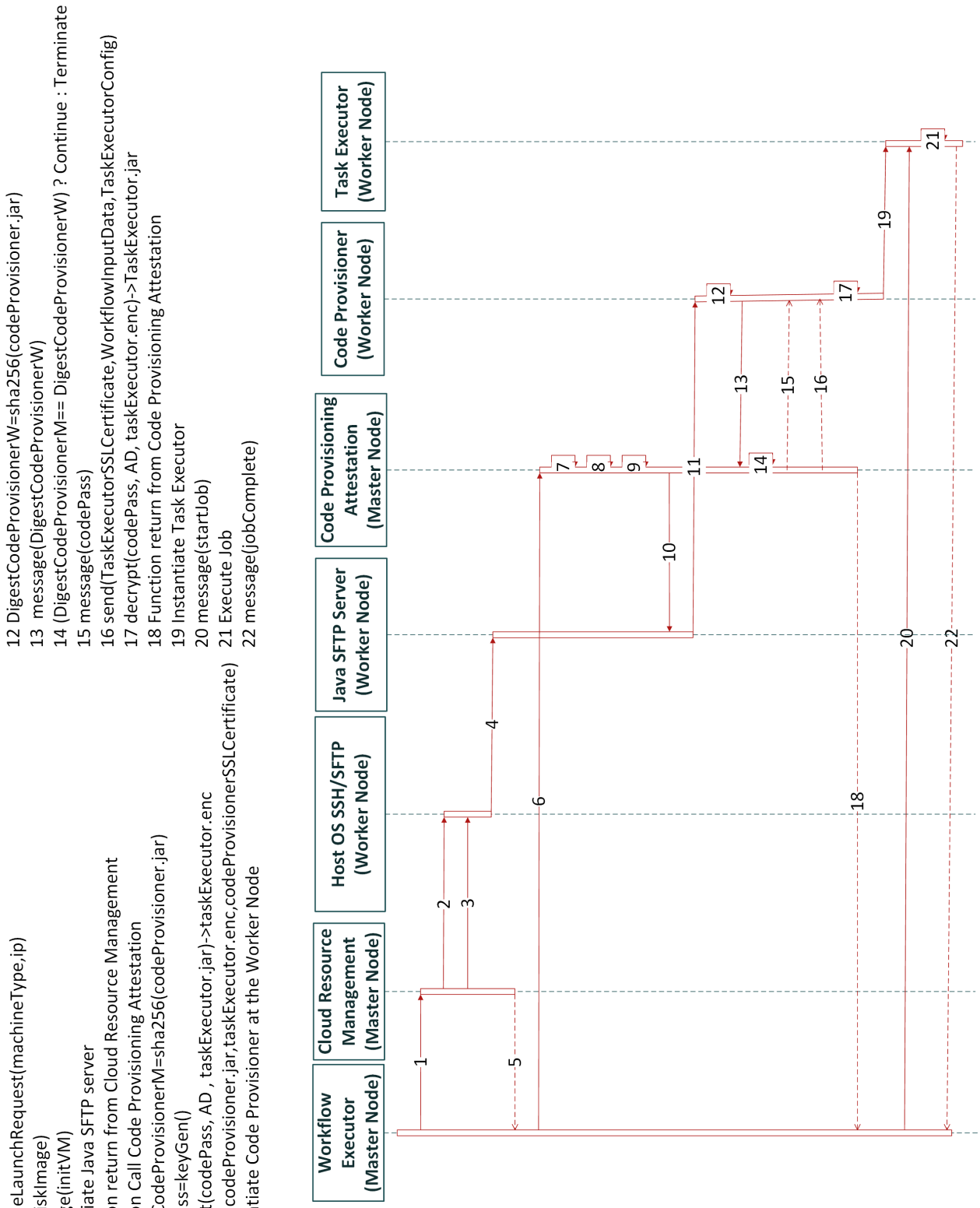


Figure 7: The Sequence Diagram of the WCPAC Protocol.