Secure Generic Remote Workflow Execution with TEEs

Lorenzo Brescia
University of Turin
Turin, Italy
lorenzo.brescia@unito.it

Marco Aldinucci
University of Turin
Turin, Italy
marco.aldinucci@unito.it

ABSTRACT

In scientific environments, the frequent need to process substantial volumes of data poses a common challenge. Individuals tasked with executing these computations frequently encounter a deficit in local computational resources, leading them to opt for the facilities of a Cloud Service Provider (CSP) for data processing. However, the data subjected to these calculations may be subject to confidentiality constraints. This paper introduces a proof-of-concept framework that leverages Gramine LibOS and Intel SGX, enabling the protection of generic remote workflow computations through SGX enclaves as Trusted Execution Environments (TEEs). The framework entails the delineation of user and CSP behavior and has been implemented using Bash scripts. Furthermore, an infrastructure has been designed for the Data Center Attestation Primitives (DCAP) remote attestation mechanism, wherein the user gains trust in the proper instantiation of the enclave within the CSP. To assess the framework efficacy, it has been tested on two distinct workflows, one trivial and the other involving real-world bioinformatics applications for processing DNA data. The performance study revealed that the framework incurred an acceptable overhead, ranging from a factor of x1.4 to x1.8 compared to unsafe execution practice.

CCS CONCEPTS

• **Security and privacy → Security in hardware**.

KEYWORDS

Trusted Execution Environment, Workflow, Intel SGX, Gramine, Privacy-Preserving, Confidential Computing

ACM Reference Format:
Lorenzo Brescia and Marco Aldinucci. 2024. Secure Generic Remote Workflow Execution with TEEs. In *Workflows in Distributed Environments (WiDE '24), April 22, 2024, Athens, Greece*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3642978.3652834

1 INTRODUCTION

The increasingly complex nature of scientific research demands the processing of massive volumes of data, a task widely facilitated by Cloud Service Provider (CSP). However, the security of sensitive data within the cloud perimeter remains a critical concern. Although encryption safeguards data during transmission and at rest, CSPs may intentionally or unintentionally alter data during computations, potentially compromising its integrity and confidentiality. To address this challenge, various secure remote computation approaches have emerged, including Fully Homomorphic Encryption (FHE) [8], differential privacy [5] and Trusted Execution Environments (TEEs). FHE presents a method where computations can be securely executed directly on encrypted data, offering a high degree of security. However, its real-world applicability is hindered by the substantial increase in execution time [15]. Another approach, differential privacy, introduces noise to data to enhance privacy preservation. Nevertheless, this method often results in less accurate models in machine learning context [2], rendering it less suitable as a generic method of ensuring data privacy. Conversely, TEEs prove effective in guaranteeing privacy across diverse contexts without imposing an overhead that compromises viability.

This paper presents a proof-of-concept framework that leverages on Intel Software Guard Extensions (SGX) TEE and Gramine Library Operating System (LibOS) to establish a secure communication channel between the user and the CSP's TEE, enabling generic remote workflow computation on confidential data without exposing it to the cloud infrastructure. The design involves specifying the behavior of:

- the user who wants to execute a workflow remotely within a cloud infrastructure.
- the CSP that performs the computation.
- the communication between the two parties

The implementation was done with Bash scripts, assuming the user could connect to the CSP through an Secure SHell (SSH) connection. The framework's efficacy was evaluated through the execution of two distinct workflows: a straightforward implementation utilizing C and Python code and a real-world application involving DNA analysis through scripts trim_galore[1] and bowtie[2]. A potential concern associated with the use of TEEs stems from the perceived overhead introduced to ensure the confidentiality and integrity of computations. However, performance evaluations have demonstrated that, under optimal conditions, these concerns are alleviated. The computational overhead observed in such scenarios ranges from x1.4 to x1.8 compared to execution on hardware lacking trusted features.

2 BACKGROUND

2.1 Intel SGX

Intel SGX represents an augmentation of the instruction set architecture of Intel CPUs, originated in 2016. This extension permits the establishment of a TEE denoted as enclave. In general, a TEE is characterized by a set of security guarantees [16] encompassing confidentiality (preventing unauthorized data access to any form

WiDE '24, April 22, 2024, Athens, Greece

ACM ISBN 979-8-4007-0546-5/24/04.
https://doi.org/10.1145/3642978.3652834

[1] https://github.com/FelixKrueger/TrimGalore
[2] https://github.com/BenLangmead/bowtie2

of software, including the operating system/hypervisor), integrity (precluding undetected alterations of code), resilience against specific classes of physical attacks and the provision of mechanisms for remote attestation. SGX enclaves embody these security features and leverage on Data Center Attestation Primitives (DCAP) as the attestation mechanism [17]. Notably, the utilization of DCAP obviates the necessity for runtime interactions with Intel servers, disassociating the execution capability from the availability and reliability of Intel public services, in contrast with Enhanced privacy ID (EPID) [9] scheme that introduces a runtime dependency with an Intel Attestation Verification Service.

The evolution of SGX technology has manifested notable improvements, such as the creation of large enclaves, with SGX v2 accommodating up to 512GB as opposed to the 128MB capacity of SGX v1 [6] and the Enclave Dynamic Memory Management (EDMM) [11]. The utilization of EDMM alleviates the necessity to predefine the size of the enclave in advance. This innovative feature empowers the enclave to dynamically manage memory, allocating it as needed while concurrently upholding security guarantees. In scenarios where this feature is not employed, the enclave's memory must be predetermined and allocated in its entirety before the start of execution, theoretically leading to slowdowns.

Lastly, despite the perceived security of the enclave, it is imperative to acknowledge the perpetual emergence of novel and insidious cyber threats, mitigated through the deployment of a multitude of corresponding solutions [7].

2.2 Gramine

Gramine [19], formerly Graphene [18], serves as a LibOS that facilitates the execution of unmodified Linux applications within SGX enclaves. The alternative SGX execution approach would involve the challenging task of recoding the application using the Intel Software Development Kit (SDK), an endeavor that is impractical for large applications. Gramine streamlines this process, requiring only a declarative compilation of a manifest containing execution parameters, such as number of threads, filesystem mounting configuration, enclave dimension and other crucial specifications.

A distinctive feature of Gramine is the capability to designate specific files for integrity verification using hash functions. Furthermore, Gramine allows to specify which files (or folders) are to be maintained confidential. After the remote attestation phase, wherein the user gains trust regarding the instantiation of the application inside the enclave, occurs a phase of secret provisioning, where the user securely transmits an arbitrary symmetric key through a secure Transport Layer Security (TLS) channel. Subsequently, this key is employed within the enclave to process confidential files.

2.3 Workflow

A scientific workflow is a description of a procedure aimed to accomplishing a specific objective, articulated in terms of tasks and their interdependencies [10]. In this paper, emphasis is placed on workflows characterized by constrained expressive capabilities. Specifically, Figure 1 shows these workflows that are detailed as sequential processes of N steps where each step generates outputs that serve as inputs for the subsequent step. The culminating outcome of the computational process is derived at the completion of the final step N. The deliberate selection of this specific category of workflow was made with the goal of concentrating on the assessment of the efficiency and the implications of trusted execution in a distributed environment.

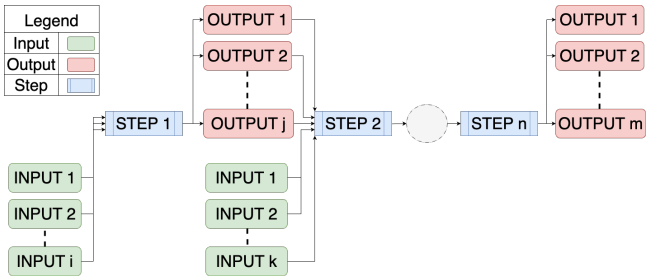

Figure 1: High-level workflow's representation

2.4 Related Works

Previous studies have been conducted to achieve trusted execution on remote machines. Hybrid Secured Flow (HySec-Flow) [20] is a framework for developing large-scale genome computing tasks using SGX. This has included the design of a task scheduler to integrate secure and non-secure containers into the workflow process. Privacy-preserving Federated Learning (PPFL) [12] utilize ARM TrustZone TEEs client-side for local training and Intel SGX TEEs server-side for secure aggregation. SecDATAVIEW [13] is a Big Data Workflow Management Systems (BDWMS) that employs Intel SGX and AMD Secure Encrypted Virtualization (SEV) TEEs [1] to protect large-scale data analytics workflows in the cloud. Nevertheless, all of these works focuses on domain-specific applications. This paper aims to establish the foundation for an investigation of generic workflow execution within an untrusted CSP.

3 METHODS

To securely execute a workflow characterized as in Section 2.3, the establishment of a comprehensive management framework is imperative. This framework must adeptly address the safeguarding of data in transit, at rest, and during execution within an untrusted cloud environment. In particular, critical aspects necessitating meticulous attention include:

- Encryption of input data requiring confidentiality and decryption of results when confidentiality constraints are absent, such as within the user's local environment.
- Provision of required information to enable the untrusted cloud to execute the workflow steps, especially to build the proper Gramine manifest.
- Implementation of remote attestation mechanisms.
- Prudent management of data movement, ensuring security throughout transitions between workflow steps.
- Definition of a standardized file system structure harmonizing between user and untrusted CSP to facilitate execution compatibility with Gramine.

3.1 Design remote DCAP attestation infrastructure

The attestation mechanism operates by generating a structure known as quote, which serves as the fingerprint of the enclave. This quote is signed through a hierarchical chain of certificate keys, with the trusted root situated in Intel. The quote is generated within the SGX hardware located in the CSP and transmitted to the user. Equipped with knowledge about how the enclave is constructed, the user can discern whether it has undergone proper initialization or if there are signs of malicious activity. In the latter case, the computation is promptly aborted. Conversely, in the former scenario, the user's attestation service completes the attestation process by starting the secret provisioning. During this phase, the user transmits an arbitrary key to the remote enclave. This key is employed for encrypting files to be kept confidential during the workflow computation and it is exclusively used within the SGX processor for internal decryption purposes.

Figure 2 illustrates the attestation infrastructure employing the DCAP scheme. Multiple entities collaborate to facilitate the attestation process:

Cloud Provisioning Certification Caching Service (PCCS) Responsible for retrieving certificates from Intel services and maintaining an updated cache containing the necessary information for the hardware to generate a verifiable quote.

Intel Provisioning Certification Service (PCS) Manages the distribution of the appropriate certificates to the Cloud PCCS, ensuring the availability of the required certificates.

CSP (with SGX enabled) The environment in which the Gramine-enclaved application is executed.

User The entity where the attestation service is executed to gain assurance that the code will be executed securely within the remote enclave.

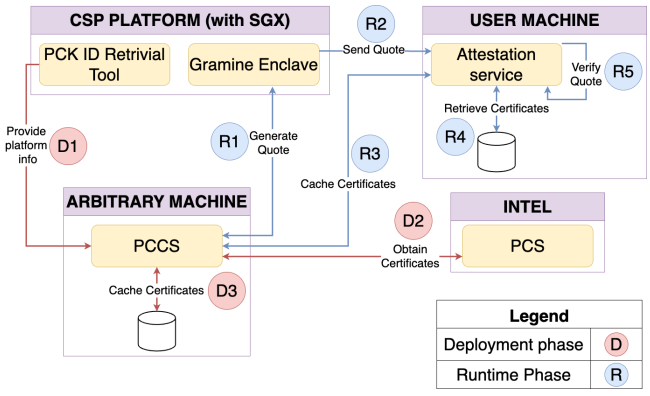

Figure 2: High-level overview of a DCAP infrastructure

The deployment phase gathers essential information, primarily certificates, to generate the quote accurately. This phase is a one-time requirement within the machine where the Gramine-enclaved application will be executed, i.e., in the CSP. To accomplish this, the Intel Provisioning Certification Key (PCK) ID Retrieval tool facilitates the provisioning of all required hardware data to the PCCS (D1). The PCCS then interfaces with Intel's external service PCS to obtain the necessary certificates (D2), subsequently storing them in its local cache (D3). The gathered information is utilized to accurately sign the quote, ensuring that the user can verify it during the runtime phase, that is the stage where the actual remote attestation takes place. The Gramine-enclaved application generates the quote by contacting the PCCS (R1) and dispatches (R2) it to the user, that through the user attestation service, retrieves (R4) information from its local cache. If local cache is empty, the attestation service first contacts the PCCS (R3) to obtain the necessary information. Subsequently, the user employs this information to verify the quote (R5). Upon successful verification, the user gains assurance that the application is instantiated within a SGX enclave, initiating the secret provisioning phase wherein an arbitrary key is transmitted to the CSP. It is used to keep confidential files until they are processed within the SGX processor.

3.2 Design user party

Gramine is a LibOS that needs a manifest in which the mounting of the file system seen by the process must be specified. Accordingly, a structured and predefined file system structure has been established to fulfill this requirement, as shown in Figure 3a. For each workflow step, a corresponding configuration file, denoted as *conf*, is supplied. This file delineates the step type and identifies the code file to be executed, if applicable. The *input/* directory is bifurcated into the *enc/* and *plain/* subfolders. The latter accommodates unencrypted files for which confidentiality is unnecessary. Also, the *enc/* directory contains plaintext files. However, they are intended for encryption before transmission to the CSP. Upon arrival at the CSP, they remain encrypted and are exclusively decrypted within the SGX processor, with the key exchanged after the remote attestation during the secret provisioning. Furthermore, a dedicated *results/* folder is designated to store the outcomes generated by the CSP at each workflow step. Lastly, within the *ssl/* folder are located:

- The executable *attestation_service*, facilitating the remote attestation process and the secret provisioning phase.
- The *provisioning_key*, deployed for the encryption of files within the local machine before transmission to the CSP. This key is subsequently utilized by the CSP for transparent decryption of files specified as encrypted within the Gramine manifest. Additionally, the same key is employed for decrypting the output.
- The user certificate (*usr.crt*), issued to the CSP during the attestation phase, enabling the CSP to authenticate the user party.

3.3 Design CSP party

Figure 3b illustrates the file system design of the CSP where are exemplified some applications compatible with Gramine, such as C and Python. Within each of these application-specific folders, there is a *makefile.template* that is used such a Makefile to build the Gramine manifest necessary to the execution. Although a Makefile could be directly employed, the template file is specifically crafted to afford users the flexibility of potentially specifying certain execution options in the future, such as the enclave size. This design choice aims to enhance the adaptability and customization of the execution environment for user adopting the framework. Within the same

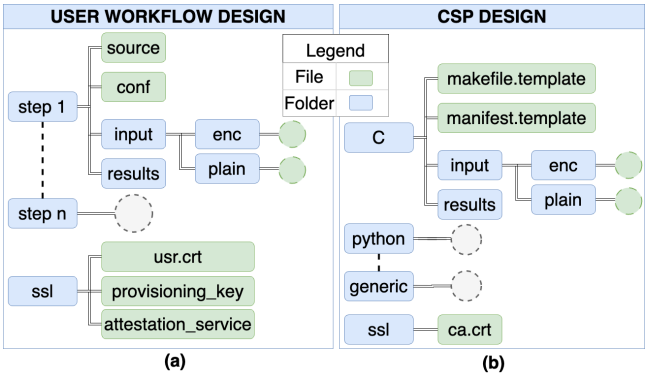

Figure 3: Design of the framework parties. (a) Design of the user File System; (b) Design of the CSP File System.

directory is provided the *manifest.template*, necessary to obtain the proper Gramine manifest because each application requires its distinct libraries and correctly configured environment variables. Hence, in order to prepare any application for the execution in a generic user workflow, it is crucial to configure the CSP to launch the respective application within the Gramine LibOS. Although this task may be straightforward for certain application classes that have undergone extensive testing in Gramine, the challenge escalates when dealing with entirely generic applications, exemplified by the bioinformatics ones.

Furthermore, an identical file system structure to that of the user is predefined, encompassing inputs (*input/enc/* and *input/plain/*) and outputs (*results/*). This uniformity between user and CSP, simplifies the file mounting process within the Gramine manifest, streamlining the integration of files in the same paths. It is imperative to note that this represents the initial folder structure. Throughout the execution of the workflow, the pertinent input files and source code are dynamically loaded into this structure from the user machine.

Finally the *ssl/* subdirectory encompasses the *ca.crt* certificate essential for validating the certificate transmitted by the user during remote attestation.

3.4 Framework implementation

The implementation of the framework has been realized using Bash script and is publicly accessible and open source[3]. An implicit assumption made throughout the preceding sections is that the user possesses the capability to access the CSP through SSH. In the subsequent discourse, the term 'send' refers specifically to the execution of the *scp* command, enabling the secure transfer of files between machines with SSH access. Algorithm 1 shows the pseudocode of user workflow management described as a set of folders denoted as *stepX/* (where $X \in 1, 2, \ldots, N$). For each of these folders, four subtasks are executed:

(1) ***prepare_step.sh*** is in charge of encrypting all files in the *input/enc/* user folder using the dedicated Gramine tool and the *ssl/provisioning_key*. Subsequently, it sends all inputs (including those intended to remain plain in *input/plain/*) and, eventually, the source code to the CSP.

(2) ***enclave_step.sh*** is responsible for the whole attestation and secret provisioning processes, executing the code within the SGX enclave of the Gramine LibOS. Outputs are generated and stored in the *results/* CSP folder.

(3) ***write_back_step.sh*** manages the retrieval of output data from the *results/* folder of the CSP to the user machine's *results/* folder. Additionally, all copied files are decrypted in the same folder.

(4) ***next_step.sh*** facilitates the movement of files from the *results/* folder of step X to the *input/enc/* folder of step $X + 1$. If it is the last step the entire workflow execution concludes.

Alongside these primary scripts, additional scripts have been developed to facilitate the cleanup procedures between the execution of different steps.

Algorithm 1 Workflow management algorithm

```
for i = 1 to N do                          ▷ N number of steps
    ./prepare_step.sh i
    ./enclave_step.sh i
    ./write_back_step.sh i
    if i <= N then
        ./next_step.sh i                   ▷ Not if is last step
    end if
end for
```

4 RESULTS

In order to assess the feasibility of safeguarding complete workflows, the framework has been applied on two distinct examples. The first example (Section 4.1) pertains to a trivial scenario, while the second (Section 4.2) involves a real-world application in the realm of biological computation of DNA. In both cases, the experiment configuration is reported in Table 1, where User and CSP machine are hosted within High-Performance Computing Center for Artificial Intelligence at the University of Turin.[4]

4.1 Workflow: C, Python

The workflow comprises two distinct steps. In the initial step, C code is executed to assess the strength of passwords stored in a dedicated file. Ratings, ranging from 1 to 10, are assigned to passwords, considering key evaluation metrics such as length, character set, entropy, predictability, and commonness [3]. Notably, a separate file containing the 10.000 most common passwords (e.g., 'querty') is employed for reference, and unlike the password file, it is devoid of any confidentiality constraints. The second step in the workflow involves a Python script that processes the generated password ratings and plots an overview histogram illustrating the security levels adopted. Figure 4 shows the execution time of the entire workflow across various configurations. Observably, Gramine as LibOS introduces negligible overhead when compared to native execution. The predominant source of overhead in this context derives from the utilization of SGX. Remarkably, the initialization time for SGX exceeds the overall execution time for both native and Gramine-Direct executions, across both workflow steps. Furthermore, EDMM avoids

[3]https://github.com/lorenzobrescia/POC-Secure-WF-SGX

[4]https://hpc4ai.unito.it

Table 1: Experiment configurations

	User Machine	CSP machine
Operating system	Ubuntu 22.04.2 LTS	Rocky Linux 9.2 Blue Onyx
Kernel	5.15.0	6.5.1
CPU	Intel Xeon Processor (Skylake, IBRS)	Intel Xeon Gold 6346 CPU 3.10GHz
RAM	8G	395G

the need to predefine the enclave size in the Gramine manifest file. This flexibility enables a less constrained configuration of the steps. However, it is noteworthy that the current implementation of EDMM in Gramine-SGX, although offering improvements in initialization time compared to Gramine-SGX without EDMM, introduces a notable extension of execution time. Nevertheless, the incurred SGX overhead, fluctuating within a range of x2 to x5 in comparison to the native execution scenario, may be considered acceptable for gaining privacy-preserving guarantees of SGX enclaves.

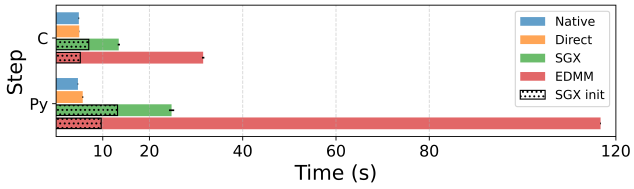

Figure 4: Mean with standard deviation on 10 execution times of different steps in various configurations: Native (without Gramine and SGX), Direct (with Gramine and not SGX), SGX (with Gramine and SGX), EDMM (with Gramine, SGX, and EDMM). SGX init indicates the time spent by SGX to build up the required enclave before the execution.

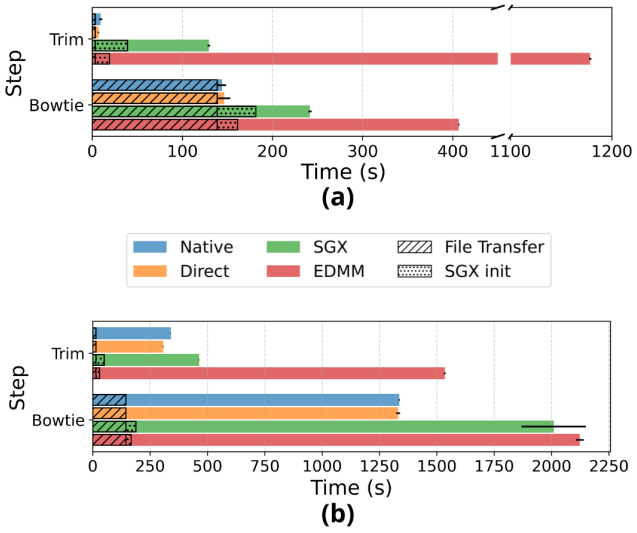

Figure 5: Mean with standard deviation on 10 execution times of different steps in various configurations: Native (without Gramine and SGX), Direct (with Gramine and not SGX), SGX (with Gramine and SGX), EDMM (with Gramine, SGX, and EDMM). File Transfer represents the time spent to move input data from user to CSP. SGX init indicates the time spent by SGX to build up the required enclave before the execution. (a) Small input size. (b) Big input size.

4.2 Workflow: trim_galore, bowtie2

This workflow encompasses the initial two steps of the Next Generation Sequencing (NGS) variant calling pipeline, fully transitioned into a cloud-High Performance Computing (HPC) [14]. The focal point of this workflow involves the analysis and sequencing of human genetic variation, dealing with sensitive DNA data. This real-world application seamlessly integrates with the established framework. The first step involves the execution of trim_galore, which utilizes sensitive data as input and generates output for the subsequent bowtie2 step. In addition to utilizing data previously generated by trim, bowtie2 necessitates the human genome hg38, which, being non-sensitive information, does not require the security guarantees offered by an enclave.

Figure 5a illustrates the execution times for different steps across various configurations. It may appear that the situation has improved compared to the previous workflow. However, the data might be misleading without proper contextualization. Specifically, not accounting for file transfer times, which are irrelevant for evaluating the efficiency of enclaves within Gramine, native and Gramine-Direct executions for both steps conclude within seconds. However, in the case of SGX, the execution time expands by a factor of x20 - x22, exacerbated further with the enabling of the EDMM option that leads to x33 - x200 overheads. This observation raises concerns about the practicality of utilizing SGX enclaves for workflow execution within an untrusted CSP. Nevertheless, Figure 5b depicts the same execution features in a configuration where the input size has been significantly augmented. In this scenario, the timing differentials, between native and SGX configuration narrow to a factor between x1.4 and x1.8 not considering the data movement time. The key observation is that with a small workload, the substantial impact of enclave management time significantly hampers performance compared to the native case, where the overhead of SGX initialization and confidential management is absent. It is possible to notice that there is also an evident increase in file transfer times due to larger inputs. Remarkably, in the case of trim, the Gramine LibOS performs even more efficiently than the native case. Although this may seem counterintuitive, it could be attributed to more effective resource utilization through virtualization in this specific scenario.

5 CONCLUSIONS

Experiments conducted using the framework have demonstrated that, under the right conditions, the overhead ranges between x1.4 and x1.8. It is rational to consider that users migrate computations to a CSP due to limited local resources, justifying the need to increase input sizes to achieve acceptable time results. Indeed, it would be illogical for the users to transition to a remote CSP for computation if they already possess all the required resources locally, enabling them to conduct the task locally. Furthermore, given the evolving nature of technologies like SGX and Gramine LibOS, performance improvements are expected to be realized, gradually mitigating overhead concerns. The implemented framework, functioning as a proof-of-concept, underscores the potential for users to define generic workflows that can be subsequently launched on remote machines without compromising data confidentiality.

Looking ahead, there is an opportunity to enhance the framework's flexibility in defining user workflows, with a future focus on integrating TEEs privacy features into existing Workflow Management System (WMS), such as StreamFlow [4]. Moreover, from the standpoint of cloud-based High-Performance Computing (HPC), it is pertinent to contemplate potential advancements in performance assessment facilitated by parallelization techniques such as multi-threading and multi-processing. Theoretically, the application of these techniques may serve to subsequently mitigate the runtime overhead attributable to TEEs.

ACKNOWLEDGMENTS

This work was supported by the Spoke 1 "FutureHPC & BigData" of ICSC - Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing, funded by European Union - NextGenerationEU.

REFERENCES

[1] AMD. 2020. Strengthening VM isolation with integrity protection and more. https://www.amd.com/system/files/techdocs/sev-snp-strengthening-vm-isolation-with-integrity-protection-and-more.pdf. Accessed: 2024-02.

[2] Eugene Bagdasaryan, Omid Poursaeed, and Vitaly Shmatikov. 2019. Differential privacy has disparate impact on model accuracy. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 32 (NIPS 2019)*.

[3] Katha Chanda. 2016. Password security: an analysis of password strengths and vulnerabilities. *International Journal of Computer Network and Information Security* (2016).

[4] Iacopo Colonnelli, Barbara Cantalupo, Ivan Merelli, and Marco Aldinucci. 2021. StreamFlow: Cross-Breeding Cloud with HPC. *IEEE Transactions on Emerging Topics in Computing* (2021).

[5] Cynthia Dwork. 2006. Differential privacy. In *AUTOMATA, LANGUAGES AND PROGRAMMING, PT 2*. SPRINGER-VERLAG BERLIN.

[6] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. 2022. Benchmarking the Second Generation of Intel SGX Hardware. In *Proceedings of the 18th International Workshop on Data Management on New Hardware*. Association for Computing Machinery.

[7] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. 2021. Security Vulnerabilities of SGX and Countermeasures: A Survey. *ACM Comput. Surv.* (2021).

[8] Craig Gentry. 2009. *A fully homomorphic encryption scheme.* Ph. D. Dissertation. Advisor(s) Boneh, Dan.

[9] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. 2016. Intel software guard extensions: EPID provisioning and attestation services. https://cdrdv2-public.intel.com/671370/ww10-2016-sgx-provisioning-and-attestation-final.pdf. Accessed: 2024-02.

[10] Bertram Ludäscher, Shawn Bowers, and Timothy McPhillips. 2009. *Scientific Workflows*. Springer US, 2507–2511.

[11] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. 2016. Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*.

[12] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. Association for Computing Machinery.

[13] Saeid Mofrad, Ishtiaq Ahmed, Fengwei Zhang, Shiyong Lu, Ping Yang, and Heming Cui. 2022. Securing Big Data Scientific Workflows via Trusted Heterogeneous Environments. *IEEE Transactions on Dependable and Secure Computing* (2022).

[14] Alberto Mulone, Sherine Awad, Davide Chiarugi, and Marco Aldinucci. 2023. Porting the Variant Calling Pipeline for NGS data in cloud-HPC environment. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE.

[15] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can homomorphic encryption be practical?. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*. Association for Computing Machinery.

[16] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted Execution Environment: What It is, and What It is Not. In *2015 IEEE Trustcom/BigDataSE/ISPA*. IEEE.

[17] Vincent Scarlata, Simon Johnson, James Beaney, and Piotr Żmijewski. 2018. Supporting Third Party Attestation for Intel SGX with Intel Data Center Attestation Primitives. https://cdrdv2-public.intel.com/671314/intel-sgx-support-for-third-party-attestation.pdf. Accessed: 2024-02.

[18] Chia-Che Tsai, Kumar Saurabh Arora, Nehal Bandi, Bhushan Jain, William Jannen, Jitin John, Harry A. Kalodner, Vrushali Kulkarni, Daniela Oliveira, and Donald E. Porter. 2014. Cooperation and security isolation of library OSes for multi-process applications. In *Proceedings of the Ninth European Conference on Computer Systems*. Association for Computing Machinery.

[19] Chia-Che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: a practical library OS for unmodified applications on SGX. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*. USENIX Association.

[20] Chathura Widanage, Weijie Liu, Jiayu Li, Hongbo Chen, XiaoFeng Wang, Haixu Tang, and Judy Fox. 2021. HySec-Flow: Privacy-Preserving Genomic Computing with SGX-based Big-Data Analytics Framework. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE.