duet: Combining a Trustworthy Controller with a Confidential Computing Environment

Istemi Ekin Akkus
Nokia Bell Labs
Stuttgart, Germany
istemi_ekin.akkus@nokia-bell-labs.com

Ivica Rimac
Nokia Bell Labs
Stuttgart, Germany
ivica.rimac@nokia-bell-labs.com

***Abstract*—Traditionally, sensitive data is protected with encryption in transit and at rest. Confidential Computing by means of hardware-based Trusted Execution Environments (TEEs) enables organizations to protect their data also *in use*, such that the owner of a TEE application can ensure its integrity via remote attestation. Process-based TEEs (e.g., Intel SGX) allow the integrity check to be performed by other users besides the owner, but require re-architecting the application and lack support for specialized hardware (e.g., GPUs). In contrast, virtual machine (VM)-based TEEs can run entire operating systems confidentially, with access to specialized hardware and ease of use without modifications to applications, but only offer an attestation based on the initial state of the VM.**

In this paper, we propose duet, combining the advantages of process-based and VM-based TEEs; thus, simultaneously providing *runtime integrity*, *ease of use* and *flexibility*. In duet, a process-based TEE application implements a trustworthy controller that acts as the agent of the application owner to deploy and maintain a confidentiality-offering service on a Confidential VM (CVM). As such, any administrative operations conducted on the CVM are performed *only* by the trustworthy controller. By combining the runtime attestation of the controller application and its logs of the administrative CVM operations, third-party users can have more confidence on the CVM's (and the service's) integrity. At the same time, application owners can benefit from specialized hardware supported by the CVM, the ease of use when deploying applications and the flexibility it offers in terms of runtime maintenance.

1. Introduction

Safeguarding data *in transit* and *at rest* has long been a cornerstone of security measures in the realm of cloud computing. As organizations increasingly entrust their sensitive data and workloads to cloud environments, protecting data *in use* has become a critical concern [7], [15], [18], [29]. Confidential computing offers a transformative solution to this challenge by leveraging hardware-based Trusted Execution Environments (TEEs) to provide integrity and confidentiality protection for sensitive computations [17]. Through remote attestation [16], the owner of an application running in a TEE can verify its integrity, thus, establish trust in the confidentiality and integrity of the computations performed within it.

Underlying the concept of confidential computing is the delineation of trust between two entities: the application owner, who establishes trust in the computing environment, and the platform operator, who manages the infrastructure. However, in many scenarios, the TEE application provides a *service to a user*, a third party with whom the application/service owner has a transactional relationship. The arising challenge is then to extend the trust established by the owner in the TEE to the service user: the user must also be assured of the environment's integrity to convince them that their interactions with the service occur in a secure and confidential manner.

Today, application owners can choose between two primary technical solutions that have emerged: process-based TEEs (e.g., Intel SGX) that isolate processes, and virtual machine (VM)-based TEEs (e.g., AMD SEV-SNP, Intel TDX) that isolate entire operating systems (OS). Each approach offers distinct advantages and considerations. Process-based TEEs enable the relay of attestation quotes and offer a granular level of transparency, allowing a user to reason about the runtime state of the application; thus, establishing trust [32]. However, such TEEs require re-architecting the application or using a library OS [48], [49], making it inflexible and limiting for existing applications. In comparison, VM-based TEEs support a wider range of hardware (e.g., GPUs) and provide greater flexibility as well as ease of use for the service owner, because the applications do not need to be modified. However, they present challenges in extending trust to all stakeholders: the service owner can interact with the VM *at runtime after the attestation of the initial VM state* [2] without the service user's awareness, potentially undermining trust in the integrity and confidentiality.

One approach to tackle this challenge is the customization of the VM image loaded into the VM-based TEE, so that the service software is loaded and all interfaces other than those provided by the service are disabled during initialization of the VM [21]. This approach demands meticulous attention and expertise given the complexity and possibilities of a modern OS, and any update to the service requires the creation and attestation of a new OS image. Another approach is the use of a virtual Trusted Platform Module (vTPM) [41], [46], such that the TPM configuration of a VM only allows the loading and execution of correct and intended software: a service user then receives not only the VM's attestation report but also the vTPM state, so that they have increased confidence that the confidentiality-offering service is not interfered with.

However, creating such a TPM configuration is typically outside the expertise of the service owner and users.

In this paper, we propose an alternative approach as a primitive for confidentiality-offering services. Our system, **duet**, combines the runtime transparency of a process-based TEE with the flexibility of a Confidential Virtual Machine (CVM). In **duet**, a process-based TEE (e.g., Intel SGX) application acts as a mediator and a controller on behalf of the service owner: it launches the CVM with a well-known OS image (e.g., Ubuntu CVM [50]), which has been presumably scrutinized by more developers compared with a custom image, and performs all administrative tasks at runtime (e.g., deployment and maintenance of service). The controller application executes publicly available code with a well-known interface and a small trusted computing base, allowing easy verification and trust establishment by remote attestation [32].

The next section provides the motivation for our work followed by related work in Section 3. We then present the design of **duet** and its implementation in Sections 4 and 6, respectively. We conclude with a discussion and directions for future work in Section 7. The source code of **duet** is publicly available [45].

2. Motivation & Goals

Here, we present our general terminology, actors and assumptions we make about them as our threat model. We also describe motivational use cases and our goals.

2.1. Terminology & Actors

We have the following actors. The **service owner** operates an application (i.e., develop, deploy, maintain). We assume that the application's main benefit is that it offers confidentiality to its users, whereby the users use the service without exposing their confidential/private assets (e.g., data, code) to the service or infrastructure (e.g., confidential ML training [27], [36], [40], [52]).

A **service user** interacts with the confidentiality-offering service to perform actions offered by the application. These actions may operate on the confidential/private assets of the user; thus, the user requires that the assets are handled in a confidential manner with TEEs and that these TEEs with the service code running on them to be attested. For example, with Intel SGX, the user may verify the integrity of SGX enclaves by remote attestation [32]. Similarly, with CVMs running on AMD SEV-SNP or Intel TDX, the user may verify the CVM's initial state with a well-known image (e.g., Ubuntu 22.04 CVM) [2], [8].

Besides these main actors, the **infrastructure provider** supplies the underlying TEE resources (e.g., cloud offering, on-premise server) to the service owner, but does not interact with the service owner nor with service users during service operation.

2.2. Threat Model & Assumptions

We assume, for a confidentiality-offering service using TEEs, the service code and its package dependencies are publicly available (e.g., open-source). Some packages may not be open-source (e.g., drivers for a GPU); however, we assume they are well-known and officially endorsed by their respective providers (e.g., firmware for H-100 confidential GPU signed by NVIDIA). We also assume that the package managers in the OS check the integrity of the packages, and only install packages from verified repositories (e.g., the package manager 'apt' for Debian and Ubuntu compares the checksum of a downloaded package with the checksum from the 'Release' file signed by the repository maintainer [37]).

We also assume that the infrastructure provider does not interfere with the mounting of the filesystem (i.e., 'rootfs') during the launch. The attestation report covers only the firmware, but not the filesystem. There is ongoing research to ensure that the 'rootfs' was mounted without tampering by encrypting the disk and sharing the key only after the successful boot [20], [35].

In addition, there may be some variation in the CVM offerings of cloud providers. For example, Azure offers CVMs with a closed-source firmware that does not allow direct access to the TEE hardware (e.g., no `/dev/sev`). Instead, it exposes a secure vTPM device, which can be used to access the attestation reports from VM-based TEE hardware. One can also use a Microsoft-provided library to obtain and verify an attestation token via Microsoft Azure Attestation (MAA), stating that the CVM launched in a trustworthy manner. The closed-source nature of the firmware and MAA forces the service owners and users to also trust Microsoft's implementation of both [8].

In comparison, a CVM obtained from AWS with AMD SEV-SNP hardware allows direct access to the TEE hardware and its attestation report [5]. This report can be verified via the recommended and open-source 'snpguest' tool from Virtee [51]. Similarly, Google Cloud offers both a software-based vTPM and direct hardware access to obtain attestation reports for VM-based TEEs (AMD SEV-SNP and Intel TDX) [22] via tools that are open-source [23], [24]. In these cases, the service owners and users not necessarily trust the infrastructure provider.

Many TEE manufacturers take a detailed approach to patching vulnerabilities in their products [34]. As such, we assume that the infrastructure provider keeps their platform up-to-date, and leave side-channel attacks on TEE hardware (e.g., SGX) outside our scope. One can also imagine that the trustworthy controller can be offered as a service from a cloud provider that does not collude with the service owner (Section 7).

2.3. Motivation for CVMs

Specialized hardware: Confidential GPUs and their attestations (e.g., NVIDIA H-100) require the use of a confidential VM using AMD SEV-SNP or Intel TDX technology [42], [43]. Access to such specialized hardware becomes more and more relevant with increasing interest in trustworthy AI/ML applications [39], [52]. Existing SGX applications providing confidentiality for their users via remote attestation cannot utilize such hardware unless they migrate to a CVM. However, their guarantees for their users change because the CVM attestation only covers the initial state of the CVM: service owners will have to ensure that the current state of the CVM is measured in the attestation, which is not straightforward (Section 3).

Ease-of-use and flexibility: Modifying a CVM for customization or configuring the TPM state are complex operations. Although cloud providers may provide vTPM implementations for their offerings, these usually apply to commonly used OS images, amortizing the effort spent. A service owner must ensure the confidentiality-offering service code is included in the measurement during the launch of the CVM [21]. This approach presents at least two issues: First, every update to the service code or dependency requires building the image and launching the CVM again. As a result, the deployment and maintenance of the confidentiality-offering service can become complex. Second, the service owner has to convince service users of the validity of the new image and its attestation measurement values (i.e., the new image is deemed trusted). Moreover, service users need to be made aware of the changes, producing additional burden.

Isolation of potentially malicious code: Some confidentiality-offering services may utilize *external* code working on private and confidential assets. This external code may need to be kept confidential due to intellectual property rights. For example, an ML platform may aim that the code is kept confidential [52], or the analysis on some private datasets may need to stay confidential [36]. Consequently, this code cannot be inspected or shared with other collaborators (e.g., data owners), making it difficult to ensure that it does not leak private data [4]. Isolating and sandboxing such code using existing SGX approaches is tricky and limiting at best [28]. Moreover, existing library OS (libOS) approaches facilitating easy deployment of applications with SGX do not consider this threat model [26], [44]: the service code, the libOS and the potentially malicious code all run in the same address space, allowing the external code to secretly modify the inner state of the SGX enclave. In contrast, a CVM can offer additional tools to restrict such code and prevent its malicious activities if any.

2.4. Goals

Our goal is to enable a service owner to prove to the service users that the service is running in a trusted environment and its integrity is preserved. We also want the owner to benefit from using a modern OS for ease of use, flexibility and confidential GPU access, while reducing deployment and maintenance effort for the service.

3. Related Work

Remote attestation has been the topic of extensive research in confidential computing [16]. Intel SGX provides a process-based TEE with the confidentiality boundary drawn around the application code, which is measured and included in the attestation quote [30]. A service user can request the quote from the TEE and verify its initial state, assuming application code availability to the user. Because Intel SGX guarantees the integrity of the code running in the enclave, the user can establish trust also in the runtime state of the service. Process-based TEEs, however, do not support lift-and-shift but require extra-effort for adaptation of applications [48], [49] and lack isolation mechanisms necessary to sandbox potentially malicious third-party code that needs to stay confidential [36], [52].

Attestation in VM-based TEEs (e.g., AMD SEV-SNP) is based on measurements of the VM state after initialization by the hypervisor, and typically includes only the first virtual firmware volume [2]. One approach for extending the trust to other boot-time binaries, including the kernel, init RAM disk, and kernel command line, has been presented in [38] and implemented in patches to the QEMU hypervisor and the virtual firmware OVMF. Another measured boot approach in AMD SEV-SNP implements a virtual Trusted Platform Module (vTPM) as a Secure VM Service Module (SVSM) in the VM firmware [41]. Although these approaches can guarantee the integrity of boot components, they cannot protect the runtime integrity of the VM (e.g., when the owner installs other software).

Besides a measured boot, Revelio [21] further provides runtime integrity of the VM by customizing its image: it is configured to start with the confidentiality-offering service on boot with a read-only filesystem and to block any inward connections, so that no outside entity can connect to it and tamper with any process. However, approaches based on customization of the VM image *per service* require a) operating system expertise from both the service owner and users (e.g., to build and reproduce the image), b) rebuilds of the VM image with every change to the service (e.g., no runtime updates to the service or OS), and c) support from the cloud provider's hypervisor. In contrast, **duet** refrains from customized VM images, thus, can be deployed on any cloud and only expects a limited level of OS knowledge (i.e., basic installation commands and their expected outcomes).

Following a similar concept to ours, Antonino et al. [3] propose using a trusted SGX enclave to provision and attest a CVM. The measurement of the CVM is embedded into the attestation protocol of the SGX enclave, addressing the limitations specific to an older version of the AMD SEV (i.e., pre-SNP) attestation. These limitations do not exist anymore in newer AMD SEV-SNP and Intel TDX attestation procedures [2] used by **duet**. In addition, this solution requires the customization of the VM image, provides evidence of the CVM state only at a specific point in its lifecycle and assumes that the application runs for a specific time, producing some data later embedded in the SGX quote on behalf of the VM. In contrast, **duet** targets long-running services and allows users to audit the state of the CVM *at any point*, even when the CVM's initial state has been updated after initialization.

The Confidential Containers (CoCo) project [19] targets deploying confidential container workloads in Kubernetes. The customized VM image running the containers includes an agent and a policy framework, covered by the attestation measurements. By exposing a well-defined API for deploying/managing the container workloads and a configurable policy layer, interactions with the trusted environment are limited and controlled. However, this approach is specific for container workloads, while **duet** provides a general solution to any VM-based workload.

4. Design

In this section, we first present **duet**'s components (Section 4.1). We then describe the details of **duet**'s trustworthy controller and its interface (Section 4.2). Section 4.3 explains how the service owner sets up the deployment

TABLE 1. **DUET**'S TRUSTWORTHY CONTROLLER INTERFACE.

API endpoints	Purpose	Operations	Invokable by
`/quote`	Attest controller enclave	1. Produce SGX quote	Owner, Users
`/start_cvm`	Start CVM	1. Provision resources for CVM	Owner
`/run_cvm_commands`	Deploy and maintain service	1. Install, launch and update service as well as dependencies 2. Record commands and outputs	Owner
`/mark_cvm`	Change the CVM mode	1. Record new mode of the CVM 2. Kill long-running processes (when changed to 'in-update')	Owner
`/get_cvm_state`	Attest CVM's current state	1. Produce CVM attestation report 2. Return logs of commands and outputs	Owner, Users
`/stop_cvm`	Stop service	1. Delete resources for CVM	Owner

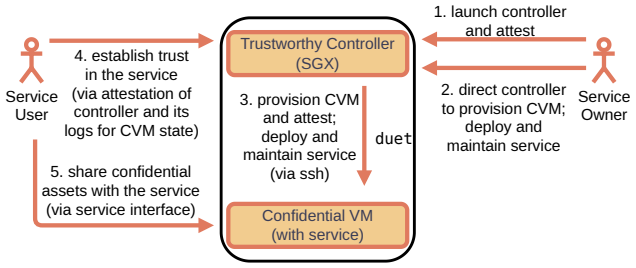

Figure 1. High-level overview of **duet**.

of the confidentiality-offering service using the controller. Finally, we show how service users gain confidence on the service's integrity by building the chain of trust starting from the trustworthy controller (Section 4.4).

4.1. duet Components

Figure 1 shows the high-level overview of a service that is deployed using **duet**. There are two components that work together to enable the service users establish trust in the service for their confidential/private assets.

Confidential Virtual Machine (CVM): The CVM is a standard VM running a well-known OS image instantiated on a VM-based TEE hardware (e.g., Ubuntu 22.04 CVM on AMD SEV-SNP or Intel TDX). As such, it allows the service owner to benefit from specialized hardware (e.g., confidential GPUs) as well as ease of use and flexibility of a modern OS as supported by the TEE. As a result, the service owner does not need to re-architect the application according to specific hardware specifications (e.g., Intel SGX) nor does she need to utilize other software (e.g., a library OS [48], [49]) to retrofit it.

Trustworthy Controller: The controller is a generic and simple program that exposes various API endpoints to the service owner and users (Section 4.2). The controller code is publicly available and runs in an SGX enclave, so that a standard remote attestation can be performed on the enclave and its code. Through its interface, the controller allows a service owner to initialize and maintain a service. At the same time, it allows service users to build trust in the service, by allowing them to inspect the CVM's initial state (i.e., via remote attestation) as well as any actions (and their outputs) applied to the CVM after the initialization (i.e., via the controller's logs).

For brevity, we assume that the controller is dealing with a single service. However, there is no reason why it cannot be utilized for multiple CVMs (e.g., if the service requires multiple cooperating CVMs) or for multiple services (e.g., if the owner operates multiple services). We discuss these additional capabilities later (Section 7).

4.2. Controller Interface & Operations

Table 1 summarizes **duet**'s controller interface API endpoints, their purposes, the high-level actions taken by the controller when invoked and who may invoke them. All communication with the controller happens with encrypted messages (e.g., TLS). Below, we describe how a service owner and users can use these API endpoints.

Some endpoints may perform privileged operations (e.g., provisioning resources for the CVM, installing packages), such that only the service owner can invoke these endpoints. During its operation, the controller performs authentication and authorization. For example, the service owner may launch the controller in SGX with her public key as an input, such that it requires the service owner's signature for privileged endpoints. For brevity, we do not further describe this check and only refer to who is authorized to utilize an endpoint.

`/quote`: This endpoint returns the standard SGX attestation quote for the enclave hosting the controller. It can be invoked by both the service owner and service users, such that they can establish trust in the controller via remote attestation [10], [31], [32]. For service users, this remote attestation is crucial because it enables them to establish trust in the CVM's and the service's runtime integrity and entrust their confidential/private assets to the service. In other words, the remote attestation of the controller forms the root of trust for the service users.

`/start_cvm`: This endpoint provisions the necessary CVM resources from the underlying infrastructure. This infrastructure can either be a cloud provider [9] or a local/on-premise CVM-capable server (e.g., AMD SEV-SNP, Intel TDX). As such, it can only be invoked by the service owner. Its input parameters define the infrastructure (e.g., 'Azure', 'on-premise'), the type of the CVM (e.g., 'sev-snp', 'tdx'), hardware resources (e.g., number of processors, memory size) and the operating system (OS) image (e.g., Ubuntu 22.04 CVM).

As part of the CVM launch process, the controller generates a random SSH key for logging into the CVM. It

uses the public part of the SSH key as the only login info while provisioning the resources from the infrastructure provider. Note that the private part of the SSH key is not accessible outside the controller enclave to anyone, including the service owner and the infrastructure provider.

After provisioning the resources for the CVM, the controller performs the necessary updates and installations. Afterwards, the controller retrieves and verifies the CVM's attestation report [11]–[13], [51], whose `HOST_DATA` field can include the public part of the SSH key [1]. The controller then stores the attestation report, which corresponds to the initial state of the CVM. Finally, the controller produces an identifier for the CVM that was launched (e.g., uuid) and returns it, such that the service owner can perform further actions on this CVM by supplying it as a parameter for other endpoint invocations.

`/run_cvm_commands`: This endpoint triggers the controller to log in to the corresponding CVM and execute a list of commands in the same order given as input. As such, it is a privileged operation and can only be invoked by the service owner. The controller logs the requested commands, executes them remotely on the CVM and records their outputs as part of the CVM's state. These commands and their outputs will be available to service users, allowing them to inspect the state of the CVM. To save memory, the controller can encrypt this metadata with a symmetric key produced by the controller inside the enclave and persist it in either remote or local storage.

`/mark_cvm`: This endpoint changes the CVM's mode of operation to either 'in-service' or 'in-update' and can only be invoked by the service owner. The initial mode of a CVM is 'in-update', in which the `run_cvm_commands` API is enabled for the service owner. After the service owner finishes running commands, she marks the CVM as 'in-service', in which the controller disables the `run_cvm_commands` API. The controller records any CVM mode change.

If the service owner wants to perform more runtime updates to a CVM (e.g., upgrading packages, patching vulnerabilities) when it is 'in-service', she uses the `mark_cvm` API endpoint to mark the CVM as 'in-update'. Before allowing any commands, the controller ensures that there are no long-running processes remaining in the CVM that were started by the service owner. After the updates are finished, the service owner again marks the CVM's mode as 'in-service' to start serving users again.

`/get_cvm_state`: This endpoint can be invoked by both the service owner and service users, and returns the current state of the CVM. This state consists of the CVM's mode, initial attestation report, the list of commands that were issued to the CVM and their outputs. For service users, the combination of these pieces provides enough information on how the confidentiality-offering service has been instantiated, run and maintained, such that they can have confidence on its correctness and integrity.

`/stop_cvm`: This endpoint stops the CVM hosting the confidentiality-offering service. It can only be invoked by the service owner using the CVM's identifier obtained after starting the CVM. Note that the metadata belonging to this CVM can still be kept, such that the service users can later audit the CVM's latest state.

4.3. Service Owner Workflow

At a high-level, the deployment and maintenance of a confidentiality-offering service using **duet** adheres to the following workflow. First, the service owner launches the controller in an Intel SGX enclave (Step 1 in Figure 2), obtains an attestation quote and verifies it, either using ECDSA- [10], [32] or EPID-based attestation [30] (Steps 2-5). The code of the controller is publicly available, such that any service user can reproduce the measurement value of the enclave included in the quote (i.e., `MRENCLAVE`).

Once the remote attestation procedure succeeds, the operator directs the trustworthy controller to launch a CVM (Step 6). The controller then provisions a VM-based TEE (i.e., AMD SEV-SNP, Intel TDX) with a well-known image (e.g., Ubuntu 22.04 CVM), either from a cloud provider (e.g., Azure [9]) or from an on-premise server (Step 7). The CVM image can be minimal, such that unnecessary services are not present. The only requirement **duet** has is that the CVM has the SSH server enabled.

Once the CVM is launched, the controller performs a remote attestation procedure (Steps 8-16), by first obtaining the attestation report from the CVM (Steps 12-13) and then checking it with the attestation service provider (Steps 14-15). The controller stores the attestation report to later supply it to service users. Because the initial CVM image is a well-known open-source image, service users can also trust the CVM's initial state. When this attestation procedure is complete, the controller returns a unique id to the service owner (Step 16). Note that before the CVM attestation, the controller may need to perform some actions on the CVM, such as installing dependencies and attestation related software [12], [13], [51]. These actions and their outputs are logged by the controller (Steps 8-11), like any other command issued by the service owner.

After the CVM attestation, the service owner requests the controller to execute commands in the CVM to install the necessary packages and software for the service (Step 17). The controller logs in to the CVM using the SSH key generated at the launch, runs the commands and collects their outputs (Steps 14-18). The controller also installs the confidential service and configures it according to the specifications (Steps 18-21). When done, the service owner marks the CVM as 'in-service' (Step 23). After the initial deployment, the owner may also maintain the CVM and the service by updating packages and software the same way, after marking the CVM as 'in-update' again and letting the controller kill the long-running processes started by the owner. Note that these packages and software are publicly available (e.g., Ubuntu packages), or well-known and assumed trustworthy (e.g., NVIDIA-signed firmware for GPUs) [43].

During these operations, the controller records the commands and their outputs (Steps 18 and 21) as well as any mode changes. Combined with the CVM's initial attestation report (Steps 13-15), these logs of the commands and outputs correspond to the CVM's latest state, enabling the service users to audit the CVM's, and in turn the service's, runtime integrity. As a result, **duet** enables service owners and users to overcome the limitations of static attestation reports of CVMs.

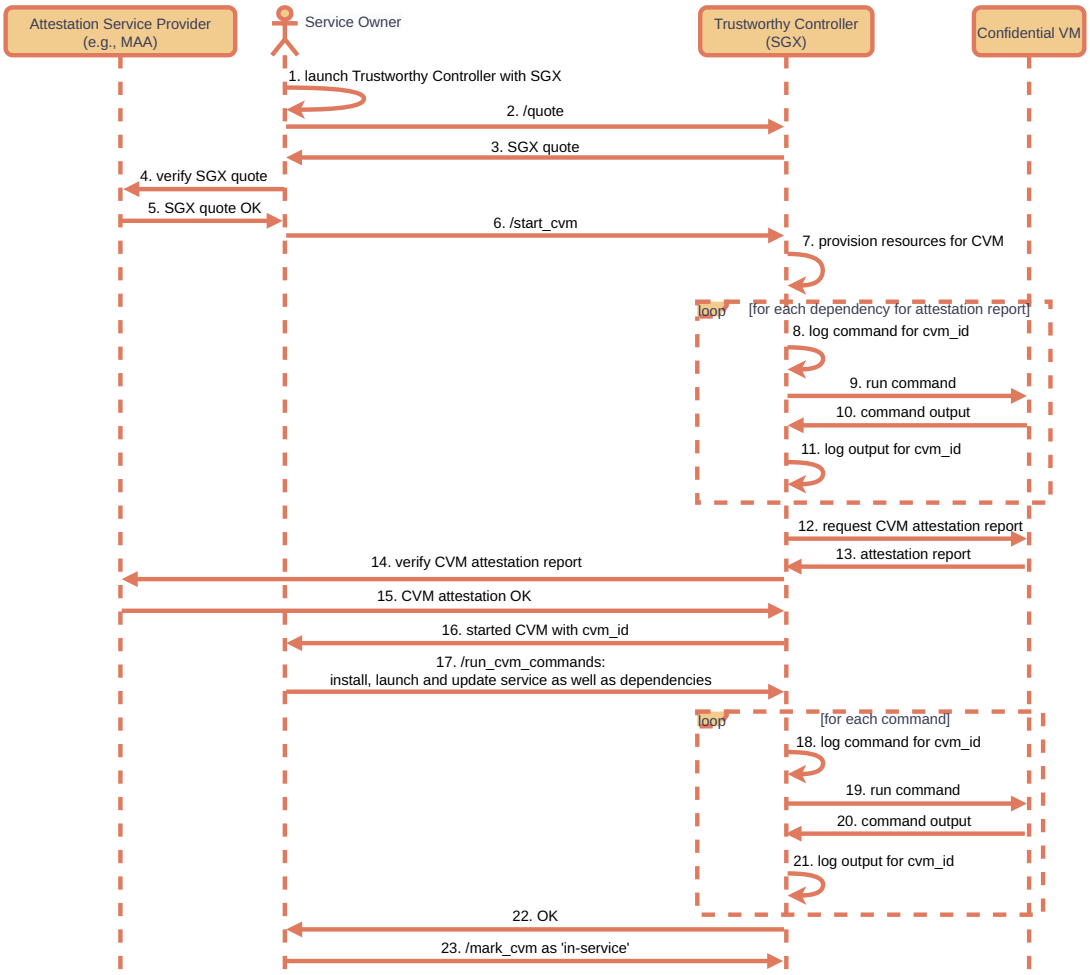

Figure 2. Service owner workflow to deploy and maintain the confidentiality-offering service via **duet**.

4.4. Service User Workflow

Once the confidentiality-offering service is configured by the trustworthy controller, it starts serving users. The users interact with the CVM as defined by the confidentiality-offering service. Before the users trust the CVM and the service with their confidential/private assets, they first need to establish trust in them. The high-level workflow of a service user establishing trust in the CVM and the service is depicted in Figure 3. The users first establish trust in the controller via SGX remote attestation (Steps 1-4 in Figure 3); trust is subsequently extended to the CVM and the confidentiality-offering service running on it: the CVM was started from a well-known state available via its attestation report, and any subsequent updates to the CVM were applied and logged by the trustworthy controller (Steps 5-9). By inspecting the list of commands, their expected outcomes and the current mode of the CVM (i.e., 'in-service'), the users can check the runtime integrity of the CVM and the service.

Once that trust is extended to the CVM and the service, the users can start using the service with their confidential/private assets (Step 10). For example, the users can share their encryption keys for their assets with a key service, so that the sharing of those keys are tied to the CVM's current state. The key service may run on another TEE (e.g., another SGX application) [47], or the controller can be extended to also store the users' keys.

5. Security Analysis

In this section, we present an analysis of **duet** according to our assumptions and threat model (Section 2.2).

Access to the CVM: The SSH interface in the CVM is only accessible by the trustworthy controller. During the launch, the controller generates an ephemeral SSH key specific to this CVM. The controller runs inside an SGX enclave, so that the private part of the SSH key is not visible externally.

Insecure packages: Package managers like 'apt' ensure the integrity of the packages they install and only use packages that were signed by the repository maintainers [37]. To install backdoored packages, the attacker would have to compromise the package repository. Disabling this check to allow unsigned packages means changing the CVM's package manager configuration, which would be visible in the controller's logs.

Runtime updates: There are two cases regarding updates.

Case 1: The service in the CVM has not served any users yet, so that no keys and confidential data are available at the CVM. Any commands and outputs (e.g., to install packages, kernel modules, services to bypass the

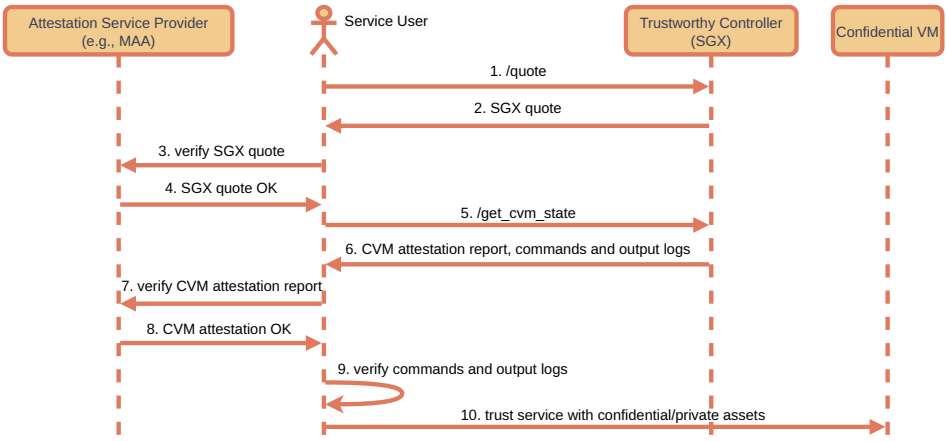

Figure 3. Service user workflow to establish trust in confidentiality-offering service via **duet**.

controller, to change configuration) will be recorded by the controller; thus, will be visible to the service users that can inspect them. Because the logs are only kept by the controller, the service owner cannot tamper with them.

Case 2: The service has already served some users, so that some confidential/private data may be available at the CVM. As a result, any additional command issued by the service owner risks leaking confidential/private data (e.g., dumping the memory of a process and transferring it out). However, to be able to issue new commands, the service owner first has to mark the CVM as 'in-update', which triggers the controller to kill any long-running processes started by the service owner (e.g., as part of the service), preventing memory dumps, before any new commands are allowed. Any code in the service bypassing this restriction (e.g., writing to disk, sending memory content over network) will be visible to service users and raise suspicion because the code of the service and installed packages are assumed to be open-source.

6. Prototype Implementation

We implemented a prototype of **duet** and tested it for feasibility. The source code for the controller and client are publicly available [45]. The controller is about 1K lines of Python code. We containerize the controller with gramine libOS [25] and deploy it on an SGX machine on Azure (DC2sv3). Our current implementation uses Azure to provision CVMs with AMD SEV-SNP (DC2adsv5) or Intel TDX (DC2edsv5). We use a well-known Ubuntu image designated for CVMs from Canonical [50] ("canonical:0001-com-ubuntu-confidential-vm-jammy:22_04-lts-cvm:latest"). We follow Azure's guidance for attesting the CVM with the HCL firmware and MAA provided by Microsoft [8]. Unfortunately, the HCL firmware and MAA code are not publicly available, forcing the service owner and users to also trust Microsoft. We plan extending our prototype to also provision CVMs from on-premise servers without this extra trust.

We also implemented a client that interfaces with the trustworthy controller to perform actions. The client is about 300 lines of Python code and currently acts as the service owner to issue some commands for testing. These commands include obtaining an SGX quote from the controller and verifying it, starting a CVM with a default package list to obtain its attestation report [12], [13], running some commands and getting their output, changing its mode, obtaining the CVM's current state and finally stopping the CVM. All packages installed by default are visible within the controller's code.

7. Discussion & Future Work

In this paper, we proposed **duet** that enables a service owner to deploy and maintain a confidentiality-offering service running in a CVM with the use a trustworthy controller running in a process-based TEE. This approach allows the service owner to benefit from a full OS for specific hardware access, ease of use and flexibility for developing, deploying and maintaining the service. At the same time, it allows service users to verify the runtime integrity of the CVM and the service via the runtime integrity of the controller, because the controller is the only entity with administrative capabilities on the CVM.

An obvious disadvantage of **duet** is that it requires two TEE technologies. Fortunately, both are readily available and cheap in today's cloud infrastructure [6], [14]. One potential bottleneck **duet** can introduce is that the service users have to first establish trust in the controller and CVM via remote attestation *before* they interact with the service. We note that the controller can be scaled out easily by sharing the SSH key with other trustworthy controllers via the sealing/unsealing primitive in Intel SGX [33].

As mentioned in Section 4.1, **duet**'s trustworthy controller can handle multiple CVMs to support services with multiple instances of the same component for scalability. For example, Citadel [52] uses separate training enclaves for each dataset owner in a collaborative and confidential/private ML training system. **duet** can facilitate these types of services to migrate to CVMs and benefit from their advantages (e.g., confidential GPU access, OS-level sandboxing mechanisms), and orchestrate their workloads.

Similarly, a single **duet** controller can be used for different confidentiality-offering services and their respective CVMs: the controller is generic with no dependencies on the launched service; thus, can be reused. A cloud provider can also offer **duet** as a service for service owners, so that they can extend process-based TEE attestation properties (i.e., service integrity) to a CVM while allowing runtime updates on it. We leave these extensions for future work.

Acknowledgements

We thank our anonymous reviewers and our shepherd for their feedback and suggestions to improve this paper.

References

[1] SEV Secure Nested Paging Firmware ABI Specification. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/56860.pdf. Last accessed on 14.05.2024.

[2] AMD SEV-SNP Attestation: Establishing Trust in Guests. https://www.amd.com/content/dam/amd/en/documents/developer/lss-snp-attestation.pdf. Last accessed on 05.03.2024.

[3] Pedro Antonino, Ante Derek, and Wojciech Aleksander Wołoszyn. Flexible remote attestation of pre-snp sev vms using sgx enclaves. *IEEE Access*, 11, 2023.

[4] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE transactions on information forensics and security*, 13(5):1333–1345, 2017.

[5] Attestation with AMD SEV-SNP - Amazon Elastic Compute Cloud. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/snp-attestation.html. Last accessed on 06.05.2024.

[6] EC2 On-Demand Instance Pricing - Amazon Web Services. https://aws.amazon.com/ec2/pricing/on-demand/. Last accessed on 04.03.2024.

[7] Key foundations for protecting your data with Azure confidential computing. https://azure.microsoft.com/en-us/blog/key-foundations-for-protecting-your-data-with-azure-confidential-computing/. Last accessed on 06.03.2024.

[8] confidential-computing-cvm-guest-attestation/cvm-guest-attestation.md at main - Azure/confidential-computing-cvm-guest-attestation - Azure Confidential VMs attestation guidance & FAQ. https://github.com/Azure/confidential-computing-cvm-guest-attestation/blob/main/cvm-guest-attestation.md. Last accessed on 04.03.2024.

[9] About Azure confidential VMs — Microsoft Learn. https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-overview. Last accessed on 04.03.2024.

[10] Sample Code for Intel SGX Attestation using Microsoft Azure Attestation. https://github.com/Azure-Samples/microsoft-azure-attestation/tree/master.

[11] What is guest attestation for confidential VMs? — Microsoft Learn. https://learn.microsoft.com/en-us/azure/confidential-computing/guest-attestation-confidential-vms. Last accessed on 04.03.2024.

[12] Use sample application for guest attestation in confidential VMs — Microsoft Learn. https://learn.microsoft.com/en-us/azure/confidential-computing/guest-attestation-example?tabs=linux. Last accessed on 04.03.2024.

[13] GitHub - Azure/confidential-computing-cvm-guest-attestation: Confidential VM Platform Guest attestation sample apps. https://github.com/Azure/confidential-computing-cvm-guest-attestation. Last accessed on 04.03.2024.

[14] Pricing - Linux Virtual Machines — Microsoft Azure. https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/#pricing. Last accessed on 04.03.2024.

[15] RBC creates relevant personalized offers while protecting data privacy with Azure confidential computing. https://customers.microsoft.com/en-us/story/1356341973555285762-royalbankofcanada-banking-capital-markets-azure. Last accessed on 06.03.2024.

[16] Henk Birkholz, Dave Thaler, Michael Richardson, Ned Smith, and Wei Pan. Remote ATtestation procedureS (RATS) Architecture - RFC 9334. https://datatracker.ietf.org/doc/rfc9334/. Last accessed on 04.03.2024.

[17] A Technical Analysis of Confidential Computing. https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3_unlocked.pdf. Last accessed on 06.03.2024.

[18] Confidential Computing: Hardware-Based Trusted Execution for Applications and Data. https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC_outreach_whitepaper_updated_November_2022.pdf. Last accessed on 06.03.2024.

[19] Confidential Containers Project. https://confidentialcontainers.org/. Last accessed on 08.03.2024.

[20] Encrypted Virtual Machine Images for Confidential Computing. https://kvmforum2021.sched.com/event/ke4Y. Last accessed on 14.05.2024.

[21] Anna Galanou, Khushboo Bindlish, Luca Preibsch, Yvonne-Anne Pignolet, Christof Fetzer, and Rüdiger Kapitza. Trustworthy confidential virtual machines for the masses. In *Proceedings of the 24th International Middleware Conference*, Middleware '23. Association for Computing Machinery, 2023.

[22] Confidential VM attestation — Google Cloud. https://cloud.google.com/confidential-computing/confidential-vm/docs/attestation. Last accessed on 06.05.2024.

[23] google/go-sev-guest offers a library to wrap the /dev/sev-guest device in Linux, as well as a library for attestation verification of fundamental components of an attestation report. https://github.com/google/go-sev-guest. Last accessed on 06.05.2024.

[24] google/go-tdx-guest offers a library to wrap the /dev/tdx-guest device in Linux, as well as a library for attestation verification of fundamental components of an attestation quote. https://github.com/google/go-tdx-guest. Last accessed on 06.05.2024.

[25] GitHub - gramineproject/gramine: A library OS for Linux multi-process applications, with Intel SGX support. https://github.com/gramineproject/gramine. Last accessed on 04.03.2024.

[26] Threat Model for gramine project - gramineproject/gramine - Discussion #1465. https://github.com/gramineproject/gramine/discussions/1465. Last accessed on 04.03.2024.

[27] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018.

[28] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM Transactions on Computer Systems (TOCS)*, 35(4):1–32, 2018.

[29] What is confidential computing? — IBM. https://www.ibm.com/topics/confidential-computing. Last accessed on 06.03.2024.

[30] Intel SGX: Intel EPID Provisioning and Attestation Services. https://www.intel.com/content/www/us/en/content-details/671370/intel-sgx-intel-epid-provisioning-and-attestation-services.html. Last accessed on 04.03.2024.

[31] Code Sample: Intel Software Guard Extensions Remote Attestation End-to-End Example. https://www.intel.com/content/www/us/en/developer/articles/code-sample/software-guard-extensions-remote-attestation-end-to-end-example.html. Last accessed on 04.03.2024.

[32] Attestation Services for Intel Software Guard Extensions. https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html. Last accessed on 04.03.2024.

[33] Innovative Technology for CPU Based Attestation and Sealing. https://www.intel.com/content/dam/develop/external/us/en/documents/hasp-2013-innovative-technology-for-attestation-and-sealing-413939.pdf. Last accessed on 06.03.2024.

[34] Affected Processors: Transient Execution Attacks & Related Security... https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/processors-affected-consolidated-product-cpu-model.html. Last accessed on 04.03.2024.

[35] James Bottomley. Deploying Encrypted Images for Confidential Computing. https://blog.hansenpartnership.com/deploying-encrypted-images-for-confidential-computing/. Last accessed on 14.05.2024.

[36] Weijie Liu, Wenhao Wang, Hongbo Chen, XiaoFeng Wang, Yaosong Lu, Kai Chen, Xinyu Wang, Qintao Shen, Yi Chen, and Haixu Tang. Practical and efficient in-enclave verification of privacy compliance. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 413–425. IEEE, 2021.

[37] apt-secure(8): Archive authentication support for APT. https://manpages.org/apt-secure/8. Last accessed on 06.05.2024.

[38] Securing Linux VM boot with AMD SEV measurement. https://static.sched.com/hosted_files/kvmforum2021/ed/securing-linux-vm-boot-with-amd-sev-measurement.pdf. Last accessed on 08.03.2024.

[39] Azure confidential computing with NVIDIA GPUs for trustworthy AI. https://azure.microsoft.com/en-us/blog/azure-confidential-computing-with-nvidia-gpus-for-trustworthy-ai/. Last accessed on 06.05.2024.

[40] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th annual international conference on mobile systems, applications, and services*, pages 94–108, 2021.

[41] Vikram Narayanan, Claudio Carvalho, Angelo Ruocco, Gheorghe Almasi, James Bottomley, Mengmei Ye, Tobin Feldman-Fitzthum, Daniele Buono, Hubertus Franke, and Anton Burtsev. Remote attestation of confidential vms using ephemeral vtpms. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23. Association for Computing Machinery, 2023.

[42] Confidential Computing on NVIDIA H100 GPUs for Secure and Trustworthy AI — NVIDIA Technical Blog. https://developer.nvidia.com/blog/confidential-computing-on-h100-gpus-for-secure-and-trustworthy-ai/. Last accessed on 04.03.2024.

[43] Confidential Compute on NVIDIA Hopper H100. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1.0.pdf. Last accessed on 04.03.2024.

[44] Discussion: Potential vulnerability for PKU feature #1351. https://github.com/occlum/occlum/issues/1351. Last accessed on 05.03.2024.

[45] Nokia-Bell-Labs/tee-duet. https://github.com/Nokia-Bell-Labs/tee-duet. Last accessed on 14.05.2024.

[46] Ronald Perez, Reiner Sailer, Leendert van Doorn, et al. vtpm: virtualizing the trusted platform module. In *Proc. 15th Conf. on USENIX Security Symposium*, 2006.

[47] SCONE Configuration and Attestation Service. https://sconedocs.github.io/CASOverview/. Last accessed on 06.05.2024.

[48] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 955–970, 2020.

[49] Chia-Che Tsai, Donald E Porter, and Mona Vij. {Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX}. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 645–658, 2017.

[50] Confidential Computing — Ubuntu. https://ubuntu.com/confidential-computing. Last accessed on 06.03.2024.

[51] GitHub - virtee/snpguest: A CLI tool for interacting with SEV-SNP guest environment. https://github.com/virtee/snpguest. Last accessed on 04.03.2024.

[52] Chengliang Zhang, Junzhe Xia, Baichen Yang, Huancheng Puyang, Wei Wang, Ruichuan Chen, Istemi Ekin Akkus, Paarijaat Aditya, and Feng Yan. Citadel: Protecting data privacy and model confidentiality for collaborative learning. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 546–561, 2021.