EasyChair Preprint

№ 13813

SVSM-KMS: Safeguarding Keys for Cloud Services with Encrypted Virtualization

Benshan Mei, Wenhao Wang and Dongdai Lin

July 3, 2024

SVSM-KMS: Safeguarding Keys for Cloud Services with Encrypted Virtualization

Benshan Mei[1,2], Wenhao Wang[1,2(✉)], and Dongdai Lin[1,2]

[1] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. In recent years, numerous instances of data breaches have emerged due to the inadvertent or intentional disclosure of cryptographic keys. To address this issue, this paper proposes SVSM-KMS, which utilizes AMD's latest Encrypted Virtualization technology (AMD SEV-SNP) to deliver an efficient and seamless integrated secure key management service. We realized multilayered defense by integrating our mechanism within a privileged layer of a confidential virtual machine (CVM), thereby minimizing the trusted computing base (TCB) to prevent key leakage from compromised CVMs. Notably, we incorporated a zero-copy mechanism between the most privileged service module and the least privileged user applications, eliminating redundant data copies. To facilitate seamless integration, we propose a proxy server for existing cloud services. A prototype of SVSM-KMS has been developed based on the latest AMD SEV-SNP hardware platform. Evaluation results indicate that the performance of the Encrypted Virtualization-empowered SVSM-KMS is on par with Hadoop KMS, highlighting the practicality.

Keywords: Confidential Computing · Trusted Execution Environment · Encrypted Virtualization · Secure VM Service Module · Key Management Systems.

1 Introduction

Cloud computing transforms the way organizations store, process, and share data, offering numerous benefits such as scalability and cost-efficiency. However, it also introduces new security challenges, and data breaches have become a recurring threat. These breaches occur when unauthorized individuals gain access to sensitive data stored in the cloud, potentially leading to severe consequences such as financial loss, reputation damage, and privacy violations. Key management is one of the most critical services on the cloud [21, 26]. If the keys are leaked, all relevant data associated with the keys are at risk of leaking [16,19,27]. A centralized key management system (KMS) is often deployed as an integrated approach for generating, distributing, and managing cryptographic keys for devices and applications [6]. However, the TCB can be significant large in a centralized KMS, leading to a potential single point of failure [26]. Although

a decentralized KMS may be desired, it still incurs substantial costs to maintain coherence among multiple nodes [13, 15].

Motivations. Secure key management in the cloud presents several challenges that need to be addressed. One major concern revolves around the cloud service provider's extensive control over the platform, potentially leading to data inspection for their own business interests or due to insider threats. Furthermore, network latency in a centralized KMS can greatly impede service efficiency. Moreover, it can lead to key exposure if the hosting system is compromised.

To tackle these challenges, we focus on a defense-in-depth design, safeguarding sensitive keys in the cloud from untrusted cloud service providers (CSPs) and guest OS. Specifically, we utilize CVM to securely host our system. The CVM is isolated from each other and the hypervisor with hardware-based memory encryption. This ensures that the data remains confidential even if the CSP is untrustworthy. However, it is not sufficient for secure key management due to the large TCB introduced by the guest OS. The key still get leaked if the guest OS gets compromised. We observe that the recently proposed Virtual Machine Privilege Level (VMPL) hardware mechanism can be very applicable to further secure the keys used for encryption and decryption. We safeguard the keys from unauthorized access or tampering by integrating the system within the highest VMPL of a CVM. This ensures that even if the guest OS is compromised, the cryptography keys remain secure. In addition to these security measures, we devote to ensuring seamless integration with existing systems on the cloud, thereby minimizing the deployment effort.

Design. To achieve this, we place the key management service within the highest VMPL over the guest OS. Our design consists of the following aspects: access control, sealed storage, attestation, configuration and seamless integration. The majority of the components, including the access control, are contained within the service module, which runs in the highest VMPL. This means that even if the guest OS is compromised, the keys will remain secure. In order to facilitate seamless integration, we design a proxy server for Hadoop clients, which forwards requests and responses and does not interfere with access control. It consults the service module for token authentication, and securely persists the in-memory storage with sealing service provided by the service module. Unlike a decentralized KMS [13, 15], we do not need to maintain distributed consistency, reducing data synchronization overhead. The performance is further improved by a zero-copy design between the proxy server and the service module, reducing extra data copies between those components.

To showcase the performance of our design, we have developed a prototype based on the latest AMD SEV-SNP platform. It consists of about 7000 lines of codes in total, and is composed of a service module, a kernel module and a proxy server. The evaluation consists of requesting service from KMS deployed on local and remote machine. Our experimental results demonstrate that our approach offers comparable performance to existing Hadoop KMS in both local and remote service scenarios. In the local service scenario, our KMS outperforms Hadoop KMS in term of service latency on most operations. While in serving Transparent

Data Encryption (TDE) of Hadoop Distributed File System (HDFS), our system performs comparably in read and write test.

Contributions. The contributions of the paper are summarized as follows.

- We introduce multilayered defense for secure key management based on the latest feature of Encrypted Virtualization, preventing key leakage from vulnerable guest OS within CVM.
- We incorporate a zero-copy design in our system to improve the performance, allowing efficient service delivery.
- We introduce a proxy server to enable seamless integration, ensuring compatibility with traditional KMS.
- We implement a prototype of the design based on the latest linux-svsm project, and evaluate the performance of our system under realistic scenarios, showing the practical aspect of our design.

2 Background

2.1 AMD SEV

Over the past few years, establishing mutual trust between customers and Cloud Service Providers (CSPs) has been a persistent challenge. It's crucial to mitigate the reliance on CSP trust within the public cloud computing market. AMD's Secure Encrypted Virtualization (SEV) represents a significant leap forward, leveraging hardware-assisted virtualization technology to address this challenge through memory-encryption enhanced isolation [23, 34]. To counter potential threats from malicious hypervisors, AMD introduced successive advancements like SEV-ES (Encrypted State), which encrypts memory pages and private register contents of Virtual Machines (VMs) using distinct keys [35, 36]. However, a lingering vulnerability persists: the hypervisor retains control over nested paging, potentially allowing SEV VM pages to be mapped to other VMs or even the hypervisor itself [40]. Despite the encryption of VM's private status and pages under separate keys, SEV/SEV-ES lacks integrity protection, leaving room for exploits such as memory replay attacks by the hypervisor.

In 2020, AMD introduced SEV-SNP (Secure Nested Paging), a technology designed to fortify the protection of CVM against malicious hypervisor [41]. In SEV-SNP, a malicious hypervisor cannot map an encrypted physical page to multiple owners. This mechanism is implemented through the use of a Reverse Mapping Table (RMP). The RMP is a metadata table controlled by the AMD Platform Security Processor (AMD PSP). It keeps track of the ownership of every system physical page and specifies read, write, and execution permissions for each VMPL. Physical memory page access is restricted by configuring each page's VMPL in the RMP. On each nested-page table walk, the RMP is consulted to determine permission and ownership for each system physical memory page. A nested page fault (#NPF) is raised upon illegal access to a physical page, which can be captured and handled by the hypervisor. The hypervisor manages the VM Saved Areas (VMSAs) assigned to four VMPLs. With hypervisor assistance, the

vCPU can operate in different VMPL by switching the corresponding VMSAs. A higher VMPL context can be seen as the secure world in the ARM TrustZone [39].

SVSM. The Secure Virtual-Machine Service Module (SVSM) is a newly proposed framework that aims at providing essential security services for the guest OS [10]. Both the SVSM and the guest OS share the guest physical address space. However, the SVSM's address space, with a higher VMPL, is inaccessible to a guest OS with a lower VMPL. The SVSM, operating in VMPL0, launches before the BIOS and guest OS in VMPL1. It occupies a fixed number of contiguous physical memory pages before transferring control flow to the BIOS and guest OS. These occupied memory pages remain unused by the guest OS. A dedicated secret page shared with the guest OS allows the guest OS to discover and request the service from SVSM.

The transitions between the guest OS and SVSM are handled by the hypervisor, while the communication protocols between the VM and hypervisor, e.g. passing parameters, and negotiating the shared memory pages, are specified in the Guest-Host Communication Block (GHCB) protocol [11]. The GHCB is a per-cpu guest physical page shared with the hypervisor. It is marked as unencrypted by clearing the C-bit of the page table entry of the guest. The communication between the VM and hypervisor is mediated through a new exception called VM Communication Exception (#VC). This exception can be triggered by Non-Automatic Exit (NAE) events.

A user application can request a service from SVSM by triggering a hypervisor trap through the VMGEXIT NAE event within the VM. Subsequently, the hypervisor schedules the SVSM. Alternatively, the guest OS can also request an SVSM service via the Model Specific Register (MSR) protocol. By creating a request in the GHCB MSR, the guest OS can ask the hypervisor to schedule the SVSM. In both scenarios, the Calling Area (CA) is used to pass the protocol version and call ID to the SVSM. The CA is a shared memory space, and its guest physical address is established through negotiation between SVSM and guest OS. Upon completion of the SVSM service routine, the hypervisor schedules the guest OS once again in accordance with the GHCB protocol.

2.2 Key Management System (KMS)

The KMS is an integrated approach for generating, distributing, and managing cryptographic keys for devices and applications [6]. It consists of various components and plays a vital role in securely managing cryptographic keys and secrets. It typically includes functionalities such as key's generation, storage, rollover and access control. The components of a KMS may include key servers, cryptographic hardware modules, APIs and management interfaces. KMS is essential for protecting sensitive information, ensuring secure communication, enabling data encryption, and meeting compliance requirements [32]. By providing centralized and controlled management of keys, KMS helps organizations maintain the confidentiality, integrity and availability of their data and systems, serving as a crucial foundation for secure operations and safeguarding against unauthorized access and data breaches.

2.3 Threat Model

Our goal is to develop a seamlessly integrated key protection scheme that operates within the highest VMPL inside a confidential VM empowered by AMD's SEV-SNP technology. Our threat model follows the standard assumptions of confidential computing, where any area outside the protected boundaries of the confidential VM is deemed to be under the control of potential attackers. We assume the framework's implementation follows the GHCB protocol reliably. However, we cannot place trust in the proxy server or kernel module. Even if these components compromised, the protected keys remains secure and inaccessible to unauthorized entities.

Ensuring client authentication is crucial for secure service delivery. In the case of SVSM-KMS, it plays a vital role in protecting the system from unauthorized access by malicious clients. To address this risk, SVSM-KMS only accepts requests from authenticated clients. However, the authentication mechanism is not included in the current design. By relying on a trusted third-party authentication mechanism, we can focus on building a strong key protection system while leveraging the security measures provided by the authentication service.

In addition, side channels and hardware attacks are out of the scope. It is assumed that the underlying hardware operates as described in the official documentation. The memory encryption and integrity protection are in place, adding an extra layer of security.

3 Design

3.1 Overview

Fig. 1 illustrates the SVSM-KMS framework, comprising of the SVSM-KMS service module, kernel module, and proxy server. The service module resides in the highest VMPL area. Within the guest OS, the kernel module serves as an intermediary between the user application and the service module, managing shared memory and relaying requests from the proxy server. Consequently, user applications can access the SVSM-KMS service through the proxy server.

The service module of SVSM-KMS is integrated into the SVSM framework. Compared to normal KMS, it is running in an area with the highest VMPL, guarded against the tampering of the malicious hypervisor and guest OS with the hardware security mechanism provided from AMD SEV-SNP. Since the transition between the guest OS and SVSM is mediated by the hypervisor, the VM trap/resume events are inevitable, incurring performance impact. It is not easy to deliver service from the most privileged SVSM-KMS to the least privileged user applications. Challenges present in integrating the key management service into the SVSM framework securely and efficiently, which outlined as follows.

Security. To ensure secure key protection throughout the entire life-cycle, the service module runs in the highest VMPL, i.e., VMPL0. The access control should not rely too much on the untrusted guest OS as much as possible. In other words, the authentication and authorization mechanism should be integrated

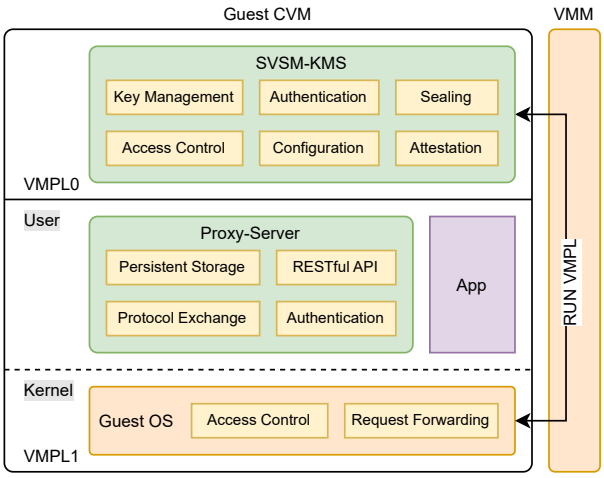

Fig. 1. An overview of the SVSM-KMS framework.

into the highest VMPL. To prevent key leakage from untrusted guest OS, the keys should be sealed outside of the service module as well.

Performance. Considering the underlying SVSM framework, it is crucial to evaluate the performance impact, as each service request incurs at least two VM trap/resume events, potentially increasing service latency. These requests can originate from the kernel or traverse intermediary layers like the proxy server, kernel module, and hypervisor before reaching the service module. Given these indirect layers, we should mitigate the impact of these events and ensure efficient service delivery for user applications.

Compatibility. To enable key management services for various clients, it is necessary to bridge the gap between the SVSM-KMS service protocol and other network protocols. A proxy server is critical for seamless integration of our service with no code change, reducing the deployment effort.

The SVSM-KMS primarily serves the guest OS on the local VM. Therefore, it is not a pressing need to manage a large number of keys. According to the design of the Calling Area (CA) in the SVSM specification, the service request is handled on vCPUs one by one. At most, N requests can be handled in parallel, where N is the number of vCPUs. Due to the lack of high concurrency requirements, a set of shared memory regions is used for each vCPU for simplicity. However, for better availability and scalability needs, we refer to existing methods that can handle unexpected events such as power-offs and shutdowns or managing a significant amount of keys [28, 29].

To facilitate efficient and seamlessly integrated key protection, our design consists of the following aspects: zero-copy mechanism, access control, sealed storage, attestation and seamless integration, which are elaborated in the following subsections. In Section 3.2, we introduce how zero-copy design minimize the performance impact caused by VMPL context switching. The access control is introduced in Section 3.3. Section 3.4 introduces the sealed storage and attes-

tation mechanisms, which establish a foundation of trust. Lastly, a proxy server is introduced in Section 3.5 for seamless integration of existing web services.

3.2 Zero-Copy Design

Between the most privileged service module and the proxy server, user requests go through two rounds of forwarding: first, they're passed through the proxy server to the kernel module, and then forwarded from the kernel to the service module via shared memory. So reducing cross-border data replication is crucial for boosting service efficiency.

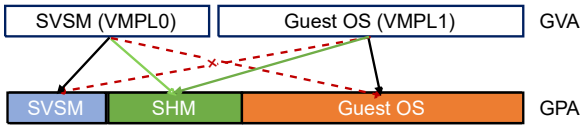

Fig. 2. An overview of the shared and private memory access.

Our system employs a zero-copy mechanism based on shared memory, allowing efficient communication between the most privileged service module and least privileged user applications. To reduce the cost of mapping pages, the guest OS shares memory pages with the SVSM-KMS in the full life-cycle of the service. An overview of the shared memory mechanism is shown in Fig. 2. The SHM is the shared memory. The dashed line with a cross indicates that the private memory space of SVSM can not be accessed by the guest OS. The dashed line without a cross indicates that the memory of the guest OS can be accessed by mapping those memory pages in SVSM on demand. The solid arrow lines show which physical memory space can be accessed by SVSM and guest OS.

The shared-memory regions allocated by the guest OS is mapped both in service module and the guest OS by issuing a shared-memory service request to the service module. A user application can map those shared-memory regions continuously to his virtual memory space. The shared-memory keeps alive until we remove the kernel module. The proxy server manages those regions for each service request from each vCPU, enabling fully utilization of the hardware resources. In this way, a zero-copy mechanism between the user application and the secure service module is realized and the overall communication overhead can be reduced to a minimum.

In addition to the performance benefits, our innovative design also includes security measures for using shared memory. Since the shared memory can only be mapped once by a user application at a time, there is no intermediary who can intercept the communication between the proxy server and the service module. Furthermore, the shared memory is divided into request and response pages. The service module applies VMPL permission to the shared memory, ensuring the confidentiality of the request payload and the integrity of the response. This additional security layer enhances the overall security of the system.

3.3 Authentication & Access Control

To address security concerns, the majority of security measures, including access control, are integrated within the service module itself rather than the web proxy. These mechanisms operate within the highest VMPL layer over the guest OS, rendering them resistant to external circumvention. The access control mechanism is referenced from the Hadoop KMS and encompasses simple token authentication (STA) and access control list (ACL) mechanisms.

Authentication. A user application can request the token authentication service with user group information. The SVSM-KMS grants a legal user an access token signed with a secret. Then a user application can request our service with this token. The secret used to sign the token can be the same as Hadoop KMS for compatibility. A secret obtained inside the SVSM-KMS can also be used for signing for security. As such, the signed token cannot be forged outside of the service module. The algorithm used to sign the token is HMAC(SHA256) for compatibility with Hadoop KMS. Since the token authentication is moved to the SVSM-KMS service module, the proxy server consults the service module for a signed token on each authentication request. The granted token is put into the Cookie on each successful authentication for later user requests.

Access Control List. The ACL mechanism includes system-level and key-level ACLs. Authenticated users have restricted access to specific operations and keys based on the ACL configuration. The service module verifies user permissions for operations or services, and also check permissions for each individual key. The ACL are also maintained within the service module.

3.4 Sealed Storage & Attestation

The AMD PSP exposes the firmware service to the VM through the GHCB protocol. To request an AMD PSP firmware service, the VM issues an SNP Guest Request using the GHCB protocol [12]. The hypervisor mediates the communication between the firmware and the VM, ensuring protection against malicious attackers through AES-GCM authenticated encryption. There are two services can be used in our system, i.e., *key derivation* and *attestation report* services. According to the SEV-SNP firmware ABI specification [12], the guest can request a key from the firmware, derived from a root key. This key can be utilized by the guest for various purposes, including sealing keys or establishing communication with external entities. Meanwhile, the guest can request the firmware to generate an attestation report, which external entities can utilize to verify the guest's identity and security configuration. The guest generates attestation reports for VMPLs that meet or exceed the current VMPL, with the desired VMPL specified in the request message. Since both the *key derivation* and *attestation report* request parameters include the VMPL level, which must be greater than or equal to the current VMPL. Therefore, we can generate a dedicated export key and authentication report for SVSM-KMS running in the highest VMPL.

Sealed Storage. Concerning secure storage, we integrate a sealing service to facilitate the transfer of sealed data between the service module and local

storage. The data stored in memory can be secure protected outside of the trusted service module for long-term storage. Compared to traditional KMS, our system persistently stores key data to the external world using a sealing key. This sealing key is obtained through a *key derivation* service request to the AMD PSP firmware within the service module and remains inaccessible to external entities, thus ensuring the confidentiality of the sealed data.

The proxy server utilizes SVSM-KMS's sealing service to store its in-memory data persistently. Before starting the web service, the proxy server transmits an unsealing request to upload the sealed data to SVSM-KMS. When the web service is about to stop, a sealing request is dispatched to locally save the sealed data. This guarantees a secure storage of the in-memory data. By proactively invoking the sealing service whenever there are updates to the in-memory keys, the proxy server not only ensures performance but also improves fault tolerance. For future improvements, it may be worth considering the implementation of an incremental backup or synchronization mechanism.

Attestation. The security status of the initial service module image can be measured and attested by remote attestation mechanism. The SVSM Attestation Protocol [10] is specified as a standard way to obtain an attestation report of the SVSM. By sending an *attestation report* service request to the firmware, the report for the service module can be obtained, enabling the attestation of its security state. As a result, a separate design for this service is unnecessary.

3.5 Seamless Integration

To enable seamless integration of our system into actual applications, we introduce a proxy server tailored for Hadoop applications, such as Hadoop KMS and DFS clients. In fact, other applications can also be supported by integrating the Key Management Interoperability Protocol (KMIP) [7]. Its main functionality is to redirect user requests to the secure SVSM-KMS and transmit the service response to HTTP clients. The proxy server consists of the following components.

Protocol Exchange. Since the service module follows standard SVSM protocol, protocol exchange is require for the proxy server to request service from the service module. The request and response of the service module are packed into an appropriate form before forwarding, i.e., JSON. Compared to standard JSON-RPC protocol [5], the request and response consist of message *header* and *body*. The *header* is transferred through general registers, while the *body* is stored in the shared memory. The user request is routed to the corresponding service handler according to the message header. The message body are in the form of JSON for simplicity, except for the sealing service where encrypted data is transferred between local and in-memory storage. The status code from the service module is mapped to the corresponding HTTP status code. In this way, we avoid parsing the message header and simplify the protocol design.

RESTful API. The RESTful API is based on the Hadoop KMS RESTful API[3] for compatibility, allowing our SVSM-KMS to support various Hadoop ap-

[3] https://hadoop.apache.org/docs/current/hadoop-kms/index.html

plications, especially the HDFS. The Hadoop KMS is used to assign encryption zone keys (EZK) to encryption zones automatically for transparently encrypting/decrypting data stored in HDFS [28]. Here, we demonstrate the seamless substitution of Hadoop-KMS with SVSM-KMS in the TDE of HDFS[4], showcasing the flexibility of our solution within existing Hadoop infrastructures.

Configuration. The configuration of the service module can be modified via the configuration service, allowing for the import of various configurations. We support the Hadoop KMS configuration format for compatibility. Presently, our system solely permits updating ACL and log configurations, along with the ability to adjust the log level for debugging purposes. Additionally, this service also enables the persistence of in-memory configurations, sealed locally with secure authenticated encryption.

4 Implementation

To showcase our design, we developed a prototype based on the infrastructure available from AMD[5]. Our project comprises three parts: the service module (over 4000 lines of Rust code), the kernel module (around 1000 lines of C code), and the proxy server (about 2000 lines of C++ code). The implementation details are elaborated in the following sections.

4.1 SVSM-KMS Service Module

Presently, we have developed the service module based on the linux-svsm project [6]. The service module provides functionalities encompassing key management, authentication, configuration, sealing, and more. Additionally, the recently open-sourced coconut-svsm project [7] offers an alternative infrastructure for deploying our system within CVM. Both frameworks are written in Rust, ensuring memory safety. The Rust language ecosystem also greatly facilitates the development of SVSM-KMS with various crates for encryption and persistent storage.

Key Management. We provide key management services for Hadoop KMS Client and HDFS. These services primarily involve managing key metadata and performing tasks like creating, updating, encrypting, and decrypting key versions. Each service has a status code that complies with the official SVSM specification. The requests for these services originate from the proxy server and are indirectly derived from the HTTP clients. These core services are implemented within the service module as service handlers, offering support for managing AES and SM4 encryption keys. To ensure secure generation of encryption keys and initialization vectors (IV), the *rdseed* instruction is utilized to retrieve hardware-generated random values, which are then used to initialize the *ChaCha* Random

[4] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/TransparentEncryption.html
[5] https://github.com/AMDESE
[6] https://github.com/AMDESE/linux-svsm/,commitID:2ea8eeab
[7] https://github.com/coconut-svsm

Number Generator (RnG). By leveraging the hardware-provided RnG, the system ensures robust and secure generation of cryptographic material.

Configuration & Sealing. The configuration parameters for the service module are primarily obtained from the configuration files of Hadoop KMS. The ACL configuration from Hadoop KMS is also utilized in our service module. Furthermore, the sealing service within our implementation employs AES-GCM encryption for sealing operations.

Compared to Hadoop KMS, our implementation does not require frequent local storage access. All requests are managed using in-memory storage and operations. As a prototype, the in-memory data is organized in a HashMap data structure. Moreover, our service module runs directly in the VM, while Hadoop KMS runs in a JVM (Java VM). This significantly reduces the impact on performance. Meanwhile, we have not integrated certain important security mechanisms into our current system, such as Kerberos Authentication and Audit Logs supported in the Hadoop KMS. Since they are not the main focus of our current work and have little impact on performance evaluation, they can be integrated in the future. Furthermore, as in-memory storage space is limited, a swap-memory mechanism can be introduced as a future enhancement.

4.2 SVSM-KMS Kernel Module

Currently, there is no official example of calling an SVSM service in user-space. Here, we implement a custom kernel module for requesting the service. This kernel module is a *misc* device driver, exposing a /dev/svsm device file in the file system. It also provides *mmap* and *ioctl* interfaces to user applications, allowing service request through the *ioctl* syscall. The kernel module then forwards these user requests directly to the service module using the GHCB protocol. To prevent invalid and illegal service requests from being forwarded to the service module, a simple service filtering mechanism is introduced within the forwarding logic.

The request and response headers of the SVSM-KMS include the value of five general registers. These registers are used by the SVSM Core Protocol as well. The user applications wrap the value of these registers into a structure. The kernel module then extracts those values from the structure to the corresponding registers before issuing an GHCB Protocol request for scheduling the service within VMPL0. After the completion of the service, the kernel module copies those values back to the user-space so that the user application can receive a complete response from the service module.

Following above design, the kernel module does nothing but prepare shared memory and forward user request and service response across the whole lifetime. Such flexible design allows providing more security services for the user applications without major changes to the kernel module, ensuring extensibility.

4.3 SVSM-KMS Proxy Server

The proxy server is implemented with the open-sourced libhv framework[8]. It maps the shared memory to the private virtual memory space with the *mmap* syscall and forwards user requests to the service module with the *ioctl* syscall. It is capable of uploading configurations to SVSM-KMS, supporting the Hadoop KMS configuration format for compatibility. Prior to uploading, the configuration is transformed into JSON format. Afterwards, SVSM-KMS updates the corresponding configurations. Both configurations and sealed data are uploaded to the service module before accepting user requests. By doing so, we ensure the compatibility and seamless integration of a wider range of applications with our SVSM-KMS solution. Currently, two clients have undergone testing, namely the Hadoop KMS Client and HDFS (TDE). These clients can request services successfully via the RESTful API offered by SVSM-KMS.

5 Evaluation

In this section, we evaluate the performance of SVSM-KMS and Hadoop KMS. The tests are performed on a dual-socket 3rd Gen AMD EPYC processor (codenamed Milan) with 128 logical cores and 64GB RAM, supporting the SEV-SNP security feature. The host system operates QEMU 6.1.50 on Ubuntu 22.04 (kernel version 6.1.0-rc4-snp-host), while the VM is allocated 8 vCPUs and 8GB RAM, running Ubuntu 22.04 (kernel version 6.2.0-snp-guest). Throughout the experiments, Hadoop-3.3.4 is deployed on both the VM and a remote machine for evaluation. Simple Token Authentication is configured for both Hadoop KMS and SVSM-KMS. Additionally, the ACL is identical for both KMS.

5.1 Micro-benchmarks

For the micro-benchmarks, we evaluate the service latency of SVSM-KMS and Hadoop-KMS. Both KMS are running on the same VM, with SVSM-KMS on VMPL0 and Hadoop KMS on VMPL1. The experiments are performed on both local and remote scenarios. In total, we evaluate thirteen RESTful APIs, excluding *Invalidate Cache*, which is not supported in our system currently. We make 150 requests to each service and record the average elapsed time for each request. To request the service in both local and remote scenarios, we utilize the popular *requests* Python package.

The latency of the service is evaluated in both local and remote situations. When calling a service of SVSM-KMS, the support of the hypervisor is necessary, thereby inevitably increasing the service latency. However, the service module operates on a real machine, not a JVM like Hadoop KMS. This latency represents the only impact that cannot be reduced through software means. In remote cases, there is network latency. The impact of the SVSM framework can be somewhat neglected. The evaluation is presented in Fig. 3. The terms Hadoop and SVSM

[8] https://github.com/ithewei/libhv

correspond to the implementation of KMS. The L and R denote local and remote scenarios, respectively.

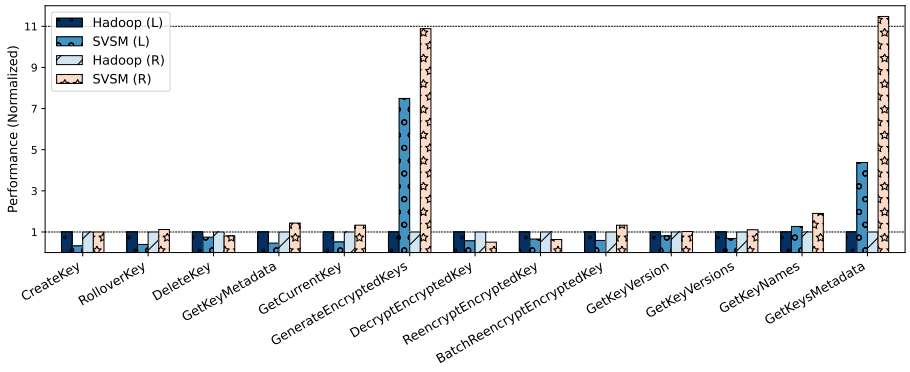

Fig. 3. The service latency of the proposed SVSM-KMS and Hadoop KMS in serving local and remote machines.

While differences exist between SVSM-KMS and Hadoop KMS, it's essential to acknowledge that SVSM-KMS serves as a prototype and does not encompass all the optimizations featured in Hadoop KMS. Nonetheless, our system demonstrates comparable performance to Hadoop KMS in both local and remote service scenarios. Due to implementation issues, it shows significant performance degradation on specific operations, such as GenerateEncryptedKeys, GetKeyNames, and GetKeysMetadata in the remote scenario. It is noteworthy that these operations are infrequently utilized in a deployed cloud platform. Therefore, their impact on cloud service performance is negligible. Conversely, SVSM-KMS holds a notable advantage in the DecryptEncryptedKey operation, serving as a hot function in HDFS (TDE).

The SVSM-KMS outperform Hadoop KMS in local service scenarios due to the zero-copy design, which helps mitigate the performance impact of the SVSM framework. While not incorporating all optimizations, SVSM-KMS show the potential of providing efficient and secure key protection services within CVM. By integrating key-cache and warm-up mechanisms into SVSM-KMS, further improvements in performance can be achieved, highlighting the scalability and extensibility of our design.

5.2 Macro-benchmarks

We perform read and write operations on an encrypted zone of HDFS ten times, with each operation resulting in a service request to the KMS. The average elapsed time of each operation is recorded. To evaluate the performance impact on HDFS, file sizes range from 32KB to 4MB. The experiments consists of both local and remote scenarios. In the local scenario, the HDFS and KMS

are deployed in the same VM. Conversely, in the remote scenario, the HDFS is deployed on another machine, sending service requests to a local or remote deployed KMS. The evaluation on HDFS (TDE) is illustrated in Fig. 4.

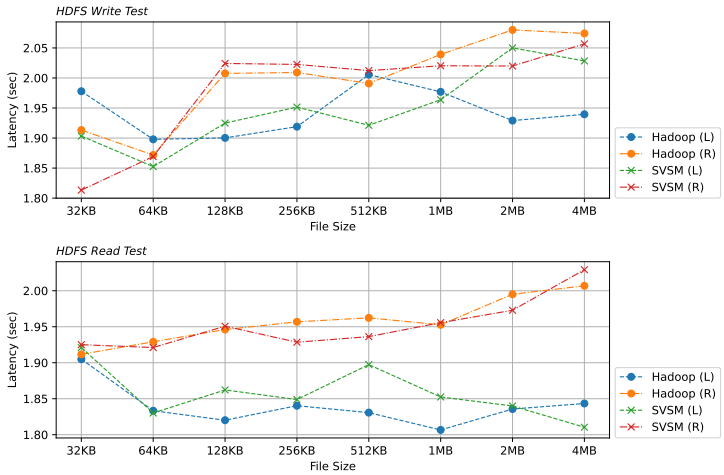

Fig. 4. The service latency (seconds) evaluation between SVSM-KMS and Hadoop-KMS in serving HDFS (TDE) under local and remote scenarios.

Fig. 4 shows the average time consumption of read and write operations in different scenarios. Since performing read and write operations in the encryption zone is time-consuming, the performance impact caused by the Key Management Service (KMS) can be largely ignored. Although each operation requires sending a request to KMS, the average time consumption shows little difference on both SVSM-KMS and Hadoop KMS when serving HDFS (TDE). Whether in local or remote scenarios, as the file size increases, the performance impact on the HDFS (TDE) can be largely ignored. Overall, these results demonstrate the effectiveness and efficiency of our prototype, indicating its potential to provide comparable or better performance than existing solutions, especially in local service scenarios and TDE (HDFS).

6 Related works

Microsoft Azure KeyVault [8], AWS KMS [1], and Google Cloud KMS [2] are cloud-based solutions for centralized management of keys, certificates, and secrets. Cloudera Navigator Key Trustee Server [3] offers key management for securing data in Hadoop, while Ranger KMS provides centralized key management for secure encryption in Apache Hadoop. HashiCorp Vault [4] offers secure storage and dynamic management of secrets. OpenStack Barbican [9] provides key management in cloud environments. While these systems are secure, they

are often deployed in VM where vulnerable guest OS can compromise their security. Their TCB is still too large to hosting such services. Therefore, further privilege separation within CVM is critical to the security of KMS on the cloud.

The Trusted Execution Environment (TEE) can effectively mitigate the risks of key compromise and unauthorized access from the untrusted OS. One advancement in improving KMS security is the STYX framework [44], which incorporates Intel Software Guard Extensions (SGX) [25]. By leveraging the capabilities of Intel SGX, cloud system KMS security is greatly enhanced [20,30], while also considering aspects such as performance and high availability. In [43], a novel technique for key management was presented to enhance security, privacy, and confidentiality. The proposed approach, combining Intel SGX and FIWARE components, is scalable and suitable for safeguarding IoT data. Another notable work proposes a key management solution based on Intel SGX for data-centric networking [38]. MultiSGX-KMS is a decentralized system designed to safeguard user keys during exchange and ensure fault tolerance through secret sharing and Intel SGX [13], which can be leveraged to realize a decentralized design of the SVSM-KMS framework. Researchers also introduced RansomClave, a ransomware family that securely manages cryptographic keys using an enclave [18]. Similarly, TZ-KMS is introduced for joint cloud computing based on ARM TrustZone [33]. Samsung's Keymaster is a key management system for mobile platforms based on ARM TrustZone [42]. In general, TEEs provide a secure and isolated environment where sensitive operations, such as key generation, storage, and cryptographic operations, can be performed with high assurance. However, these Intel SGX and ARM TrustZone based proposals are not multi-layered defense as SVSM-KMS in the realm of confidential computing.

Previous works have surveyed traditional key management across various domains, including the smart grid [22], wireless sensor networks [24,45] in Internet of Things (IoT) scenarios [17], as well as cryptographic methodologies within cloud security contexts [14,19,21,37]. A comparative analysis of cryptographic key management systems is detailed in [31]. Recently, researchers investigated three popular key management services that are currently in use, namely AWS CloudHSM, Keyless SSL, and STYX [26]. However, their threat model clearly contrasts to SVSM-KMS, where the guest OS is considered vulnerable.

7 Conclusion

SVSM-KMS is a secure key protection mechanism that leverages the latest features from AMD SEV-SNP for cloud services. It integrates a series of mechanisms to enhance performance and establishes a layered defense. A zero-copy mechanism is introduced to optimize overall performance, utilizing shared memory throughout the entire service life-cycle to eliminate redundant data transfers between the service module and user applications. To realize multilayered defense, we integrate our core service into the SVSM framework, ensuring that critical key management service operates at the highest VMPL in a CVM, shielded from vulnerable guest OS. The system's security is further fortified through sealed per-

sistent storage and the capability to measure and attest the security state using the standard SVSM Attestation Service. Meanwhile, hardware instructions are leveraged for the secure generation of keys, effectively addressing security and privacy concerns associated with centralized and decentralized KMS. Our solution focuses on facilitating seamless integration by introducing a proxy-server for Hadoop clients, streamlining the deployment process in utilizing our system. Evaluation results demonstrate the performance of our proposed system across local and remote service scenarios, including the compatibility as a drop-in replacement for Hadoop KMS in HDFS (TDE), demonstrating comparable performance across various usage scenarios.

In conclusion, the SVSM framework can be leveraged to provide security critical services (i.e., key management) for a variety of applications. Future works would be the key-cache and warm-up mechanisms for performance. A decentralized design of the SVSM-KMS can be considered for security.

Acknowledgment. This work was supported by National Natural Science Foundation of China (Grant No.62272452). Corresponding author: Wenhao Wang (wangwenhao@iie.ac.cn).

References

1. Amazon Key Management Service | Manage encryption keys | Amazon Web Services. https://www.amazonaws.cn/en/kms/ (2024)
2. Cloud Key Management | Google Cloud. https://cloud.google.com/security-key-management?hl=zh-cn (2024)
3. Cloudera Navigator Key Trustee Server Overview | CDP Private Cloud. https://docs.cloudera.com/cdp-private-cloud-base/7.1.3/security-key-trustee-server/topics/cm-security-kts-overview.html (2024)
4. HashiCorp Vault - Manage Secrets & Protect Sensitive Data. https://www.hashicorp.com/products/vault (2024)
5. Json-rpc 2.0 specification. https://www.jsonrpc.org/specification (2024)
6. Key management. https://en.wikipedia.org/wiki/Key_management (2024)
7. Key management interoperability protocol specification version 2.0. https://docs.oasis-open.org/kmip/kmip-spec/v2.0/os/kmip-spec-v2.0-os.html (2024)
8. Key Vault | Microsoft Azure. https://azure.microsoft.com/en-us/products/key-vault (2024)
9. OpenStack Key Manager (barbican) — Barbican 16.1.0.dev14 documentation. https://docs.openstack.org/barbican/latest (2024)
10. Secure vm service module for sev-snp guests. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/58019.pdf (2024)
11. Sev-es guest-hypervisor communication block standardization. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/56421.pdf (2024)
12. Sev secure nested paging firmware abi specification. https://www.amd.com/system/files/TechDocs/56860.pdf (2024)

13. Abdulsalam, Y.S., Bouamama, J., Benkaouz, Y., Hedabou, M.: Decentralized sgx-based cloud key management. In: International Conference on Network and System Security. pp. 327–341. Springer (2023)
14. AlBelooshi, B., Damiani, E., Salah, K., Martin, T.: Securing cryptographic keys in the cloud: A survey. IEEE Cloud Computing **3**(4), 42–56 (2016)
15. An, H., Choi, R., Kim, K.: Blockchain-based decentralized key management system with quantum resistance. In: Information Security Applications: 19th International Conference, WISA 2018, Jeju Island, Korea, August 23–25, 2018, Revised Selected Papers 19. pp. 229–240. Springer (2019)
16. Arul Oli, S., Arockiam, L.: A framework for key management for data confidentiality in cloud environment. In: 2015 International Conference on Computer Communication and Informatics (ICCCI). pp. 1–4 (2015)
17. Bartsch, W., Gope, P., Kavun, E.B., Millwood, O., Panchenko, A., Pasikhani, A.M., Polian, I.: Design rationale for symbiotically secure key management systems in iot and beyond. In: ICISSP. pp. 583–591 (2023)
18. Bhudia, A., O'Keeffe, D., Sgandurra, D., Hurley-Smith, D.: Ransomclave: ransomware key management using sgx. In: Proceedings of the 16th International Conference on Availability, Reliability and Security. pp. 1–10 (2021)
19. Buchade, A.R., Ingle, R.: Key management for cloud data storage: methods and comparisons. In: 2014 Fourth International Conference on Advanced Computing & Communication Technologies. pp. 263–270. IEEE (2014)
20. Chakrabarti, S., Baker, B., Vij, M.: Intel sgx enabled key manager service with openstack barbican. arXiv preprint arXiv:1712.07694 (2017)
21. Chandramouli, R., Iorga, M., Chokhani, S.: Cryptographic key management issues and challenges in cloud services. Secure Cloud Computing pp. 1–30 (2013)
22. Ghosal, A., Conti, M.: Key management systems for smart grid advanced metering infrastructure: A survey. IEEE Communications Surveys & Tutorials **21**(3), 2831–2848 (2019)
23. Hashimoto, M.: Overview of memory security technologies. In: 2021 International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA). pp. 1–2 (2021)
24. He, X., Niedermeier, M., De Meer, H.: Dynamic key management in wireless sensor networks: A survey. Journal of network and computer applications **36**(2), 611–622 (2013)
25. Hu, X., Li, J., Wei, C., Li, W., Zeng, X., Yu, P., Guan, H.: Styx: A hierarchical key management system for elastic content delivery networks on public clouds. IEEE Transactions on Dependable and Secure Computing **18**(2), 843–857 (2021)
26. Huang, X., Chen, R.: A survey of key management service in cloud. In: 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS). pp. 916–919. IEEE (2018)
27. Jang-Jaccard, J., Manraj, A., Nepal, S.: Portable key management service for cloud storage. In: 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom). pp. 147–156. IEEE (2012)
28. Jin, W., Geng, K., Yu, M., Guo, Y., Li, F.: Efficiently managing large-scale keys in hdfs. In: 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys). pp. 353–360. IEEE (2021)
29. Karande, V., Bauman, E., Lin, Z., Khan, L.: Sgx-log: Securing system logs with sgx. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 19–30 (2017)

30. Kuan, S.: Improving the Security of KMS on a Cloud Platform Using Trusted Hardware. Master's thesis, Aalto University. School of Science (2018)
31. Kuzminykh, I., Ghita, B., Shiaeles, S.: Comparative analysis of cryptographic key management systems. In: International Conference on Next Generation Wired/Wireless Networking. pp. 80–94. Springer (2020)
32. Lei, S., Zishan, D., Jindi, G.: Research on key management infrastructure in cloud computing environment. In: 2010 Ninth International Conference on Grid and Cloud Computing. pp. 404–407. IEEE (2010)
33. Luo, S., Hua, Z., Xia, Y.: Tz-kms: A secure key management service for joint cloud computing with arm trustzone. In: 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE). pp. 180–185. IEEE (2018)
34. Mattioli, M.: Rome to milan, amd continues its tour of italy. IEEE Micro **41**(4), 78–83 (2021)
35. Mofrad, S., Zhang, F., Lu, S., Shi, W.: 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, Association for Computing Machinery, New York, NY, USA (2018)
36. Ning, Z., Zhang, F., Shi, W., Shi, W.: Position paper: Challenges towards securing hardware-assisted execution environments. In: Proceedings of the Hardware and Architectural Support for Security and Privacy. HASP '17, Association for Computing Machinery, New York, NY, USA (2017)
37. Oruganti, R., Churi, P.: Systematic survey on cryptographic methods used for key management in cloud computing. In: International Conference on Innovative Computing and Communications: Proceedings of ICICC 2021, Volume 2. pp. 445–460. Springer (2022)
38. Park, M., Kim, J., Kim, Y., Cho, E., Park, S., Sohn, S., Kang, M., Kwon, T.T.: An sgx-based key management framework for data centric networking. In: International Workshop on Information Security Applications. pp. 370–382 (2019)
39. Pinto, S., Santos, N.: Demystifying arm trustzone: A comprehensive survey. ACM computing surveys (CSUR) **51**(6), 1–36 (2019)
40. Qin, H., Song, Z., Zhang, W., Huang, S., Yao, W., Liu, G., Jia, X., Du, H.: Protecting encrypted virtual machines from nested page fault controlled channel. In: Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy. pp. 165–175 (2023)
41. Sev-Snp, A.: Strengthening vm isolation with integrity protection and more. White Paper, January p. 8 (2020)
42. Shakevsky, A., Ronen, E., Wool, A.: Trust dies in darkness: Shedding light on samsung's {TrustZone} keymaster design. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 251–268 (2022)
43. Valadares, D.C.G., da Silva, M.S.L., Brito, A.E.M., Salvador, E.M.: Achieving data dissemination with security using fiware and intel software guard extensions (sgx). In: 2018 IEEE Symposium on Computers and Communications (ISCC). pp. 1–7. IEEE (2018)
44. Wei, C., Li, J., Li, W., Yu, P., Guan, H.: Styx: a trusted and accelerated hierarchical ssl key management and distribution system for cloud based cdn application. In: Proceedings of the 2017 Symposium on Cloud Computing. pp. 201–213 (2017)
45. Xiao, Y., Rayi, V.K., Sun, B., Du, X., Hu, F., Galloway, M.: A survey of key management schemes in wireless sensor networks. Computer communications **30**(11-12), 2314–2341 (2007)