electronics

MDPI

Article

Secure Multiparty Computation Using Secure Virtual Machines

Danko Miladinović *, Adrian Milaković, Maja Vukasović, Žarko Stanisavljević and Pavle Vuletić

School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11000 Belgrade, Serbia; aki@etf.bg.ac.rs (A.M.); majav@etf.bg.ac.rs (M.V.); zarko@etf.bg.ac.rs (Ž.S.); pavle.vuletic@etf.bg.ac.rs (P.V.)
* Correspondence: danko.miladinovic@etf.bg.ac.rs

Abstract: The development of new processor capabilities which enable hardware-based memory encryption, capable of isolating and encrypting application code and data in memory, have led to the rise of confidential computing techniques that protect data when processed on untrusted computing resources (e.g., cloud). Before confidential computing technologies, applications that needed data-in-use protection, like outsourced or secure multiparty computation, used purely cryptographic techniques, which had a large negative impact on the processing performance. Processing data in trusted enclaves protected by confidential computing technologies promises to protect data-in-use while possessing a negligible performance penalty. In this paper, we have analyzed the state-of-the-art in the field of confidential computing and present a Confidential Computing System for Artificial Intelligence (CoCoS.ai), a system for secure multiparty computation, which uses virtual machine-based trusted execution environments (in this case, AMD Secure Encrypted Virtualization (SEV)). The security of the proposed solution, as well as its performance, have been formally analyzed and measured. The paper reveals many gaps not reported previously that still exist in the current confidential computing solutions for the secure multiparty computation use case, especially in the processes of creating new secure virtual machines and their attestation, which are tailored for single-user use cases.

Keywords: trusted execution environments; secure multiparty computation; secure virtual machine; remote attestation; confidential computing

Citation: Miladinović, D.; Milaković, A.; Vukasović, M.; Stanisavljević, Ž.; Vuletić, P. Secure Multiparty Computation Using Secure Virtual Machines. *Electronics* **2024**, *13*, 991. https://doi.org/10.3390/electronics13050991

Academic Editor: Cheonshik Kim

Received: 31 January 2024
Revised: 25 February 2024
Accepted: 27 February 2024
Published: 5 March 2024

1. Introduction

The use and development of applications whose operations rely on computing and processing various datasets have grown significantly over the last decade, induced especially by advances in artificial intelligence and machine learning. While, on one hand, the useful features and potential of these applications are undoubtful, data processing is a sensitive issue because the data that are processed are often considered either personally identifiable or confidential information. Processing such data requires careful security protection either because of the strict legal regulations (e.g., European Union's General Data Protection Regulation (GDPR)) or to protect the business interest of the data owners.

Secure processing or performing secure computation means performing a computation on some data while that data remains secret to unauthorized actors. If the computation is conducted locally on a single computer or within the infrastructure of a single company without any data transfers outside, there is no need to additionally secure the data as long as the physical and network access to the computing infrastructure are secured and allowed only to the authorized actors. However, there are many cases in which the data owner does not have the adequate capacity to process the data, which led to the rise of cloud computing [1]. Also, there are situations when there is a need to conduct the computation over merged datasets of multiple data owners to achieve some common benefit. In both cases, data owners might want or have to preserve the privacy of their data. While the methods for protecting data at rest (on some kind of storage) and data in transit (during an exchange over an untrusted communication channel) have been known for a long time

using traditional encryption techniques and protocols, efficiently protecting data in use and providing performant secure computations has remained an elusive goal for a long time.

1.1. Outsourced and Secure Multiparty Computation

There are two similar models of secure and verifiable computation [2]: outsourced and secure multiparty computation (SMC). With outsourced computation, a single data owner sends the data in the encrypted form to the other party, who performs the computation on the encrypted data and sends the encrypted result back to the data owner, without being able to have an insight into the raw data [3]. In the case of SMC, a group of data owners want to perform some computation over their joint datasets without revealing the raw data to any of the parties.

1.1.1. Homomorphic Encryption

The initial research of both models was in the area of cryptographic algorithms, which are capable of securing data processing on untrusted hardware. The key enabler for the outsourced computation is homomorphic encryption, which allows computation over the encrypted data. On the other hand, secure multiparty computation can be achieved using mechanisms like garbled circuits with the oblivious transfer, secret-sharing, the extension of homomorphic encryption to the multi-user case, or functional encryption. The key issue of all of the previously mentioned mechanisms is the data processing performance. Both outsourced and SMC using these mechanisms are by several orders of magnitude slower compared to non-protected data processing on regular hardware. Even the latest reported hardware- and software-accelerated secure multiparty computation systems based on garbled circuits [4] are slower by 2–4 orders of magnitude than general purpose processors in performing some specific computations, like dot product or gradient descent. Similar results are obtained for homomorphic encryption [4], which, in addition, has other implementation issues, making secure computations difficult: noise growth, limits of the range of the numbers used in computation, and a limited set of supported mathematical operations requiring the changes in the computation algorithms. This renders the previously mentioned algorithms an expensive and far from optimal solution for large-scale and big data secure data processing.

1.1.2. Federated Learning

Another privacy-preserving computation approach tailored specifically for machine learning is federated learning. With federated learning, many clients collaboratively and independently train a model under the orchestration of a central server. Parts of the model are trained locally by clients without any privacy protection because the data do not leave the client's devices. The clients exchange a minimal amount of information (e.g., intermediate results or model parameters) needed to fulfill the machine learning task, while the raw data remains fully decentralized. However, one of the greatest challenges for federated learning remains reaching the accuracy of centralized machine learning performed over the whole dataset gathered from all the clients, in cases when the clients' data are not independent and identically distributed (IID). There is also a tension between data privacy and robustness (reliable results in cases of malicious clients who tend to poison the models) [5], which might be critical in the case of sensitive data analysis (e.g., medical data), which requires both strict privacy and very reliable results.

1.1.3. Trusted Execution Environments

The other branch of secure computation and processing development is in the area of confidential computing, which uses new processor capabilities that enable secure execution and data-in-use protection. Trusted Execution Environments (TEE) or secure enclaves isolate, and with some technologies encrypt, the data in memory during the processing process. There are different approaches, supported by different hardware solutions [6]. ARM TrustZone isolates the critical security firmware, assets, and private information from

the rest of the applications while using the full processing power of the main cores. It only separates secure from non-secure applications while not providing any additional privacy protection using cryptographic algorithms. The other approach is to isolate and encrypt the portions of the code that process security-sensitive data in an application. The data are encrypted and decrypted using well-known symmetric cryptographic algorithms on the fly (before being written to the system memory and upon reading from it) using a hardware-based encryption engine, which resides on the CPU, and using cryptographic keys, which cannot be exported from it. This approach was introduced by Intel Software Guard Extensions (SGX) in 2015. A slightly different strategy is to isolate and encrypt the memory of the entire virtual machine (VM). This approach is called Secure Encrypted Virtualization (SEV) [7], introduced by AMD in 2016. It has been improved twice so far (in 2017 SEV-ES (Encrypted State) [8] and in 2020 SEV-SNP (Secure Nested Paging)) [9]. At the beginning of 2023, with the 4th generation of Xeon processors, Intel released Trust Domain Extensions (TDX), technology that, similarly to the AMD SEV, allows for deploying hardware-isolated, virtual machines, called trust domains, using memory encryption using AES-128-XTS and integrity using 28-bit MAC [10]. This latest development suggests that key processor manufacturers have converged towards the strategy to allow full virtual machine isolation and memory encryption.

Figure 1 shows a high-level architecture of an AMD-SEV virtual machine memory encryption. AMD EPYC processors contain a separate ARM-based Platform Secure Processor (PSP), which creates and stores symmetric cryptographic keys. Whenever the data are being written to or read from the memory of a secure virtual machine (SVM), PSP intercepts the data and encrypts or decrypts it, respectively. Figure 1 shows which parts of computing resources on a remote server can be trusted by the AMD SEV virtual machine owner (depicted in green) and which are not trusted (red). Enabling secure processing in such a hostile environment requires careful SVM installation and a process to verify the installation (called attestation), which are described in this paper in detail.

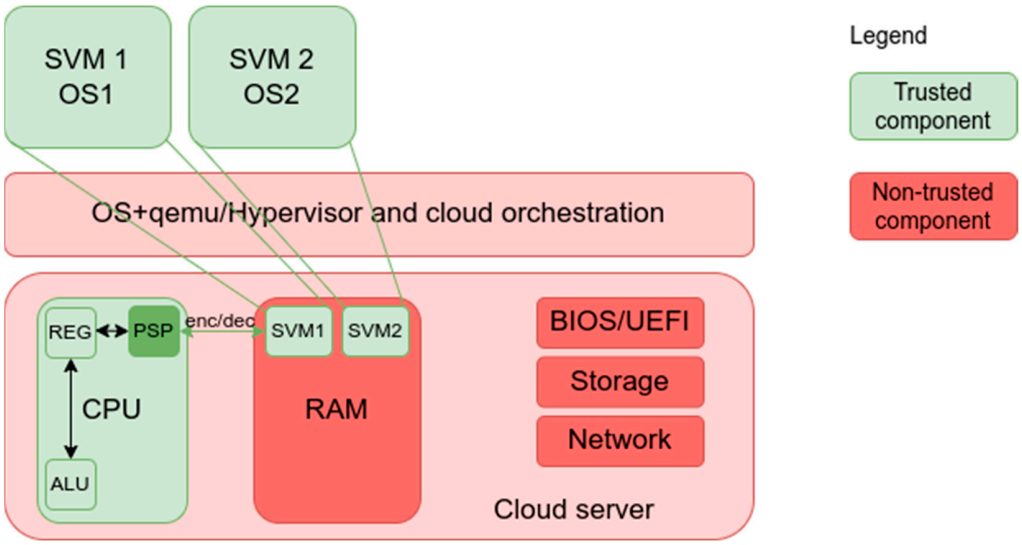

Figure 1. High-level architecture of the AMD SEV VM protection.

The performance penalty of the on-the-fly data encryption and decryption during memory writes and reads in the TEE is on the order of a few percent compared to processing without the encryption turned on [11]. This development provoked the second branch of research on protecting data in use and the rise of confidential computing—the use of TEE in protecting the data that are processed on untrusted hardware. Confidential computing specifically targets outsourced computing in a cloud environments use case, in which the hardware is not controlled by its user, but rather by cloud providers. Data privacy was for a long time one of the key blockers of wider cloud service deployment. Without additional

protection, malicious cloud administrators and owners can access the resources used by the user and compromise the data. While the user can secure the data that are on cloud disks by encrypting it using the user's keys, the data have to be decrypted to be processed. This is the moment in which a malicious cloud provider can access both the user's encryption key and all of the data by simply dumping the RAM content. Using a hardware-based TEE can prevent such scenarios by encrypting all of the user's sensitive information in a TEE. The data in server RAM are encrypted using cryptographic keys, which are stored and protected on the CPU. If a server administrator tries to perform a memory dump, this time it will not be possible to reveal the content of the user's data. Since cryptographic keys for memory encryption are stored on the CPU, distributed processing even on a single multi-processor computer is a non-trivial problem, yet to be solved.

1.2. Motivation and Contributions

The possibility to execute any code and computation without any restrictions and process the whole datasets on untrusted hardware without revealing the data content, with a performance penalty which is negligible compared to the purely cryptographic solutions mentioned above, makes TEE a perfect candidate for outsourced and secure multiparty computations. Several attempts in the research literature describe the use of SGX-based enclaves for SMC, as will be reported in the next section. However, to the best of our knowledge, there have been no studies so far that analyze the use of virtual machine-based enclaves (using SEV or TDX) for secure multiparty computation. Like any other technology, virtual machine-based enclaves come with their own set of challenges, and despite the obvious benefits they promise, the implementation for the SMC use case is far from straightforward. They have a larger trusted compute base (TCB), specific remote attestation procedures which are not suitable for the SMC, and specific programming rules that have to be followed to provide completely secure data processing. The main contributions of this paper are: (1) the analysis of the TEE technologies that enable secure virtual machines, their current status, and their suitability for the SMC; (2) the identification of the key gaps that exist at the moment of writing this paper; (3) the proposal and description of a new protocol and a novel Confidential Computing System for Artificial Intelligence (CoCoS.ai) that enables secure multiparty computation in semi-honest settings; (4) security analysis of the proposed solution; and (5) performance analysis of the proposed solution.

The rest of the paper is organized as follows. Section 2 provides background, gives an overview of related work, and enlists and discusses the limitations of the study. Section 3 describes the proposed CoCoS.ai system. Section 4 provides implementation details by presenting the proof of concept that was developed. Section 5 concludes the paper and presents open questions and future research directions.

2. Background and Related Work

This section gives an overview of the related research work in the fields of secure multiparty computation, trusted execution environments, and especially SMC, realized using TEE. The overview of the current limitations concludes the section.

2.1. Secure Multiparty Computation

Secure Multiparty Computation allows multiple actors to compute a joint function of their private inputs while revealing only the computation output. One of the roles in SMC is the *input party*—a party that provides protected data for the computation. *Computing parties* carry out the computation and the SMC protocol. Finally, the *result parties* obtain the results of the computation. One party can have multiple roles, depending on the SMC scenario [12]. For example, in the case of a machine learning-based medical diagnosis system trained on multiple hospital data, there are many input parties (hospitals that provide their encrypted patient data for system training), and there may be many result parties that overlap with the input parties (doctors from any of the participating hospitals who want to learn the

estimated diagnoses of their patients). On the other hand, in the case of a regulatory body that checks the financial health reports of some consortium members without revealing the raw data, there are many input parties (institutions whose financial health is checked) and one result party (regulatory body), which is not among the input parties [13].

Lindell recently gave an overview of the key definitions, challenges, research directions, and current status of cryptography-based SMC protocols [14]. We adopt a few key definitions from that paper which are important for further discussion. The security properties of the SMC system are *privacy*, *correctness*, *independence of inputs*, *guaranteed output delivery*, and *fairness*. If these properties are fulfilled by the SMC system, then each party can be sure that: 1. the other parties can learn only the output of the computation, and not the raw data; 2. the result of the computation is going to be correct given the inputs; 3. all parties choose their inputs independently; and 4. all parties can receive the output.

In real-world scenarios, not all of the parties in the SMC are necessarily honest. There are two main types of adversarial behaviors described in the research literature: *semi-honest* and *malicious*. Semi-honest adversaries follow the SMC protocol strictly, but if an opportunity occurs, this adversary will try to reveal the data of other parties (also called honest but curious or passive). Malicious adversaries actively try to manipulate the SMC protocol and reveal the data of other parties. SMC secures only the computation process from malicious parties. Adversarial behavior in SMC does not include manipulations with the input data because all of the inputs are allowed (including false data). Ultimately, there is no generic protection against false data being input by adversarial parties into the SMC system [14]. Also, SMC does not protect from revealing the data of some party in cases in which the computation result itself reveals the data (e.g., in a two-party case, any calculation of the average of some parameter entered by both parties reveals to one party the input value of the other).

2.2. Trusted Execution Environments

Three key available technologies that provide memory encryption while the data are being processed, ordered by the time of their appearance, are: Intel SGX, AMD SEV, and the most recent one, Intel TDX. As the TDX was released on processors early in 2023, there have been no studies so far that have analyzed it in production. There are only a few theoretical analyses of the specifications that appeared before the actual release of TDX-enabled processors [15,16].

SGX's approach is to keep the trusted compute base small, limiting it to only the CPU and the enclave (trusted part of the application), thus keeping the operating system (OS) and the hypervisor out of the TCB. On the other hand, the SEV approach uses a larger TCB, with a virtual machine operating system inside it, and the hypervisor out. The problem with a larger TCB is that there are more possibilities for potential vulnerabilities and that the user needs to check more lines of code that belong to the TCB to make sure it can be trusted. However, the SGX approach requires that an application is divided into trusted and untrusted parts, making it expensive for existing applications to use this approach, as, most often, they need to be redesigned and rewritten. Another downside is the limited capacity of an SGX enclave, which is limited by the size of the Enclave Page Cache (EPC). The exact EPC size depends on the specific processor. Before the Intel Ice Lake processors generation, EPC memory ranged between 32 and 256 MB [17], which was small for many large data applications. Starting with the Ice Lake series of Intel processors, the EPC size has been substantially increased. The maximum EPC size can now reach up to 512 GB per processor [18]. This size can be increased to 1 TB on multi-socket systems (512 GB per processor). Enclaves share the EPC space. Having more enclaves decreases the performance of each enclave. As of the 11th generation, Intel has stopped supporting SGX on desktop processors, but it has continued to support SGX on Intel Xeon processors.

The SEV approach, on the other hand, has no memory limit other than the amount of RAM that is available for the virtual machine and can run any existing application in the secure VM. Even early comparison studies [17] have concluded that the SGX approach

is suitable only when performance is not essential but security is extremely important, while the SEV approach is more appropriate for performance-intensive applications. Later experiments confirmed these conclusions. Akram et al. compared the SGX and SEV approaches on high-performance computing (HPC) benchmarks and concluded that SGX is not appropriate for HPC because it has a limited secure memory size and a complicated programming model, which leads to significant performance degradation compared to the unencrypted execution [11].

The primary research focus of this paper is on confidential computing for artificial intelligence, which means that high-performance computing requirements have to be considered. This is one of the reasons we decided to use AMD SEV as a basis for our research.

2.3. SMC Using TEE

The first SMC applications for TEE were built for the SGX as the first widely available confidential computing technology. An up-to-date literature review of the research on the Intel SGX architecture, including its applications, was published by [19]. In this study, SMC is recognized as one of the SGX potential applications, classified by context under distributed data processing. The authors found six SMC system proposals based on the SGX, which all try to improve the efficiency of the computation, one solution that proposes a sandbox for remote computation for resource usage accounting, and one review paper on trusted hardware adoption in SMC. Will and Maziero also found a lot of solutions that address the issue of distributed machine learning, which is out of the scope of this paper. One of the solutions that attempts to improve the efficiency of computation is the work by [20], who constructed a general SMC protocol based on the SGX approach. The same paper was the only one related to SMC that was also mentioned in a recent survey of the Intel SGX approach and its applications [21]. According to this solution, only the user's private data are stored in the trusted part of the application, and the rest of the load, i.e., the functionality to be computed, is in the untrusted part. This design is somewhat forced by the limitations of the SGX, mentioned in Section 2.2. The focus of this proposed solution was on the protocol, achieving attestation and assurance about the security properties of the TEE. There are other solutions for SMC using SGX for some specific use cases, which were not included in the previously mentioned literature review. One such SMC solution based on SGX and focused on big data was presented by [22]. The authors focused on providing SMC for relational analytics queries without using cryptographic SMC techniques. This approach aimed to separate computations by those that can be run outside the trusted part and those that have to be run in the trusted part, minimizing the use of the enclaves. Again, such a solution was imposed by the SGX constraints that require that the applications to have secure and insecure parts. Later experiments confirmed the limitations of the SGX and concluded that SEV would be a better fit, as was presented in Section 2.2.

To the best of our knowledge, there are no research papers describing SMC using SEV published in the open literature. On the other side, Google recently launched Confidential Space, a cloud product that offers secure multiparty data sharing and collaboration, while allowing organizations to preserve the confidentiality of their data using secure virtual machines and dockerized applications [23]. At the moment of writing this paper, Confidential Space relies on the first version of AMD SEV. Demo applications show that the current version of Confidential Space assumes that the user's data are either uploaded to the cloud and stored on disks in cleartext (relying on general cloud storage encryption operated by Google) or encrypted using the Google cloud key management system, which again requires that the user trusts the cloud provider—in this case, Google. This is a significant simplification of one of the key SMC requirements—that the computing party is not trusted. Other potential weaknesses detected in a security assessment report are inadequate auditing and logging, over-permissive permission, a lack of separation of duty, lack of assurance of measured boot, and missing security controls in the underlying networks, applications, or operating systems [24].

2.4. Limitations of the Study

The main limitations of the SMC implementation using encrypted trusted execution environments come from the technology readiness level and some security issues of the existing trusted execution environments. Similar conclusions were found in a paper recently published by [25]. Research papers and vulnerability reports that describe potential attacks on TEE-enabled processors show that the technology is still maturing. For example, [26] presented a known plaintext attack on AMD SEV or SEV-ES. Using this approach, an attacker who controls the hypervisor and knows parts of the SVM kernel can successfully determine the tweak values used in the AES XE mode and can insert random 16-byte blocks in the encrypted memory, which ultimately leads to the execution of the random code, compromising the confidentiality this way. This attack exploits the lack of memory integrity in the first versions of the AMD SEV. Two other attacks that use the lack of memory integrity, but also the fact that CPU registers content that is not protected in the original AMD SEV, are described in [27,28]. Both attacks manipulate the Virtual Machine Control Block (VMCB). In the first attack, the memory content is transferred to the instruction pointer register (RIP) in the VMCB, and then an encryption/decryption oracle is created, leading to memory leakage. In the second attack, by observing the VMCB content, an attacker can reconstruct the executed code, and using instruction-based sampling (IBS), can detect which applications are running inside the VM. There are also other types of attacks. For instance, ref. [29] attacked the improper use of the address space identifier (ASID), which enabled the attacker to connect the victim's ASID with the attacker's VM, leading to the loss of confidentiality. The attacker has to be in control of the hypervisor and the attack is possible for SEV and SEV-ES. And even more specific attacks, like [30–32], focus on address-translation vulnerabilities, using either page faults to detect memory locations or the hypervisor's control of the nested page table (NTP) to detect and alter guest physical address (GPA), all leading to the loss of confidentiality. Finally, there are also hardware-based attacks, like the one in [33], where the authors showed how to use a voltage glitching attack, on any version of the SEV, that enables an attacker to load custom firmware and execute it on the platform's secure processor (PSP).

As this brief overview shows, SEV technologies have evolved and there have been three versions in only four years. Changes were primarily made upon the discovery of some vulnerabilities. In the original SEV approach, the memory was encrypted, keeping the hypervisor outside of the TCB, but some secret information would be available in the CPU registers, thus enabling the hypervisor to access them when VM stops running. This issue was addressed by AMD in their SEV-ES release [8] by encrypting CPU registers when a VM stops running. Both SEV and SEV-ES suffered from one obvious design flaw, and that was the lack of memory integrity; even though researchers tried to repair this issue with their custom solutions, like in [34], the issue was addressed by AMD in their SEV-SNP approach [9]. However, there are still a few attacks that are successful on all generations of SEV, like [33]. Given the previously described trend, it is expected that in the future, there will be further technology updates and that, at a certain point in time, there will be a mature and fully secure technology for secure virtual machines, with all known vulnerabilities eliminated. Until then, it is important to be aware of the limitations and the final reach of the existing technologies.

Finally, we need to stress that all TEE technologies used today rely on the cryptographic algorithms that are not quantum computer resistant. Given the typical operational lifetime of the servers procured today, there is a possibility that, during that time, the key exchange and signature mechanisms used in their processors will become obsolete. However, we believe that swapping to some post-quantum mechanisms is not going to present a significant burden for processor manufacturers.

3. Suggested Approach

In this section, we describe a characteristic SMC use case and key actors and their roles in it. Next, we discuss the trust model associated with the proposed use case. Then,

we describe the conditions that need to be met to establish SMC using TEE. After that, we present the remote attestation process used in the AMD SEV approach and how it could be used for the SMC use case. Finally, based on all of the previous discussions, we propose a novel system architecture and an associated protocol for the initialization of our SMC secure module and conclude this section with the security evaluation and the threat analysis.

3.1. TEE-Based SMC Key Actors

The SMC use case that we considered is a multi-party medical diagnostic system. It assumes that several hospitals want to use an AI-based diagnostic system for some specific diseases. To create a more reliable system by covering as many as possible patient characteristics, hospitals want to join their datasets, but without revealing the raw data to the other hospitals. Also, hospitals do not have the capacity to create an AI-based diagnostic system, so they would like to hire a company specialized in such systems. This company also does not want to reveal the code and work methodology to the hospitals and considers the code to be their intellectual property. Hospitals and a code provider create a consortium and look for a provider of secure multiparty computation services. Secure multiparty computation provisioning consists of both providing secure virtual machines on TEE-enabled hardware (classical cloud service) and orchestrating the SMC process. These functions can be offered by two different companies or by a single company (e.g., the case of Google Confidential Space). In our scenario, we consider the former case, which can easily be generalized to a single-company case by joining both functions into a single actor. The key entities in this SMC scenario and their roles, as depicted in Figure 2, are:

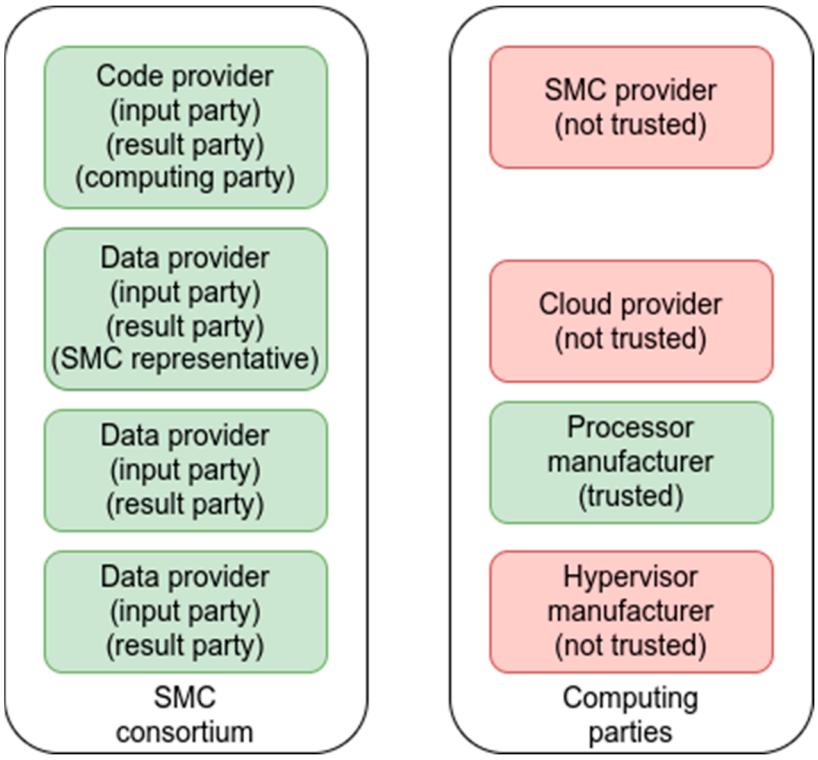

Figure 2. Involved parties in Secure multiparty computation.

- *SMC consortium*, which consists of multiple data providers and a code provider. It is expected that they are bound by some internal consortium agreement that regulates their rights and obligations. Data providers share their encrypted datasets (e.g., medical records) and want to use the results of the diagnostic system. They can be both input and result parties. The code provider provides the data computation code. If it does not want to reveal the code, it is assumed that the code is sent encrypted

to the SMC system. The code provider is generally the input-only party, but in some cases, if it has to use some intermediate computation results (e.g., for machine learning hyperparameter tuning), then the code provider can also be a result party. Since the code provider creates the code which computes the results of the SMC, it is also a computing party.

- *SMC representative* is one member of the SMC consortium who is chosen to represent the consortium in contracting the SMC service with the SMC provider. The SMC representative can be either a code or data provider and there can be only one member of the consortium with this role. The existence of one "special" consortium member with a slightly different role from the others is not entirely in keeping with the spirit of SMC, by which all participants/input parties should be equal. However, as will be explained later, the secure virtual machine attestation process as it is designed on the existing technologies assumes that there is one party (guest owner) that participates in the attestation process and implies such an asymmetric organization.
- *SMC provider* is a company that facilitates the SMC process, orchestrates the virtual machine provisioning process, and provides the interface between the cloud system and SMC consortium. This can be the cloud provider, but to provide a different trust model, as will be discussed further, it can be another entity.
- *Cloud provider* owns servers with TEE-enabled processors, hypervisors, and related cloud management infrastructure and offers secure virtual machines as Infrastructure as a Service. The cloud provider allows the attestation of the virtual machines it offers. The cloud provider is not trusted by any of the consortium members.
- *Processor, hypervisor* (or *virtual machine emulator*) and *server* and *virtual machine BIOS/UEFI providers*, although not directly involved in the contracted secure multiparty computation use case, implicitly participate in the SMC. The processor manufacturer ensures that the code is executed in a secure virtual machine through the remote attestation process, which provides cryptographic proof that the code is executed on the appropriate hardware. The server's BIOS/UEFI enables secure virtual machine encryption (SEV functionality), while the virtual machine UEFI participates in defining which memory pages will be encrypted and is an essential part of the attestation process. The virtual machine UEFI of choice for AMD SEV is Open Virtual Machine Firmware (OVMF) [35], although there are solutions that use qboot [36]. The hypervisor also participates in the secure virtual machine attestation, as will be described in Section 3.4.

3.2. Trust Model

In this paper, we make several assumptions under which it is possible to achieve SMC on TEE, which have an impact on the overall solution security:

- All parties that participate in SMC are semi-honest (honest but curious) and follow the SMC protocol honestly. Data providers input their real data and do not provide fake data to forge the computation results. As mentioned earlier, there is no general way of preventing malicious behavior in which some party provides fake data. In general, the assumption is that there is a common interest among the consortium members and that all members agree to share their data to obtain the overall benefit from computations over richer joint datasets.
- There is no collusion between the CPU manufacturer, hypervisor producer, and cloud provider. Memory encryption keys stay within the CPU and cannot be leaked to third parties. Therefore, memory content cannot be leaked to the CPU manufacturer, hypervisor producer, and cloud provider.
- There is no collusion between the SMC representative and SMC and/or cloud provider. The way such collusion can result in data leakage specifically in the SMC use case is explained in detail in the next section.
- Underlying hardware, firmware, and hypervisor security can be measured and verified. For the SMC use case, it is assumed that these components are secure.

- The SMC computation code can be inspected and certified that it does not leak the data to any of the consortium or third parties.

3.3. Towards Secure Multiparty Computation in Trusted Enclaves

Secure virtualization technologies provide the privacy of the virtual machine code and data stored in the RAM from external actors like the hypervisor, server operating system and virtual machine emulator (e.g., qemu [37]), server owner, or other virtual machines. On the other hand, an actor who is logged into a secure virtual machine (e.g., using ssh access) can see the data processed by that virtual machine by executing commands that dump the memory content. Once logged into the secure virtual machine, all the memory content of that virtual machine is visible. For a single user of an outsourced computation on a remote TEE-based machine, this is not an issue. However, to preserve secure multiparty computation data privacy from internal adversaries in a TEE-based environment, either shell access on a well-known open-source operating system should not be allowed to any of the SMC actors because it enables actors logged into the system to see the data of other parties or a secure virtual machine operating system has to be customized in such a way that executing memory dump commands is not available or possible for any of the users logged into the shell. In the former case, closing some communication channels on a secure virtual machine can be conducted at the boot time, as will be described in the remainder of this section. For full security, the latter solution implies that a full software audit of a customized operating system has to be conducted before the start of its use, which can be a costly and time-consuming process.

Without the shell access and to allow sending the data securely into the SVM RAM for processing, encrypted network connections (e.g., TLS) must be established from each input party directly into the SVM RAM. The data can either be sent in cleartext through such an encrypted network connection, which protects data in transit, while SEV protects the data in RAM, or without a secure channel, but encrypted using the input party's symmetric key, in which case the data can even be stored encrypted on the disk. However, in the latter case, the cryptographic keys needed to decrypt and later use the data must be sent securely and directly into the secure VM RAM using a secure network connection. Therefore, in both cases, a server process on the SVM that will serve as the endpoint for secure network connections is needed to provide full data-in-transit and in-use protection. Initiating such a server on secure virtual machines on a machine with untrusted hardware and a hypervisor and providing it with the necessary cryptographic material is a non-trivial task for which we propose one method in Section 3.7.

Multiple virtual machines can exist on the same server and host the same network services. Therefore, virtualization platforms often use some sort of port forwarding and/or address translation to enable external communication towards the deployed VMs. Figure 3 shows an example of two VMs with identical operating systems and application software (e.g., the same operating system image file is used on both) deployed on one server, where a web server on the first VM is visible from the internet over port 4431 and the second over port 4432 (in both cases, these ports are translated to the common 443 web port on a VM). Also, both machines use qemu management protocol (QMP) to enable some remote management functions, including obtaining the remote attestation measurement. QMP ports on two machines cannot overlap and must be different. In this example, one machine is secured using SEV and the other is not. The difference in the boot process configurations of the two machines is in those lines of a configuration command which defines the type of SEV encryption that defines port forwarding for web servers and QMP parameters (marked red on Figure 4). If a secure virtual machine is installed properly, but the users of its web service obtain the port of a non-secure VM for communication (e.g., a malicious platform owner or hypervisor intentionally misinforms them), they could conduct all of the processing as if the data are being sent to the secure VM, only their data will be processed on the non-secure VM and visible to the platform owner. This is a case of the cuckoo attack [38], in which a malicious hypervisor or platform owner plants a non-secure VM

and directs users to use it instead of a secure VM. To prevent this type of attack, secure virtual machine users must be assured that the ports they use to upload the data into the secure virtual machines correspond to the ports that are configured for the verified and attested secure virtual machine (specific ports that are forwarded should correspond to the attestation setup and files). This further means that a high-touch secure virtual machine provisioning methodology in which the SVM user can fully control the installation of a secure virtual machine on a bare metal server deployment provides more trust in computation security than those deployments conducted through the cloud provider's closed source VM provisioning automation software.

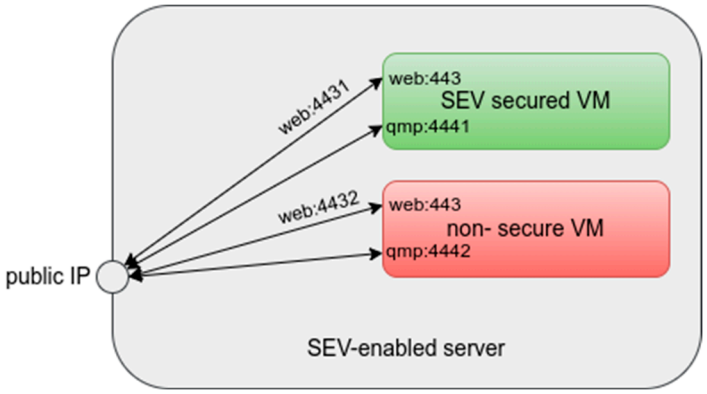

Figure 3. Port forwarding on a virtualization server.

```
/PATH/qemu-system-x86_64 \
  -enable-kvm -cpu EPYC-v4 -machine q35 -smp 4,maxcpus=32 \
  -m 4096M,slots=5,maxmem=30G \
  -no-reboot \
  -drive if=pflash,format=raw,unit=0,file=/PATH/OVMF_CODE.fd,readonly=on \
  -drive if=pflash,format=raw,unit=1,file=/PATH/focal-server-cloudimg-amd64.fd \
  -netdev user,id=vmnic,hostfwd=tcp::2222-:22,hostfwd=tcp::5550-:5050,hostfwd=tcp::5551-:5051 \
  -device virtio-net-pci,disable-legacy=on, iommu_platform=true, netdev=vmnic,romfile= \
  -drive file=/PATH/focal-server-cloudimg-amd64.img, if=none, id=disk0, format=qcow2 \
  -device virtio-scsi-pci,id=scsi0,disable-legacy=on,iommu_platform=true \
  -device scsi-hd,drive=disk0 \
  -machine memory-encryption=sev0,vmport=off \
  -object sev-guest,id=sev0,cbitpos=51,reduced-phys-bits=1,policy=0x39,dh-cert-file=/PATH/
godh.cert.base64,session-file=/PATH/launch_blob.base64 \
  -nographic \
  -qmp tcp:localhost:4444,server=on,wait=off \
  -monitor pty \
  -monitor unix:monitor,server,nowait
```

Figure 4. An example of qemu command for booting AMD SEV secure virtual machine with port forwarding for SSH access to the web application on the SVM, attestation, and QMP (these commands highlighted in red).

The previous discussion implies that there are several important additional constraints that have to be respected to obtain a trusted SMC:

- Console access, which is commonly used for debugging remote resources in various virtualization systems (e.g., VMware, qemu), must be switched off. Console access allows shell access to a virtual machine, and thus to dumping memory content. A party that has console access would have an unfair advantage over the other parties.
- For the same reason, shell access or memory dump functionality on the secure virtual machine must be disabled and no one should be able to connect to the SVM and dump the memory content.
- Data (including both the data that are processed and the cryptographic material needed to decrypt the data) must be sent to the secure virtual machine via an encrypted network session which ends on the SVM and cannot be stored on cloud disks unprotected.

- These constraints mean that to have full trust in the TEE-based SMC system, one has to be sure about the following:
- that the system is executed on the trusted hardware with the appropriate TEE functionality turned on;
- that the hypervisor or virtual machine emulator have the appropriate configuration (e.g., no console access, the exact configuration of port forwarding, well-known application ports, etc.);
- that the operating system has the appropriate configuration and that only the required services are turned on.

The remote attestation process, described in the following section, currently only partially fulfills these requirements.

3.4. AMD Attestation

Remote attestation is a process that proves to the SVM user that SEV protection is in place and that the virtual machine was not subject to manipulation. Before sending secrets to the SVM, the SVM user must verify the attestation information. While this process is the same for the SEV and the SEV-ES, it changed with the SEV-SNP. From the SMC use case point of view, there is little difference between the SEV and the SEV-SNP attestation processes. In the following subsections, the SEV and the SEV-SNP attestations are described, and their differences are highlighted.

3.4.1. AMD SEV, SEV-ES Attestation

This process involves three main actors: AMD, which proves that the processors have the SEV capability and that it is switched on for each particular virtual machine; the platform owner (PO), which verifies its authenticity; and the secure virtual machine user—guest owner (GO), who verifies the installation. The first operation in the process of the remote attestation is the creation of a certificate chain that is used to verify the platform on which the SVM will be executed. The keys that are parts of the certificate chain are:

- AMD Root Key (ARK)—an RSA 2048 asymmetric key pair;
- AMD SEV Signing Key (ASK)—an RSA 2048 asymmetric key pair;
- Chip Endorsement Key (CEK)—an ECDSA curve P-384 asymmetric key pair;
- Platform Endorsement Key (PEK)—an ECDSA curve P-384 asymmetric key pair;
- Owner Certificate Authority (OCA)—an ECDSA curve P-384 asymmetric key pair;
- Platform Diffie-Hellman Key (PDH)—an ECDSA curve P-384 asymmetric key pair.

At the root of the certificate chain is ARK. The ARK private key is used to sign ASK. ARK and ASK public key certificates can be obtained from the AMD Key Distribution Server (KDS). The ASK private key is then used to sign the CEK. CEK is unique for each AMD SEV processor, so by signing CEK with ASK, AMD enables users to verify that the platform is run on an authentic AMD SEV processor. PEK is then signed using the private key of CEK and with the private key of OCA. The OCA belongs to the PO. By signing the PEK with OCA, the PO enables the GO to verify the authenticity of the owner of the platform. The PEK private key is used to sign PDH. PDH will be available to the GO at the beginning of the remote attestation process.

The PO should export the PDH certificate and the whole certificate chain (Figure 5) to the GO.

The GO needs to take some steps in order to be sure that his/her code/data will run safely inside an SVM. These steps are:

- Fetching the PO's certificate chain and the code of a virtual machine firmware (in the case of AMD SEV, OVFM is the firmware that is supported and recommended);
- Verification of fetched certificates. Verification is conducted using the AMD's and the PO's certificates.

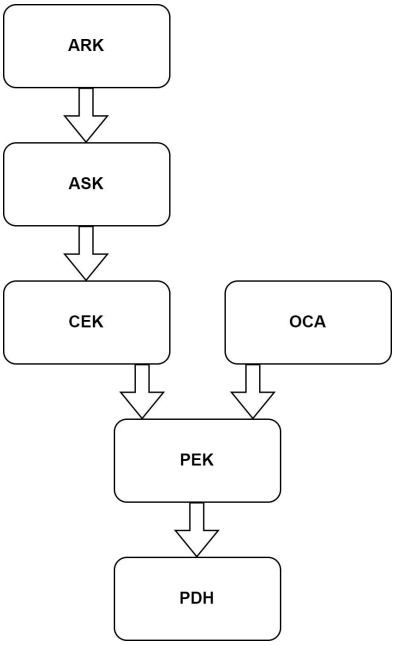

Figure 5. AMD SEV certificate chain.

The next step for the GO is to exchange the master secret with the PO (Figure 6). The GO generates an ECDH key pair and sends the public key, along with a nonce value (N) to the PO. Now, both the PO and the GO can calculate the same secret value (Z) using the ECDH algorithm. For this algorithm, the GO is using the generated ECDH private key and the PDH public key, and the PO is using the received ECDH public key and the PDH private key. Then, the master secret (M) can be created using the counter mod [39], as a key derivation function based on Z and N in a HMAC-SHA-256 pseudorandom function. Finally, the Z value is deleted, and the value M can be used for further communication.

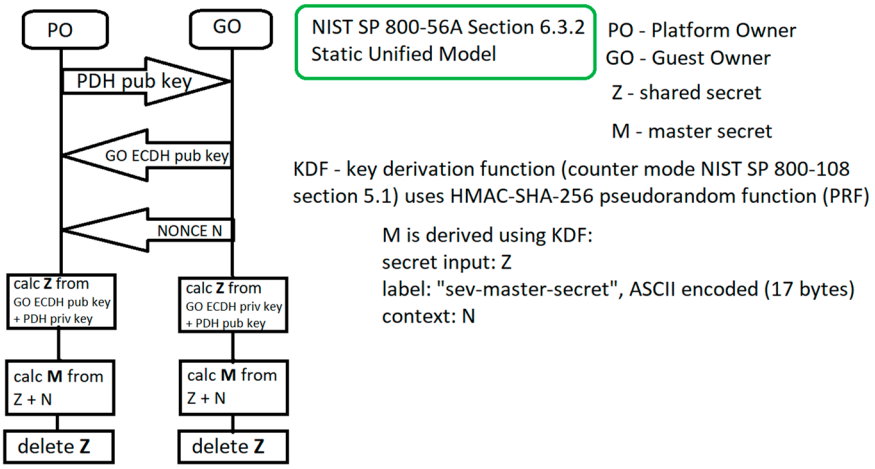

Figure 6. Master secret generation.

Additional keys need to be generated by the GO:

- Transport Integrity Key (TIK)—HMAC-SHA-256 symmetric key;
- Transport Encryption Key (TEK)—AES-128 symmetric key.

The communication (Figure 7) is started by the client, who randomly generates the TIK key and the TEK key. The KIK and the KEK keys are generated next based on the master secret M and finally, the triplet IV, C, and MAC are calculated on the client side and sent to the server. The server is capable of generating the KEK and the KIK keys, which can be used to decrypt the TEK and the TIK keys, which can be used for further secure communication.

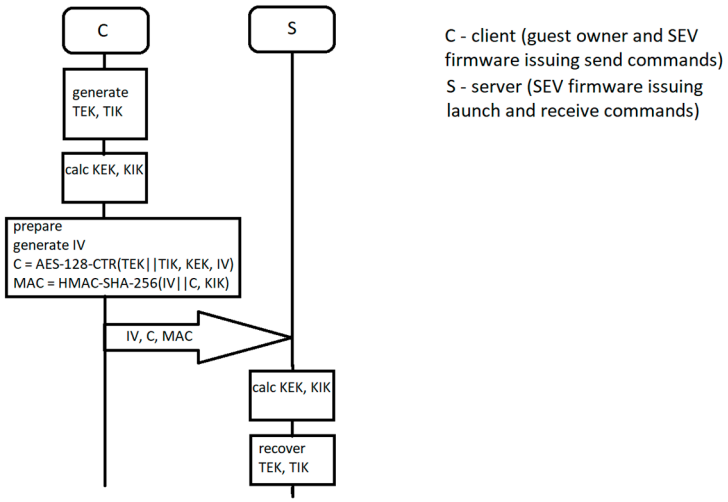

Figure 7. Client server secure communication.

Following all of the previous steps, the client now can perform remote attestation by obtaining the measurement from the server and making sure that the measurement matches the expected value. The measurement is calculated as an HMAC over seven concatenated values, with the TIK key used for calculation. The values are:

- Context—always has value 0x04;
- API_MAJOR—major AMD SEV API version;
- API_MINOR—minor AMD SEV API version;
- BUILD—build version;
- GTCX.POLICY—defined by the SMC member;
- GTCX.LD (DIGEST)—SHA256 output digest over OVMF UEFI used during VM launch;
- MNONCE—a random value generated by the PSP firmware at the server.

GTCX designates Guest Context and the GTCX.POLICY is the value used by the client to define certain virtual machine configuration options, like enabling the SEV-ES or debugging mode. This value is set by the client and verified during the SVM boot process. The UEFI used during the SVM launch must be built as a stateless firmware file that does not use a non-volatile random-access memory (NVRAM) store. AMD supports OVMF at the moment, so if only the basic SEV VM is started, GCTX.LD is SHA256 of the UEFI file. The server has to send the UEFI image and the MNONCE to the client so that he/she can calculate the expected value of the measurement. If the calculated value of the measurement matches the value received from the server, then the remote attestation is successful. The hypervisor incorporates the attestation process into the boot process of a secure virtual machine. If the client–server secure communication is successfully established, then the machine will be booted up and the entire VM content will be encrypted using the VM Encryption Key (VEK), which is an AES-128 symmetric key randomly generated and available only to the processor for encryption/decryption.

In addition to the calculation of the UEFI digest during the attestation process, as described above, the qemu software extended the input to the HMAC function to include additional elements and supports the following measurement: *hash*(*firmware_blob*||*kernel_hashes_blob*||*vmsas_blob*), where *firmware_blob* is the content of the entire firmware flash file (for example, OVMF.fd), *kernel_hashes_blob* is the content of PaddedSevHashTable (including the zero padding), which itself includes the hashes of kernel, initrd, and cmdline that are passed to the guest, and *vmsas_blob* is the concatenation of all VMSAs of the guest vcpus. This means that some necessary components of the hypervisor and operating system configuration are still not measured and checked. At the moment, there is no out-of-the-box solution for verifying all the critical components (e.g., qemu configuration or the whole SVM operating system). Decentriq recently reported on additional hardening of the secure

virtual machine's initial boot, which includes measuring the OS disk, gaining the integrity of the loaded operating system [36].

3.4.2. AMD SEV-SNP Attestation

The process of attestation in the SEV-SNP involves the same actors as in previous SEV versions. However, the SEV-SNP introduces two new keys for the attestation report signing: the Versioned Chip Endorsement Key (VCEK) and the Versioned Loaded Endorsement Key (VLEK). The VCEK is a private ECDSA key constructed using the CEK key hashed together with the version numbers of all the TCB components. The VCEK is unique to each AMD chip with the specific TCB. The VCEK is signed using the ASK and ARK keys to authenticate the attestation report. The TCB components are:

- Microcode of the CPU;
- SNP firmware;
- PSP operating system;
- PSP bootloader.

The VLEK is an alternative to the VCEK and can replace it in the attestation report signing. The VLEK is an ECDSA P-384 signing key signed by AMD. Unlike the VCEK, which obtains the unique seed from the chip, each PO needs to enroll with AMD and obtain a unique VLEK seed from the AMD Key Derivation Service (KDS). To use the VLEK, the PO needs to send to the KDS the current TCB version and the chip ID of the processor that the PO is using. The AMD KDS calculates the so-called VLEK hashstick from the delivered TCB and the POs VLEK seed and wraps it using an AES-256-GCM key derived from the chip ID information. The PO then provisions the platform with the wrapped VLEK hashstick. Once the VLEK hashstick is loaded, the firmware can use the VLEK as a replacement for the VCEK. VLEK functionality is introduced in the latest available version of the SEV SNP Firmware ABI Specification, and its use is not yet available as a service from AMD [40].

The GO can provide the PO with an identity (ID) block. The ID block is provided to the hypervisor at the launch of the SVM. The ID block is a data structure that contains the expected measurement launch digest, the GO policy, and the cryptographic signature of the ID block. The ID block is signed using a private ECDSA ID key. The launch of the SVM will fail if the ID block measurement is different from the calculated measurement at launch. The GO sends the hypervisor the ID block and the public ID key needed for signature verification. The hypervisor passes this public ID key to the SNP firmware.

The SEV-SNP guest policy information has the same role as the guest policy in the first two SEV versions. It contains binary flags that defines how the SVM should be booted. The guest policy contains information like the debug flag and the minimum major and minor versions of the SNP firmware.

Another change introduced in the SEV-SNP is a different process of fetching and sending the attestation report to the GO. The attestation report is retrieved from the SVM using a kernel driver (*sev-guest* driver) to interact with the SNP firmware and retrieve the report. The *sev-guest* driver communicates with the SNP firmware by sending messages encrypted using the VM platform communication keys (VMPCKs) through the hypervisor. During the launch of the SVM, a special secrets memory page that contains the VMPCKs is inserted into the SVM. The firmware decrypts the messages, and if a message is an attestation report request, sends the attestation report back to the hypervisor and then to the guest.

The most important fields in the SEV-SNP attestation report are the measurement of the launched SVM, the policy provided by the GO, TCB information, the digest of the ID key that signed the ID block, and the report data provided by the guest. The report data are a 512-bit data block provided to the firmware at the moment of the attestation report request. The report data are not interpreted by the firmware and are placed in the attestation report without modifications. As a result, the report data can be used to transfer relevant information to the GO. For example, the report data can be a public key of a

generated keypair, which can be used to establish a trusted channel with the GO. Upon receiving the attestation report, the GO verifies the report by checking the signature and ensuring that the information is valid.

3.5. AMD SEV Attestation for SMC

The default attestation protocols described in the previous two sections do not provide all the assurances defined in Section 3.3 required for the SMC use case. It does not make any assurance that the hypervisor or virtual machine emulator have the appropriate configuration (e.g., no console access, port forwarding configuration, etc.), and that the operating system has the appropriate configuration with only the required services turned on. With the latest versions of qemu and the development of the AMD Secure VM Service Module (SVSM) [41], it is possible to add the kernel and initial file system in the TCB. In the CoCos.ai system, we have managed to boot secure virtual machines with just a kernel, an initial file system (initramfs [42]), and a kernel command line. The entire VM is measured by the firmware thanks to the AmdSev package of OVMF [35], which allows for inserting the hashes of the kernel, initramfs, and the kernel command line into the firmware binary and can be verified by the guest owner. However, hypervisor configuration still remains outside of the attestation process.

Further, the attestation processes for both SEV and SEV-SNP are designed to be performed by a single guest owner, primarily supporting a single-user secure outsourced computing use case. The key exchange methods used in them are point-to-point between the GO and the platform and excludes other consortium members from the process. For an SMC case, such attestation protocols imply that one member of the consortium (e.g., SMC representative) who is performing the attestation process has to conduct the attestation, but this way, the SMC representative obtains an advantage over other consortium members as they have to trust that there will be no collusion with the platform owner. SMC representatives must actively participate in the high-touch process of exchanging cryptographic material, establishing the appropriate policy and attested booting of the secure virtual machine to ensure the correct configuration. The use of the derived guest key via an SNP_GUEST_REQUEST call does not help in this process. Guest keys are derived in a deterministic way from the set of parameters that are either the same for all the virtual machines on one processor (e.g., VCEK or TCB version) or using the data that are known to the hypervisor at the launch (e.g., host data or ID Key/Author Key) or have a limited number of possible values (e.g., Virtual Machine Privilege Level—VMPL) [40].

Therefore, after completing the attestation process, there are two options for other SMC consortium members to verify the attestation process:

- The SMC representative could distribute the TEK and TIK keys to the other members of the SMC consortium, enabling them to request the measurement from the SMC provider through a channel, like qemu management protocol (QMP), and check it in the same way as the SMC representative did (Figure 8a);
- Other members can obtain the attestation report and the PEK certificate chain directly from the SMC provider. After they verify the PEK certificate chain, they can use it to validate the attestation report that also contains SHA256 over UEFI, and is signed using the PEK (Figure 8b).

However, if malicious, this SMC consortium representative can, in collusion with the SMC provider, steal the data of the other consortium partners by misinforming them about the exact ports of the secure virtual machine and redirecting their data to a non-secure machine (using, e.g., cuckoo attack, described in Section 3.3).

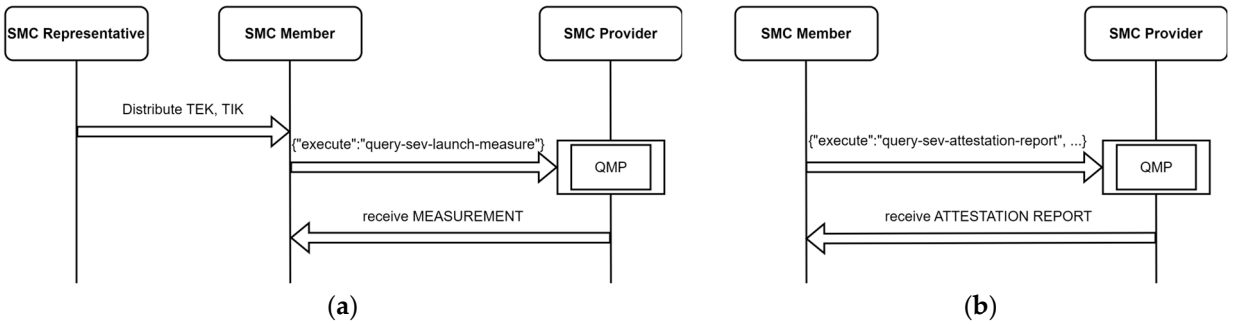

Figure 8. (**a**) SMC member attesting the measurement; (**b**) SMC member verifies attestation report.

3.6. System Architecture

The previously mentioned constraints have further implications on the design of the TEE-based SMC system. Trust in the operating system can be obtained only by providing an open-source operating system image with preinstalled open-source SMC orchestration software, which can be inspected by the SMC consortium members and measured.

The SMC orchestration solution proposed in the CoCos.ai project consists of two components:

- SMC Business Application (BA): A web application installed on any web server (not necessarily secure). Its role is to create accounts and roles for all the parties in the SMC and consortium and to provide a graphical user interface for all the parties and a web service for the SMC Secure Modules;
- SMC Secure Module (SM): a web service on a secure virtual machine that exposes an API used by the SMC BA to securely send the encrypted data and code, and SMC consortium members to send the cryptographic keys, receive the results, and monitor the computation process.

As mentioned previously, one of the key issues is the initialization of the SMC SM web service on the secure virtual machine. If the communication towards the SMC SM is supposed to be secure using TLS, for example, then the cryptographic material needed for it to run must be placed on the operating system image. If that cryptographic material is unprotected in the image file, then it is visible to the platform owner. If it is protected using some symmetric encryption algorithm and keys that are not stored in the image, then there is a problem with securely transferring this key to the secure virtual machine.

3.7. Protocol for SMC Initialization

We propose a simple and secure protocol for SMC SM initialization. A secure virtual machine image is an image of the operating system with a preinstalled minimal set of components needed for its operation and the SMC SM software which is supposed to start upon the start of the secure virtual machine. The SMC consortium (or their representative) uploads the data necessary for secure initialization to the secure virtual machine image (e.g., using *virt-customize* command). That data consists of:

- The public key of the SMC BA $\mathbf{Pu_{WA}}$ (e.g., from the SMC BA web server TLS certificate);
- Contact URLs on SMC BA that will be used by the SMC SM to contact the business application;
- A random session identifier $\mathbf{N_S}$ encrypted using the randomly chosen symmetric key of the SMC Consortium representative $\mathbf{K_{CC}}$–$\mathbf{K_{CC}(N_S)}$.

All of these elements are either public ($\mathbf{Pu_{WA}}$, contact URLs) or encrypted ($\mathbf{K_{CC}(N_S)}$) and can be stored in the operating system image file without worries that something might be compromised. An adversary can know the public data, but not the value of the session identifier $\mathbf{N_S}$ one consortium has created. Upon SVM startup:

1. SMC SM generates a random symmetric encryption key $\mathbf{K_S}$ and sends it to the BA encrypted using BA's public key $\mathbf{Pu_{WA}(K_S||N_1)}$, where $\mathbf{N_1}$ is an anti-replay nonce. The key $\mathbf{K_S}$ is kept on the SMC SM only in RAM and must never be stored on the disk.
2. SMC BA decrypts the session key using its own private key $\mathbf{Pr_{WA}}$ and sends the TLS certificate and keys needed for the secure VM to start the web service encrypted with the session key $\mathbf{K_S}$: $\mathbf{K_S(CERT_{SMCSM}||Pr_{SMCSM}||N_1)}$, where $\mathbf{CERT_{SMCSM}}$ and $\mathbf{Pr_{SMCSM}}$ are the TLS certificate and a private key that the SMC SM web service will use. Each new secure virtual machine that contacts BA must obtain a different (CERT, Pr) pair.
3. SMC SM decrypts the $\mathbf{CERT_{SMCSM}}$ and $\mathbf{Pr_{SMCSM}}$ and starts the TLS-based web interface. Again, these keys must be kept at all times only in RAM and never stored on disks.

The purpose of these steps is to establish a TLS-secured web service on the SMC SM, which will be trusted by the SMC BA (hence the certificate and key pair sent from the BA). Until this moment, any adversary can conduct steps 1–3, assuming that the process is somehow made publicly available (among other reasons, to allow security assessment of the whole process). However, a random adversary will not have its secure VM pre-initialized with the appropriate ($\mathbf{K_{CC}(N_S)}$) value. If the hypervisor is malicious and performs a cuckoo attack, then the same $\mathbf{K_{CC}(N_S)}$ value can be on a non-secure machine. The final steps for verifying the identity of the SMC SM instance are:

4. SMC BA establishes a TLS session with the SMC SM through the port known from the SVM initialization process, verifies using the $\mathbf{CERT_{SMCSM}}$ the identity of the SMC SM and sends to the SMC SM ($\mathbf{K_{CC}||N_2}$), where $\mathbf{N_2}$ is an anti-replay random nonce. If the hypervisor is not malicious and has not fooled the SMC representative to establish the connectivity towards the fake non-secure VM, then only the SMC SM can obtain the $\mathbf{K_{CC}}$ key.
5. SMC SM decrypts $\mathbf{K_{CC}(N_S)}$, which was preinstalled on it using the obtained $\mathbf{K_{CC}}$, and returns to the SMC BA decrypted $\mathbf{(N_S||N_2)}$ through the TLS session. An adversary that does not have an appropriate ($\mathbf{K_{CC}(N_S)}$) value will not decrypt an accurate $\mathbf{N_S}$ value and the BA will be able to reject such a secure VM.

The whole protocol is depicted in Figure 9. The exchange consists of two separate phases related through the $\mathbf{N_S}$ parameter and separated by the TLS exchange, which protects the second phase. In the next section, we analyze the security of the proposed scheme.

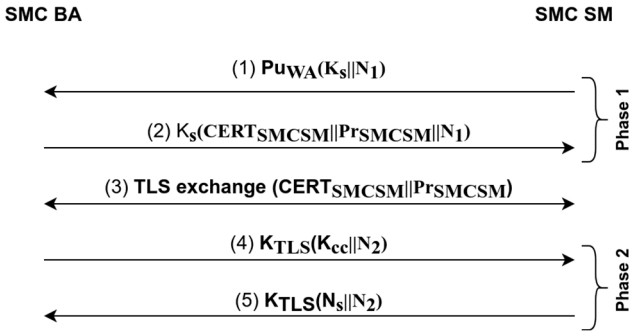

Figure 9. Protocol for SMC initialization.

3.8. Security Evaluation and Threat Analysis

There are two types of adversaries who can attack the previously described SMC initialization protocol and secure virtual machines: external—those that are outside the set of actors described in Section 3.1, and internal—actors described in Section 3.1. We discuss potential attacks coming from both types of adversaries.

External adversaries can exist at any point on the internet between the SMC actors and can intercept, copy, replay, modify, delete, and/or insert any protocol message or even perform other types of attacks (e.g., Denial of Service—DoS). The way external adversaries

can attack the protocol and secure virtual machines depends on the way the access to the SMC resources is set up by the cloud provider. SMC BA and SMC SM can be behind the firewall and accessible by SMC consortium members via an encrypted VPN or be accessible openly over the public internet. In the former case, the protocol exchange is protected by the VPN tunnels and the adversary cannot see the message content or change it unnoticeably. Also, the SVM is behind the firewall, not accessible directly over the internet, which reduces the risk of other types of attacks (e.g., DDoS). The security of the SMC protection in that case, in addition to the security of the protocol, depends on the security of the VPN and firewall solutions. In the latter case, the adversary can see the protocol messages, as described in Section 3.7, and could perform various attacks, as mentioned above. For this more challenging case, we have performed a formal analysis of the exchange protocol security using the ProVerif tool v.2.0.4 [43]. We have analyzed both exchange phases separately. The first phase was tested for the security of the TLS credential transfer $\mathbf{(CERT_{SMCSM} || Pr_{SMCSM})}$. The exchange passed formal verification under a reasonable assumption that the SMC BA private key ($\mathbf{Pr_{WA}}$) is not compromised and known to the attacker. The second phase was tested for the security of the session nonce ($\mathbf{N_S}$) transfer, which allows SMC BA to finally confirm the identity of the secure VM. This phase also passed formal verification, under the assumption that TLS session keys are not compromised. The ProVerif code for both formal tests is available at [44]. Although the exchange is proven secure, even in the latter case, it is recommended that access to the SMC resources is established through the firewall and VPN to reduce the population of potential adversaries and the set of attacks that could be performed.

As introduced in Section 2.1, internal adversaries (key SMC actors) can be semi-honest or malicious. The case in which internal adversaries are semi-honest is similar to the case of external adversaries when there is no VPN protection because semi-honest adversaries can see protocol messages. However, semi-honest SMC members are passive and do not attempt to change the protocol and gain access this way. The formal analysis of the protocol exchange shows that the protocol is secure in cases of passive message observation. Therefore, it can be concluded that using this SMC initialization protocol, and under the assumption that the secure virtual machine booting and verification process is improved according to the recommendations given in Section 3, secure multiparty computation using secure virtual machines can be possible in semi-honest adversarial settings.

Finally, it does not seem that SMC is possible in malicious adversarial settings, in which active internal adversaries modify or inject messages or collude to steal the data of other participants. A malicious SMC BA owner can expose and use SVM private TLS key $\mathbf{Pr_{SMCSM}}$ or session nonce $\mathbf{N_S}$ to compromise the SMC initialization process. In collusion with a malicious hypervisor, it is possible to perform the cuckoo attack by creating a non-secure virtual machine capable of responding to the SMC initialization protocol the same way as a legitimate SVM as it can access all of the necessary data for this (decrypted $\mathbf{N_S}$). For this reason, it is recommended that the SMC provider and cloud provider are not the same entity.

4. Proof of Concept

In this section, we give a description of an implementation of the SMC system, which is secure under the assumption that there is no collusion between the SMC representative and platform owner, and the performance evaluation of the proposed implementation. A proof of concept of the SMC system developed for AMD SEV is described in the remainder of this chapter. The system was installed on a server with an AMD EPYC 7313P 16-Core processor with 32 GB of RAM and 2 TB SSD. This processor belongs to the 3rd generation of AMD EPYC processors and supports the SEV, the SEV-ES, and the SEV-SNP. The SEV software used on this server was from AMD's SNP development branch [sev-snp-devel]. The version of the server and SVM kernels are 6.1.0-rc4-snp-host-93fa8c5918a4 and 5.4.0-135-generic, respectively.

4.1. Establishing SVM

The first step taken by the SMC provider is the installation of the sev-tool [45]. This tool can be downloaded from the official sev-tool GitHub. The sev-tool is a command-line utility specifically designed for working with SEV-enabled systems and provides various commands for performing tasks related to SEV administration, key management, attestation, and more. Users can also employ this tool on systems that do not support SEV. The SMC provider also needs to prepare an image of a virtual machine that will become an SVM. The image needs to be publicly available so that the members of the consortium can verify the image. SMC SM should be installed on the image and configured in a way that it automatically runs at the start of the SVM. The image should also have a tool that can check for backdoors to the SVM so that the SMC members can be sure that no information is being exported without their knowledge. Finally, for the purpose of the remote attestation process, the SMC provider initiates the creation of a certificate chain that is used to verify the platform on which the SVM will be executed. The commands for the creation of a certificate chain can be found on the official sev-tool GitHub repository.

After the SMC provider has completed the previous steps, the SMC representative first needs to install the sev-tool locally on his/her machine. After that, the SMC representative obtains the SMC provider's certificate chain and verifies it. The sev-tool can also be used here for verification. The SMC representative should verify that the obtained ARK and ASK from the SMC provider match the ARK and ASK that are obtained from AMD.

The next step for the SMC representative is to generate an ECDH keypair. In order to create and start the secure VM, according to the steps described in Section 3.4.1, the SMC representative prepares his/her launch blob, which is an encrypted file, and his/her certificate and sends them to the SMC provider. The policy is a hex value that represents the configuration of a VM. The SMC provider's app must send the command for starting the VM with a previously loaded launch blob, certificates, and the rest of the configuration of the VM.

As we decided to use secure encrypted virtualization, our attestation protocol is related to AMD SEV remote attestation capabilities. Following all of the previous steps, the SMC representative can now perform remote attestation by obtaining the measurement from the server and making sure that the measurement matches the expected value. After that, other SMC members can verify the attestation report. The measurement is obtained using the QEMU Machine Protocol (QMP). QMP is a communication protocol for managing and controlling VMs running on the QEMU emulator.

After the server has been prepared and the SMC provider, the SMC representative, and the SMC members have completed the required steps, the SMC members can send their data and code to the server.

4.2. SVM SM

Upon the secure virtual machine boot and attestation processes, the application script that is installed on the SVM starts the SMC SM. The script exchanges with the SMC BA cryptographic material, as described in Section 3.7, and launches a separate web service for secure multiparty computation using the certificates that were transmitted in the previous exchange. This web service enables the addition of the code for data processing, the data, and keys to decipher the code and data by all SMC consortium members through an API. All of the communication of this web service is protected using TLS. In this section, we outline the key steps in exchanging the necessary keys, input data, and computation results, along with examples of the corresponding API calls. Web service is implemented as a RESTful web application in the Python programming language using the Flask framework. Each functionality is achieved through a specific API call, where parameters are passed either in the JSON format or as form-data. Table 1 shows a list of API calls and the corresponding parameters. The code of the SMC SM is publicly available here [44]. More details about the code encryption key management and the code decryption problems can be found in [46].

Table 1. List of API calls.

API Call Name	API Parameters
add-code	code-file, code-key
add-data	data-file, data-key
add-result-key	result-key
check-computation	/
start-computation	/
collect-results	/

The code provider submits the data processing code script through the *add-code* API call. As this script is encrypted by the code provider, for the application to be able to run it, the code provider must also send the appropriate key, through the same API call. Figure 10 contains the example of the CURL command for the *add-code* API call. Each input party provides the data, encrypted using keys created by each input party. The encrypted data files, as well as the necessary keys for their decryption, are uploaded through the API call *add-data*.

```
curl -X POST -F "code_key=<key_value>" -F
"code_file=@<filepath>" -k https://ipaddress:port/add-code
```

Figure 10. The example of API call.

The encrypted data and code files are stored in a predetermined location in the file system. As the keys are transferred in an unencrypted form, they are stored in the application's memory only, instead of the file system, where they would be exposed to the hardware owner.

A subset of the participants in the SMC, the result parties, are able to receive the computation results. After the computation is finished, the results are encrypted and transmitted to the users in a secure manner. Therefore, the result parties need to provide the encryption keys for the results. These keys are passed to the application through the API call *add-result-key* for each user that expects the results. The computation cannot start until all the necessary data, codes, and corresponding keys are present in the application. Each user can check the computation status at any moment through the API call *check-computation*, which returns the current state of the computation, e.g., NOT_READY, READY, STARTED, DONE.

The SMC representative can initiate the computation when its status is set to READY (all the data, codes, and keys uploaded) through the API call *start-computation*. Once the computation has started, the supplied data processing code is decrypted. The decrypted code is directly run from the application memory. For this, we used the Marshal module for serialization to convert an array of bytes to an executable code object. This module provides functions that can read and write Python values in a binary format that is independent of the machine architecture. Such an approach is crucial to ensure secure code execution without any leakage towards the platform owners.

The SMC SM web service passes the input data in the encrypted form to the decrypted data processing code that performs the computation. The data processing code decrypts the data using supplied keys. In addition, the SMC SM forwards the keys necessary for encrypting the computation results to the data processing code. Once the results are derived, the script encrypts them for each user expecting the results. Users can retrieve the encrypted results through the *collect-results* API call.

4.3. Performance Evaluation

As indicated in the introduction, there are several previous studies that have analyzed the performance penalty of trusted execution environments in different settings (e.g., [11,17]). In addition to these studies, we have tested the performance of machine

learning applications executed in the CoCos.ai environment. We used the SciML-bench tool [47], which provides the most versatile set of features compared to the other scientific machine learning benchmarking approaches [48]. The secure virtual machine we used for testing on a server, mentioned at the beginning of Section 4, had allocated 16 cores and 16 GB of RAM. We tested two different use cases with real-world datasets: improving signal-to-noise ratios of the electron microscope images—*em_denoise* (Test 1) and searching for patterns in X-ray images—*dms_structure* (Test 2). Names written in italic are the names used to invoke the test in the SciML-bench tool. The sizes of the datasets for these two tests are 5 GB and 8.6 GB, respectively. All the tests consist of several phases: parsing input arguments, saving input arguments, loading the dataset, training, and saving training model, history, and performance metrics. We have modified the SciML-bench tool in a way that allows the measurements to be invoked through the CoCos.ai system, that the cryptographic key is passed to the SciML-bench, and that the SciML-bench decrypts and loads the dataset, which is stored encrypted on the disk to preserve data privacy. Table 2 shows the execution times for each of the test phases. The key additional processing time when the CoCos.ai system used is the time to decrypt the encrypted dataset. With the AES-256 that we used, this time was around 1.9% and between 10 and 12% of the total training execution time for Tests 1 and 2, respectively. This percentage depends on the dataset size and the number of files in it, as well as the choice of the training algorithm.

Table 2. SciML performance evaluation.

	Test 1			**Test 2**		
	NO SEV	**SEV**	**SEV-SNP**	**NO SEV**	**SEV**	**SEV-SNP**
Parsing and saving input arguments [s]	0.0011	0.0009	0.001	0.0011	0.0015	0.015
Decrypting datasets [s]	0	108.75	115.83	0	238.31	288.28
Loading datasets [s]	0.0030	0.0023	0.0027	3.505	3.387	3.381
Creating the model [s]	0.009	0.008	0.0097	0.405	0.509	0.457
Training the model [s]	5404.9	5584.7	5857.3	1817.1	1961.6	2079.4
Saving model history and metrics [s]	0.008	0.011	0.009	0.141	0.145	0.186
Total [s]	5404.9	5693.5	5973.2	1821.2	2204.1	2371.8

We have further focused on the training phase because it is a part of the test that, in our measurements, took, in all of the tests, more than 87% of the total test time, it does not depend on the interaction with other computer components (e.g., disk), and is the most impacted by the SEV RAM encryption during dataset processing. For all of the tests, we have calculated the training time slowdown due to the use of SEV or SEV-SNP as $(t_{SEV(SNP)} - t_{noSEV})/t_{noSEV}$, where $t_{SEV(SNP)}$ is the time to complete the training with SEV or SEV-SNP, and t_{noSEV} is the time to complete the training without SEV switched on. The SciML-bench detailed output showed a uniform use of all 16 virtual machine cores with small variations and no use of swap in all of the tests. Table 3 shows the training time measurement results for tests 1 and 2 using real datasets.

Table 3. Training time performance evaluation.

	SEV			**SEV-SNP**		
Test	**Slow Down [%]**	**CPU Load [%]**	**RAM Usage [%]**	**Slow Down [%]**	**CPU Load [%]**	**RAM Usage [%]**
1	3.328	87.372	32.901	8.371	86.003	33.486
2	7.954	90.744	58.155	14.435	88.233	59.864

The performance evaluation results show that there is a difference in the performance of different SEV versions. Additional data integrity protection of SEV-SNP using the new

Reverse Map Table is paid with an additional slowdown compared to the first SEV version. Also, the slowdown is not the same for the tested algorithms and probably depends on the memory access patterns of the algorithm, which should be explored further. However, in all cases, the performance penalty was less than 15%, which is negligible compared to the other solutions for privacy-preserving data processing, as mentioned in Section 1.

5. Conclusions and Further Work

In this paper, we have given an overview of the existing trusted execution environment technologies and analyzed their suitability for secure multiparty computation. We also showed that it is possible to perform secure multiparty computations in the TEE environments and described an original protocol for it. However, truly secure computations of multiple parties on untrusted hardware are possible under a set of assumptions that are a consequence of the technological limitations of the existing secure enclave and system administration procedures. TEE and data-in-use protection technologies have converged towards solutions that protect the memory of the whole virtual machine (AMD SEV and Intel TDX). The technologies are still maturing, with vulnerabilities being removed with every new release. TEE attestation is designed primarily for the single-user outsourced computing use case, and its process produces asymmetric roles of the SMC consortium members. One of the consortium members must be in direct contact with the SMC provider and must tightly control the SVM booting process. Such a setup brings the danger of collusion between the server provider and SMC representative, who can, if acting maliciously, jointly steal the data of other parties. Such an organization is not following the basic SMC principles that all the input parties have the same rights and possibilities. Further, TEE technologies protect data privacy from malicious hypervisors, which, if the SVM is booted properly, cannot see the content of the guest machine RAM. However, there are still attack vectors that are possible through malicious hypervisor configuration and port forwarding. The hypervisor operation must be verifiable, and its setup and network configuration must be protected in a way that does not allow it to tamper with the VM management and port forwarding configuration. SMC deployments that are built on top of bare metal servers and in which all software components installed on the server can be inspected and controlled by the SMC consortium provide more trust than closed-source VM provisioning automation software and cloud provider systems. If closed source hypervisors and closed virtual machine automation systems are used, like in all big cloud providers, the SMC participants must trust that the server/cloud provider will not maliciously alter the operation of the hypervisor and steal the data using, e.g., cuckoo attack. This is also directly against the basic principle of secure multiparty computation—not trusting the computation provider. If SMC partners have to trust the server provider, then it is easier not to use confidential computing technologies at all. Finally, the computation code and the operation of the SVM must be closely monitored to ensure to all the SMC consortium members that there is no data leakage by the computation code prepared by the code provider (e.g., disk writes or network communication). The computation performance obtained when TEEs are used is a big reason for the further development of confidential computing technologies. On the other hand, the set of issues for the SMC implementations listed in this section indicates that, in the future, we can expect to see research attempts in several different directions aiming to solve the listed issues. There is also a potential for new commercial services supporting confidential computing and secure multiparty computation (e.g., multiparty attestation, software auditing, secure virtual machine monitoring solutions, etc.).

Author Contributions: D.M.: investigation, software, visualization, attestation, and writing-original draft; A.M.: investigation, software, and writing—original draft; M.V.: investigation, software, and writing-original draft; Ž.S.: investigation, methodology, software, visualization, attestation, and writing-original draft; P.V.: conceptualization, methodology, visualization, supervision and writing—original draft. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Serbian government's Innovation Fund (Collaborative Grant Scheme Program, project ID 50314).

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: We want to thank our industry partner, UV Consult, for their cooperation, fruitful discussions, and support during the work on the CoCos.ai project and efforts in testing our Secure Multiparty Computation solution.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rashid, F.Y. The Rise of Confidential Computing: Big Tech Companies Are Adopting a New Security Model to Protect Data While It's in Use—[News]. *IEEE Spectr.* **2020**, *57*, 8–9. [CrossRef]
2. Evans, D.; Kolesnikov, V.; Rosulek, M. A Pragmatic Introduction to Secure Multi-Party Computation. *Found. Trends Priv. Secur.* **2018**, *2*, 70–246. [CrossRef]
3. Yang, Y.; Huang, X.; Liu, X.; Cheng, H.; Weng, J.; Luo, X.; Chang, V. A Comprehensive Survey on Secure Outsourced Computation and Its Applications. *IEEE Access* **2019**, *7*, 159426–159465. [CrossRef]
4. Mo, J.; Gopinath, J.; Reagen, B. HAAC: A Hardware-Software Co-Design to Accelerate Garbled Circuits. In Proceedings of the 50th Annual International Symposium on Computer Architecture, Orlando, FL, USA, 17–21 June 2023; pp. 1–13.
5. Kairouz, P.; McMahan, H.B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A.N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.* **2021**, *14*, 1–210. [CrossRef]
6. Peisert, S. Trustworthy Scientific Computing. *Commun. ACM* **2021**, *64*, 18–21. [CrossRef]
7. AMD Secure Encrypted Virtualization (SEV). Available online: www.amd.com/en/developer/sev.html (accessed on 30 January 2024).
8. Kaplan, D. Protecting VM Register State with SEV-ES. *White Pap.* **2017**, *13*. Available online: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/Protecting-VM-Register-State-with-SEV-ES.pdf (accessed on 30 January 2024).
9. Sev-Snp, A.M.D. Strengthening VM Isolation with Integrity Protection and More. *White Pap.* **2020**, *53*, 1450–1465.
10. Intel® Trust Domain Extensions (Intel® TDX). Available online: https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html (accessed on 30 January 2024).
11. Akram, A.; Giannakou, A.; Akella, V.; Lowe-Power, J.; Peisert, S. Performance Analysis of Scientific Computing Workloads on General Purpose TEEs. In Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, 17–21 May 2021; pp. 1066–1076.
12. Laud, P.; Kamm, L.; Veeningen, M. *Applications of Secure Multiparty Computation*; IOS Press: Amsterdam, The Netherlands, 2015; Volume 13.
13. Bogdanov, D.; Talviste, R.; Willemson, J. Deploying Secure Multi-Party Computation for Financial Data Analysis: (Short Paper). In Proceedings of the Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, 27 February–2 March 2012; Revised Selected Papers 16. pp. 57–64.
14. Lindell, Y. Secure Multiparty Computation. *Commun. ACM* **2021**, *64*, 86–96. [CrossRef]
15. Sardar, M.U.; Musaev, S.; Fetzer, C. Demystifying Attestation in Intel Trust Domain Extensions via Formal Verification. *IEEE Access* **2021**, *9*, 83067–83079. [CrossRef]
16. Cheng, P.-C.; Ozga, W.; Valdez, E.; Ahmed, S.; Gu, Z.; Jamjoom, H.; Franke, H.; Bottomley, J. Intel TDX Demystified: A Top-Down Approach. *arXiv* **2023**, arXiv:2303.15540.
17. 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, Los Angeles, CA, USA, 2 June 2018; Association for Computing Machinery: New York, NY, USA, 2018.
18. El-Hindi, M.; Ziegler, T.; Heinrich, M.; Lutsch, A.; Zhao, Z.; Binnig, C. Benchmarking the Second Generation of Intel SGX Hardware. In Proceedings of the 18th International Workshop on Data Management on New Hardware, Philadelphia, PA, USA, 13 June 2022; Association for Computing Machinery: New York, NY, USA, 2022.
19. Will, N.C.; Maziero, C.A. Intel Software Guard Extensions Applications: A Survey. *ACM Comput. Surv.* **2023**, *55*, 1–38. [CrossRef]
20. Bahmani, R.; Barbosa, M.; Brasser, F.; Portela, B.; Sadeghi, A.-R.; Scerri, G.; Warinschi, B. Secure Multiparty Computation from SGX. In Proceedings of the International Conference on Financial Cryptography and Data Security, Sliema, Malta, 3–7 April 2017; pp. 477–497.
21. Zheng, W.; Wu, Y.; Wu, X.; Feng, C.; Sui, Y.; Luo, X.; Zhou, Y. A Survey of Intel SGX and Its Applications. *Front. Comput. Sci.* **2020**, *15*, 153808. [CrossRef]
22. Volgushev, N.; Schwarzkopf, M.; Getchell, B.; Varia, M.; Lapets, A.; Bestavros, A. Conclave: Secure Multi-Party Computation on Big Data. In Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, 25–28 March 2019; Association for Computing Machinery: New York, NY, USA, 2019.
23. Pandey, P.; Lu, J. Secure Shared Data in Use with Confidential Space. Available online: https://codelabs.developers.google.com/codelabs/confidential-space#0 (accessed on 30 January 2024).

24. Gazdag, V. Confidential Space Security Review. Available online: https://research.nccgroup.com/2022/12/06/public-report-confidential-space-security-review/ (accessed on 30 January 2024).
25. Chen, K. Confidential High-Performance Computing in the Public Cloud. *IEEE Internet Comput* **2023**, *27*, 24–32. [CrossRef]
26. Wilke, L.; Wichelmann, J.; Morbitzer, M.; Eisenbarth, T. SEVurity: No Security Without Integrity—Breaking Integrity-Free Memory Encryption with Minimal Assumptions. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020. [CrossRef]
27. Hetzelt, F.; Buhren, R. Security Analysis of Encrypted Virtual Machines. In Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Xi'an, China, 8–9 April 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 129–142.
28. Werner, J.; Mason, J.; Antonakakis, M.; Polychronakis, M.; Monrose, F. The SEVerESt Of Them All: Inference Attacks Against Secure Virtual Enclaves. In Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, Auckland, New Zealand, 9–12 July 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 73–85.
29. Li, M.; Zhang, Y.; Lin, Z. CrossLine: Breaking "Security-by-Crash" Based Memory Isolation in AMD SEV. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 15–19 November 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 2937–2950.
30. Li, M.; Zhang, Y.; Lin, Z.; Solihin, Y. Exploiting Unprotected I/O Operations in AMDs Secure Encrypted Virtualization. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 1257–1272.
31. Morbitzer, M.; Huber, M.; Horsch, J.; Wessel, S. SEVered: Subverting AMD's Virtual Machine Encryption. In Proceedings of the 11th European Workshop on Systems Security, Porto, Portugal, 23–26 April 2018; Association for Computing Machinery: New York, NY, USA, 2018.
32. Morbitzer, M.; Huber, M.; Horsch, J. Extracting Secrets from Encrypted Virtual Machines. In Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, Richardson, TX, USA, 25–27 March 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 221–230.
33. Buhren, R.; Jacob, H.-N.; Krachenfels, T.; Seifert, J.-P. One Glitch to Rule Them All: Fault Injection Attacks Against AMD's Secure Encrypted Virtualization. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 15–19 November 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 2875–2889.
34. Gu, J.; Wu, X.; Zhu, B.; Xia, Y.; Zang, B.; Guan, H.; Chen, H. Enclavisor: A Hardware-Software Co-Design for Enclaves on Untrusted Cloud. *IEEE Trans. Comput.* **2021**, *70*, 1598–1611. [CrossRef]
35. OVMF. Available online: https://github.com/tianocore/tianocore.github.io/wiki/OVMF (accessed on 30 January 2024).
36. Slemmer, A.; Deml, S. Swiss Cheese to Cheddar: Securing AMD SEV-SNP Early Boot. Available online: https://www.decentriq.com/article/swiss-cheese-to-cheddar-securing-amd-sev-snp-early-boot (accessed on 30 January 2024).
37. QEMU. A Generic and Open Source Machine Emulator and Virtualizer. Available online: https://www.qemu.org/ (accessed on 30 January 2024).
38. Parno, B. Bootstrapping Trust in a "Trusted" Platform. In Proceedings of the HotSec, San Jose, CA, USA, 29 July 2008.
39. Chen, L. *Recommendation for Key Derivation Using Pseudorandom Functions*; US Department of Commerce, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2008.
40. SEV Secure Nested Paging Firmware ABI Specification. Available online: https://www.amd.com/system/files/TechDocs/56860.pdf (accessed on 30 January 2024).
41. Secure VM Service Module for SEV-SNP Guests. Available online: https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/58019.pdf (accessed on 30 January 2024).
42. Babar, Y. Initramfs. In *Hands-on Booting: Learn the Boot Process of Linux, Windows, and Unix*; Apress: Berkeley, CA, USA, 2020; pp. 207–234. ISBN 978-1-4842-5890-3.
43. ProVerif: Cryptographic Protocol Verifier in the Formal Model. Available online: https://bblanche.gitlabpages.inria.fr/proverif/ (accessed on 30 January 2024).
44. Cocosapi. Available online: https://github.com/cocosetf/cocosapi/ (accessed on 30 January 2024).
45. SEV-Tool. Available online: https://github.com/AMDESE/sev-tool (accessed on 30 January 2024).
46. Vukasovic, M.; Miladinovic, D.; Milakovic, A.; Vuletic, P.; Stanisavljevic, Z. Programming Applications Suitable for Secure Multiparty Computation Based on Trusted Execution Environments. In Proceedings of the 2022 30th Telecommunications Forum (TELFOR), Belgrade, Serbia, 15–16 November 2022; pp. 1–4.
47. Thiyagalingam, J.; Leng, K.; Jackson, S.; Papay, J.; Shankar, M.; Fox, G.; Hey, T. SciMLBench: A Benchmarking Suite for AI for Science. 2021. Available online: https://github.com/stfc-sciml/sciml-bench (accessed on 30 January 2024).
48. Thiyagalingam, J.; Shankar, M.; Fox, G.; Hey, T. Scientific Machine Learning Benchmarks. *Nat. Rev. Phys.* **2022**, *4*, 413–420. [CrossRef]