# Architecture and Advancement in Virtualization of TPM

Hem Parekh *hp5643*
Nisarg Dave *nd9565*
Praj Shete *ps7600*

## 1 Introduction

A Trusted Platform Module (TPM) is one of the core components of Trusted Computing. It is the root of trust in a computer. One of the prime facts about the TPM is that it is a hardware-based approach. Hence, virtual TPM, or vTPM, is required to extend the functionalities of TPM to a virtual machine. One of the significant services that use virtual machines is cloud service. In this research survey, we will explore the architecture of vTPM in virtual machines and cloud environments along with its advancements. Also, we will focus on the security aspect of key sharing during the creation and migration of the vTPM in virtual and cloud environments. Despite all the advancements in the vTPM, there are security flaws that are still not addressed. Hence, we will also elaborate on various security flaws and attacks on vTPM.

## 2 Literature Review

There has been a development in the architecture of vTPM since 2006. A research 'Virtualizing the Trusted Platform Module'[16], gave the very first architecture of a virtual TPM. They showed how they use a hardware TPM's functionality and create instances of vTPM to give each virtual machine its instance of vTPM which has the same functionality as a hardware TPM. In this survey, we will be going over the architecture designed by the authors of the research paper 'Virtualizing the Trusted Platform'. Another development of virtual TPM has been done considering its usage in cloud services. A research paper 'A cloud Architecture of Virtual Trusted Platform Module', showed how the functionalities of a TPM can be virtualized to make use of it for Cloud-based virtual machines. This survey also covers the mentioned research paper and their way of implementing vTPM.

Virtual Trusted Platform Modules are the only root of trust for all cloud-based computing devices. However, the vTPM being a software-based component cannot entirely replicate the security provided by the hardware TPM and thus face some security challenges. The first challenge is the migration of VMs which was discussed and addressed in paper [9]. We will study how the data related to the vTPM instance will be migrated through an unsecured channel. The other security challenges that we will discuss in our study are mentioned in the paper [21]. This paper discusses and provides a state-of-the-art solution to the features that were missing in vTPM as compared to that of hardware-based TPM. Current vTPM only follows TPM 1.2 standards and lacks support for TPM 2.0. [11] outlined a few federated identity issues and showed how FIM, or Federated Identity Management, based on standards, allows for and facilitates joining federated organizations in terms of sharing user identity attributes, streamlining authentication, and allowing or denying using service access requirements. SSO is described as employing it to allow users to log in to subsequent service providers inside a federated identity after completing a single authentication with their home identity provider. In a federated identity environment, there are various ongoing issues and worries such as user identity information exploitation through SSO capabilities in service providers and identity providers, identity theft of users, service providers and identity providers, and user reliability. This paper proposes how the A Trusted Platform Module can be used by SPs, IDPs, and consumers of cloud computing to authenticate their hardware. It is possible to do platform authentication because each TPM (VTPM) chip or Virtual TPM (VTPM) has an exclusive and secret RSA key burnt into it when it is created. Each system should be independent of any trusted third party on the trusted platform. In addition to independence, each system should have the ability to clearly identify users who can be trusted both within and across business boundaries.

## 3 vTPM Concepts

When it comes to virtualizing the Trusted Platform Module for an x86 architecture, several ideas were floated. When it comes to the level of virtualization of the TPM, we may divide it into three categories: instruction-level virtualization, hardware-level virtualization, and operating program library virtualization [8][16][18][19]. In addition, we were able to divide hardware-level virtualization into three categories: total virtualization, near-resembling virtualization, and hardware-assisted virtualization. All of these categories made use of every computer component, including the CPU, memory, and so on. We shall go through each category given in depth in this study survey document.

### 3.1 Need of vTPM

Virtualization of a Trusted Platform is providing all the functionality that a hardware TPM provides to virtual computing platforms. This functionality was needed so that all the virtual machines can have improved security, and can also be trusted when it comes to their credibility. In this research paper, we concentrated on three main features that a virtual trusted platform module should provide.

- Virtualization of a Trusted Platform Module is nothing but virtualization of an Input/Output device.
- All the virtual machines that are in the need of a virtual TPM should be able to use all the functionalities that a vTPM provides.
- And lastly, as mentioned all virtual machines should have improved security, and any user should be able to trust their credibility.

### 3.2 Basic Trusted Platform Module Architecture

When it comes to different types of hardware virtualization, what kind of implementation takes place depends on what kind of Input/Output device implementation is done and how the implementation of the Virtual Machine Monitor is done. We are already familiar with what TPM is and how is it implemented, so how it is made use of in virtualization depends on the functions that are needed.

#### 3.2.1 Full Virtualization

When it comes to Full Virtualization of the Trusted Platform Module, in this the Virtual Machine Monitor(VMM) is capable of virtualizing all the virtual machines in the same environment as the Hardware TPM. Because of this feature, all the various virtual machines or Guest OS have a single vTPM that is usually controlled by the Hardware TPM and is present inside the VMM[19]. Usually, all of the Input/Output commands are also present inside the VMM but in the case where certain instructions aren't then in such scenarios binary code translator is used.

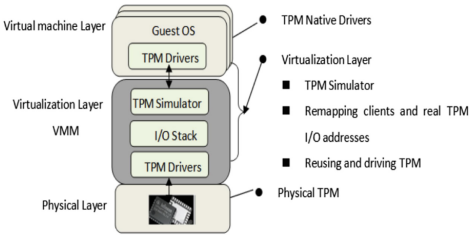


Figure 1: TPM Stand-Alone Model[19]

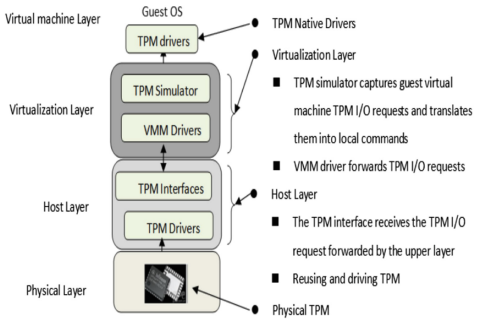


Figure 2: TPM Hosted-Based Model[19]

Now, full Virtualization can be further categorized into two different categories: the Independent TPM model, shown in Fig1, and Hosted-based vTPM, shown in Fig2.

#### 3.2.2 Para-Virtualized Trusted Platform Module

This type of virtualized TPM acts differently than the fully virtualized TPM. Since in this scenario, the VMMs do not have access to privileged instructions directly there needs to be a channel through which they can access the functionalities of TPM. This channel is provided by using coded front-end and coded back-end drivers which allow them to make requests and then the front and back-end drivers can work to get access to the functionalities. Fig3 Shows the basic architecture of the Para Virtualized TPM. Figure 3 shows how a basic architecture of para-virtualized TPM.

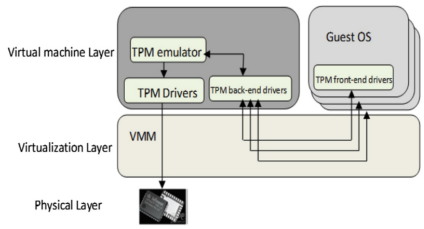


Figure 3: Para-Virtualized Trusted Platform Module[16]

### 3.2.3 Hardware-Assisted Virtualized TPM

This is a type of virtualization where the kernel of the Virtual machine operating system needs no modification. This type of virtualization takes the help of the hardware component of the host machine to implement virtualization. These hardware components usually involve Intel's VT-x technology or AMD's AMD-v technology. Figure 4 shows how hardware-assisted virtualization of the Trusted Platform Module takes place.

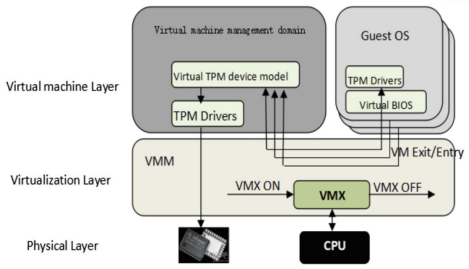


Figure 4: Hardware Assisted vTPM[16]

These were the basic classification of any virtualized trusted platform module. We saw how this classification is mainly divided into 3 and how each classified model works. Now, we will see the different classifications of virtualized Trusted Platform Module based on the technology used.

## 4 Classification of vTPM module based on Technology

Again based on the technology that is used to implement any vTPM, there can be 3 different classifications: Software mode vTPM, Hardware Sharing Mode, and Aggregation Method. [8][18][19]

### 4.1 Software Mode vTPM Method

To put it simply, the Virtual Machine Monitor (VMM) implements a virtualized instance of a trusted platform module and ensures that each virtual machine has its own vTPM instance in order to offer a trusted execution environment (TEE). Whenever a TPM's capability is required, it is delivered by this freshly constructed instance of the trusted platform module via the VMM. Figure 5 depicts a rudimentary implementation of this strategy. The fundamental advantage of this technology is that it supports all of the basic vTPM models stated above, as well as full virtualization, para-virtualization, and hardware-assisted virtualization. Other benefits include ease of deployment, flexibility, and extremely strong performance; in addition, it facilitates migration.

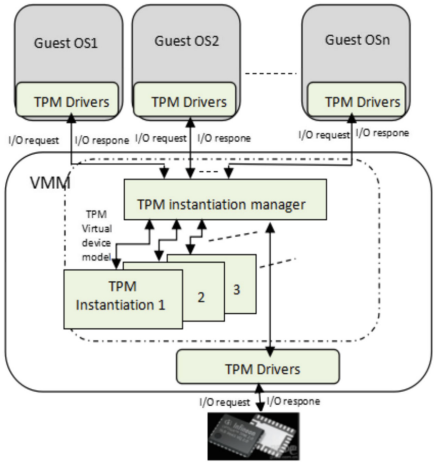


Figure 5: Software-Based vTPM[17]

### 4.2 Hardware Implementation Method

This approach relies on hardware TPM sharing[2]. This sharing works because it makes use of time. All of the virtual computers employ hardware TPM by sharing it on time. When there are numerous virtual machines, the VMM creates a request queue[2] containing all the Input/output requests and then schedules them according to regulations. Once all requests have been completed, the VMM places all replies in a new queue called the response queue[2] and returns them to the appropriate virtual machines. Figure 6 depicts a simplified form of the hardware-based implementation paradigm.

### 4.3 Aggregation Method

This sort of virtualization employs a TPM known as the VF-vTPM[2]. This VF-vTPM includes all TPM features and is capable of assigning vTPM to any virtual machine that demands it. The key advantages of this technique are that it delivers very strong security and fast performance while also being highly scalable. The main issue is that

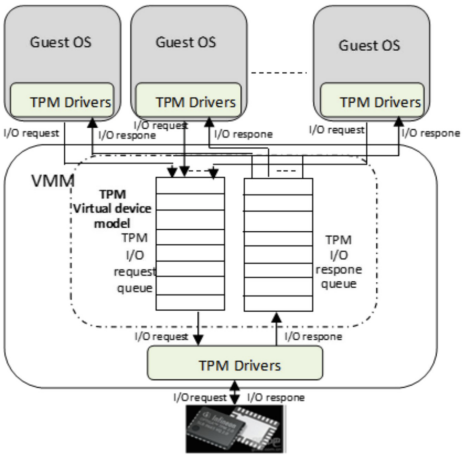


Figure 6: Hardware-Based vTPM[2]

this approach is not straightforward to implement, and TPM1.2 and TPM2.0 do not currently support it since the LPC bus inside TPM does not enable Single Root I/O virtualization [2].

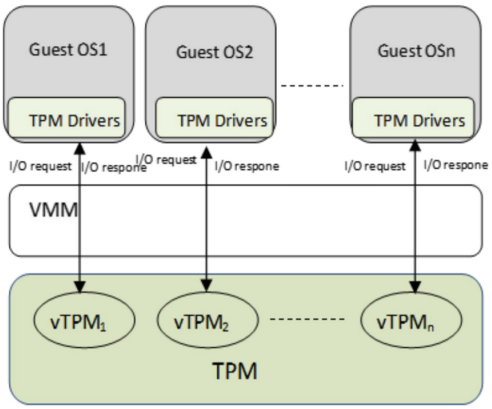


Figure 7: Aggregation Method vTPM[2]

# 5 Classification of vTPM based on Key Technology

So far, we've seen the overall architecture of vTPM, as well as how distinct vTPM are classed depending on the technology employed. We also learned about the many benefits and how one model differs from the others. We realize that key creation and management are important functions of any trusted platform module. When it comes to vTPM, a few new features are included, such as support for key migration and extending the trust of certificates and keys.

## 5.1 Basic System Architecture available of vTPM

When it comes to virtualized trusted platform modules, this part addresses the entire system architecture. When it comes to software, hardware, and aggregated simulation vTPM system architecture, this part provides the entire component architecture and functional modules[2].

### 5.1.1 Software Simulation of vTPM System Architecture

This idea was first introduced by S Berger in his research paper [19] and figure 8 shows the architecture that was proposed. This model contains vTPM, vTPM manager, Client-side TPM driver, and Server-side. In this architecture, it is the vTPM manager that is responsible for all the vTPM instance lifecycles [19]. This lifecycle includes the creation, deletion, migration, and restoration of vTPM instances. It is also responsible for managing all the vTPM requests generated by virtual machines and making sure the the request reaches correct vTPM instance. The client-side driver and server-side driver usually communicate via Virtual Machine Monitor (VMM).

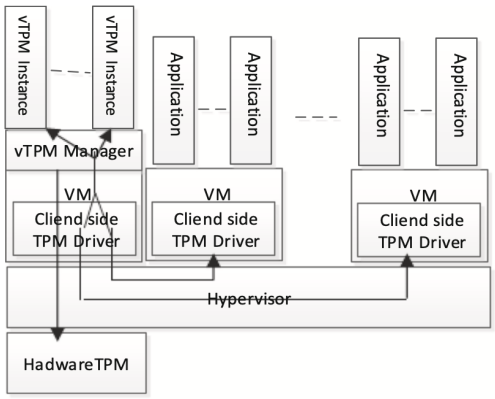


Figure 8: vTPM Architecture[19]

In 2007, another software-simulated design was presented [15]. This architecture was known as GVTPM, and it comprised primarily of a GVTPM manager, a factory, and a Protected Storage service. The GVTPM manager is in charge of delivering all vTPM lifecycle features in this architecture. Another component, the GVTPM factory, is in charge of supplying the GVTPM manager with all of the tools needed to build and delete GVTPM services[15]. The key advantage of both architectures is that the vTPM manager, vTPM, GVTPM, and GVTPM manager can only be accessed with administrator permissions, making them typically secure. The 'General architecture of GVTPM' is seen in Figure 11.

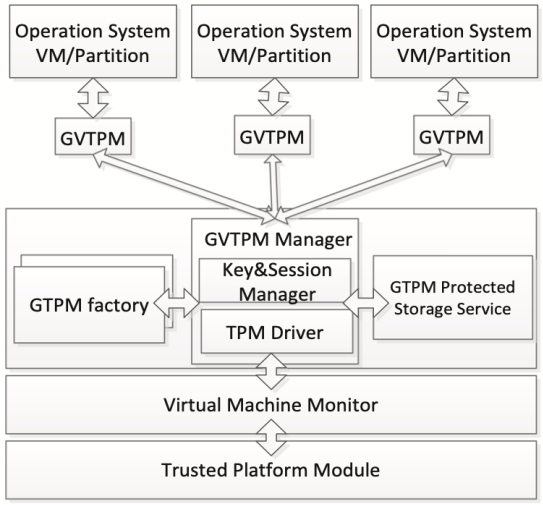


Figure 9: GVTPM Architecture[15]

Because the above-mentioned basic vTPM design did not place a high priority on security, different architectures were developed to improve anti-privilege security [19]. This design only had one feature: an external security co-processor, which was originally IBM's PCIVCC. This co-processor is capable of offering the highest level of security. Figure 12 depicts the architecture included in IBM's PCIVCC.

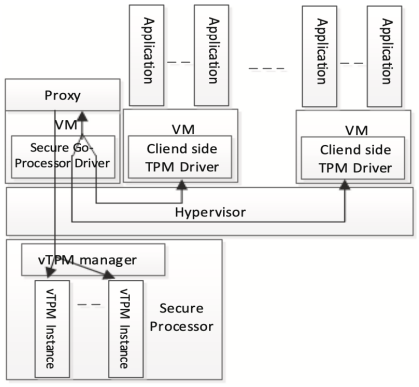


Figure 10: vTPM Architecture with PCIVCC[19]

Another security-based vTPM architecture was proposed in a research paper [12], which featured Xen. This architecture largely focuses on segregating the vTPM instance from all the privileged domains. It added a new functionality known as vTPMD Deamon. In this design, all apps in the DOMB may send TPM instructions to the vTPM instance via the vTPM manager and vTPMD. Figure 11 depicts vTPM's Xen architecture. One of the most significant advantages of this design was its capacity to successfully increase the vTPM's ability to withstand any shared security risks, hence reducing the strain on the Domain0 and eventually allowing vTPM migration.

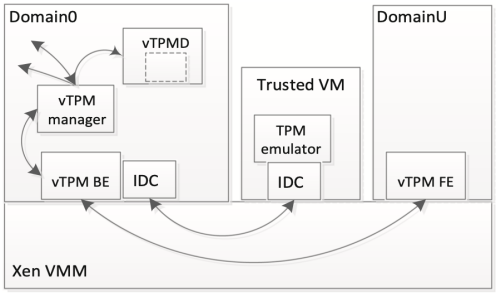


Figure 11: Xen Architecture of vTPM[12]

The previous architecture kept everything, including the vTPM instance and manager, in the privilege domain. The second architecture introduced us to a revolutionary design that effectively isolated the vTPM instance from the privilege domain while maintaining its security. Another research published in 2008 [8] was able to successfully separate the vTPM manager, vTPM instance, and privilege domain without affecting its security. In this design, the "Builder Domain" [8] is built on a MiniOS named DomB, and the vTPM manager and instance are separated from DomB. DomB was made up of TPM's native driver and virtual platform configuration store. The fundamental advantage of this paradigm was that the DomB was able to successfully isolate vTPM instances from the privilege domain while also keeping the vTPM instance lifecycle separate from the privilege domain. The DomB architecture is seen in Figure 12. Because a single DomB was isolated from the rest of the architecture and maintained in the privileged domain, the aforementioned architecture is also known as a 'Single-Isolated Domain.'

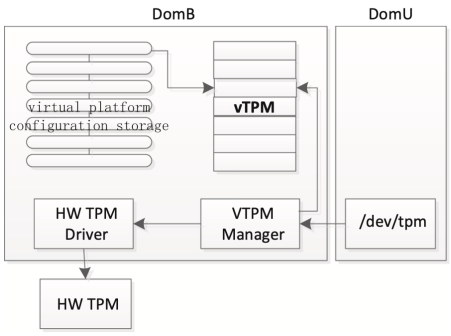


Figure 12: DomB Architecture of vTPM[8]

OpenTC was able to offer an architecture that specified the general design of the virtual trusted platform module back in 2009. Everything from the vTPM data structure through vTPM and VM binding were specified in this research study. Aside from that, the article developed safe migration mechanisms for the vTPM and its associated VM [8]. Figure 15 depicts OpenTC's introduction to the architecture. DomB and DomU were separate domains in the design. Instead of vTPM instances and vTPM managers inside DomB, this design relies on VP-Builder, HIM (Trusted Virtual domain integrity management), and CIS (Virtual Domain Configuration)[8] to handle the construction of virtual trusted domains. DomU, on the other hand, is where the vTPM model may be found. This vTPM design, which consists of two separate domains, was also capable of increasing performance speed and improving security.

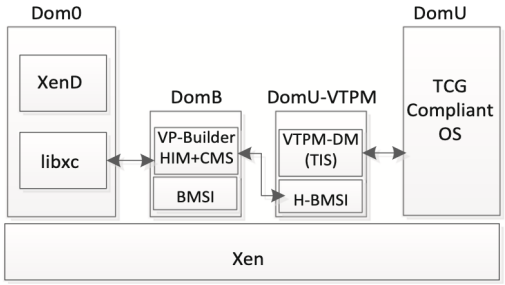


Figure 13: Double Isolated Domain[8]

During the same year, China developed another model [12] that solved the problem of the TCB being too big in prior architectures. They advocated isolating vTPM, vTPM managers, and TPM native drivers by creating a separate domain with administrative access. The study provided two reasons for doing so in their recommended architecture. The first was that all vTPM-related components were secured against unwanted access. The second reason was that they were able to change the TPM access method while simultaneously protecting the DomA. Figure 14 shows the model proposed in the research paper. [8][16][18][19]

### 5.1.2 Hardware Technology-based vTPM Architecture

In this part, we will go through all of the planned vTPM architectures that will employ hardware or even share hardware.

The original design, presented in 2008, offered a mechanism for creating a vTPM instance using hardware components[10]. The suggested architecture included HyperVTPMDriver, HyperCallDispatcher, TPM service,

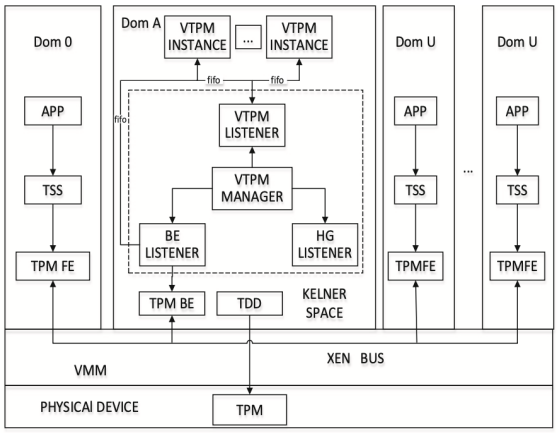


Figure 14: Domain -A Based vTPM[12]

and TPM Driver. The TPM Service, which is in charge of supplying all TPM features to each virtual machine, is at the heart of this design. Because there is only one physical TPM in the provided architecture, it is critical that all TPM attributes are shared by the VM and that its consumption is planned. One shortcoming of this architecture is that it was never put into practice. However, this design offers various advantages, including high security. Figure 15 shows the architecture that was presented by this research.

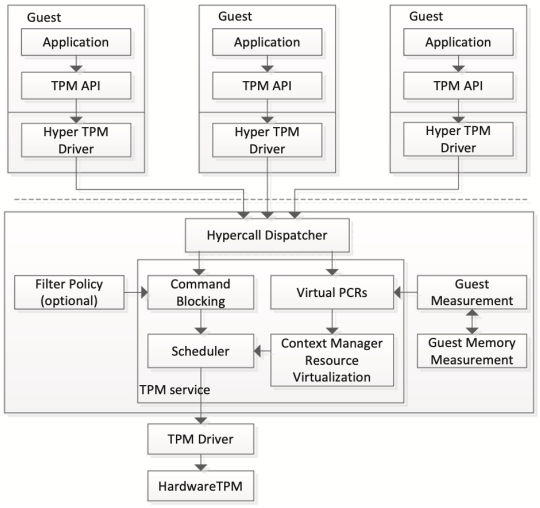


Figure 15: Basic Hardware Shared TPM Architecture[19]

During the same year, another study [18] developed an enhanced version of the prior architecture. It concentrated on increased security. This design might enable multi-client virtual machine sharing [2]. Non-volatile storage also featured two distinct keys, Active TPM-

Control and Root-Data [2]. This architecture[18] might potentially provide a hardware safety mechanism for the CPU. It supported a ring architecture, with the VM running on CPU ring 1 and the VMM running on CPU ring 0. Another advantage of this design was that because it could manage and schedule TPM, the VMM required less software code. Figure 16 depicts the architecture that was proposed by the study[18].

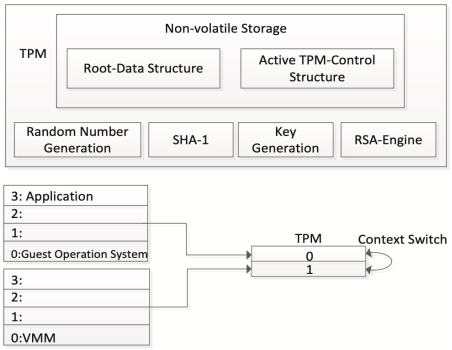


Figure 16: Hardware Shared TPM Architecture with CPU Protection[2]

Another patented architecture demonstrated how the Trusted Platform Module may be used to share data processing systems[3]. TPM Software Stacks, TPM Device Drivers, LPAR, INPUT/OUTPUT QUEUES, and other components were incorporated in this design. LPAR TPM TSS and TPM Device Drivers performed the functionality of providing an interface and drivers for the application. Requests and answers were queued and processed according to a plan. All TPM functionalities were handled by logical TPM (LTPM). Figure 17 shows the architectural design of hardware shared vTPM with shared Data Processing.

In 2011[10], a new design was presented that took into account the most recent technology, such as mobile computing and cloud computing. Dynamic-Context TPM was the name given to this design. The architecture was made up of FPGAs and several peripheral control interfaces [14]. This design also had several physical TPMs. dcTPM controlled the entire context of either vTPM or physical TPM. This design is said to be adaptable, fits user requirements, and offers strong security. Figure 18 shows dcTPM architecture.

### 5.1.3 Aggregated vTPM Architecture

[19]

There has been relatively little development in the subject of aggregated vTPM architecture in recent years. Despite the broad generality, the core design is based on the virtualization of Input/Output Hardware Devices

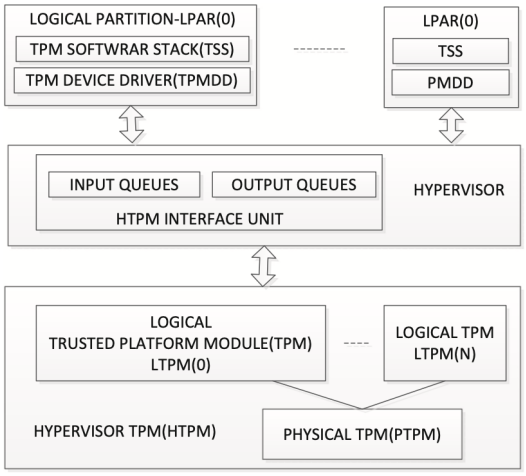


Figure 17: Hardware Shared TPM Architecture with Shared Data Processing[3]

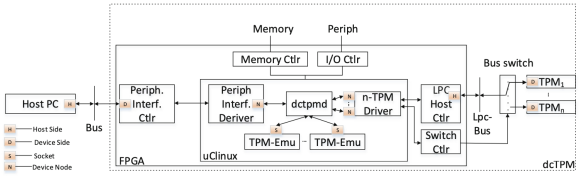


Figure 18: dcTPM[10]

[19]. There have been a few studies, such as SR-IOV [19], that discuss the virtualization of hardware protocols. These protocols enabled efficient sharing of hardware resources, allowing for theoretically good I/O performance.

TCG (Trusted Computing Group) was said to have just given out the functionality and specifications of TPM but did not specify to what degree TPM features may be used. The TCG VPWG (Virtualized Platform Work Group) is actively researching the protocols established by SR-IOV and working with the architecture of Aggregated vTPM.

## 6 vTPM in Cloud Environment

One of the major applications of vTPM is its use in cloud environments. There are many services and devices hosted on the cloud platforms that can benefit from the functionalities of a TPM, and vTPM provides them with that. vTPM allows the use of TPM functionalities to those devices that do not have an actual physical TPM chip. There are certain features that need to be present in such a vTPM. Some of these features are:

- Any device on the cloud should be able to utilize vTPM in such a way that it can use the functionalities of a physical TPM as much as possible
- Multiple devices on the cloud should be able to access the physical TPM functionalities at the same time
- Devices should be able to access their keys and data even if they move to a new platform
- The migration of keys should be secure.

There are various architectures of vTPM proposed for cloud environments throughout the years that try to incorporate the above-mentioned features into their architecture. We have discussed some of those models in detail in this section.

## 6.1 A Cloud Architecture of vTPM

The architecture discussed in this part is one of the earliest and basic ones. The major components in this architecture are vTPM service, user management component, cryptographic service, and a number of physical TPMs and their management [7]. This pictorial representation of this architecture is shown in Figure 19.

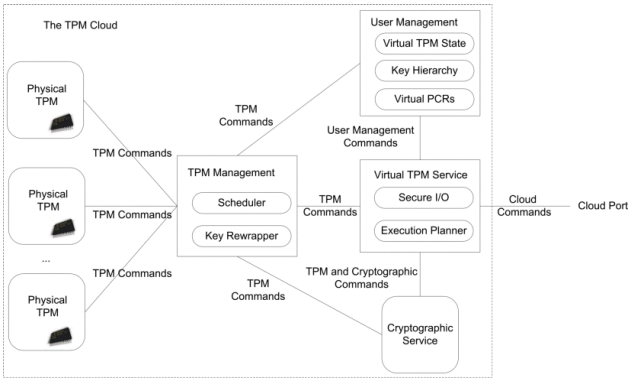


Figure 19: Cloud Architecture for vTPM [7]

The general concept of this architecture is that the vTPM service communicates with the cloud users through the cloud port. Furthermore, cloud users can also send pre-defined cloud commands to use TPM services. Some examples of the TPM services are registering new TPM instances and using them to create keys and seal data. Lastly, it is ensured that the migration of keys and the service results in the exchange for cloud commands are secure.

The secure I/O is responsible for the secure communication between the cloud user and the TPM cloud. As seen in Figure 20, cloud commands that are entered by

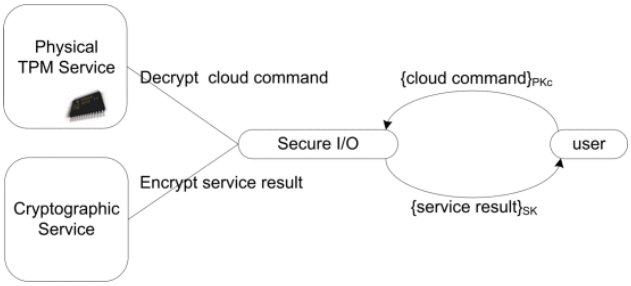


Figure 20: The secure I/O communication model [7]

the users are encrypted with the public key of the TPM cloud, hence only the TPM cloud can decrypt it.

The command can be decrypted only with the private key of the TPM cloud. Hence, we require a physical TPM for this purpose, as no secret should leave the TPM chip. The command is decrypted using the cryptographic service and the required service is carried out. The reason for the commands to be encrypted is that the commands contain a secret key, generated at the user end, as an argument. This secret key is later used in encrypting the service results.

The user management component manages the state of users and maintains a TPM instance for each user. TPM instance includes vTPM state, key hierarchy, and set of virtual PCRs.

The physical TPM generates a virtual endorsement key (EK) and a virtual storage root key (SRK) with its SRK as the parent key. Also, the virtual EK is created as a binding key.

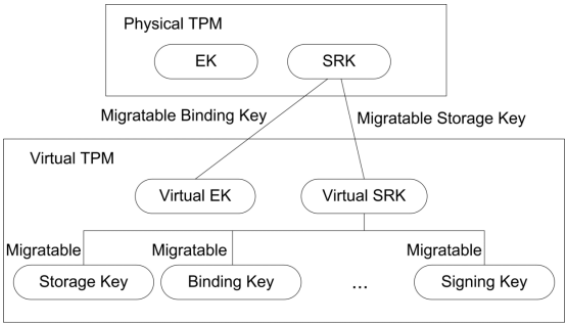


Figure 21: The key heirarchy of vTPM [7]

As evident from Figure 21, all keys in the vTPM are migratable. Hence, they are not bound to a physical TPM.

Virtual PCRs act similarly to PCRs in physical TPM. However, since virtual PCRs are not in physical TPM, they need to be encrypted. They are encrypted using the virtual EK public key. When needed, they are decrypted using the virtual EK private key and then encrypted again after use. Also due to this, PCRs need to be re-written

before sending commands to physical TPM, since certain physical TPM functionalities need PCR readings for execution. This process is carried out by a cryptographic service.

However, PCRs can only be extended and not all of them are resettable. If a command needs the accurate values of PCRs, then this command is implemented in the cryptographic service. If the command only refers to the state of PCRs, then this command is executed in physical TPM after some rewriting.

For scalability, the TPM cloud supports a number of physical TPMs. The physical TPM management component coordinates all physical TPMs to execute multiple TPM commands in parallel. The scheduler decides which command to be executed by which TPM so as to minimize and distribute the load.

The key wrapper is responsible for transferring keys from one physical TPM to another. The key present on TPM-A can be transferred to TPM-B by wrapping the key with the public SRK of TPM-B and then migrating it. After reaching TPM-B, these keys are now unwrapped and loaded for use.

## 6.2 cTPM

The architecture proposed uses TPM 2.0 functionalities and further extends it by adding the ability to share a primary seed with the cloud, and the ability to access cloud-hosted non-volatile (NV) storage [4]. In this subsection, we will discuss its architecture.

Each device will have its own unique cTPM with a unique primary seed that is shared with the cloud and is used to derive additional keys. If the owner has multiple devices, then all those devices will have their keys tied to the owner's credentials. This way, the cloud can offer cTPM services that create shared keys across all devices that are owned by the same user.

There are two components of cTPM. One is running on the device and the other is running on the cloud. Both components implement the full TPM 2.0 software stack with additional cTPM features. As shown in Figure 22, the cTPM software stack runs in the TPM chip on the device side, while on the other hand, it runs in a VM on the cloud side.

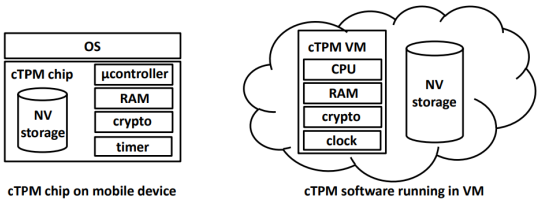


Figure 22: The high-level architecture of cTPM [4]

The cTPM uses the cloud primary seed to generate an asymmetric storage root key, called the Cloud Root Key (CRK), and also a symmetric communication key, called Cloud Communication Key (CCK). These key derivations occur on both the device end and cloud end independently. Since the key derivations are semantic, both platforms end up having identical keys. The CRK, similar to SRK, is used to encrypt all objects that are protected within the cloud domain. CCK is used to encrypt data that is exchanged with the cloud.

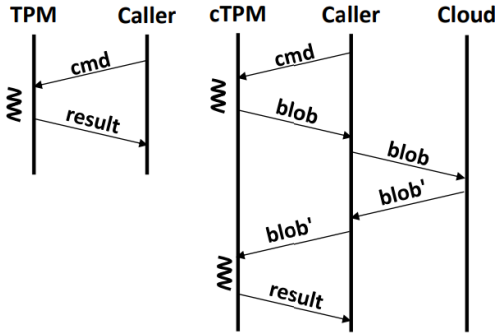


Figure 23: Steps of Asynchronous Communication for executing a command [4]

cTPM supports asynchronous communication between the local cTPM and the cloud for the commands that require remote NV access. As depicted in Figure 23, when a command that demands access to remote NV is received, cTPM returns an encrypted blob. The caller sends this blob to the cloud, and the cloud returns another encrypted blob reply to the caller after accepting the initial blob. The caller now passes this blob to the cTPM and then the command completes. However, all those commands that do not require access to remote NV, remain synchronous.

## 6.3 vTPM 2.0 for Cloud Computing

It is the first time that vTPM 2.0 based on Kernel-based Virtual Machines (KVM) is proposed, designed, and implemented [13]. Furthermore, in order to assure runtime security, this architecture also utilizes Intel SGX. We will also discuss the key distribution and protection mechanism based on Key Management Center (KMC) for vTPM 2.0 as proposed by the authors.

As shown in Figure 24, the main components in the vTPM 2.0 architecture are vTPM 2.0 management, Libptms 2.0, NVRAM files, tpm2driver, `tpm_tis`, and SeaBIOS. The vTPM 2.0 management module is responsible for implementing vTPM management, such as creating vTPM, processing commands, etc. SeaBIOS is a

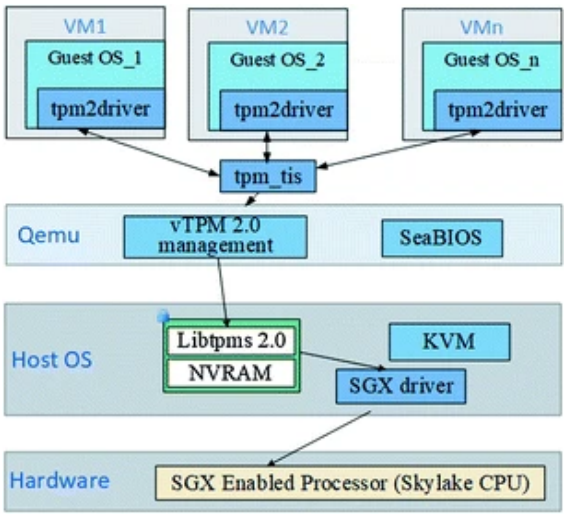


Figure 24: vTPM 2.0 architecture [13]

virtual BIOS that serves as a guest OS. It plays a major role in the creation of VM on the KVM platform.

The main module Libtpms 2.0 provides TPM 2.0 functionalities and command sets and also supports multiple virtual machines on the same physical TPM 2.0. Hence, Libtpms 2.0 is designed as a shared software library which is located in the host OS. It consists of a TPM module, a platform module, and a Crypto Engine module. The TPM module is responsible for implementing the reset of NVRAM, initialization of TPM components, and the process of command. The platform module implements the creation of the NVRAM file. The Crypto Engine module is responsible for cryptographic algorithms provided by TPM and is implemented through calling interfaces provided by OpenSSL.

As we saw, Libtpms 2.0 is a core module and handles sensitive data that needs to be protected. Hence, we isolate the module using SGX enclave. The SGX enclave is created when the Libtpms 2.0 module is loaded. The code of Libtpms 2.0 is executed in the Enclave Page Cache (EPC). This provides the necessary runtime security to the Libtpms 2.0 module.

NVRAM is similar to TPM memory. It is designed as a separate file that saves keys, seeds, PCR values, and other private data. Since NVRAM stores sensitive data that needs to be secured, we use SGX sealing to protect it. The sealing function of SGX encrypts the data inside the enclave and stores it in a permanent storage medium for future use. The private data of NVRAM is sealed by the seal key which is generated in the corresponding enclave. The NVRAM file will be unsealed and isolated in an enclave when it is loaded. Hence, only Libtpms 2.0 can access the NVRAM file at this point and no other software can access it. NVRAM is also migrated with the VM on the new platform.

Tpm2driver is used for providing interfaces to access TPM 2.0 hardware devices. `Tpm_tis` is used for emulating the hardware interface of TPM Interface Specification (TIS) in QEMU and implements the interfaces to call Libtpms 2.0. When a VM sends a TPM command, the tpm2driver of the VM will first talk to the `tpm_tis` frontend which is emulated by QEMU to deliver the TPM request. The `tpm_tis` delivers this request to Libtpms 2.0 which will then process the TPM command and return the results.

## 6.4 vTPM using Trusted Verification Server

This architecture introduces a trust verification server (TVS) that will provide the TPM functionalities. This will allow servers to use this server as a high-performance TPM chip. Using a server instead of actual physical TPM results in performance that is more than 100 times better [23]. Furthermore, a certificate authority (CA) is also included in this architecture. In this subsection, we will discuss this architecture.

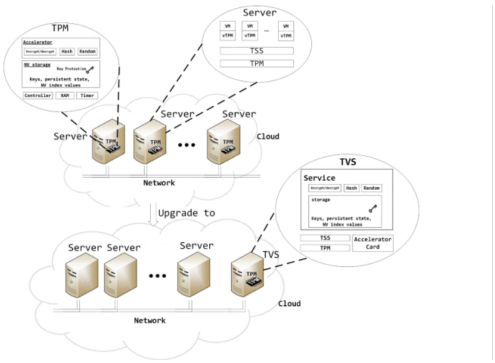


Figure 25: TVS motivation [23]

The main motivation of this architecture is depicted best in Figure 25. We can see that without a TVS, each server is equipped with a physical TPM and these physical TPMs are then used as root TPMs for cloud users. However, when we introduce a TVS server, we will require only one physical TPM that will reside in the TVS server. The TVS server will extend the functionalities of the TPM to all the servers and all these servers can use TVS as a TPM chip. Since TVS is a server and has more computing power, it has a much better performance than a physical TPM chip.

The architecture contains mainly three elements, namely ordinary servers, TVS, and CA, as shown in Figure 26. TVS is also the Root of Trust (RoT) in this architecture.

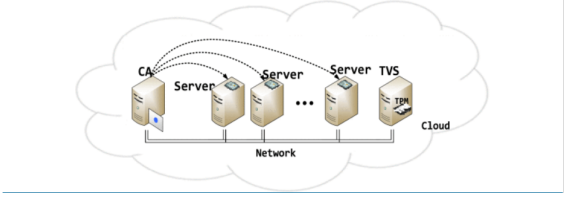


Figure 26: High Level TVS Architecture [23]

The servers here must apply for a certificate from the CA before starting its service. The CA connects multiple servers in a network and can issue electronic authentication certificate to each server and it also has strict security systems and national standards.

Server and TVS exchange communication in order to generate a symmetric session key which will be used for further communication. This initial communication, also known as session key generation, is encrypted using public key cryptography. The public keys are signed by the CA. For generating a session key, the server generates a random number and sends it to the TVS. The TVS then generates a session key based on that random number and shares it with the server.

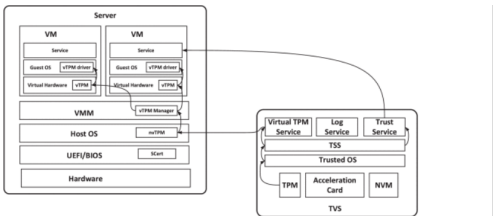


Figure 27: TVS Components and its interaction with the server [23]

The main modules of TVS and its interaction with the server is shown in Figure 27. As we can see, the TVS has a physical TPM. The EK of this TPM uniquely identifies the TVS. TVS uses the Acceleration Card to for faster calculation of encryption and decryption. The TPM ensures the boot time security of the TVS as it measures the trust of host OS which then loads the TPM Software Stack (TSS). The TSS is used to load vTPM Services. This vTPM service interacts with the server and implements a networked virtual TPM (nvTPM) in the host OS of the server. The vTPM Manager in the VMM regards this nvTPM as a physical TPM. Thus, VMM can now help establishing the vTPMs for cloud users in that server.

The logical process of multi-level trust with TVS has three main components and three trust verification steps, as shown in Figure 28. A complete trust chain is built on TVS by using multiple PCR.

First, the server enters the `Server_trust` stage when

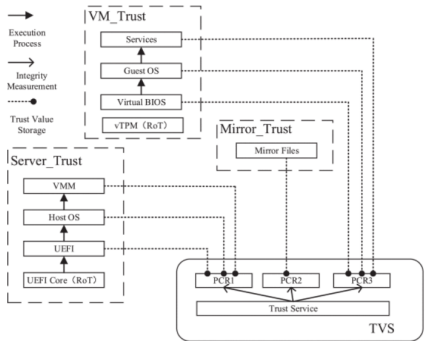


Figure 28: The logical process of multi-level trust with TVS [23]

it starts. In this stage, it connects TVS to request vTPM service. For this, the result is measured on remaining stages of UEFI/BIOS and securely stored in TVS. The trust chain now comes to VMM service in a sequential manner, and it starts to boot and schedule the `Mirror_Trust`. The VM needs to be verified and is verified by the vTPM. The vTPM communicates with VMM and becomes virtual RoT for the VM, whenever the VM boots or restores. Then, the vTPM continues the trust measurement of subsequent boot process until t comes to VM's service application. TVS also provides integrity measurement by performing hash checks, secure storage, trusted reporting, and encryption and decryption services for multiple servers.

To sum it up, the VMM boots the vTPM manager and the VM boots the vTPM. The vTPM then connects to the TVS and measure the boot process of VM from virtual BIOS, Guest OS to applications and then form a trust measurement summary.

## 6.5 vTPM-SM

Currently, vTPM only supports standard cryptographic algorithms approved by the TCG, that are ECDSA, RSA, AES and SHA256. However, relevant studies have shown that the Chinese commercial cryptographic algorithms, namely SM2/SM3/SM4 are more secure than ECDSA/SHA256/AES [1], [5], [6]. In this subsection, we will discuss a vTPM implementation with extended support of SM2/SM3/SM4 algorithms, known as vTPM-SM. SM2 is used for digital signatures, similar to ECDSA. SM3 is used for integrity check, similar to SHA256. SM4 is used for block cipher symmetric encryption, similar to AES-128. It is deducted by experiments that use of SM2/SM3/SM4 algorithms have reduced the time overhead by approximately 31.6%, 83.3%, and 15.5% respectively [14].

The architecture of the vTPM-SM is more or less same as a vTPM. However, there are a few additions as shown

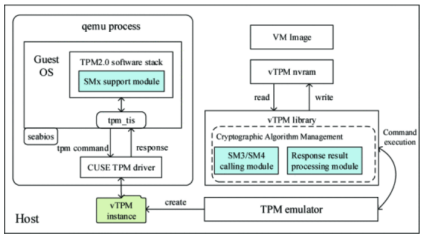


Figure 29: vTPM-SM architecture [14]

in Figure 29. In the vTPM simulation function library, we have added two modules: SM3/SM4 calling module and a response result processing module. Also, we have added a SMx support module in the TPM 2.0 software stack.

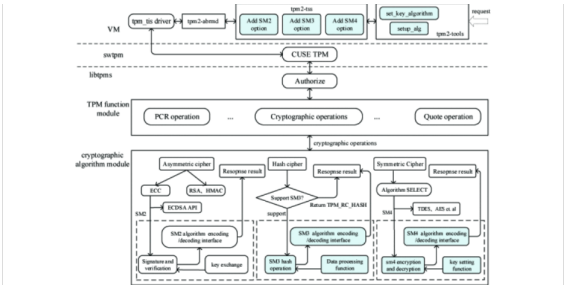


Figure 30: vTPM-SM process implementation scheme [14]

SM3/SM4 calling module adds the registration and definition of the SM3/SM4 algorithms to the function library. The response result processing module is added to process the results of SM3/SM4 operations. SMx support module adds calling options for SM2/SM3/SM4 algorithms for functions involving their tasks, that is signature, hash and symmetric encryption in the tpm2-tss software stack. The same thing is also done in the tpm2-tools too. The process of a VM interacting with vTPM-SM is shown in Figure 30.

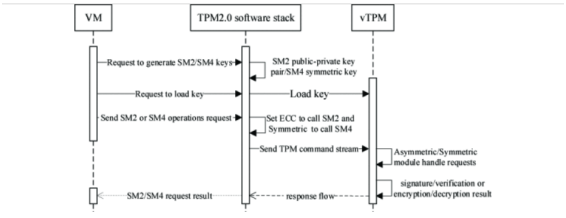


Figure 31: SM2/SM4 implementation via vTPM [14]

The implementation of SM2 signature/verification algorithm via vTPM is shown in Figure 31. The steps followed are:

- The VM generates a request to create SM2 key pair. The TPM 2.0 software stack generates a SM2 key pair and loads it into the vTPM when VM demands.
- When the VM sends a command to issue a SM2 signature or for verification, the TPM 2.0 software stack sets the ECC interface to use SM2 and it then uses SM2 for this purpose and passes the request accordingly.
- The signature, or the verification result is encoded and sent to the software stack through `tpm_tis` driver, and the software stack parses the result and returns it to the VM.

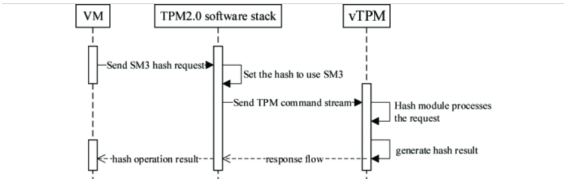


Figure 32: SM3 implementation via vTPM [14]

The implementation of SM3 hashing algorithm via vTPM is shown in Figure 32. The steps followed are:

- The VM issues a SM3 hash request. The software stack performs the SM3 hash by mapping the hash algorithm interface to SM3 hashing function.
- The operation result is encoded and is sent the software stack through `tpm_tis` driver, and the software stack parses the result and returns it to the VM.

The implementation of SM4 encryption/decryption via vTPM is shown in Figure 31. The steps followed are:

- The VM requests for the creation of SM4 key. The software stack maps the symmetric key generation function with the SM4 key generation algorithm and then generates the SM4 key. The software stack loads the key to the vTPM when commanded by the VM to do so.
- The VM issues a encryption/decryption request. The software stack maps the symmetric encryption/decryption algorithm to the SM4 encryption/decryption algorithm and performs the encryption/decryption.
- The result is encoded and sent to the software stack through `tpm_tis` driver, and the software stack parses the result and returns it to the VM.

# 7 Motivation and Security Challenges in vTPM

## 7.1 Secure vTPM-VM migration

The physical TPM consists of NVRAM which is used to store secrets that TPM generates, however, vTPM cannot have this NVRAM hence it uses a file present in the hardware flash of the TPM is used to store secrets such as seeds, root keys, and PCRs. Since the NVRAM file stores secrets of the vTPM it is necessary to protect the NVRAM file. The motivation here is to protect the file and meet the security requirements such as :

- Contents in the NVRAM file must be encrypted and stored securely
- Other VMs should not be able to access the NVRAM files and only authorized VM should be able to do so.
- Only enclaves that create the files should be authorized to access them.

Malicious actors can launch NVRAM replacement attacks by replacing an NVRAM file and launching the vTPM. In our study, we will present how to protect the file using a software-based vTPM approach (SvTPM)

## 7.2 Rollback Attacks

Snapshot is one of the features provided by the cloud environments which allows the users to return to the previous state of the VM if a certain exception occurs. Generally, TPM allows a limited number of attempts to bypass the security check mechanism, after a threshold is achieved TPM will enter a lock state [22]. In such cases, if in a cloud-based environment, after the lock state is achieved the attacker could roll back to the previous snapshot and hence might launch a dictionary attack. It is necessary that the states of vTPM are also rolled back if a snapshot feature is used hence making the state of VM and vTPM remain consistent. Thus, we will see the protection mechanism used to prevent these rollback attacks.

## 7.3 Fine-Grained trusted clock

Hardware TPM chips require a trusted clock for the following reasons :

- Determine the period of the lockout state after a particular dictionary attack: If an attacker tries to launch a dictionary attack on the TPM and crosses the threshold of maximum attempts, the TPM will enter the lockout state. The timer here will define the duration that TPM will remain in the lockout state
- Time-stamped authorizations: The trusted clock helps the TPM to create an authentication request authorized for a specific time. If TPM creates a particular key and TPM wants the key to be valid for pre-defined time intervals the trusted clock will help TPM in doing so.

The hardware clock is thus missing in vTPM. In SGX, SGX RDTSC(real-time-stamp counter) provides secure clocking service in the SGX which is not permitted in SGX1 enclave mode[2]. SGX2 allows the clock-based instruction, but any compromised software or malware is capable to manipulate the value of this timer in privileged mode[2], hence cannot be trusted. In our study, we will see how this clock is implemented in the vTPM to overcome the above-mentioned challenges.

## 7.4 Use of vTPM in cloud user authentication

In a federated identity environment, there are various ongoing issues and worries such as user identity information exploitation through SSO capabilities in service providers and identity providers, identity theft of users, service providers and identity providers, and user reliability. Federated identification has determined that although having sound architectural foundations, real estate implementation still has some security issues that need to be considered. Another issue with federated identity is the conversion of authentication systems to an SSO solution. It implies that people still need to be educated and that if the transition is not carried out properly, the user base may be lost [20]. The fact that there is no demand from consumers for a Web SSO solution makes the problem much worse.

# 8 Secure vTPM-VM migration

## 8.1 Attack Points in VM Migration

Attackers or intruders may take advantage of the live migration procedure for virtual machines. The points that are exposed to security risks are shown in Figure 33. Attack points are indicated by the star-marked numbers 1, 2, 3, and 4.

- Control Panel: Through the control panel, the system administrator manages server operations. Normally, only the system administrator is permitted to carry out any tasks, including building, removing, moving, and changing VM configurations. As a result, once an attacker has access to this interface,

the system is at risk. System administrators' configuration mistakes might also raise the possibility that the entire system will be compromised.

- Intruding between VMs: Despite being isolated, each VM may still communicate with other ones. As a result, there is a greater chance that a malicious VM may target other VMs using the same platform.
- Intruding between the host OS and guest VM: Given that the host operating system allocates all its resources, a virtual machine could undoubtedly communicate with its host and vice versa. All guest VMs operating on the host OS is entirely under the host OS's control. The guest VM may suffer damage from a compromised host. Like how a bad guest VM can damage the host OS.
- Eavesdropping in transmission channel: The migration data is not by default encrypted by the virtual machine migration protocol. Over the network, the migration data appears as plain words. They can be attacked by a man in the middle. The assault took place in the transmission channel and involved changing kernel memory, intercepting messages to look for sensitive information like passwords and keys, and replaying authenticated packets.

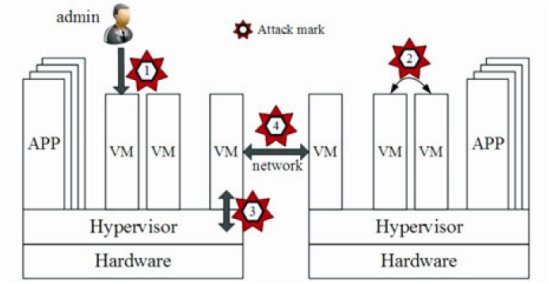


Figure 33: Attack Points[9]

## 8.2 vTPM-VM Live Migration

In this part, we go through how to migrate a virtual machine with a vTPM instance from one platform to another using the vTPM-VM live migration procedure. Integrity Measurement Architecture (IMA) is the component that is used to assess the integrity of a platform or an application. The system administrator or specific load-balancing algorithms select the source and destination platforms. These two platforms are chosen because their RAM, CPU, hypervisor types, and versions have been examined and found to be in compliance with live virtual machine migration standards.[9] The migration process can be divided into two phases as shown in Figure 34, in the following section we will be discussing the phases in detail.

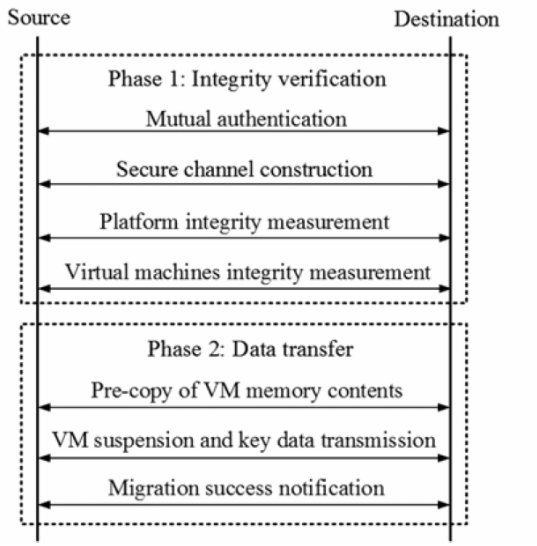


Figure 34: Phases of vTPM-VM live migration[9]

## 8.3 Phase 1: Integrity Verification

For reciprocal identity authentication, the source and destination platforms share their credentials. To create a secure channel based on the TPM, they negotiate a session key. Platform and virtual machine integrity testing then start to guarantee that all involved parties are trustworthy.

## 8.4 Phase 2: Data Transfer

A symmetric key generated on the source platform is used to encrypt the key data and the vTPM instance state. The memory of the VM starts to copy itself into the destination platform iteratively, once done the source platform suspends the VM and vTPM instance. The key data encrypted with the symmetric key is sent to the destination where it is decrypted and loaded into the destination platform.

## 8.5 Integrity Verification

An integrity verification policy based on TPM is used to verify the reliability of each participating entity in the live migration: the source platform, the target platform, and the vTPM-VM that will be moved, and the empty vTPM-VM container. The integrity verification process is divided into 10 steps (Figure 35) :

- A secure TLS connection is used to authenticate the source and destination. K(Channel) is used to encrypt the messages exchanged, here K(Channel) is used to verify the confidentiality and hashing algorithm to maintain the integrity.
- To check if the destination is reliable, the source platform sends a remote platform measurement request Measure reqs(D). To guarantee its freshness, a nonce Ns1 created by TPM is delivered with the message. M1= key channel (Measurereqs(D)||Ns1) represents the message delivered to the target platform.
- The destination computes M1's hash digest and checks the message's authenticity. If the message is correct, the destination calculates its own. The results are extended to PCRs values in TPM and the integrity of codes and data in the kernel and hypervisor.
- The source platform will get the PCRs values signed by an AIK, the measurement request from the source platform, Measure reqs(S), the nonce Nd1 created by TPM, and the accepted nonce Ns1. M2=keychannel(AIKsign(PCRs)||Measurereqs(S)||Nd1||Ns1) is how the message is represented.
- The source evaluates the integrity of its own codes and data in the kernel and hypervisor and extends the measurement findings to PCRs values in TPM after confirming the validity of M2 and the results of the destination platform's integrity measurement.
- The destination platform will receive PCRs values signed by an AIK, vTPM-VM integrity migration request Mig reqs, new nonce Ns2 issued by TPM, and the acceptable nonce Nd1. M3=keychannel(AIKsign(PCRs)||Mig reqs||Ns2||Nd1) is the message's expression.
- A new empty vTPM-VM container is built to accept the incoming VM and vTPM data after validating M3 and the destination platform's integrity measurement findings. The source platform will get the new nonce Nd2 created by TPM and the acceptable nonce Ns2, as well as the successful creation notice Create(vTPM-VM)bs, the integrity measurement request of the vTPM-VM to be migrated, and the Attestreqs(vTPM-VM). M4=keychannel(Create(vTPM VM)bs||Measure req(vTPMVM)|| Nd2||Ns2) is how the message is represented.
- The source platform starts the integrity measurement procedure of the virtual machine to be migrated and its accompanying vTPM instance after confirming the authenticity of M4. Along with PCR values in TPM, the assay data are additionally signed with an AIK. The target platform will then receive the signed PCRs values, the integrity measurement request of the vTPM-VM container Measurereqs(vTPM-VM)blank, the new nonce Ns3 created by TPM, and the accepted nonce Nd2. M5=keychannel(Measurereqs(vTPM-VM)bs||AIKsign(PCRs)||Ns3||Nd2) is how the message is represented.
- The destination platform analyzes the integrity of the vTPM-VM container and extends its measurement findings to PCRs values in TPM after validating M5 and the integrity measurement results of the vTPM-VM that will be migrated. Then, the PCRs signed by an AIK, the current nonce Nd2, and the acceptable nonce Ns3 would be delivered to the source platform. M6=keychannel(AIKsign(PCRs)||Ns3||Nd2) is how the message is represented.
- All four involved entities are satisfied that M6 is legitimate and that the vTPM-VM container's integrity measurement findings are accurate and proven reliable. The data transmission phase then starts.

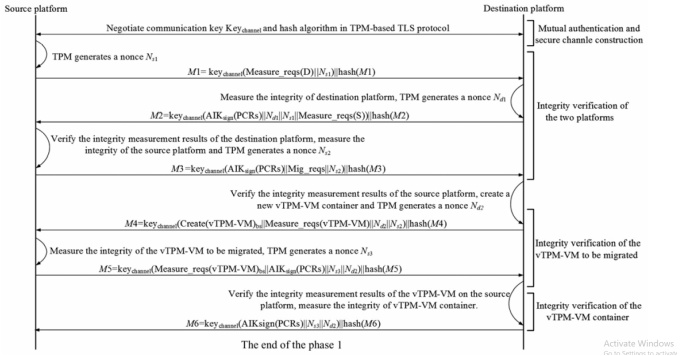


Figure 35: Integrity Verification Steps[9]

## 9 NVRAM Replacement Attack

The primary cause of the NVRAM replacement threat is the weak connection between VM and NVRAM. All of the NVRAM data is encrypted after the sealing procedure. Each NVRAM is sealed by a different key when using the KEY POLICY MRSIGNER key derivation policy if the user's enclave can be signed with the user's secret key during the enclave creation phase. This key derivation policy complies with the specifications. However,

it is still insufficient to create a binding connection between the VMs and NVRAMs in terms of secure storage and isolation. A binding link between the VM and the NVRAM file must be established in order to fight against NVRAM replacement attacks [21].

In order to fix the VM and vTPM binding issues. Create a binding scheme based on the user's private key for signing VM images and enclave. Users are asked to produce their own special key pairs and use those key pairs to sign the user-specific enclave files when they establish virtual machines with virtual TPMs. It can be made sure that each user's NVRAM is isolated from one another by employing the KEYPOLICYMRSIGNER key derivation scheme following this step. Additionally, when the VM is switched up, this key can be used to sign the VM image file and verify its integrity. a binding connection between the virtual machine image and the NVRAM file using the two protection techniques mentioned above can be created [21]. This creates a binding verification technique based on a remote third party in addition to establishing the binding relationship between VM and NVRAM. After the NVRAM has been correctly unlocked at startup, the vTPM sends its identity data to the cloud platform over the trusted channel. The vTPM measures the VM when it is launched and communicates the measurement of the VM. When the cloud gets two pieces of data, it compares the measurements from the enclave and the measurements from the virtual machine to assess if the binding connection is accurate.

# 10 Rollback attack prevention

## 10.1 Software Based Approach

The non-rollback data in vTPM has been synchronized using a design for a synchronization mechanism. By guaranteeing that the rollback operation cannot influence the non-rollback data in vTPM, this approach defeats the rollback attack against vTPM [21]. A Global Failedtries value outside the rollback area of vTPM status is added as an example of the FailedTries indicated above and is unaffected by the rollback process. In vTPM's typical functioning, this variable is synchronized using the FailedTries value. To make sure that FailedTries in vTPM space are not impacted by the rollback process, FailedTries can be synchronized using this value at the conclusion of the rollback operation because it is unaffected by the rollback action (Figure 36).

## 10.2 Hardware Based Approach

In vTPM, non-rollback data like FailedTries is represented using hardware monotonic counters from Intel ME [18]. The monotonic counter is applied to the vTPM

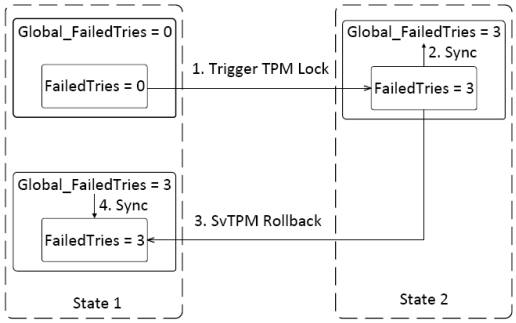


Figure 36: Software Based Approach[21]

when it is first initialized using the sgx createmonotoniccounter() function from the SGX SDK (i.e., Intel ME), which produces a Counter UUID. This value in NVRAM has to be stored for further manipulation. A previously stored Counter UUID is utilized to access and adjust the hardware counter when a vTPM needs to act on FailedTries. For the monotonic counter, SGX offers three access strategies: the first has the same signature key, the second uses an enclave with the same measurement value, and the third uses an enclave with the same signature key and measurement value (Figure 37).

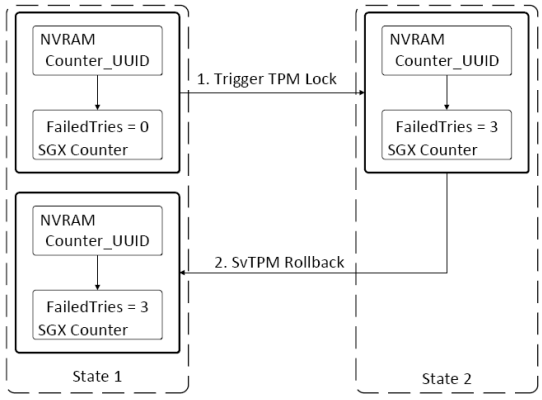


Figure 37: Hardware Based Approach[21]

# 11 Trusted Clock

A reliable clock is necessary for the creation of SvTPM, but regrettably cannot be directly supported by SGX enclaves. a possible solution is mentioned in this section to tackle this issue. Through the SGX Platform Service, SGX may get verified clock functions. Application enclaves may get a timer from the Platform Service Enclave (PSE) via the Converged Security and Management Engine (CSME). An application enclave operating on an

offline platform can utilize this trusted time service to keep track of how much time has passed since the timer was last read. Additionally, it provides a nonce that, as long as it stays the same, complies with Intel criteria for trustworthiness [21]. However, this reliable clock service offers clock values relative to the reference point in seconds. This goes against the TPM 2.0 specification's need for millisecond granularity.

Combining the coarse-grained PRTC and the fine-grained software clock is one strategy that may be used. A different software thread that runs in a loop and increments a counter at a set pace is used to implement the software clock inside the same enclave. As a result, in vTPM, this approach only uses the trusted clock values through the PSE when a coarse-grained clock is needed. When a fine-grained clock is required, we start a software thread that creates one inside the enclave and recalculates its values by regularly checking the PRTC.

# 12 Uses of vTPM in cloud user authentication

Trust must be built up in each element of the infrastructure in order to prevent a weak and susceptible connection in business security. To exchange the necessary key secrets that enable the signing of any messages sent back and forth, it is necessary to create a connection while taking into account the identities of all the devices and parties inside the trust domain [20]. The suggested concept is innovative since OpenID hasn't yet made it available. The concept, which combines cloud computing, trusted computing, and federated identification, can improve the security and privacy of cloud computing. A Trusted Platform Module can be used by SPs, IDPs, and consumers of cloud computing to authenticate their hardware [11]. It is possible to do platform authentication because each TPM (VTPM) chip or Virtual TPM (VTPM) has an exclusive and secret RSA key burnt into it when it is created. Each system should be independent of any trusted third party on the trusted platform. In addition to Independence, each system should be able to clearly identify users who can be trusted both inside and outside of businesses.

Platform agnostic and adaptable trustworthy computing should be used in OpenID to meet the demands of today's scaled computing systems. The suggested situation is shown in Figure 38. This diagram illustrates how the user's browser, RP, and IDP have trust connections that are managed by the trusted authority. Their negotiations reveal a six-step process. This model's usage of the TPM (VTPM), Virtual TPM (VTPM), and OpenID protocol, which tries to fulfill the duties of identity federation, authentication, and authorization, is highlighted [11].

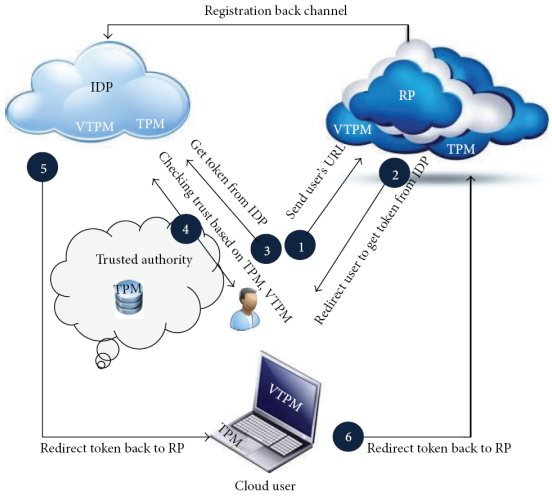


Figure 38: Federated identity Architecture[11]

## 12.1 Logging-in into user's URL

We can sign into websites utilizing OpenID by using a single URL-based identity. This URL, known as the Personal Identity Portal (PIP) URL, is used in place of our login and password. PIP makes it unnecessary to keep track of many usernames and passwords for the service providers. PIP also gives the freedom and elasticity to only share the data required with each service provider.

## 12.2 Redirection of client authentication

The SP determines the user's location in this stage and generates an authentication token. SP requests identification proof from the user. For this reason, the browser does the subsequent action after receiving the RP's request. In OpenID Authentication version 2.0, the TPM (VTPM) hash code can be used in place of the RSA method secret key to create a connection between the IDP and RP.

## 12.3 IDP Client Authentication

In this case, the browser continues the token exchange process via the SAML protocol. Depending on the circumstances of the engagement, users have three alternatives for proving their identity to IDP. First, the user must submit his or her user name and password to validate their identification. Second, if people enable cookies in their browsers, this may be accomplished. Thirdly, you may utilize software like the SeatBelt extension for Firefox, a plug-in that enables users to authenticate into

OpenID websites and IDP using the PIP URL. When a user clicks on an OpenID sign-in box, the SeatBelt extension recognizes it and prompts them to log in. The user will instantly be sent to the OpenID sign-in page after signing up for the first time, where his or her PIP has already been entered.

## 12.4 Mutual Attestation

The most important step in the suggested OpenID trust-based Federated Identity Architecture is step 4. The assumption made here is that IDP and SP could work together in our environment to link user alliances and gather user actions. Users' browsers, Relying on Parties (RP) or Service Providers (SP), and IDP must demonstrate their identity based on a mutual attestation process utilizing their TPM- (VTPM-) capable platforms and confirmed by the TA, using Trusted Authority (TA) as the core. We have assumed that the Trusted Authority is reliable and must have the approval of all parties involved in the Federated Identity Architecture or be sanctioned by the national government. In this case, the tester (TA) sends the challenge to the attested (IDP, USER, and SP) to verify their honesty. By default, TA checks their integrity via binary remote attestation, but in this case, it applies the DAA approach.

## 12.5 Browser Authorization

The IDP will only transmit a SAML token to the user's browser if and only if the mutual attestation procedure has been successful, which means that both the user and IDP have faith in one another.

## 12.6 Browser Request

The user now has more faith that the SP will provide a service based on the SAML token. IDP transmits an encrypted token based on the user's public key to demonstrate its legitimacy and verification by a reputable body. Using their private key, the user decrypts the token. Finally, the user uses the token to access more RP or SP services. It's crucial to remember that the TA in our suggested architecture will have to handle Virtual Machines (VM). It will be connected to various services that the client has requested. As a result, each virtual TPM (VTPM) that is connected to a VM on ClientMachine needs to be attested by TA.

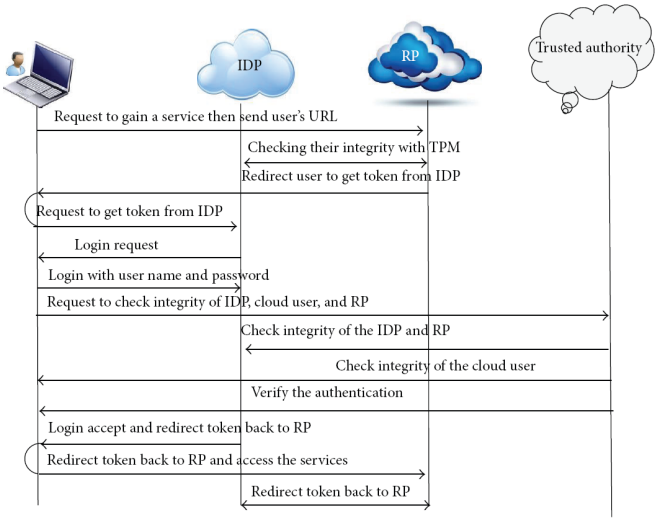


Figure 39: Trusted Based Sequence Diagram[11]

## References


[1] A YANG, J NAM, M. K. Provably-secure (chinese government) sm2 and simplified sm2 key exchange protocols[j]. In *The Scientific World Journal* (2014), Hindwai.

[2] ANDERSON, M. J., MOFFIE, M., DALTON, C. I., ET AL. Towards trustworthy virtualisation environments: Xen library os security service infrastructure. *HP Tech Reort* (2007), 88–111.

[3] BADE, S. A., BETZ, L. N., KEGEL, A. G., KELLY, M. J., AND TERRELL, W. L. Method and system for virtualization of trusted platform modules, May 27 2008. US Patent 7,380,119.

[4] CHEN CHEN, HIMANSHU RAJ, S. S. A. W. ctpm: A cloud tpm for cross-device trusted applications. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)* (2014), Usenix.

[5] D BAI, H YU, G. W. Improved boomerang attacks on round-reduced sm3 and keyed permutation of blake-256[j]. In *ET Information Security*, pp. 167–178.

[6] D WANG, L. W., AND ZHANG, X. Key-leakage hardware trojan with super concealment based on the fault injection for block cipher of sm4[j]. In *Electronic Letters*, pp. 810–812.

[7] DONGXI LIU, JACK LEE, J. J. S. N. J. Z. A cloud architecture of virtual trusted platform modules. In *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing* (2010), Springer.

[8] ENGLAND, P., AND LOESER, J. Para-virtualized tpm sharing. In *International Conference on Trusted Computing* (2008), Springer, pp. 119–132.

[9] FAN, P., ZHAO, B., SHI, Y., CHEN, Z., AND NI, M. An improved vtpm-vm live migration protocol. *Wuhan University Journal of Natural Sciences 20*, 6 (2015), 512–520.

[10] FELLER, T., MALIPATLOLLA, S., KASPER, M., AND HUSS, S. A. dctpm: A generic architecture for dynamic context management. In *2011 International Conference on Reconfigurable Computing and FPGAs* (2011), IEEE, pp. 211–216.

[11] GHAZIZADEH, E., ZAMANI, M., AB MANAN, J.-L., AND ALIZADEH, M. Trusted computing strengthens cloud authentication. *The Scientific World Journal 2014* (2014).

[12] JIN, X., WANG, L.-N., YU, R.-W., KOU, P., AND SHEN, C.-L. Administrative domain: security enhancement for virtual tpm. In *2010 International Conference on Multimedia Information Networking and Security* (2010), IEEE, pp. 767–771.

[13] JUAN WANG, FENG XIAO, J. H. D. Z. C. F. W. H. H. Z. A security-enhanced vtpm 2.0 for cloud computing. In *ICICS 2017: Information and Communications Security* (2017), Springer, pp. 557–569.

[14] MINGXING ZHOU, SHUHUA RUAN, J. L. X. C. M. Y. Q. W. vtpm-sm: An application scheme of sm2/sm3/sm4 algorithms based on trusted computing in cloud environment. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)* (2022), Springer.

[15] MURRAY, D. G., MILOS, G., AND HAND, S. Improving xen security through disaggregation. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2008), pp. 151–160.

[16] PEREZ, R., SAILER, R., VAN DOORN, L., ET AL. vtpm: virtualizing the trusted platform module. In *Proc. 15th Conf. on USENIX Security Symposium* (2006), pp. 305–320.

[17] SCARLATA, V., ROZAS, C., WISEMAN, M., GRAWROCK, D., AND VISHIK, C. Tpm virtualization: Building a general framework. In *Trusted Computing*. Springer, 2008, pp. 43–56.

[18] STUMPF, F., AND ECKERT, C. Enhancing trusted platform modules with hardware-based virtualization techniques. In *2008 Second International Conference on Emerging Security Information, Systems and Technologies* (2008), IEEE, pp. 1–9.

[19] TAN, L., XIAO, H., AND WANG, J. Research and development of tpm virtualization. In *Chinese Conference on Trusted Computing and Information Security* (2019), Springer, pp. 206–250.

[20] WAN, X., XIAO, Z., AND REN, Y. Building trust into cloud computing using virtualization of tpm. In *2012 Fourth International Conference on Multimedia Information Networking and Security* (2012), IEEE, pp. 59–63.

[21] WANG, J., FAN, C., WANG, J., CHENG, Y., ZHANG, Y., ZHANG, W., LIU, P., AND HU, H. Svtpm: A secure and efficient vtpm in the cloud. *arXiv preprint arXiv:1905.08493* (2019).

[22] XIA, Y., LIU, Y., CHEN, H., AND ZANG, B. Defending against vm rollback attack. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)* (2012), IEEE, pp. 1–5.

[23] ZHILOU YU, HONGJUN DAI, X. X. M. Q. A trust verification architecture with hardware root for secure clouds. In *IEEE Transactions on Sustainable Computing* (2018), Springer.

0