Enabling Secure VM-vTPM Migration in Private Clouds

Boris Danev, Ramya Jayaram Masti, Ghassan O. Karame and Srdjan Capkun
Department of Computer Science
ETH Zurich, Switzerland
{bdanev, rmasti, karameg, capkuns}@inf.ethz.ch

ABSTRACT

The integration of Trusted Computing technologies into virtualized computing environments enables the hardware-based protection of private information and the detection of malicious software. Their use in virtual platforms, however, requires appropriate virtualization of their main component, the Trusted Platform Module (TPM) by means of virtual TPMs (vTPM). The challenge here is that the use of TPM virtualization should not impede classical platform processes such as virtual machine (VM) migration.

In this work, we consider the problem of enabling secure migration of vTPM-based virtual machines in private clouds. We detail the requirements that a secure VM-vTPM migration solution should satisfy in private virtualized environments and propose a vTPM key structure suitable for VM-vTPM migration. We then leverage on this structure to construct a secure VM-vTPM migration protocol. We show that our protocol provides stronger security guarantees when compared to existing solutions for VM-vTPM migration. We evaluate the feasibility of our scheme via an implementation on the Xen hypervisor and we show that it can be directly integrated within existing hypervisors. Our Xen-based implementation can be downloaded as open-source software. Finally, we discuss how our scheme can be extended to support live-migration of vTPM-based VMs.

1. INTRODUCTION

Trusted Computing [1] is a set of technologies that provide hardware and software support for secure storage and software integrity protection. Its integration into virtualized computing systems [2] enables the hardware-based protection of private (sensitive) information and the detection of malicious software that aims to subvert the operation of virtualized environments. While these enhancements add an additional layer of security to the underlying data and applications [3], the use of Trusted Computing in virtual platforms raises several challenges with respect to virtualization of its hardware root of trust, the Trusted Platform Module (TPM), which provides secure storage and cryptographic operations.

In fact, there is typically a single TPM module per hardware platform. Therefore, its functionality has to be efficiently shared by the virtual machines (VM) running on the same hardware. This is typically achieved by virtual TPMs (vTPMs) that mimic the interface and functionality of the hardware TPM. One important challenge is to realize vTPMs that comply with TPM specifications while not impeding platform processes such as VM migration.

Although a number of realizations of vTPMs have been proposed [4–7], several issues remain unresolved with respect to the application of these realizations to the migration of vTPMs and their corresponding VMs (VM-vTPM). In this respect, current schemes for secure VM-vTPM migration protocols either increase the virtualized platform dependence on Privacy CA [4, 7, 8] or violate TPM usage restrictions [5]. Moreover, as far as we are aware, there is no implementation of a secure VM-vTPM migration protocol.

In this work, we consider the problem of enabling secure vTPM-based VM (VM-vTPM) migration in virtualized environments. We first extend and detail the requirements that a secure VM-vTPM migration protocol should satisfy in private cloud environments. In these environments, a central provider owns a number of virtualized servers and wishes to transfer VMs along with their corresponding vTPMs across these servers (e.g., for load balancing purposes). Given this, we identify three security requirements related to the authenticity of the migration initiator, the preservation of the trust chain among entities during migration and the confidentiality of the migration process. We argue that these requirements are sufficient to support secure VM-vTPM migration in private cloud environments.

Second, we discuss the implications of current vTPM key hierarchy designs on the efficiency and performance of the VM-vTPM migration process. More specifically, we show that existent vTPM key structures are either decoupled from their corresponding hardware TPM keys or are tightly bound to the hardware TPM. The former approaches do not benefit from the security guarantees of TPM, while the latter make the vTPM keys non-migratable according to the TPM specification and therefore required costly key regeneration at the destination after the VM-vTPM migration process. Based on these observations, we derive a novel vTPM key hierarchy that introduces an intermediate layer of keys between the TPM and vTPM and provides a logical separation of the vTPM keys according to their usage in the VM. Our proposed key hierarchy also enables vTPM key migration

ACSAC '11 Dec. 5-9, 2011, Orlando, Florida USA
Copyright 2011 ACM 978-1-4503-0672-0/11/12 ...$10.00.

according to the TPM usage restrictions and minimizes the dependence of vTPM keys on Privacy CA.

In addition, we propose and analyze a secure VM-vTPM migration protocol that leverages on our vTPM key hierarchy. Our proposed protocol provides stronger security guarantees when compared to existing solutions for VM-vTPM migration. Furthermore, we implement a preliminary Xen-based prototype of our protocol and we evaluate its performance. Our implementation demonstrates that our secure VM-vTPM migration solution can be directly integrated with the open-source hypervisor Xen [9]. We note that our implementation is open source and is available for download at [10]. Finally, we discuss how our scheme can be extended to support live-migration of vTPM-based VMs.

We point out that while prior work has addressed different aspects of secure VM migration, including vTPM migration, to the best of our knowledge, this work is the first to explicitly define the requirements, propose a suitable vTPM key hierarchy and design and implement a complete VM-vTPM migration protocol.

The rest of this paper is organized as follows. In Section 2, we detail our system and attacker models and derive the corresponding security requirements. We then present a novel vTPM key hierarchy and we describe a secure migration protocol in Section 3. In Section 4, we present a feasibility study and preliminary performance results extracted from a preliminary prototype implementation using the Xen hypervisor. Finally, we overview the related work in Section 5 and conclude in Section 6.

2. PROBLEM AND SECURITY REQUIREMENTS

2.1 System Model

In this paper, we consider a setting where a cloud provider $\mathcal{P}$ possesses several (> 2) virtualized servers that are equipped with physical TPMs and wishes to *securely* migrate virtual machines (VM) among these servers (e.g., for load-balancing purposes). Here, each virtual machine interfaces with the physical TPM through a software-based virtual TPM (vTPM) (refer to Section 3.1 for further details). We assume that vTPMs do not contain hardware and hypervisor configuration information; this information (stored in the TPM) is obtained by querying the TPM. Similarly, the hardware TPM does not include any VM specific information. This procedure decouples the vTPM from hardware-specific characteristics and enables its migration.

We assume that $\mathcal{P}$ wishes to migrate a virtual machine from a source server $\mathcal{S}$ to a destination server $\mathcal{D}$. We assume that $\mathcal{S}$ and $\mathcal{D}$ are equipped with public/private key pairs that are persistently stored on their respective TPMs[1]. During the migration process, we assume that the virtual machine can be suspended on $\mathcal{S}$ before it is transferred to $\mathcal{D}$; once the transfer is completed, the virtual machine is resumed on $\mathcal{D}$.

Given this setting, we consider the problem of enabling secure migration of a VM along with its vTPM from $\mathcal{S}$ to $\mathcal{D}$. Here, several challenges need to be overcome to ensure the liveliness and soundness of the migration process, namely: *(i)* only trusted servers should execute correct VMs, *(ii)* no external entity should be able to modify/learn the contents of the VM during the migration process and *(iii)* the migration can only be initiated by trusted parties (e.g., $\mathcal{P}$ or $\mathcal{S}$ in our case). Furthermore, this entire process should not violate the trusted computing standards [1] and should be easy to integrate with current vTPM platforms/architectures.

[1]Alternatively, the private keys could be sealed with TPM-specific keys and stored on disk.

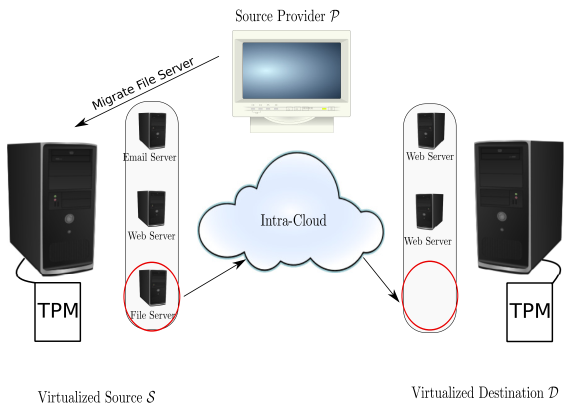

Source Provider $\mathcal{P}$
Migrate File Server
Email Server
Web Server
File Server
Intra-Cloud
Web Server
Web Server
TPM
TPM
Virtualized Source $\mathcal{S}$
Virtualized Destination $\mathcal{D}$

Figure 1: System model: a service provider $\mathcal{P}$ wishes to migrate a virtual machine from a virtualized source server $\mathcal{S}$ to a virtualized destination server $\mathcal{D}$ given some security constraints. Here, both $\mathcal{S}$ and $\mathcal{D}$ are equipped with hardware TPMs.

In what follows, we detail the attacker model and the security properties that a migration protocol should satisfy.

2.2 Attacker Model

We assume the presence of an attacker $\mathcal{A}$ that can eavesdrop, modify, insert or delete messages in the network. We assume that $\mathcal{A}$ is interested in abusing the migration protocol to increase her benefit in the network (e.g., starting her own VM, acquiring information about the transferred VM, etc.). We further assume that $\mathcal{A}$ is capable of exploiting software vulnerabilities of remote servers.

However, we point out that $\mathcal{A}$ does not have physical access to any server in the network managed by $\mathcal{P}$. In addition, we assume that $\mathcal{A}$ is computationally bounded, in the sense that she cannot forge signatures, break authentication schemes, without possessing the correct credentials.

Thus, we can safely assume that the TPM embedded in the various virtualized servers can be trusted. This trust assumption also extends to the software tool (e.g., IMA [12], HyperSafe [13]) that measures the system state for attestation using a (dynamic) root of trust (e.g., Intel TXT [14], Flicker [15]). Otherwise, little can be done to ensure that the software hosted on a given server is "authentic" and can be trusted. For simplicity and without loss of generality, we assume here that $\mathcal{A}$ cannot compromise or modify the state of software on the source and/or destination server during the migration protocol. That is, $\mathcal{A}$ can only compromise the software hosted by $\mathcal{S}$ and/or $\mathcal{D}$, either *before* the start or *after* the end of the migration process.

2.3 Requirements for Secure VM-vTPM Migration Protocols

We now present the requirements that any secure VM-vTPM migration protocol should satisfy. These requirements are sufficient for secure VM-vTPM migration and concern the parties involved in the migration as well as the communication channel over which the migration occurs. In Section 3, we also describe a protocol that fulfills them.

Requirement 1. ***(VM-vTPM Confidentiality and Integrity)***
An untrusted entity should not be able to learn any meaningful information about the VM-vTPM during the migration process. This includes the suspension, transfer and resumption of the VM-vTPM from the source to the destination. Furthermore, any modification to the VM-vTPM during the migration process should be detectable.

In addition to the basic confidentiality property, preserving the integrity of the vTPM during the migration process emerges as an important requirement for any secure VM-vTPM migration protocol. Otherwise, an adversary can convince $\mathcal{D}$ to accept a different VM image by modifying the contents of its vTPM.

Requirement 2. ***(Initiation Authenticity)***
An untrusted entity should not be able to migrate any VM-vTPM. Only $\mathcal{P}$ should be allowed to initiate the VM-vTPM migration process.

Restricting the initiation of the migration process solely to those authorized entities prevents an adversary from continuously migrating VMs across servers, thus alleviating Denial of Service (DoS) attacks against the entire system. Ensuring that only trusted parties can initiate the migration also prevents collocation attacks, where the attacker places the target VM on the same physical server together with another VM that it controls; this would create a covert-channel that might leak information about the target VM [11].

Although the initiation authenticity notion is important to prevent abuse of the migration process, it is has not been addressed, as far as we are aware, in prior work.

Requirement 3. ***(Preserving the Trust Chain)***
Only trusted servers can receive correct VMs. More specifically, (i) *trusted servers should not hold incorrect VMs-vTPMs and* (ii) *untrusted servers should not acquire correct VMs-vTPMs.*

By a correct VM, we refer to a VM which is found to be correct according to a trusted integrity measurement module (e.g., TPM-based attestation).

Ensuring the integrity of the software hosted on both $\mathcal{S}$ and $\mathcal{D}$ prior to the migration process is of paramount importance. A correct VM running in a trusted environment should not be transferred to a server that might be compromised (e.g., the VM might contain sensitive data). Similarly, a trusted server should not accept to run an incorrect VM that might have been compromised. For instance, if the hypervisor in $\mathcal{S}$ was compromised, then $\mathcal{D}$ cannot trust any protocol it establishes with $\mathcal{S}$. Note that this requirement does not address the case where an untrusted server executes incorrect VMs.

3. A SECURE VM-VTPM MIGRATION PROTOCOL

In this section, we outline an efficient solution that enables secure VM-vTPM migration. We start by providing the necessary background on TPM and vTPM keys.

3.1 vTPM Key Hierarchy

The design of the vTPM key hierarchy should provide the same functionality as the original TPM key hierarchy, i.e., allow proof of authenticity, attestation and secure storage [16]. In addition, it should comply with the TPM key usage restrictions and introduce minimal overhead during VM-vTPM migration (e.g., minimal key regeneration). In what follows, we provide a brief background on TPM functionality and keys and discuss several issues with existent vTPM key hierarchy proposals. We then introduce our vTPM hierarchy and discuss its implications on VM-vTPM migration.

Background on TPM Keys: The hardware TPM enables proofs of authenticity, attestation and secure storage based on three main cryptographic keys, namely the *Endorsement Key* (EK), the *Storage Root Key* (SRK) and the *Attestation Identity Key* (AIK). The EK is a persistent non-migratable encryption key that is used to establish the authenticity of the TPM. The use of this key for transaction authentication in the network is not recommended as it would enable TPM transaction linking. The SRK is a non-migratable encryption key that is used to protect the storage of other TPM keys outside the TPM. The AIK is an asymmetric non-migratable signing key generated inside the TPM and certified by a Certified Authority (Privacy CA). It is used as a one time key to establish authenticity of the TPM during attestation [1]. The AIK certificate proves the the AIK was created by a genuine TPM. Since it does not expose the EK, it can be safely used in network transactions without privacy concerns. The Platform Configuration Registers (PCR) are additional components used for attestation and secure storage; these components reside inside the TPM and store platform configuration measurements[2]. The latter are used either to attest the system integrity during remote attestation or seal data to particular system configurations [1].

Background on vTPM Keys: vTPM key hierarchies include keys analogous to their TPM key hierarchy counterparts. Each vTPM typically has its own virtual EK (vEK), virtual SRK ($vSRK$) which is used to protect the storage of other vTPM keys and virtual AIKs ($vAIK$s) used for platform attestation purposes.

The relationship between vTPM and TPM key hierarchies is an important design choice that needs to be taken into account in secure VM-vTPM migration. Several vTPM key hierarchy proposals completely decouple their keys from the TPM keys [4, 7]. This is achieved by obtaining the vTPM EK (vEK) and AIK ($vAIK$) credentials from a local authority. While this procedure avoids generating those keys on the platform vTPM after migration, it is not clear how it removes the need for vTPM credential regeneration. The inclusion of TPM PCRs in the certificate of a vEK to achieve VM-vTPM binding would require its frequent regeneration if TPM PCRs are periodically modified (extended) by means of dynamic system measurements [4]. All $vAIK$s obtained before the TPM PCRs changed would not be valid anymore. In addition, using a permanent vEK to prove vTPM-TPM binding during attestation [4] allows linking vTPM transactions. On the other hand, tight coupling of the vTPM and TPM (as discussed in [4]) by signing vTPM credentials

[2]These measurements often consist of hashing the state of the software running on the platform.

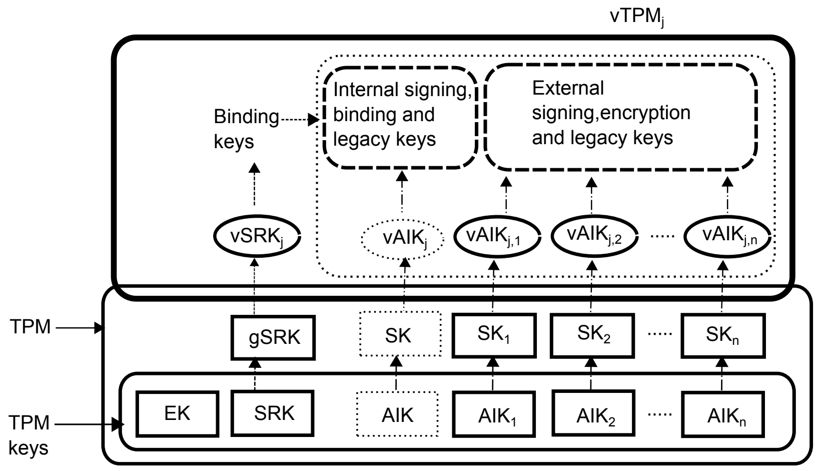


Figure 2: vTPM key set and hierarchy. Our proposed hierarchy consists of an intermediate layer of a global SRK ($gSRK$) and a set of signing keys (SKs) that connect the TPM SRK and AIKs to the vTPM $vSRK$ and $vAIK$. We also logically separate the vTPM keys into internal and external keys.

using the TPM AIK directly mandates that the corresponding keys be non-migratable and thus, requires extensive regeneration of vTPM keys on the destination platform after VM migration. The same limitation arises for the $vSRK$ if it is encrypted directly using the TPM SRK. Refer to Appendix A for more details regarding existent vTPM key hierarchy designs.

Our vTPM Key Hierarchy: In order to enable migration with minimized key regeneration after VM-vTPM migration, we propose a vTPM key hierarchy which introduces an intermediate layer of keys between the TPM and vTPM. This intermediate layer consists of one global SRK ($gSRK$) and a set of signing keys (SK) that connect the TPM SRK and AIKs to the vTPM $vSRK$ and $vAIK$ respectively (Figure 2). Even though this renders the signing keys $gSRK$ and SKs non-migratable[3], it allows the migration of the $vSRK$ and $vAIK$s and preserves the strong binding between the TPM and vTPM. Furthermore, using a separate SK with every $vAIK$ used in external communications prevents linking different vTPM transactions. We point out here that the $vSRK$ and $vAIK$ credentials can only be generated on a TPM containing the corresponding SRK and AIK. Generating the $vAIK$s on the platform itself removes the need for vEK because the authenticity of the vTPM only depends on the TPM AIK.

We further separate the vTPM keys into internal and external keys (see Figure 2). Internal vTPM keys are retained across VM migration. These include the $vSRK$ and the encryption and signing keys used only within the VM. The encryption keys are part of the $vSRK$ hierarchy and the credentials of the signing keys are signed by a $vAIK$ key linked to a TPM AIK (the key chain is shown in dotted ovals in Figure 2). Given that one such $vAIK$ could be sufficient for all internal signing, binding and legacy keys, VM-vTPM migration would incur minimal regeneration at the destination[4]. External vTPM keys are those keys used for signing and encrypting data exchanged between VMs over the network. Corresponding $vAIK$s are therefore restricted to one-time use in order to prevent vTPM transaction linking. Hence, these $vAIK$s are not part of a migrating vTPM.

[3]This is the case in order to preserve compliance with the TPM key usage restrictions

[4]We note, however, that several $vAIK$ can be used if needed.

Below we provide a summary description of our vTPM hierarchy keys and discuss the implications of this hierarchy on VM-vTPM migration:

- *vSRK*: Analogous to the TPM SRK, the vSRK protects the storage of other TPM keys. However, the storage of the vSRK itself is protected using the global SRK.

- *Global SRK (gSRK)*: This is an non-migratable asymmetric encryption key that is a direct descendant of the TPM SRK. It is used to protect the $vSRK$ of individual TPMs (by sealing) which in turn protects the other keys of their respective vTPMs (also by sealing). Creating this intermediate $gSRK$ makes the $vSRK$s migratable which would have not been possible if they were direct descendants of the TPM SRK.

- *vAIK*: Analogous to the TPM AIK, a $vAIK$ can be used to establish the authenticity of the vTPM and to sign other keys. We use a special $vAIK$ instance to sign data and/or certificates used only within the VM. This instance is transferred to the destination during vTPM migration. If a $vAIK$ signs data and/or certificates to be sent over the network, it is restricted to one time use to prevent vTPM transaction linking. Such AIKs are not part of a migrating vTPM. All $vAIK$s are linked to the TPM AIK via their own signing keys (SK).

- *Signing Keys (SKs):* These are an intermediate layer of non-migratable TPM signing keys that associate $vAIK$s with TPM AIKs. At least one SK is used for the special $vAIK$ instance (see above). Note that this SK can be common to all vTPMs on the same platform. All the other SKs are used to bind $vAIK$s to TPM AIKs intended to sign data and/or certificates to be sent over the network. SKs are not migrated during vTPM migration and therefore need to be generated on the destination platform. This entire key hierarchy is depicted in Figure 2.

Similar to most software-based vTPM solutions, our vTPM keys are stored outside the TPM and are prone to leakage and unauthorized modification. While the confidentiality of vTPM keys is protected by the $vSRK$, it is also possible to protect their integrity by the use of hash verification. This enables the detection of key modification, but does not prevent denial of service attacks by modifying the hashes themselves (on the filesystem).

During migration, the $vSRK$ of the vTPM is unsealed from the TPM using the corresponding $gSRK$ and is transferred along with other vTPM keys that are used only within the VM including the special $vAIK$ instance. At the destination, after migration, the $vSRK$ is sealed to the destination's TPM using its $gSRK$. Furthermore, the credentials for the special $vAIK$ instance are regenerated using the destination's special AIK and SK instances.

3.2 Protocol Description

Given the above vTPM key hierarchy, we proceed to presenting a possible construction of a secure VM-vTPM migration protocol (Figure 3).

Our exemplary construction mainly consists of three stages: the authentication stage, the attestation stage and the data transfer stage. In the first stage, the *authentication stage*, $\mathcal{S}$ and $\mathcal{D}$ mutually authenticate each other using their public key certificates and establish a secure channel for their subsequent communication. This can be achieved, for example, by using a non-migratable binding key that is stored on the TPM and that is bound to a secure configuration of either $\mathcal{D}$ or $\mathcal{S}$. Although this approach has clear advantages, it becomes rather costly as the message size increases; that is, the protocol will incur a prohibitively high overhead e.g., when a VM RAM is transferred from $\mathcal{S}$ to $\mathcal{D}$. In that case, a more suitable approach would be to rely on the establishment of Diffie-Hellman symmetric keys [17] between $\mathcal{S}$ and $\mathcal{D}$. For instance, this can be realized by using the TLS handshake protocol [18]. Once a session key K is established, $\mathcal{S}$ and $\mathcal{D}$ can use it to ensure the confidentiality and integrity of their communication. This can, for example, be done by concatenating each message with its hash (for integrity verification) and encrypting the result using key K (for confidentiality). Since we assume that an attacker cannot compromise the machines of $\mathcal{S}$ and $\mathcal{D}$ during the migration process, the established session keys can be stored in the system memory of both $\mathcal{S}$ and $\mathcal{D}$.

Once the authentication stage is completed, the *attestation stage* starts. This stage mainly consists of the integrity verification of both $\mathcal{S}$ and $\mathcal{D}$. In Section 3.3, we show that this verification prevents a considerable number of security threats.

To verify the integrity of $\mathcal{D}$, $\mathcal{S}$ proceeds as follows. It initiates the attestation process by sending $\mathcal{D}$ a freshly generated random nonce N_s. This would trigger a measurement module in $\mathcal{D}$ to perform a system measurement. System measurements typically include load and/or run time properties of the hypervisor [12,13]. These properties can be measured using a number of techniques such as [19,20].

The load time integrity of the measurement module itself can be further protected using a dynamic root of trust (like in Flicker [15]), which also provides a secure isolated run time environment. The measurement module also extends the public key certificate of $\mathcal{D}$ (or its hash) into the PCRs[5]. Given this, $\mathcal{D}$ then sends a signed copy of its PCRs (i.e., $\mathcal{D}$ sends $Sign_{AIK}(PCR \parallel N_s)$ signed using an AIK key obtained from a Privacy CA) containing details about the execution of the measurement module, the system configuration, its public key certificate along with a freshly generated random nonce N_s. We point out that these integrity measurements do not include any information corresponding to the contents of the VM being transferred. Instead, the integrity of the transferred VM is verified by $\mathcal{S}$ prior to migration (if any) and by $\mathcal{D}$ before resumption. $\mathcal{S}$ then verifies that the extracted PCRs correspond to those of $\mathcal{D}$ by checking the public key certificate extension into the PCRs. It then checks the validity of the AIK to verify the authenticity of $\mathcal{D}$'s TPM and $\mathcal{D}$'s PCRs to verify $\mathcal{D}$'s integrity. Similarly, $\mathcal{D}$ also verifies the integrity of $\mathcal{S}$. If these verifications pass, then the *data transfer stage* can start.

In this last stage, the actual transfer of the VM-vTPM occurs. Here, $\mathcal{D}$ sends $\mathcal{S}$ a freshly generated random nonce N_d indicating its readiness to receive the migrating VM-vTPM. $\mathcal{S}$ then transfers the contents of the VM-vTPM along with the received nonce on the established secure channel. In our construction, we require that $\mathcal{D}$ also checks the integrity of the migrated VM (for the reasoning why, refer to Section 3.3). Since it is assumed that the vTPM (or VM) queries the underlying TPM to obtain hardware and hypervisor measurement information, no separate mechanisms are required to update the vTPM with this information after migration. Finally, $\mathcal{S}$ deletes its local copy of the VM-vTPM and both $\mathcal{S}$ and $\mathcal{D}$ resume their operation.

3.3 Security Analysis

In what follows, we briefly analyze the security of our protocol construction.

The establishment of a secure channel between $\mathcal{S}$ and $\mathcal{D}$ ensures the confidentiality and integrity of all their exchanged messages. Furthermore, the use of Diffie-Hellman session keys ensures the forward security of the exchanged messages. That is, an attacker $\mathcal{A}$ cannot acquire the session key K once the migration protocol is terminated, even if it gains full control of $\mathcal{S}$ and all the exchanged messages between $\mathcal{S}$ and $\mathcal{D}$.[6] To acquire the key, $\mathcal{A}$ has to compromise $\mathcal{D}$. As such, our protocol construction satisfies the VM-vTPM confidentiality and integrity requirement (Requirement 1).

Furthermore, since the public key certificate of $\mathcal{D}$ (and $\mathcal{S}$, respectively) is extended in the PCRs during its integrity verification, $\mathcal{S}$ can ensure that the measured PCRs correspond to the physical machine of $\mathcal{D}$.[7] This prevents $\mathcal{A}$ from presenting measurements performed on *another* machine and claiming that they pertain to the machine of $\mathcal{D}$ (or $\mathcal{S}$, respectively); in this case, this misbehavior will be

[5]A variant scheme for linking the public key to the PCRs relies on the use of special TLS certificate extensions—which might, however, increase the size of trusted computing base (TCB) [21].

[6]Recall in this case that $\mathcal{S}$ can securely delete K at the end of the migration process.

[7]Linking the PCR measurements to $\mathcal{D}$ cannot be achieved solely by the use of the AIK of $\mathcal{D}$. This is because the AIK does not contain any information that could be used for identifying the entity to which it was issued (in this case, $\mathcal{S}$ or $\mathcal{D}$) [1].

$\mathcal{S}$ (Source)		$\mathcal{D}$ (Destination)
	Start of TLS Session	
Derive session key K	$\xleftrightarrow[\text{}]{\text{Mutual authentication, Key exchange}}$	Derive session key K
	Attestation of $\mathcal{S}$ and $\mathcal{D}$	
Pick $N_s \in \{0,1\}^n$	$\xrightarrow{\{N_s\}_K}$	
Verify N_s	$\xleftarrow{\{m_1\}_K}$	$m_1 = Sign_{AIK}(PCR \mid\mid N_s)$
	$\mathcal{S}$ verifies $\mathcal{D}$ platform integrity. A similar attestation of $\mathcal{S}$ is also performed by $\mathcal{D}$. If these verifications fail, the protocol is aborted.	
	VM-vTPM Transfer	
Suspend and lock the VM and its vTPM	$\xleftarrow{\{N_d\}_K}$	Pick $N_d \in \{0,1\}^n$
$m_2 = VM \mid\mid vTPM \mid\mid N_d$	$\xrightarrow{\{m_2\}_K}$	Verify N_d.
	$\xleftarrow{\{DELETE\}_K}$	Verify VM integrity
Delete VM and vTPM	$\xrightarrow{\{RESUME\}_K}$	Import VM-vTPM Resume VM and its vTPM
Delete K		Delete K
	End of TLS Session	

Figure 3: Sketch of a secure VM-vTPM migration protocol. The session key K is derived using key exchange protocols (e.g., TLS handshake) and used only for the current protocol instance. $\mathcal{S}$ and $\mathcal{D}$ verify each other's integrity by examining PCRs signed using an AIK key obtained from a Privacy CA. $Sign_X(Y)$ refers to the signature of Y using key X, n is a security parameter.

directly detected by $\mathcal{S}$. Since we assume that $\mathcal{A}$ cannot modify software on $\mathcal{S}$ and/or $\mathcal{D}$ during the migration process, little can be done by $\mathcal{A}$ to convince $\mathcal{S}$ that its machine is "honest", while it hosts in reality malicious/compromised software (Requirement 3). Given that the integrity of $\mathcal{S}$ is also verified, the authenticity of the migration initiation can also be ensured. This is the case since $\mathcal{D}$ verifies that it is interacting with an "honest" source and therefore can trust that $\mathcal{S}$ will abide by the protocol specification. This also includes trusting that $\mathcal{S}$ *(i)* will initiate the migration process upon the request of $\mathcal{P}$ (Requirement 2) and *(ii)* will *securely* delete the key K at the end of the migration process [22,23].

Note that this trust does not extend to the contents of the migrated VM. Indeed, while $\mathcal{S}$ might be "honest", the VM itself might be compromised by $\mathcal{A}$ prior to the start of the migration process. This use-case is countered by requiring that $\mathcal{D}$ checks the integrity of the migrated VM after the data transfer. As a result, $\mathcal{D}$ can ensure that only correct VMs can be migrated to its environment, thus conforming with Requirement 3 (Section 2).

4. PRACTICAL CONSIDERATIONS

So far, we have discussed secure VM-vTPM migration in the context of VM migration using the suspend- transfer-resume paradigm where the vTPM is a process inside the VM itself. Here, we discuss several challenges in extending it to alternative VM migration mechanisms (e.g., live migration) and vTPM architectures (e.g., each vTPM inside a separate VM, all vTPMs in a separate privileged VM). The performance of secure VM-vTPM migration is critical to its adoption and optimizing it requires understanding the nature of the overhead that it imposes. For this purpose, we present a prototype implementation and an initial study of the overhead in terms of end-to-end migration time and CPU usage.

4.1 Feasibility Study

In order to demonstrate the feasibility of our secure VM-vTPM migration protocol, we implemented a prototype and integrated it in the Xen hypervisor [9]. Our implementation emulates suspended VM-vTPM migration where each VM runs its own vTPM in the user space. We also show preliminary performance results on the overhead incurred by securing VM-vTPM migration in terms of end-to-end migration time and CPU usage. Finally, we discuss additional protocol characteristics and possible optimizations.

Implementation setup

In our implementation, we considered a private cloud virtual environment similar to the one described in Figure 1. We used two Thinkpad W510 (1.73 GHz, 8 GB RAM) machines with identical hardware as migration source ($\mathcal{S}$) and destination ($\mathcal{D}$). Both machines were configured to use the 64-bit Xen hypervisor (version 4.0.2-rc2-pre). The virtual machines (VM) running on $\mathcal{S}$ and $\mathcal{D}$ had their configuration files, disk and swap spaces on a separate NFS shared server. Each VM instance had its own virtual TPM running in user space. $\mathcal{S}$, $\mathcal{D}$ and the NFS server were on the same 1 GB Ethernet local network (LAN). Therefore, the migration protocol only transfers the VM-vTPM RAM im-

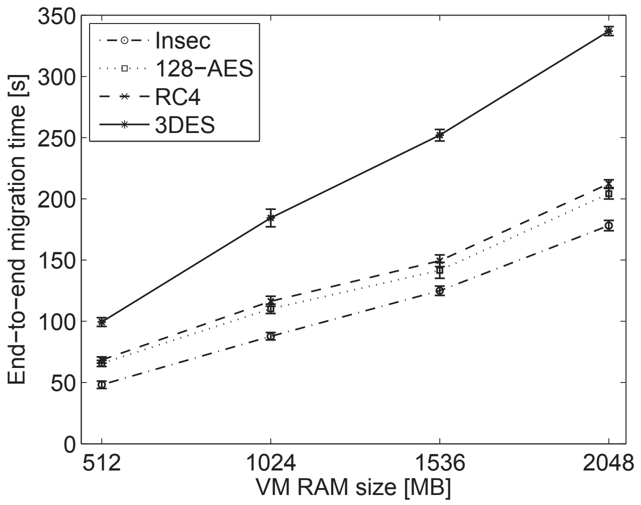


Figure 4: End-to-end VM-vTPM migration downtime for different encryption ciphers and VM RAM sizes. The results are validated over five independent migration protocol executions. We also show the corresponding 95% confidence intervals.

age. We note that this configuration is common in private cloud environments [24, 25].

Our implementation leverages on the Python-based Xen VM suspension and resumption. We used the TPM emulator [26] as a vTPM and executed it within the VM itself. We integrated OpenSSL (version 0.9.8o) to establish an authentic and secure channel and used the Privacy CA [27] to obtain AIKs for remote attestation. For simplicity, we only used boot time integrity measurements during attestation. We also implemented an insecure version of VM migration which simply transfers the VM-vTPM using the standard TCP socket interface. This implementation was used for comparison. Our source code is open-source and available for download at [10].

Preliminary results

One of the most common metrics used for evaluating the performance of migration protocols is the total end-to-end VM migration downtime [24, 25, 28]. It is defined as the time elapsed between the initiation of the migration protocol at the source and the completion of the VM migration at the destination. It is indicative of the actual VM downtime perceived by the end user. We measured the end-to-end migration time for different VM RAM sizes and encryption ciphers. Figure 4 shows the results validated with five independent migration protocol executions. For VM RAM sizes of 1 GB (typical in private cloud environments), the secure protocol with 128-AES encryption completed within approximately 110 s, that is 20% slower than the insecure version (88 s). We note that the absolute values can only significantly improve if the industry-standard virtualized hardware is used.

We further investigated the underlying overheads in the migration process by considering the CPU usage. The CPU time consumed by a process is indicative of the actual overhead imposed on the system as well as the contributions of the different components. We used the Google CPU profiler, part of the Google-perf tools [29] to instrument and measure the CPU usage during migration. Figures 5(a) and 5(b) show the distribution of CPU time between I/O and cryptographic operations measured for different VM RAM sizes and encryption ciphers. We notice that most of the time is spent in cryptographic (encryption, secure hashing) and optimized SSL I/O operations during the secure transfer of the VM RAM image. Note that this overhead is common to all secure solutions. We should also note that the overhead on Privacy CA and key regeneration in our design is independent of the number of migrated vTPM keys. In terms of the overhead due to attestation, this overhead depends on the type of properties being measured and the size of the measurement target. We note that the time required to measure hypervisor static properties is typically of the order of tens of milliseconds [19, 20].

The incurred security overhead may be tolerable in applications without strict timing constraints (e.g., email). In the case of time sensitive applications (e.g., video streaming), migration process optimizations would be required in order to reduce the total security overhead. These would include hardware and software as well as cryptographic-related optimizations.

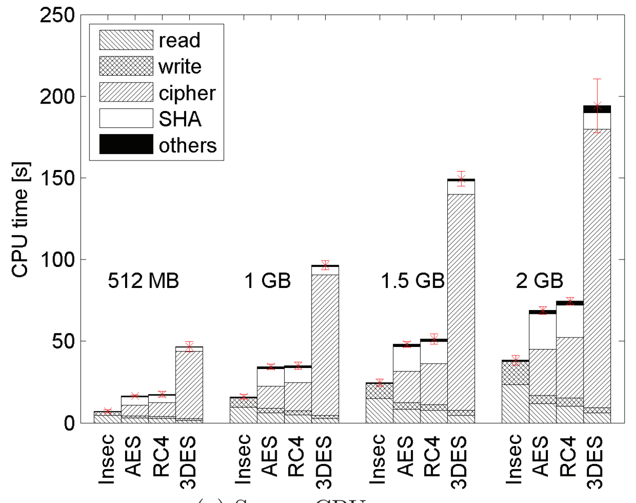
4.2 Discussion

In order to reduce the total migration downtime of time sensitive applications running on a VM, one direction is to consider performing live VM-vTPM migration [24, 25, 28]. However, secure live migration of VMs with vTPMs requires synchronizing VM-vTPM state during the transfer. Given that each vTPM (in our implementation) runs as a process within its own VM, existing memory synchronization techniques can be used to synchronize the VM-vTPM state on the source and destination during migration. This allows the resumption of the VM on the destination before the complete transfer of VM-vTPM state [24]. We note however that live migration of the vTPM may require specific VM memory partitioning in order to ensure that the vTPM state is transferred at once to avoid state corruption. The feasibility of such a secure VM-vTPM live migration approach needs more detailed investigation.

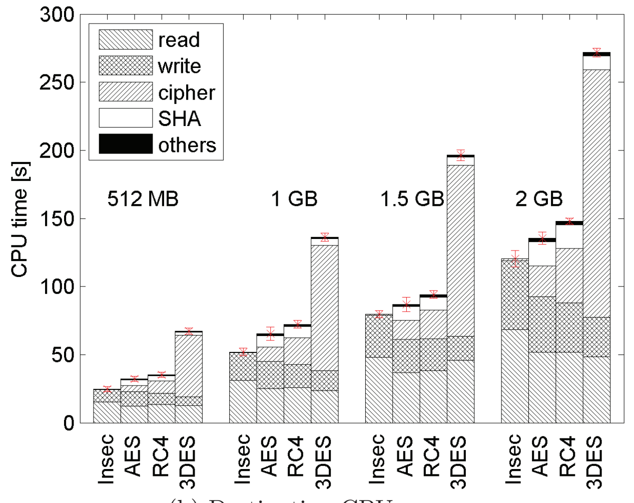
Other vTPM architecture designs could also help to improve performance. All vTPMs on a given hardware platform can run on separate, dedicated VM referred to as vTPM manager [4]. Such a dedicated VM can be migrated using secure VM migration as well. It is also easy to associate each vTPM to its VM in such cases. However, when all vTPMs run in a privileged VM, additional process migration techniques have to be devised to migrate the vTPMs. Further work is required to devise such migration and compare its properties and performance to the approach implemented in this work.

5. RELATED WORK

Previous work on VM-vTPM migration includes several protocol proposals in [4, 5, 7] and an implementation in the Xen hypervisor [9]. All protocols meet the confidentiality of the VM-vTPM during migration. However, only [4] used an integrity protection mechanism to ensure that migrated data has not been tampered with. For the other protocols, this is implicitly handled by the encryption mechanism in a sense that any modification of the encrypted VM image would re-



(a) Source CPU usage



(b) Destination CPU usage

Figure 5: CPU usage of prototype implementation for different encryption ciphers and VM RAM sizes. The results are validated over five independent migration protocol executions. We also show the corresponding 95% confidence intervals.

sult with high likelihood in problems with VM resumption at the destination. There are no means for the migration process to understand the causes of failed resumption (e.g., due to intentionally corrupted VM image during transfer or server execution problems). Measuring the system integrity before migration was suggested in [5, 7]. While in [5], the integrity of the destination is checked as part of the secure channel establishment, in [7] the process is not completely specified. None of these works checked the integrity of the migration initiator (source). Furthermore, the initiator authenticity was also not considered as a requirement in the above works. We believe that this is an important feature for DoS and collocation attack prevention.

Previous work on vTPM architectures include proposals for vTPM key hierarchies [4–8]. Creating vEK credentials using the underlying TPM EK [5] does not comply with the EK usage restrictions. Moreover, using the TPM AIK to sign vEK or $vAIK$ credentials makes these keys non-migratable and hence, require fresh key generation at the destination. Finally, obtaining vEK or $vAIK$ credentials from a trusted third party (Privacy CA) increases the external dependencies and is not easy to realize in practice. Besides, one would have to inform the Privacy CA on every migration because the only basis for issuing such a credential can be the underlying platform. In [6], it is unclear how vTPMs can be migrated if the TPM EK and AIK are shared between several vTPMs. In [8], generation of vTPM keys and credentials has been only discussed from the perspective of the types of TPM keys that could be used in a virtual TPM.

Virtual machines (VMs) can be migrated in three ways, namely, stop-transfer-start paradigm, suspend-transfer-resume paradigm and using live migration in increasing order of efficiency [24]. The first two techniques either stop or suspend the VM before its transfer to the destination respectively. Live migration techniques are the most popular because they minimize VM downtime [24, 25]. A summary of live migration techniques can be found in [28]. Furthermore, VM migration may involve the transfer of the VM RAM only or the transfer of the VM RAM along with the disk image [25].

6. CONCLUDING REMARKS

In this work, we considered the problem of enabling secure VM-vTPM migration in private cloud virtualized environments. We analyzed the requirements for secure VM-vTPM migration in internal virtualized environments. We also proposed a vTPM key hierarchy that provides robust functionality to construct secure VM-vTPM migration protocols. Our key hierarchy is compliant with the TPM key usage recommendations, minimizes key regeneration after vTPM migration and prevents vTPM transaction linking. By leveraging on this hierarchy, we proposed and analyzed a secure VM-vTPM protocol and we evaluated its performance by means of implementation using the Xen hypervisor. This implementation demonstrates that our proposed secure VM-vTPM migration scheme can be directly integrated in open-source virtual systems. Preliminary results on the performance of our scheme showed that—compared to the cost of encrypting data—our migration scheme only incurs negligible overhead in the regeneration of the vTPM keys at the destination. The implementation is open-source and available online [10]. Finally, we also discussed how our scheme can be extended to support live migration of VM-vTPMs.

In terms of future work, we intend to extend the implementation to support different vTPM key hierarchies and analyze the performance of VM-vTPM migration in realistic virtualized computing environments. We also plan to extend our work to possibly secure live VM-vTPM migration.

Acknowledgements

The authors thank Dr. Aurélien Francillon for his help during the implementation. The authors would also like to acknowledge the anonymous reviewers for their helpful feedback and comments. The work presented in this paper was partly supported by the Swiss National Science Foundation under Grant 200021-127294.

7. REFERENCES

[1] TCG Architecture Overview, v1.4. `http://www.trustedcomputinggroup.org`.

[2] Amazon Elastic Compute Cloud, `http://aws.amazon.com/ec2/`.

[3] Tal Garfinkel and Mendel Rosenblum. When virtual is harder than real: security challenges in virtual machine based computing environments. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 20–20, 2005.

[4] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 305–320, 2006.

[5] Frederic Stumpf and Claudia Eckert. Enhancing Trusted Platform Modules with Hardware-Based Virtualization Techniques. In *SECURWARE '08: Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, pages 1–9, 2008.

[6] Paul England and Jork Loeser. Para-Virtualized TPM Sharing. In *Trust '08: Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies*, pages 119–132, 2008.

[7] Ahmad-Reza Sadeghi, Christian Stüble, and Marcel Winandy. Property-Based TPM Virtualization. In *ISC '08: Proceedings of the 11th international conference on Information Security*, pages 1–16, 2008.

[8] Vincent Scarlata, Carlos Rozas, Monty Wiseman, David Grawrock, and Claire Vishik. TPM Virtualization: Building a General Framework. In *Trusted Computing*, pages 43–56, 2007.

[9] Xen hypervisor, `http://www.xen.org`.

[10] Secure VM-vTPM protocol implementation. `http://www.syssec.ethz.ch/software/vtpm-migration.zip`.

[11] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212, 2009.

[12] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, 2004.

[13] Zhi Wang and Xuxian Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pages 380–395, 2010.

[14] Intel trusted execution technology. `http://www.intel.com/Assets/en_US/PDF/whitepaper/323586.pdf`.

[15] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: an execution infrastructure for tcb minimization. *SIGOPS Oper. Syst. Rev.*, 42:315–328, 2008.

[16] TPM Main Part 1 Design Principles. `http://www.trustedcomputinggroup.org/resources`.

[17] W. Diffie and M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22:644 – 654, 1976.

[18] The transport layer security (TLS) protocol v 1.1. `http://www.rfc-editor.org/rfc/pdfrfc/rfc4346.txt.pdf`.

[19] Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang, and Nathan C. Skalsky. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 38–49, 2010.

[20] Jiang Wang, Angelos Stavrou, and Anup Ghosh. Hypercheck: a hardware-assisted integrity monitor. In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, RAID'10, pages 158–177, 2010.

[21] Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N. Asokan. Beyond secure channels. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 30–40, 2007.

[22] Radia Perlman. The ephemerizer: Making data disappear. *Journal of Information System Security*, 1:55 – 68, 2005.

[23] Radia Perlman. File system design with assured delete. In *In Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2007.

[24] Christopher Clark, Keir Fraser, Steven H, Jakob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–286, 2005.

[25] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 169–179, 2007.

[26] Mario Strasser and Heiko Stamer. A Software-Based Trusted Platform Module Emulator. In *Trust '08: Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies*, pages 33–47, 2008.

[27] Privacy certificate authority. `http://www.privacyca.com`.

[28] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post copy live migration of virtual machines citation. *ACM SIGOPS Operating Systems Review*, 43(3):14–26, 2009.

[29] Google Perftools, `http://code.google.com/p/google-perftools/`.

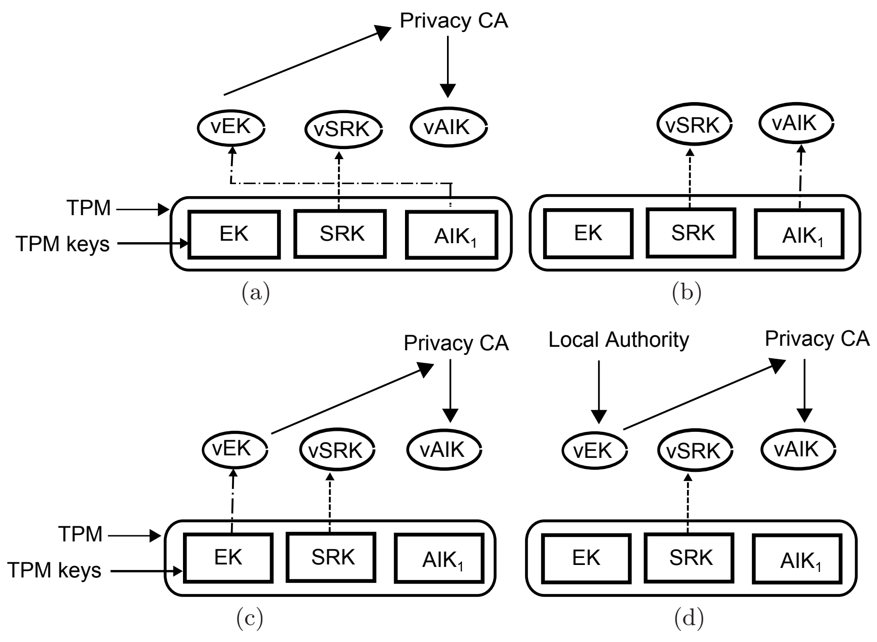


Figure 6: Existing vTPM key hierarchy designs. These designs range from tightly coupled TPM and vTPM keys to fully independent configurations which only depend on an external authority.

APPENDIX

A. PREVIOUS DESIGNS OF VTPM ARCHITECTURES

In [4], Berger et al. discuss the trade-offs between keeping vTPM key hierarchy independent from the TPM key hierarchy (as in Figure 6(d)) and linking it to the TPM via the AIK (as in Figures 6(a), 6(b)). The authors claim that the design in Figure 6(d) *(i)* minimizes the key and credential regeneration that are required after migration and *(ii)* complies with the procedure of obtaining AIKs from a Privacy CA. Sadeghi et al. [7] also adopt a similar design. In [5], Stumpf et al. propose a vTPM key hierarchy which uses the TPM EK to sign the vEK (see Figure 6(c)). Here, the AIK is obtained from a Privacy CA as in the case of the TPM.