Trusted Launch of Virtual Machine Instances in Public IaaS Environments

Nicolae Paladi[1], Christian Gehrmann[1], Mudassar Aslam[1], and Fredric Morenius[2]

[1] Swedish Institute of Computer Science, Stockholm, Sweden
[2] Ericsson Research, Stockholm, Sweden
{nicolae, chrisg, mudassar.aslam}@sics.se, fredric.morenius@ericsson.com

Abstract. Cloud computing and Infrastructure-as-a-Service (IaaS) are emerging and promising technologies, however their adoption is hampered by data security concerns. At the same time, Trusted Computing (TC) is experiencing an increasing interest as a security mechanism for IaaS. In this paper we present a protocol to ensure the launch of a virtual machine (VM) instance on a trusted remote compute host. Relying on Trusted Platform Module operations such as *binding* and *sealing* to provide integrity guarantees for clients that require a trusted VM launch, we have designed a trusted launch protocol for VM instances in public IaaS environments. We also present a proof-of-concept implementation of the protocol based on OpenStack, an open-source IaaS platform. The results provide a basis for the use of TC mechanisms within IaaS platforms and pave the way for a wider applicability of TC to IaaS security.

Keywords: IaaS, security, trusted computing, trusted virtual machine launch, OpenStack

Disclaimer: ”The original publication is also available at www.springerlink.com”

1 Introduction

One of the distinguished trends in IT operations today is the consolidation of IT systems onto common platforms. A key technology in realizing this is system virtualization [**?**]. System virtualization makes it possible to streamline IT operations, save energy and obtain better utilization of hardware resources. A virtualized computing infrastructure allows clients to run own services in form of Virtual Machines (VM) on shared computing resources. This approach however introduces new challenges, as it means that information previously controlled by one administrative domain and organization, is now under the control of a third party provider and that the information owner loses direct control over how data and services are used and protected. IaaS [**?**] is one of the business models based on system virtualization and security aspects are among the main identified obstacles for its adoption [3]. The problems with securing IaaS are evident not least through the fact that widely known platforms such as Amazon EC2, Microsoft Azure, services provided by RackSpace and other IaaS services are plagued by vulnerabilities at several levels of the software stack, from the web based cloud management console [**?**] to VM side-channel

[3] AFCEA Cyber Committee – October, 2011, `http://www.afcea.org/mission/intel/documents/cloudcomputingsecuritylessonslearned_final.pdf`

attacks, to information leakage, to collocation with malicious virtual machine instances [?].

A promising approach towards handling IaaS security threats and a mean to provide service confidence is the use of Trusted Computing technologies as defined by the Trusted Computing Group (TCG) [?]. The core component in the TCG-defined security architecture is the Trusted Platform Module (TPM), a hardware module that can be used as a trust anchor for software integrity verification in open platforms that also offers protected storage for sensitive parameters. TPM usage and deployment models for IaaS clouds are currently an active research area [?,?,?,?,?,?]. Earlier research has introduced principles of a trusted IaaS platform [?], later extended to cover both trusted VM launch [?] and VM migration [?]. These research results demonstrate principles of combining basic TPM attestation mechanisms with standard cryptographic techniques to design an infrastructure for VM protection. However, such solutions have limitations with respect to security, complexity and target compute host selection procedures.

In this paper we describe a trusted VM launch process that addresses these limitations and present a trusted launch protocol that does not require secure pre-packaging of the VM image on the client side. Furthermore, in order to be usable in a significant proportion of IaaS deployment scenarios and to provide full scheduling flexibility on the IaaS side, the protocol allows the IaaS provider to select a target trusted compute host without directly involving the client. The main contributions of this paper are:

1. Description of a trusted launch protocol for VM instances in public IaaS environments.
2. Implementation of the proposed protocol based on a widely-known IaaS platform.

The paper is further organized as follows: In section 2 we define the trust and attack models, formulate the problem area and define requirements for a scheme to address the identified issues; section 3 presents the main contribution of the paper, namely a platform-agnostic protocol for trusted virtual machine launching. In section 4 we perform a security analysis of the proposed protocol and continue with a description of the prototype implementation based on the OpenStack IaaS platform in section 5. In section 6 we provide summaries of significant related work within trusted computing in IaaS environments. We conclude in section 7 and provide a set of further research suggestions.

2 Trust and attack models, problem description and requirements

Next we describe the trust and attack models we assume in this paper, list the top security and general design requirements applicable given the defined trust and attack models and revisit virtual machine images in the context of a trusted VM instance launch. We also discuss the characteristics that can be expected from a well-designed VM instance launch.

2.1 Trust and attack models

In the trust model and consequently the attack model used in this paper, the client does not implicitly trust any aspect of the IaaS provider. Although potentially true for many IaaS environment types, this trust model should be particularly relevant to public IaaS environments (according to the definition in [?]), where the relationship between the client and the IaaS provider is often formal and lacks prerequisites for implicit trust.

We share the attack model with [**?**,**?**,**?**] which assume that privileged access rights can be maliciously used by IaaS provider remote system administrators ($\mathcal{A}_r$). This scenario assumes that $\mathcal{A}_r$ can log in remotely to any host maintained by the IaaS provider and obtain root access. However, in this model $\mathcal{A}_r$ does not have physical access to the hosts. The only possibility for $\mathcal{A}_r$ to circumvent this constraint is by succeeding to force a client to launch their VM instance on an $\mathcal{A}_r$-controlled compute host outside the physically secured IaaS provider perimeter. Furthermore, we assume that an $\mathcal{A}_r$ obtaining remote root access to the compute host will not be able to access the volatile memory of any VM residing on the compute host at that time, i.e. the compute host offers VMs a closed box execution environment [4]. This assumption is required in order to ensure that $\mathcal{A}_r$ can not access the nonce provided by $\mathcal{C}$ and its implementation is out of the scope of this paper.

In a trusted VM launch context this means that we consider both the attack where $\mathcal{A}_r$ attempts to launch a VM instance on a non-trusted compute host instead of on a trusted one and the attack where $\mathcal{A}_r$ attempts to substitute the VM image requested by the client with a maliciously modified VM image.

In the current attack model, a VM instance is considered trusted if and only if it fulfils the following criteria:

1. The VM image used for the instance is itself trusted;
2. The VM instance is started on a trusted compute host;
3. The VM instance has the client-generated verification token injected (see section **??**)

2.2 Virtual machine images

As an implication of the above trust and attack models, we consider the following two properties of virtual machines in the context of trusted computing:

- No VM instance, or any entity communicating with the VM instance, can determine whether the hypervisor the VM instance is running on is trusted or not.
- A VM instance cannot be trusted to reliably determine if it has the configuration originally requested by the client.

To overcome these issues, we suggest a launch protocol where we use standard TPM v1.2 functionality to first ensure that the client can detect the situation when it is communicating with a VM instance that is not launched on a trusted platform and subsequently utilize the trusted platform to verify the integrity of the VM image prior to VM launch.

It is essential, in the scope of the protocol, that no modifications or customizations of the VM image to be launched are performed by the IaaS provider without the client's knowledge.

2.3 Requirements for a trusted VM launch protocol

Considering the trust and attack models above, it is important for the client to be able to obtain reasonable security guarantees from the IaaS provider. These include both trustworthiness of the computing resources, as well as guarantees regarding VM integrity and confidentiality. In order to also be cost and implementation efficient, the underlying infrastructure should provide such guarantees with a minimal operational overhead without increasing structural complexity. The expectations can be summarized as a set of basic requirements towards a trustworthy VM launch process:

[4] This does not include any VMs which are part of the hosting infrastructrure, such as Xen dom0 VM)

- R1: The client shall have the mechanisms to ensure that the VM instance has been launched on a trusted compute host.
- R2: The client should have the possibility to reliably determine that it is communicating with a VM instance launched on a trusted compute host, and not with a different VM instance.
- R3: The integrity of the VM image scheduled to be launched must be verifiable by the target trusted compute host.
- R4: The trusted VM launch procedure should be scalable and have a minimum impact on the performance of the IaaS platform.
- R5: Clients should have a transparent view of the trusted launch procedure.

3 A trusted launch protocol for virtual machine images in IaaS environments

Based on the above requirements for a trusted launch protocol for VM instances in IaaS environments, we present a platform-agnostic protocol that shows principles of using TPM functionality to ensure the integrity of the compute host and of the VM image requested to be launched by the client. The below protocol addresses the security concerns presented above by focusing on simplicity, transparency, scalability and minimal interference with the currently known setup of the IaaS implementations. Furthermore, the protocol is based on widely-used and verified techniques, such as hashing and asymmetric cryptography in combination with TPM functionality.

The protocol requires the participation of four entities, three of which are typically involved in VM launch procedures in IaaS architectures:

1. *Client* ($\mathcal{C}$) is a IaaS user and intends to launch a VM instance. In this paper, $\mathcal{C}$ is considered to be a *non-expert*, i.e. one not capable of assessing the security of platform configurations based on values contained in the measurement list. $\mathcal{C}$ requires a VM instance to be launched on a trusted platform. Furthermore, it is important for $\mathcal{C}$ to be able to either verify or trust the security of VM images provided for launch.
2. *Scheduler* ($\mathcal{S}$) is responsible for receiving requests for VM instance launches from $\mathcal{C}$, as well as scheduling and rescheduling of VM instances on available compute hosts at the IaaS provider. $\mathcal{S}$ should be able to function with minimal involvement in the security-specific message passing.
3. The *compute host* ($\mathcal{CH}$) is the target resource that will be chosen by $\mathcal{S}$ to run the particular VM instance. $\mathcal{CH}$ represents a physical or virtual server that is able to host one or more VM instances (however, this paper considers exclusively the case when the $\mathcal{CH}$ is a physical server). For the purposes of the proposed protocol, a $\mathcal{CH}$ must also be equipped with a TCG-compliant TPM as well as be immune to modification attempts when in a trusted state.
4. The *Trusted third party ($\mathcal{TTP}$)* is, as the name implies, trusted by both the *Client* and the *IaaS provider* and can not be controlled or manipulated by the IaaS provider. The recent breaches of Certificate Authorities have emphasized the drawbacks of centralized security models and their susceptibility to attacks [**?**]. The more complex the operations performed by the $\mathcal{TTP}$, the higher the probability of it having exploitable vulnerabilities. It is therefore important to keep the implementation of the $\mathcal{TTP}$ as simple as possible. The main task of the $\mathcal{TTP}$ is to attest the configuration of the $\mathcal{CH}$ that will host the VM instance and assess its security profile according to predefined

policies. Within the current trust model, $\mathcal{TTP}$s could be implemented by an *expert* $\mathcal{C}$, as long as the IaaS provider agrees to that and $\mathcal{C}$ has the capability to set up and operate an attestation and evaluation engine.

For the purpose of the protocol, we also introduce the concept '*security profile* of a $\mathcal{CH}$':

Definition 1. *A security profile ($\mathcal{SP}$) is a verified setup of an OS including underlying libraries and configuration files, which is considered to be* trusted *by all parties. $\mathcal{SP}$ can range on an ascending integer scale which reflects the level of verification, from least to most strict (and hence more restrictive).*

The information needed to calculate the $\mathcal{SP}$ and also to compare the setup of two $\mathcal{CH}$s is stored in the *integrity measurement log* (IML), as the IML contains hashes of the components that were loaded or used during the boot sequence of the $\mathcal{CH}$. The validity of the IML is confirmed through a signature using the attestation identity keys (AIK) of a TPM. The AIK are persistent, non-migratable keys that are used to sign and authenticate by the means of an AIK certificate (denoted by $AIK{-}cert$) the validity of the information provided by the TPM in case of an external attestation [**?**]. We thus assume that the $\mathcal{SP}$ of any given $\mathcal{CH}$ can be deterministically calculated by each of the parties involved in the protocol. [5]

3.1 Platform-agnostic protocol description

The following steps are required in order to perform a trusted VM launch (Fig. **??**, the steps of the protocol correspond to the steps in the figure [6]).

1. Before initiating the launch procedure, $\mathcal{C}$ generates a sufficiently long nonce $\mathcal{N}$, to be used as a proof token in communications between $\mathcal{C}$ and the VM instance and must be kept confidential to untrusted parties throughout the launch process.
2. $\mathcal{C}$ creates a token which we denote by $\mathcal{T}$, representing a data structure with information necessary for the trusted VM launch. $\mathcal{T}$ contains $\mathcal{N}$, the minimum $\mathcal{SP}$ and the hash of the VM image used for launch, denoted by $H_{VMimage}$ [7]. Finally, the token is encrypted with the public key of $\mathcal{TTP}$, represented by PK_{TTP}, while the encrypted token is represented by $\mathcal{T}_{PK_{TTP}}$.
3. $\mathcal{C}$ provides the *scheduler (S)* the following parameters:
   - VM image identifier and optionally the VM image to be launched;
   - $\mathcal{SP}$;
   - URL of the $\mathcal{TTP}$;
   - Encrypted token $\mathcal{T}_{PK_{TTP}}$ generated in step **(2)**;

   $\mathcal{SP}$ will determine the lower bound of trust level required from $\mathcal{CH}$ on which the VM will run, with stricter security profiles accepted.
4. S schedules a VM on the appropriate $\mathcal{CH}$, depending on its membership in the respective security profile group and sends the $\mathcal{CH}$ a request to generate a bind key PK_{Bind}, also providing the URL of the $\mathcal{TTP}$.

[5] The methodology for calculating the $\mathcal{SP}$ of a $\mathcal{CH}$ is out of the scope of this paper.

[6] Due to space limitations, "Attestation data" was chosen as the condensed notation for: $\mathcal{T}_{PK_{TTP}}, PK_{Bind}, \texttt{TPM\_CERTIFY\_INFO}, {H_{\texttt{TPM\_CERTIFY\_INFO}}}^{AIK},\ IML, AIK-cert$

[7] If non-repudiation of VM launch is required, the client should also sign the VM image hash and include the signature and corresponding client certificate into the token.

5. Once the destination $\mathcal{CH}$ receives the bind key request, it retrieves a PCR-locked non-migratable TPM-based bind key PK_{Bind}. This key can be periodically regenerated by $\mathcal{CH}$ according to a administrator-defined policy, using the current platform state represented by the TPM PCRs. It is important to note that the values of the PCRs should not necessarily be in a trusted state in order to create a trusted state bind key.
6. In order to prove that the bind key is a non-migratable, PCR-locked, asymmetric TPM key, $\mathcal{CH}$ uses the `TPM_CERTIFY_KEY` TPM command in order to retrieve the `TPM_CERTIFY_INFO` structure signed with the TPM attestation indentity key [**?**], which we denote by PK_{AIK}; we also denote the signed structure by $H_{\texttt{TPM\_CERTIFY\_INFO}}{}^{AIK}$. The `TPM_CERTIFY_INFO` data structure contains the hash of the bind key and the PCR value required for the key usage.
7. $\mathcal{CH}$ sends an attestation request to the $\mathcal{TTP}$ through an HTTPS session using the URL supplied by $\mathcal{C}$. The following arguments are sent in the request to $\mathcal{TTP}$:
   - Client-provided token $\mathcal{T}_{PK_{TTP}}$
   - *Attestation data*, which includes the public bind key, the `TPM_CERTIFY_INFO` structure, the hash of `TPM_CERTIFY_INFO` signed with the AIK[8], the IML and the AIK-certificate collectively represented as: PK_{Bind}, `TPM_CERTIFY_INFO`, $H_{\texttt{TPM\_CERTIFY\_INFO}}{}^{AIK}$, IML, AIK-cert .
8. $\mathcal{TTP}$ uses its private key PrK_{TTP}, which corresponds to the public PK_{TTP} to attempt to decrypt the token $\mathcal{T}_{PK_{TTP}}$.
9. $\mathcal{TTP}$ validates the attestation information obtained from $\mathcal{CH}$ as follows:
   - Validates the AIK certificate;
   - Validates the structure of the AIK-signed `TPM_CERTIFY_INFO`;
   - Validates the key PK_{Bind} by comparing its digest with the digest received in `TPM_CERTIFY_INFO`;
   - Calculates the hash of the PCR values H_{PCR} based on the information in the IML and compares it with the hash of `PCR_INFO`, which is a component of `TPM_CERTIFY_INFO`
10. $\mathcal{TTP}$ examines the entries in the IML in order to determine the trustworthiness of the $\mathcal{CH}$ and decides whether $\mathcal{SP}$ is satisfied.
11. If step **10** is true, $\mathcal{TTP}$ encrypts $\mathcal{N}$ and the hash $H_{VMimage}$ with the bind key PK_{Bind} obtained from $\mathcal{CH}$, to ensure that $\mathcal{N}$ is only available to $\mathcal{CH}$ in a trusted state. By sending $\mathcal{N}$ and $H_{VMimage}$ encrypted with the public key PK_{Bind} available to the trusted configuration of $\mathcal{CH}$, the security perimeter expands to include three parties: $\mathcal{C}$ itself, $\mathcal{TTP}$ and $\mathcal{CH}$ in its trusted configuration. This implies that all actions performed by $\mathcal{CH}$ in its trusted configuration are trusted by default.
12. Prior to launching the VM, $\mathcal{CH}$ decrypts $\mathcal{N}$ and $H_{VMimage}$ using the TPM-issued PrK_{Bind}, which is available to it in its trusted configuration but stored in the TPM; next, $\mathcal{CH}$ compares $H_{VMimage}$ obtained from the $\mathcal{TTP}$ with the hash of the VM image to be used for launch and accepts the image only in case the values are equal.
13. $\mathcal{CH}$ injects $\mathcal{N}$ into the VM image prior to launching the VM instance.
14. $\mathcal{CH}$ returns an acknowledgement to S to confirm a successful launch.
15. To verify that the VM instance has been launched on a trusted platform, $\mathcal{C}$ challenges the VM instance to prove its knowledge of $\mathcal{N}$.

The fact that $\mathcal{N}$ is kept confidential allows it to be used as an authentication token while establishing a secure communication channel between $\mathcal{C}$ and the launched *VM*

[8] Expressed as $H_{\texttt{TPM\_CERTIFY\_INFO}}{}^{AIK}$

instance. $\mathcal{N}$ can be used as the pre-shared secret in order to add protection against man-in-the-middle attacks when using Diffie-Hellman key exchange, as specified in the password-authenticated key-exchange protocol [**?**].

Some of the operations can be optimized taking into account the operational environment. For example, the validity period of PK_{Bind} created in step **(5)** can be adjusted. In a similar way, $\mathcal{TTP}$ can have a cache of the PK_{Bind} keys created by $\mathcal{CH}$s with verified trusted configurations. In this case, steps **(9)** and **(10)** can be skipped for a certain number of cases, which can also be regulated by an administrative policy. However, it is important to remember that the use of such a cache introduces further complexity to $\mathcal{TTP}$, the analysis of which is out of the scope of this paper.

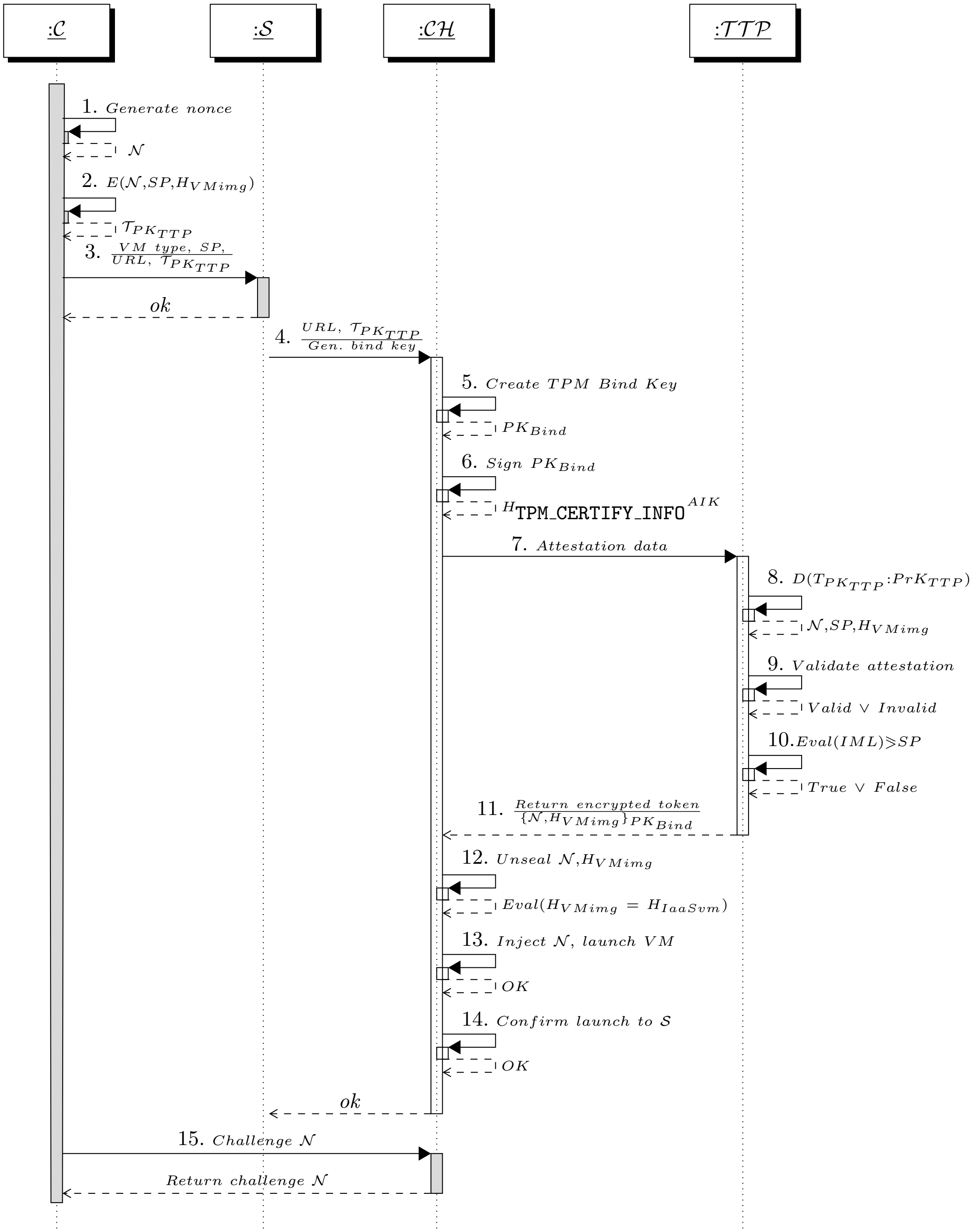
4 Protocol security analysis

In this section we present a critical review of the protocol and highlight improvement areas that were left as future work. We begin with a security analysis of the protocol, in order to outline its strengths and weaknesses.

Returning to the security concerns expressed in the requirements on the trusted launch protocol formulated in section **??**, they are addressed as follows. Let φ be the guest VM instance launched on $\mathcal{CH}$, then:

- R1: Following above protocol, $\mathcal{C}$ and φ have a shared secret $\mathcal{N}$. The fact that φ is running on a *trusted platform* is ensured by the properties of the bind key used to seal the shared secret $\mathcal{N}$ to the trusted configuration of $\mathcal{CH}$;
- R2: The fact that $\mathcal{C}$ is communicating with φ and not any other unexpected VM instance φ' is ensured through the combination of: **a.** verification of $\mathcal{CH}$ by the $\mathcal{TTP}$, **b.** presence of the token $\mathcal{N}$ injected into φ where $\mathcal{N}$ is only available to $\mathcal{CH}$ in a trusted state; **c.** the VM image integrity verification performed by the $\mathcal{CH}$ prior to the launch. A failure at any of the steps of the above sequence would prevent the trusted VM launch, a fact that would be verifiable by $\mathcal{C}$.
- R3: Integrity of the VM image is ensured through the verification performed by $\mathcal{CH}$ in a trusted state, prior to the trusted VM launch. Thus, the VM image is verified using the hash value obtained from the TTP. By comparing the hash of the VM image with the expected $H_{VMimage}$ provided by $\mathcal{C}$, $\mathcal{CH}$ ensures a one-to-one correspondence between the VM image to be used for launch and the VM image expected by $\mathcal{C}$. The chain is completed once $\mathcal{C}$ verifies the presence of $\mathcal{N}$ injected into φ. The presence of the correct token $\mathcal{N}$ guarantees the integrity of φ requested by $\mathcal{C}$.
- R4: Scalability of the protocol is ensured by the lightweight nature of operations that must be performed by both $\mathcal{TTP}$ and $\mathcal{CH}$ and the flexibility in the choice of $\mathcal{TTP}$. While a challenging topic, especially in the case of high-availability and heavy load IaaS setups, the design of a scalable $\mathcal{TTP}$ architecture is out of the scope of this paper.
- R5: Transparency of the trusted VM launch procedure is ensured by the introduction of client parameters, such as the URL of the $\mathcal{TTP}$, the trust level of $\mathcal{CH}$ and the secret token generated by $\mathcal{C}$. The ability to choose $\mathcal{TTP}$ opens the possibility for $\mathcal{C}$ to ensure the trustworthiness of the $\mathcal{CH}$ attestation procedure, either through audit controls of the $\mathcal{TTP}$ or by itself serving the role of $\mathcal{TTP}$.

Fig. 1. Trusted VM launch protocol: $\mathcal{C}$: Client; $\mathcal{S}$: Scheduler; $\mathcal{CH}$: Compute Host; $\mathcal{TTP}$: Trusted Third Party;


:$\mathcal{C}$
:$\mathcal{S}$
:$\mathcal{CH}$
:$\mathcal{TTP}$
1. $Generate\ nonce$
$\mathcal{N}$
2. $E(\mathcal{N}, SP, H_{VMimg})$
$\mathcal{T}_{PK_{TTP}}$
3. $\frac{VM\ type,\ SP,}{URL,\ \mathcal{T}_{PK_{TTP}}}$
ok
4. $\frac{URL,\ \mathcal{T}_{PK_{TTP}}}{Gen.\ bind\ key}$
5. $Create\ TPM\ Bind\ Key$
PK_{Bind}
6. $Sign\ PK_{Bind}$
$H_{\texttt{TPM\_CERTIFY\_INFO}}^{AIK}$
7. $Attestation\ data$
8. $D(T_{PK_{TTP}} : PrK_{TTP})$
$\mathcal{N}, SP, H_{VMimg}$
9. $Validate\ attestation$
$Valid \vee Invalid$
10. $Eval(IML) \geqslant SP$
$True \vee False$
11. $\frac{Return\ encrypted\ token}{\{\mathcal{N}, H_{VMimg}\}_{PK_{Bind}}}$
12. $Unseal\ \mathcal{N}, H_{VMimg}$
$Eval(H_{VMimg} = H_{IaaSvm})$
13. $Inject\ \mathcal{N},\ launch\ VM$
OK
14. $Confirm\ launch\ to\ \mathcal{S}$
OK
ok
15. $Challenge\ \mathcal{N}$
$Return\ challenge\ \mathcal{N}$

4.1 $\mathcal{TTP}$ verification model

The stateless architecture of the $\mathcal{TTP}$ implies that it does not maintain knowledge of $\mathcal{N}$ except for at the moment of sealing it to $\mathcal{CH}$ and does not maintain any session state at any point of the protocol. As a result, an $\mathcal{A}_r$ can only obtain $\mathcal{N}$ from $\mathcal{TTP}$ if they obtain $\mathcal{TTP}$'s private key PrK_{TTP}. Furthermore, assessment of the trust level of a $\mathcal{CH}$ according to a deterministic algorithm which only takes two inputs (in the form of static set of reference measurement data and dynamic attestation calls from any $\mathcal{CH}$) will be easily traceable and reproducible based on the original input data, without the need to recreate or rely on a certain state of the TPP's internal data. Finally, a stateless architecture of the $\mathcal{TTP}$ contributes indirectly towards requirement R4.

4.2 Protocol caveats

One aspect that requires more attention is the possibility of a post-launch modification of the software stack of $\mathcal{CH}$. The runtime process infection method, which is a method for infecting binaries during runtime [9] is one of the malicious approaches that could be used to this end. This scenario is in fact a common threat to all TCG-based systems, also touched upon in [**?**], described in detail in [**?**] and should thus be prevented using means within the platform which is part of the trusted computing base verified at boot time, the presence of which is verified by the above protocol.

5 Protocol implementation

In order to validate the assumptions made during the protocol design phase, we have implemented it as an extension to OpenStack, an open source IaaS platform chosen given the open access to its codebase, its large community and the traction it has gained. This section briefly introduces the OpenStack architectural model and changes made for the prototype implementation.

5.1 OpenStack IaaS platform

The Essex release of OpenStack comprises five core components (projects), namely Compute (Nova), Image Service (Glance), Object Storage (Swift), Identity Service (Keystone) and Dashboard (Horizon). Nova has several sub-components: nova-api, nova-compute, nova-schedule, nova-network, nova-volume, plus an SQL database and message queue functionality to pass messages between sub-components. OpenStack components affected by the protocol implementation are mentioned here in more detail:

- Nova-api is the interface for nova- compute and volume API calls. It is through this interface most of the cloud orchestration operations are performed. The interface supports both the OpenStack and Amazon EC2 API.
- Nova-compute handles VM instance life cycle tasks through hypervisor API calls. Notably the libvirt and XenAPI hypervisor APIs are supported.
- Nova-schedule is responsible for selecting $\mathcal{CH}$(s) to run VM instances on. The $\mathcal{CH}$ selection process is determined by which scheduling policy/algorithm is employed.

[9] Runtime process infection, `http://www.phrack.org/issues.html?issue=59&id=8&mode=txt`

- The nova SQL database holds tables and relations to describe the state of nova, such as launched instances and network configurations.
- The Dashboard is a web based GUI for OpenStack operation and administration. It interfaces nova-api.

5.2 Prototype implementation

Below are the main additions to OpenStack required for the prototype implementation.

Nova SQL database The nova SQL database has been extended to include tables to hold the available $\mathcal{CH}$s and their $\mathcal{SP}$s:

- An $\mathcal{SP}$ is an integer in the range 1-10, with a higher number being more trusted than a lower number.
- The security profile of a $\mathcal{CH}$ is global, rather than specific per e.g. tenant.

Dashboard and nova-api The Dashboard web based GUI has been extended to include the option to request $\mathcal{CH}$ attestation, minimum $\mathcal{SP}$ selection, token $\mathcal{T}_{PK_{TTP}}$ entry and $\mathcal{TTP}$ URL provision **(3)** into the *"Launch Instance"* dialog. This information is included in the OpenStack API HTTP payload to nova-api, which propagates the information to the scheduler.

In the prototype implementation, steps **(1)** and **(2)** are performed by a script which outputs $\mathcal{T}_{PK_{TTP}}$, which then can be manually input into the Dashboard dialog. Note that it is not an option to let Dashboard provide functionality for generating $\mathcal{T}_{PK_{TTP}}$, since Dashboard is not trusted by $\mathcal{C}$.

Scheduler, compute host and virtualization driver The nova scheduler $\mathcal{S}$ is a central component as it decides on which $\mathcal{CH}$ a certain VM instance will be launched. Each $\mathcal{S}$ works according to a specific configurable algorithm and several $\mathcal{S}$ implementations are available in OpenStack by default. In the SimpleScheduler implementation, $\mathcal{S}$ identifies the least loaded $\mathcal{CH}$ and schedules the VM instance to be launched on that $\mathcal{CH}$.

We extend the behaviour of the SimpleScheduler to include the policy that a $\mathcal{CH}$ must belong to a certain $\mathcal{SP}$ or stricter in order to be acceptable for hosting the VM instance. This policy is realized as follows: first $\mathcal{S}$ looks up the recorded $\mathcal{SP}$ of $\mathcal{CH}$ in the nova database and proceeds if $\mathcal{SP}$ is sufficient compared to the requirements of $\mathcal{C}$ (corresponds to **(4)**). The second step is to request $\mathcal{CH}$ to attest itself with $\mathcal{TTP}$. If $\mathcal{SP}$ was not sufficient, the next eligible $\mathcal{CH}$ is selected.

Steps **(5)**-**(7)** are perfomed by $\mathcal{CH}$, followed by $\mathcal{TTP}$ in steps **(8)**-**(11)**.
Token $T_{CH} = \{\mathcal{N}, H_{VMimage}\}_{PK_{Bind}}$ is returned from $\mathcal{TTP}$ to $\mathcal{CH}$ after which $\mathcal{CH}$ includes the token in the return message to $\mathcal{S}$. If the attestation was successful, $\mathcal{S}$ requests the now trusted $\mathcal{CH}$ to launch the VM instance and includes $\mathcal{T}_{CH}$ in the request.

Next, $\mathcal{CH}$ decrypts $\mathcal{T}_{CH}$ and obtains $\mathcal{N}$ and $H_{VMimage}$. To verify the integrity of the VM image, $H_{VMimage}$ is included in the call to the virtualization driver (`libvirt` is used by the prototype), which fetches the VM image from Glance and caches it locally on $\mathcal{CH}$. The hash of the cached image is calculated and compared to $H_{VMimage}$. If the hashes do not match, an exception is raised. Otherwise, the launch procedure continues **(12)** and the file injection capability of Nova is used to inject $\mathcal{N}$ into the file system of the VM image **(13)**. The VM image is then used to launch the VM instance on $\mathcal{CH}$ and steps **(14)** and **(15)** are completed.

6 Related work

Application of trusted computing principles within IaaS environments has been the focus of several research papers examined below.

Santos et al propose the design of a "trusted cloud compute platform" (TCCP) that ensures VMs are running on a trusted hardware and software stack with a remote and initially untrusted $\mathcal{CH}$ [?]. The authors propose a remote attestation process where a trusted coordinator ($\mathcal{TC}$) stores the list of attested $\mathcal{CH}$s that run a "trusted virtual machine monitor" which can securely run the client's VM. A trusted $\mathcal{CH}$ maintains in its memory an individual *trusted key* (TK) used for identification each time the client $\mathcal{C}$ instantiates a VM on the trusted $\mathcal{CH}$. The paper presents a good initial set of ideas for trusted VM launch and migration, in particular the use of a $\mathcal{TC}$. A limitation of this solution is that the TK resides in the memory of the trusted $\mathcal{CH}$, which leaves the solution vulnerable to cold boot attacks [?] with keys extractable from memory. Furthermore, the authors require that the $\mathcal{TC}$ maintains information about all $\mathcal{CH}$ deployed on the IaaS platform, but do not mention mechanisms for anonymizing this information, making it valuable to an attacker and unacceptable for a public IaaS provider. Finally, the solution lacks both mechanisms for revocation of the TK and considerations for the re-generation of TK outside of $\mathcal{CH}$ reboot.

A decentralized approach to integrity attestation is adopted by Schiffman et al in [?]. The primary concerns addressed by this approach are the limited transparency of IaaS platforms and the limits to scalability imposed by third party integrity attestation mechanisms, as described in [?]. The authors examine a trusted cloud architecture where the integrity of the IaaS $\mathcal{CH}$ is verified by the IaaS client through a "cloud verifier" ($\mathcal{CV}$) proxy that resides in the application domain of the IaaS platform provider and is accessible by the client. Thus, in the first step of the protocol the client evaluates the integrity of the $\mathcal{CV}$ in order to include the $\mathcal{CV}$ into its trust perimeter if the integrity level of the $\mathcal{CV}$ is considered satisfactory. Next, the $\mathcal{CV}$ sends attestation requests from $\mathcal{CH}$, i.e. the $\mathcal{CH}$ where the guest VM instance can potentially be deployed, thus extending the trust chain to the $\mathcal{CH}$. Finally, $\mathcal{CH}$ verifies the integrity of the VM image, which is countersigned by the $\mathcal{CV}$ and returned to the client which evaluates the VM image integrity data and allows or disallows the VM launch on the $\mathcal{CH}$.

While the idea of increasing the transparency of the IaaS platform for the client is indeed supported in industry [?,?], the authors do not clarify how the introduction of an additional integrity attestation component in the architecture of the IaaS platform has positive effects on the transparency of the IaaS platform. Furthermore, the proposed protocol increases the complexity model for the IaaS client both by introducing the evaluation of integrity attestation reports of the $\mathcal{CV}$ and $\mathcal{CH}$ and introduction of additional steps in the trusted VM launch, where the client has to take actions based on the data returned from the $\mathcal{CV}$. This requires either human interaction or a fairly complex integrity attestation evaluation component (or a combination thereof) on the client side, making a wide-scale adoption of the solution difficult.

In [?], Aslam et al proposed principles for trusted VM launch on public cloud platforms using trusted computing techniques. In order to ensure that the requested VM instance is launched on a $\mathcal{CH}$ with verifiable integrity, the client encrypts the VM image (along with all injected data) with a symmetric key sealed to a particular configuration of $\mathcal{CH}$, which is reflected through the values in the platform configuration registers (PCR) of the TPM deployed on the $\mathcal{CH}$. The solution proposed by Aslam et al presents a suitable model in the case of trusted VM launch scenarios for enterprise clients. It requires that the VM image is pre-packaged and encrypted by $\mathcal{C}$ prior to IaaS launch. However the proposed model does not cover the very common scenario of launching an unmodified VM image made available by the IaaS provider or uploaded by $\mathcal{C}$. Furthermore, we believe that reducing the number of steps required from $\mathcal{C}$ will facilitate the adoption of the trusted IaaS model. Likewise, direct communication between $\mathcal{C}$ and $\mathcal{CH}$, as well as significant changes to the existing VM launch implementations in IaaS platforms hamper the implementation of this protocol. This paper reuses some of the ideas proposed in [**?**] and directly addresses the above limitations, namely actions to be performed by $\mathcal{C}$, also touching upon the requirements towards the launched VM instance and required changes to the IaaS platform.

7 Conclusion

In this paper we have presented a detailed trusted launch protocol for VM instance launch in public IaaS environments. Furthermore, we have provided a prototype implementation of the launch protocol in OpenStack. Detailed performance measurement and evaluation, as well as alternative implementation choices have been left for future work.

The presented results make a case for broadening the range of use cases for trusted computing by applying it to IaaS environments, especially within the security model of an untrusted IaaS provider. Trusted computing offers capabilities to securely perform data manipulations on remote hardware owned and maintained by another party by potentially preventing the use of untrusted software on that hardware for such manipulations. The presented design is directly applicable to the process of developing a trusted virtualized environment, e.g. a public IaaS service.

Future research recommendations can be grouped into three categories:

First is the extension of the trust chain to other operations on VM instances (migration, suspension, updates, etc.), as well as data storage and virtual network communication security.

The second category includes addressing certain assumptions of the proposed launch protocol, e.g. the assumption that the $\mathcal{CH}$ configuration is not changed after the trusted launch of the VM instance, since even in the case of a bona fide IaaS provider the $\mathcal{CH}$ can be compromised through runtime process infection. A technique to enable $\mathcal{C}$ to either directly or through mediated access discover such events and protect the data used by the VM instance is a promising research topic.

The third category focuses on the design and implementation of the evaluation policies of the TTP. The current assumption is that the TTP has access to information regarding "secure" configurations and the PCR values, something which needs to be directly addressed as evaluating exactly how secure a certain software stack is, is a challenge. Furthermore, taking into account the diversity of available libraries as well as the different combinations in which they can be loaded during the boot process, verification of PCR values (such as values stored in `PCR10` and reference values in `binary_runtime_measurements`) becomes a less trivial task.

References

1. Smith, J., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann (June 2005)
2. Krutz, R.L., Vines, R.D.: Cloud Security: A Comprehensive Guide to Secure Cloud Computing. John Wiley & Sons (August 2010)
3. Somorovsky, J., Heiderich, M., Jensen, M., Schwenk, J., Gruschka, N., Lo Iacono, L.: All Your Clouds Are Belong to us: Security Analysis of Cloud Management Interfaces. In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security. CCSW '11, New York, NY, USA, ACM (2011) 3–14
4. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In: Proceedings of the 16th ACM Conference on Computer and Communications Security. CCS '09, New York, NY, USA, ACM (2009) 199–212
5. Pohlmann, N., Reimer, H.: Trusted Computing - eine Einfhrung. In Pohlmann, N., Reimer, H., eds.: Trusted Computing. Vieweg+Teubner (2008) 3–12 10.1007/978-3-8348-9452-6_1.
6. Neisse, R., Holling, D., Pretschner, A.: Implementing Trust in Cloud Infrastructures. In: Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on. (may 2011) 524 –533
7. Sadeghi, A.R., Stüble, C., Winandy, M.: Property-Based TPM Virtualization. In Wu, T.C., Lei, C.L., Rijmen, V., Lee, D.T., eds.: Information Security. Volume 5222 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2008) 1–16 10.1007/978-3-540-85886-71.
8. Danev, B., Masti, R.J., Karame, G.O., Capkun, S.: Enabling Secure VM-vTPM Migration in Private Clouds. In: Proceedings of the 27th Annual Computer Security Applications Conference. ACSAC '11, New York, NY, USA, ACM (2011) 187–196
9. Santos, N., Gummadi, K.P., Rodrigues, R.: Towards Trusted Cloud Computing. In: Proceedings of the 2009 Conference on Hot Topics in Cloud Computing. HotCloud'09, Berkeley, CA, USA, USENIX Association (2009)
10. Aslam, M., Gehrmann, C., Rasmusson, L., Björkman, M.: Securely Launching Virtual Machines on Trustworthy Platforms in a Public Cloud - An Enterprise's Perspective. In Leymann, F., Ivanov, I., van Sinderen, M., Shan, T., eds.: CLOSER, SciTePress (2012) 511–521
11. Aslam, M., Gehrmann, C., Björkman, M.: Security and Trust Preserving VM Migrations in Public Clouds. In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), TRUSTCOM, Liverpool (2012)
12. Mell, P., Gance, T.: The nist definition of cloud computing. Technical report, National Institute of Standards and Technology (September 2011)
13. Goyal, P.: Application of a Distributed Security Method to End-2-End Services Security in Independent Heterogeneous Cloud Computing Environments. In: Services, 2011 IEEE World Congress on. (july 2011) 379 –384
14. Trusted Computing Group: TCG Specification, Architecture Overview, revision 1.4. Technical report, Trusted Computing Group (2007)
15. Boyko, V., MacKenzie, P., Patel, S.: Provably secure password-authenticated key exchange using diffie-hellman. In Preneel, B., ed.: Advances in Cryptology – EUROCRYPT 2000. Volume 1807 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2000) 156–171
16. Price, M.: The Paradox of Security in Virtual Environments. Computer **41**(11) (November 2008) 22–28
17. Wojtczuk, R., Rutkowska, J., Tereshkin, A.: Attacking intel trusted execution technology. In: Black Hat USA 2008, August 7th, Las Vegas, NV. (2008)
18. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest We Remember: Cold-Boot Attacks on Encryption Keys. Commun. ACM **52** (May 2009) 91–98

19. Schiffman, J., Moyer, T., Vijayakumar, H., Jaeger, T., McDaniel, P.: Seeding Clouds With Trust Anchors. In: Proceedings of the 2010 ACM Workshop on Cloud Computing Security. CCSW '10, New York, NY, USA, ACM (2010) 43–46
20. Molnar, D., Schechter, S.: Self Hosting vs . Cloud Hosting : Accounting for the Security Impact of Hosting in the Cloud. In: Workshop of the Economics of Cloud Security. (2010) 1–18
21. Chen, Y., Paxson, V., Katz, R.: The Hybrex Model for Confidentiality and Privacy in Cloud Computing. Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley (January 2010)