Domain-Based Storage Protection (DBSP) in Public Infrastructure Clouds

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

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

Abstract. Confidentiality and integrity of data in Infrastructure-as-a-Service (IaaS) environments increase in relevance as adoption of IaaS advances towards maturity. While current solutions assume a high degree of trust in IaaS provider staff and infrastructure management processes, earlier incidents have demonstrated that neither are impeccable.
In this paper we introduce Domain-Based Storage Protection (DBSP) a data confidentiality and integrity protection mechanism for IaaS environments, which relies on trusted computing principles to provide transparent storage isolation between IaaS clients.
We describe the building blocks of this mechanism and provide a set of detailed protocols for generation and handling of keys for confidentiality and integrity protection of data stored by guest VM instances. The protocols assume an untrusted IaaS provider and aim to prevent both malicious and accidental faulty configurations that could lead to breach of data confidentiality and integrity in IaaS deployments.

1 Introduction

Following a period of establishment and early adoption, cloud computing is gaining widespread popularity and is now present in the product portfolio of many large software vendors in one of the three archetypes outlined by the US National Institute of Standards and Technology (NIST): Infrastructure-as-a-Service , Platform-as-a-Service or Software-as-a-Service [1]. Other factors which testify to the impact of the field are the emergence of legal frameworks that regulate provisioning and usage of public cloud computing services [2] and protection of data transferred to public cloud storage [3]. However, despite growing popularity, cloud computing continues to present a wide range of unsolved security concerns [4, 5, 6]

Considering that security concerns were long cited as barriers to wider adoption of public cloud services, emerging regulation will likely require public cloud providers to operate with an even wider set of tools to safeguard when needed the confidentiality, integrity, authenticity and even geolocation of data stored in public clouds. Governmental programs, such as e.g FedRAMP in the USA propose a "standardized approach to security assessment, authorization, and continuous monitoring for cloud products and services" [7]. While such programs are an important extension of the cloud security ecosystem, they often assume manual execution steps which can not reliably exclude audit or reporting errors due to human factors. In addition, the outcome of such programs is a certification result based on a *snapshot view* of the public cloud providers' infrastructure, processes and policies, while an adversary with full logical access to the underlying infrastructure

can conceal traces of an eventual security breach. Thus, while security assessment and continuous monitoring of the infrastructure are valuable tools in ensuring security of IaaS infrastructure deployments, we consider that effective *prevention* mechanisms have a higher potential to increase the IaaS customers' trust with respect to data processing and storage in public IaaS environments.

A core enabling technology of IaaS is system virtualization [8], which enables hardware multiplexing and redefinition of supported hardware architectures into software abstractions. This redefinition is performed by the hypervisor, a software component that abstracts the hardware resources of the platform and presents a virtualized software platform where guest virtual machine (VM) instances can be deployed. In addition, the hypervisor also manages the I/O communication between VM instances and external components, including storage devices allocated to the VM instance. This is one of the vulnerable areas of IaaS environments since, as demonstrated in [6], improper allocation of block storage can lead to a breach of data confidentiality.

Certain aspects of IaaS security have been addressed through the use of Trusted Computing technologies as defined by the Trusted Computing Group (TCG) [9]. A core component in the TCG-defined security architecture is the Trusted Platform Module (TPM), a cryptographic module that offers protected storage for sensitive parameters and can be used as trust anchor for software integrity verification in open platforms. TPM usage and deployment models for IaaS clouds have already been addressed in earlier research [10, 11, 12, 13, 14, 15]. The early principles of a trusted IaaS platform [13] were later extended to cover both trusted VM launch [14, 16] and VM migration [15].

These results demonstrate the capabilities resulting from combining basic TPM attestation mechanisms with standard cryptographic techniques to design an infrastructure for VM protection. However, while much of the research effort has been directed towards protection of VM instances in IaaS environments, ensuring the protection of data generated by such instances has received far less attention. We address this aspect in the current paper.

Contribution

The focus of this paper is DBSP, a trusted storage protection mechanism which provides per-VM instance access control, allowing the client to control a VM instance's read and write access rights over a storage unit at launch time.

- In this paper we introduce an approach to ensuring confidentiality and integrity of stored data in public IaaS deployments with the additional capability of domain-based isolation. Such *Domain-Based Storage Protection* allows a IaaS Compute Host (CH) to encrypt and integrity protect data before it is stored. Encryption and integrity protection is performed using TPM-protected keys which are only available to a CH in a trusted state, which excludes the possibility of decryption and/or modification on a simulated deployment. Furthermore, DBSP allows to enforce storage management policies to provide control over allocation of and access to storage in a fully virtualized environment.
- We introduce a storage allocation protocol that reduces the risk of accidental or premeditated breach of data isolation between different tenants (an attack vector introduced in [17] with actual vulnerabilities described in [6]) by introducing and enforcing the concept of *administrative domains* in the context of storage resources.

- We present a set of protocols for transparent full disk encryption performed at the hypervisor level; while hypervisor-based background encryption has been explored earlier [18], our protocol focuses on a different key handling mechanism where the control over the domain master keys protecting the data storage is transferred to an external trusted party.
- We extend previously introduced protocols for trusted launch of VM instances in public IaaS environments [14, 16] by introducing additional parameters to direct the allocation of storage resources to a certain *administrative domain*.

2 System Model

In this paper we assume an IaaS deployment model as defined by NIST, where an IaaS client is able to provision processing, storage, networks, and other fundamental computing resources as well as able to deploy and run arbitrary software supported by the hypervisor. [1]; moreover, the same definition explicitly states that IaaS clients do not have control over the underlying infrastructure. In a typical usage scenario, IaaS clients communicate over an insecure network with the IaaS platform which provisions computing resources and launches guest VM instances[3] and allocates storage resources.

According to our system model, the domain of the IaaS provider is limited to the IaaS software platform and the hypervisor environment (including the hypervisor itself, any administrative domains, e.g. Dom0 according to the Xen hypervisor model [19] and the communication channels between administrative domains and the VM instances). Practical IaaS deployments assume that the VM image repository and data storage provided for the VM instances could be either controlled by the IaaS provider or by a third party. We assume for simplicity (but without affecting the applicability of DBSP) that the IaaS provider is in full control of both the image store and the data storage. The IaaS provider domain is marked with bold dashed lines in Figure 1.

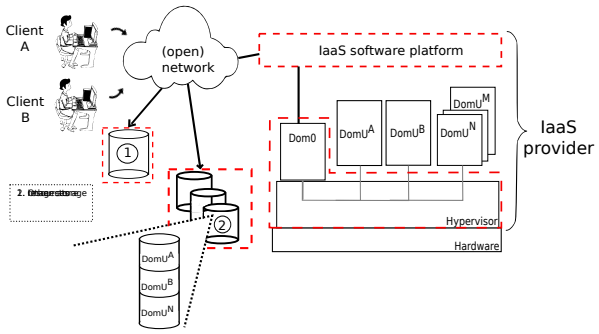


Fig. 1: Data flow in the cloud

We share the attack model with [13, 14, 15, 16], which assume that privileged access rights can be maliciously used by IaaS provider system administrators ($\mathcal{A}_r$) with remote access. In addition, $\mathcal{A}_r$ can obtain root access on any host maintained by the IaaS provider, but not can not obtain physical access.

[3] VM images can originate from the clients themselves, the IaaS provider or a public image repository.

We assume that an $\mathcal{A}_r$ obtaining remote root access to a compute host within the secured IaaS provider perimeter will not be able to access the volatile memory of any VM instance residing on the compute host at that time, i.e. the compute host offers VM instances a closed box execution environment[4], e.g. similar to the model in [20]. The attack model also includes unintentional configuration errors caused by $\mathcal{A}_r$, such as incorrect allocation of storage devices or unintended network connectivity between physical or virtual devices.

Runtime attacks on the hypervisor are excluded from the model, since they represent a separate research topic. One promising solution towards this problem is presented in [21]. Denial-of-Service attacks are also explicitly excluded from our model, because according to the definition of the IaaS model in [1] the client has limited access to the networking infrastructure of the IaaS deployment and the IaaS provider could start a DoS attack simply by severing the network communication between the client and the VM instance.

We consider a VM trusted if the integrity of the VM image used for launch is ensured, the VM instance is spawned on a trusted compute host and the VM instance can prove knowledge of a client-verification token (see section 4.1).

Based on the model presented above, we define a set of requirements towards a solution which aims to ensure the confidentiality and integrity of data processed and stored by a VM instance in an untrusted IaaS setting.

1. The solution must ensure integrity and confidentiality protection of data processed and stored by VM instances on resources hosted by an untrusted IaaS provider.
2. The solution must be capable to enforce access rights, such that a guest VM only can access a certain storage domain if explicitly assigned by the IaaS client.
3. The solution must prevent both accidental and intended breaches of storage resource isolation between VM instances triggered by IaaS $\mathcal{A}_r$s.

These requirements will be revisited in section 5 as part of the evaluation of the proposed solution.

3 Building Blocks

Before presenting the set of protocols comprising DBSP, we provide some details about essential components of the proposed solution and component specific properties which we rely on in the remainder of this paper.

3.1 Trusted Platform Module

The Trusted Platform Module (TPM) is a cryptographic coprocessor, developed according to the specifications of the Trusted Computing Group (TCG) [9]. Given that the final specification for TPM version 2.0 is not yet released at the time of writing, we assume TPM version 1.2 for the remainder of this paper.

TPM provides a set of standard, unmodifiable functionality implemented by vendors according to the specifications published by the TCG and offers data protection through asymmetric cryptography using internally maintained keys. Two of the operations supported by the TPM that are particularly relevant for the proposed solution are *bind* and

[4] This does not include any VM instances part of the hosting infrastructure, such as administrative VMs

seal. According to [9], a message encrypted ("bound") using a particular TPM's public key can only be decrypted using the private key of the same TPM. Sealing is a special case of binding, where an encrypted message produced through binding can only be decrypted in a certain platform state (defined through the platform configuration register values) to which the message is *sealed*. Refer to [9] for a detailed description of the bind and seal operations.

3.2 Trusted Third Party

For the purposes of the protocol, we introduce a standalone component referred to as trusted third party (TTP). We assume that the security guarantees provided by the trusted third party with regard to guest VM launch and storage protection are sufficient for the IaaS client. We further assume that the IaaS provider allows communication between its servers and the TTP for platform attestation and key management purposes. We continue by enumerating of the functionality assumed to be provided by the TTP:

- Communication with components deployed on the compute host, such as integrity attestation information, authentication tokens and cryptographic keys;
- Integrity attestation based on the integrity attestation quotes provided by the TPM hardware component installed on the compute host;
- Verification of client supplied electronic signature authenticity;
- Sealing of data to a certain trusted compute host configuration;
- Generation of nonce values and of confidentiality and integrity protection keys according to the input data received from the compute host.

Given the central role the TTP plays in our model, we assume that the TTP communicates with the components of the IaaS deployment through reliable channels.

3.3 Secure Component

Another part of the proposed solution is a verifiable execution module referred to as the "Secure Component" (SC) in the protocols. The secure component provides the following functionality:

- Communication with the TTP for authentication and cryptographic key exchange;
- Verification of VM instance access rights to storage resources;
- Fetching, caching and storing confidentiality and integrity protection keys per guest VM instance;
- Encryption, decryption, integrity protection and verification of data written or read to/from allocated storage resources;

Having defined the responsibilities of SC, we turn to the possible ways to integrate it into currently used virtualization stacks. Figure 2 shows a Xen hypervisor deployed on a hardware node, with a set of boxes on top of it representing the guest VM instances ($DomU^1$ to $DomU^N$), along with an administrative VM instance, Dom0. While implementing SC as part of Dom0 is feasible, this would only increase the (already large) amount of code that must be included in the trusted computing base.

An alternative implementation of SC on a Xen virtualization platform follows the disaggregation principles described in [22] to implement a "DomSC" executing the functionality of the secure component described above. The trusted computing base is thus be

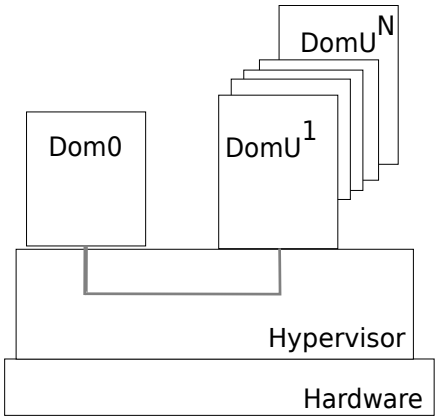


Fig. 2: Typical representation of a Xen hypervisor

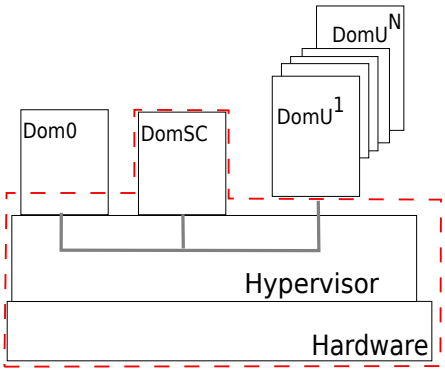


Fig. 3: Representation of a Xen hypervisor in the proposed model

reduced to the underlying hardware, the bare-bones hypervisor, a necessary minimum of Dom0 and DomSC, as depicted in Figure 3 (the TCB within the dashed line). While full exclusion of Dom0 from the trusted computing base is indeed desirable, it is a non-trivial task, as discussed in [22].

Implementation(s) of the *SC* for Xen and/or KVM is planned as future work.

4 Design Principles

We assume a scenario where data is stored in an IaaS storage using any suitable units, such as block storage devices (iSCSI or similar). Confidentiality and integrity of the data during storage is ensured by the secure component, as described in section 3.3.

All data stored at the IaaS provider using the scheme described in this paper is associated with specific *storage domains*. A storage domain in this context typically corresponds to a particular organization or administrative domain that utilizes public cloud services (including the storage service) offered by an IaaS provider, i.e. a single administrative entity that typically only handles data storage for its own domain and not for any other domains. All data in a single domain is protected with the same storage protection domain master key, denoted by K_M. This key is generated by the TTP and cannot leave TTP's logical perimeter.

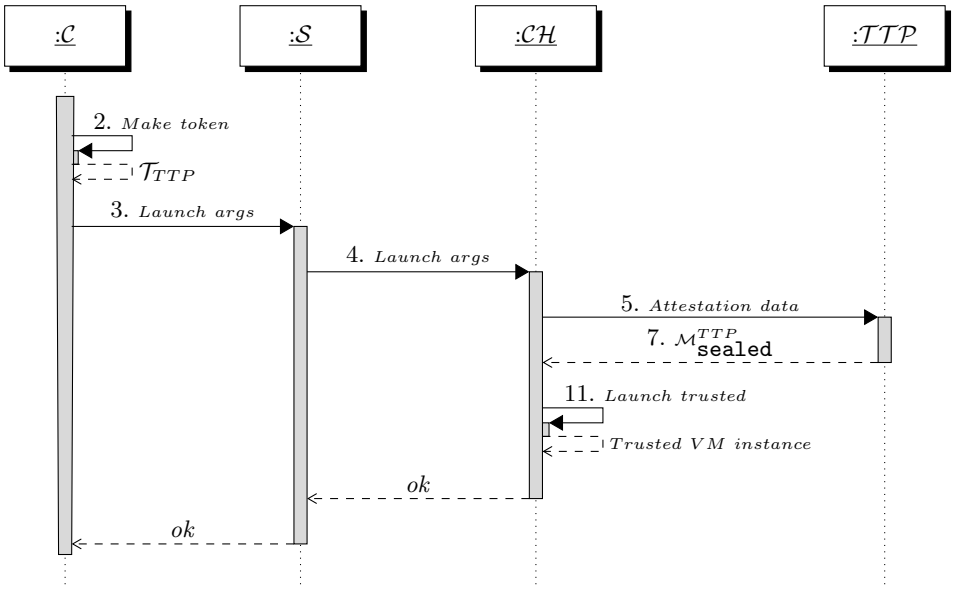
We assume that at guest VM launch, the VM instance is assigned a unique identifier (ID_{VM}). During the entire lifetime of the VM instance, ID_{VM} is reliably associated with a particular storage domain. Keys used for data confidentiality and integrity protection and verification in a single domain are derived by the TTP.

Below we describe three protocols necessary for the data handling functionality of a VM instance, namely protocols for secure VM instance launch plus initial and subsequent storage usage. Migration of VM instances is a relevant and important topic, but due to space considerations it is out of the scope of this paper.

4.1 VM Instance Launch

We suggest the following principles for securely associating a VM instance with a particular storage domain at VM launch. It is important to note that the following launch protocol description (also presented in Figure 4 focuses mainly on the extensions to the trusted launch protocol in [16] and does not revisit all its details. In the following description, the extensions to the protocol are marked with a **bold** font.

Fig. 4: Trusted VM launch protocol: $\mathcal{C}$: Client; $\mathcal{S}$: Scheduler; $\mathcal{CH}$: Compute Host; $\mathcal{TTP}$: Trusted Third Party;



1. Client C prepares a VM launch package similar to the one described in [16] or [14]. The launch package contains a launch message, $\mathcal{M}$, which consists of the following parameters:
   (a) **The identifier of the VM to be launched, ID_{VM}.**
   (b) **A storage domain identifier, ID_D.** For this protocol assume $ID_D = A$.
   (c) **An assertion, $\mathcal{AS}$, proving to the TTP, that C is authorized to request the launch of VM instances with access to storage domain A.**[5]
   (d) A nonce ($\mathcal{N}$) encrypted with the public key of the TTP (PK_{TTP}); denote the resulting encrypted nonce by $\mathcal{N}_{PK_{TTP}}$
   (e) Optional additional parameters, such as required target platform Security Profile (SP) and a hash H_{VM} of the target VM image [6].

[5] We assume here that an assertion in the SAML format is used.

[6] Here we assume that an unmodified, "vanilla" VM image available from a public image repository is used. The protocol can be adapted for client-customized images, by encrypting the image with a symmetric key K, protecting K with PK_{TTP} and including that into $\mathcal{M}$.

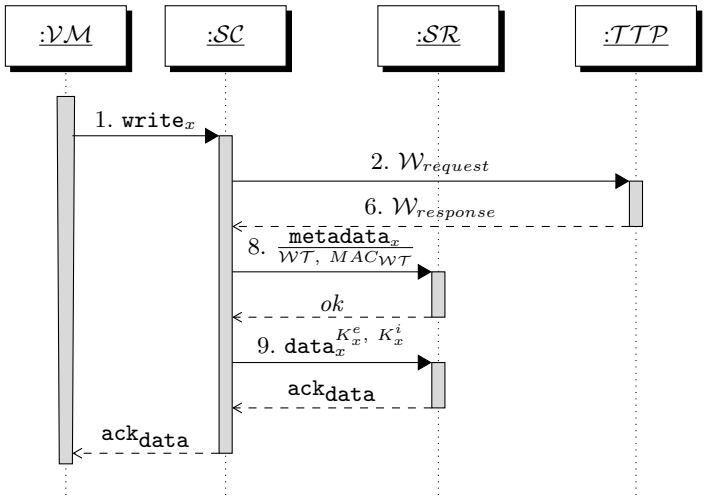
2. **The client produces a digital signature, $\mathcal{SIG}$, over all the values in $\mathcal{M}$ using the client's Private Key (PrK_C), with the corresponding public key certified in $CertC$.** Denote the data structure containing $\mathcal{M}$, $\mathcal{SIG}$ and $CertC$ by $\mathcal{T}_{TTP}$.
3. $\mathcal{T}_{TTP}$ is sent to the IaaS provider along with VM_{image} or an indication of the VM Image (ID_{VMimg}) that should be chosen for launch from a publicly available VM image repository.
4. The scheduler (S) selects a suitable available compute host in the provider network and transfers $\mathcal{T}_{TTP}$ and VM_{image}/ID_{VMimg} to the chosen compute host.
5. Once the compute host receives $\mathcal{T}_{TTP}$, it sends $\mathcal{T}_{TTP}$ to the TTP for verification.
6. The TTP follows the below steps to verify the contents of the received $\mathcal{T}_{TTP}$, attest the trustworthiness of the compute host and generate the necessary keys:
   (a) **TTP verifies $CertC$ and the signature $\mathcal{SIG}$ and if they are valid, TTP proceeds to the next step; otherwise it aborts with an error message to CH.**
   (b) **TTP checks the assertion, $\mathcal{AS}$ (using the client's identity and key information provided in $CertC$) and verifies that the client is authorized to use storage domain A**[7]. If that is true, TTP proceeds with next step, otherwise it aborts with an error message to the compute host.
   (c) Using its private key, TTP decrypts the received $\mathcal{N}_{PK_{TTP}}$ contained in $\mathcal{M}$.
   (d) **TTP generates a *session domain key* for the domain specified by ID_D (in this example we assume domain "A") and the target platform. We denote this key by SDK_A.**
7. **Parameters ID_{VM} and SDK_A** (together with other parameters such as $\mathcal{N}$ and H_{VM}, similar to the mechanism in [16]) **are TPM *sealed* to a protected state of the compute host and only made available to a trusted state of the compute host.** The encrypted message, denoted $\mathcal{M}^{TTP}_{\texttt{sealed}}$, is sent back to the compute host, which concludes the trusted launch. We maintain our earlier assumption that C has requested the launch of a publicly available VM image and provided H_{VM} for verification:
8. The compute host *unseals* $\mathcal{M}^{TTP}_{\texttt{sealed}}$ and **ensures SDK_A is available to the secure component running on the platform.**
9. The compute host compares the received H_{VM} with the hash of the available VM image to ensure its integrity.
10. **The VM is assigned ID_{VM} and is launched in a secure isolated execution compartment on the trusted platform.**
11. The compute host injects $\mathcal{N}$ into the VM image prior to launching the VM instance, launches the VM instance and returns an acknowledgement to the client to confirm a successful launch.
12. To verify that the VM instance has been launched on a trusted platform, the client challenges the VM instance to prove its knowledge of $\mathcal{N}$.

4.2 Initialization And First Time Data Writes

The protocol for set up and first time data write is presented in Figure 5 and explained in detail below.

[7] We assume that remote attestation of the compute host will also be performed at this point; however this is not included in the current description

Fig. 5: Secure block write procedure: $\mathcal{VM}$: VM instance; $\mathcal{SC}$: Secure component; $\mathcal{SR}$: Storage resource; $\mathcal{TTP}$: Trusted Third Party;



1. The VM instance on the compute host requests access to a new storage resource, e.g. a block device or database in the provider network; the storage resource is denoted by SR.
2. The storage resource reference is denoted SR_R. Using SR_R as reference, the VM specifically requests a data write to a storage entity, x, in SR (this being a block or other storage structure).
3. This request is intercepted or received by the secure component.
4. The secure component sends, protected under key SDK_A, a write request ($\mathcal{W}_{request}$) to the TTP for new storage entity keys for entity x, domain A and SR_R from the VM instance identified by VM_{ID}. The request is confidentiality and integrity protected using SDK_A [8].
5. The TTP executes the verification steps outlined below:
   (a) TTP verification, using SDK_A, that $\mathcal{W}_{request}$ is correct, which includes a verification of the domain access rights of the key SDK_A. The protocol execution only proceeds if the key SDK_A is associated with the requested domain.
   (b) If so, TTP fetches the master key K_M for the requested domain A and generates a nonce $\mathcal{N}^{TTP}$.
   (c) Next, TTP uses a suitable pseudo-random function, $PRF(K_M,\ \mathcal{N}^{TTP})$ to generate data encryption and integrity protection keys. In this way, the generated keys are associated with a specific domain indicated by the domain identifier provided by the customer. The VM_{ID} is associated with the domain for ancillary purposes, such as billing.
   (d) Denote encryption and integrity protection keys by K_x^e and K_x^i respectively [9].
   (e) Next the TTP generates a token ($\mathcal{WT}$) consisting of $\mathcal{N}^{TTP}$, A and SR_R.

[8] Several alternatives for confidentiality and integrity protection are applicable using well-established protocols, e.g. TLS with pre-shared keys.

[9] In certain cases, *only integrity* or *only confidentiality* of data is required and thus one of the two keys suffice. Here we assume both confidentiality and integrity protection is needed.

(f) The TTP uses an internal integrity key, which never leaves the logical domain of the TTP, to calculate an integrity check value over $\mathcal{WT}$, denoted as $MAC_{\mathcal{WT}}$.

6. Next, the write response token $\mathcal{W}_{response}$ (containing $\mathcal{WT}$, $MAC_{\mathcal{WT}}$, K_x^e, K_x^i) is confidentiality and integrity protected using SDK_A [10] and sent to the secure component.
7. The secure component receives $\mathcal{W}_{response}$ from the TTP, decrypts $\mathcal{WT}$, $MAC_{\mathcal{WT}}$, K_x^e and K_x^i and associates them with domain A and VM_{ID}.
8. The secure component stores $\mathcal{WT}$ and $MAC_{\mathcal{WT}}$ received from TTP as part of storage metadata for the new storage entity x in SR.
9. The secure component uses keys K_x^e and K_x^i to protect the confidentiality and integrity of the data stored in storage entity x in SR.

Performance and Efficiency Considerations The storage entity unit should be selected so that the communication frequency between the secure component and TTP on one hand and the secure component's activities on the other hand do not incur a larger performance penalty than what is acceptable by the involved parties. Also, the storage entity unit should be selected so that integrity protection meta-data does not consume a larger portion of storage than what is acceptable by the involved parties. [11]

4.3 Subsequent Data Reads and Writes

The protocol for subsequent data reads and writes is introduced and explained in detail below; a high-level view of the key retrieval protocol is presented in Figure 6.

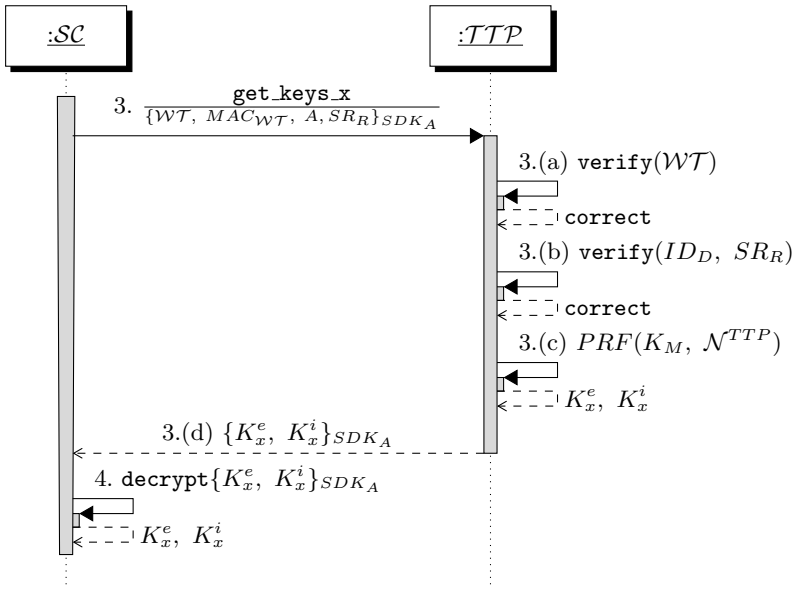
The VM identified to the compute host by VM_{ID} requests a data write or data read from entity x using SR_R as a reference handle. This request is intercepted or received by the secure component and the following procedure applies:

1. The secure component checks if the required integrity and confidentiality keys needed to verify and decrypt the requested storage entity x are cached locally. If that is the case, it proceeds to step 5. Otherwise, the protocol executes the following steps:
2. First, it locates $\mathcal{WT}$ and MAC_{WT} used to protect x in SR in the meta data of storage entity x.
3. The secure component sends to the TTP a read-write request ($\mathcal{RW}_{request}$) containing $\mathcal{WT}$, $MAC_{\mathcal{WT}}$, A, SR_R and the VM_{ID}, and a request for the data entity keys for x. The write request is confidentiality and integrity protected under key SDK_A[10]. The TTP executes the following steps to provide the necessary encryption and integrity protection keys:
   (a) The TTP verifies the correctness of the $\mathcal{RW}_{request}$ and of the token $\mathcal{WT}$ (using its own internal MAC key). Similar to the initialization protocol, the TTP verifies the domain access rights associated with SDK_A and only proceeds if the key SDK_A is associated with the domain identifier requested by the secure component.
   (b) If $\mathcal{WT}$ is valid, TTP checks that the domain identifier and SR_R contained in $\mathcal{WT}$ match the ID_D and SR_R indicated by the secure component.

[10] Several alternatives for confidentiality and integrity protection are applicable using well-established protocols, e.g. TLS with pre-shared keys.

[11] We plan to investigate the relation between the storage entity unit size and performance penalty in a prototype implementation.

Fig. 6: key retrieval procedure for subsequent data reads and writes: $\mathcal{SC}$: Secure component; $\mathcal{TTP}$: Trusted Third Party;



(c) If all the above verifications are successful, TTP uses the K_M domain master key and $\mathcal{N}^{TTP}$ in $\mathcal{WT}$ to derive K_x^e and K_x^i for VM instance VM_{ID}.

(d) Next, the TTP sends to the secure component the keys K_x^e and K_x^i in a read-write response message ($\mathcal{RW}_{response}$) which is confidentiality and integrity protected using SDK_A[10].

4. The secure component receives $\mathcal{RW}_{response}$ from TTP and decrypts the keys.
5. The secure component uses K_x^e and K_x^i to encrypt and/or integrity protect (write) or decrypt and/or integrity check (read) data at storage entity x.

5 Security Evaluation

To assess the solution, we first discuss the involved components and the expected security properties and continue with a brief discussion of the protocol verification using ProVerif, a cryptographic protocol verifier [23].

In the system model currently considered, integrity of *VM images* is universally important, while confidentiality is mostly relevant in the case of client-customized VM images. In the proposed solution, integrity is ensured by calculating $\mathcal{SIG}$ of the H_{VM} on the client side and verifying it on the compute host at the time of launch. We apply previously introduced principles for trusted *VM instance* launch [14, 16] in order to ensure that the respective VM instance has been launched on a trusted compute host, i.e. on a compute host running a trusted code base, attested by a TTP. The TTP has, within

the scope of this protocol, extensive responsibilities and must itself be protected from software attacks[12].

A key assumption of the protocol is the reliance on an attested and trusted compute host, which is performed by the TTP using *Direct Anonymous Attestation* [25] defined in version 1.2 of the TCG specification.

Integrity verification of the stored data is performed using integrity keys K_x^i, which are generated by the TTP. Key generation requires the presence of the correct session domain key SDK, which is sealed to the trusted configuration of the compute host. According to TPM properties, the sealed SDK can only be decrypted by the compute host if it is in the trusted state the key was sealed to [9]. Consequently, a compute host booted into an untrusted state (configuration) or a malicious third party would be unable to obtain K_x^i. The same mechanism protects the confidentiality protection key K_x^e. Security of the keys K_x^e and K_x^i regenerated for subsequent reads and writes is ensured by the integrity verification of the token $\mathcal{WT}$ created by the TTP and stored in the meta data.

Enforceable access rights management is ensured by the requirement for the VM launch process to present an assertion $\mathcal{AS}$ generated by the client to the TTP. If no such assertion is available, the VM launch process is aborted by the TTP (the IaaS can still launch a VM instance, however such an instance would not be *trusted* by the IaaS client). Furthermore, possession of a session domain key for the respective domain generated in the process of trusted launch is necessary to obtain the integrity and confidentiality protection keys K_x^e and K_x^i. Thus, a VM instance which does not possess the client-provided $\mathcal{AS}$ for a certain domain would not complete a trusted launch procedure and would not obtain the session domain key for the respective administrative domain. The step requiring the VM launch process to present a client-generated assertion during the guest VM launch procedure to obtain a session domain key which is in turn necessary for data access operations satisfies requirement 2 by enforcing access rights based on the credentials provided by the IaaS client.

Domain isolation of data is cryptographically enforced by the TTP in several steps. First, session domain keys SDK_A are generated based on the information received from the client, in particular assertion (proving to the TTP that the client is authorized to launch VM instances with storage access to a certain domain) and the client certificate. Subsequent generation of confidentiality and integrity protection keys K_x^e and K_x^i is only done by the TTP based on the possession of a correct session key for the respective domain. A VM instance can thus only obtain read or write access if it has been launched in the same domain and the respective assertion has been provided by the IaaS client. Malicious or accidental allocation of the respective storage resource to an arbitrary VM instance would not have any effect on the confidentiality of the data protected under K_x^e. Such cryptographic isolation satisfies requirement 3 stated above.

5.1 Protocol Verification with ProVerif

We have verified the security properties of the proposed protocol using ProVerif, an automatic cryptographic protocol verifier in the formal (Dolev-Yao) model [23] [13]. The verification has demonstrated the confidentiality of both the stored data and consequently

[12] Anecdotal cases, such as NIST taking the National Vulnerability Database (NVD) offline in response to learning that it had been hacked [24] point to the fact that attestation services (such as the TTP) are important attack vectors and must therefore be closely monitored.

[13] The ProVerif script is available at https://github.com/nicopal/dbspVerification

of the confidentiality protection keys[14], as well as of the TTP-generated nonce that is necessary in order to regenerate the confidentiality and integrity protection keys K_x^e and K_x^i, demonstrating that requirement 1 has been satisfied.

6 Related Work

The importance of data confidentiality protection and isolation of data between tenants in a IaaS environment is underlined by the attention it has received from the research community. However, many public IaaS providers still harbour "low hanging fruits" in terms of vulnerabilities, such as for example the one addressed in [6], when researchers have identified vulnerabilities in the disk allocation procedure whereby the disk space allocated to a certain tenant would contain fragments of information written by other previous tenants. This particular vulnerability was caused by the fact that the hosting OS's file API, which by default zeroes uninitialized data, was not used in the disk allocation process. Our approach prevents such cases by encrypting the data prior to writing it to disk. Management of allocated disk space according to security domains is another barrier that would prevent an attacker from gaining access to disk space potentially containing remnants of data. In this scenario, in case an improperly sanitized disk is allocated to a guest VM from a different administrative domain, the guest VM would only access encrypted data.

While the authors in [6] suggest full disk encryption as one of the mitigation techniques, management of encryption keys is not trivial and has been the focus of a large body of research. For example, [26] focuses on managing access rights upon shared versioned encrypted data on cloud infrastructure for a restricted, flexible group. The authors base their model on several components, namely a Key Graph, encrypted updates to the Key Graph (denoted as Key Trails) as well as actual versioned data, where the latter two are stored in the untrusted cloud and the first one is stored with a trusted third party. Key Trails are used to both adapt on the fly the Key Graph based on granted or revoked data access, as well as format for deltas between two versions of the Key Graph. This model focuses on key management, leaving the encryption and decryption operations to the clients of the cloud storage. The approach builds on earlier research in the area, namely [27] and utilizes the generation of encrypted key updates by storing Key Trails on highly available and scalable but untrusted cloud infrastructures parallel to the encrypted data. All keys are versioned equivalent with the data, in order to allow a rather granular data access control, where the client can access a certain version or all previous versions of the data. The authors also describe a potential extension of the scheme that would allow a granular, per version client access to the data. While this approach is attractive in many ways, especially considering the granular control of data, the requirement for client-side encryption limits the applicability of the scheme for client guest VMs that (for any reason) do not have the ability to run custom confidentiality/integrity protection code. In addition, our proposed solution allows protection of persistent data at storage in an IaaS deployment almost transparently from the client point of view.

The trusted hypervisor approach has received much attention in the research community, as builds on the idea of separating resource allocation from resource isolation, such that a specialized, trusted hypervisor is deployed on ring 0 below the commodity

[14] The verification model assumes confidentiality protection also includes integrity protection so not separate integrity verification of data was modelled

hypervisor and protects the memory space of a guest VM from an untrusted commodity hypervisor. This is done without intentionally interfering with resource allocation, which is usually left to the commodity hypervisor, hence the separation between resource allocation and isolation. Variations of this scenario include eliminating the commodity hypervisor as a whole and relying on a trusted hypervisor with reduced functionality (e.g. support of a single guest VM). Below follow two examples of this approach, which could be used in combination with the protocols described in this paper. BitVisor (introduced in [28] and further in [29]) is a thin hypervisor based on Intel VT-x and AMD-V designed to enforce I/O device security of virtualized guests. The hypervisor uses a parapass-through architecture that allows to forward a subset of the I/O instructions (keyboard and mouse actions) without modification in order to have a minimal impact on the performance of the VM instances. The approach describes a method to offer access to both encrypted and unencrypted areas of the disk in a manner transparent to the VM instance. Different from other approaches, BitVisor assumes a minimal hypervisor functionality which facilitates deployment efforts.

While it introduces several novel ideas and has a code implementation which also has been extended by other researchers, BitVisor trades the ability to have concurrently executing VM instances for simplicity and ease of installation. This limitation severely reduces the applicability of BitVisor in virtualized IaaS environments, where hardware multiplexing is an important requirement.

In [30] the authors propose a full disk background encryption model by introducing TCVisor, a BitVisor-based hypervisor with a parapass-through architecture which introduces a novel key-management approach and TPM support. Support for TPM is added in order to store parts of cryptographic keys and whole-disk checksums for integrity checking. The authors use Merkle trees for integrity verification and protect the root value relying on TPM functionality. However, the exact procedure of storing or sealing the root value of the Merkle tree hash is not discussed. A modified version of AES is used for data encryption; however the undisclosed modifications to AES raise concerns about the necessity of modifying a standard verified encryption algorithm and about the effects the of introduced modifications. The authors also examine the topic of key revocation and propose an aggressive key revocation scheme triggered by user input. The approach suggested in the paper does indeed address some aspects of protecting privacy-sensitive data in IaaS storage. However, by building TCVisor on top of BitVisor, the model inherits the limitations of BitVisor, e.g. support for only one executing VM instance.

As mentioned above, the DBSP protocols presented in this paper can be applied as an extension of trusted hypervisor approaches, since similar to such hypervisors, DBPS protocols require external attestation from a third party.

7 Conclusion

In this paper we have introduced a set of complementary protocols intended to ensure transparent domain-based isolation between data stored by guest VMs. Transparency is ensured by introducing a 'secure component' SC, which is trusted by the hypervisor. This secure component performs key management on the compute host side, along with background confidentiality and integrity protection of stored data. We furthermore introduce domain-based isolation, which uses symmetric encryption to ensure that guest VMs only obtain data read or write access if they are authorized to do so by the IaaS client. We rely on trusted computing principles and earlier defined trusted VM launch

protocols in order to ensure that guest VM instances are only launched on trusted IaaS compute hosts. We perform a theoretical security evaluation of the proposed solution. Description and evaluation of an implementation of the solution on either the Xen or KVM virtualization platforms are left for future work.

References

1. Mell, P., Grance, T.: The NIST definition of cloud computing (draft). NIST special publication **800** (2011)
2. The 112th US Congress: Cloud Computing Act of 2012, S. 3569 (112th) (2012)
3. European Commission: Regulation of the European Parliament and the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data. In: C7-0025/12. (January 2012)
4. 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
5. 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
6. Jordon, M.: Cleaning up dirty disks in the cloud. Network Security **2012**(10) (2012) 12–15
7. U.S. General Services Administration Industry Advisory Council: Federal Risk and Authorization Management Program (FedRAMP). http://www.gsa.gov/graphics/staffoffices/2012_01_11_ACT_IAC_FedRAMP_FINAL.pdf (2012)
8. Smith, J., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Processes, Morgan Kaufmann (June 2005)
9. Trusted Computing Group: TCG Specification, Architecture Overview, revision 1.4. Technical report, Trusted Computing Group (2007)
10. 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
11. 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.
12. 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
13. 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)
14. 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
15. 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)
16. Paladi, N., Gehrmann, C., Aslam, M., Morenius, F.: Trusted launch of virtual machine instances in public iaas environments. In Kwon, T., Lee, M.K., Kwon, D., eds.: Information Security and Cryptology – ICISC 2012. Volume 7839 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 309–323

17. Jansen, W., Gance, T.: Guidelines on security and privacy in public cloud computing. Technical report, National Institute of Standards and Technology (December 2011)
18. Omote, Y., Chubachi, Y., Shinagawa, T., Kitamura, T., Eiraku, H., Matsubara, K.: Hypervisor-based background encryption. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing. SAC '12, New York, NY, USA, ACM (2012) 1829–1836
19. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. ACM SIGOPS Operating Systems Review **37**(5) (2003) 164–177
20. Jin, S., Ahn, J., Cha, S., Huh, J.: Architectural support for secure virtualization under a vulnerable hypervisor. In: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, ACM (2011) 272–283
21. Azab, A.M., Ning, P., Wang, Z., Jiang, X., Zhang, X., Skalsky, N.C.: Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In: Proceedings of the 17th ACM conference on Computer and communications security, ACM (2010) 38–49
22. Murray, D.G., Milos, G., Hand, S.: Improving xen security through disaggregation. In: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, ACM (2008) 151–160
23. Blanchet, B., et al.: An efficient cryptographic protocol verifier based on prolog rules. In: In 14th IEEE Computer Security Foundations Workshop (CSFW-14. (2001)
24. www.sans.org: National vulnerability database taken offline after malware is found on servers. Technical report, SANS NewsBites - Volume: XV, Issue: 21 (2013)
25. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM conference on Computer and communications security, ACM (2004) 132–145
26. Graf, S., Lang, P., Hohenadel, S., Waldvogel, M.: Versatile key management for secure cloud storage. Submitted at EuroSys **11**(11.4) (2012) 2012–13
27. Waldvogel, M., Caronni, G., Sun, D., Weiler, N., Plattner, B.: The versakey framework: Versatile group key management. Selected Areas in Communications, IEEE Journal on **17**(9) (1999) 1614–1631
28. Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., et al.: Bitvisor: a thin hypervisor for enforcing i/o device security. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, ACM (2009) 121–130
29. Omote, Y., Chubachi, Y., Shinagawa, T., Kitamura, T., Eiraku, H., Matsubara, K.: Hypervisor-based background encryption. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, ACM (2012) 1829–1836
30. Rezaei, M., Moosavi, N., Nemati, H., Azmi, R.: Tcvisor: A hypervisor level secure storage. In: Internet Technology and Secured Transactions (ICITST), 2010 International Conference for, IEEE (2010) 1–9