A Protocol for Preventing Insider Attacks in Untrusted Infrastructure-as-a-Service Clouds.

Khan, Imran; Anwar, Zahid; Bordbar, Behzad; Ritter, Eike; Rehman, Habib-ur

DOI:
10.1109/TCC.2016.2560161

A Protocol for Preventing Insider Attacks in Untrusted Infrastructure-as-a-Service Clouds

Imran Khan, Zahid Anwar, Behzad Bordbar, Eike Ritter, and Habib-ur Rehman

Abstract—Recent technical advances in utility computing have allowed small and medium sized businesses to move their applications to the cloud, to benefit from features such as auto-scaling and pay-as-you-go facilities. Before clouds are widely adopted, there is a need to address privacy concerns of customer data outsourced to these platforms. In this paper, we present a practical approach for protecting the confidentiality and integrity of client data and computation from insider attacks such as cloud clients as well as from the Infrastructure-as-a-Service (IaaS) based cloud system administrator himself. We demonstrate a scenario of how the origin integrity and authenticity of health-care multimedia content processed on the cloud can be verified using digital watermarking in an isolated environment without revealing the watermark details to the cloud administrator.
Finally to verify that our protocol does not compromise confidentiality and integrity of the client data and computation or degrade performance, we have tested a prototype system using two different approaches. Formal verification using ProVerif tool shows that cryptographic operations and protocol communication cannot be compromised using a realistic attacker model. Performance analysis of our implementation demonstrates that it adds negligible overhead.

Index Terms—Cloud Computing, Trusted Computing, Protocol, Late Launch, Digital Watermarking

1 Introduction

Cloud Computing is an exciting and promising new paradigm that allows clients to outsource storage and computational resources on demand. While cloud computing bases on current technologies such as virtualization and service oriented architecture, the major driving factors of this technology are the advancement in machine architecture, the requirement to process and/or maintain large data sets and high bandwidth network channels. Additionally, features such as multi-tenancy, auto-scaling and low cost enables cloud computing to flourish more successfully than its predecessor- the Grid.

One third of the IT company respondents in a recent cloud computing survey [23], stated that they are already using cloud based services. An additional 40% respondent companies are in a transitionary phase towards adopting cloud based services. Recently iCloud has played the role of a crime fighter [24], serving to track down the iPhone of a passenger which was stolen on a cruise ship. In this work, our focus is on the Infrastructure as a Service (IaaS) based cloud model. As IaaS resides at the lowest level, it allows the development of verifiable security solutions and then layer the software stack on top of it.

Companies are adopting cloud based IT solutions as public clouds become the source of a rich and novel range of IT solutions ranging from massive online collaborative content storage to health-care workflow management systems. At the converse, the wide adoption of cloud based services is badly suffering due to confidentiality and security concerns especially from insider attacks [1]. One way to ensure confidentiality in the cloud environment is to constantly store customer data in encrypted form and decrypt it on the cloud platform on the fly when being retrieved or being operated on. However this approach is not practical due to its high computational cost [43][44] and in case of a untrusted cloud platform the confidentiality of the data can be compromised at the point the data is decrypted for computation. Researchers have proposed homomorphic encryption schemes [2], that allow computations to be carried out on encrypted content, producing an encrypted result which, when decrypted, matches the result of operations performed on the plaintext. However so far only primitive operations are supported and there is a large amount of overhead. Moreover there is a strong requirement to make the operations of the IaaS based cloud transparent to clients. That means that clients be able to verify the underlying cloud platform

- *Imran Khan is with the Department of Computer Science, National University of Computer and Emerging Sciences, FAST-NUCES, Islamabad, Pakistan. E-mail: imrankhan@nu.edu.pk.*
- *Zahid Anwar is with the National University of Science and Technology (NUST), Islamabad, Pakistan and with the University of North Carolina at Charlotte, USA. Emails: zahid.anwar@seecs.edu.pk, zanwar@uncc.edu.*
- *Behzad Bordbar is with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham, UK. E-mail: B.Bordbar@cs.bham.ac.uk.*
- *Eike Ritter is with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham, UK. E-mail: eike@cs.rittere.co.uk.*
- *Habib-ur Rehman is with the Department of Computer Science at Al Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, KSA. E-mail: habibr@ieee.org*

- *This work was conducted at and supported by the National University of Computer and Emerging Sciences (FAST-NUCES), Islamabad, Pakistan, National University of Sciences and Technology (NUST), Islamabad, Pakistan and University of Birmingham, Birmingham, UK.*

and services, to ensure that the platform owner is not compromising the integrity and confidentiality of their data and computation.

In current research work [12][11], on cloud platforms security has predominantly focused either on protecting these platforms from malicious cloud clients or on protecting cloud clients from each other's unwanted activities. The problem of protecting clients from the possible malicious acts of insiders such as cloud providers is not adequately addressed. There are organizations, for instance health-care and military, which are hesitant to move to cloud based services due to confidentiality concerns. Therefore practical solutions in this direction are required for wide adoption of cloud based services.

In this paper, we propose an approach to ensure the confidentiality and integrity of client data and computation on the cloud platform. This is to ensure that private data is not exposed to internal parties such as the cloud administrator and other cloud clients. Our approach makes use of remote attestation [4], and a late launch based technique, called Flicker [29], to verify the integrity of the cloud platform. This technique secures the virtual machine (VM) launch operation and further allows the launched VM to perform operations on sensitive data in full isolation. To test our approach, we have implemented a prototype by extending a popular open source cloud computing solution known as Eucalyptus [15]. The extra integrity verification processing overhead of our approach is found to be minimal. To illustrate the practicality of our proposed protocol, we have demonstrated how it can be used to verify presence of a hidden watermark in a health-care multimedia context. This is done in a manner that preserves the confidentiality of the watermark contents and the integrity of the verification process. The contribution of our work is as follows:

- We propose a protocol for secure launch of a client VM on a trusted cloud node. Other than secure launch, our second proposed protocol enables a client to protect the confidentiality and integrity of its data and computation from other client applications in the cloud and from the cloud system administrator.
- In our proposed protocol architecture, the Trusted Computing Base (TCB) is reduced to the size requirement of the Flicker based code executed and its input and output. The software stack from the BIOS up to the virtual machine monitor (VMM) level is *thus* removed from the suggested TCB of client sensitive code executed on the cloud platform.
- In a virtualized cloud environment, past system configuration cannot guarantee current or future trustworthiness of a system. We have shown how to provide assurance to clients in such an environment.
- We have verified the confidentiality and integrity security properties of our proposed protocols using the ProVerif automatic cryptographic protocol verifier. We have also verified that our proposed protocols are secure against man-in-the-middle attacks.

The rest of the paper is organized as follows. Section 2 provides background knowledge about Trusted Computing, cloud virtualization environments and protocol verification. The design and details of our proposed protocols are presented in Section 3. Implementation details are presented in Section 4. Verification of security properties of our proposed protocols is discussed in Section 5. Evaluation is presented in Section 6. Section 7 provides a review of related work and existing research. Finally, we have concluded the discussion on our work in section 8.

2 Problem Background

2.1 Trusted Computing

Major hardware vendors, including Intel, Dell and HP, have founded a consortium called Trusted Computing Group (TCG). The objective of this group is to build trust in computing devices such as PDAs, mobile devices and PCs and to provide a transparent view of the platform software stack to its owner. According to the TCG specifications [20], all electronic devices complying with TCG standards should be equipped with a hardware chip called Trusted Platform Module (TPM) [20]. A TPM is a secure storage area where cryptographic keys and other secure data can be stored. The key and data stored inside the TPM is protected from malicious alteration. The data stored inside the TPM normally includes platform configuration status. The platform status stored inside TPM can then be provided to external entities, through a process called Remote Attestation, to convey platform trustworthiness. The Trusted Computing Base (TCB) of a system is the collection of all hardware, firmware, and/or software modules that are vital for the security of the overall system. Any vulnerabilities occurring inside the TCB can compromise the security of the entire system.

2.2 Remote Attestation

In remote attestation, the platform (firmware and software) configuration is captured and stored in a tamper resistant and cost effective chip called a TPM. Confidential information is held inside the TPM and is then signed and reported to a remote entity for verification and attestation purposes. This entire process is termed as remote attestation by the TCG [20]. In remote attestation, to the TPM chip some form of integrity measurement system such as Linux Integrity Measurement Architecture (IMA) [21], is needed to generate and report the attestation of the system to the remote entity. TPMs [20], store platform integrity in

the form of hashes of loaded software in data registers called Platform Configuration Registers (PCRs). *Quote* operation is used to attest the values of TPM PCRs. TPM *Quote* comprises of a subset of PCRs values together with a nonce all signed by a TPM Endorsement Key (EK). The private part of the EK is used for signing purposes during *Quote* generation and is used to convince remote verifiers that assertions in the TPM *Quote* have been signed by a trusted TPM.

2.3 Virtualization

According to Sempolinski et al. [28], there are six fundamental components of a generic cloud computing stack; (1) hardware and OS, (2) VMM, (3) VM disk image archive, (4) front-end, (5) network and (6) cloud framework. Among these, virtualization is the key enabling technology of an IaaS based cloud. Virtualization, cost effectively abstracts system resources and supports multiple and heterogeneous operating systems simultaneously on a single hardware platform. In addition to the computation resources, the networking resources are also virtualized. Xen and KVM are two of the most popular open source VMMs [28]. A typical VMM generally includes a hypervisor which in turn supports and executes multiple clients VMs. Particularly in case of Xen a special administrative VM called dom0, runs and controls client guest VMs. The dom0 VM runs under the control of a platform owner.

2.4 Late Launch

Late launch [20], commonly refers to technologies that allow the execution of a secure kernel or secure VM on a system after running un-trusted software. This means that the chain of trust is not started from system boot but is rather initiated dynamically at a later stage. Certain family of processors from both Intel and AMD provides an implementation of this technology. Intel named its implementation 'Trusted eXecution Technology (TXT)' [30] while AMD calls their technology 'Secure Virtual Machine (SVM)' [3].

TPM v1.2 allows for dynamic PCRs (PCRs 17-23), which can be reset without rebooting the system. Late launch on Intel systems consists of calling the GETSEC [SENTER] instruction in CPU protection ring0, which takes as an argument a physical memory address range. This memory range is called Measured Launch Environment (MLE). The processor protects the MLE against various attacks through hardware based defenses. The processor disables direct memory access (DMA) to the MLE memory pages. Interrupts and debuggers are also disabled to protect the MLE launch.

To invoke a late launch with SENTER, first of all the Authenticated Code Module or ACMod must be loaded into memory. The ACMod is then executed after the platform's chipset (with its built-in public key) verifies the signature and extends its measurement into PCR 17. ACMod then measures the equivalent of an AMD SLB i.e. Measured Launch Environment (MLE) [30], extends the measurement into PCR 18, and then executes it. For further information, we refer the reader to Flicker [29].

2.5 Sealed Storage

One of the key features provided by the TPM for securing sensitive code and data is *sealed storage*. A TPM contains a special 2048-bit key called Storage Root Key (SRK). The private part of the SRK never leaves the TPM in plaintext. The storage key is used to *seal* other data and sensitive information. The *seal* operation takes a set of PCRs as input, and then encrypts the given data using the SRK. The *seal* operation outputs cipher text C along with the list of PCRs provided and its corresponding values.

The corresponding *unseal* operation takes cipher text C and the PCRs list as input. It then compares the PCRs list against their current values. It decrypts C only if a match occurs and then decrypts the suggested data. All these operations take place inside the TPM.

2.6 Flicker

Flicker [29], is an infrastructure based on late launch technology for secure execution of a small piece of security sensitive code, called Piece of Application Logic (PAL), on systems where BIOS, OS and DMA devices are not trusted. A PAL is a piece of application logic that performs a well defined task. Flicker executes the PAL in full isolation on the system from all other software and hardware (including OS and VMM). This isolation is possible due to hardware enhancements in modern commodity platforms from both Intel and AMD (AMD's secure virtual machine Technology and Intel's TXT). On Intel platforms, invoking a Flicker session suspends the current execution environment (OS and VMM) and then executes the SENTER instruction for setting up the secure environment for PAL execution. At the end of Flicker session, the previous execution environment is resumed.

The main goal of the Flicker architecture is to minimize the mandatory TCB of a security sensitive code. Thus, the attestation provided is both meaningful and provable to a remote party due to a small TCB. Fig. 1 shows the minimization in TCB by using Flicker. Suppose we want to execute App n, which contains security sensitive code S. The left hand side of Fig. 1 shows the execution of App n in a standard model. Here the shaded region shows the suggested TCB of App n. In a standard model the chain of trust starts from machine boot. The suggested TCB in this case is very large and consists of BIOS, boot loader, OS and so forth. The right hand side of Fig. 1, shows the scenario when Flicker is used. In this case, the suggested TCB

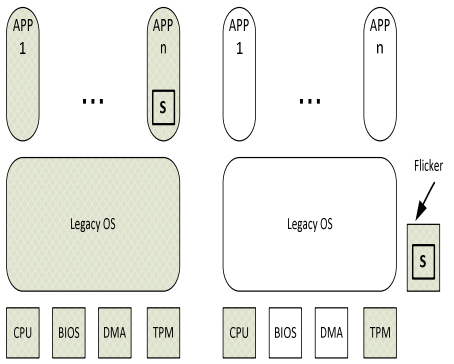


Fig. 1. TCB of applications running with and without Flicker protection [29]

is minimized, and includes CPU (in some situations additional chipsets), TPM and the Flicker framework used to execute security sensitive code *S* in complete isolation.

2.7 Protocol Verification

We will later define security protocols for secure VM launch and secure computation. As the adversary is actively looking for weaknesses, correctness of security protocols is very important. However, significant flaws have been found in widely used protocols, often years after the protocol has been defined. Examples are the Needham-Schroeder public-key protocol [33] which was found to have a serious flaw 17 years later [34], the SAML-based Single Sign-On for Google Apps [35], or the still re-occurring flaws in widely used protocols such as openssl [36] and openssh [37]. Formal models have been developed for reasoning precisely about security protocols in order to detect such flaws. Models based on process calculi (e.g. applied pi-calculus [38]) model the participants in a security protocol as processes and the messages as communication between processes.

Automated tools for the verification of security protocols based on these models have been developed, e.g. ProVerif [39] and SATMC [40]. We will use ProVerif which is well suited to handle the kind of security property required in this paper. We will verify a correspondence property, which states that a certain event is always preceded by another event. This property enforces that the decryption of a virtual machine happens only with a key that the client has sent.

3 DESIGN

In the first part of this section, we will introduce the two definitions "*Level I* security" and "*Level II* security" that are frequently used in the rest of the paper and are very significant to our protocol design.

3.1 *Level I* vs *Level II* security

Trusted Computing and remote attestation enable a remote party to challenge a given platform and verify its security properties remotely. If the current state of a system is successfully verified, the remote party can trust this system for future operations. Here we introduce two definitions related to Trusted Computing.

Definition 1

Level I security: *Platform Integrity Attestation*, where before transferring computation and data, a remote party verifies through remote attestation, that the target platform belongs to the actual cloud hosting provider as well as executes trustworthy hardware, firmware and software. The client can then trust the challenged platform after verifying its current state through remote attestation for future operation.

Definition 2

Level II security: *Integrity and confidentiality*, where a remote party not only verifies the integrity of the target platforms hosting provider, hardware, firmware and software but also requires additional security measures to ascertain that the confidentiality and integrity of sensitive operations executed on the target platform will not be compromised. *Level II* security assurance can be provided with Intel TXT technology based mechanism called Flicker and will be detailed in the coming sections.

Trusted Computing and remote attestation is based on the concept of trust. If we verify the current status of a platform using remote attestation which shows that the platform current status is trustworthy that means that the platform is running some well-known good software stack (configuration). Then traditionally [17][18], on the basis of current trust the platform is trusted for future operations. The basis for this argument is that if the current platform status is well-known and trustworthy then the platform is likely to behave expectedly in the future and hence can be trusted for future operations. Our above *Level I* security definition corresponds to this property.

Now consider a cloud virtualized environment with the system administrator running in dom0 and controlling the overall environment of the cloud Node Controller (NC). After initial attestation to the client, the system administrator can then run any arbitrary process in dom0, get access to client memory and can thus compromise his data confidentiality and integrity. It shows that the client cannot trust the current cloud node configuration on the basis of a previous verification. Therefore, only *Level I* security assurance is not sufficient for the virtualized cloud environment.

According to our proposed protocol, the client first verifies the platform configuration before launching his VM in the cloud. After VM launch, the client computation on normal data is performed as usual in the cloud environment. However, when the client wants some computation on highly sensitive data

then it cannot simply rely only on *Level I* security. As the client has previously verified cloud platform during VM launch then according to *Level I* security it can trust the platform for sensitive computation as well. However, the problem is that the cloud system administrator can run an arbitrary process in dom0 after initial verification and hence can compromise client confidentiality and integrity. Therefore, we propose *Level II* security whereby computation on client sensitive data is performed in full isolation from the cloud system administrator. Such assurance is achieved through our proposed approach based on late launch and the Intel Trusted eXecution Technology (TXT) hardware based mechanism, Flicker. Flicker architecture employs a TPM chip based on Intel's Trusted eXecution Technology (TXT) for storage of session configuration representing a hash of the computation. After launch of the Flicker session the Flicker specific kernel executing user sensitive computation has isolated access to the full platform processing capabilities.

In terms of security requirements, cloud clients can be divided into two broad categories. One category of clients fully trusts cloud providers for its data and encrypted form on the CS and computation. For example, a banking application wants to find the hour of the day at which the highest number of clients have visited the bank. The other category of clients needs strong assurances from the cloud platform owner in order to trust it for its sensitive data and computation. For example, a health-care application wants to find the address of a particular patient without revealing private attributes, such as the type of disease, inside the cloud platform.

We propose that there should be two different sets of infrastructure within a cloud for these two different categories of clients. For the first category, the current cloud offering with isolation provided by the underlying virtualization technology is sufficient. However, for the second category of clients, NCs should be equipped with a TPM chip and have support for late launch mechanism. The primary goal of our proposed protocol architecture is to support the security needs of the second category clients.

Here we present a protocol architecture for secure VM launch and for ensuring a secure execution environment for client sensitive data and computation inside a client guest VM on a cloud platform. Fig. 2 shows a general cloud design. The client VMs are stored on Cloud Storage (CS). For a security sensitive client, its corresponding VM will be stored in encrypted form on CS. The Client VM executes on the NC whereas the Cloud Controller (CC) provides an interface to the client. For the protocol details described in this section, we assume the case of a security sensitive client on an Intel machine.

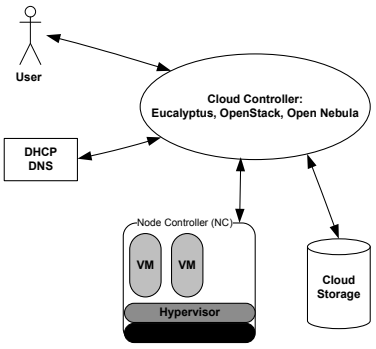


Fig. 2. Generalized architecture of a cloud environment

3.2 Secure VM Launch

The client VM is stored in an encrypted form on the CS, so that it can only be launched on trusted NCs. The purpose of the secure VM launch protocol is to get the VM decryption key Dk_{VM} securely from the client, decrypt the VM and then launch it on a trusted node.

The protocol proceeds in two phases. In the first phase, we certify the public keys of the client (pk_c) and Flicker (pk_f). This is performed by using the TPMs of the client and the NC to establish a secure channel between the client and the Flicker session-using the *secure channel* protocol of Flicker [29].

In the first step, the client sends a request to the NC for its VM launch. When the NC receives this request, it initiates a Flicker session and executes PAL P and extends PCR 18 with the measurement of PAL P and with its input and output (Flicker session configurations). P is an application code used to generate Flicker asymmetric keys and support the VM decryption process. Here we represent the private and public portion of the Flicker asymmetric key by Pk_f and pk_f respectively. The private part of the asymmetric key Pk_f generated inside Flicker is then *sealed* for the subsequent invocation of the same Flicker session. The purpose of this asymmetric key pair is to get the VM decryption key Dk_{VM} securely from the client. In the next step, NC then sends the Flicker public key pk_f and the TPM *Quote* of the system to the client. When NC generates a *Quote*, it includes the value of PCR 18 in the *Quote* operation and hence *Quote* reflects the Flicker session configurations signed with the TPM private key. Attestation from the Flicker session can convince the client that the PAL P executed inside Flicker protections and that the public key pk_f was a valid output of the session. After verifying the TPM *Quote* the client then establishes a secure channel to the Flicker session.

Inside the Flicker session, PCR 18 is extended with the measurement of PAL and with its input and output (Flicker session configurations). The output in this case is the public part of the Flicker asymmetric

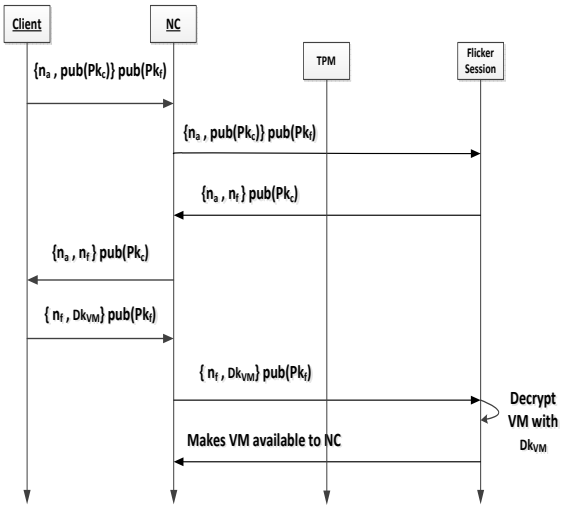


Fig. 3. Secure VM launch protocol

key pk_f generated inside the Flicker session. In this way, extending the measurement of PAL output into PCR 18 enables the NC to convince the client that the Flicker asymmetric key pk_f was indeed generated inside Flicker protection. The client can check the value of PCR 18 in the TPM *Quote* to verify whether a legitimate PAL was executed and the asymmetric key pk_f was indeed generated inside the Flicker session. The client can verify this by re-computing the SHA1 hash of the PAL and its output (asymmetric key in this case) and matching it with the value of PCR 18. If the NC self generates the asymmetric key pk_f and sends it to the client, the client will see a difference in the value of PCR 18 from the *Quote* sent by the NC.

The second phase consists of the protocol to securely communicate the VM decryption key Dk_{VM} to the Flicker session and is illustrated in Fig. 3. Here Pk_c and Pk_f are the private keys of the client and the Flicker respectively and pub is the function generating the public key from the private key. n_a and n_f are nonces which are used to protect against replay attacks. According to our protocol, the client sends NC the nonce n_a and its public key pk_c, both encrypted with the Flicker public key pk_f. Therefore, the message can only be decrypted inside the Flicker session. The NC then initiates a Flicker session with the client message as its input.

Inside the Flicker session, the nonce and the public key of the client pk_c are verified. After verification, the Flicker generates a new nonce n_f. The nonce n_f, the public key of the client pk_c and the private key of the Flicker session Pk_f are stored in the TPM using *sealed* storage. Flicker then returns nonces n_a and n_f encrypted with the client public key pk_c so that only the corresponding client can decrypt the message. The NC then forwards the message to the client. The client then verifies nonce n_a from the message to

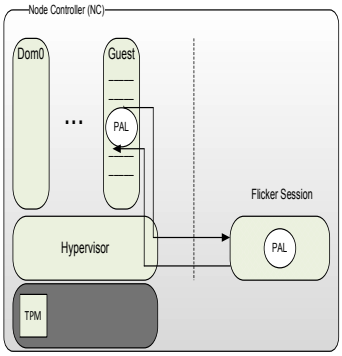


Fig. 4. Confidentiality sensitive computation

make sure that the message was indeed sent from the corresponding Flicker session. The client then constructs a message with the VM decryption key Dk_{VM} and nonce n_f encrypted with Flicker public key pk_f. The message is then forwarded for the corresponding Flicker session to the NC.

The NC then captures the message and initiates another Flicker session and provides the client message as its input. Inside the Flicker session the nonce n_f, the public key of the client pk_c and the private key of Flicker Pk_f are read from *sealed* storage. The nonce n_f is verified and then the VM is decrypted with the decryption key Dk_{VM}. The decrypted VM is then made available to the NC which launches the VM and connects the client to its VM. The *sealed* storage is used as described in Section 2.5 above, ensuring that *only* same Flicker sessions can read or write this storage.

3.3 Confidentiality Sensitive Computation

After secure VM launch through the protocol proposed in section 4.2, the next step is to ensure the confidentiality of client sensitive data and computation. Here we present a protocol for confidentiality sensitive computation based on *Level II* security.

The computation inside the security sensitive client VM is divided into two categories, normal computation and security sensitive computation. The normal computation takes place as usual on the virtualized platform with the system administrator running in dom0. The same procedure however, cannot be followed for security sensitive computation as an insider such as the system administrator can run arbitrary processes in dom0 and can compromise the confidentiality of client data and computation. The integrity information exchanged during VM launch cannot guarantee this protection.

In our proposed protocol the computation on sensitive data is organized as PAL, which executes inside Flicker protection. The sensitive data is only visible to the client PAL inside the Flicker session and is processed there in full isolation from the rest of the system, as shown in Fig. 4. In this way, confidentiality

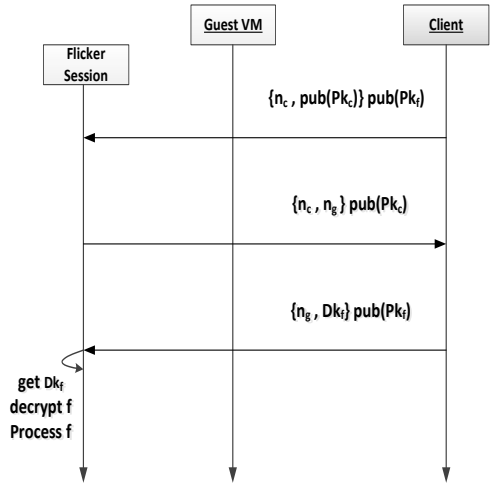


Fig. 5. Confidentiality sensitive computation protocol

of client data is enforced. Consider as way of illustration that the client has a confidential file f stored in encrypted form on the NC. He wishes to perform some computation on f, without revealing its content to cloud NC system administrator.

Similar to the protocol used for secure VM launch, we first establish a secure channel between the client and the Flicker session using the TPM. The detailed protocol which is used afterwards is shown in Fig. 5. The client starts by forwarding a nonce n_c and pub(Pk_c) to its VM on the Node Controller (NC). The client VM then executes PAL C inside Flicker protection. After verifying nonce n_c and client public key pk_c, the Flicker then sends nonce n_c and its nonce n_g encrypted with the client public key pk_c to its VM running on the NC. The client VM then forwards the message from the Flicker session to the client application.

After verifying nonce n_c, the client then forwards the message containing file f decryption key DK_f along with nonce n_g encrypted with Flicker public key pk_f to the Flicker session on the NC. The client guest VM on the NC captures the messages, initiates the Flicker session and forwards the message as input to the session. In this way, the DK_f will only be acquired by PAL C executed inside the Flicker session. The client VM subsequently executes PAL C inside Flicker protection and decrypts file f with DK_f. File f is then processed inside the Flicker session and the result is then returned to the client VM after processing f inside Flicker protection. The entire process shows that f is only visible in plaintext inside the Flicker session. In this way, the client can perform sensitive computation on the cloud node in full isolation from the underlying NC software stack. In this way, client confidentiality is ensured, protecting the data from other guest VMs and the NC system administrator.

3.4 Privacy-preserving watermark verification of health-care data on the cloud

We present a scenario based on the proposed approach for ensuring patient privacy by protecting data confidentiality and integrity of the watermark [42] verification process on an untrusted cloud for laboratory test results in the health-care domain. Watermarking is a method of secretly transferring digital information through a carrier signal, such as an image. Here we consider as an example multimedia content from a particular health-care scenario. A public figure such as a politician (John) is admitted to clinic C for medical treatment. There is a strong requirement that the particulars of the medical condition of the visiting politician be kept highly confidential. John's physician at C decides that Magnetic resonance imaging (MRI) of John's head is required for diagnosing his ailment and as a result John visits laboratory L for his head MRI.

C hosts health-care application, P for managing patient data on a cloud platform. Johns' MRI image must be transferred from L's database to P in order to diagnose Johns' disease. As the cloud platform is potentially untrusted, there are two main requirements for the MRI image to be transferred from L to P. Firstly, in order to ensure confidentiality of Johns' medical data, the MRI image should be transferred from L to P in such a manner that the information-watermark in the MRI image should only be viewable to P. Secondly, the application will necessitate non-repudiation or undeniable information (a proof) from L that the MRI image for patient John was indeed forwarded by L. This is necessary for C, so that in case of a conflict, a wrong MRI image (for example a forgery) sent by the laboratory or an attacker, it may be easily detected.

Using digital watermarking, L secretly transfers to P, the MRI image along with a unique watermark M as proof of origin. The pseudo code for hidden watermark transfer along with Johns' MRI image is shown in the Table 1. In the given pseudo code, L first creates a watermark M that combines the patient ID and the Lab-ID of L. The watermark M is then embedded into MRI image I using Watermark encoding algorithm A and session key K, creating a watermarked image $\bar{I}$. Session key K is used in the embedding process so that only the receiver with key K can decode the image for the given watermark. For non-repudiation and integrity check, L signs the hash (SHA1) of $\bar{I}$ and stores it in $\bar{i}$. For secure transfer, key K is encrypted with P's public key pk_p and stored in sec. L then forwards the watermarked image ($\bar{I}$), signs watermark image hash ($\bar{i}$) and forwards the encrypted key (sec) to P.

P first verifies the signature and then the integrity of the received watermark image by re-computing hash of $\bar{I}$ and comparing it with $\bar{i}$. As the cloud

Notation:
I: Image to be watermarked; $\bar{I}$: Watermarked image; K: Session key
A: Watermarking algorithm; P-ID: Patient ID; Lab-ID: L laboratory ID
Pk_l: Private key of L laboratory; pk_l: Public key of L laboratory
Pk_p: Private key of P cloud application
pk_p: Public key of P cloud application

Pseudo code:
L laboratory:
M ← (P-ID, Lab-ID) // Generate a watermark
$\bar{I} \leftarrow$ encode.A(I, K, M) // Embed a watermark
$\bar{i} \leftarrow Pk_l$ (hash($\bar{I}$)) // For non-repudiation and integrity check
sec← pk_p(K) // Encrypt K with P public key
L laboratory $\xrightarrow{\bar{I},\ \bar{i},\ sec}$ P cloud application

P cloud application:
P verifies signature and applies integrity check using $\bar{I}, \bar{i}, pk_l$
Flicker($\bar{I}$, *sec*) // P initiates flicker session
K ← Pk_p(sec) // Gets session key K to decode $\bar{I}$
M ← decode.A($\bar{I}$, K) // Gets embedded watermark M
Gets P-ID // Gets patient ID
Gets Lab-ID // Gets Lab-ID of L laboratory

TABLE 1
Digital watermarking application running under Flicker-based isolation

platform is untrusted, P then checks the watermark inside the Flicker session using the following procedure: sec is first decrypted using P's private key Pk_p, to get key K. Watermark M is then obtained by using the watermark decoding algorithm A, session key K and watermark image $\bar{I}$. The patient ID and Lab ID is then verified from the obtained watermark M. As watermark is obtained and checked for the patient ID and Lab ID inside the Flicker session, its value remains confidential from the cloud platform administrator, other super users and clients.

4 IMPLEMENTATION

In this section, we present a brief description of our proof-of-concept implementation of the proposed protocol architecture.

For the realization of remote attestation, we used the most popular and widely used approach called Integrity Measurement Architecture (IMA) [21]. IMA is based on binary attestation. When configured, IMA calculates and extends hashes of all software components loaded after the boot process into relevant PCRs. To preserve privacy of the NC, we have used the Attestation Identity Key (AIK) for signing PCRs values during the *Quote* generation. We have used a Trusted Java [6] based software stack known as Java Trusted Software Stack (jTSS) for communicating with the TPM through the TPM driver.

We have used the open source IaaS based cloud-Eucalyptus for our testing due to availability of its various modular features such as CC and NC. The design of Eucalyptus [15], is an open-source answer to the commercial Amazon EC2 cloud and is API-compatible with EC2. Other open source IaaS based cloud systems such as OpenStack [25], Nimbus [27] and OpenNebula [26], may also be used. Most of these platforms have the ability to be deployed as an overlay on top of highly decentralized resource configurations, for instance multiple clusters, workstation pools, distributed storage, and locally stored running virtual disks. Eucalyptus in particular has a hypervisor-agnostic architecture and supports two well known hypervisors, Xen and KVM. The majority of Eucalyptus components have well defined web-service based interfaces (described by WSDL documents) and are developed using standard Web-services packages such as Apache, Axis2 and Rampart.

The NC is the core component of the Eucalyptus cloud where client VMs execute and is a major focus of our discussion. In our particular implementation the Eucalyptus NC is an HP elitebook 8560 laptop with 2.2 GHz processor and 4 GB of primary memory. NC is running Linux kernel 2.6 in dom0 and Xen is used as VMM. Eucalyptus NCs are divided into two groups, one for normal clients and one for security sensitive ones. For security sensitive clients, the NC is also equipped with a TPM chip and supports the Intel TXT technology.

As a typical cloud based VM can be up to 2 GB in size, it will be computationally infeasible to encrypt or decrypt the entire VM. A VM normally consists of three images: a boot disk image, a kernel image and an initial ramdisk image. Therefore, from a performance prospective, it is desirable to encrypt only that portion of a VM image which is important from a security point of view. The kernel is the most fundamental part of the operating system of a VM, and supports user level application requests via system calls. We want to ensure that a client VM is running with a client provided trusted kernel. Therefore, instead of encrypting the entire VM, we encrypted only the kernel. The Linux kernel we used in our experimentation was 50 MB in size. Therefore, here in our discussion, whenever we mention VM encryption or decryption, we are only referring to the kernel.

The Eucalyptus NC is the central point of our discussion, as client, VMs actually execute on the NC. In the original Eucalyptus design, the NC receives a request from the CC for VM launch which then launches the VM from the CS. However in case of the security sensitive group of NCs, the VM to be launched is stored in encrypted form on the CS. After reading an encrypted VM from the CS, the NC first executes a special PAL *P* inside the Flicker protection,

by calling the SENTER instruction. P is trusted by the client and includes the functionality needed to support getting the VM decryption key Dk_{VM} from the client. After getting the VM decryption key Dk_{VM} using the protocol described in the previous section, the NC then launches the VM and connects the client to its VM.

According to default Flicker implementation, the PAL can have a maximum size of 512 KB and can have input and output as a maximum size of 116 KB. We have made changes to the Flicker code to allow it to read encrypted VM as input and to return a decrypted VM as output. We then pass the VM image and its decryption key as input to the Flicker session for decryption, so that it can be launched after returning from the Flicker session. The VM is then decrypted inside the Flicker session, and is made available to the NC as output of the Flicker session. The decrypted VM is then launched by the NC. In this way, VM launch is restricted to trusted NC only.

5 VERIFICATION

The suggested protocol ensures confidentiality and integrity of the clients' data. We use ProVerif [39] to automatically verify that the data is not revealed to the attacker and that the protocols proposed do not suffer from man-in-the-middle attacks. Checking these two properties is sufficient to ensure confidentiality because we show that only the client can access the data, and secondly no other principal can masquerade as the client. We make the following assumptions:

- The attacker has access to all communication (except the communications on private channels). We assume that the cloud provider may be part of the attacker.
- The attacker may modify, replay and re-arrange messages but not break cryptography.
- Neither the Flicker session nor the TPM nor the client is compromised. However, the attacker can interact with the Flicker session.
- The public keys of the client and Flicker can be used to provide a secure communication channel between the client and the Flicker session. For this purpose we use the TPM as described in section 2.2.

ProVerif is based on the applied pi-calculus [38], which models a security protocol as follows: Agents are modeled as processes, and messages between agents are modeled as communication between processes. The attacker is modeled as the environment and hence has access to all communication (except the communications on private channels).

ProVerif transforms the protocol specified as a process into Horn clauses. The Appendix at the end of this paper lists the ProVerif code. There are two files, one for the verification that only the client can access the data, and the other one for the verification that man-in-the-middle attacks are not possible. The given code consists of the following parts:

- Description of the channels involved in the protocols (line 3-7).
- Description of the cryptographic operations such as symmetric and asymmetric encryption (lines 8-19).
- The property that only the client can access the data is formalized as the property that the attacker does not get access to the virtual machine of the client (line 23).
- The absence of a man-in-the-middle attack is modeled by the *query* whether one event (the Flicker session has received the key for the virtual machine) is always preceded by another unique event (the client has sent the key for the virtual machine) (line 22-29).
- Description of client and the Flicker session (lines 31-56).

Since we assume that the attacker has access to ring 0 and therefore has full access to the node controller, the node controller is effectively part of the attacker. Hence there is no agent (process) corresponding to the node controller in our model.

When we run ProVerif on both files, it terminates very quickly and shows that both queries are true. All three protocols were verified and satisfy confidentiality.

The integrity of the data is guaranteed as the virtual machines are encrypted with a key which is known only to the client.

6 EVALUATION AND DISCUSSION

Although we conducted our experimentation on Intel based machines but the work can easily be adapted to an AMD based architecture. On an AMD architecture, the *SKINIT* instruction is used to invoke a Flicker session whereas in our Intel based machine the Flicker session is started with *GETSEC* [SENTER]. However the security property and protection provided by a Flicker on both machines is similar. On an Intel machine, the attestation requires at least PCRs 17 and 18 to be sure that a given PAL is executed inside Flicker protection. While in case of an AMD architecture it is reduced to only PCR 17. The reason is that an AMD architecture does not use a chipset module for launching a Flicker session.

Our approach provides security at two different levels with varying TCB size. Before launching a VM, the client attests the integrity of the entire platform and the suggested TCB starts from the BIOS up to the VMM. This attestation depicts the current behavior of the platform. After trusting the current platform behavior and VM launch, our approach takes care of protecting client confidentiality and integrity on

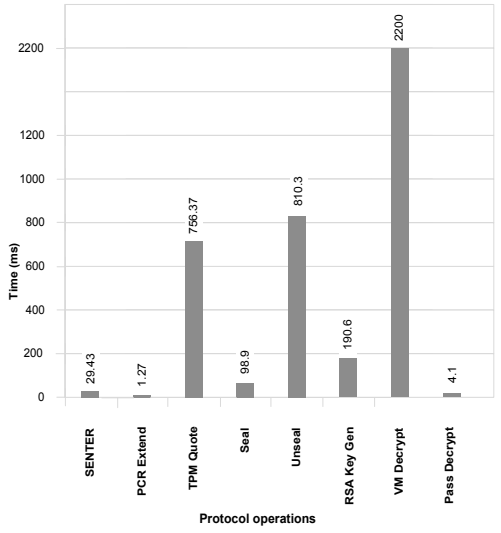


Fig. 6. Performance evaluation

the NC from the cloud clients and cloud system administrator through isolated Flicker based execution. The property of our isolated Flicker based sensitive computation is that it has very small TCB. The suggested TCB consists only of Flicker code, its input and output, and the client PAL. In our confidentiality sensitive computation protocol, input to the Flicker session is an encrypted file and Flicker output is the result of performing some computation on that file (decrypted inside Flicker). So for client based sensitive computation, it is not required to trust the software stack from BIOS up to the VMM.

Initially a guest VM needs two Flicker sessions to perform computation on sensitive encrypted data. The first session is used to get the decryption key of encrypted confidential data from the client. The second Flicker session is then used to process the given data. Subsequent operations on the same confidential data will need only a single Flicker session, as the guest VM has now the key for data decryption and performing operation on data inside the Flicker session.

The Flicker session suspends the normal execution of a cloud node and runs in full isolation. This can be considered a type of financial loss to the platform owner because the platform resources are not fully and efficiently utilized. In order to avoid the suspension of normal execution on a node, McCune et al. [14] suggest an architecture to have only a subset of CPU cores assigned to a Flicker session while normal execution continues on other cores. This way secure and normal execution will take place at the same time on the same cloud node. Further research is desirable in this direction in order to enable execution of sensitive and normal code side by side on the same platform.

We call an outsource computation as sensitive if it operates on some confidential data. According to our protocol, confidential data is only processed inside Flicker protection. So sensitive computation can only process sensitive data inside Flicker protection. As a result, confidentiality of both data and sensitive computation is preserved. According to our proposed protocol, sensitive computation executes as PAL inside Flicker protection. Therefore a client can enforce through remote attestation used by our protocol that only client trusted outsourced computation can access client confidential data. Hence, clients can send decryption key for confidential encrypted data to only those Flicker sessions whose computation the client trusts.

In this paper, our main focus is on protection of the PAL from the cloud system administrator and all other software running in the system. Here we discuss how to protect the Node Controller (NC) from a malicious PAL. The NC can only allow execution of legitimate PALs, by verifying it in some manner, e.g. using proof carrying code [9]. X86 architecture has 4 privilege rings, with ring 0 and ring 3 being most and least privileged respectively. As GETSEC [SENTER] is a privileged instruction used to launch a Flicker session, therefore a Flicker session can only be invoked by code executing in CPU protection ring 0. In this way, the NC allows execution of only legitimate PALs that it trusts or verifies in some manner [9].

Inside a Flicker session, paging is being disabled and segmentation is enabled, and the PAL is executed in CPU protection ring 3. The NC sets relevant segment register [29], to define the memory region the PAL can access. The PAL executes in CPU ring 3 protections and hence it can only access the memory defined in the segment registers. We have extended the default input/output size to allow for VM encryption/decryption. In this way the PAL can only access the memory limited by segment registers and cannot access the memory region of other processes and VMs.

Here we consider the performance aspects of our protocol. The performance intensive operation can be categorized into those that occur inside the Flicker session and those that occur outside. The most expensive operation outside the Flicker session was the TPM *Quote* operation which took 756.37ms (millisecond), as shown in Fig. 6. The Flicker session was invoked on the NC with the *SENTER* instruction with a total execution time of 29.4ms. PCR extend operation was used to extend the measurement of PAL into PCR 18 which took 1.27ms. The asymmetric 1024-bit RSA key for the Flicker was generated in 190.6ms. The *sealing* and *unsealing* of the private part of the RSA key was performed in 98.9ms and 810.3ms respectively. The VM key was decrypted using 1024-bit RSA key in 4.1ms. The PAL then decrypts the VM with client supplied 128-bit Advance Encryption Standard (AES) symmetric key which took 2.2 seconds.

As mentioned earlier, our audience is security sen-

sitive clients whose main concern is security and confidentiality of their data and computation. Therefore, the tradeoff overhead is acceptable for the isolation and security features obtained by using our proposed protocol.

7 RELATED WORK

Our work benefits from related work in trusted computing, trusted virtual machine monitors and specialized encryption based approaches. Several approaches in the first two categories are based on reliance on a trusted hypervisor also known as trusted virtual machine monitor (TVMM) [17][18]. The suggested TVMM model prevents platform administrators and other super users from examining or modifying client VMs. TVMM has the property of being able to defend its own integrity on the deployed platform. Hypervisors are unfortunately complex software and as a result inflate the Trusted Computing Base (TCB). Due to hypervisors' complexities and large TCB there are many challenges that need to be addressed for the emergence of a practical TVMM.

Approaches that rely on a Trusted Computing Base

We further classify approaches that make use of a TCB into two generic categories, i.e. trusted virtual machine monitor and modified VMM. Approaches in the first category build their solution on top of an existing TVMM assuming that it would provide the desired properties of confidentiality and integrity and a root of trust. Whereas modified VMM based approaches detail certain modifications to the design of existing hypervisor's for providing desired root of trust.

Trusted virtual machine monitor based approaches

Terra [17] is one of the initial and influential research works that aims to provide secure closed box execution environment and an open general purpose system side by side on a single platform. Terra is based on a TVMM to make the function of a single node communicating in a distributed environment, transparent. This approach protects client computation integrity and confidentiality by providing a closed-box execution environment to a user VM from the platform administrator interference and inspection. The TVMM provides an interface to a management VM for allowing the allocation of memory, storage and other resources to the client VM. TVMM allows remote parties to trust the client VM closed-box execution environment by providing the attestation of the environment to the remote parties.

A technique for achieving client computational integrity and data confidentiality in an IaaS based cloud is presented in Khan et al. [22]. The technique is based on using remote attestation and a trusted virtual machine monitor (based on a TVMM). A trusted third party is used for the establishment of trust between cloud clients and cloud nodes. The cloud provider first attests and registers its nodes with a trusted third party which in turn verifies the cloud platform properties using remote attestation.

Modified VMM based approaches

Murray et al. [19], analyzed the Xen architecture and proposed architectural recommendations for the emergence of a TVMM based on the Xen hypervisor. In order to reduce the TCB and make the attestation more meaningful, they presented a technique based on disaggregation property in a Xen based environment. The technique moves the domain building process of the administrative domain into a special domain called DomB which is removed from the TCB.

Another approach called Self-Service Cloud (SSC) [32], attempts to restrict the privileged administrative domain in Xen from examining client VM computation and data. SSC divides administrative rights into a per-client administrative domain and system-wide domain. The given model provides clients the ability to manage privileged system operations related to their own VMs.

CloudVisor [31], uses nested virtualization for preventing the administrative domain from encroaching upon the integrity and confidentiality of client VMs. CloudVisor divides the virtualization functionality into VM management and security protection. The management VM is responsible for management of client VMs while a tiny security monitor that runs beneath a commodity VMM such as Xen provides protection of client VMs integrity and confidentiality. Nested virtualization however imposes high overheads because privileged operations must be handled by both the bare-metal and nested hypervisor's, slowing down I/O intensive client applications.

TVEM [10], provides a virtual environment on a cloud platform to ensure protection of client data and computation. TVEM is a software based appliance and is supported by hardware based solutions, i.e. Intel Virtualization Technology for Directed I/O (VT-d) [13] and Trusted eXecution Technology (TXT). Separate responsibilities are assigned to the information owner and the service provider in the proposed virtual environment. Virtual environment is the software component encompassing layers from the operating system to the application software in the VM, and is controlled by the information owner. TVEM provides the information owner the ability to attest the virtual environment for the desired integrity and confidentiality properties.

Dewan et al. [16], present a technique to guard critical data of a client application running in a virtualized environment where a VM may include malware. The proposed approach uses a lightweight hypervisor for fine grained software-based run time memory protection. It places application critical data in protected memory regions and then registers it with the client application. Access to the protected memory region is

controlled by the VMM on the basis of authenticity of an application. The approach also suggests a data locker component for the VMM to prevent leakage of the application persistent storage to malwares and rootkits.

Specialized encryption based approaches

Lei et al. [5], proposed homomorphic encryption schemes [2] for performing sensitive computation on encrypted data inside an untrusted cloud platform that provides input/output confidentiality and resulting integrity. The authors have shown how common engineering and scientific computational tasks such as matrix inversion computation (MIC) can be securely outsourced to an untrusted cloud platform. The original matrix is encrypted before being outsourced to the cloud and then processed in encrypted form. Transformation is applied on the result received from the cloud to get accurate inversion of the original matrix. Homomorphic encryption, however limits the types of operations that can be performed. Moreover this approach overburdens the client and is not desirable for thin client environments, such as mobile devices, with limited resources.

Vimercati et al. [8], consider a cloud scenario where storage and computation services are used from different cloud providers. Data resides in encrypted form on a storage provider and is transferred to a computation provider for processing in encrypted form. The outcome of the result is then again encrypted and returned to the client, which then transforms the result into clear text and verifies its integrity. The main shortcoming of this approach is that if limited computation needs to be performed on the data the client will still need to outsource storage and computation from multiple providers. Also it needs to bear the reservation and communication overhead involved in moving the data to the computation service provider.

A protocol for providing confidentiality and controlling access to the data in the cloud is proposed by Tysowski et al. [7]. The given approach recommends modifications to attribute-based encryption to control access to data based on possession of certain attributes by authorized users. The data owner determines attributes required for data access to protect data confidentiality from unauthorized users. The data owner then generates, encrypts and then uploads data to the cloud to be accessed only by authorized users.

Ports et al. [41], attempt to protect application code and its data from the compromise of untrusted operating system in a virtualized environment by providing encrypted view of the memory pages of a client application to the operating system. The proposed approach allows a protected application running in a virtual machine (VM) to interact directly with the VMM through a user level code, called *shim*. The direct communication between the *shim* and VMM allows the application to protect its resources, such as files in memory. However, the proposed approach does not consider a cloud environment where the attacker is potentially more powerful and may have control of the platform and communication channel in addition to the operating system.

Unlike existing approaches our proposed protocol considers user computation integrity and data confidentiality against a powerful attacker such as an untrusted cloud system administrator. The application code is executed with hardware protection, in complete isolation from a potentially compromised VMM. Cryptographic operations and computation is performed at the provider side, which allows support for thin clients, such as mobile devices.

8 CONCLUSION AND FUTURE WORK

In the last few years, cloud computing has experienced very high growth rates and is showing great prospects. One of the biggest challenges to the wide adoption of cloud based services is client confidentiality and integrity concerns. In this paper, we have presented and formally verified a practical solution to address this problem. Our solution includes a protocol for secure VM launch which enables clients to verify cloud platform configuration before launching their VMs on the cloud. In addition, a protocol for performing sensitive computations in a cloud environment is presented. We have formally verified the security properties of our proposed protocols using ProVerif. Currently our implementation is for Intel based systems but it can easily be adapted to AMD. Evaluation results show that our solution is practical in terms of performance. In the future, we are planning to perform rigorous penetration testing of our protocol using an actual deployment.

ACKNOWLEDGMENTS

We wish to thank Chris Dalton from HP lab Bristol for his valuable advice and many enjoyable discussions.

REFERENCES

[1] N. Kroes. *"Setting up the European Cloud Partnership"*. World Economic Forum, 2012.

[2] M. Naehrig, K. Lauter, and V. Vaikuntanathan. *"Can homomorphic encryption be practical?"*. In Proceedings of the 3rd ACM workshop on Cloud computing security workshop, pp. 113-124, New York, USA, 2011.

[3] Advanced Micro Devices. AMD64 virtualization: Secure virtual machine architecture reference manual. AMD Publication no. 33047 rev. 3.01, May 2005.

[4] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. OHanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen. *"Principles of remote attestation"*. International Journal of Information Security, Volume 10, Issue 2, pp. 63-81, 2011.

[5] X. Lei, X. Liao, T. Huang, H. Li and C. Hu. *"Outsourcing Large Matrix Inversion Computation to A Public Cloud"*. IEEE Transactions on Cloud Computing, Volume 1, Issue 2, pp. 78-89, 2013.

[6] Trusted-Java: Jsr321: Trusted computing api for java(tm) (2009) Available at: http://jcp.org/en/jsr/detail?id=321. Accessed on 06/09/2013.

[7] P. Tysowski and M. Hasan. *"Hybrid Attribute- and Re-Encryption-Based Key Management for Secure and Scalable Mobile Applications in Clouds"*. IEEE Transactions on Cloud Computing, Volume 1, Issue 2, pp. 172-186, 2013.

[8] S. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati.*"Integrity for Join Queries in the Cloud"*. IEEE Transactions on Cloud Computing, Volume 1, Issue 2, pp. 187-200, 2013.

[9] X. Leroy. *"Formal certification of a compiler back-end, or: programming a compiler with a proof assistant"*. In 33RD Proceedings of ACM Symposium on Principles of Programming Languages, 2006.

[10] F. Krautheim, D. Phatak, and A. Sherman. *"Introducing the trusted virtual environment module: a new mechanism for rooting trust in cloud computing"*. In Proceedings of the 3rd international conference on Trust and trustworthy computing, 2010.

[11] A. Baldwin, C. Dalton, S. Shiu, K. Kostienko, and Q. Rajpoot. *"Providing secure services for a virtual infrastructure"*. Operating Systems Review-ACM Special Interest Group on Operating Systems, Volume 43, Issue 1, PP. 44-51, 2009.

[12] P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield. *"Breaking up is hard to do: security and functionality in a commodity hypervisor"*. In Proceedings of the Symposium on Operating Systems Principles, PP. 189-202, 2011.

[13] Intel Corporation. *"Intel Virtualization Technology for Directed I/O"*. Intel Publication no. D51397-004 rev. 1.2, 2008.

[14] J. McCune, B. Parno, A. Perrig, M. Reiter, and A. Seshadri. *"How low can you go? Recommendations for hardware-supported minimal TCB code execution"*. In Proceedings of the ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2008.

[15] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. *"The Eucalyptus Open source Cloud computing System"*. In Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, China, 2009.

[16] P. Dewan, D. Durham, H. Khosravi, M. Long, and G. Nagabhushan. *"A Hypervisor-Based System for Protecting Software Runtime Memory and Persistent Storage"*. In Proceedings of the Spring simulation Multi-Conference, SpringSim, Boston, USA, 2008.

[17] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. *"Terra: A Virtual Machine-Based Platform for Trusted Computing"*. In Proceedings of the ACM Symposium on Operating Systems Principles, 2003.

[18] N. Santos, K. Gummadi, and R. Rodrigues. *"Towards Trusted Cloud Computing"*. In Proceedings of the Hot Topics in Cloud Computing, USA, 2009.

[19] D. Murray, G. Milos, and S. Hand. *"Improving Xen security through disaggregation"*. In Proceedings of the International Conference on Virtual Execution Environments, New York, USA, 2008.

[20] TCG Specification Architecture Overview. https://www.trustedcomputinggroup.org. Accessed on 05/09/2013.

[21] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. *"Design and Implementation of a TCG-based Integrity Measurement Architecture"*. In Proceedings of the 13th USENIX Security Symposium, San Diego, USA, 2004.

[22] I. Khan, H. Rehman, and Z. Anwar. *" Design and Deployment of a Trusted Eucalyptus Cloud"*. In Proceedings of the 4th IEEE International Conference on Cloud Computing, Washington, USA, 2011.

[23] Research: 2012 State of Cloud Computing. http://reports.informationweek.com/abstract/5/8658/cloud-computing/research-2012-state-of-cloud-computing.html. Accessed on 04/06/2013.

[24] Cloud computing to the rescue! Stolen iPhone tracked via iCloud(2012). http://www.gmanetwork.com/news/story/259410/scitech/technology/cloud-computing-to-the-rescue-stolen-iphone-tracked-via-icloud. Accessed on 08/09/2013.

[25] The OpenStack Community *"OpenStack Cloud Software"*. http://www.openstack.org/, 2011. Accessed on 11/09/2013.

[26] The OpenNebula Project *"OpenNebula: The Open Source Toolkit for Cloud Computing"*. http://opennebula.org/, 2011. Accessed on 15/08/2013.

[27] Nimbus Home Page. http://www.nimbusproject.org/. Accessed on 15/08/2013.

[28] P. Sempolinski, and D. Thain. *"A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus"*. In Proceedings of the International Conference on Cloud Computing Technology and Science, Indianapolis, USA, 2010.

[29] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki. *"Flicker: An execution infrastructure for TCB minimization"*. In Proceedings of the ACM European Conference in Computer Systems, 2008.

[30] Intel. Intel Trusted Execution Technology Measured Launched Environment Developers Guide. http://download.intel.com/technology/security/downloads/315168.pdf, 2008.

[31] F. Zhang, J. Chen, H. Chen, and B. Zang. *"CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization"*. In Proceedings of the ACM Symposium on Operating Systems Principles, 2011.

[32] S. Butt, H. Cavil, A. Srivastav, and V. Ganapathy. *"Self-service Cloud Computing"*. In Proceedings of the ACM Conference on Computer and Communications Security, 2012.

[33] R. Needham and M. Schroeder. *"Using encryption for authentication in large networks of computers"*. Communications of the ACM, Volume 21, Issue 12, 1978.

[34] G. Lowe. *"Breaking and Fixing the Needham-Schroeder Public-key Protocol using CSP and FDR"*. In Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, pages 147-66. Springer-Verlag, 1996.

[35] A. Armando, R. Carbone, L. Compagna, J. Cuellar and L Tobarra. *"Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps"*. In Proceedings of the 6th ACM workshop on Formal methods in security engineering, Pages 1-10, New York, USA. 2008.

[36] OpenSSL vulnerabilities. Available at: https://www.openssl.org/news/vulnerabilities.html. Accessed on 12/12/2014.

[37] M. lbrecht, K. aterson and G. Watson. *"Plaintext Recovery Attacks against SSH"*. In Proceedings of the 30th IEEE Symposium on Security and Privacy, pages 16-26, 2009.

[38] M. Arapinis, T. Chothia, E. Ritter and M. Ryan. *"Analyzing Unlinkability and Anonymity Using the Applied Pi Calculus"*. In the 23rd IEEE Computer Security Foundations Symposium (CSF), pages 107-121, IEEE Computer Society, 2010.

[39] *"ProVerif: Cryptographic protocol verifier in the formal model"*. Available at: http://prosecco.gforge.inria.fr/personal/bblanche/proverif/. Accessed on 10/10/2014.

[40] A. Armando, R. Carbone and L. Compagna. *"Satmc: A SAT-based model checker for security critical systems"*. In the 20Th International Conference on Tools and Algorithms For The Construction and Analysis Of Systems, France, 2014.

[41] D. Ports and T. Garfinkel. *"Towards application security on untrusted operating systems"*. In Proceedings of the 3rd conference on Hot topics in security, Berkeley, USA, 2008.

[42] V. Potdar, S. Han and E. Chang. *"A survey of digital image watermarking techniques"*. in the Proceedings of the IEEE 3rd International Conference on Industrial Informatics (INDIN), Perth, Australia, 2005.

[43] C. Hota, S. Sanka, M. Rajarajan and S. Nair, *"Capability based Cryptographic Data Access Control in Cloud Computing"*. in International Journal of Advanced Networking and Applications, Volume 01, Issue 01, 2011.

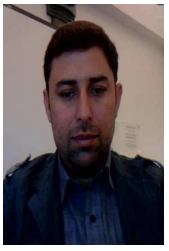
[44] K. Seny and L. Kristin, *"Cryptographic Cloud Storage"*. In the Proceedings of the 14th International Conference on Financial Cryptography and Data Security, Tenerife, Spain, 2010.

APPENDIX

Table 2, contains ProVerif code used for the verification of security properties of our proposed protocol.

```
1  set verboseClauses = explained.
2
3  free DK_VM:bitstring [private].
4  free getQuote:bitstring.
5  free c, d: channel.
6  free e, f, g: channel [private].
7  free success: bitstring.
8
9  (* encryption and decryption functions *)
10  fun enc (bitstring, bitstring, bitstring): bitstring.
11  fun dec (bitstring, bitstring): bitstring.
12  fun senc (bitstring, bitstring, bitstring): bitstring.
13  fun sdec (bitstring, bitstring): bitstring.
14  fun pub (bitstring): bitstring.
15
16  (* decryption followed by encryption produces original
message *)
17  equation forall xk: bitstring, xr:bitstring, xm:
bitstring; sdec(xk,
18  senc(xk, xr, xm)) = xm.
19
20  equation forall xk: bitstring, xr:bitstring, xm:
bitstring; dec(xk,
21  senc(pub(xk), xr, xm)) = xm.
22
23  query attacker (DK_VM).
24
25
26
27
28
29
30
31
32  (* the actions of the client *)
33  let client(Pk_c:bitstring) =
34      in (e, pubF:bitstring);
35      new na:bitstring;
36      new r1:bitstring;
37      out (c, enc (pubF, r1, (na, pub(Pk_c))));
38      in (c, x:bitstring);
39      let (=na,  nF:bitstring) = dec (Pk_c, x) in
40      new r2:bitstring;
41      out (c, enc (pubF, r2, (nF, DK_VM))).
42
43
44  (* the action of the flicker session *)
45  let flicker =
46      new Pk_f: bitstring;
47      out (e, pub (Pk_f));
48      out (c, pub (Pk_f));
49      in (c, x:bitstring);
50      let (xna:bitstring, xpubC:bitstring) = dec (Pk_f, x) in
51      new nF:bitstring;
52      new r:bitstring;
53      out (c, enc (xpubC, r, (xna, nF)));
54      in (c, y:bitstring);
55      let (=nF, DK_VM:bitstring) = dec (Pk_f, y) in
56      out (c, success).
57
58  (* the specification of the whole system: key generation,
afterwards running client and flicker in parallel *)
59  process
60      ! ((new Pk_c:bitstring; out(c, pub(Pk_c)); (client
(Pk_c))) | flicker)
```

TABLE 2
ProVerif security verification code



I mran Khan received his M.S. degrees in Computer Sciences in 2009 from the National University of Computer and Emerging Sciences, Islamabad, Pakistan. Imran has worked as Software Developer and researcher at University of Birmingham, UK, Security Engineering Research Group (SERG) IMSciences, Pakistan, on system and web security related projects. Imran has several distinction including Higher education Commission of Pakistan (HEC) MS leading to PhD Fellowship and International Research Support Initiative Program (IRSIP) Fellowship to UK. Currently he is pursuing his PhD from FAST-NUCES Islamabad.

Z ahid Anwar received his Ph.D. and M.S. degrees in Computer Sciences in 2008 and 2005 respectively from the University of Illinois at Urbana-Champaign (UIUC), USA. Zahid has worked as a software engineer and researcher at IBM, New York, USA, Intel, Oregon, USA, Motorola, Schaumburg, Illinois, USA and the National Center for Supercomputing Applications (NCSA), Urbana, Illinois, USA on various projects related to information security and operating system design. Currently he is an Assistant Professor at the School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad, Pakistan and an Associate member of the Graduate Faculty in the Software and Information Systems Department at the University of North Carolina at Charlotte, USA.

B ehzad Bordbar has his BSc, MSc and Ph.D in Mathematics (PhD from Sheffield, UK). Following his PhD, he worked as a researcher on a number of projects at University of Ghent, Belgium and University of Kent, UK. He is currently affiliated to the School of Computer Science, University of Birmingham, UK, where he teaches courses in Software Engineering and Distributed Systems. In recent years, he has had close collaborative research with various academic and industrial organizations, among them Ghent University, Osaka University, Colorado State University, BT, IBM and HP research laboratories. His research activities are mostly aimed at using modelling to produce more dependable software and systems in shorter development cycles and at a lower cost. His current research projects are dealing with Formal methods, Model Analysis, Software Tools, Model Driven Development and Fault-tolerance in Service Oriented Architectures and Cloud.

E ike Ritter obtained his masters' degree from Erlangen, Germany and his PhD from Cambridge. Following his PhD, he worked as a researcher in Cambridge and Oxford before joining Birmingham University to which he is currently affiliated to. He has worked on categorical logic and type theory. His currently research interests are in security, in particular in design and verification of protocols, security for mobile phones and hardware-based systems.

H abib-ur Rehman has completed his doctoral studies (Dr.-Ing.) in 2009 at the Technische Universitaet Carolo Wilhelmina zu Braunschweig, Germany. He is currently Assistant Professor in the Department of Computer Science at Al Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, KSA. His primary research interests are the design and development of network protocols, schemes and models with a focus on the issues of Routing, MAC, streaming, security, information sharing and cloud computing.