Convergence in trusted computing and virtualized systems: A new dimension towards trusted intelligent system

Thinh Le Vinh, Samia Bouzefrane, Soumya Banerjee

Convergence in Trusted Computing and Virtualized Systems: A new Dimension towards Trusted Intelligent System

Thinh LE VINH, Samia Bouzefrane
Conservatoire National des Arts et Métiers
Paris, France
firstname.lastname@lecnam.net

Soumya Banerjee
Birla Institute of Technology
Mesra, India
soumyabanerjee@bitmesra.ac.in

***Abstract*—The advantages of virtualization are tremendous in the current computing ecosystem. The virtualization technology is an abstract mechanism that enables a single hardware platform to run multiple environments. This technology is presented in many forms such as process, storage, and network virtualization. In this paper, we introduce the different techniques of virtualization such as those implemented by hypervisors or by isolated containers, in order to investigate the convergence between trusted computing and virtualized systems. By introducing Trusted Platform Module (TPM) in the virtualized environment, we describe the way VMs are attested to be used as a building block for any trusted intelligent system.**

Keywords—Trusted platform module; virtualization; intelligent system.

I. Introduction

Since the first introduction in the mid of 1960s by IBM and the increasing of cloud technology, virtualization has involved extremely to the paradigm of computing technology. In fact, the virtualization concept evolved from software-based technique to, more recently, hardware-based solution by virtualizing memory, processor and devices more efficiently. In addition, with the appearance of the new virtualization technology such as isolation container, applications may be run in isolated environments with a minimum overhead unlike the traditional virtualization solutions. Due to the open and the interoperation of current computing systems, the trusted computing has been widely designed to secure the integrity of the systems from software attacks and a few hardware attacks. Trusted computing, which is based on hardware component such as Trusted Platform Module (TPM), can be also recognized as a set of technologies that provide a primitive security mechanism to protect computing infrastructure. In general, TPM [1, 8] is a hardware platform dedicated to PCs, servers, personal digital assistant (PDA), printer, or mobile phone to enhance security in an ordinary, non-secure computing platform and convert them into a trust environment. Each platform contains only one physical TPM to be a trusted platform. TPM implements mechanisms and protocols to ensure that a platform has loaded its software properly. This has been named as remote attestation. TPM is independent from the host operating system (OS); it stores secret keys to encrypt data files/messages, to sign data, etc. By taking advantages of virtualization technology benefits, physical TPM delivers its functionality to the multi guest operating system in the abstract way. As a result, TPM virtualization provides security architecture in the virtualizable platforms. In this paper, our aim is firstly to highlight the different virtualization techniques used by the virtual machine manager (VMM) and the alternative solutions that may offer better performances, and secondly to discuss how to secure guest operating systems using TPMs. Finally, we propose a road map towards Trusted Intelligent System (TIS) which may assist as a better kernel of trust while deploying various computationally intelligent components. For this reason, the rest of this paper is constructed by the following sections. We first present the general principles of virtualization in Section 2. Following by Section 3 which shows that isolated containers can be a good candidate to virtualize applications with good performance. We then highlight the use of TPMs in virtual machines in Section 4. Last, but not least, we compare the mechanism behind the containers and the virtual machines in Section 5. This is followed by the proposed road-map for a Trusted Intelligent system (TIS). Finally, we sum up the article in Section 6.

II. Virtualization

Nowadays, virtualization technology is widely used to share the capabilities of physical computers by splitting the resources among operating systems. In 1964, the concept of virtual machines (VMs) is used in the IBM's project called CP/CMS system. The Control Program (CP) acted to split the physical machine to several copies (VMs), each copy with her own OS CMS. In 1974, Goldberg and Popek published a paper [2] which introduces a set of sufficient conditions for computer architecture to efficiently support system virtualization. In 1999, the company VMware presented its product VMware Virtual Platform for the x86-32 architecture [3]. In 2006, Intel and AMD proposed extensions to support virtualization. In the next subsections, we will recall the basic principles of virtualization.

A. *Virtualization techniques*

The virtualization layer called hypervisors or Virtual Machine Monitor (VMM) manages the hardware resources between the different VMs. The guest OS is the OS that is running within a VM. Currently, there are several virtualization techniques that we summarize in the followings.

1) Full virtualization: In full virtualization, the source code of the guest OS is not modified. So, when executed, the guest OS is not aware to be virtualized. Binary translation (BT) is used to emulate a small set of instructions, i.e., privileged instructions are trapped and sensitive but non-privileged instructions are detected then translated to show a fake value. The rest of the instructions are directly executed by the host CPU. There are two types of hypervisors that handle the full virtualization. Type 1 corresponds to a native or standalone hypervisor that runs directly on the bare hardware. Type 2 is hypervisor that runs on top of the host OS (like an ordinary application). The drawback of full virtualization is that the VMM must use special tricks to virtualize the hardware for each VM, which generates an additional overhead when accessing the hardware.

2) Para virtualization: It refers to the collaboration between the guest OS and the hypervisor to improve the performance. This collaboration involves modifying the source code of guest OS to call directly, using hypercalls, the hypervisor to execute privileged instructions. So the guest OS is aware to be virtualized. The drawback of this technique is the poor compatibility and portability of OSs.

3) Hardware-assisted virtualization: To simplify the virtualization techniques, hardware vendors such as Intel and AMD introduced new extensions to support virtualization. Hence, two mode features for CPU execution are introduced in Intel Virtualization Technology (VT-x) and AMD's AMD-V so that the VMM runs in a root mode below ring 0 (i.e., ring -1) while the guest OS runs in non-root mode (ring 0). The state of the guest OS is stored in Virtual Machine Control Structures (for Intel VT-x) or Virtual Machine Control Blocks (for AMDV). The advantage of the hardware-assisted virtualization is the reduction of the overhead caused by the trap-and-emulate model [4, 5].

B. *Virtualization of hardware*

In order to run successfully the virtualized systems, the virtualization layer (VMM) needs to share CPU, memory and devices. Several mechanisms are implemented to do so.

1) Processor virtualization: Generally, the OS is designed to run directly on the physical machine and fully has use of the computer resources. The traditional x86 architecture has four levels of privileges (called rings 0 to 3) dedicated to the operating system and the applications in order to manage access to the computer hardware. User applications run in Ring 3 (user mode), while the operating system runs in Ring 0 (kernel mode) to have all privileges. In the new CPU generations, an extension for virtualization called Virtual Machine Extension (VMX) has been added. In the Intel VT-x architecture, there are two execution levels: VMX root mode used to execute the VMM functions; and VMX non-root mode which corresponds to a reduced privilege to run guest OSs.

2) Memory: In traditional operating systems, a physical memory is allocated for each process defined with its own virtual address space. When the process wants to access memory, the process virtual address is translated by memory management unit (MMU) that relies on the process page table. When a guest OS runs on a hypervisor, an additional translation level is added. In fact, the guest OS maps guest virtual addresses (GVA) with guest physical addresses (GPA), and the translation from GPA to machine physical addresses (MPA) is done by the hypervisor [7].

3) Devices: The VMM has to present devices to the guest OS such as disk, network interface, USB and timers. Since the drivers of the guest OS are not aware to be virtualized, all the instructions generated by the driver shave to be trapped and emulated, which is the source of overhead. However, if the drivers are aware, a more direct path to handle devices inside the VM can be used by installing particular drivers which talk to the para-virtual devices (like virtio interface in Linux) [6].

III. Isolation by Containers

A. *Background*

In the virtualization based solutions, the hypervisor scheduling operates at two levels as explained in the following. The first scheduling occurs at the hypervisor level where the hypervisor schedules VMs, and the second one at the virtual machine level where the guest operating system schedules processes. Taking that into account, it is obvious that this approach creates a significant overhead. Another virtualization approach, based on containers, has been proposed to reduce significantly this overhead, and thus providing better performances, especially in terms of elasticity and density within a lean data center.

In the container based virtualization, scheduling occurs only at one level. The containers contain processes, while the kernel and the libraries are shared between the containers. Hence, the scheduler is used only at process level to schedule processes. The behavior is similar to a classical operating system; the only difference is that, in container-based virtualization, processes are affected to a container, and between each container, an isolation is provided by the operating system. In this section, we will try to cover the technologies behind the container-based virtualization that we call containerization. Many solutions like Lxc, OpenVZ and Docker are based on containers. However, in this article, we will not focus on a specific solution, but instead we will try to explain the principles as provided by the Linux kernel and used by all of these solutions.

B. *Containers*

We can define a container as a confined environment under the global environment. The Linux operating system provides some mechanisms to create these confined environment or container, and somehow forces each container to follow a predefined policy. The isolation can occur at two levels: Isolation between processes, and Isolation between processes and the hardware. For the first type of isolation, Linux OS uses the name spaces mechanism and for the second one it relies on the cgroup mechanism.

1) Namespaces: The namespaces mechanism is a powerful Linux solution for isolating processes. Instead of isolating processes only at the filesystem level like with chroot, the namespace mechanism isolates processes at different levels and creates a really confined environment, thus making processes wrongly think that they are alone on the system. Currently, Linux OS implements different types of namespaces[1]. Each namespace is responsible for isolating the process according to a certain feature.

- *Mount namespace:* The mount namespace creates isolation at the filesystem level.
- *PID namespace:* The PID namespace isolates the process identifier space. In fact, every process p will have a new process identifier within the new PID namespace, in addition to the process identifier that p has in the parent PID namespace.
- *IPC namespace:* This namespace isolates inter-process communication mechanisms like semaphore, shared memories, etc.
- *Network namespace:* The network namespace provides isolation at a network level, and each namespace has its own network devices, IP address, IP routing table, etc.
- *UTS namespace:* This functionality allows each UTS namespace[2] to have its own hostname and NIS domain name.

2) Control Groups: While the namespaces mechanism allows creating an isolation between processes, Cgroup (abbreviated from control groups) creates an isolation between a group of processes and the hardware. Cgroup[3] is a Linux kernel feature proposed to limit, account, and isolate resource usage (CPU, memory, disk I/O, etc.) for a group of processes. In cgroup's terminology, there are two commonly used terms; the term subsystems which refers to the hardware (CPU, memory, etc.), and cgroup hierarchies which are entities with predefined parameters for accessing the hardware. A cgroup hierarchy can be divided into different sub-hierarchies and for each hierarchy or sub-hierarchy, a list of processes is associated to it. Each hierarchy contains a set of parameters which are dependent on the type of the attached subsystems, and the system will enforce, for the attached processes, the whole policy defined in these parameters. A subsystem can be attached to at most one hierarchy; however, many subsystems can be attached to the same hierarchy.

[1] http://lwn.net/Articles/531114/

[2] UNIX Time-sharing System

[3] http://en.wikipedia.org/wiki/Cgroups

The combination of the namespaces and cgroup Linux functionalities allow the design of containers which are isolated from both the process and the hardware perspectives. Even if these functionalities are developed by different teams, the two Linux solutions are orthogonal, and most of the container-based virtualization solutions like Lxc and Docker rely on them. After we described the principles of the existing virtualization-based solutions, we highlight in the next section the role of a trusted platform initially used to attest a host operating-system platform in order to see if a trusted platform can also be used to protect virtual entities like VMs and containers.

IV. Virtualization of Trusted Platforms

In the context of trusted platforms, trusted platform module (TPM) is a secure chip attached mainboard on the physical platform (such as a PC, a printer, etc.) to keep the integrity of the system. Its specification is created and promoted by the Trusted Computing Group (TCG). According to TCG, TPM is equipped with volatile and non-volatile memory for storing computational variables and secret data. By providing platform monitoring, secure storage, encryption operations and authentication service [8], the TPM is also known as the primitive security for the computing platform. For integrity measurements, TPM uses a special registration that is the PCR (Platform Configuration Register) to store the most important volatile data. The non-volatile memory deals with storing TPM secrets such as security keys. The TPM has two distinguished keys: the Endorsement Key (EK) and the Storage Root Key (SRK). The EK is generated by the manufacturer and unique to the TPM chip. This key is used for TPM identity and is never revealed externally. Related to the EK are Attestation Identity Keys (AIKs). The AIK is a 2048 bit RSA key for signing data that is generated locally by the TPM but may be leaked outside. The latter, Store Root Key, is the first key to be created by the TPM in the initializing process. The SRK is a non-migratable key located as the root of TPM's key hierarchy for cryptographic protection of keys held outside the TPM. With the advantage of virtualization technology as discussed earlier, the convergence between trusted computing and virtualized computing enables a novel security method. In this context, because the physical TPM that is linked to the underlying physical machine cannot attest directly guest OSs, it has been virtualized to provide its functionalities for each VM, which is running on a single platform. This makes VM feel it has its own private TPM. As a result, by virtualizing, a single physical TPM works as the Root of Trust which can be served by multi virtual environments on the trusted platform. For this reason, there are different implementation models for TPM virtualization [9, 10, 11, 12]. As we will see it in the next subsections, normally the TPM virtualization can be implemented by migrating the real TPM to the virtual machines or emulating the TPM in software [12]. Table 1 presents the difference between the physical TPM (noted pTPM) and virtual TPM (noted vTPM).

A. TPM virtualization via Virtual Machine Monitors

This implementation is proposed by many authors for the migration of virtual TPMs (vTPMs) and their associated VMs to provide secure storage and cryptographic operations. Berger et al. [9] introduced an architecture where the TPM's specifications are available in virtual environment. Sadeghi et al. [13] proposed a vTPM architecture, which is based on [9], to improve the maintainability and applicability of VMM. England and Loeser, in [10], proposed a para-virtualized TPM sharing the approach in which a physical TPM can be shared among virtualized hosts. To enhance the security, Danev et al. [14] delivered a new vTPM key hierarchy by proposing an intermediate layer of keys between pTPM and vTPM. In what follows, the article introduces a brief background on TPM virtualization in terms of architecture and migration.

TABLE I. THE DIFFERENCE BETWEEN PTPM AND VTPM

Criterion	Physical TPM (pTPM)	Virtual TPM (vTPM)
Design	Hardware (TPM chip)	Software
Resources	Endorsement Key (EK), Storage Root Key (SRK), Attestation Identity Key (AIK), and Platform Configuration Registers (PCRs)	vEK, vSRK,vAIK, and vPCRs
Specifications	Standardized by Trusted Computing Group (TCG)	Imitate the functionality of the pTPM
Security	Trust anchor, high security level	Low security level in comparison with pTPM
Operation platform	One to one	Multi to one

1) vTPM architecture: Berger et al. [9] designed a vTPM architecture that enables physical TPM run on systems running parallel multiple operating systems. In this architecture, the vTPM manager is responsible for multiplexing requests and creating a guest vTPM instance corresponding to its guest VM with configured TPM support. Each vTPM instance imitates the interface and functionality of the hardware TPM. Figure 1 demonstrates the generic architecture of vTPM.

2) vTPM migration: vTPM migration protocol is one of the most important features on TPM virtualization based VMM. To keep the correct operation of guest applications, the vTPM must be migrated to the corresponding VM. A secure migration relies on the TPM key migration facilities which is supported by the existing TPM standard. It also requires synchronizing VM-vTPM state during the migration. In [9], the authors presents vTPM migration procedure by using asymmetric and symmetric key. In the protocol, the state of vTPM is encrypted and packaged on the source platform and decrypted on the destination platform. Instead of using a migratable TPM storage key, the authors in [13] propose to use the migration procedure in a trusted channel, which is used to create a secret encryption key related to the TPM on the destination and to the configuration of trusted computing base.

B. TPM based Software Emulator

As mentioned above, one limitation associated with the security properties of the physical TPM is that it only permits one TPM per platform. Since Berge et al. [9] introduced their vTPM architecture, their works have become a foundation solution for other authors. However, there are no open source implementations. The TPM based software emulator is another solution to overcome the limitation of TPM. In virtualization environment, each vTPM is implemented as software based emulator that has its own emulated specifications. Because of software form, the solution has less security guarantees in comparison with hardware form. It neither keeps the hardware based root of trust nor supports vTPM migration. Although remaining some drawbacks, this solution is currently reasonable for study purposes. We present here two examples of emulators.

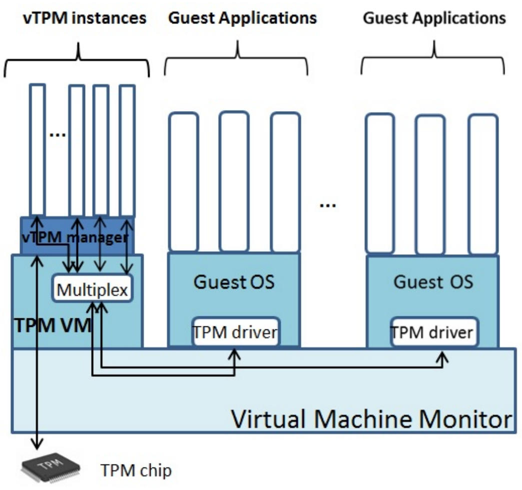

Fig. 1. vTPM architecture

1) TPM emulator: Strasser and Stamer [12] presented an open source TPM emulator. It becomes a new standard for other approaches using software TPM. The solution provides the ability to run more than one TPM emulator instance on a single platform and execute TPM based software in a trusted virtualization environment.

2) XEN- embedded TPM emulator: XEN is an open source hypervisor that enables for multiple VMs to run on a single system. It provides a service to support TPM functionality to VMs by embedding a TPM emulator in it. This allows guest application to interact with a vTPM in a same way they interact with a hardware TPM [17]. Each VM has its own software TPM, which is simulated by TPM simulator. vTPM manager domain manages TPM specifications and is responsible for interacting between vTPM and pTPM. The implementation model of vTPM in XEN has the following features [16]: non transparent vTPM, vTPM's secret bound to pTPM, configurable TPM ownership and Storage Root Key authentication, pass through of certain Platform Configuration Register (PCR), extension of Chain of Trust from the host machine to the virtual. However migration of vTPMis not supported.

C. Recent Challenges : Trust and Virtualization in the Cloud

A substantial survey and an analysis are being accomplished in trusted computing and virtualization in recent years [19]. Erickson research group in Lund university in 2016 [20] elaborated a discussion and presentation concerning trusted computing and trusted platforms. They discussed about the hypervisor and the microkernel component in pure virtualization with respect to the operating system. The protection scheme of guest virtual machine is accomplished based upon the securely protecting the data-centre by using hardware processor of cloud provider [21]. A review on Infrastructure as a Service (IaaS) and its underlying virtualization technologies has been presented with special mention of hypervisor [22]. The interplay of the hypervisor and the kernel has been demonstrated in virtualized network outset. They designed blueprint—a set of abstractions, general principles, and low-level implementation details—for efficient deterministic replay in a modern hypervisor. They evaluated the proposed architecture in Xen, a full-featured hypervisor [23]. Trusted computing in cloud is also supported by layered attestation services. Bundling Evidence for Attestation is presented with trusted computing scenario with formal proofs [24]. An elaborated and novel framework is presented in [25] for registration, for verification and for detection of virtual machines. A TPM-based node registration process initially establishes trust. Attested heartbeat procedure periodically verifies the trustworthiness of every node in the system. Tampered or miss-configured nodes can be identified quickly by reputation based.

V. DISCUSSION

The addressed issue here is to figure out the virtualization mechanisms in order to answer to the following question: is-it possible to define a trust environment based on trusted platforms such as TPM for the virtualized entities like VMs and containers in the same way as we can do it for hosted OSs of physical machines? The preceding section showed that to use TPM for VMs, the existing solutions either propose a software to emulate a TPM for each VM or introduce a mapping of the physical TPM with virtual ones so that each vTPM is linked to a VM while relying on the pTPM to have a real trust root. In this section, we aim to investigate the possible use of TPM for containers. For this purpose, we first compare the features of the VMs and those of the containers, and then propose a road map for an intelligent system based on trusted platforms.

A. Virtualization based on Containers versus Virtual Machines

Unlike VMs that rely on a hypervisor, containers are viewed as applications within the host operation system. Consequently, as stated in [17], the performances given by using containers are better than those for virtual machines, which is a little obvious since containers add almost no overhead comparing to VMs. We might think that containers are the best solution to choose when we are dealing with the virtualization, however this is not always true. Indeed, in the case where we have to run completely different operating systems on the same machine, the container's solution is not suitable anymore, since theses latter do not share the same kernel. In addition to this, while there is many mature solution for the live VM migration [18], it is not the case for containers, thus if migration is a concern, VM based virtualization remains more suitable. The information in table 2 pinpoints the issue that the Trusted Computing and Virtualized System can converge in case of virtual machine.

TABLE II. DIFFERENCE BETWEEN CONTAINERS AND VMS

	Containers	VMs
Require	A host OS	A hypervisor of type 1 type 2
Size of images	Images are lightweight (for example, basic Ubuntu image for Docker containers has 180MB)	A VM hosts a guest OS that is a very big file (for example Ubuntu 14.04.1 has more than 1GB)
Platform dependency	Dependent-platform: the container is viewed as a bunch of processes vis-à-vis the host OS	Independent-platform: a guest OS is hosted by a VM that is managed by a hypervisor. A guest OS is not aware of that and behaves as an OS that has total access to the physical resources.
Migration	Migration is possible only if the source and the destination platforms are Linux based. Even if Docker container engine is provided for Windows platform for example, this engine uses Linux-specific kernel features and requires a VM4	Migration is suitable for VMs viewed as files easily migratable and manageable by a hypervisor.
pTPM	Not available (N/A) on the container since it is viewed as a bunch of processes within the host OS	Operate on host OS and Hypervisor type 1 (TPM-VM)
vTPM	N/A	The vTPM is associated with the corresponding VM
vTPM migration	N/A	Hypervisor type 1 (TPM-VM)

B. Trusted Intelligent System (TIS): Proposed Road map

This paper initiates a novel paradigm of intelligent yet trusted system for cloud and virtualized network based on VMs. We investigate recent works in the form of fuzzy logic and heuristic based virtual machine consolidation approach to achieve energy-QoS balance [26, 27]. Generically, fuzzy technique is an attractive approach to handle uncertain, imprecise, or un-modeled data in solving control and intelligent decision-making problems. Different VM selection methods offer different advantages and thus it could be contemplated with fuzzy and other heuristic interpretation. Considering the research references, it is realized that machine and reinforcement learning can deploy a trusted intelligent system for virtualization of network. TIS could be formally positioned with statistical parameters of conventional trusted platform and finally it could be functional as autonomic system. However, the other trends of evolutionary

4 https://docs.docker.com/installation/windows/

computational intelligence may not be suitable intelligent solution. Soft computing e.g. fuzzy logic and rough set association rules can be used in cloud network security measures. While studying decision systems, all researches confront the question of dropping some (superfluous) condition attributes without altering the basic properties of the system [28]. Rough Set Theory provides a sound mechanism of carrying out this task. The process is referred as attribute reduction or feature selection [27]. The reduction thus obtained retains the minimal set of attributes that preserves the information of interest.

Decision rules are then deduced based on these reductions. The intelligence induced by rough set in virtualized platform may indicate several criteria:

- Discretization of numerical attributes of the virtualized network.
- Completion of missing values in decision tables using different trusted computing or belief algorithms.
- Partition of data into training and testing groups using random number generators.
- Efficient computation of reduces.
- Provides support for supervised and unsupervised learning.
- Generation of IF-THEN rules using reduction.
- Execution of script files.
- Support for cross validation.
- Post processing
- Advanced filtering of sets of reductions and rules.
- Validation and analysis.
- Application of induced rules on testing data.
- Generation of confusion matrices and ROC curves.
- Supports statistical hypothesis testing.
- Performs clustering using tolerance relations.

These all features could be well placed for computationally intelligent and trusted platforms, either these platforms are attested with vTPM or pTPM.

VI. CONCLUSION

Virtualization aims to empower the end user by providing a flexible used resource in hardware and software terms.

Depending on the context, each technique has its own advantage and disadvantage. In this article, we have presented not only the traditional virtualization techniques for VMs and the new open platform for container based virtualization but also the virtualized trusted platform. In terms of system resource, the isolated container technique such as Docker is more efficient than the hypervisor. Highlighting the motivation for the trusted computing, the article proposes a road map for a trusted intelligent system. The future work aims at designing an intelligent system based on a network of trusted VMs using attested platforms.

REFERENCES

[1] T. Le Vinh, S. Bouzefrane - Trusted Platforms to secure Mobile Cloud Computing, The 16th IEEE International Conference, August 2014, pp. 1096-1103.

[2] G. J. Popek, R. P. Goldberg, Formal requirements for virtualizable third generation architectures, Commun. ACM 17 (7) (1974).

[3] VMWare Inc, Introducing vmware virtual platform, technical white paper (February 1999)

[4] Understanding Full Virtualization, Paravirtualization, and Hardware Assist, VMWre white paper.

[5] Fernando Rodriguez-Haro.The 2012 Iberoamerican Conference on Electronics. Engineering and Computer Science - A summary of virtualization techniques

[6] Matías Zabaljáuregui- -New Virtualization Techniques

[7] Jui-Hao Chiang, Optimization Techniques for Memory Virtualization based Resource Management, Stony Brook University, December 2012.

[8] TPM main specification - http://www.trustedcomputinggroup.org/resources

[9] R. Perez, R. Sailer, L. van Doorn, and others, vTPM: virtualizing the trusted platform module,in Proc. 15th Conf. on USENIX Security Symposium, 2006, pp. 305–320.

[10] P. England and J. Loeser, "Para-virtualized TPM sharing, in Trusted Computing-Challenges and Applications, Springer, 2008, pp. 119–132

[11] F. J. Krautheim, D. S. Phatak, and A. T. Sherman, Private Virtual Infrastructure: A Model for Trustworthy Utility Cloud Computing, DTIC Document, 2010.

[12] M. Strasser and H. Stamer, A software-based trusted platform module emulator, in Trusted Computing-Challenges and Applications, Springer, 2008, pp. 33–47.

[13] A.-R. Sadeghi, C. Stüble, and M. Winandy, Property-based TPM virtualization, in Information Security, Springer, 2008, pp. 1–16.

[14] B. Danev, R. J. Masti, G. O. Karame, and S. Capkun, Enabling secure VM-vTPM migration in private clouds, in Proceedings of the 27th Annual Computer Security Applications Conference, 2011, pp. 187–196.

[15] J. Cucurull and S. Guasch, Virtual TPM for a secure cloud: fallacy or reality?, RECSI 2014, Alicante, 2-5 September 2014.

[16] Virtual Trusted Platform-XEN. http://wiki.xenproject.org/wiki

[17] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio, An Updated Performance Comparison of Virtual Machines and Linux Containers (IBM report).

[18] SharathVenkatesha, Shatrugna Sadhu, Sridhar Kintali, Survey of Virtual Machine Migration Techniques (VM migration).

[19] Security in cloud computing: Opportunities and challenges Mazhar Ali Samee U. Khan, Athanasios V. Vasilakos, Information Sciences Volume 305, 1 , pp.357–383 June 2015.

[20] https://www.trustedcomputinggroup.org/wpcontent/uploads/TCG_RSA_Booklet_2016_WebRevC.pdf

[21] Soumiya. N. and Shanthi. S., Secure Virtualized Multi Tenancy Architecture in Cloud Computing using H-SVM International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 1, January 2016, DOI 10.17148/IJARCCE.2016.5125.

[22] Bashir Aliyu Yauri, Joshua Abah Mitigating Security Threats in Virtualized Environments, IJCSNS International Journal of Computer Science and Network Security, VOL.16 No.1, January 2016.

[23] Anton Burtsev, David Johnson, Mike Hibler, Eric Eide, John Regehr. Abstractions for Practical Virtual Machine Replay.. In Proceedings of the 12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'16), April 2016.

[24] Paul D. Rowe Principles of Layered Attestation, arXiv:1603.01244v1 [cs.CR] 3 Mar 2016.

[25] Contractor, D., Patel, D. and Patel, S. (2016) Trusted Heartbeat Framework for Cloud Computing, Journal of Information Security 7, 103-111. http://dx.doi.org/10.4236/jis.2016.73007.

[26] Mastroianni C, Meo M, Papuzzo G Probabilistic Consolidation of Virtual Machines in Self-Organizing Cloud Data Centers", IEEE Transaction of Cloud Computing, vol.1, pp. 215–228. 2013.

[27] Monil and Rahman VM consolidation approach based on heuristics, fuzzy logic, and migration control Journal of Cloud Computing: Advances, Systems and Applications, Springer 5:8 DOI 10.1186/s13677-016-0059-7, 2016.

[28] Polkowski, L. Rough Sets: Mathematical Foundations. Springer Science & Business Media, Berlin, 2013.