Secure Mobile Intensive Tasks Offloading to the Cloud

By

Syed Luqman Shah

2008-NUST-MS-CCS-08

Advisor

Dr. Zahid Anwar

A thesis submitted to the faculty of Department of Computing, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad in partial fulfillment for the requirements of an M.S in Computer and Communication Security

April, 2012

ABSTRACT

Mobile devices with internet access are switching towards the Cloud for resource intensive tasks. Cloud Computing has emerged as an attractive platform for mobile users to offload their resource intensive computations (e.g. consider a large image that has to be compressed and placed as wallpaper) due to limited processing and power capabilities of mobile devices. However, individuals are still reluctant to offload their tasks to the cloud due to a number of security concerns like loss of control and maintaining confidentiality and privacy of data. In this thesis, we address Mobile and Cloud Computing security concerns by encompassing Trusted Computing concepts. Trusted Computing incorporates tamper resistant hardware commodity Trusted Platform Module (TPM) and enables parties to verify what code is running on the remote computer through Remote Attestation process. We propose different scenarios 1) data computation on the mobile device 2) data computation on cloud node 3) data computation on cloud node with encryption 4) data computation on cloud node with encryption and Remote Attestation 5) data computation on cloud node with two-way Remote Attestation and encryption, each scenario more secure than the previous at the cost of some additional performance overheard. Since, there is always tradeoff between security and performance thus users can adopt any scenario matching their requirements based on threshold.

We took Android as a client of cloud services and image compression as workload to analyze the performance of proposed scenarios. We performed image compression experiments using both android emulator and a real phone on different image sizes. For emulator, offloading a 244KB image to the Cloud (Scenario -2) saves 34% of mobile device resources and reduces overall execution time to 25%. Offloading the same in Scenario-3 and 4 caused an increase of 20% in mobile device resource utilization with increase of 1.3 and 4.1 times respectively in overall execution time. However, offloading large sized images like 3MB in Scenario-3, 4 and 5 saved mobile device resources up to 88% for Scenarios-3 and 4 and 76% for Scnenario-5 with reduction of 89%, 82%, 69%, and 58% in overall execution times for Scenrios-2, 3, 4 and 5 respectively. Moreover, we performed experiments on Android Nexus One Phone and experiments show that offloading 2.1MB image to the Cloud (Scenario-3, Scenario-4) saves 79% of mobile device resources.

DEDICATION

All Praise and Thanks to Almighty Allah, The Most Beneficent and the Most Merciful, Master of The Day Of Judgment. O Allah! Guide Us With Courage And Right Path, Path of Those to Whom You Have Bestowed Your Blessings.

Dedicated to My Parents and Teachers

CERTIFICATE OF ORIGINALITY

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Syed Luqman Shah

Signature: _____

ACKNOWLEDGEMENT

Firstly, I would like to thanks my advisor Dr. Zahid Anwar, Department of Computing, NUST School of Electrical Engineering and Computer Science for his continuous encouragement and able guidance. Without his support and guidance this thesis could' not be completed in time.

I am grateful to Mr. Qasim Mehmood Rajpoot, Department of Computing for his selfless and generous support in the completion of this thesis.

I am also thankful to my committee members Dr. Fouzan Mirza, Department of Computing and Dr. Awais Shibli, Department of Computing for their able guidance and support.

I am very thankful to my family, especially my parents for their unceasing prayers, support and encouragement throughout my academic career.

Syed Luqman Shah

Table of Contents

LIST OF ABBREVIATIONS

CSP	Cloud Service Provider
TCG	Trusted Computing Group
TPM	Trusted Platform Module
AIK	Attestation Identity Key
EK	Endorsement Key
TK	Trusted Key
TSS	Trusted Software Stack
jTSS	Java Trusted Software Stack
VM	Virtual Machine
VMI	Virtual Machine Image
TC	Trusted Coordinator
TCCP	Trusted Cloud Computing Platform
OS	Operating System
CRTM	Core Root of Trust for Measurement

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Introduction

Enterprises are rapidly transferring their businesses to the Cloud due to economic benefits and because they do not want to maintain their own IT infrastructure. This new technology is getting significant acceptance, both at individual and enterprise level (1). Cloud Service Providers (CSPs) offer services at various levels of the software stack (2) including Platform as a Service (PaaS), Software as a Service (SaaS) and Infrastructure as a Service (IaaS). IaaS is the most popular way having many commercial providers like Amazon (3), Flexiscale (4), GoGrid (5), IBM (6), etc. Cloud Computing enables enterprises to go for rental IT resources such as software, development platforms, storage and network components according to their needs. Despite of these potential advantages still customers prefer running their businesses at their own infrastructure due to security issues in the Cloud. Recently a survey (7), interviewed 500 executives and IT managers in 17 countries, results have shown that executives prefer their internal infrastructure on the Cloud due to security issues. These issues include loss of control, confidentiality and privacy of data.

Trusted Computing Group (TCG) (8) provides a way that enables customers to verify trustworthiness of providers' infrastructure before sending their data to the Cloud for computations by utilizing Trusted Platform Module (TPM) (9), a co-processor residing on the motherboard of the computer and capable of generating and storing cryptographic keys and functions. Trusted Computing (TC) can be combined with Cloud Computing to provide security. Researches in this regard (10) (11) have shown that this combination can provide a considerable level of security.

These days mobile phones are being used more than expected in the past, and have successfully captured a major chunk of users' attention away from desktop machines and offer competitive internet connectivity to add to their appeal. However, the catch lies in the constrained processing capabilities and limited battery life of these devices. Mobile consumers are looking towards more powerful machines connected via GPRS, Wifi and

related networking technologies for resource-intensive tasks. The new paradigm of Mobile Cloud Computing (12) is now being touted as the next upcoming technology trend. Cloud Computing offers an attractive platform of choice for hosting resource-intensive tasks on behalf of mobile devices. Resource intensive tasks can be offloaded to the Cloud at various levels of granularity by working out a trade-off between resources required for performing computation on device against those required for offloading tasks to the Cloud. Like traditional clouds, Mobile Cloud Computing is also facing security issues. Trusted Computing can work out here too.

Currently hand held devices are missing TCG's hardware TPM. Standards for Mobile Trusted Modules (MTM) (13) have been proposed for hardware-based attestation of mobile devices and standardized hardware implementations are expected to be available soon. However, equipping mobile devices with software TPM (TPM Emulator) (14) for experimentation are good to check the feasibility of Remote Attestation between cloud node and mobile device and other TPM functions.

Trusted Cloud Computing Platform is a pioneer platform that combined Trusted and Cloud computing. In our research work we are using protocols presented by TCCP (10) with slight modifications and extend their idea.

This work aims to benchmark following five scenarios.

Benchmark 1: Data computation on mobile device

Benchmark 2: Data computation on the Cloud

Benchmark 3: Data Computation on the Cloud with encryption

Benchmark 4: Data computation on the Cloud with encryption and Remote Attestation

Benchmark 5: Data Computation on the Cloud with encryption and two-way Remote Attestation

In the above five scenarios, Android mobile device being the emerging market will be taken as a client of the Cloud services. To achieve benchmark 5, we propose a protocol for two-way Remote Attestation before sending data or launching VM on the Cloud.

1.2 Research Methodology

The first step involved extensive literature review. We sub-divided literature review in two separate phases first phase to understand different background concepts and technologies like Trusted Computing, Cloud Computing and Mobile Cloud Computing

and in second phase we studied different platforms and architectures to understand that how Trusted Computing can be combined with Cloud Computing to provide security.
In the second step, we configured systems for TPM functionalities and installed different TPM management tools. Third step includes implementation and evaluation. We used Java, Android programing and jTSS to implement all the operations involved in different scenarios separately. We used Java functions to implement different operations involved in our proposed scenarios independently and then calculated the execution time of each operation. Image compression was taken as workload and different image sizes ranging from 244KB to 9.7MB were tested. We calculate the overall execution time of a scenario by summing up the execution times of all operations involved in that scenario. Similarly, we calculate the resource utilization of mobile device by summing only on-device operations. We performed experiments using Android emulator as well as on real device Android Nexus One phone (processor QUAL Com QSD 8250, 1 GHz, RAM 512MB). For real android phone we did not calculate network and file read/write overheads. Results for android emulator show, offloading 2.1 Mb image in Scenario-2 and Scenario-3 reduces overall execution time 74% and 58% respectively with reduction of mobile device resources utilization 79% and 64% respectively. Results on Android Nexus One phone reveals, offloading 2.1MB image saves mobile device resources up to 79% for Scenarios-3 and Scenario-4.

1.3 Document Organization

This thesis has been organized in chapters. Chapter 2 provides the background study, concepts and technologies. Chapter 3 gives the literature review focusing upon how Trusted and Cloud computing can be combined to achieve security. Different Cloud Computing architectures that leverage Trusted Computing are presented and different related protocols have been discussed in chapter 3. Chapter 4 provides brief description of our proposed scenarios. Protocols used in proposed scenarios are discussed and the proposed two-way Remote Attestation protocol is also presented with complete details in chapter 4. Implementation details are given in chapter 5. Chapter 6 provides results and evaluation with discussions. Chapter 7 concludes the thesis with future recommendations.

CHAPTER 2

Background

2.1Virtualization

Virtualization is a mechanism of dividing hardware or some subset of hardware of a computer system among several virtual machines (VMs), which bear resemblance to the physical system. Virtual Machine Monitor (VMM) is a layer that divides the hardware and carries out virtualization. Xen (15) provides a virtualization layer that ensures separation of VMs and execute instructions on their behalf. At the same time, Xen manages a domain (Dom0), which controls the access of VMs to physical hardware, and facilitates communication among several machines hosted on shared hardware. The virtualized system attempts to provide virtualized network and storage devices, thus providing an environment similar to that of real system.

2.2 Cloud Computing

Cloud Computing has gained momentous acceptance (1) in the recent years. It reduces enterprises' costs of maintaining their own IT infrastructure thus, making them more competitive and more focused in their critical business operations. Companies and individuals are rapidly offloading their tasks to the Cloud due to several advantages provided by this new technology.

Cloud Service Providers (CSPs) are offering services at various layers of software stack (2) including Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). SaaS stands at the top of the stack and at the layer CSPs rent out applications software like GoogleApps (16) to their customers. In IaaS, customers can use the Virtual Machines (VMs) provided by CSPs or can launch their own VMs on providers' infrastructure and at PaaS layer CSPs offer development platforms.

2.2.1 Mobile Cloud Computing

Mobile phone industry is one of the fastest growing industries. Recent advancements, enhanced capabilities and rich connectivity provided by the mobile technology is

continuously attracting users and has already captured a major chunk of users' attention away from the desktop machines. Attractive features provided by the technology and its wide spread usage, on the other hand opened new doors for application developers to come up with innovative applications. However, the catch lies in the constrained processing capabilities and limited battery life of these devices. This is where mobile devices have to look towards more powerful machines for resource intensive computations. Cloud Computing offers an attractive platform of choice for hosting resource-intensive applications on the behalf of mobile devices. Solutions have already been proposed for augmented execution of mobile applications on the Cloud (17). Mobile phone users may utilize the software services hosted by the Cloud such as Google Apps (16) or even deploy a clone of mobile phones on the Cloud such as Amazon (EC2) (3).

2.3 Trusted Computing

2.3.1 Trusted Computing Group (TCG)

Trusted Computing Group (TCG) (8) is an industry alliance composed of major IT stakeholders like IBM, HP and Microsoft. This group works for the actual implementation of Trusted Computing (TC) concepts and aims to provide trusted platforms. So far, TCG has published TPM Main Specification (18) that presents protocols and standards for attestation, secure storage and key migration.

2.3.2 Trusted Platform Module (TPM)

Major outcome of TCG is TPM (9), a hardware chip residing on the mainboard of a computer, capable of generating and storing cryptographic keys and functions. It is a tamper-resistant hardware having a number of keys, secure data storage and a cryptographic engine. Figure 2-1 shows the structure of the TPM and we discuss major components in the coming sections.

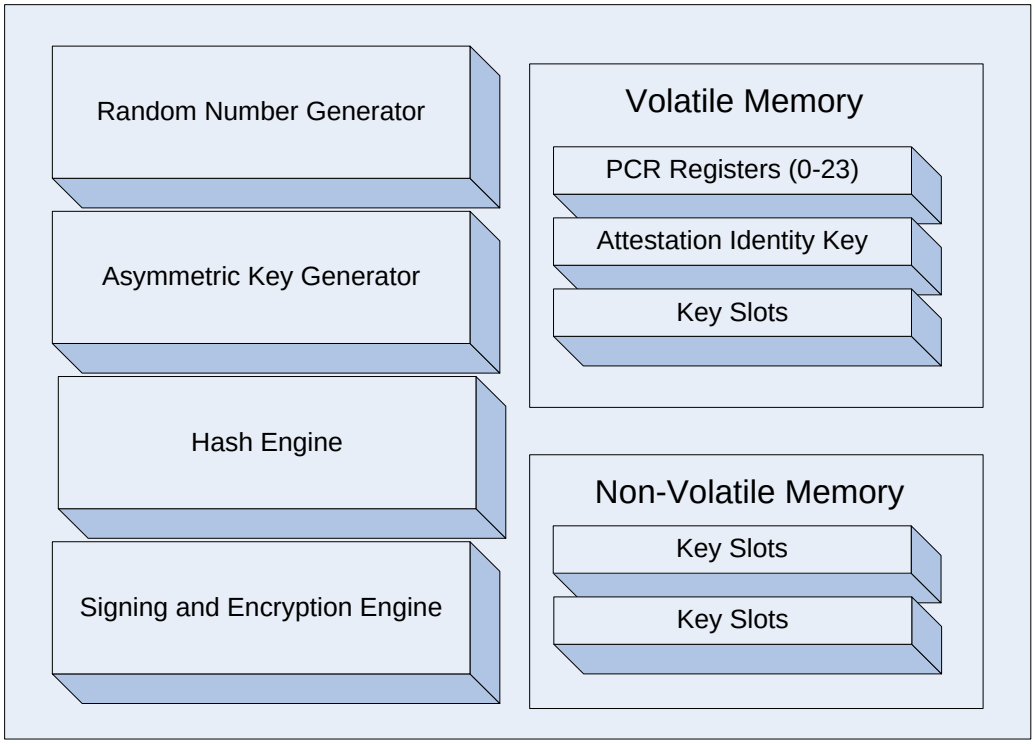


Figure 2-1 TPM Architecture

2.3.2.1 Non-Volatile Memory

TPM stores Endorsement and Storage Root Keys in its non-volatile memory and they are preserved and stay unchanged even when the computer is switched off or there is no power on the TPM. Endorsement Key (EK) is a unique RSA 2048-bit key pair embedded in the TPM at the time of manufacturing. The public part of EK is used to uniquely identify the TPM and remains the same throughout the life cycle of the TPM while manufacturers destroy information about the private part at the time of embedding.

Storage Root Key (SRK) is also a RSA key pair and used to store other storage and signing keys. TPM comes with only EK, and SRK is set each time a new user takes ownership. SRK always resides in the TPM and its public part encrypts other keys. The SRK is required to be non-migratable.

2.3.2.2 Volatile Memory

Data stored in the volatile memory of TPM is lost when computer turns off. TPM stores PCR values and Attestation Identity Key (AIK) in its volatile memory.

Platform Configuration Registers (PCRs)

As per latest specifications (18), TPM comprises of 24 PCRs that store cryptographic hashes of the software stack loaded and running on the system. These cryptographic hashes are calculated by taking SHA-1 hash of the code, module or applications loaded into the system memory.

Values in the PCRs are established automatically throughout the system boot process and up above the boot process. These values get erased when the computer reboots. PCRs are tampered resistant storage and the values can only be changed by PCR extend operation. PCR Extend operation concatenates new value with current value of PCR, calculates SHA1 and resulting value is stored in the respective PCR. Extend operation ensures that the values in the PCRs cannot be manipulated and reached via any other route. Extend operation works as follow:

New value in PCR = SHA1 (current value of PCR|| new measurement)

Attestation Identity Key (AIK)

AIK is RSA key pair and is alias of EK. It identifies a TPM instance. Both the EK and AIK can be used to identify a TPM however; using EK would compromise the privacy of end-user. AIK is used to sign messages during Remote Attestation process (discussed in section 2.3.5).

2.3.2.3 Random Number Generator (RNG)

Inclusion of nonces ascertains the freshness of different messages. TPM contains a RNG for generating nonces, symmetric keys and other random data.

2.3.2.4 Asymmetric Key Generator

TPM uses a number of asymmetric keys for cryptographic operations. Asymmetric key generator carries out generation of these keys inside the TPM.

2.3.2.5 Hash Engine

Hash engine placed inside the TPM supports different hashing algorithms like SHA-1 and MD5.

2.3.2.6 Encryption and Signing Engine

Encryption and signing are the fundamental functions of any cryptographic co-processor. The encryption and signing engine contained by TPM performs desired encryption or signing function.

2.3.3 Trusted Grub

Grand Unified Boot Loader (Grub) is the first program that execute when a computer starts. Afterwards, Grub passes control to the operating system kernel and so on. Trusted Grub (19) is the modification of standard Grub to detect and support TPM functions.

Trusted Computing is based on the principal of measuring each software and hardware component loaded since boot of the system. In order to start measurements and to provide trust there must be a root of trust. BIOS code of the computer being the only entity to trust forms the Core Root of Trust for Measurement (CRTM), and then the every next link in the chain is get measured by prior one. CRTM first calculates the BIOS, extends values in the relevant PCR and then transfers control to the BIOS. Then BIOS measures ROM configuration/data and extends values in TPM register. Similarly, all the components get measured right from the BIOS to Operating System (OS) level and measurements are recorded in the PCRs. Trusted Grub adds the measurement capabilities and connection with TPM.

2.3.4 Integrity Measurement Architecture (IMA)

Integrity Measurement Architecture (IMA) (20) is an enhancement of Linux kernel to measure each executable, library or kernel module loaded into the runtime of a system before it affects the system. IMA mechanism uses Linux Security Module (LSM) to calculate the SHA-1 of every executable mapped in the memory of the system before execution. All contents loaded into the system since boot are measured and stored in a kernel held Storage Measurement Log (SML). The integrity over SML is guaranteed by extending values in tamper-resistant PCRs of TPM.

Trusted Grub provides a way to carry out measurements to OS level and then extends these values in the PCR 0-8 of TPM. While IMA measures applications running above the OS level and stores them in SML. SML reflects both the measurements calculated by Trusted Grub and IMA thus extending trust level up to applications level.

2.3.5 Remote Attestation

TCG has provided a way to verify the integrity of a remote system by exploiting cost-effective TPM. Configuration values since the boot of the system are measured and stored in the TPM. TPM acts as trust anchor and proves stored values to a challenger through Remote Attestation protocol (21). In the remote attestation process a challenger asks the remote system (attester) to prove its integrity. Attester in response sends its PCR values signed by TPM key. Trust up to application can be established by equipping the system with some integrity measurement architecture like IMA.

2.3.6 Trusted Software Stack (TSS)

Trusted Software Stack (TSS), designed and standardized by TCG is a core software component to interact with TPM. Design and standards of TSS has been published by TCG in its 750 pages TPM Main Specification (18). TSS consists of discrete modules having clear interfaces among them as shown in Figure 2-2.

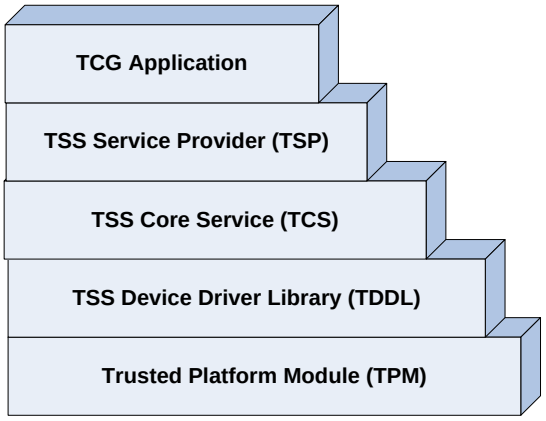


Figure 2-2 Layers of Java Trusted Software Stack

Java Trusted Software Stack (jTSS) (22) is implementation of TSS in java language. It is pure java implementation unlike the TrouSers (23), which uses C language. jTSS implements all layers of TSS including TDDL, TSP and TCS and it is passing through experimental stages yet.

CHAPTER 3

Literature Review

3.1 Introduction

This chapter gives a detailed review of providing security in the Cloud computing using Trusted Computing. We discuss different Trusted Computing protocols suitable for the Cloud. This chapter also presents a number of suggested platforms for Cloud Computing security.

3.2 Trusted Cloud Computing Platform (TCCP)

Verifying the authenticity of the providers' infrastructure whether it is safe to run customers' virtual machine and not running any malicious software, which can tamper with or can steal or harm customers' data, are problems that make enterprises hesitant to send their critical business on the cloud. In (10), authors address stated problems and propose Trusted Cloud Computing Platform (TCCP). TCCP comprises of Trusted Platform Module (TPM), Trusted Virtual Machine Monitor (TVMM) and a set of protocols. TVMM ensures the confidential execution of customer VMM for IaaS providers while set of proposed protocols enables customers to verify the integrity of the cloud infrastructure and secures operations of VM launch and VM migration.

3.2.1 TCCP Components:

Cloud nodes that host Customer VM, Cloud Manager that manages the Cloud nodes and Trusted Coordinator (TC) maintained by Trusted Third Party are main components of TCCP, as shown in Figure 3-1. Cloud manager is an untrusted entity and makes different services available to the cloud nodes.

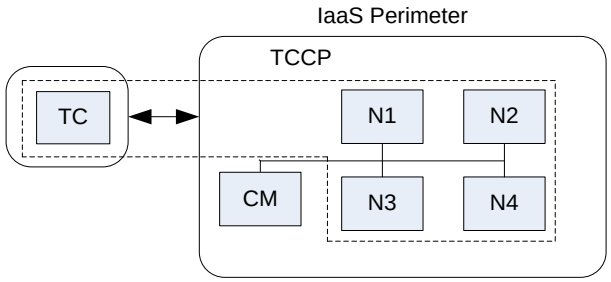


Figure 3-1 TCCP Components

3.2.2 TCCP Protocols:

TCCP secures different operations by introducing a set of protocols, which include Node Registration Protocol, VM Launch Protocol and VM Migration Protocol. Next sections discuss the proposed protocols of TCCP. These protocols utilizes TPM functionalities thus, both the Cloud Nodes and Trusted Coordinator must have a TPM chip.

3.2.2.1 Node Registration Protocol

TC maintains a directory to manage the cloud nodes. Cloud nodes have to register themselves with TC in compliance with Node Registration Protocol. Once a node attests itself with TC, TC adds its key in its directory and considers it as trusted.

1. n_N
2. $\{ML_{TC}, n_N\}EK_{TC}^{p}$, n_{TC}
3. $\{\{ ML_N, n_{TC}\}EK_N^{p}$, TK_N^{P} $\}TK_{TC}^{P}$
4. {accepted}

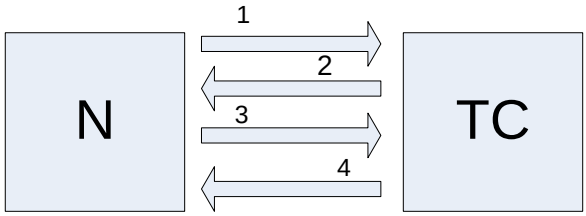


Figure 3-2 TCCP Messages Exchange Node Registration Protocol

Cloud node initiates registration process by sending its nonce to TC. In step 2, upon receiving node nonce TC sends its Measurement list (ML_{TC}) and node nonce encrypted with private part of its Endorsement Key (EK_{TC}^{p}). TC also includes its own nonce (n_{TC}) in step 2. Upon receiving message 2, Node verifies ML of TC and if it matches the expected configuration, it means TC is trusted. In step 3, Node sends its Measurement list (ML_N) and nonce of TC (n_{TC}) encrypted with private part of endorsement key (EK_N^{p}). Node also generates a Trusted Key and sends its public part to the TC (TK_N^{P}). Whole message in step 3 is again encrypted with public part of TC's trusted key (TK_{TC}^{P}). TC verifies the ML_N and if it matches with the expected one TC adds TK_N^{P} in its directory. Node stores its Trusted Key in the memory, which ensures the whole registration process to be repeated in case node reboots.

3.2.2.2 Virtual Machine Launch Protocol:

TCCP proposes the following protocol to secure the customers' VM launch operation. Proposed protocol ensures that VM is launched on a trusted node and system administrator could not compromise integrity and confidentiality of VM initial state during launch process. Following are the steps of VM launch protocol.

1. $\{VMI, \#VMI\}K_{VM}$, $\{n_u, K_{VM}\}TK^{P}_{TC}$
2. $\{\{\{n_U, K_{VM}\}TK^{P}_{TC}, n_N\}TK^{p}_{N}, N\}\ TK^{P}_{TC}$
3. $\{n_U, n_N, K_{VM}\}\ TK^{P}_{N}\}\ TK^{p}_{TC}$
4. $\{n_U, n_N\}\ K_{VM}$

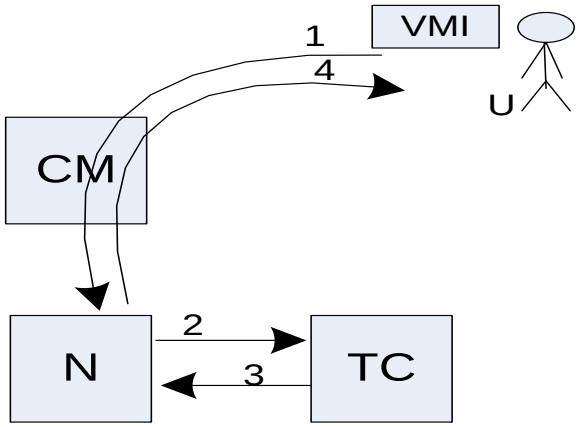


Figure 3-3 TCCP Messages Exchange VM Launch Protocol

In step 1, customer imitates VM launch operation by sending Initial state of virtual machine (VMI) and its hash (#VMI) encrypted with a session Key (K_{VM}) along with KVM encrypted with public part of Trusted Key of the TC ($TK^{P}_{TC)}$ to the cloud. CM designates a node for hosting VM. Since K_{VM} is encrypted with, TK of TC and only TC can decrypt the key. In step 2, node asks the TC to decrypt key. TC verifies the trust of node by checking node's TK in its directory. If the node is found trusted, K_{VM} is forwarded to that node in step 3. Finally, in step 4, node sends its identity to user.

3.2.2.3 VM Migration Protocol

Live migration of VM is the process of transferring the running state of a virtual machine between two nodes on the Cloud. Normally, live migration process is carried out for load balancing data among the cloud nodes. TCCP presents a protocol to secure this operation, which guarantees that VM is migrated only on trusted node. This protocol also ensures the confidentiality and integrity of virtual machine state whilst in transit on network. Following are the messages exchanged in this protocol.

1. $\{\{N_d, n_{s1}\}TK^p_N, N_s\}TK^P_{TC0}$
2. $\{n_{s1}, TK^P_{Nd}\}\ TK^P_{Ns}\}\ TK^p_{TC}$
3. $\{K_S, n_{s2}\}\ TK^p_{Ns}, N_s\}\ TK^P_{Nd}$
4. $\{N_S, n_d\}\ TK^p_{Nd}, N_d\}TK^P_{TC}$
5. $\{n_d, TK^P_{Ns}\}\ TK^P_{Nd}\}TK^p_{TC}$
6. $\{n_d\}\ K_S$
7. $\{VM_{id}, \#VM_{id}\}\ K_S$

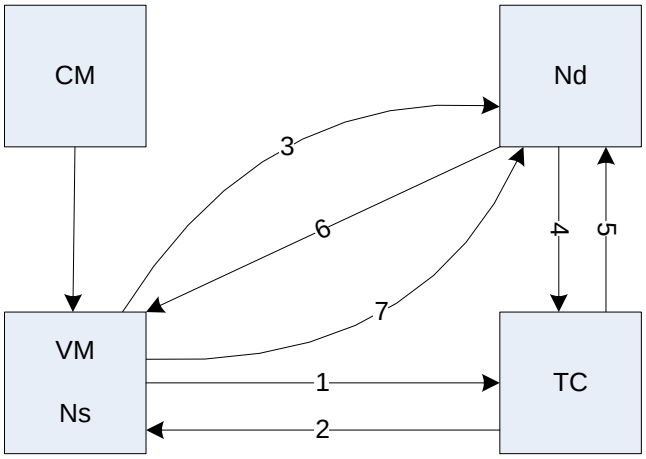


Figure 3-4 TCCP Message Exchange VM Migration Protocol

3.3 Trusted Cloud Computing Platform with Sealed Storage Ability (TSSC)

In the Cloud computing customers' sensitive data is under the control of Cloud Service Provider (CSP), which is a security threat and an obstacle for enterprises and individuals to send their critical data on the Cloud. Trusted Computing technology enables customers to verify the integrity of CSP platform before launching their VM through Remote

Attestation protocol. TCCP (10) presents a mechanism, which enable customers to attest the providers' platform upfront and ensures closed box execution of customers' VM. Since, Trusted Third Party is used for Remote Attestation in TCCP, which makes the Cloud resources visible to the third party. Secondly, TCCP does not have a mechanism to protect clouds' customer data, once the cloud nodes become compromised at runtime. In (11), authors present Trusted Cloud Computing Platform with Sealed Storage Capability (TSSC). TSSC also leverages Trusted Computing to address the above stated problems by introducing the following mechanism.

- Remote Attestation to attest providers' platform without exposing Cloud resources to third party.
- Sealed storage mechanism to ensure the privacy of customers' data.

3.3.1 TSSC Components

TSSC consists of Remote Delegation Service (RDS), VMM monitor with Trusted Boot ability (TVMM), Sealed Storage Module (SSM) and Remote attestation module (RAM). Coming sections explain each of these components.

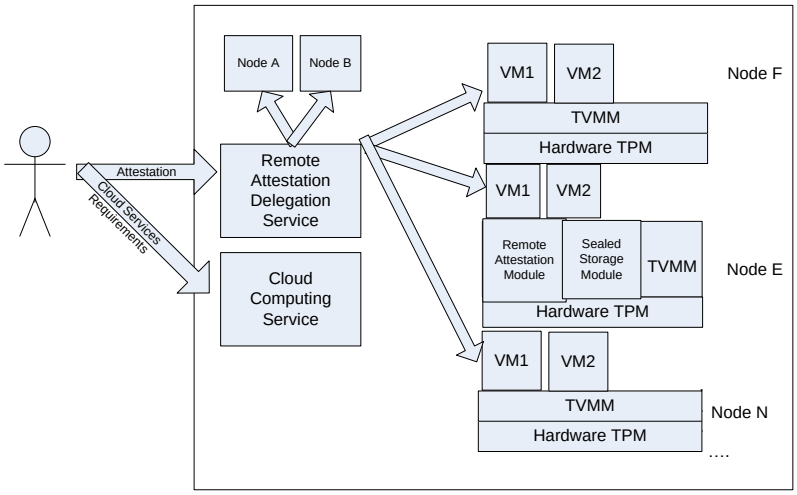


Figure 3-4 TSSC Components

3.3.1.1 Remote Delegation Service (RDS)

RDS is an independent entity equipped with TPM and IMA and owned by service provider. Customers attest Cloud back-end nodes through RDS. RDS communicates with RAM, SSM and resource management software for Remote Attestation and monitoring the integrity changes of nodes to establish sealed storage. Since, cloud owner maintains

RDS thus ensures protection of Cloud resources against the third party but on the other hand, Cloud owner may have incentive to deceive customer. This problem can be catered by assuming the RDS close to control and secrecy features.

3.3.1.2 Trusted Virtual Machine Monitor

Cloud back-end nodes must be equipped with a virtual machine monitor which supports TCG-style chain of trust. Such trusted layer can prevent privileged system administrator to inspect or tamper applications or data in customer VM.

3.3.1.3 Remote Attestation Module

Remote attestation module runs at cloud back-end node and reports integrity measurements of node to RDS.

3.3.1.4 Sealed Storage Module (SSM)

Sealed storage is the property of TPM, which encrypts the data with key and particular values of Platform Configuration Registers (PCR) and decryption of data can only be done when PCR values match the one at the time of encryption.

In TSSC, SSM residing at back-end node dynamically encrypt data specified by cloud user. RDS coordinates with SSM to achieve sealed storage.

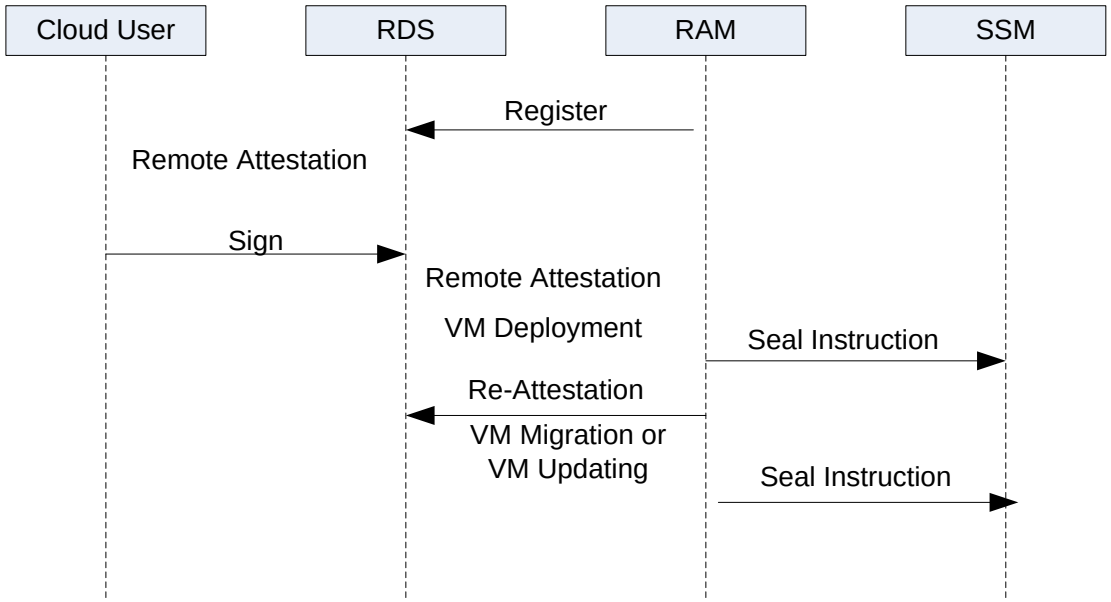


Figure 3-5 TSSC Messages Exchange

Authors introduced the concept of RDS, which ensures that Cloud Resources (Measurement List) is not exposed directly to the third party. Sealed storage mechanism

has been introduced which protects the user data even the back-end node becomes compromised after successful attestation. Authors implemented a prototype of the proposed work and results have shown that Remote Attestation costs double time as compared to un-secure cloud. For sealed storage mechanism, (encryption/decryption) performance degradation is 15% for 4 MB file and increases with file size. Security of Cloud Computing eventually depends upon VMM. Working on Trusted VMM is proposed in this article.

3.4 Multi Tenancy Trusted Computing Environment Model (MTCEM)

Cloud computing is different from traditional IT service in a sense that the owner and users of the Cloud services are separated. CSPs provide services in the form of applications, platform or hardware and customers rent them for their businesses. This requires a security duty separation between the customer and cloud service provider. CSP should protect the services they provide and cannot exceed customer authorities while customer should secure their data. Authors, in (24), propose a Multi Tenancy Computing Environment Model (MTCEM) to address this problem. MTCEM utilizes Remote Attestation and transitive trust features of Trusted Computing and provides a security duty separation between cloud customer and CSP.

MTCEM provides security duty separation for IaaS model, where CSP provides space to customers for launching their virtual instances. Figure3-6 depicts the typical IaaS model.

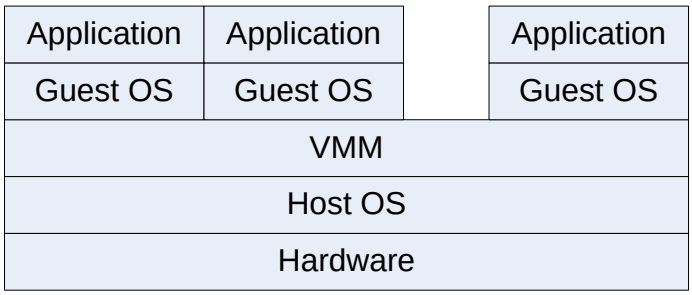


Figure 3-6 An IaaS Model Example

In IaaS, whole system is clearly divided into two parts. First level starting from CRTM to VMM level and second from the boot of guest Operating System (OS) to launched applications in guest OS as shown in Figure 3-7.

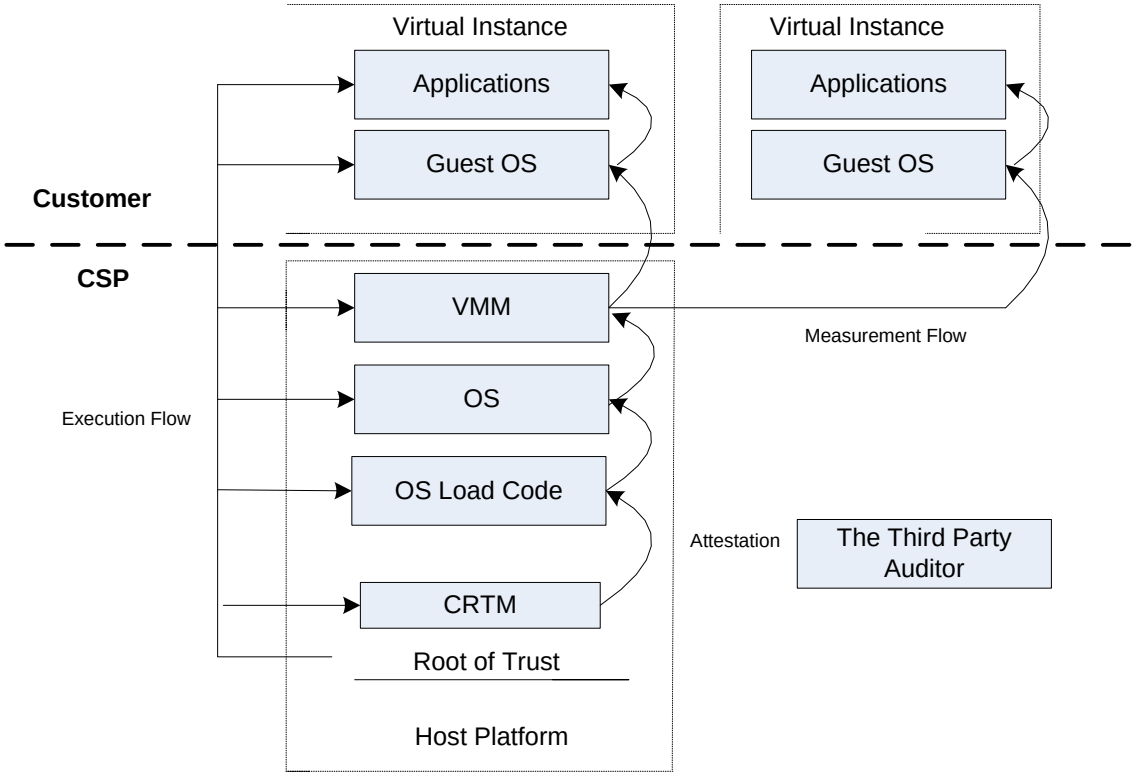


Figure 3-7 MTCEM

Since, first level is under the full control of CSP thus, CSP should ensure the security of this level. Third Party Auditor can optionally be used to attest the providers' infrastructure up to VMM level. Ensuring the security for second level lies with cloud customer. It must be ensured that neither other customers nor CSP have privileges to interfere with customers launched VM and applications running in the VM.

Authors implement a prototype of proposed model and it works with less than 1% performance degradation. MTCEM is proposed for IaaS, however, it can be extended to other delivery models like PaaS and SaaS.

3.5 Integrity Measurement Model Based on Trusted Virtual Platform

Ensuring the secure execution environment is fundamental need for distributed computing, so that integrity of the providers' infrastructure should be verified before launching services on them. In this paper (25) authors, propose a secure model, which combines trusted computing and virtualization to provide a secure environment.

3.5.1 Virtual TPM

Hardware TPM and IMA measures and reports integrity values from boot time to load of Operating System and up above to application level. Limitation of traditional TPM is that it can only measure the integrity value of VMM but not the applications running inside a VM. In order to measure applications running inside a VM there should be a virtual TPM bound to each VM. In (26), vTPM has been proposed which can be ported to Xen Hypervisor to make IMA capabilities available to each virtual instance.

3.5.2 Integrity Model Based on Xen

Authors propose integrity model based on Xen hypervisor and utilize concept of virtual TPM to attest applications running inside the virtual machine. Management Domain Dom0 of hypervisor makes hardware TPM functions available to virtual machine. Xen Hypervisor/VMM consists of many Domains starting from Dom0, Dom1 and so on. Each of these domains hosts a virtual machine instance except Dom0, which is management Domain for other Domains.

In the proposed architecture, Dom0 has many compartments as shown in Figure 3-8. Main compartments of Dom0 include Attestation agent (AA), Virtual Machine Controller (VMC) and Policy Enforcer (PE). AA measures integrity of VMM core and DomU and reports to remote service requester. VMC is responsible for launching new virtual machine according to requirements of Service Requester (SR). Each virtual machine is bound with a vTPM instance to make TPM functions available to VM.

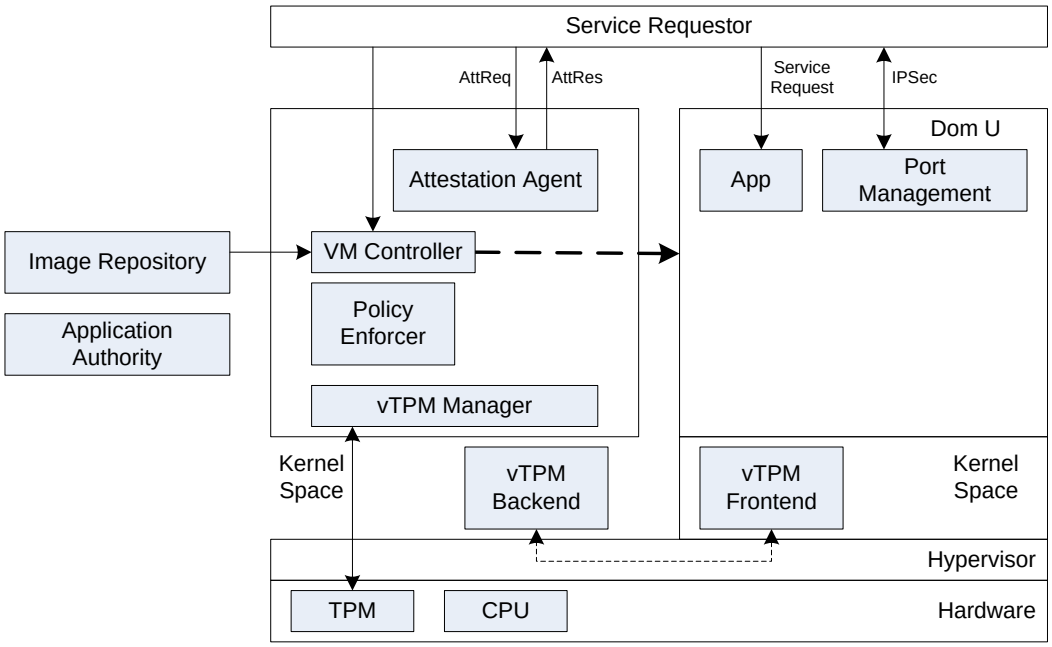


Figure 3-8 Integrity Model Based on XEN

3.5.3 Remote Attestation of Service Provider

Proposed model enables Service Requester to verify the integrity of Service Provider's platform. Remote attestation protocol proposed here can be divided in two phases. In first phase, SR verifies integrity up to VMM level of SP through traditional Remote Attestation process of hardware TPM, then SR requests to launch a VM. After VM launch process, the integrity of requested VM is verified through vTPM in second phase.

3.6 Information Leakage in Third-Party Compute Clouds

Cloud computing providers multiplex many customer virtual machines on shared single medium. In the underlying article (27), authors showed that how cross-VM-channel attacks can be launched to extract information from target machine by co-resident machine. In this article, authors took EC2 cloud as a case study.

CHAPTER 4

Design and Architecture

4.1 Introduction

This chapter presents a brief description of the proposed scenarios. Proposed two-way remote attestation protocol is also presented in this chapter.

4.2 Proposed Scenarios

Recent advancements in the mobile phone industry have significantly increased the number of users and mobile phones are in use more than expected in the past. In fact, users are expecting hand-held devices to perform their resource intensive computations. However, limited processing and power capabilities of hand-held devices are major constraints. Users can perform on-device computations to some extent but then processing, power and memory limitations of mobile devices make them to look towards machines that are more powerful. This is what Cloud and Mobile Cloud computing are offering.

Mobile cloud computing provides an attractive platform of choice and can be thought as good alternate of limited processing and power capabilities of the hand-held devices. Despite of the advantages provided by the cloud computing still individuals are reluctant to offload their data to the cloud due to privacy concerns. These concerns may include confidentiality and privacy of data whilst in transit on the network between the mobile device and the cloud platform. Encryption can be a good solution to this problem.

Other concerns customers may have while offloading data to the cloud include the integrity of the providers' platform. Since, the provider's infrastructure is untrusted for the mobile client thus it can run malicious software that can copy or tamper with customer data. Remote attestation provides a way to check the integrity of providers' platform before offloading data to the cloud.

Besides the security concerns of the users there may come situations when the cloud also wants to verify the integrity of the customers device before granting it access to launch VM. The solution can be two-way remote attestation.

In this work, we present five different scenarios ranging from on-device computation to offload data computation with two-way remote attestation, each scenario introducing an increased level of security from the previous one. Following sections provide the brief description of each scenario.

4.2.1 Data Computation on the Mobile Device

This sets the first scenario of our work. In this scenario, the mobile device performs data computation locally (on-device). We present this scenario for comparison of offloading data to the Cloud with local computation.

4.2.2 Data Computation on the Cloud Node without Encryption

Customer (mobile client) sends data for computation to the Cloud node without encryption as depicted in Figure 4-1. This scenario sets the second benchmark of our work. It is designed without any encryption or remote attestation, and will help us to compare the results and to calculate the overhead incurred in next scenarios.

Messages Exchange:

1. Data
2. Result

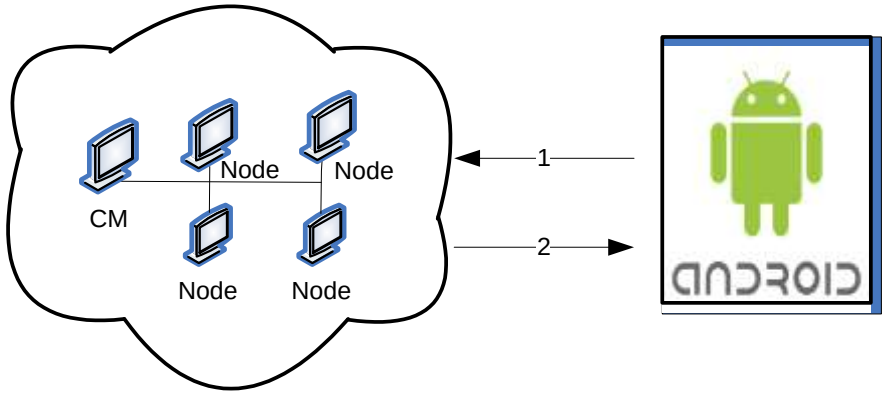


Figure 4-1 Data Computation on the Cloud Node without Encryption

4.2.3 Data Computation on the Cloud Node with Encryption

Third scenario is designed with encryption where the customer sends encrypted data to the cloud node for computation. The cloud node decrypts the customer's data; performs

requested operation and sends back the encrypted results to the customer. Figure 4-2 illustrates the messages exchanged between the mobile phone and the cloud node.

Messages Exchange:

1. Encrypted Data
2. Encrypted Results

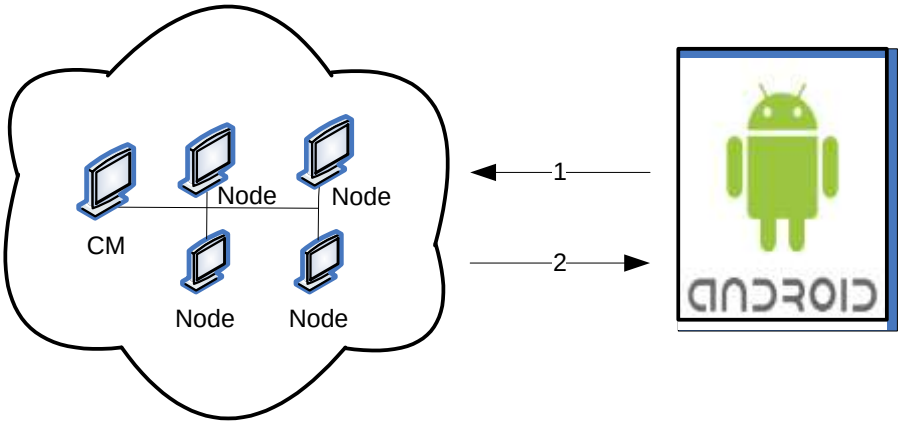


Figure 4-2 Data Computation on the Cloud Node with Encryption

4.2.4 Data Computation on the Cloud with Node Attestation and Encryption

This scenario sets fourth benchmark of our work and introduces end-to-end encryption with a mechanism to verify the integrity of cloud's infrastructure. The major participating entities in this scenario include 1) a mobile client, 2) a cloud node, which performs the offloaded computations, and 3) a Trusted Coordinator (TC) that acts as the trusted third party and attests the cloud nodes. TC is just like a certification authority and can easily be maintained. Cloud node and trusted coordinator both are TPM-equipped systems and performs remote attestation of the each other. Figure 4-3 provides an overview of third scenario.

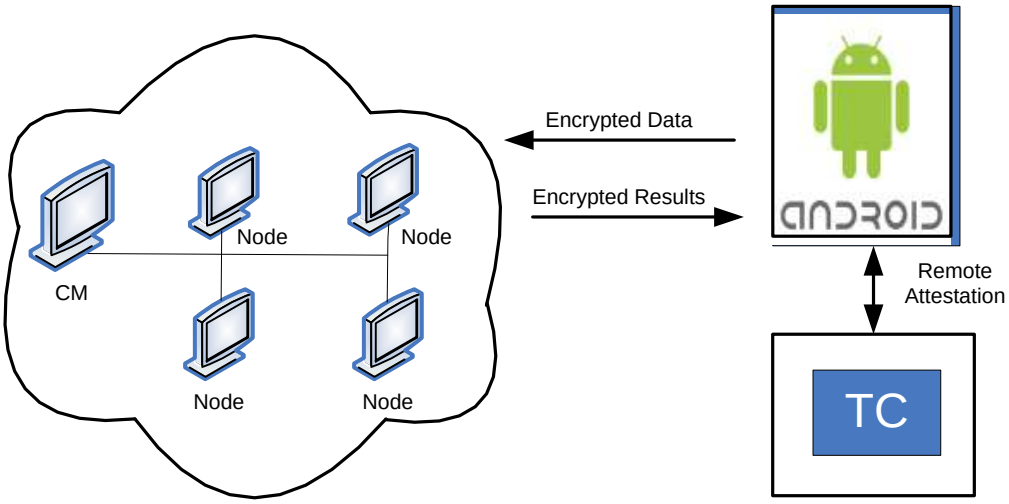


Figure 4-3 Data Computation with Remote Attestation and Encryption

In this scenario, each cloud node must have to register itself through Remote Attestation protocol with the Trusted Coordinator to perform computations on the behalf of the customer. Thus, implementation of this scenario requires a) Remote Attestation of the cloud node by the TC b) Data computation by the cloud on the behalf of the customer as described in coming sections.

4.2.4.1 Node Attestation

Registration of each cloud node with TC is a mandatory step that is performed before the node can be used in the cloud to perform any computations on behalf of the mobile clients. During the registration process, each cloud node generates a RSA key pair and stores the public part of generated RSA Key pair (TK_N^{public}) in its volatile memory. TK_N^{public} must be known to the TC that is later used to exchange the key between the node and the TC as explained in the coming section. Figure 4-4 shows the messages exchanged between the nodes and the TC during the registration process.

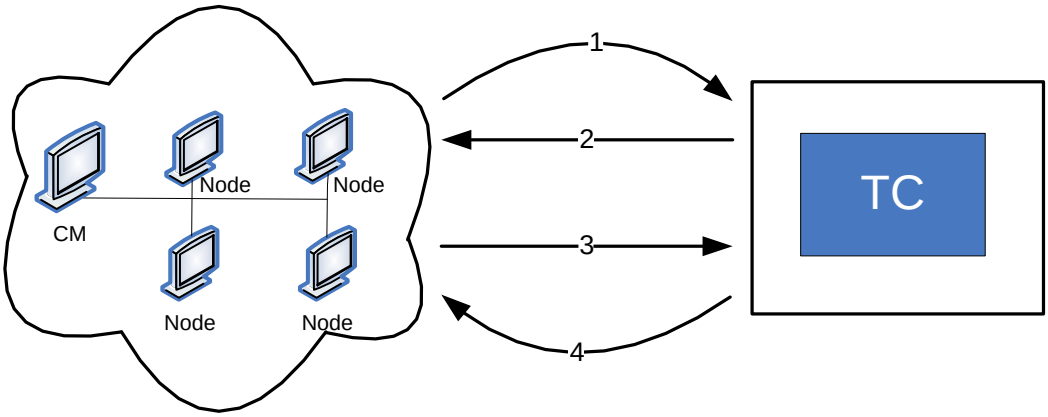


Figure 4-4 Messages Exchanged in the Node Attestation Protocol

1. n_{Ni}
2. $AIK_{TC}^{private}\{n_N, PCR10\}, n_{TC}, SML_{TC}$
3. $TK_{TC}^{public}\{AIK_N^{private}\{n_{TC}, PCR10\}, TK_N^{public}\}, SML_N$
4. $TK_N^{public}\{Registered\}$

When a node wishes to register itself with the TC, 1) it generates a nonce (n_N) and sends it to the TC. Upon receiving the nonce, 2) the TC signs the nonce along with the value of the PCR-10 register using its AIK ($AIK_{TC}^{private}$). It also sends the SML_{TC} and a nonce (n_{TC}) along with the encrypted content to the node. The node verifies the signature of the TC using AIK of the TC (AIK_{TC}^{public}) and then verifies the SML of the TC against the signed PCR-10 value. Since the nonce generated by the node (n_N) is also signed along with the PCR-10 contents, the node can be sure that it is a fresh value reflecting the current configuration of the TC. Afterwards, 3) the node sends the value of its PCR-10 and the nonce sent by the TC (n_{TC}) signed by its AIK ($AIK_N^{private}$). It also sends the SML_N so that the TC can verify the SML_N against PCR-10. The node also sends the public part of its Trusted Key (TK_N^{public}) to the TC, which is stored with the TC and is used during key exchange as explained in the next section. All the message content, except for the SML_N, is encrypted using the TK_{TC}^{public} so that it is only accessible by the TC.

Upon receiving message (3), TC decrypts the message using its $TK_{TC}^{private}$, verifies the signature on PCR-10 and matches the SML_N against the PCR-10. TC also compares the

signed nonce sent back by the node to ensure that the SML_N are current. If all is verified, 4) TC sends the registration message signed by the newly received TK_N public of the node. This signals the successful registration of the node with the TC along with the acceptance of the node's TK_N^{public}.

The node registration procedure is independent of the fact that the nodes host data/applications sent by a mobile client or a computer. We used here the notion of Trusted Clouds (10).

4.2.4.2 Data Computation

This section describes data computation on the cloud node. Both the mobile client and cloud nodes have access to the trusted key of TC ($TK_{TC}^{public)}$. The TC also has the trusted keys of the nodes (TK_N^{public}).

When a mobile client wishes to offload the computation of a resource-intensive task to the cloud, 1) it encrypts the associated data using an $RC4_{key}$ before sending it to the cloud. The mobile device also needs to communicate the $RC4_{key}$ so that the cloud node can decrypt the data. The $RC4_{key}$ is encrypted using the TK_{TC}^{public} restricting the cloud node to access the $RC4_{key}$ and hence the encrypted information, without first contacting the TC and proving that it has already been registered. The mobile client also sends a nonce (n_U i.e. nonce generated by the user) encrypted with the $RC4_{key}$ to ensure the freshness of the transaction.

Upon receiving the encrypted message, the node must have to prove to the TC that it is trustworthy so that the TC may decrypt and send the $RC4_{key}$ to the node. The various messages exchanged between communicating parties for trusted computation are shown in Figure 4-5.

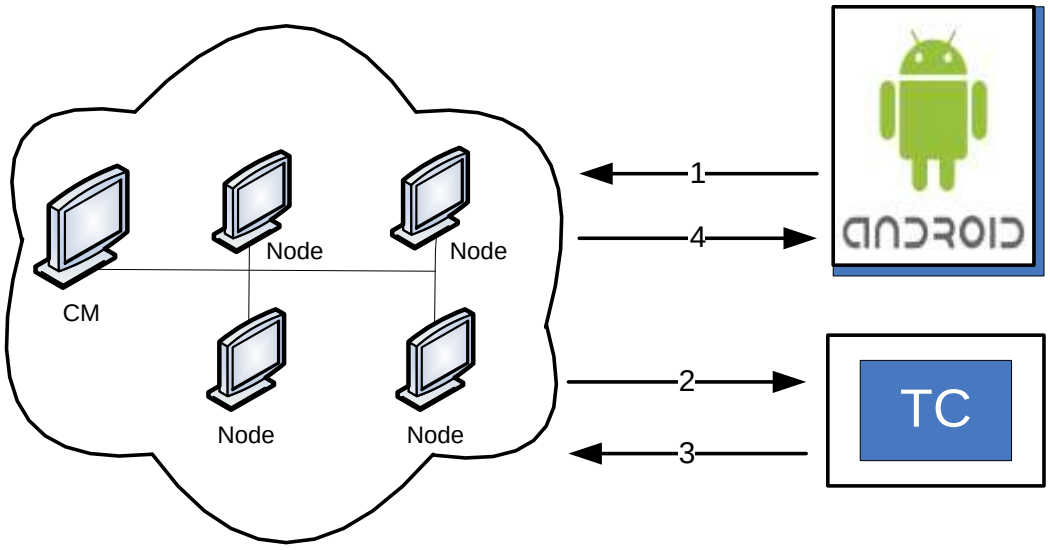


Figure 4-5 Messages Exchanged for Data Computation

1. $RC4_{key}\{Data\}, TK_{TC}^{public}\{n_u, RC4_{key}\}$
2. $TK_N^{private}\{SHA\text{-}1\{TK_{TC}^{public}\{n_U, RC4_{key}\}, n_N, N\}, TK_{TC}^{public}\{n_U, RC4_{key}\}, n_N, N\}$
3. $TK_{TC}^{private}\{SHA\text{-}1\{n_N, n_U, RC4_{key}\}\}, TK_N^{public}\{n_N, n_U, RC4_{key}\}$
4. $RC4_{key}\{n_U, N, Result\}$

To gain the access of $RC4_{key}$ encrypted with the TK_{TC}^{public}, 2) the cloud node passes on the encrypted key and nonce sent by the mobile user n_U to the TC. It also sends a nonce n_N and node identity N to the TC using TK_{TC}^{public}. In order to demonstrate the fact that the node is already registered with the TC, it computes the hash of the above parameters and signs it using node's trusted key $TK_N^{private}$. If the node is already registered with the TC, TC is able to decrypt this part using TK_N^{public} of the node - provided at the time of registration (explained in section 4.2.4.1) - decrypts the $RC4_{key}$ and 3) sends back to the node the $RC4_{key}$ along with the n_U and the n_N encrypted with the TK_N^{public} of the node. In order to ensure that the message is from the TC, TC also computes the hash of the contents and then signs it using its $TK_{TC}^{private}$. In case the node is not already registered, it is first required to register itself with the TC as explained in section 4.2.4.1.

Once the node receives $RC4_{key}$, it decrypts the message, performs the requested computations on behalf of the mobile client, and 4) sends back the results encrypted by

the same symmetric key ($RC4_{key}$). The cloud node also sends back the original nonce n_U to validate that the results are from a current computation.

4.2.5 Data Computation on the Cloud with Two Way Attestation and Encryption

This scenario is designed with two-way remote attestation. The participating entities in this scenario are a) Cloud node b) TPM enabled Trusted Coordinator c) TPM enabled mobile device. We propose a protocol for two-way attestation Figure 4-6 depicts messages flow among the participating entities.

TC attests the Cloud node as described in (section 4.2.4.1) and then Cloud node attests the mobile device. Cloud node performs computations only after successful attestation of mobile device.

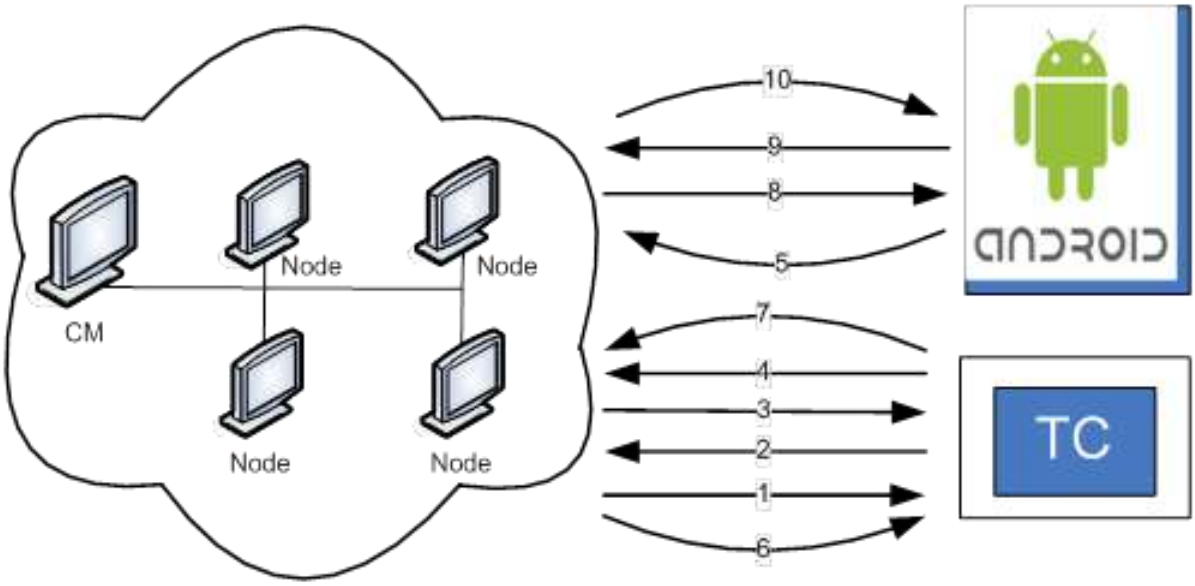


Figure 4-6 Messages Exchanged Two-way Remote Attestation Protocol

1. n_N
2. $AIK_{TC}^{private}\{n_N, PCR10\}, n_{TC}, SML_{TC}$
3. $TK_{TC}^{public}\{AIK_N^{private}\{n_{TC}, PCR10\}, TK_N^{public}\}, SML_N$
4. $TK_N^{public}\{Registered\}$

5. $TK_{TC}^{public}\{n_{Android}, RC4_{key}\}$, $AIK_{Android}^{public}$
6. $TK_{TC}^{public}\{TK_{N}^{private}\{TK_{TC}^{public}\{n_{Android}, RC4_{key}\},n_{N}\},N\}$
7. $TK_{TC}^{private}\{TK_{N}^{public}\{n_{Android}, n_{N},RC4_{key}\}\}$
8. $RC4_{key}\{n_{Android}, n_{N}, TK_{N}^{Public}\}$
9. $RC4_{key}\{AIK_{Android}^{private}\{PCR, n_{N}\}, VMI, ML_{Android}\}$
10. $RC4_{key}\{Acknowledgement\}$

Steps 1 to 4 have been borrowed from Remote Attestation Protocol as explained in node registration section (3.2.2.1). Upon the successful registration of node in 5) mobile device generates a nonce $n_{Android}$ and a symmetric key $RC4_{key}$ (which later will be used to encrypt data), encrypts both with the Public Part of TC's Trusted Key (TK_{TC}^{public}). Mobile Device also includes its Public Part of Attestation Identity Key ($AIK_{Android}^{public}$) and sends to the Cloud node through CM. Upon receiving this messages Cloud node in 6) forwards a part of message encrypted with TK_{TC}^{public}, a node nonce n_N signed with Private Part of its Trusted Key ($TK_{N}^{private}$) to the TC. Node also sends its identity (N) and encrypts whole message with TK_{TC}^{public} to ensure confidentiality of the message. TC decrypts sent message and sends ($RC4_{key}$), nonces $n_{Android}$ and n_N encrypted with Public part of Node's Trusted Key (TK_{N}^{public}) and signed with its own Private part of TK ($TK_{TC}^{private}$) in step 7. Encryption with Node's public part ensures confidentiality while encryption with its own private part ensures integrity of message. In 8) node sends nonces along with Public Part of its Trusted Key (TK_{N}^{public}) encrypted with $RC4_{key}$ to mobile device (Android phone). Upon receiving this message mobile device decrypts this messages, verifies nonce against its sent nonce. If verified in 9) sends its PCR-10 and Node Nonce (n_N) signed with private part of its Attestation Identity Key $AIK_{Android}^{private}$ (TPM quote operation) along with its Measurement List (ML) and Virtual Machine Image/Data (VMI) to the Cloud node. Mobile device encrypts whole message (9) with $RC4_{key}$. Upon receiving this message, Cloud node verifies PCR value of mobile device against ML. If verified launches VMI/data of mobile device on its cloud and in 10) sends back encrypted results or not launched response in case not verified.

CHAPTER 5

Implementation

5.1 Introduction

This chapter provides implementation details. We used TPM enabled computers, Java Trusted Software Stack (jTSS) and Java programing language for the operations involving TPM and used java functions for implementing mobile device related operations and tested both on the Android Emulator as well as on real Android device. We took Image compression as a workload to analyze the performance. Coming sections provide complete implementation details.

5.1.1 Configuring TPM in the Linux Kernel

Utilizing TPM capabilities requires some kernel configurations and compilation. First step in our implementation involved making necessary configurations in the Linux kernel and then compiling it to access TPM. We used TPM enabled computers (Core 2 Duo) having Ubuntu 9.04 operating system and followed the below given steps to configure and compile the Linux kernel for TPM.

1. Downloaded the kernel source code from www.kernel.org. We have used 2.6.30.
2. Extract the source in usr/src.
3. Go to source directory and run: `sudo make menuconfig`
4. Select Device Drivers Option and Press Enter.
5. Select Character Device option and press ENTER.
6. In the Character Devices Menu Scroll to the Bottom and Select TPM Hardware Support . The TPM Hardware support is by default as Module as indicated by <M>, <*> means it is part of kernel and won't be loaded as module, If you want to make it part of kernel then press space on <M> to make it as <*>.
7. Inside TPM Hardware Menu Enable as Kernel part
8. Exit and save configurations.
9. Compile Kernel: `sudo make`
10. Compile Kernel Modules: `sudo make modules`

Figure 5-1 Configuring Kernel for TPM

5.1.2 Configuring Trusted Grub

Trusted Grub (19) provides TCG measurements capabilities. There must be a complaint boot-loader to utilize the TPM functionalities Figure 5-2 illustrates the steps to build the Trusted Grub for the TPM enabled computer.

1. Download source code from
2. http://sourceforge.net/projects/trustedgrub/files/TrustedGRUB-1.1.5/TrustedGRUB1.1.5.tar.gz/download
3. `sudo tar -xzvf TrustedGRUB-1.1.5.tar.gz`
4. `cd TrustedGRUB-1.1.5/`
5. `sudo apt-get install automake`
6. `sudo apt-get install autoconf`
7. `sudo apt-get install gcc`
8. `/build_tgrub.sh`
9. Then, before installing anything copy your actual Grub Folder and prepare a live Cd.
10. `sudo mv /boot/grub /boot/grub old`
11. `sudo dpkg -r grub`
12. `cd TrustedGRUB-1.1.5/`
13. `sudo make install`
14. `sudo mkdir /boot/grubsudo cp ../default /boot/grub/`

Figure 5-2 Steps to Build the Trusted Grub

5.1.3 Integrity Measurement Architecture

IMA measures each executable loaded into the runtime of system but before it affects the system. Measurements are being stored in the kernel held SML and integrity over the measurements is assured by extending these values in the PCR of TPM. These measurements provide a way for remote parties to verify what code is running on the target computer. We used IBM IMA architecture and enabled it in the kernel and followed the steps given in Figure 5-3 to enable a IBM IMA in Linux Kernel.

1. Start Compiling kernel 2.6.30
2. `make menuconfig`
3. Select Security Options in the menu list.
4. Press Enter and scroll to Integrity Measurement Architecture (IMA) option.
5. Press Y to make the Integrity Measurement Architecture as the part of kernel. You will see a * after pressing Y
6. After enabling IMA the TPM is enabled automatically as part of Kernel not as module. It can be confirmed by navigating to Device Drivers>Character Devices>TPM Hardware Support. You will see that a * is appeared at the TPM Hardware Support Option.
7. Exit and save Changes to menuconfig and continue Kernel compilation.
8. Reboot the System with new Kernel and check if IMA is running. Use the following command:
9. `dmesg | grep ima`
10. Output of the above command will be like this on Ubuntu Systems if IMA is running.

```
[ 3.280110] Trying to unpack rootfs image as initramfs...
[ 9.568204] PM: Checking hibernation image.
[ 54.308667] ima_file_free: drm mm object open/free imbalance
(r:0
w:0 o:0 f:0)
[ 54.308682] [<c02b38d5>] ima_file_free+0x105/0x120
```

11. Mount securityfs:
    `mount -t securityfs securityfs /sys/kernel/security`
12. IMA will create a new directory for calculation of SML. This directory will appear in /sys/kernel/security/ima. IMA directory contains following files.
    - ascii_runtime_measurements
    - binary_runtime_measurements
    - runtime_measurements_count
13. SML can be read as ASCII measurements. Use the following command to read SML list:
    `cat /sys/kernel/secuirty/ima/ascii_runtime_measurements`
14. Output will be like this:

```
10 3440388c0fcda4f914f519c247e8de283413bd7f ima
68d632b50d84bcfb6f7ea6373e8654f7bd1ebb3c
boot_aggregate
10 8a11aa2017bfdf52ae1ab8cfb277fc651bc7d611 ima
e6d56d44e22b8f6b783c039d45703e8fd28cb796 /init
10 a078e19e5ea2bf75ed353fc6613f7132863618d5 ima
3d90e18f67f1c580c1212126a3c22cf07c7288dd /init
```

15. The first entry in the above list is PCR number here is 10. Second entry is hash (file data + file name hint). Fourth entry is hash of file data and last one is file name hint.

Figure 5-3 Steps to Enable IMA in the Linux Kernel

5.1.4 Interaction with TPM

After making the necessary configurations as explained in the previous sections, we installed TPM tools on the computer for interaction with TPM. These tools include a) TrouSers b) TPM Manager.

a) TrouSers

TrouSers is an open source implementation of Trusted Software Stack. It was created and released by IBM under GPL and provides a well-defined interface for application developer to access all TPM functions. TrouSers runs as a daemon. We installed TrouSers and used different TPM functions via command line. Following command installs TrouSers on the PC.

```
sudo apt-get install tpm-tools
```

Since, TrouSers runs as a daemon so it can be started or stopped any time.

```
sudo /etc/init.d/trousers start
```

Few TrouSers Commands

TPM Version	`sudo tpm_version`
Getting Public Endorsement Key	`sudo tpm_getpubek`
Taking Ownership	`sudo tpm_takeownership`
Viewing PCR values	`sudo cat /sys/class/misc/tpm0/device/pcrs`

b) TPM Manager

In the preliminary stages of implementation, we also installed the TPM Manager to get an overview of TPM capabilities**.** TPM Manager is an open source software and provides GUI interface to view, manage and change settings of TPM like enabling and disabling TPM, viewing PCR values, changing ownership etc. This tools uses TrouSers daemon so, TrouSers must be running on the system.

5.1.5 Programing TPM Related Operations

jTSS is a java-based implementation of Trusted Software Stack (TSS) including TSP and TCS layers with clear and defined interfaces among them. The TSP siting at the top of TSS provides an interface for java applications to use TPM functions and it interacts with the lower layer (TCS) operations in the software stack.

For TPM related operations, we wrote separate java programs using jTSS. Figure 5-4 illustrates the steps involved in setting up initial development environment.

1. jTSS uses the tpm0 file located in /dev/tpm0.
2. Install Java:
   a. `apt-cache search jdk`
   b. `apt-get install sun-java6-jdk sun-java6-jre`
3. Install Netbeans, Eclipse or any other IDE if required.
4. Download jTSS from http://sourceforge.net/projects/trustedjava/files/
5. Unzip jTSS-0.5 folder.
6. Copy the jar libraries and ini files located in jTSS_0.5/lib to /usr/lib/jvm/java-6-sun-1.6.0.20/jre/lib/ext/ OR set class variables for above folder.
7. In order to use full security capabilities of java add JCE Unlimited Strength Jurisdiction Policy (JCE)Files.

Figure 5-4 Setting up Development Environment using jTSS

5.1.5.1 Sample Java Programs to use TPM functionalities

jTSS can be used in two modes a) local binding b) SOAP bindings. We used the local bindings mode of jTSS. Local Bindings requires proper rights to TPM device file /dev/tpm0 before execution of any program.

```
sudo chmod 777 /dev/tpm0
```

TrouSers must also be stopped if running by using the following command.

```
sudo /etc/init.d/trousers stop
```

Following are few sample programs that utilize TPM functions.

a) Creating Context Object:

Interacting with TPM requires context object. First step to write a program is to create a context object. Following program creates context object and then displays true if it is connected. Exceptions must be caught while interacting with TPM.

```
import iaik.tc.tss.api.tspi.TcIContext;
import iaik.tc.tss.impl.java.tsp.TcTssLocalCallFactory;
import iaik.tc.tss.api.exceptions.common.TcTssException;
```

Figure 5-5 Few of Headers Required for Accessing TPM

```
import iaik.tc.tss.api.tspi.TcIContext;
import iaik.tc.tss.impl.java.tsp.TcTssLocalCallFactory;
import iaik.tc.tss.api.exceptions.common.TcTssException;
public class RT {
public static void main(String[] args) {
  try{
TcIContext context = new
TcTssLocalCallFactory().newContextObject();
        context.connect();
        System.out.println(context.isConnected());
```

Figure 5-6 Creating Context Object

b) Taking Ownership of TPM

Context Object is further used to create Tpm Object to access TPM and for performing operations on TPM. Following Program will set the Owner for TPM. If ownership is already resumed by any other tool clear it from BIOS and then execute this program.

During the process of taking ownership Owner enters the shared secret (Owner password) and a Storage Root Key is generated at the time of taking ownership.

```
import iaik.tc.tss.impl.java.tsp.TcTssLocalCallFactory;
import iaik.tc.tss.api.exceptions.common.TcTssException;
import iaik.tc.tss.api.tspi.TcITpm;
public class TakeOwn {
public static void main(String[] args) {
      try{
      TcIContext context = new
      TcTssLocalCallFactory().newContextObject();
      context.connect();
      System.out.println(context.isConnected());
//Creating TPM Object
      TcITpm tpm = context.getTpmObject();
//Creating SRK object because the takeOwnship method will take it as an argument.
      TcIRsaKey srk =
      context.createRsaKeyObject(TcTssConstants.TSS_KEY_SIZE_DEFAULT
      |TcTssConstants.TSS_KEY_TYPE_STORAGE);
// Taking Ownership: In second argument Public Endorsement Key (EK) can be passed but if the his argument
//is set as null the method will query TPM for Public EK.
      tpm.takeOwnership(srk, null); }
      catch (TcTssException e) {
      e.printStackTrace (); }}}
```

Figure 5-7 Taking TPM Ownership

c) Printing Public Part of Endorsement Key

```
public class ViewPEK {
public static void main(String[] args) {
try{
TcIContext context = new TcTssLocalCallFactory().newContextObject();
      context.connect();
      System.out.println(context.isConnected());
//Creating TPM Object
      TcITpm tpm = context.getTpmObject();
//Create Key Object to store Public EK
            TcIRsaKey pek =
            context.createRsaKeyObject(TcTssConstants.TSS_OBJECT_TYPE_RSA
            KEY);
// Get Public EK and store in Key Object.
                  pek = tpm.getPubEndorsementKeyOwner();
                  System.out.println(trkp.getPubKey().toHexString());
}
       catch (TcTssException e)
      {
      e.printStackTrace();
      }
}
}
```

Figure 5-8 Printing Public Part of Endorsement Key

d) Creating Attestation Identity Key

Private Part of TPM Endorsement Key is used to sign data, which can be verified by remote parties using Public part of EK. It is unique for each TPM chip and creates privacy issues. Alternative to this problem is to create another key known as AIK for signing data. Figure 5-9 illustrates the code for creating AIK.

```
// Create Storage Root Key Object
      TcIRsaKey srk =
      context.createRsaKeyObject(TcTssConstants.TSS_KEY_TSP_SRK);
// create policy for Storge root key : None or Different kinds of  policies can be set.
      TcIPolicy srkpolicy =
      context.createPolicyObject(TcTssConstants.TSS_POLICY_USAGE);
// assign policy to SRK
      srkpolicy.assignToObject(srk);
// Create Attestation Identity Key Object
      TcIRsaKey aik =
      context.createRsaKeyObject(TcTssConstants.TSS_KEY_SIZE_2048 |
            TcTssConstants.TSS_KEY_MIGRATABLE
                  |TcTssConstants.TSS_KEY_TYPE_SIGNING);
//create key usage policy: This policy will ask for a set secret (here 123) when using the AIK.
      TcIPolicy keyusagepolicy =
      context.createPolicyObject(TcTssConstants.TSS_POLICY_USAGE);
      TcBlobData keyUsageSecret = TcBlobData.newString("123");
      keyUsagePolicy.setSecret(TcTssConstants.TSS_SECRET_MODE_PLAIN,
      keyUsageSecret);
//create key Migration Policy: This policy will prompt for password (here 456) when migrating the
//AIK object.
      TcIPolicy keymigrationpolicy=
      context.createPolicyObject(TcTssConstants.TSS_POLICY_MIGRATION);
      TcBlobData keyMigrationSecret = TcBlobData.newString("456");
      keyMigPolicy.setSecret(TcTssConstants.TSS_SECRET_MODE_PLAIN,
      keyMigrationSecret);
//Creating & Loading Key in TPM
      aik.createKey(srk, null);
      aik.loadKey(srk);
```

Figure 5-9 Creating Attestation Identity Key

Generating Nonce inside the TPM

```
TcBlobData NodeNonce = TcBlobData.newUINT32(123456);
byte[] NonceArray;
NonceArray = new byte[20];
NodeNonce = tpm.getRandom(20);
```

Figure 5-10 Nonce Generation in the TPM

e) TPM Quote Operation

During the remote attestation process, two entities participate a challenger and a attester. The challenger initiates remote attestation process by sending a nonce and asks the attester to prove his integrity. On receiving the nonce, the attester creates quote operation. The quote is an operation/function of the TPM that takes the following arguments.

- Nonce (received form challenger)
- Value of PCR-10
- Attestation Identity Key

Attester prepares the quote object and sends it to the challenger along with its Measurement List (SML). On receipt, challenger verifies signature on the quote operation by using public part of attester's AIK. After successful signature verification challenger matches the received nonce (in the quote operation) with the sent and verifies the attester SML against the quoted PCR-10 value. Quote operation utilizes specialized structures as depicted in the Figure 5-11 and Figure 5-12.

```
//define which pcrs to quote
      TcIPcrComposite pcrs =
      context.createPcrCompositeObject(TcTssConstants.TSS_PCRS_STRU
      CT_INFO_SHORT);
// set PCR Index for quote operation: More than once PCR registers can be set here
      pcrs.selectPcrIndexEx(10,
      TcTssConstants.TSS_PCRS_DIRECTION_RELEASE);
// create TcTssValidation Object for assigning Challenger Nonce
      TcTssValidation NodeNonceVal = new TcTssValidation();
// Assign Nonce to created object
      NodeNonceVal.setExternalData(NodeNonceBlob);
//Do Quote
      tpmquote = tpm.quote(aik, pcrs , NodeNonceVal);
      System.out.println(tpmquote);
```

Figure 5-11 Performing Quote Operation

Output of the quote operation has three parts.

1. External Data-20bytes (Here Nonce)
2. Rgb Data – 48 bytes (20 bytes of Rgb Data contains quoted PCR value ranging from byte 8-28.
3. Rgb Validation Data – 256 bytes (Generated by TPM)

```
// Verifying Signature:
```

1. Create hash object

```
TcIHash hobj =
context.createHashObject(TcTssConstants.TSS_HASH_SHA1);
```

2. Update Hash Object with Rgb Data

```
hobj.updateHashValue(rgbBlob);
```

3. Verify Signature: TcPubK is TcIRsaKey Object which holds the Public Part of Aik of Attester.

```
hobj.verifySignature(rgbVBlob,TcPubK);
```

4. Verifying Signature: if no exception is thrown it means signature is verified.

```
System.out.println("Signature Verified");
```

rgbBlob (Rgb data read in the TcBlobData of TPM)

TcPubK (Attester's AIK public part)

Figure 5-12 Verifying Signature on the Quote Operation

5.2 Image Compression and Encryption Programs

The image compression (workload) programs for android and the cloud node were written using java built-in functions. Similarly, we did coding of all operations involved in the proposed scenarios independently and then calculated the measurement time of each operation by using timers, repeated each experiment thrice and measured execution times. Coming figures provide java code of Image compression and Image encryption.

```
long starttime = System.currentTimeMillis();
String filepath = "/data/data/";
File imagefile = new File(filepath + "1.jpg");
FileInputStream fis;
fis = new FileInputStream(imagefile);
Bitmap bi = BitmapFactory.decodeStream(fis);
boolean imgc = false;
String extStorageDirectory =
Environment.getExternalStorageDirectory().toString();
File file = new File(extStorageDirectory+"/35Comp.jpg");
FileOutputStream outStream = new FileOutputStream(file);
imgc=bi.compress(Bitmap.CompressFormat.JPEG, 10outStream);
long endtime = System.currentTimeMillis();long executiontime=endtime-
starttime;
```

Figure 5-13 Image Compression Program for Android

```
long starttime1 = System.currentTimeMillis();
String path = "/home/luqman/Desktop/Results/TestImages/3.jpg";
BufferedImage src = ImageIO.read(new File(path));
System.out.println("whatsfsdfsdf");
Iterator<ImageWriter> i ImageIO.getImageWritersByFormatName("jpeg");
ImageWriter jpegWriter = i.next();
ImageWriteParam param = jpegWriter.getDefaultWriteParam();
param.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);
param.setCompressionQuality(0.1f);
FileImageOutputStream out = new FileImageOutputStream new
File("/home/luqman/Desktop/ImgComNod/3Comp.jpg"));
jpegWriter.setOutput(out);
jpegWriter.write(null, new IIOImage(src, null, null), param);
jpegWriter.dispose();
out.close();
long stoptime = System.currentTimeMillis();
long executiontime = stoptime-starttime1;
System.out.println(executiontime);
```

Figure 5-14 Image Compression Program for Cloud Node

```
String filepath = "/data/data/";
File imagefile = new File(filepath + "20Comp.jpg");
byte[] b = new byte[(int) imagefile.length()];
FileInputStream fis;
fis = new FileInputStream(imagefile);
fis.read(b);

//Key Setup
long starttime1 = System.currentTimeMillis();
byte[] password = "password".getBytes("UTF-8")
MessageDigest digest;
digest = MessageDigest.getInstance("MD5");
byte[] hash = digest.digest(password);
long endtime1 = System.currentTimeMillis();
long executiontime1=endtime1-starttime1;
//Encryption RC4
long starttime2 = System.currentTimeMillis();
Cipher rc4Enc;
rc4Enc = Cipher.getInstance("RC4");
rc4Enc.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(hash, "RC4"));
byte[] ctext = rc4Enc.doFinal(b);
long endtime2 = System.currentTimeMillis();
long executiontime2=endtime2-starttime2;
long EncTime=executiontime1+executiontime2
TextView tv1 = new TextView(this);
tv1.setText("Encryption Time” +EncTime+");
setContentView(tv1);
```

Figure 5-15 RC4 Image Encryption Program

```
//Key Setup
long starttime1 = System.currentTimeMillis();
byte[] password = "password".getBytes("UTF-8")
MessageDigest digest;
digest = MessageDigest.getInstance("MD5");
byte[] hash = digest.digest(password);
long endtime1 = System.currentTimeMillis();
long executiontime1=endtime1-starttime1;
long starttime3 = System.currentTimeMillis();
Cipher rc4Dec;
rc4Dec = Cipher.getInstance("RC4");
rc4Dec.init(Cipher.DECRYPT_MODE, new SecretKeySpec(hash, "RC4"));
byte[] dectext = rc4Dec.doFinal(ctext);
long endtime3 = System.currentTimeMillis();
long executiontime3=endtime3-starttime3;
long DecTime=executiontime1+executiontime3;

 TextView tv1 = new TextView(this);
 tv1.setText("Decrypition TIme      "+ DecTime);
 setContentView(tv1);
```

Figure 5-16 RC4 Image Decryption Program

```
long st2 = System.currentTimeMillis();
java.io.InputStream is = ClientSocket.getInputStream();
DataInputStream dis = new DataInputStream(is);
int reclen = dis.readInt();
System.out.println ("Received Image Length="+reclen);
byte[] RecImg = new byte[reclen];
int i=0;
while (i<=RecImg.length-1460){
ClientSocket.getInputStream().read(RecImg, i, 1460);
i=i+1460;
}
int jj = i;
for(jj=i;jj<=RecImg.length-1;jj++)
{
ClientSocket.getInputStream().read(RecImg, jj, 1);
}

long et2 = System.currentTimeMillis();
long ex2 = et2-st2;
```

Figure 5-17 Image Receiving Program

```
long st7 = System.currentTimeMillis();
java.io.OutputStream os = ClientSocket.getOutputStream();
DataOutputStream dos = new DataOutputStream(os);
dos.writeInt(b.length);
System.out.println ("Compressed Image Length="+b.length);
int ii=0;
while (ii<=b.length-1460){
ClientSocket.getOutputStream().write(b, ii, 1460);
ii=ii+1460;
}
int jc = ii;
for(jc=ii;jc<=b.length-1;jc++)
{
ClientSocket.getOutputStream().write(b, jc, 1);
}

long et7 = System.currentTimeMillis();
long ex7 = et7-st7;
long endtime = System.currentTimeMillis();
```

Figure 5-18 Image Sending Program

5.3 Interpolation and Threshold

Interpolation is used to estimate the values of f(x) in the given range of the values of x. In this section, we present a way to estimate the execution time for any size of images for a given scenario using Newton Forward Interpolation. Basic idea is to provide estimated execution time of an image size provided by users for all scenarios. Based on estimated execution times users can decide any scenario matching their requirements.

5.3.1 Estimation of Overall Execution Time using Interpolation

First step involves the construction of interpolation table. We take four image sizes (which we have used in experimentation) 244KB, 1.1MB, 2.1 MB and 3 MB for the construction of interpolation table as shown in Table 4-1.

IS	ET(Si)	First Difference	Second Difference	Third Difference
244	ET_1: $ET(Si)_{244Kb}$			
		D_{11}: $(ET(Si)_{1100Kb} - ET(Si)_{244Kb})/(1100-244)$		
1100	ET_2: $ET(Si)_{1100Kb}$		$D:_{21}$ $(D_{12} - D_{11})/(2100-244)$	
		D_{12}:$(ET(Si)_{2100Kb} - ET(Si)_{1100Kb})/(2100-1100)$		D_{31}:$(D_{22}.D_{21})/(3000-244)$
2100	ET_3: $ET(Si)_{2100Kb}$		$D_{22:}$ $(D_{13} - D_{12})/(3000-1100)$	
		D_{13}: $(ET(Si)_{3000Kb} - ET(Si)_{2100Kb})/(3000-2100)$		
3000	ET_4: $ET(Si)_{3000Kb}$			

Table 5-1 Interpolation Table

Where IS: Image size in KB

ET(Si): Execution time of image in scenario-i

$ET(Si)_{244Kb:}$ Execution time of 244 KB image in Scenario-i

$ET(Si)_{1100Kb:}$ Execution time of 1100KB image in Scenario-i and so on

Users can estimate the overall execution time for a given size of image by substituting values in Table 5-1 and then in equation 5-1.

$$ET_{estimated} = ET_1 + (IS_{user} - 244)*D_{11} + (IS_{user} - 244)*(IS_{user} - 1100)*D_{21} + (IS_{user} - 244)*(IS_{user} - 244)*(IS_{user} - 2100)*D_{31} \quad \text{(Equation 5-1)}$$

Where $ET_{estimated}$ is estimated execution time of Image Size IS_{user} provided by user.

5.3.2 Estimation of Execution Time on Mobile and Threshold

While offloading data to the cloud the execution time required to offload data has significant importance due to constrained capabilities of hand-held devices. Offloading is beneficial only when the execution time required to offload should not exceed from on-device computation. Mathematically,

$$ET_{offload} < ET_{on\text{-}device}$$

Where $ET_{offload}$ is Execution Time required to offload data and $ET_{on\text{-}device}$ is Execution Time required to perform on device computation. Using the Table 4-1 and Equation 4-1 users can calculate the respective execution times and then can decide which is the best scenario for performing computation.

CHAPTER 6

Results and Evaluation

6.1 Results

This section provides the detailed implementation results of each scenario.

6.1.1 Data Computation on the Mobile Device (Scenario-1)

This is the first benchmark of our work. For performance analysis of this scenario, we developed android image compression application and tested different images sizes both on the Android Emulator and on real Android Device. We used Android Emulator and Android Nexus One phone and recorded the execution time for each image. Each experiment was performed thrice. Table 6-1 reveals average results of Scenario-1.

Table 6-1 Execution Time of Scenario-1

Image Size	Execution Time (ms)	
	On Android Emulator	On Real Device
244 KB	3356	503
1.1 MB	4392	737
2.1 MB	26091	2934
3 MB	72800	Error
4.6 MB	Error	Error
9.7 MB	Error	Error

Experiments show that compressing 4.6MB image throw error and compressing 3MB image took too much on android emulator whereas, for real android phone compressing 3MB image throw error.

6.1.2 Data Computation on the Cloud Node (Scenario-2)

In Scenario-2, mobile device offloads images for compression to the Cloud Node. Cloud Node performs image compression on the behalf of mobile device. In this scenario, participating entities are Cloud Node (a Dual Core Machine with Ubuntu 9.04) and a Android device. For the performance analysis of this scenario, we developed a Java image compression application running on the Cloud node, which performed image

compression for Android device. Table 6-2 describes execution time of image compression of scenario-2.

Table 6-2 Overall Execution Time of Scenario-2 using Android Emulator

Image Size	Execution Time (ms)						
	Reading Image from Local Disk (Android Emulator)	Socket Creation (Android Emulator)	Sending Image to Node (Android Emulator)	Image Compression on Node (Node)	Receiving Compressed Image from Node (Android Emulator)	Writing Compressed Image to Local Disk (Android Emulator)	Total
244 KB	214	15	1522	287	443	46	2527
1.1 MB	1284	30	2060	414	236	34	4058
2.1 MB	1164	61	3859	1509	354	51	6998
3 MB	1155	17	3790	2941	560	88	8551
4.6 MB	1362	5	4254	2621	739	147	9128
9.7 MB	1433	16	4739	4544	313	86	11131

**Entries in brackets () represent the entity on which the operation was performed.*

6.1.3 Data Computation on the Cloud Node with Encryption (Scenario-3)

In this scenario, mobile device offloads encrypted image to the Cloud for compression**.** For implementation, we did breakdown of the operations involved in this protocol as shown in Table 6-3 and implemented each operation independently. We took RC4 encryption algorithm and used Java libraries for implementation. We summed up the execution times of individual operations to obtain the overall execution time of a scenario for a given image size. Table 6-3 depicts the results of each operation and then accumulative results using Android Emulator whereas Table 6-4 reveals results on the real Android Phone. For Emulator, we calculated times to send/receive images to node and read/write operation of local disk as depicted whereas, for real phone as shown is Table 6-4 does not include send/receive and read/write operations.

Table 6-3 Overall Execution Time of Scenario-3 Using Android Emulator

Image Size	Execution Time (ms)										
	Reading Image from Local Disk (Android Emulator)	Encrypting Image (Android Emulator)	Socket Creation (Android Emulator)	Sending Encrypted Image to Node (Android Emulator)	Decrypting Received Image (Node)	Compressing Received Image (Node)	Encrypting Compressed Image (Node)	Receiving Encrypted Image (Android Emulator)	Decrypting Compressed Image (Android Emulator)	Writing Compressed Image to Local Disk (Android Emulator)	Total
244 KB	214	1738	15	1522	48	287	37	443	181	46	4531
1.1 MB	1284	3126	30	2060	55	414	45	236	420	34	7704
2.1 MB	1164	3229	61	3859	64	1509	48	354	699	51	11038
3 MB	1155	3661	17	3790	64	2941	48	560	821	88	13145
4.6 MB	1362	5208	5	4254	80	2621	47	739	1480	147	15943
9.7 MB	1433	11838	16	4739	118	4544	55	313	2609	86	25751

**Entries in brackets () represent the entity on which the operation was performed.*

Table 6-4 Overall Execution Time of Scenario-3 Using Android Real Phone

Image Size	Execution Time (ms)					
	RC4 Encryption on Android Phone (Image)	RC4 Decryption on Cloud Node	Image Compression On Cloud Node	RC4 Encryption on Cloud Node	RC4 Decryption on Android Phone (Compressed Image)	Total
244 KB	149	48	287	37	49	570
1.1 MB	214	55	414	45	84	812
2.1 MB	434	64	1509	48	202	2257
3 MB	621	64	2941	48	376	4050
4.6 MB	835	80	2621	47	391	3974
9.7 MB	1498	118	4544	55	513	6728

6.1.4 Data Computation on the Cloud Node with Encryption and Remote Attestation (Scenario-4)

Scenario-4 introduces both Remote Attestation and Encryption. Participating Entities are Trusted Coordinator (TC), a Cloud Node and Android emulator/phone. This scenario has two parts a) Remote Attestation between the Cloud Node and TC b) Data Computation on the Cloud. In the coming sub-sections we describe results of remote attestation between the Cloud Node and then provide overall execution time of Scenario-4.

6.1.4.1 Remote Attestation between the Cloud Node and the TC

We did breakdown of the Remote Attestation protocol and implemented all operations involved in Remote Attestation independently on the TPM enabled Core to Duo PC installed with Ubuntu 9.04. Table 6-5 provides breakdown and operation-wise execution times while Table 6-6 provides step-wise execution time of the protocol.

Table 6-5 Operation-wise Execution Time of Remote Attestation Protocol

S. No	Operation	Execution Time (ms)
1	Context and TPM object creation for interaction with TPM.	170.2
2	Nonce Generation	173.5
3	Attestation Identity Key Creation (Prompts Owner Password for two time)	21951
4	Attestation Identity Key loading	2636
5	Doing Quote Operation	903
6	Signature Verification on Quote Operation	21
7	Nonce Verification Quote Operation	0
8	Measurement List (Assumed Reference PCR) comparison with Quoted Operation	71
9	Reading Measurement list from kernel files	33.6
10	Calculating Hash of Measurement list	171.9
11	Trusted Key Pair Generation Time (RSA Java Key of 2048-bit)	1144.5
12	Encryption with Trusted Key (Public Part)	448.7
13	Decryption with Trusted Key (Private Part)	35.2

Table 6-6 Step-wise Execution time (ms) of Remote Attestation Protocol

Step 1	1*									173.5
	173.5									
Step 2	4*	5*	1*	9*						3746.1
	2636	903	173.5	33.6						
Step 3	6*	7*	8*	4*	5*	11*	9*	12*	10*	4399.7
	21	0	71	2636	903	1144.5	33.6	448.7	171.9	
Step 4	6*	7*	8*	10*	13*	12*				747.8
	21	0	71	171.9	35.2	448.7				
Total										**9067.1**

*Numbers with * refer to S. No of Table 5-4*

Step-wise execution time of Remote Attestation Protocol is calculated by adding the execution time of operations involved in the respective step. In the Table 6-5, we included the execution time of the Attestation Identity Key Loading only and assume that both the Cloud Node and the TC has already generated their AIKs, and they just load it during Remote Attestation process. We obtain the overall execution time of remote attestation protocol by summing the execution times of individual operations. Network delays, latencies and file read/write overheads are not included in overall execution time.

6.1.4.2 Overall Execution Time of Scenario-4

Table 6-7 provides total execution time Scenario-4 for Android Emulator whereas Table 6-8 provides for Android real phone. We are calculating the overall execution time, which include execution time of Remote Attestation between Cloud Node and TC. The operations performed by each participating entity are indicated in the respective tables. For real android phone, we are not addressing the times required to read/write and time required to send image on the network.

Table 6-7 Overall Execution Time of Scenario-4 Using Android Emulator

Image Size	Total Execution Time of Scenario 4																					
	Execution Time (ms)																					
	Remote Attestation	RC4 Key + Nonce Encryption with TK_{public}^{TC}	Reading Image from Local Disk	Image Encryption using RC4	Sending Image to Node	SHA-1 Calculation	Signing Hash	Signature Verification	SHA-1 Verification	RC Key and Nonce Decryption	Encrypting Key	SHA-1 Calculation	Signing SH1 Hash	Signature Verification	SHA-1 Verification	Decrypting Key	Image Decryption	Image Compression	Encryption Compressed Image	Receiving Compressed Image	Decryption Compressed Image	Writing Compressed Image to Local Disk	Total
	N&T*	A*	A*	A*	A*	N*	N*	TC*	TC*	TC*	TC*	TC*	TC*	N*	N*	N*	N*	N*	N*	A*	A*	A*	
244Kb	9067.1	11	214	1738	1522	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	48	287	37	443	181	46	13818
1.1 Mb	9067.1	11	1284	3126	2060	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	55	414	45	236	420	34	16976
2.1 Mb	9067.1	11	1164	3229	3859	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	64	1509	48	354	699	51	20279
3 Mb	9067.1	11	1155	3661	3790	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	64	2941	48	560	821	88	22430
4.6 Mb	9067.1	11	1362	5208	4254	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	80	2621	47	739	1480	147	25240
9.7 Mb	9067.1	11	1433	11838	4739	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	118	4544	55	313	2609	86	35037

*Entries having * indicate the entities on which the given operation is performed. N&T indicates Remote Attestation between Cloud Node and TC. And Indicates Android Emulator and Nod Indicates Cloud Node.*

Total includes 19millisecond s for socket creation with every value.

Table 6-8 Overall Execution Time of Scenario-4 using Real Android Phone

Total Execution Time of Scenario 4																			
Image Size	Execution Time (ms)																		
	Remote attestation	RC4 Key + Nonce Encryption with TK_{public}^{TC}	Image Encryption RC4	SHA-1 Calculation	Signing Hash	Signature Verification	SHA-1 Verification	RC Key and Nonce Decryption	Encrypting Key	SHA-1 Calculation	Signing SH1 Hash	Signature Verification	SHA-1 Verification	Key Decryption	Image Decryption	Image Compression	Encryption Compressed Image	Decryption Compressed Image	Total
	N&T*	And*	And*	Nod*	Nod*	TC*	TC*	TC*	TC*	TC*	TC*	Nod*	Nod*	Nod*	Nod*	Nod*	Nod*	And*	
244Kb	9067.1		149	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	48	287	37	49	9842
1.1 Mb	9067.1		214	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	55	414	45	84	10084
2.1 Mb	9067.1		434	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	64	1509	48	202	11529
3 Mb	9067.1		621	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	64	2941	48	376	13322
4.6 Mb	9067.1		835	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	80	2621	47	391	13246
9.7 Mb	9067.1		1498	0.6	50	0.6	0.6	46	9.33	0.6	50	0.6	0.6	46	118	4544	55	513	16000

*Entries having * indicate the participating entities. N&T indicates Remote Attestation between Cloud Node and TC. And Indicates Android Real Phone and Nod Indicated cloud Node. RC4 Key here is just a string and time required to encrypt can be negligible.*

6.1.5 Data Computation on the Cloud Node with Encryption and Two-way Remote Attestation (Scenario-5)

In the Scenario-5, TC attests Cloud Node and Cloud also attests the Mobile Device. Table 6-9 reveals the breakdown and results of the operations involved in this scenario. We performed remote attestation operations on a TPM enabled Dual Core machine running Ubuntu 9.04. We assume that the Android emulator in this scenario is equipped with TPM emulator. We also assume the execution time of TPM related operations on Android Emulator to those of the TPM operations on computers. In total execution time of this scenario, the Remote Attestation time between Cloud Node and TC is also included. We take AIK generation time of the android device as that of Java RSA key generation time and include the time in total execution time of this scenario.

Table 6-9 Overall Execution Time of Scenario-5 using Android Emulator

Image Size (MB)	Execution Time (ms)																								
	Remote Attestation	Nonce Generation (TPM Op)	AIK Generation (TPM Op)	RC4 Key + Nonce Encryption with TK_{public}^{TC}	Reading Image from Local Disk	Image Encryption with RC4 key	Sending Image to Node	Signing Message with $TK_{private}^{Node}$	Encrypting Message with TK_{public}	Decryption of message sent by Node	Signature verification of message sent by node	Decryption of message (encrypted by And)	Encrypting message for node	Signing message for node	Signature Verification	Decryption of message	Encrypting with Symmetric key of Android	Decrypting message sent by node	Quote Operation (TPM Op)	Verifying Quote Operation (TPM Op)	Image Decryption received from And	Image Compression	Compressed Image Encryption	Receiving Compressed Image	Compressed Image Decryption	Writing Compressed Image to Local File	Total
	N&T*	A*	A*	A*	A*	A*	A*	N*	N*	TC*	TC*	TC*	TC*	TC*	N*	N*	N*	A*	A*	N*	N*	N*	N*	N*	A*	A*	A*	
244 K	9067.1	1.6	7162	11	214	1738	1522	404	3	168	3	156	3	387	3	175	2	4	903	92	48	287	37	443	181	46	23061
1.1 M	9067.1	1.6	7162	11	1284	3126	2060	404	3	168	3	156	3	387	3	175	2	4	903	92	55	414	45	236	420	34	26219
2.1 M	9067.1	1.6	7162	11	1164	3229	3859	404	3	168	3	156	3	387	3	175	2	4	903	92	64	1509	48	354	699	51	29522
3 M	9067.1	1.6	7162	11	1155	3661	3790	404	3	168	3	156	3	387	3	175	2	4	903	92	64	2941	48	560	821	88	31673
4.6 M	9067.1	1.6	7162	11	1362	5208	4254	404	3	168	3	156	3	387	3	175	2	4	903	92	80	2621	47	739	1480	147	34483
9 .7 M	9067.1	1.6	7162	11	1433	11838	4739	404	3	168	3	156	3	387	3	175	2	4	903	92	118	4544	55	313	2609	86	44280

*Entries having * represent Participating entities. N&T represents N Attestation between Cloud Node and TC, N, A and TC represent Cloud Node, Trusted Coordinator and Android Emulator respectively. TPM Op represents TPM Operations on Android Emulator that we are assuming.*

6.2 Discussion

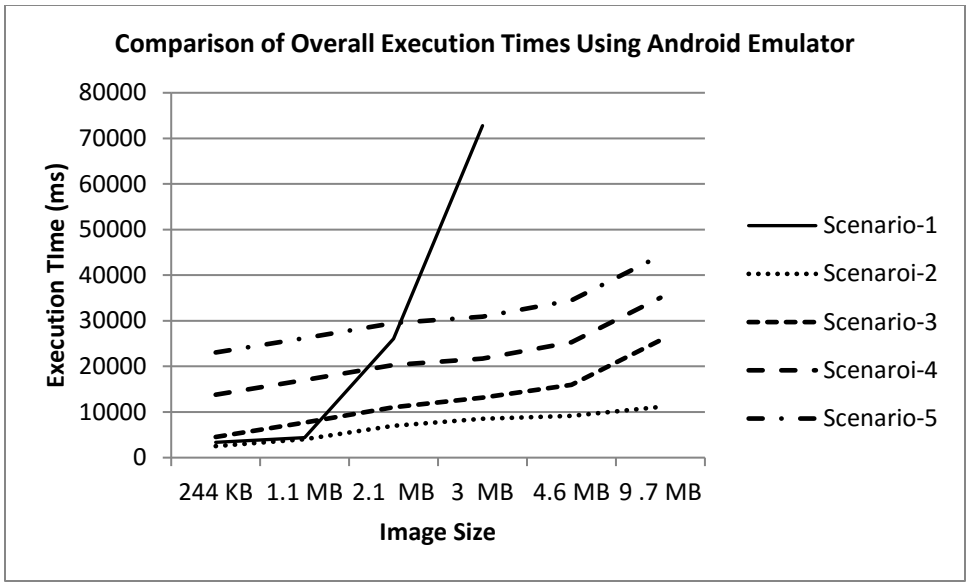
This section provides result comparisons of all scenarios. First perspective of comparison is the total execution time of each scenario and other resource utilization in terms of execution time on the mobile device for each scenario. Moreover, we analyze different TPM operations with respect to execution time.

6.2.1 Overall Execution Time of Each Scenario

We calculated the overall execution time of each scenario by summing up the individual operations that are performed at each participating entity. Remote Attestation times between TC and Cloud Node (Scenarios-4 and Scenario-5) are included here. Results of using both Android emulator and Android real phone are presented separately in Table 6-10 and Table 6-11 respectively.

Table 6-10 Overall Execution Times of All Scenarios using Android Emulator

Image Size	Execution Time (ms)				
	Scenario-1	Scenario-2	Scenario-3	Scenario-4	Scenario-5
244 KB	3356	2527	4531	13818	23061
1.1 MB	4392	4058	7704	16976	26219
2.1 MB	26091	6998	11038	20279	29522
3 MB	72800	8551	13906	22430	31673
4.6 MB	Error	9128	15943	25240	34483
9 .7 MB	Error	11131	25751	35037	44280

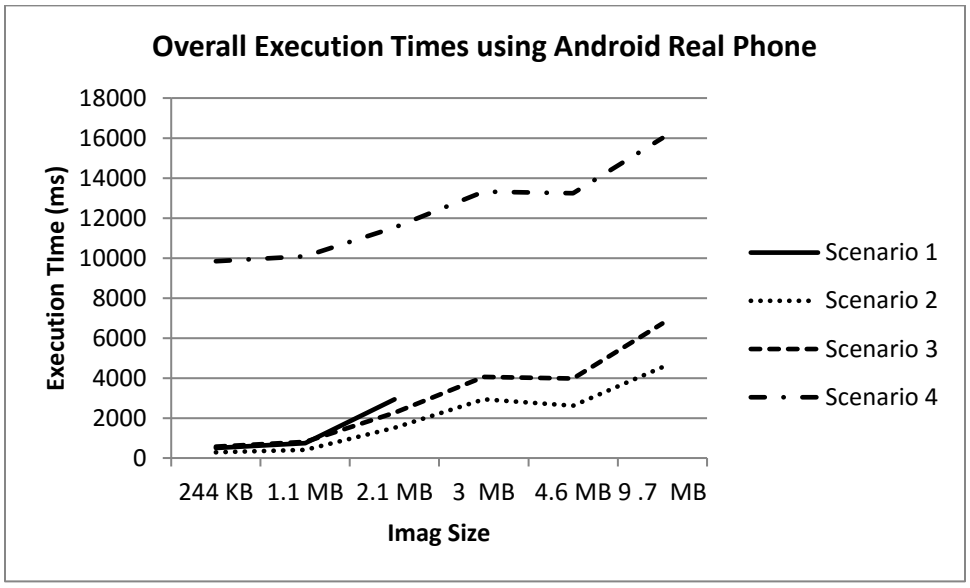


For android emulator, offloading 244KB image in Scenario-2 reduces overall execution time 25% and offloading the same in Scenario-3, Scenario-4 and Scenario-5 increases overall execution time 1.3, 4.1 and 6.8 times respectively. However, offloading 3MB image reduces overall execution time to 89%, 82%, 69%, and 57% in Scenrios-2, 3, 4 and 5 respectively. Results show that offloading small size image like 244KB in Scenarios-3,4 and 5 significantly increases the overall execution time as compared to local processing but on the other hand offloading large sized images like 3MB reduces over-all execution time in all scenarios.

Table 6-11 Overall Execution Times of All Scenarios using Android Real Phone

Image Size	Execution Time (ms)			
	Scenario 1	Scenario 2	Scenario 3	Scenario 4
244 KB	503	287	570	9842
1.1 MB	737	414	812	10084
2.1 MB	2934	1509	2257	11529
3 MB	Error	2941	4050	13322
4.6 MB	Error	2621	3974	13246
9 .7 MB	Error	4544	6728	16000

For Real android phone we performed experiments for scenarios 1,2,3 and 4 and File read/write and network times are not included.



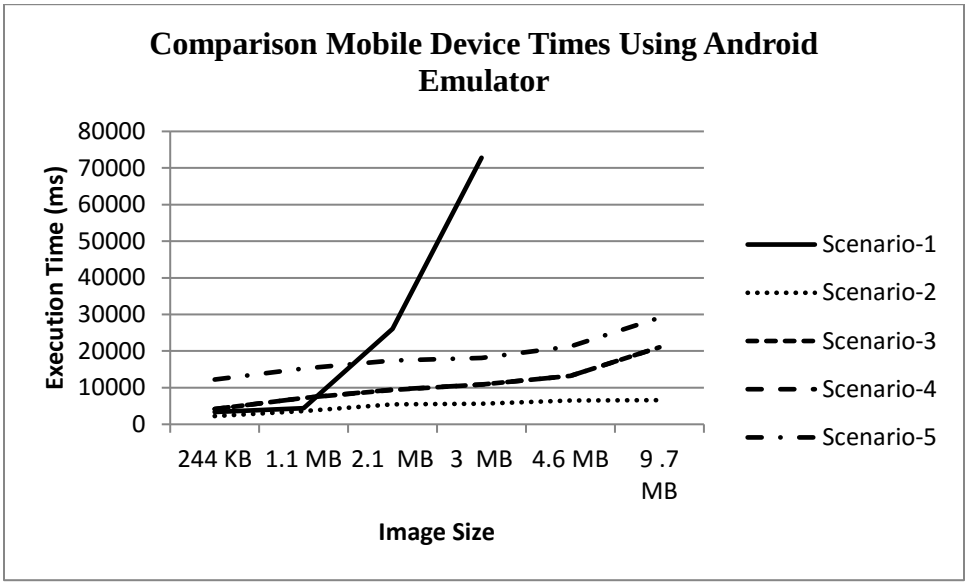
Using Android Nexus One Phone, offloading 2.1MB to the Cloud reduces overall execution time to 49% and 23 % in Scenario-2 and Scenario-3 respectively, while overall execution time increases 4 times using Scenrio-4.

6.2.2 Mobile Device Resource Utilization

Second perspective of result comparison is to analyze the resource utilization of mobile device in terms of execution time. Table 6-12 and Table 6-13 show the execution time on Android emulator and Android real phone respectively for each scenario. Graphical representations are also given in the respective graphs.

Table 6-12 Mobile Device Resource Utilization using Android Emulator

Image Size	Execution Time (ms)				
	Scenario-1	Scenario-2	Scenario-3	Scenario-4	Scenario-5
244 KB	3356	2240	4159	4174	12226
1.1 MB	4392	3644	7190	7190	15242
2.1 MB	26091	5489	9417	9386	17438
3 MB	72800	5610	10092	10105	18157
4.6 MB	Error	6507	13195	13220	21272
9 .7 MB	Error	6587	21034	21048	29100

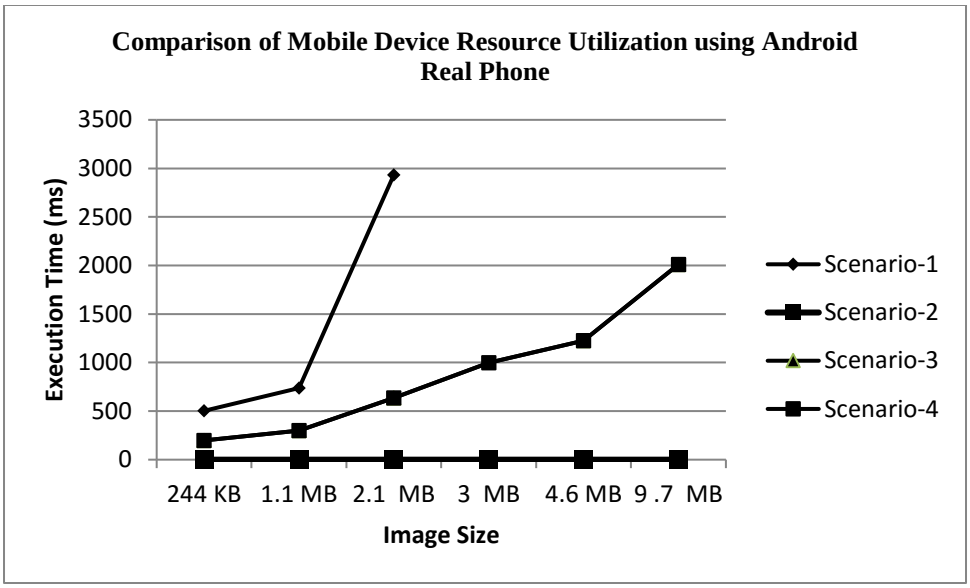


We measure the mobile device resource utilization with respect to execution times. Offloading 244KB image using Android Emulator in Scenario-2 saves the mobile device resources 34% while offloading the same (244 KB image) increases mobile device resources utilization up to 20% in Scenario-3 and Scenario-4 and increases 73% for Scenario-5. Since, offloading small size image like 244KB increases mobile device resource utilization in Scenarios-3, 4 and 5 but offloading the large image 3Mb saved mobile device resources up to 87% for Scenarios 3 and 4 and 76% for Scnenario-5.

Offloading either in Scenario-3 or in Scenario-4 does not make much difference on mobile device resource utilization.

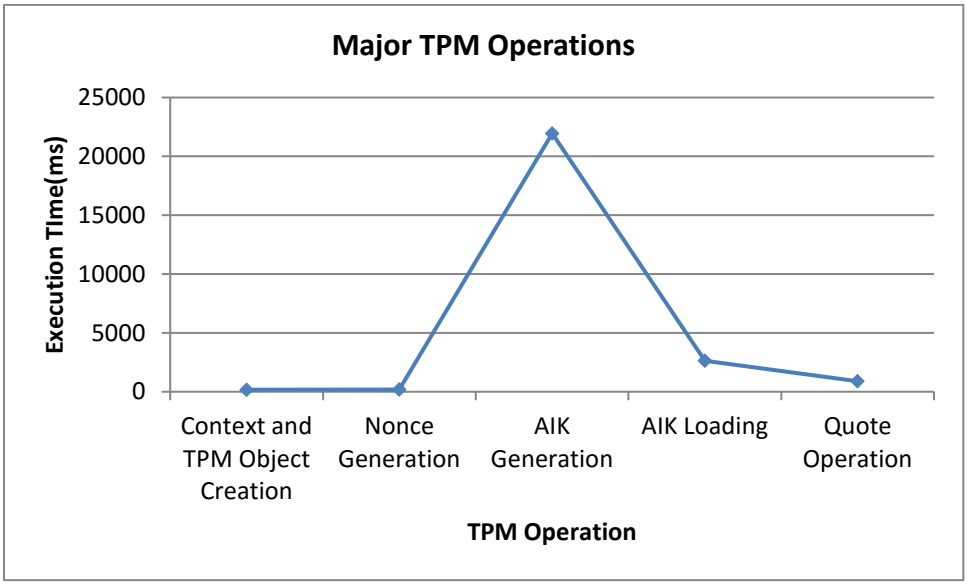
Table 6-13 Mobile Device Resource Utilization using Android Real Phone

Image Size	Execution Time (ms)			
	Scenario-1	Scenario-2	Scenario-3	Scenario-4
244 KB	503	0	198	198
1.1 MB	737	0	298	298
2.1 MB	2934	0	636	636
3 MB	Error	0	997	997
4.6 MB	Error	0	1226	1226
9 .7 MB	Error	0	2011	2011



For Android Nexus One phone, offloading 2.1MB image saves mobile device resources 100% in Scenario-2(Since we are not addressing Network and Read/write times) and 79% in Scenario-3 and Scenario-4.

6.2.3 Comparison of Major TPM Operations



We have implemented major TPM operations involved in remote attestation protocol and results show that AIK generation is the most time consuming operation. Execution time of other operations like Context and TPM Objects creation and Nonce Generation are quite negligible.

CHAPTER 7

Conclusions and Future Recommendation

7.1 Conclusions

Cloud Computing has emerged as an attractive platform for mobile devices to offload their computations to the cloud and hand-held devices with internet access are looking towards the cloud for resource intensive tasks. However, like traditional desktop clients mobile devices are also facing privacy, confidentiality and integrity issues while performing computations or moving data to or from the cloud.

In this thesis, we analyze the performance of different proposed scenarios. We analyzed two aspects of the performance with respect to execution time. The first aspect is overall execution time of a given scenario and second the mobile device resource utilization in terms of execution time. In the overall execution time we are taking the case where the remote attestation time between cloud node and TC are included. This time (Remote Attestation) may be excluded if TC has already attested the cloud node. The image size has significant impact on the performance while offloading data. Results show that for small size images performing on-device computation is more efficient as compared to offloading in a secure scenario. However offloading large size images reduces the overall execution time as compared to on-device computation.

Second aspect of our analysis is mobile device resource utilization that is of much importance due to limited battery and power processing capabilities of hand-held devices. In this thesis, we calculated the resource utilization of the mobile device with respect to the execution times that a mobile device consumes for different operations involved in a particular scenario. Time required to offload data and secure the data in terms of encryption should be taken into account while offloading. Offloading is beneficial only when these times should not exceed from that of on-device computation. Sending large size encrypted images to the cloud for compression saved mobile device resource utilization.

Moreover, we also compared different TPM operations and results show that AIK generation is one of the most resource intensive tasks of the TPM. Summing up

individual operations' time of remote attestation between the cloud node and TC (excluding AIK generation) resulted in overall execution time of about 10 seconds. Calculating execution times of file/read write operations and network overheads may increase the overall execution time of remote attestation to some extent.

The proposed scenario-5 and the operations involved in two-way remote attestation were only implemented on the android emulator. The execution times of the operations involving mobile TPM have been assumed in our implementation due to un-availability of TPM on the android.

For experimentation on real android phone we are not addressing the network and read write overheads.

7.2 Future Recommendations

In this thesis, we implemented operations involved in proposed scenarios independently and analyzed their performance. We recommend complete implementation of each scenario with offloading of virtual machines or applications instead of data. Another interesting area for research would be to write a light-weight Trusted Software Stack like jTSS for android based applications to use TPM emulator functionalities or to make changes in the existing jTSS library to make it compatible with android based applications.

While implementing the operations of remote attestation protocol the cumulative hash of the SML didn't match with the quoted PCR value. This may be an implementation problem and needs to be fixed in the future work.

Bibliography

1. Kelton Research. *Global Survey: Has Cloud Computing Matured?* Kelton Research, 2011.

2. Michael Armbrust, Armando Fox, Rean Griffih, Anthony D. Joseph, R. Katz, A . Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing.* California : RAD Lab, 2009.

3. Amazon. Amazon Elastic Compute Cloud . *Amazaon Cloud Services.* [Online] 2011. [Cited: December 24, 2011.] http://aws.amazon.com/ec2/.

4. Flexiant Ltd. FlexiScale Public Cloud. *Flexiant Ltd.* [Online] 2011. [Cited: December 24, 2011.] http://www.flexiant.com/products/flexiscale/.

5. Go Grid. Cloud Hosting: How GoGrid Works? *Cloud Hosting, Cloud Servers, Hybrid Hosting from GoGrid.* [Online] [Cited: December 24, 2011.] http://www.gogrid.com/cloud-hosting/.

6. IBM. IBM Cloud Computing Overview. *IBM.* [Online] 2011. [Cited: December 24, 2011.] http://www.ibm.com/cloud-computing/us/en/.

7. Reporter, CircleID. Survey: Cloud Computing 'No Hype', But Fear of Security and Control Slowing Adoption. *CircleID.* [Online] [Cited: December 24, 2011.] http://www.circleid.com/posts/20090226_cloud_computing_hype_security/.

8. Trusted Computing Group. Trusted Computing Group. *Trusted Computing Group.* [Online] [Cited: December 24, 2011.] http://www.trustedcomputinggroup.org/.

9. Trusted Platform Module. *Trusted Computing Group.* [Online] [Cited: December 24, 2011.] http://www.trustedcomputinggroup.org/developers/trusted_platform_module.

10. *Towards Trusted Cloud Computing.* Nuno Santos, Krishna P. Gummadi, Rodrigo Rodrigues. San Diego : HotCloud'09, 2009. MPI-SWS, HotCloud '09.

11. *Sealed Storage for Cloud Computing.* Cheng, Ge and Ohoussou, Alex K. Hebei, China : ICCDA, 2010. nternational Conference on Computer Design and Applications.

12. *Clone Cloud: Elastic Execution between Mobile Device and Cloud.* Chun, B.G., et al., et al. Salzburg : EuroSys, 2011. 6th European Conference on Computer Systems (EuroSys 2011).

13. Trusted Computing Group. Mobile. *Trusted Computing Group.* [Online] [Cited: Decebmer 24, 2011.] http://www.trustedcomputinggroup.org/developers/mobile.
14. Strasser, Mario and Stamer, Heiko . Software-based TPM Emulator. *Software-based TPM Emulator.* [Online] 2004. [Cited: December 24, 2011.] http://tpm-emulator.berlios.de/.
15. *Xen and the Art of Virtualization.* Barham, Paul, et al., et al. Bolton Landing, NY USA : SOSP '03, 2003. 19th ACM Symposium on Operating Systems Principals (SOSP '03).
16. Google. Google Apps for Business. *Google Apps for Business.* [Online] [Cited: December 24, 2011.] http://www.google.com/apps/intl/en/business/index.html.
17. *Augmented Smart Phone Applications through Clone Cloud Execution.* B-G, Chun and P, Maniatis. Monte Verità, Switzerland : HotOS'09, 2011. USENIX hot OS XII.
18. Trusted Computing Group. TPM Main Specification. *Trusted Computing Group.* [Online] [Cited: December 24, 2011.] http://www.trustedcomputinggroup.org/resources/tpm_main_specification.
19. Trusted Grub. *TrouSers (The Open Source TCG Software).* [Online] [Cited: December 24, 2011.] http://trousers.sourceforge.net/grub.html.
20. *Attestation based Policy Enforcement for Remote Access.* Sailer, Reiner , et al., et al. Washington, DC, USA : CCS '04 , 2004. 11th ACM Conference on Computer and Communications Security (CCS 2004).
21. *A Robust Integrity Reporting Protocol for Remote Attestation.* Stumpf, Frederic , et al., et al. Tokyo : WATC'06, 2006. Workshop on Advances in Trusted Computing (WATC '06 Fall).
22. IAIK. Trusted Computing for the Java (tm) Platform. *Trusted Computing for the Java (tm) Platform.* [Online] [Cited: December 24, 2011.] http://trustedjava.sourceforge.net/index.php?item=jtss/about.
23. Trusted Computing Group. TrouSers (The Open source TCG Software Stack). *TrouSers.* [Online] [Cited: December 24, 2011.] http://trousers.sourceforge.net/.
24. *A trusted computing environment model in cloud architecture.* Li, Xiao-Yong , et al., et al. Qingdao : Ninth International Conference on Machine Learning and Cybernetics, 2010. Ninth International Conference on Machine Learning and Cybernetics.

25. *Integrity Measurement Model Based on Trusted Virtual Platform.* Qiu, Gang , Want, Yeulei and Zhou, Lihua . Shenzhen, China : IEEE ICGEC-2010, 2010. Fourth International Conference on genetic and Evolutionary Computing.
26. *vTPM: Virtualizing the Trusted Platform Module.* Berger, S, et al., et al. Vancouver B.C., Canada : Security 06, 15th USENIX Security Symposium, 2006. 15th Conference on USENIX Security Symposium (USENIX).
27. *Hey, You, Get off My Cloud: Exploring Information Leakage in Third-Party Compute Clouds.* Ristenpart, Thomas, et al., et al. Chicago : CCS '09, 2009. CCS'09.