2010

A TRUSTED STORAGE SYSTEM FOR THE CLOUD

Sushama Karumanchi
University of Kentucky, ska226@uky.edu

ABSTRACT OF THESIS

A TRUSTED STORAGE SYSTEM FOR THE CLOUD

Data stored in third party storage systems like the cloud might not be secure since confidentiality and integrity of data are not guaranteed. Though cloud computing provides cost-effective storage services, it is a third party service and so, a client cannot trust the cloud service provider to store its data securely within the cloud. Hence, many organizations and users may not be willing to use the cloud services to store their data in the cloud until certain security guarantees are made. In this thesis, a solution to the problem of securely storing the client's data by maintaining the confidentiality and integrity of the data within the cloud is developed. Five protocols are developed which ensure that the client's data is stored only on trusted storage servers, replicated only on trusted storage servers, and guarantee that the data owners and other privileged users of that data access the data securely. The system is based on trusted computing platform technology [11]. It uses a Trusted Platform Module, specified by the Trusted Computing Group [11]. An encrypted file system is used to encrypt the user's data. The system provides data security against a system administrator in the cloud.

KEYWORDS: Trusted Storage, Cloud Computing, Trusted Platform, Encrypted File System, Cloud Storage

Sushama Karumanchi

Summer 2010

A TRUSTED STORAGE SYSTEM FOR THE CLOUD

By

Sushama Karumanchi

Dr. Mukesh Singhal
Director of Thesis

Dr. Raphael A. Finkel
Director of Graduate Studies

July 08, 2010

RULES FOR THE USE OF THESIS

Unpublished dissertations submitted for the Master's and Doctor's degrees and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part requires also the consent of the Dean of the Graduate School of the University of Kentucky.

THESIS

Sushama Karumanchi

The Graduate School

University of Kentucky

2010

A TRUSTED STORAGE SYSTEM FOR THE CLOUD

THESIS

A thesis submitted in partial fulfillment of the
requirements of the degree of Master of Science in the
College of Engineering at the University of Kentucky

By

Sushama Karumanchi

Lexington, Kentucky

Director: Dr. Mukesh Singhal, Professor of Computer Science

Lexington, Kentucky

2010

Dedicated to my mother Srimathi Gayathri Devi

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Dr. Mukesh Singhal, for his constant support and help in making this work possible. I would like to thank him for encouraging me to step into the research field and also for his bright guidance throughout this work. He was always available for regular meetings and discussions which helped me gain fruitful ideas enabling me to produce this thesis. I would also like to thank Dr. Zongming Fei and Dr. Tingjian Ge for taking their time to serve on my committee. Finally, I thank God and my family for their never ending blessings for me.

Table of Contents

List of Figures

Chapter 1: Introduction

In this thesis, a framework/system for trusted storage of a client's data within the cloud is developed. The proposed system is called 'A Trusted Storage System for the Cloud'. As an enormous quantity of electronic data is being generated, there is a requirement of vast storage systems which can hold that data. The requirement is not just storing the data but storing it securely, i.e., the confidentiality and integrity of the data should be maintained. The question of confidentiality and integrity of data comes into the picture when the owner's data is being stored in third party storage systems like the cloud. National Institute of Standards and Technology (NIST) defines cloud computing as follows: "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [10]. Though cloud computing provides cost-effective storage services, it is a third party service and therefore, a user/client cannot trust the cloud service provider to store its data securely within the cloud. Hence, many organizations and users may not be willing to use the cloud services to store their data in the cloud until certain security guarantees are made.

1.1 Problem Statement

Unencrypted data of the client cannot be stored in the cloud because the cloud provider will have access to the data and hence the confidentiality of the data will be lost. Also, a malicious cloud provider (system administrator) can modify the client's data and hence, the integrity of the data will be lost. To achieve confidentiality and integrity of the data, cryptographic techniques can be used to encrypt data. Encrypted file systems (EFS) can be used to encrypt the client's data within the cloud. An encrypted file system is used to encrypt the user's data, manage and create keys which are used for data encryption and decryption [8]. The encryption and decryption of files is transparent to the user and the application [8]. The EFS provides a convenient way to encrypt and decrypt data since encryption, decryption and storage of data is automatic and it performs key

management and key creation functions. Encrypted file systems are implemented in the kernel of the operating system. The keys used for encryption and decryption are stored in disk and are encrypted with the public key of the data owner (user of the computer). The encrypted key is decrypted with the user's private key. Encrypted file systems work well when they are used in personal computers. If they are used in third party organizations like the cloud to protect a client's data, there would be data security breaches. This is because the encryption keys used to encrypt data are stored in the disk and even when the encryption keys are encrypted, they are encrypted using the public key of the user of that storage node. The user of the storage node in the cloud is the system administrator who has the maximum privileges on that storage server. Thus, he could easily get hold of the encryption/decryption keys of the encrypted file system stored in the disk and thereby decrypt the client's data and can even modify it. Hence the confidentiality and integrity of the client's data might be lost. Loss of data confidentiality and integrity is undesirable for a client. They are the two main security issues for a client who is using the cloud services to store his data.

To enable a client to establish trust in the cloud provider's ability to securely store his data within the cloud, we need a solution to achieve confidentiality and integrity of client's data within the cloud.

In this thesis, a solution to the problem of securely storing the client's data by maintaining the confidentiality and integrity of the data within the cloud is developed. The proposed system is called 'A Trusted Storage System for Cloud'. The system is based on trusted computing platform technology [11]. It uses a Trusted Platform Module (TPM), specified by the Trusted Computing Group [11], in every storage node of the cloud to build a trusted storage cloud. The TPM is a chip installed on the motherboard of a computer, that performs basic cryptographic functions like encryption of secret data like encryption/decryption keys, hashing data, storing platform state among others. It also stores the encrypted data within itself. In the proposed system, the TPM is used for remote attestation and EFS key storage. Storing the EFS keys within the TPM

provides security of keys against the system administrator. The TPM achieves secure data storage by encrypting the data using its public key, and the corresponding private key that is used for decryption is known only to the TPM. Remote attestation involves a remote entity authenticating the platform of a storage node, i.e., checking whether the platform is what it claims to be. When the client sends a request to retrieve his data, the EFS has to decrypt the data to send it back to the client. The TPM which has the EFS encryption/decryption keys stored within it, passes on the corresponding keys to the kernel for decryption when requested by the kernel, only after verifying if the platform of the storage node is as expected. The EFS then decrypts the data with these keys. Data decryption is done in the memory. A fraud system administrator can install a malicious code in the kernel to access the memory while the data is being decrypted and hence, he can obtain the keys as well as the decrypted data while they are in the memory. The TPM prevents this undesired memory access by comparing the current platform state with the expected platform state (stored in the TPM), and only when they match does the TPM pass on the decryption keys to the kernel. All the storage nodes within the cloud are attested by a trusted third party node. Attestation is required to ensure safer computing in all environments. Five protocols have been proposed to solve the problem of data confidentiality and integrity.

1.2 Contributions of the Thesis

In the thesis, a solution to the data confidentiality and integrity problem within the cloud is provided by developing a Trusted Storage System for the Cloud. The Trusted Storage System consists of trusted storage servers, a front-end server and a trusted third party node (TTPN) that is maintained by a trusted third party. The TTPN checks the correctness of the platform of a storage server, i.e., it remotely attests to the storage server. After attesting to the storage server, the server is deemed trusted and hence it can be used to store the client's data securely. A client communicates with the cloud through the front-end server and vice-versa.

Five protocols are developed to solve the following problems:

- How a storage node joins the Trusted Storage System in a cloud and hence, becomes trusted.
- How a client can store his data securely on 'trusted' storage nodes.
- How a trusted storage node becomes ready to accept data for replication.
- How a trusted storage node sends the data for replication to other 'trusted' storage nodes.
- How a client accesses his data securely.

Chapter 2: Background and Related Work

2.1 Background

2.1.1 Cloud Computing

National Institute of Standards and Technology (NIST) defines cloud computing as "A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [10]. Cloud computing is a developing technology that enables the IT industry to minimize computing costs by providing services that are charged on usage basis. The cloud users pay as they go, i.e., they pay for the services they use on demand. It is just like paying the monthly electricity bill, i.e., paying just for the electricity used. Therefore, Cloud Computing is also known as utility computing.

The term 'Cloud' is taken from the symbol used to represent the internet. The cloud denotes a collection of servers and computers that can be accessed by the public through the internet. These servers and computers are owned and operated by a third party in multiple data center locations. These machines can run any number of operating systems. Cloud computing delivers hosted services to the clients over the internet [5].

The services provided by the cloud are divided into three main categories: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Software as a Service).

IaaS (Infrastructure as a Service): The cloud provider provides virtual server instances to the clients. The clients can configure the virtual servers and the storage assigned to them [5].

Example: Amazon's EC2.

PaaS (Platform as a Service): The cloud provider allows clients to build their software/applications on the provider's platform over the internet. The platform is nothing but the software and product development tools [5].

Examples: Google Apps Engine and Microsoft's Azure Cloud Platform.

SaaS (Software as a Service): The cloud provider provides software applications to the client and allows him to use them for as long as he needs [5].

Examples: Salesforce.com, Gmail and MapQuest.

Types of clouds:

1. Public Cloud
2. Private Cloud

Public Cloud: It is the cloud that is available to anyone who can access the internet.

Private Cloud: It is the cloud that is available only to a group of people who belong to an organization.

2.1.1.1 Cloud Architecture

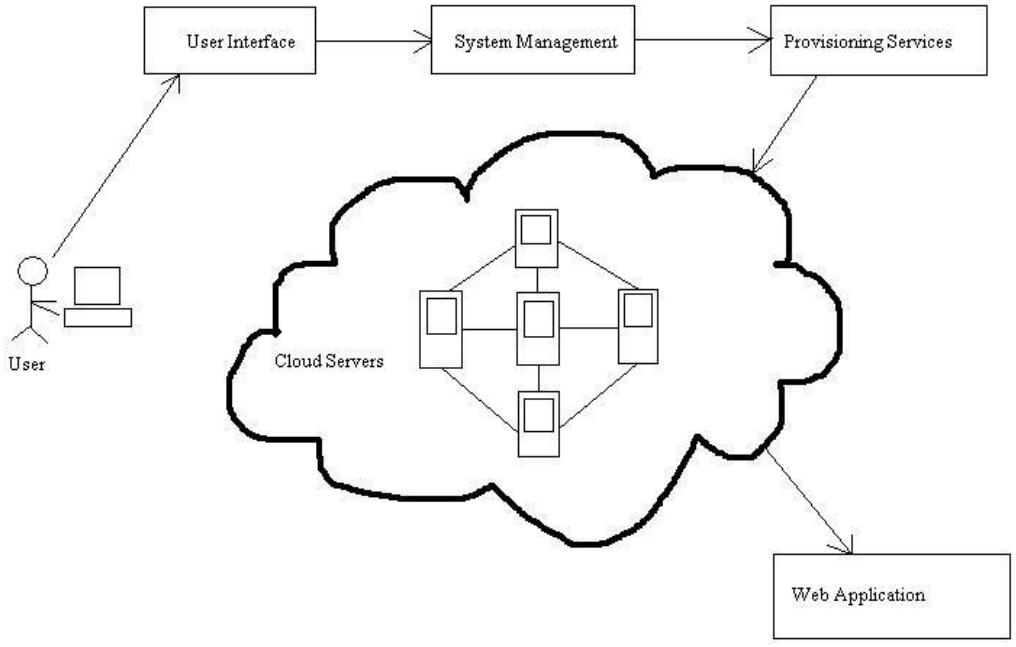

Figure 2.1: Architecture of a Cloud Computing System [6].

The user interacts with the front-end interface from which he selects a service, for example, to store his files, to access a document, or to run an application [6]. The user's request is transferred to the System Management which finds the appropriate resources to be assigned, and calls upon the Provisioning Services mechanism to provision the resources to the user [6]. The Provisioning Services mechanism contacts the cloud servers and processes the user's request [6]. After processing the user's request, the cloud system monitor tracks the usage of resources by the user and records it in his profile [6]. Thus, the cloud provider charges the user according to his cloud usage.
All of these management tasks are automated in the cloud computing system [6].

2.1.1.2 Cloud Storage

The user's data is stored in multiple servers of the cloud rather than on a dedicated server like in the traditional data center storage [6]. To the user, it appears as if his data is stored in a particular location but in fact, the data might move from one location to another in time.

Cloud Storage System Architecture:

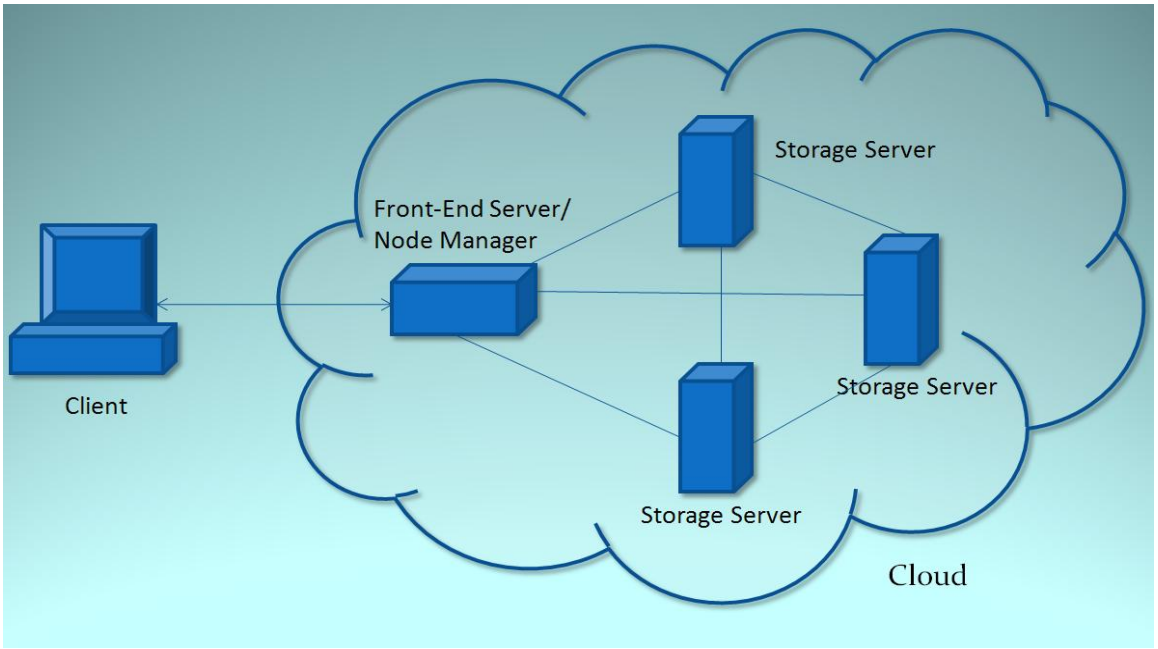

Figure 2.2: Cloud Storage System Architecture.

Figure 2.2 depicts the architecture of a cloud storage system. The cloud consists of several storage servers and a front-end server/node manager which manages the storage servers within the cloud. A client communicates with the cloud through the front-end server and vice-versa. The client uses a web-based interface to communicate with the front-end server of the cloud. The client sends his data/files via the internet to the cloud for storage. The front-end server after receiving the client's files chooses a storage server within the cloud and sends the client's files to it.

The Client's data is replicated within the cloud storage servers for reliability. A storage server sends copies of its data to other storage servers of the cloud.

2.1.2 Encrypted File Systems

An Encrypted File System is used to encrypt files stored in the file system. Encryption by an encrypted file system does not occur at the application level but it occurs at the file system level [8]. Hence, the encryption/decryption procedures are transparent to the user and to the applications. Encrypted file systems provide an easy way to encrypt data by doing this automatically for users, so they do not need to encrypt their data using cryptographic techniques and to manage keys.

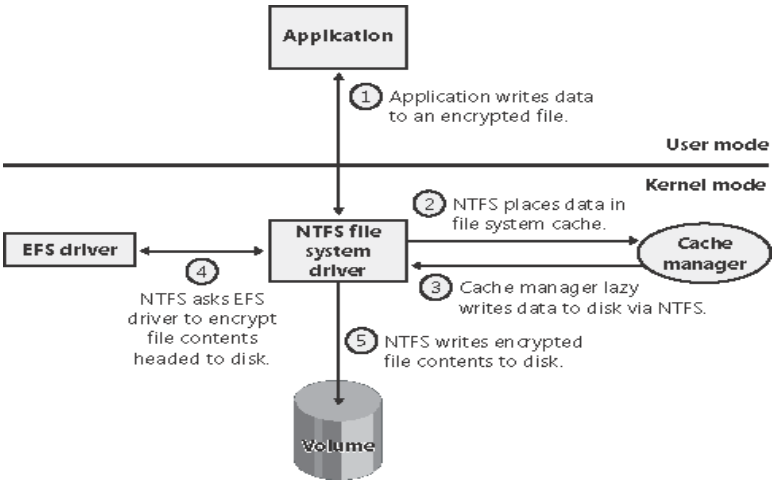

Figure 2.3: Example of flow of an encryption process in an Encrypting File system (Microsoft Windows). (taken from [9])

Figure 2.3 depicts the flow of encryption process in the Encrypting File System (EFS) of Microsoft Windows. The application passes on data that needs to be encrypted to the NTFS file system driver. The NTFS then passes on this data to the cache manager which writes the data to disk through NTFS. The NTFS then contacts the EFS driver to encrypt the data. The EFS driver encrypts the received data using its keys and returns the encrypted data along with its associated encryption/decryption keys to the NTFS. The NTFS writes the encrypted data and the associated keys to the disk.

File encryption key (FEK) for each file is generated by the EFS. This is a symmetric cryptographic key used for data encryption. The FEK is encrypted using the user's public key. The EFS uses the corresponding private key of the user to decrypt the encrypted FEK. The private key of the user is known only to the EFS and the user, thus providing security for the FEK.

2.1.3 Trusted Platform Module

2.1.3.1 What is a TPM?

The Trusted Platform Module (TPM) is a computer microchip or a microcontroller which is designed to perform various security-related and cryptographic functions. It can securely store the artifacts used to authenticate the platform of a computer. The artifacts can include encryption keys, certificates, passwords, and integrity metrics of a platform. The TPM can be used in the process of remote attestation of a platform of a machine which will be discussed further later. It is typically installed on the motherboard of a computer. The TPM uses a hardware bus to communicate with the rest of the system.

The TPM is a specification or implementation of that specification as a chip. The specification is provided by the Trusted Computing Group [11].

2.1.3.2 The Trusted Computing Group (TCG)

The Trusted Computing Group (TCG) is a non-profit international industry standards group which develops, defines and promotes open standards for trusted computing that is beneficial to the users [2]. The TCG develops specifications and publishes them for use and implementation by the industry. Experts from various technologies work together for developing the specifications. Anyone can join or become members of the TCG including non-profit corporations and for-profit corporations and use their specifications for implementation.

The TCG has several work groups which work on different areas. The work groups include: Authentication, Hardcopy, Infrastructure, Mobile Phone, PC Client, Server Specific, Storage, Trusted Network Connect, Trusted Platform Module, TCG Software Stack, and Virtualized Platform. In our problem, we use the Trusted Platform Module.

2.1.3.3 Architecture of the TPM

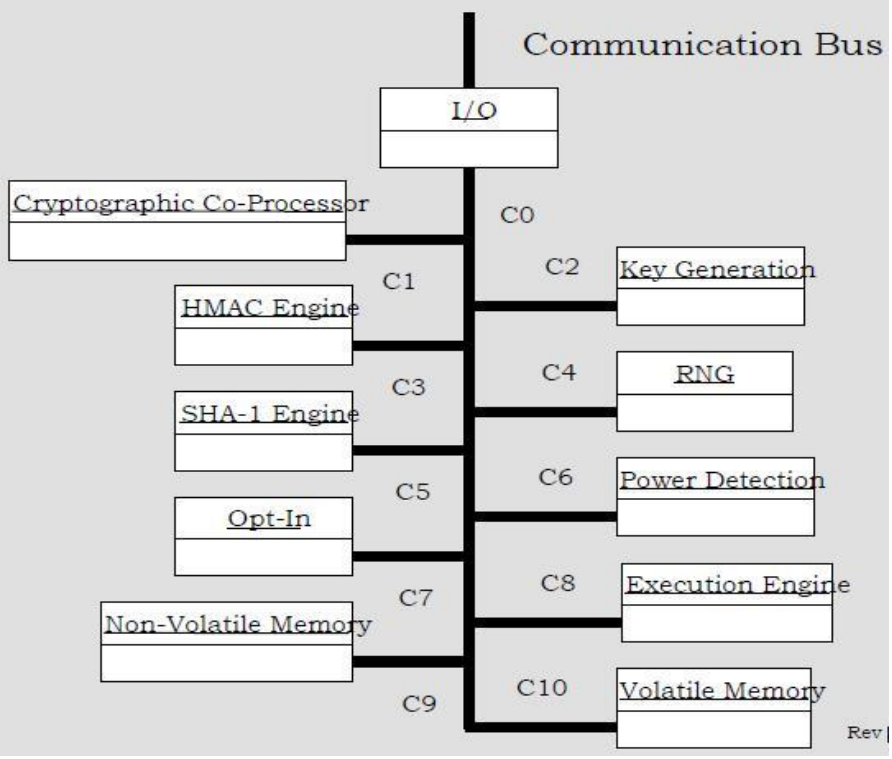

Figure 2.4: Component Architecture of a TPM (taken from [4])

Input and Output (I/O)

The Input and Output component of the TPM controls the information flow through the communication bus [4]. It sends the received messages to the appropriate components in the TPM.

Cryptographic Co-Processor

The Cryptographic Co-Processor performs cryptographic functions within the TPM. The functions include Asymmetric Key Generation (RSA), Asymmetric Encryption/Decryption (RSA), Hashing (SHA-1), and Random Number Generation (RNG) [4].

These functions are used for encrypting data, generating keys, random numbers, and generating hash of data.

Key Generation

The Key Generation Component generates RSA key pairs and symmetric keys [4].

HMAC Engine

The HMAC engine generates two proofs. One is proof of knowledge of authentication data provided by the user and the other is the proof that the request received by the TPM is authorized and no changes have been made to the command in transit [4].

Random Number Generator

The Random Number Generator generates random values which are used as nonces, used for key generation and obtaining randomness in signatures [4].

SHA-1 Engine

The SHA-1 Engine consists of the SHA-1 algorithm whose interface is exposed outside the TPM to support platform measurement taking during the boot time of the machine [4].

Power Detection

The Power Detection Component manages the power states of the TPM in accordance with the power states of the platform [4].

Opt-In

The Opt-In component provides mechanisms to turn-on and turn-off, enable and disable, and activate and deactivate the TPM [4].

Execution Engine

The Execution Engine executes the TPM commands received from the I/O port [4].

Non-Volatile Memory

The Non-Volatile Memory is used for storing persistent identity and state associated with the TPM [4]. It can also be used to store data created by entities authorized by the TPM owner [4].

2.1.3.4 Platform Integrity Measurements

Platform integrity measurements or integrity metrics are the measurements reflecting the integrity of the software state of the platform of a machine [1], for example, the hash of the kernel, sequence of hashes of the software involved in the boot sequence of the machine. These integrity metrics must be reliably measured, stored and reported to a challenger who requests for the platform state of the machine so that the challenger can determine if the platform is in an expected state.

There must be two roots of trust in a trusted platform namely, “root of trust for measuring integrity metrics” and “root of trust for storing and reporting integrity metrics” [1]. A trusted measurement root measures certain platform characteristics, logs the measured data in a measurement store present in the platform, and stores the

final result in a TPM [1]. The trusted measurement root may also measure characteristics of another measurement agent before passing on control to it. In the same way, this second measurement agent may also measure the characteristics of another measurement agent, log the measured data, store the final result in a TPM, and then pass on control to a third measurement agent and so on [1]. For instance, the BIOS stores its own measurements in the TPM and then the BIOS measures the boot loader and logs the measured data in the measurement store and stores it in the TPM. The boot loader now measures the operating system and logs the measured data in the measurement store and stores it in the TPM.

The measured data stored in the measurement store is called “measurement list”, ML.

2.1.3.5 Platform Configuration Register (PCR)

There is a register called platform configuration register (PCR) within the TPM in a shielded location which is used to store the platform integrity metrics. There are a minimum of 16 platform configuration registers in the TPM. The platform integrity measurements made at the initial state (boot time) of the computer are stored in the PCRs. The root of trust for storing and reporting integrity metrics which is in the TPM is responsible for storing and reporting the integrity metrics of the platform in the PCR registers.

Integrity metrics may keep changing in time. So, updated integrity metrics must exist within the PCR. It is difficult to authenticate the source of measurement of integrity metrics. Hence each new integrity metric is not allowed to overwrite the existing integrity metric [4]. The old values and the new values should be stored separately. For this to happen, an indefinite amount of memory must exist. The PCR registers are designed in such a way that they can hold an unlimited number of measurements [4]. The PCR uses a hash function to achieve this.

A pseudo code representing how a PCR stores the measurements [4]:

“PCRi New = HASH (PCRi Old value || value to add)”

2.1.3.6 Trusting a TPM

The TPM has an Endorsement Key pair (private and public keys). The Endorsement private key uniquely identifies the TPM and is not exposed outside the TPM [4]. The Endorsement key is generated by the manufacturer at the time of TPM manufacture. The endorsement public key is signed by the manufacturer to guarantee the correctness of the chip and validity of the key. This is published as an Endorsement Certificate. The Endorsement Certificate certifies that the Endorsement Key is unique and is protected within the TPM at all times. An endorsement key is not used for signing any content which travels outside the TPM due to privacy concerns. There is an attestation identity key, AIK, generated in the TPM which is an alias for the endorsement key [4]. The AIK is an asymmetric key pair. The private portion of the AIK is protected by the TPM and is never exposed outside the TPM. The AIK is a signature key and is not used for encryption. It only signs data generated internally by the TPM. The AIK is used to sign the Platform Configuration Register (PCR) values of the TPM during the platform attestation process which is discussed in section 2.1.3.7. When the TPM signs the platform state stored in the PCRs using the private attestation identity key and sends it to a remote party challenging the TPM, the remote party retrieves the attestation identity key certificate from a certification authority and decrypts the signature of the TPM. Hence the remote party trusts the TPM and believes that the data is prepared by the TPM.

2.1.3.7 Remote Attestation

Attestation is the process of authenticating a platform running in a machine by checking if the platform is in an expected genuine state. Remote attestation is the attestation done by an entity/ challenger which is remote. When a challenger requests a machine to provide its platform state, the TPM of the machine signs the PCR values (made at boot time) stored within it with the TPM's private attestation identity key, pri(AIK). The machine sends the challenger the measurement list, ML, stored in its measurement store, along with the TPM's signed PCR values. The challenger decrypts the signed PCR

values of the TPM with the public attestation identity key, pub(AIK), of the TPM. Since pri(AIK) is known only to the TPM, the challenger is assured that the PCR values have not been modified by the machine or an attacker after successful decryption of the signed content by the challenger. The challenger compares the received measurement list, ML, with the PCR values of the TPM. If they are not the same, the challenger understands that the measurement list has been modified by the machine which happens when the machine is compromised and thus, the attestation fails. If they are the same, the challenger trusts the measurement list to be the actual measurements of the platform running in the machine. The challenger has the expected measurement/state of the platform stored within it. The challenger now compares the received measurement list, ML, with the expected measurements stored within it. If they are the same, the challenger attests to the platform running in the machine. The remote party now trusts the machine (platform) which is using that TPM and can now perform confidential transactions with that machine. If they are not the same, the attestation fails.

2.2 Related Work

He and Xu [7] propose a scheme to protect data on a personal computer platform. They use the trusted computing platform developed by the trusted computing group (TCG) to develop a secure and reliable model for user authentication and data encryption. Their model uses a storage protocol to encrypt data and uses Trusted Platform Module (TPM) to authenticate different users of the PC. The host computer has the trusted computing platform installed within itself, i.e., it has TPM installed in it. First, a trusted connection is established between the host computer and the storage device, i.e., the host and the storage device authenticate each other before any data storage takes place within the storage device and before data is accessed from the storage device by the host. Then, security control is provided by storing access control policies within the storage device. The access control policies apply to users, devices and applications. Message control between the host and the storage device is provided by implementing session-oriented secure data transmission.

Sailor, Doorn and Ward [12] describe a trusted computing platform which extends the hardware-rooted trust guarantees of TCG (Trusted Computing Group) technologies to the operating system and all its applications and allows remote parties to check the trust guarantees. The TCG defines standards for measuring and reporting integrity metrics at the system boot time. However, the TCG does not provide information on the trustworthiness of the runtime of the operating system. Sailor et al. [12] provide a solution to this problem. They give a procedure that explains how a challenging party can attest to the software stack of an untrusted machine. Trust is established between two parties after mutual attestation takes place. The kernel of the attested system (system that needs to be attested) is instrumented to generate measurements of post-boot events which affect the run-time of the system. The components of the software stack that have semantic value are measured. The measured components are kernel modules, executables and shared libraries, configuration files and other important input files that affect trust into the run-time software stack. The measurements are done as soon as executable content is loaded into the system and before it is executed. The attested system creates and stores a measurement list which is a list of hashes of the software stack components. This list is also stored in the TPM (Trusted Platform Module) for achieving integrity. The measured list and the list stored in the TPM are sent to the challenging party. The challenging party compares the two of them and trusts the measurement list of the attested system if both turn out to be the same. After that, the challenging party compares the received measurement list with its stored expected list of measurements. If the two turn out to be the same, the challenging party attests to the system and hence the challenging party is assured that the system is running the expected software stack.

Santos, Gummadi and Rodrigues [13] propose the design of a trusted cloud computing platform (TCCP) which enables the 'infrastructure as a service' providers to provide a closed box execution environment that guarantees confidential execution of a client's virtual machines. Before launching their virtual machines, the clients can attest to the service provider to determine whether or not the service provided is secure. Each node

in the cloud runs a Trusted Virtual Machine Monitor (TVMM) that hosts the client's virtual machines and prevents privileged users from inspecting or modifying them. Each node in the cloud embeds a TPM to enable secure booting and for attestation. Santos et al. propose TCCP protocols to securely launch virtual machine of a client on a machine in the cloud and migrate it safely to other machines in the cloud. The TVMM guarantees that a virtual machine is launched on a trusted machine in the cloud and that the system administrator will not be able to inspect or tamper with the initial state of the virtual machine as it traverses the path between the user and the machine hosting the virtual machine.

Chapter 3: Proposed Trusted Storage System for the Cloud

A client/user stores his data within the cloud storage servers. Since cloud is a third party system, it cannot be trusted. We consider the confidentiality and integrity issues of client's data within the cloud and provide a solution to these issues by proposing a framework/system called a Trusted Storage System for Cloud. The system provides data security against the cloud provider, especially against the system administrator who has the maximum number of privileges over the storage nodes in the cloud.

3.1 System Architecture

The proposed system consists of the following entities:

- User/Client
- Trusted Third Party Node (TTPN)
- Front-End Server (FES)
- Group of Storage Servers/Nodes

User/Client

The User/Client is the entity that uses the cloud to store its data. It trusts the Trusted Third Party Node (TTPN) only.

Trusted Third Party Node (TTPN)

The TTPN is an entity maintained by a trusted third party and does not belong to the cloud. It is the integral entity of the Trusted Storage System. It checks each node/storage server in the cloud for correctness of platform and attests to it. After attestation by the TTPN, a node is deemed trusted and can then store the user's data within itself.

Front-End Server (FES)

The Front-End Server is an entity which forwards the messages it receives from the entities of the system (like the users, TTPN, storage servers) to other entities in the system. The FES maintains a directory which contains information about the location of

each client's data, i.e., in which storage node within the cloud each client's data is stored.

Group of Storage Servers/Nodes

The Storage Servers/Nodes are the entities which are within the cloud and they store the user's data. They are attested by the TTPN so that they can store the user's data safely. Every storage node has a TPM installed in it.

The cloud storage servers use an encrypted file system for encrypting the data of the clients. We assume that the key creation and management is done by the encrypted file system and it takes care of the encryption and decryption of data and storage within the storage server. If the user encrypts his data with his keys, he should reveal his keys to the other users with whom he wants to share his data so that they decrypt the data using his keys. Revealing user's encryption/decryption keys is not desirable.

The TTPN sends the user a one-time symmetric key to encrypt his data with before storing his data in the cloud for data confidentiality so that the user need not create or manage encryption keys. After the cloud receives the client's encrypted data, it decrypts the data which is being encrypted with the one-time symmetric key and then, the encrypted file system re-encrypts the data with its own keys which are more secure. The user just needs to create a user key, which the TTPN and the storage nodes use for secure communication with the user.

3.2 Role of the TPM in the system

3.2.1 Remote Attestation

A remote node can attest to the storage node in the cloud with the help of TPM installed in the storage node. To attest to a storage node is to certify that the node is trustworthy and can henceforth be used for storage. Trustworthiness of a node means the platform of the node is in an expected secure state and that it is not being modified for malicious actions.

The remote node in the proposed system is the Trusted Third Party Node (TTPN) which authenticates the platform of the storage node and then attests to the storage node.

Why do a platform authentication?

There must be a method to check whether the platform is in a state which it claims to be in. This is called platform authentication. A malicious system administrator in the cloud can run a malicious program within the kernel, for instance, to read the memory during data decryption. Hence he will be able to obtain the decrypted data as well as the keys used during decryption in memory. In order to avoid such attacks by privileged users like the system administrator, the platform needs to be checked during the system/node boot time if it is in the expected state, i.e., no malicious program is installed in the platform. The system administrator should reboot the node in order to successfully install a malicious program in the kernel. Therefore, whenever a node reboots, platform authentication must be performed.

Platform authentication and storage node attestation is explained in section 3.3 in protocol 1.

3.2.2 Protected Storage of Encryption Keys

The TPM stores the keys (secret data) of the encrypted file system which are used for encryption of client's data. The encrypted file system, after encryption of the client's data, transfers the keys to the TPM for secure storage. The secret data (keys) is encrypted using a key known only to the TPM. So, only the TPM can decrypt the secret data stored in it using its key.

There is a key hierarchy for all the keys which are used for protected storage within the TPM [3]. The root key for this hierarchy is called Storage Root Key (SRK). Each of the keys in the hierarchy is encrypted with the key above it in the hierarchy. Key blobs and data blobs are formed as a result of the encryption of keys and the data. Figure 3.1 depicts key storage within the TPM. FEK is the file encryption key used to encrypt files. The keys are represented as data in the figure. Each file encryption key is encrypted with

a TPM key which is again encrypted with another key in the hierarchy and so on. The root key used is called Storage Root Key (SRK) [1]. RSA encryption algorithm is used for key/data encryption. The encrypted data is called data blob and the encrypted key hierarchy is called key blob. The Storage Root Key is known only to the TPM and is not exposed outside TPM [1]. Hence, the keys stored within the TPM are secure.

The size of the keys which are going to be stored in the TPM cannot be bigger than the size of the keys of the TPM which are used to encrypt them [1]. If the key used for file encryption is small, the encryption of that key is performed within the TPM chip using its RSA engine [3]. If the key is large, the encryption of the key can be done in either of the following ways:

1. The platform itself can encrypt the key using a one-time symmetric key of small size. And that symmetric key can be stored in the TPM safely [3].

2. The platform can divide the key into smaller blocks and then send these blocks to the TPM for encryption of each block separately. The platform is responsible for reassembling the blocks when the key is retrieved from the TPM [3].

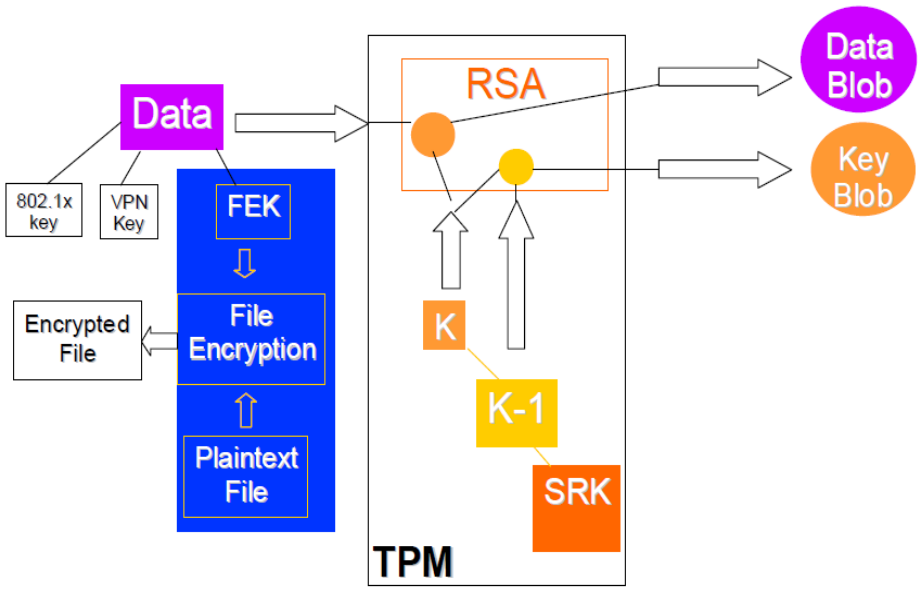

Figure 3.1: Key Storage within the TPM. (taken from [3])

Secure access to keys:

When the kernel requests the TPM for access of EFS keys for data decryption, the TPM decrypts those keys and passes them on to the kernel when it believes that the platform on that node is trustworthy, i.e., the platform is in the expected state. First, the TPM verifies if the TTPN has attested to the storage node. If the storage node has been attested by the TTPN, the current platform state is compared with the expected genuine platform state stored in the PCRs of the TPM. If they are not the same, the TPM does not pass on the requested keys to the platform. If they are the same, the TPM passes on the keys to the platform.

3.3 Joining of a storage node in the Trusted Storage System of the Cloud

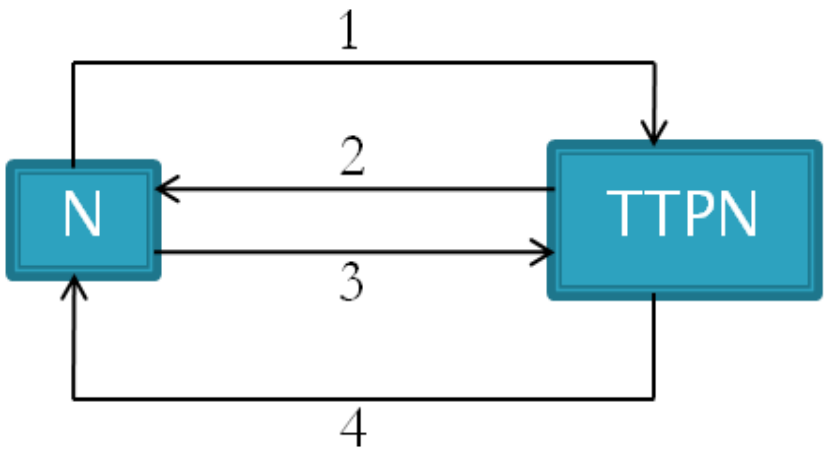

Figure 3.2: Joining of a storage node in the Trusted Storage System of the Cloud.

A storage node in the cloud must join the Trusted Storage System before it can store a client's data. The TTPN tests if a TPM is being installed on the storage node and if installed, it checks for the correctness of the platform of the storage node. This process

is called attestation. After successful attestation of the platform of the storage node by the TTPN, the storage node is deemed trusted and can store a client's data securely.

Notations

n_{TTPN}: Nonce of the TTPN

$PCRV_N$: The PCR value stored in the TPM of node N (Hash of the kernel of the node and sequence of hashes of the software involved in the boot sequence of the node - bootstrap loader, BIOS)

ML_N: Measurement list of the node N

pri(AIK): Private attestation identity key of the TPM of node N

pub(AIK): Public attestation identity key of the TPM of node N

C(AIK): Attestation identity key certificate of the TPM of node N

pub(N): Public key of the node N

Nid: ID of the node N

pub(TTPN): Public key of the TTPN

pri(TTPN): Private key of the TTPN

Protocol 1

Message 1: {"Join System", Nid}

Message 2: {"Send platform state", n_{TTPN}}$_{pri(TTPN)}$

Message 3: {{$PCRV_N$, n_{TTPN}}$_{pri(AIK)}$, ML_N, pub(N), pub(AIK), C(AIK)}$_{pub(TTPN)}$

Message 4: {"Joined"}$_{pri(TTPN)}$

A storage node N sends message 1, a "Join System" message which includes its ID, to the Trusted Third Party Node (TTPN). The TTPN in response, sends message 2 which contains a nonce (a one-time large random number) to the storage node, N, as a challenge to prevent replay attacks. The TTPN signs message 2 with its private key, pri(TTPN).

The node N prepares a measurement list ML which contains the hash of the kernel and sequence of hashes of the software involved in the boot sequence of the node – bootstrap loader and BIOS at the boot time. The node stores this measurement list in a measurement store within it. This measured list is also stored in the PCR registers of the TPM. When the storage node N receives message 2 from the TTPN, the TPM of that node signs the stored PCR value, $PCRV_N$, and the received nonce of the TTPN, n_{TTPN}, with its private attestation identity key, pri(AIK). Since the private attestation identity key, pri(AIK), is known only to the TPM, the signature assures that the content is prepared by the TPM only. The TPM passes on this signed content to the storage node along with the public attestation identity key, pub(AIK), and the attestation identity key certificate, C(AIK). The storage node prepares message 3 which contains the content received from the TPM and the measurement list, ML, and the public key of the node N, pub(N). The node creates a private key, pri(N), corresponding to the public key, pub(N), so that it can use this pair of keys in further communications with the TTPN and the client. The private key is stored in the memory of the node and is not saved anywhere. The key pair denotes the identity of the storage node. Message 3 is encrypted with the public key of the TTPN to achieve confidentiality of the message.

When the TTPN receives message 3, it decrypts the message using its private key, pri(TTPN). It first views the attestation identity key certificate, C(AIK), and then trusts the public attestation identity key of the TPM, pub(AIK). It uses pub(AIK) to decrypt the following: $\{PCRV_N , n_{TTPN}\}_{pri(AIK)}$. After a successful decryption with the public attestation identity key of the TPM, pub(AIK), the TTPN is assured that a TPM is being installed in that storage node and that the received content is being prepared by the TPM. The

TTPN retrieves the PCR value, $PCRV_N$, and the nonce, n_{TTPN}. It compares the received PCR value with the received measurement list, ML_N, of the storage node N. If the values do not match, the TTPN comes to know that the measurement list has been changed and the attestation fails. If the TTPN finds that both the values are equal, it trusts the measurement list, ML_N, of the node N to be a genuine measurement of the current platform of the node N. The TTPN compares this measurement list with the expected state of the platform stored within itself. If the values do not match, the attestation fails. If the TTPN finds the values to be the same, it trusts that the platform of the storage node N has not been changed or tampered and hence attests to the storage node N. The TTPN then stores the public key of the node along with its ID in its database.

The private key of the storage node, pri(N), is stored in the memory of the storage node. So, when the node reboots, the private key is lost and hence a compromised node cannot use the same public and private key pair to communicate with the TTPN. Hence, the storage node must rejoin (by repeating this protocol) the trusted storage system if it gets rebooted.

The TTPN now attests to the storage node by sending a "joined" message encrypted with its private key, pri(TTPN), conveying the node that it has been accepted as part of the Trusted Storage System of the Cloud. The storage node N is now deemed "trusted". It can now securely store a client's data.

Correctness

We show that only storage nodes with the expected platform state join the trusted storage system.

A storage node prepares a measurement list, ML_N, which denotes the measurement of the current state of the system at boot time. The measured state of the system is securely stored in the TPM of the storage node for maintaining the integrity of the measured state. The TTPN has the expected state of the platform stored in it. When a

TTPN receives a “join system” request from the storage node, it sends a “send platform state” request to the storage node. In return, the storage node sends the measurement list, ML_N, and the platform state, $PCRV_N$, stored in the PCR registers of the TPM which is signed by the TPM. The TTPN trusts the signed content of the TPM after viewing the attestation identity key certificate which is signed by a certification authority.

There is a possibility that a malicious/ compromised node can modify the measurement list, ML_N, to represent a genuine platform state. The TTPN compares the signed PCR value, $PCRV_N$, with the measurement list, ML_N, of the node. If they are different, the TTPN does not attest to the storage node. The attestation fails. If they are equal, it trusts the measurement list, ML_N, of the node. The TTPN now compares the measurement list, ML_N, with the expected state stored within it. If they are different, the attestation fails. If they are equal, the TTPN trusts the platform running on the storage node and hence attests to the node and sends a “joined” message to the node.

So, only nodes with an expected/genuine platform are allowed to join the trusted storage system.

3.4 Data Storage in the Trusted Storage System

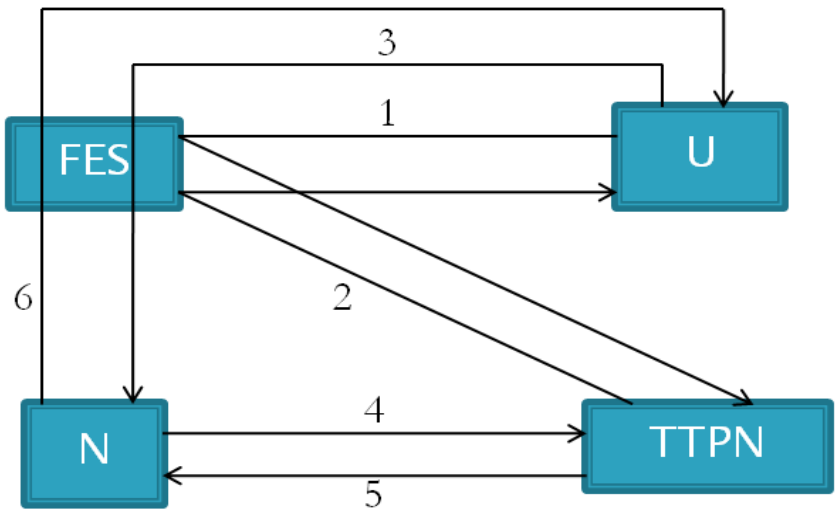

Figure 3.3: Data Storage in the Trusted Storage System.

Notations

UK: User Key

Uid: User ID

n_U: Nonce of the user

n_{TTPN}: Nonce of the TTPN

n_N: Nonce of the node

SK: One time session key

pub(TTPN): Public key of the TTPN

pri(TTPN): Private key of the TTPN

pub(N): Public key of the node

pri(N): Private key of the node

$\{files\}_{SK}$: User's/client's files encrypted with the session key

Nid: Node ID

Protocol 2

Message 1: $\{\text{"Data Send Req"}, UK, Uid, n_U\}_{pub(TTPN)}$

Message 2: $\{\{SK, n_U, n_{TTPN}\}_{UK}, \{H(n_U)\}_{pri(TTPN)}\}$

Message 3: $\{\{files\}_{SK}, \{H(\{files\}_{SK}), n_{TTPN}\}_{pub(TTPN)}\}$

Message 4: $\{Message\ 3, (H(Message\ 3))_{pri(N)}, \{n_N, Nid\}_{pub(TTPN)}\}$

Message 5: $\{\{files\}_{SK}, \{\{n_N, UK, Uid, SK, H(\{files\}_{SK})\}_{pub(N)}\}_{pri(TTPN)}\}$

Message 6: $\{\text{"Data Stored"}\}_{UK}$

The client/user sends a "data send request" (message 1) to the front-end server (FES) of the cloud to store his data in the cloud. The front-end server forwards the message to

the TTPN. The user generates a symmetric encryption key, UK, called user key, to use it in further communications with the cloud. Message 1 includes the user key, user ID, Uid, and the user nonce, n_U, used for preventing replay attacks. The message is encrypted with the public key of the TTPN so that only the TTPN can decrypt and view the message.

In return, the TTPN sends message 2 to the client via the front-end server. The TTPN creates a one-time session key, SK, so that the client can encrypt his data with it. This session key is used only for that user session and will not be used again. The TTPN includes its nonce in the message, n_{TTPN}, along with the nonce of the user, n_U, which it received. The TTPN encrypts the session key and the nonces with the user key UK so that only the user can decrypt the message with UK. It signs the hash of the user's nonce, $H(n_U)$, with its private key pri(TTPN) to prove the user that the message is prepared only by the TTPN and is not tampered by an attacker.

When the user receives message 2, it decrypts the message with his key, UK, and retrieves the session key, SK, and the nonces. It also retrieves the hash of its nonce, $H(n_U)$, which is signed with the TTPN's private key, pri(TTPN). The user decrypts the hash with the TTPN's public key, pub(TTPN) and is now assured that the message is prepared by the TTPN only. The user then uses that session key to encrypt his files. He sends message 3 to the front-end server which includes the encrypted files which are being encrypted with the session key, SK. Message 3 also includes the hash of the encrypted files, $H(\{files\}_{SK})$, and the nonce of the TTPN, n_{TTPN}, both encrypted with the public key of the TTPN, pub(TTPN).

When the front-end server receives message 3 from the client, it picks a storage server (node) within the cloud and forwards message 3 to that node. The node must authenticate itself to the TTPN before it can store the data of the client. Since the client's data is encrypted with SK, the node cannot view the client's files before it gets itself authenticated to the TTPN. The storage node N computes a hash of Message 3 and signs it with its private key, pri(N), to convey the TTPN that the message is not modified

and also for its authentication. The storage node prepares Message 4 and sends it to the TTPN for authentication. Message 4 includes message 3 sent by the user, (H(Message 3))$_{pri(N)}$, the nonce of the node, n_N, to avoid replay attacks and the ID of the storage node, Nid. The node encrypts the nonce, n_N, and the node ID, Nid, with the public key of the TTPN, pub(TTPN).

When the TTPN receives message 4 from the node, it retrieves the node ID, Nid, and the nonce of the node, n_N, by decrypting the content with its private key, pri(TTPN). It also retrieves message 3 and its hash, H(Message 3) signed by the node N with the private key of the node, pri(N). It checks for that node's public key, pub(N), in its database. If the TTPN finds the public key, the TTPN concludes that the storage node N is a member of the trusted storage system. The TTPN decrypts the signed hash using the node's public key. The TTPN computes a hash of the received message 3 and compares the hash with the received hash. If the hashes are the same, the TTPN is assured that the contents of message 3 are not modified. The TTPN decrypts message 3 using its private key, pri(TTPN) and retrieves its nonce, n_{TTPN}, sent to the user in message 2 and hence considers this message 4 to be a fresh message. It also retrieves the hash of the encrypted files and the encrypted files encrypted with the one-time session key, SK. The TTPN computes a hash of the received files and compares it with the received hash. If they are not the same, the TTPN identifies that the files are modified in transit and hence does not authenticate the storage node. If the values are the same, the TTPN authenticates the node to store the user's files. If the TTPN does not find the public key of the node, pub(N), in its database, it rejects the data storage request of the node and does not authenticate the node.

After authentication, the TTPN prepares message 5 and sends it to the node. Message 5 includes the nonce of the node, n_N, the user key, UK, the user ID, Uid, the session key, SK, which is used by the user to encrypt his files and also the hash of the encrypted files of the user. All of this content is encrypted using the public key of the node, pub(N), so that only the node can decrypt the content. This content is signed by the TTPN so that

the node is assured that the message is prepared by the TTPN. Message 5 also contains the user's files encrypted with the session key, SK.

When the storage node N receives message 5 from the TTPN, it decrypts the message using the public key of the TTPN, pub(TTPN), and now trusts that the message is prepared by TTPN only. It decrypts the inner content of the message using its private key, pri(N), and retrieves its nonce, n_N, the user key, UK , the user ID, Uid, the session key, SK, and the hash of the encrypted files. The storage node also retrieves the encrypted files. It computes a hash of the received files and compares it with the received hash. If they are not the same, the node identifies that the files are modified in transit. If they are the same, the node is assured that the files are not modified. The node then decrypts the files using the session key. It uses its encrypted file system to re-encrypt the files with the encrypted file system's keys. After encrypting the files, the EFS stores them in the disk. The EFS stores the corresponding encryption keys and the user key along with the user ID in the TPM. Since the encryption keys and the user key are stored in the TPM, they are safe. This is because the TPM encrypts these keys using its RSA public key whose corresponding private key is known only to the TPM.

The node now sends message 6 (encrypted with the user key, UK) to the user via the front-end server. Message 6 assures the user that his data is being stored in the cloud.

Data Integrity

During data storage, the EFS computes a hash of the user's data and stores it in the TPM. When the data is accessed by the user, the EFS computes the hash of the user's data and compares it with the hash stored in the TPM. If both the hashes are equal, the EFS is assured that the user's data is not modified. If the hashes are not the same, an integrity breach is identified.

Correctness

We show that a user's data is securely transmitted to the cloud and is stored on trusted storage nodes only, thus, achieving confidentiality and integrity.

User's data is transmitted securely and is stored only on trusted storage nodes:

The user encrypts his data with the one-time symmetric key, SK, which is created and provided to the user by the TTPN which is trusted by the user. The user sends the encrypted data to the cloud. A storage node within the cloud can decrypt the user's data, re-encrypt it with the EFS keys and store the user's data within it only after getting itself authenticated by the TTPN.

The storage node sends an authentication request to the TTPN by signing the request with its private key. The TTPN authenticates the storage node if it is a member of the trusted storage system by checking its database if it has the public key of the storage node. The TTPN does not authenticate the storage node in the following situations:

1. If the storage node is not a member of the trusted storage system, i.e., if the public key of the storage node is not found in the database of the TTPN.

Or

2. If the public key found in the database does not decrypt the signature of the node. This is the case when the storage node is compromised/ rebooted.

A storage node is compromised when a malicious program is installed in the node which tries to access the user's data or modify the EFS which is in the kernel, to perform malicious activity. The user's data is encrypted and stored in the disk of the storage node at all times. It is in an unencrypted form when it is decrypted by the storage node for re-encryption by the EFS during data storage or for re-encryption with the user's key during data access. Data decryption is done in the memory. There is a possibility that an attacker or a fraudulent system administrator can read the memory during data decryption. To achieve this, the attacker needs to install a program in the kernel to access the data in the memory. We assume that an attacker needs to reboot the system in order to install a malicious code, i.e., in order to compromise the system.

After a system reboot, the storage node will lose its private key whose corresponding public key is stored in the TTPN during the execution of protocol 1. This is because, the private key is stored in the storage node's memory and after a reboot, it is lost and the storage node cannot use the same private key in further communications with the TTPN. Hence, a compromised storage node cannot pose as a trusted storage node for data storage and so, cannot store the user's data. After a system reboot, the storage node must rejoin the trusted storage system by repeating protocol 1.

After successful authentication of the storage node, it is allowed to store the client's data securely.

Secure storage of keys

The EFS encryption/decryption keys and the user key, UK, are stored in the TPM. The TPM encrypts these keys using one of its RSA public keys and the corresponding RSA private key is known only to the TPM. When the EFS keys or the user keys are requested by the storage node (kernel), the TPM checks if the current state of the platform is as expected and only then releases the keys to the storage node (kernel). If the current state of the platform is not as expected, the TPM does not release the keys.

3.5 Data Replication

Each storage server replicates its data onto other servers in the cloud to achieve data reliability. The data stored in a trusted storage server should be replicated onto other trusted storage servers only. That is, the data should be replicated onto storage servers which are members of the trusted storage system.

3.5.1 Storage node ready to accept data for replication

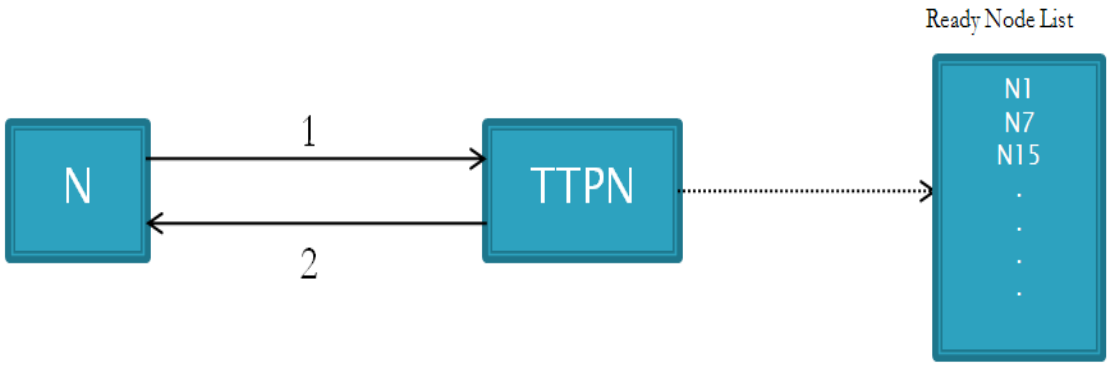

Figure 3.4: Storage node ready to accept data for replication.

In order to ensure that the data for replication is transferred to trusted storage servers, the TTPN maintains a list of trusted storage servers that are ready to accept data for replication. This list is called "ready node list". The ready node list is a dynamic list which keeps changing when new nodes are added to it or existing nodes get deleted from it. A storage server which is ready to accept data for replication sends a "ready for replication" message to the TTPN. The TTPN checks if that server is a member of the trusted storage system. If it is a member, it adds that server to the ready node list.

Notations

pri(N): Private key of the node N

Nid: ID of the node N

Protocol 3

Message 1: {"Ready for replication", {"add to ready node list"}$_{pri(N)}$, Nid}

Message 2: {"Added/ Rejected"}

A storage server which is ready to accept data for replication sends “ready for replication” message which includes its ID and “add to ready node list” message which is encrypted with the private key of the node, pri(N), so that the TTPN can authenticate the node by decrypting it using the public key of the node, pub(N).

When the TTPN receives message 1, checks its database if it has the public key, pub(N), of that node stored, i.e., it checks if that node is a member (“trusted node”) of the Trusted Storage System. If the TTPN finds the public key, it decrypts the “add to ready node list” message and thus successfully authenticates and adds the storage node to its ready node list. If the TPPN does not find the public key of that node or if it is not able to decrypt the message with the existing public key of that node, it identifies that the node is not a member of the trusted storage system or that the node has been rebooted and compromised. The TTPN will not add that node to its ready node list. So, the ready node list contains trusted storage nodes only. In response to message 1, the TTPN sends message 2 containing “added/rejected” message.

When a node cannot accept any more data for replication, it sends “delete from ready node list” message to the TTPN. The TTPN deletes that node from the ready node list.

3.5.2 Storage node sends data for replication

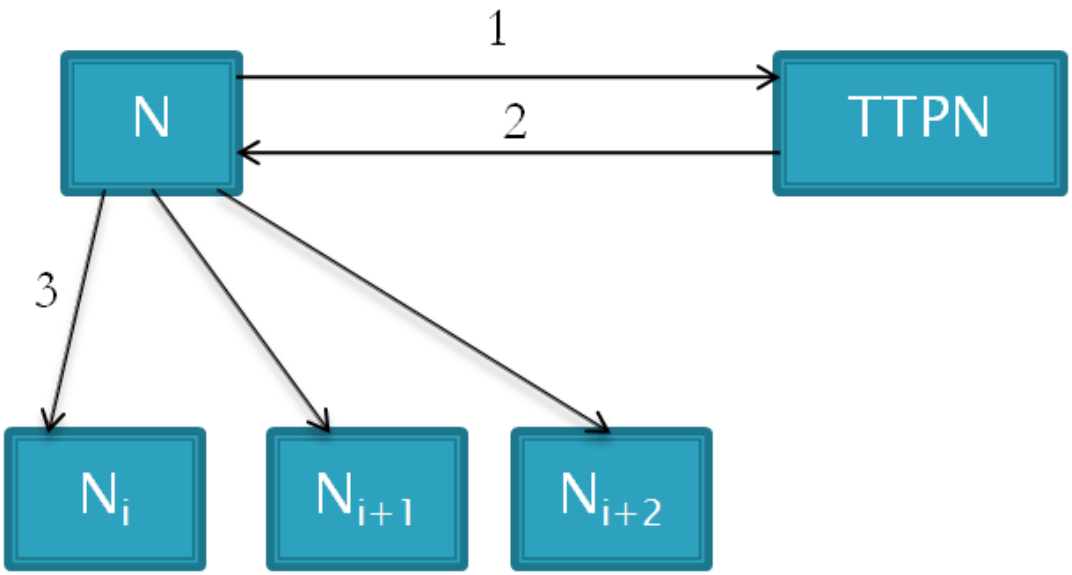

Figure 3.5: A trusted storage node sends its data for replication to other trusted storage nodes.

A storage node N in the cloud which is going to replicate its data on other storage servers, should ensure that they are trusted, i.e., they are members of the trusted storage system, before storing data in them. The storage node N requests the TTPN to send a list of ready nodes that accept data for replication. Since, the ready node list consists of storage nodes that are trusted, the storage node N is assured that the data will be replicated securely. After receiving the ready node list from the TTPN, the storage node N sends its data to the storage nodes, (N_i, N_{i+1}, N_{i+2},…), in the received list.

Notations

Nid: ID of node N

{(N_i, pub(Ni)), (N(i+1), pub(N(i+1))), (N(i+2), pub(N(i+2))),……..)}: Pairs of IDs and public keys of ready nodes.

Pri(TTPN): Private key of the TTPN

EFSK: Symmetric key created and used by the encrypted file system of the storage node to encrypt/decrypt the client's data

$\{files\}_{EFSK}$: Client's files/data encrypted with the encryption file system key

Uid: User ID

UK: User key

Protocol 4

Message 1: {"Data Replication", Nid, n_N}

Message 2: $\{n_N, (Ni, pub(Ni)), (N(i+1), pub(N(i+1))), (N(i+2), pub(N(i+2))),\ldots\ldots\}_{pri(TTPN)}$

Message 3: $\{\{files\}_{EFSK}, \{H(\{files\}_{EFSK}), EFSK, Uid, UK\}_{pub(Ni)}\}$ // Note: Message 3 is sent to each node in the received ready node list encrypted with the ready node's public key.

Protocol 4 ensures that the data is replicated only on trusted storage nodes in the cloud.

The storage node N prepares a "Data Replication" message that includes the node's ID and nonce, n_N, to prevent replay attacks. The storage node sends this message to the TTPN.

When the TTPN receives message 1, it prepares a list of ready node IDs and their public keys, {(Ni, pub(Ni)), (N(i+1), pub(N(i+1))), (N(i+2), pub(N(i+2))),……..} and sends this list and the received nonce, n_N, to the storage node N as message 2 which is signed with the TTPN's private key, pri(TTPN), so that the node is assured that the list is being prepared only by the TTPN and hence, it can trust the list of ready nodes. There is a chance that an attacker modifies the ready node list prepared by the TTPN or creates his own list of malicious nodes and sends it to the storage node N. Since message 2 is signed with the private key of the TTPN, the list cannot be tampered or changed by any attacker.

The node decrypts message 2 with the public key of the TTPN, pub(TTPN) and retrieves the ready node IDs and their public keys. The node now sends the encrypted data, that has to be replicated, along with the hash of the encrypted data and the encryption information, i.e., the key used by the EFS to encrypt/decrypt the client's data and the data owner's (user's) information, i.e., its user ID, Uid, and the user key, UK, as message 3 to node Ni which is ready to accept the data for replication. Message 3 is sent to each node in the received set. Hash of the files, EFSK, UK and Uid are encrypted with the public key of the node Ni, pub(Ni), so that only that node can retrieve the encrypted files and the other information.

Each node Ni that receives message 3 decrypts the message with its private key, pri(Ni), and retrieves the hash of the encrypted files along with the encryption/decryption key and the user information. It also retrieves the encrypted files. The storage node Ni computes the hash of the received encrypted files and compares it with the received hash. If they are not equal, the node identifies that the files are modified and hence does not store the files within it. If the values are equal, the node is assured that the files are not modified. It stores the encrypted files in the disk. The encryption/decryption key and the user information are stored in the TPM securely.

Correctness

We show that a user's data is replicated only on trusted storage nodes.

The TTPN prepares a list of storage nodes which are ready to accept data for replication. This ready node list contains trusted nodes only. A storage node sends a request to the TTPN to send the ready node list to it for replicating its data. When the TTPN receives the request, it sends the ready node list to the storage node. There is a chance that an attacker modifies the ready node list prepared by the TTPN or creates his own list of malicious nodes and sends it to the storage node N. Since message 2 is signed with the private key of the TTPN, the list cannot be tampered or changed by any attacker. Hence, the storage node is assured that the ready node list that it receives is prepared by the

TTPN only. The storage node replicates its data onto the nodes in the received list. Hence, data is replicated on trusted storage nodes only and is secure.

3.6 Data Access

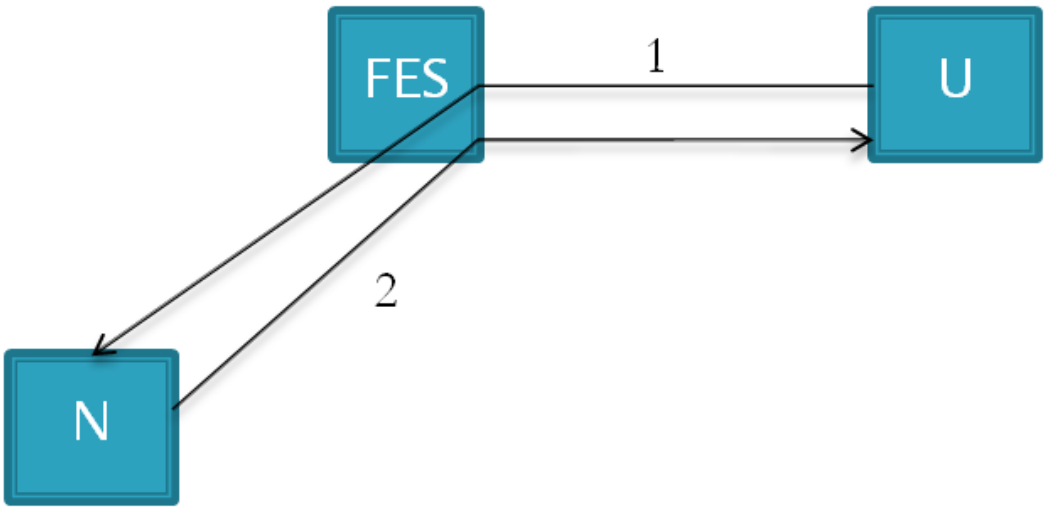

Figure 3.6: User accesses his data.

Protocol 5 explains how the user accesses his data from the cloud securely. The user contacts the front-end server, FES, to access his data. The FES sends the user's request to a storage node which has the user's files and the storage node securely sends the user files to the user.

Notations

Uid: User ID

n_U: nonce of the user

files: Client's files

pub(N): Public key of the storage node N

pri(N): Private key of the storage node N

UK: User key

Protocol 5

Message 1: {"Access Data", $\{n_U$, file name/set of file names$\}_{UK}$, Uid}

Message 2: $\{\{files, n_U, pub(N)\}_{UK}, (n_U+1)_{pri(N)}\}$

The user accesses his data stored in the cloud by sending the file name or a set of file names it needs to access and the user's nonce, n_U, and the user ID to the front-end server (FES) in message 1. The user encrypts the nonce and the file names with its key, UK, to get itself authenticated by the cloud so that the cloud does not send the requested files to someone else.

When the front-end server, FES, receives message 1 from the user, it picks a storage node N from its directory which has information about where each user's files are stored, and forwards the user's request (message 1) to that storage node. The storage node N checks if it has the file/set of files stored in it. If it has the files, the encrypted file system (EFS) of that node should decrypt the files with its keys which are stored in the TPM. The storage node contacts the TPM for the respective keys. The TPM passes on the keys to the kernel of the node for decryption if the current state of the platform is as expected, i.e., if the measurement list of the current platform is same as the PCR values stored in the TPM.

After decryption of files, the node re-encrypts the files, the received user nonce, n_U, and the public key of the node, pub(N), with the user key, UK, to achieve data confidentiality. The storage node also signs (n_U+1) to assure the user that the message is prepared by the storage node only. These two contents are sent in message 2 to the user.

When the user receives message 2, it decrypts $\{files, n_U, pub(N)\}_{UK}$ using its key, UK, and hence retrieves its files securely. It checks whether the received nonce matches its

nonce. If it matches, the user deems the message to be fresh. The user key, UK, should not be compromised at any time.

The user uses the public key of the node, pub(N), to decrypt the signature of the node, $(n_U+1)_{pri(N)}$. After successfully decrypting the signature, the node is assured that the message is prepared by the storage node only.

Correctness

We show that the protocol allows secure data access by a user.

A user U requests the cloud to access his data by encrypting the required file name/ set of file names with his user key, UK. When a user U sends a request to access his data, the storage node encrypts the user's data with the user's key, UK, and sends the encrypted data to the user. UK is known only to the user, TTPN and the trusted storage nodes. So the data is securely accessed by the user. The signature, $(n_U+1)_{pri(N)}$, assures the user that the message is prepared by the storage node only. The user nonce in the messages ensures that the message received by the user is fresh.

When some other user of the cloud, U', requests the cloud to access data of user U, the storage node sends the data to U' (encrypted with the user key of U') only when the storage node finds the ID of user U' in the access control list of the data. So, a user's data is accessed by those users of the cloud to whom the user has given access privilege.

Nonces are being used in messages of all the protocols to prevent replay attacks. So, each entity in the system is assured that it is receiving fresh messages.

Chapter 4: Conclusions

Currently, organizations and individual users are not coming forward confidently to use the cloud service for data storage though it is a cost-effective service. The main reason for this is that the cloud is a system maintained by a third party and hence it cannot be trusted. Data security is the main challenging issue in cloud computing since all communications with the cloud are done through internet which is prone to attacks. We consider the confidentiality and integrity problems of client's data stored in the cloud. In this thesis, a framework for trusted storage of client's data within the cloud is developed. The framework provides data security against the system administrator who has the maximum number of privileges on a storage node. The system uses a Trusted Platform Module (TPM) chip in each storage node which provides protected storage of encryption/decryption keys of client's data and remote attestation of each storage node. Each storage server has an encrypted file system which encrypts the client's data and the corresponding keys are stored in the TPM. Cryptographic techniques are used to provide secure communication between the client and the cloud. The system ensures that the client's data is stored only on trusted storage servers and it cannot be transferred by malicious system administrators to some corrupt node. The system architecture and the proposed protocols together form a Trusted Storage System for Cloud. The system achieves confidentiality and integrity of the client's data stored in the cloud.

Bibliography

[1] TCG published, *Trusted Computing Platform Alliance (TCPA),* Main Specification, Version 1.1b, 2003.

[2] http://www.trustedcomputinggroup.org/about_tcg

[3] Sundeep Bajikar, *Trusted Platform Module (TPM) based Security on Notebook PCs*- White Paper, Mobile Platforms Group, Intel Corporation, June 20, 2002.

[4] TCG Published, *TPM Main, Part 1, Design Principles*, Specification Version 1.2, Level 2, Revision 103, 9 July, 2007.

[5] http://www.searchcloudcomputing.com

[6] Michael Miller, *Cloud Computing : Web-Based Applications That Change the Way You Work and Collaborate Online*. Published Aug 11, 2008 by Que. First Edition.

[7] Jian He, Mingdi Xu, *Research on Storage Security Based on Trusted Computing Platform,* Proceedings of the 2008 International Symposium on Electronic Commerce and Security, 2008, IEEE Computer Society.

[8] http://technet.microsoft.com/en-us/library/cc700811.aspx

[9] http://www.tar.hu/wininternals/ch12lev1sec8.html

[10] Peter Mell and Tim Grance, *The NIST Definition of Cloud Computing,* Version 15, 10-7-09.

[11] http://www.trustedcomputinggroup.org

[12] Reiner Sailer, Leendert van Doorn, James P. Ward, *The Role of TPM in Enterprise Security,* Datenschutz und Datensicherheit (DuD), September, 2004 (also IBM Research Report 23363).

[13] Nuno Santos, Krishna P. Gummadi and Rodrigo Rodrigues, *Towards Trusted Cloud Computing,* Proceedings of Hot Cloud, 2009.

Vita

Date of Birth: December 30, 1986

Place of Birth: Visakhapatnam, India

Education

Bachelor of Technology in Computer Science and Engineering, Andhra University, 2008.

Sushama Karumanchi