The Pennsylvania State University

The Graduate School

Computer Science and Engineering Department

ASYNCHRONOUS ATTESTATION SCHEME FOR PRESERVING THE INTEGRITY OF LONG-TERM DIGITAL ARCHIVES

A Thesis in

Computer Science and Engineering

by

Dhivakar Mani

© 2009 Dhivakar Mani

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2009

The thesis of Dhivakar Mani was reviewed and approved* by the following:

Trent Jaeger
Associate Professor
Computer Science and Engineering
Thesis Advisor
Chair of Committee

Patrick McDaniel
Associate Professor
Computer Science and Engineering

Raj Acharya
Professor and Head of Department
Computer Science and Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Increasing number of organizations want to retain data like customer records, business application data, e-mail and databases, for well over 50 or 100 years. In most cases the lifespan of the storage devices and the applications are far less than the perceived time of data retention. The archived data may have to be physically and logically migrated from one device or format to another at intermediate points in time to prevent data corruption. In this report the challenge of asserting the integrity of the archived data in the future even in the absence of the originator of the data is discussed. The Trusted Platform Module along with a Trusted Time Server is used to attest and verify the integrity of an archive tool running in the user domain of a Xen Virtual Machine. The root of trust installation is followed for the root domain (Dom-0) of the Xen Virtual Machine Monitor and a virtual Trusted Platform Module (vTPM) is used to attest the integrity of the system state of the user domain (Dom-U) created in the Dom-0. A trusted Time Server is used to create time-stamped Dom-U attestations. The attestations provide an integrity proof of the system state of Dom-U and Dom-0 by including the PCR values of the vTPM and the integrity state of the archive by including the Merkle hash tree of the file-system that was archived. An archive manager is used to manage the security information of the archives. The archive manager does a verifiable, periodic re-keying of the content proofs with a new signing key pair. It also generates proofs from the content proofs in response to a challenge from a remote verifier. The proofs can be used to verify the integrity of the system state of Dom-U and Dom-0, the archive tool and the archive content over a period of time. The protocol of the archive process, the archive manager and the trusted time server is detailed and analyzed in this report. The proposed protocol will enable the archive manager to prove the integrity of archive content proofs (which were created years ago) to a remote third party.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to my thesis advisor Dr. Trent Jaeger, for his valuable suggestion, guidance and constant encouragement. I sincerely thank Dr. Patrick McDaniel for his support and encouragement. I sincerely thank Thomas Moyer, Joshua Schiffman, Hayawardh Vijayakumar, Sandra Rueda and Divya Muthukumaran for their guidance and encouragement.

I thank my parents, sister, brother-in-law and my friends who constantly encouraged me throughout my masters at the Academy.

Chapter 1

Introduction

In the last 50 years, computer systems and information automation have moved work processes and records online which results in enormous amount of digital resources. Digital collections are vast, heterogeneous, and are growing at a rate that outpaces the ability to manage and preserve them. Digital resources like scientific databases, medical records and government statistics are accumulated over long periods of time at considerable expense. Many of the digital resources that are created today will be re-purposed and re-used in the future for various reasons. One unique aspect of digital preservation is the aspect of long term, where long term may mean long enough to be concerned about the obsolescence of technology or may mean decades or centuries. When long-term preservation spans several decades, generations, or centuries, the threat of interrupted management of digital objects becomes critical. Unlike many physical objects that can withstand some period of neglect without resulting in total loss, digital objects require constant maintenance. Redundancy, replication, and security against intentional attacks on archival systems and against technological failures are critical requirements for long-term preservation.

Long Term Archiving Requirements

For a digital archive system to effectively monitor the content of the archives and to assess its preservation needs, the archive system must know as much as possible about the technical and functional characteristics of its digital archive objects and record that information as metadata. Digital archiving systems should maintain the archived contents, extract metadata of

the contents, restructure and manage metadata over time. It will be essential for future users of archived materials to recover and relate the metadata schema used when the entity was created. Likewise, managing the identity of preserved digital objects over time is a challenge for digital archives because the identifiers assigned to digital objects can be changed easily and the technologies for naming and tracking digital objects evolve over time. These requirements trigger the development of methods for unique and persistent naming of archived digital objects, tools for certification and authentication of the integrity of the preserved digital objects, methods for version control, and interoperability among naming mechanisms used by different content providers. Tools are also needed to automatically transform preserved digital objects from obsolescing to contemporary into the formats, standards, and data models and to document the effects of these transformations.

Long Term Archiving Driving Factors

In 2007, the Storage Networking Industry Association (SNIA) completed a comprehensive survey involving hundreds of individuals in a wide variety of organizations from countries around the world [1]. An overwhelming 53% of the respondents said they have information that must be retained permanently and 83% said over 50 years [1]. The Figure 1-1 shows that the long-term retention needs are real. The survey also says that most archives are less than 5 TB but 18% of the archives are over 100TB, as shown in Figure 1-2. The type of data includes databases, custom business application data, customer records and e-mail. The time to retain information far exceeds the typical lifespan of storage systems like disk or tape, and applications. Even the physical media start to degrade and may become unreadable long before the retention period expires. The current practice is to migrate data both physically and logically, every 3 to 5 years. Physical migration requires moving information from one physical storage

system to another or from one media format to another to maintain physical readability, accessibility, and integrity. Factors triggering physical migration include media failure, media or storage system obsolescence and system changes. Logical migration requires moving information from one logical format to another—such as from an old version of an application to a new version—to preserve readability and interpretability. Factors triggering logical migration include changing application formats and obsolete applications. Inhibitors to both types of migrations include cost, complexity, sheer volume of information, and lack of time.

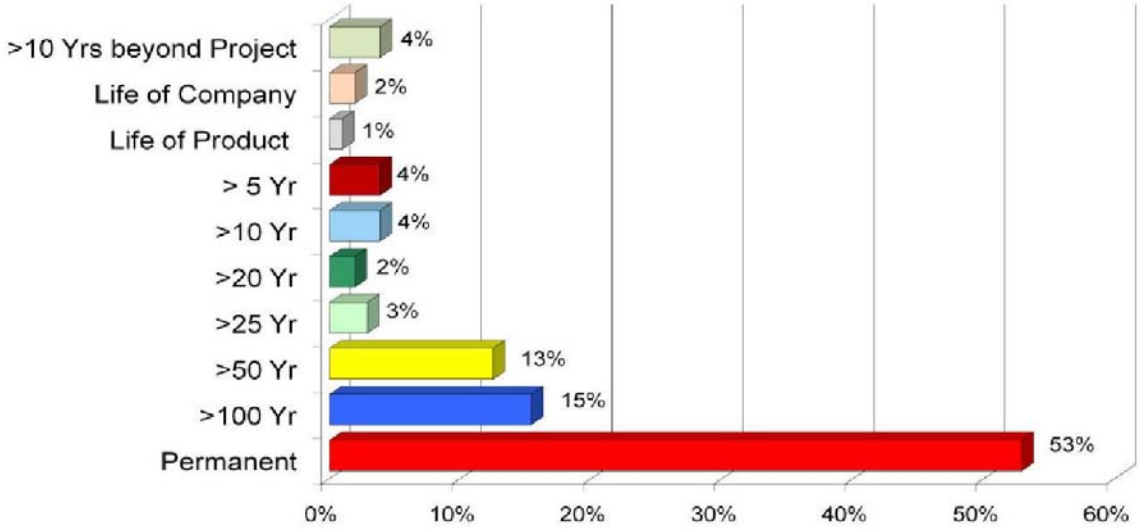


Figure **1-1**: Longest Retention Requirement. Source [1].

Long Term Archiving Challenges

The long-term archiving has the following inherent risks and its associated failures. Systems must expect that all storage media degrade with time, causing irrecoverable bit errors, and to be subject to sudden catastrophic irrecoverable loss of bulk data such as disk crashes or loss of off-line media. It should also be expected that all hardware components suffer transient recoverable failures, such as power loss, and catastrophic irrecoverable failures, such as burnt-out

power supplies. Most of the software components suffer from bugs that pose a risk to the stored data. Systems cannot assume that the network transfers they use to ingest or disseminate content will either succeed or fail within a specified time period, or will actually deliver the content unaltered. Systems must anticipate that the external network services they use, including resolvers such as those for domain names and persistent URLs, will suffer both transient and irrecoverable failures both of the network services and of individual entries in them.

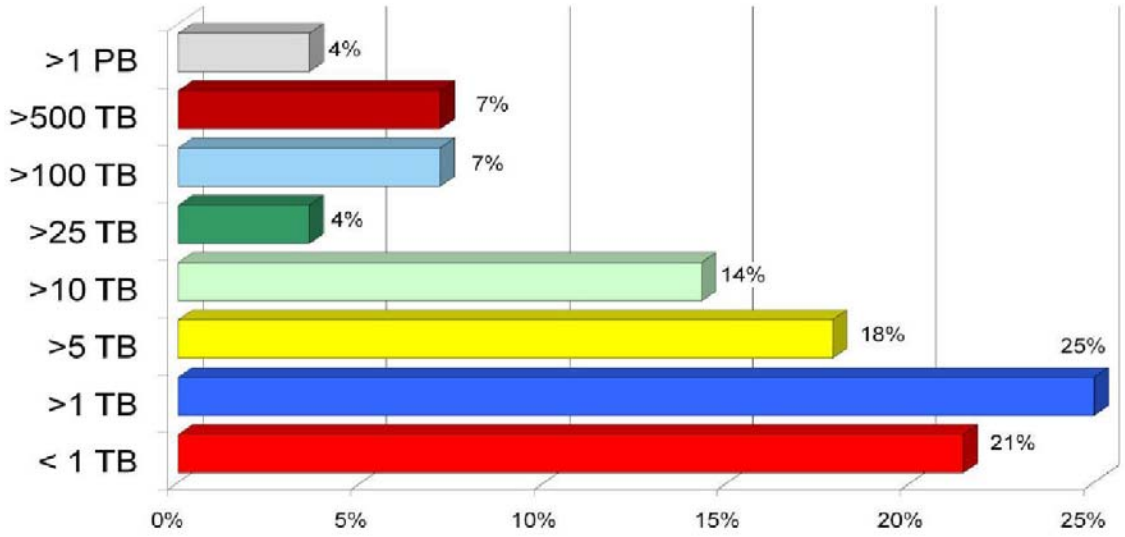


Figure **1**-**2**: Size of long-term archives. Source [1].

All media and hardware components will eventually fail. Before that, they may become obsolete in the sense of no longer being capable of communicating with other system components or being replaced when they do fail. This problem is particularly acute for removable media, which have a long history of remaining theoretically readable if only a suitable reader could be found. Software components will become obsolete. This will often be manifested as format obsolescence when, although the bits in which some data was encoded remain accessible, the information can no longer be decoded from the storage format into a legible form. Operator actions must be expected to include both recoverable and irrecoverable errors. This applies not merely to the digital

preservation application itself, but also to the operating system on which it is running, the other applications sharing the same environment, the hardware underlying them, and the network through which they communicate. Natural disasters, such as flood, fire and earthquake must be anticipated. All systems connected to public networks are vulnerable to viruses and worms. Digital preservation systems must either defend against the inevitable attacks, or be completely isolated from external networks. Much abuse of computer systems involves insiders, those who have authorized access to the system. Even if a digital preservation system is completely isolated from external networks, it must anticipate insider abuse.

Commitments of a Long Term Archiving Scheme

A long-term archiving scheme has to guarantee three key principals, namely, security, transparency and proof. The access to the archived data must be guarded by enforcing custom security policies. Archived data should be stored in durable devices with schemes that can prevent undetectable degradation or corruption. The risk of unrecoverable loss or corruption can be prevented by using redundancy and secured distributed storage. An archive scheme should be able to make its methods, processes, technology, business mechanisms, and public statistics transparent to any verifier who wants to verify the integrity of the archived data. The archive scheme should also ensure techniques that can provide a proof of integrity of the archived data to any third party. More generally, the archive scheme must be able to build trust over time by providing trusted information about the security of the archived data transparently to any interested and trusted third party. This report specifically focuses on the problem of providing the proof of integrity of the archived data that is expected to have a lifespan of more than 50 or 100 years.

Chapter 2

Related Work

Many papers propose schemes to tackle long-term archival problems. Ganger et al. present PASIS [28], a survivable storage based on decentralized architecture. It uses data distribution and redundancy schemes to ensure fault tolerance and to protect integrity and confidentiality of the documents by forcing the attacker to compromise several nodes in order to become a real threat. Another approach based on distributed storage, SafeStore [14], achieves data durability by combining replication across different publicly available Storage Service Providers. An efficient audit protocol is provided to check that the integrity of the stored documents is preserved over time. POTSHARDS [17] is a distributed scheme where secrets are not replicated but split into shares and disseminated through different machines. A file can be recovered by recovering a portion of the shares that allows the original file's recovery. A user can only recover the file is he or she knows the correct combination of shares. It also uses 'approximate pointers' to allow data recovery even when the key is no longer available. Another approach, the LOCKSS [16], is a peer-to-peer system where documents are replicated over peers. These peers cooperate to detect and repair damage to their content by majority voting. But in this approach, the main goal is availability of content in the future, even if little modifications have happened to the documents. Other systems focus on dealing with the future obsolescence of software and hardware, of how to store and migrate data such that it is readable well into the future. The Public Record Office Victoria propose a system called 'VERS' [27] that uses XML encapsulation to include metadata together with the stored document, which stores enough information to read the document in the future even if the software or hardware which was used to create the document is no longer available. This system uses digital signatures to preserve the integrity of a document but does not provide a method to confirm the validity of these signatures

over time. Other papers [25, 26] address the obsolescence of cryptography when dealing with digital signatures validation far in time (including the fact that the public key certificates used may be invalid or no longer available at the time of validation). In these approaches, a digitally signed document is stored in a Secure Long-Term Archival System (SLTAS) which uses timestamps in order to protect the validity of the initial signature over time. The archive verifies the signatures at regular intervals, and, if they are valid, re-timestamps them. This process serves to account for any weakness that may have appeared in the signing algorithms, under the reasonable assumption that a Time Stamping Authority will always use a non-broken state-of-the-art algorithm to issue timestamps. The system proposed in [5] enables the use of the TPM to tie the web server integrity state to the web content delivered to browsers. An asynchronous usage model is proposed to remove the TPM from the critical path of serving content to users. The web server creates request-independent attestations by combining the time with a hash tree of the served content. The system protects the web server from several types of threats including root-kits and malicious patches through the use of integrity measurement [7].

Chapter 3

Background Study

Trusted Platform Module

The Trusted Computing Group (TCG) was formed in 2003 to develop and support open industry specifications for trusted computing across multiple platform types. The TCG issued a specification for a Trusted Platform Module (TPM), which is a dedicated security chip designed to enhance software security. The TPM is realized as a hardware chip attached to the motherboard which can be used to securely store confidential information, such as private keys. It also can be used to store signatures (hash values) of software running on the computer, allowing non-allowed software (such as viruses) to be rejected.

TPM Background

The Figure 3-1 presents the main components of the TPM such as the Random Number Generator (used for generating asymmetric as well as symmetric keys and nonce that provide freshness), Platform Configuration Registers (PCRs), Secure Hash Algorithm (provides SHA-1 functionality), RSA key generator and Hash Message Authentication Code (HMAC). The Endorsement Key (EK) is a pair of RSA keys that is installed when the TPM is manufactured. The public EK value is used to uniquely identify a TPM and will not change during the TPM's lifetime. The Storage Root Key (SRK) is also a pair of RSA keys that is used to encrypt other keys stored outside the TPM. SRK is in effect the Root of Trust for Storage. SRK can change when a new user takes ownership of the TPM. The TPM contains a number (at least 16) of Platform Configuration Registers (PCRs), essentially internal memory slots. At boot time, all

PCRs are initialized to a known value (0 for PCRs 1–16 and -1 for PCRs 17–22). They are used to store platform configuration measurements. These measurements are normally hash values (SHA-1) of entities (applications) running on the platform. PCRs cannot be written directly; data is stored by a process called extending the PCR. The only way for software to change the value of a PCR is by invoking the TPM operation:

$$\text{PCRExtend}(\text{index}, \text{data})$$

When this operation is invoked on the TPM, it updates the value of the PCR indicated by index with a SHA-1 hash (H) of the previous value of that PCR concatenated with the data provided. In other words, the TPM performs the following update:

$$\text{PCRindex} \leftarrow H(\text{PCRindex} \| \text{data})$$

Extend operation relies on the infeasibility of finding two different measurement values such that when extended returns the same value. It preserves the order in which entities measurement were extended and allows an unlimited number of measurement to be stored in a PCR because the result is always a 160-bit value.

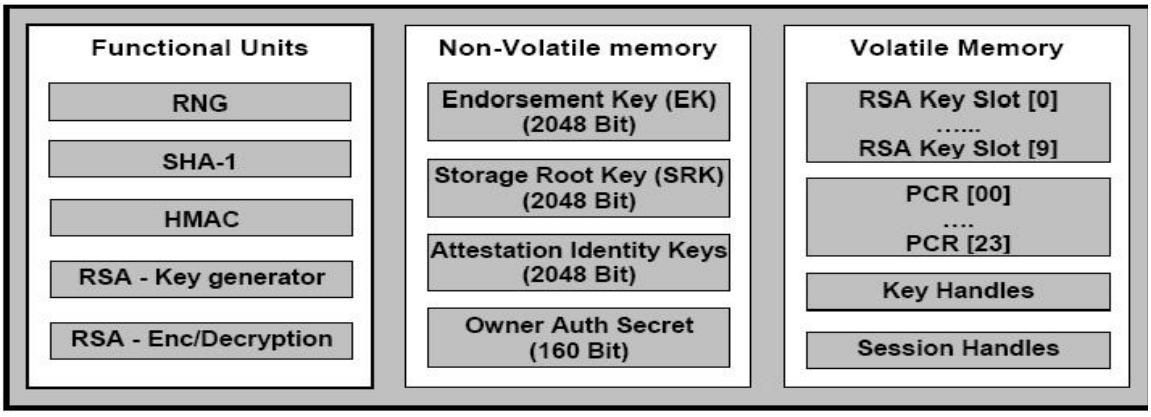


Figure **3-1**: TPM Architecture. Source [4].

Core Root of Trust Measurement

Core Root of Trust Measurement (CRTM) is either the BIOS boot block or the entire BIOS. At boot up, the CRTM measures the integrity metrics that show the software state, for e.g. the master boot record, BIOS and the code from the other firmware. CRTM [6] measures these metrics as hash of the current state of the software in terms of version, patch level and extends a particular PCR of the TPM. The whole process of measurement is done in a chain of trust manner, i.e. the CRTM initially measures itself, and reports to the TPM. Then it would move up the hierarchy and measure the BIOS and report the hash to the TPM. Then the BIOS loads the boot loader and boot loader, in turn, measures the Operating System (OS). OS then has the access to the TPM to report the software modifications anytime. So, suppose a pirated version of software was running on the machine, then the OS (in the trusted zone) would report that to the TPM. Thus PCRs in the TPM are used to store the sequence of measurement values. The TPM provides reporting of PCR values through the quote operation. To prevent replay of the measurement, the requesting party issues a 160-bit random nonce to the attesting system, creating a challenge. The TPM has a Storage Root Key stored inside it, which only it knows. It uses this key to generate an Attestation Identity Key (AIK), which comprises an RSA key pair, the public portion of which (AIKpub) is available through a key management infrastructure. The TPM loads the private portion of the AIK pair (AIKpriv) and performs the Quote function, where it signs a message containing the values of one or more PCRs and the nonce with AIKpriv. The TPM securely transports the result of the Quote function along with their respective logs to the requesting party. The attesting party can then verify the integrity of the message using AIKpub, and subsequently, every element of the measurement list up to the value stored in the PCR may be validated. If the configuration of the platform has changed as a result of unauthorized activities

then access to data and secrets can be denied and sealed. Accordingly the requesting party will make its decision to carry out the communication in a trusted or a non trusted environment.

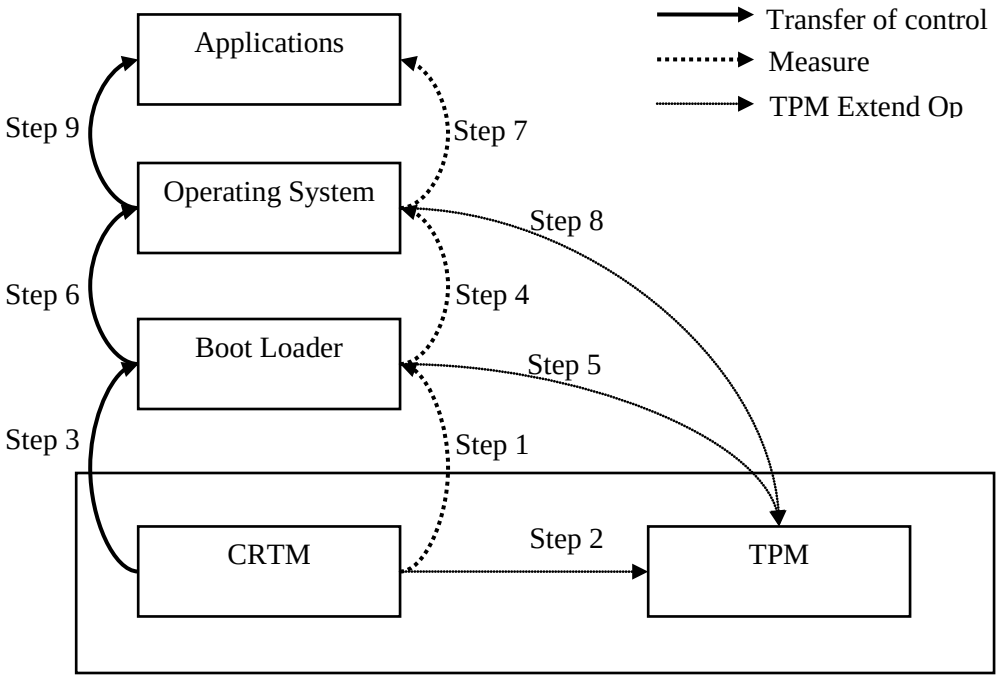


Figure **3-2**: Chain of trust measurement using TPM Extend operation.

Remote Attestation Protocol

Remote Attestation (RA) is a method to prove to a remote party that the local PC is a trusted platform (TPM-enabled) and to show its current configuration. The remote party needs to trust the host to reliably measure and report its configuration. On receipt of a request for attestation, the attestor generates a public/private key pair, called the attestation identity key (AIK), and send the public part of AIK signed by the EK to a trusted third party (TTP) called a

Privacy CA. The Privacy CA checks the EK's signature and status on the revocation list, and signs the AIK. The remote computer just sees the AIK signed by the Privacy CA, and thus cannot link it with the EK. Different sessions will use different AIKs, so they cannot be linked either. The host can now send its PCR values (signed with AIK), Stored Measurement Log (SML) and the received AIK certificate to the challenger. The challenger does the following steps to ensure the trust of the host.

1. Verifies the AIK certificate with the TTP public key.
2. Uses AIK to verify the signature on the PCR values.
3. Recalculates the value from the measurement list within SML.
4. Compares the calculated value with PCR's value. If the PCR value and SML do not match, it implies that the SML had been tampered, and the verifier will not to trust the host. If they do match, the verifier goes through the fingerprint list in SML and looks for any unapproved entity.

Virtual Machine Monitor

Virtualization separates an operating system from the underlying platform resources. Traditionally, the software is bound to the hardware, allowing better utilization of the resources at hand, but on the other hand creating compatibility issues and limitations. The virtualization approach allows higher compatibility and even independence of the software on the hardware running it. Virtualization achieves the separation by creating consistent interfaces, implemented and used differently depending upon the hardware and the software. There are different types of virtualization such as the Software Virtualization and the Resource virtualization. The main Software virtualization types are Application virtualization (a single running program is wrapped by a layer providing it with additional compatibility to the running environment), OS-level

virtualization (a single kernel running otherwise isolated environments) and the Virtual Machines, which are Hardware level virtualization - totally isolated environments running in parallel on one machine. The two main approaches in Virtual Machine implementation are Full Virtualization and Para-Virtualization. Full Virtualization is a fully-simulated hardware set (including every real life component in a software form) running a "guest OS" in a closed shell, where the guest has no way of knowing it. Para-virtualization is a relatively thin layer between each VM and the hardware below, which tries to minimize the extent of intervention in running processes while maintaining the integrity (and isolation) of each VM- both from the hardware and from the other VMs running on the machine.

Xen Virtual Machine Monitor

Xen is an open-source software project that provides high-performance, resource-managed virtualization on the x86 processor architecture. It allows multiple operating system instances to run concurrently on a single physical computer. It incorporates the principles of Para-virtualization to create a VMM (Virtual Machine monitor) which is a thin layer between software and hardware, allowing the interaction of the two, in the case of more than one guest OS concurrently running. Each Xen system has a single privileged OS, called Domain-0 that is responsible for starting and managing the other unprivileged OS instances. Domain-0 is the OS that boots when the system starts and it has the tools necessary to manage other domains. Xen manages the computer's hardware resources so they are shared effectively among the operating system instances, called domains. Xen shows good performance and isolation of each VM, while featuring unprecedented options like resource control and live migration. Xen virtualization provides many exciting benefits over traditional single OS computers, including server consolidation, application mobility, secure computing, and research/testing.

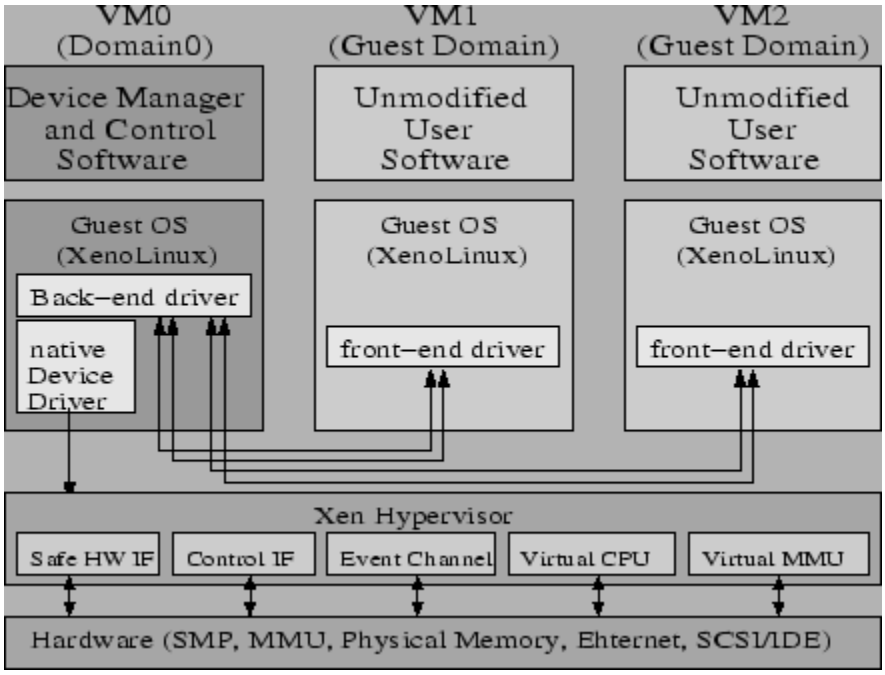


Figure **3-3**: Xen Virtualization Architecture Overview. Source [11].

Virtual Trusted Platform Module

The main goal described in [12] is to blend the two technologies of Virtualization and Trusted Computing or specifically the Trusted Platform Module. These two technologies can ensure the complementary requirements of Virtualization for the high availability, the integrity and the isolation of each virtual machine and TPM for the security, the chain of trust and the remote attestation. Virtualizing the TPM is required to provide TCG services in the virtual machines. The virtualization base system should handle the TPM device for its usage and export to each virtual machine a TPM emulated device to extend the chain of trust. So each virtual machine will be able to use the cryptographic resources, store secret objects and realize remote attestations. The first requirement is then to ensure the same level security provided by the hardware TPM for the virtualized TPMs. Moreover, new requirements are introduced due to the specificities of a virtualized architecture. The vTPM implementation is composed of a vTPM

Manager which manages the hardware TPM, provide services to manages multiple TPM emulated devices and a vTPM instance for each virtualized host, which implements the full TCG TPM 1.2 specification.

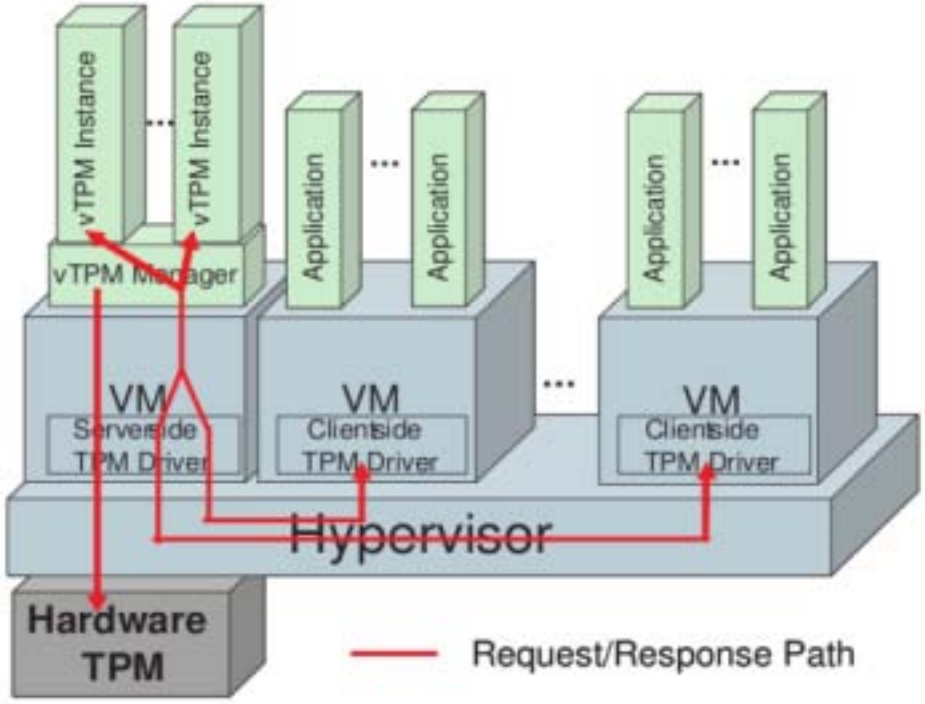


Figure **3-4**: Virtual Trusted Platform Module (vTPM) Architecture. Source [12].

A vTPM instance is the TPM of a VM. It implements the full TCG TPM Specification version 1.2. Each VM has its associated vTPM instance running throughout the lifetime of the VM, so there as much vTPM instances as there is VMs running. A vTPM instance associated to a VM is unique. The vTPM implementation in Xen is software-based, so a vTPM instance is just a piece of software running in the Dom0. The vTPM manager creates and manages vTPM instances. When a VM is created, it will spawn a vTPM instance that will be associated to this VM. When running a paravirtualized DomU, the vTPM manager also redirects the TPM commands from the DomU (by listening to the Back-End, /dev/vtpm) to the associated vTPM instance. To make TPM

functionality available to a paravirtualized DomU, Xen uses the split-driver model. So the vTPM driver is split in two parts as shown in Figure 3-4.

1. The Front-End (FE): FE is the client side part of the vTPM driver that runs on the DomU. It exposes the /dev/tpm device file on the DomU to receive TPM commands and it will issue these commands to the backend. On Linux, the driver module is called tpm_xenu.
2. The Back-End (BE): BE is the server side part of the TPM driver that runs on the Dom0. It exposes the /dev/vtpm device file on the Dom0 so that the vTPM manager can process the TPM commands. On Linux, the driver module is called tpmbk.

This driver is based on the Xen network driver. Data exchange between the FE and the BE is ensured by the XenBus which provides an API to use grant tables (a single shared memory ring) and an event channel for asynchronous notifications of activity. The back-end prepends a 4-byte vTPM instance identifier to each TPM commands to identify the vTPM instance of the VM. The identifier is prepended in the BE so the VMs cannot forge commands and send them to another vTPM instance. The commands are multiplexed inside the character device file /dev/vtpm. This special file will be read by the vTPM manager which will redirect the command to the proper vTPM instance. Each vTPM instance has a Storage Root Key (SRK) as root for its key hierarchy and an Endorsement Key (EK). To allow instance and vTPM migration, these keys are unlinked from the key hierarchy of a TPM hardware component. This also allows faster key management and cryptographic operations. However, if the SRK, EK and other data of virtual TPM are stored in a persistent storage, they must be stored encrypted with a key stored in the hardware TPM device. This symmetric key must be sealed or protected with a password. The trust in the VM is trustable only if the trust in the environment (TCB, hypervisor...) is guaranteed. For this reason the chain of trust must be guaranteed from the hardware TPM to the vTPM, from the TCB to the VMs. So, the architecture in [7] provides the vTPM PCRs a merged version of the measures. A lower set of PCRs in the vTPM shows measures from the hardware TPM and the upper the

measures for the VM, as shown in the Figure 3-5. By this way, a challenger can see all relevant measurements during a remote attestation. But in the vTPM implementation, there is no PCRs mapping between the vTPM and the hardware TPM (the PCRs 0 to 8 are not the same in the vTPM and in the hardware TPM) because there is disagreement on how to do the signatures for quotes correctly. The vTPM would sign the complete quote, but it does not own the mapped PCRs which is a problem.

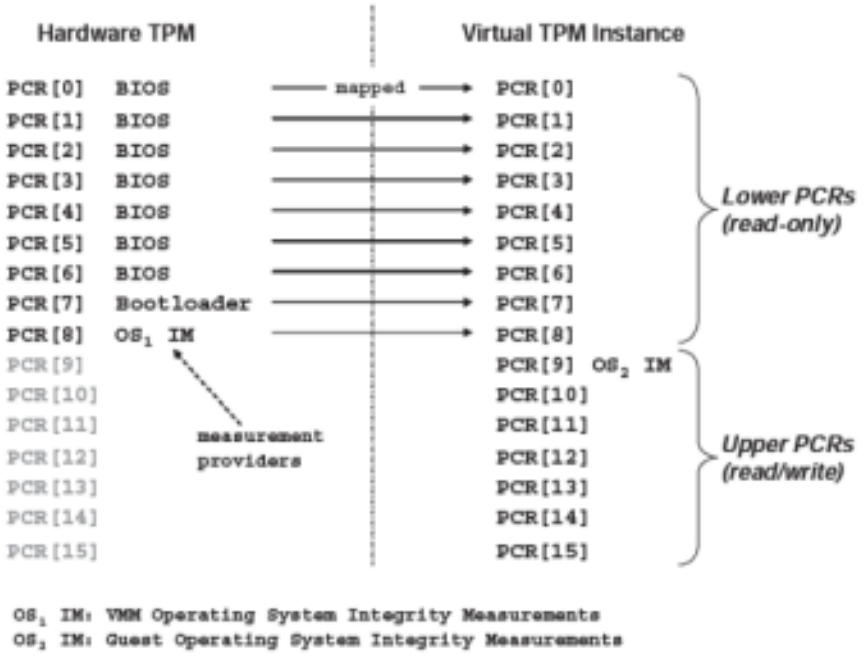


Figure **3-5**: Extension of lower PCR values of vTPM. Source [12].

Asymmetric Key Cryptography

Whitfield Diffie and Martin Hellman introduced the concept of public-key cryptography in 1976. In their system, each person gets a pair of keys, one called the public key and the other called the private key. The public key is published, while the private key is kept secret. The need for the sender and receiver to share secret information is eliminated; all communications involve

only public keys, and no private key is ever transmitted or shared. In this system, it is no longer necessary to trust the security of some means of communications. The only requirement is that public keys be associated with their users in a trusted (authenticated) manner. Anyone can send a confidential message by just using public information, but the message can only be decrypted with a private key, which is in the sole possession of the intended recipient. Furthermore, public-key cryptography can be used not only for privacy (encryption), but also for authentication (digital signatures) and other various techniques.

Public-key Encryption

In cryptography, encryption is the process of obscuring information to make it unreadable without special knowledge. For example, When Alice wishes to send a secret message to Bob, she looks up Bob's public key in a directory, uses it to encrypt the message and sends it off. Bob then uses his private key to decrypt the message and read it. No one listening in can decrypt the message. Anyone can send an encrypted message to Bob, but only Bob can read it (because only Bob knows Bob's private key). Encryption can be used to ensure secrecy, but other techniques are still needed to make communications secure, particularly to verify the integrity and authenticity of a message; for example, a message authentication code (MAC) or digital signatures. Another consideration is protection against traffic analysis.

Digital Signatures

A digital signature method generally defines two complementary algorithms, one for signing and the other for verification, and the output of the signing process is also called a digital signature. Digital signature schemes rely on public-key cryptography. In public-key

cryptography, each user has a pair of keys: one public and one private. The public key is distributed freely, but the private key is kept secret and confidential; another requirement is that it should be infeasible to derive the private key from the public key. A general digital signature scheme consists of three algorithms, namely, a key generation algorithm, a signing algorithm and a verification algorithm. For example, consider the situation in which Bob sends a message to Alice and wants to be able to prove it came from him. Bob sends his message to Alice and attaches a digital signature. The digital signature is generated using Bob's private key, and takes the form of a simple numerical value (normally represented as a string of binary digits). On receipt, Alice can then check whether the message really came from Bob by running the verification algorithm on the message together with the signature and Bob's public key. If they match, then Alice can be confident that the message really was from Bob, because the signing algorithm is designed so that it is very difficult to forge a signature to match a given message (unless one has knowledge of the private key, which Bob has kept secret). For efficiency reasons, Bob first applies a cryptographic hash function to the message before signing. This makes the signature much shorter and thus saves time since hashing is generally much faster than signing in implementations. However, if the message digest algorithm is insecure (for example, if it is possible to generate hash collisions), then it might be feasible to forge digital signatures.

Aggregate Signature Scheme

In a general signature aggregation scheme, if user i signs the message Mi to obtain a signature σi, then anyone can use a public aggregation algorithm to take all n signatures $\sigma 1,...,\sigma n$ and compress them into a single signature σ. Moreover, the aggregation can be performed incrementally—signatures $\sigma 1,\sigma 2$ can be aggregated into $\sigma 12$ which can then be further aggregated with $\sigma 3$ to obtain $\sigma 123$, and so on. There is also an aggregate verification algorithm

that takes PK1,...,PKn; M1,...,Mn, and σ and decides whether the aggregate signature is valid on the given messages under the given keys. Thus, an aggregate signature provides non-repudiation at once on many different messages by many users [29]. This mechanism is referred to as general aggregation since aggregation can be done by anyone and without the cooperation of the signers. The general aggregate signature scheme due to Boneh, Gentry, Lynn, and Shacham [31] uses bilinear maps from algebraic geometry.

Sequential Aggregate Signature Scheme

Sequential aggregate signatures are a variant of aggregate signatures. In a sequential aggregate signature scheme, signatures are not individually generated and then combined into an aggregate. Rather, a would-be signer transforms a sequential aggregate into another that includes a signature on a message of his choice. Signing and aggregation are a single operation. Sequential aggregate signatures are built in layers, like an onion; thefirst signature in the aggregate is the inmost. As with general aggregate signatures, the resulting sequential aggregate is the same length as an ordinary signature. For sequential aggregate signatures, aggregation and signing are performed in a single combined operation. The operation takes as input a private key SK, a message Mi to sign, and a sequential aggregate signature σ0 on messages M1,...,Mi-1 under respective public keys PK1,...,PKi-1, where M1 is the inmost message. It adds a signature on Mi under SK to the aggregate, outputting a sequential aggregate σ on all i messages M1,...,Mi. The aggregate verification algorithm, given a sequential aggregate signature σ, messages M1,...,Mi, and public keys PK1,...,Pki, verifies that σ is a valid sequential aggregate (with M1 inmost) on the given messages under the given keys [29].

Forward Secure Signature Scheme

In a forward secure signature scheme, a user registers a public key PK and keeps private the corresponding secret key SK0. The time during which the public key PK is desired to be valid is divided into T periods from 1 to t. While the public key stays fixed, the user evolves the secret key with time. Thus in each period, the user produces signatures using a different signing key: SK1 in period 1, SK2 in period2 and SKt in period t. The secret key in period i is derived as a function of the one in the previous period; specifically, when period i begins, a public one-way function h is applied to SKi-1 to get SKi [28]. At that point, the private key SKi-1 is deleted. An attacker breaking in during period i will get the key SKi, but not the previous keys SK0,…, SKi-1, since they have been deleted [28]. A digital signature employing the forward secure signature scheme always includes the value j of the time period during which it was produced. The verification algorithm takes in the public key PK, a message and candidate signature, and verifies that the signature is valid, i.e., it was produced by the legitimate user in the period indicated in the signature [28]. Although the user's secret key evolves with time, the public key stays fixed throughout, so that the signature verification process is unchanged, as are the public key certification and management process.

Chapter 4

Design and Implementation

The design to achieve long-term integrity of archives relies on using digital signatures and time-stamped attestations. The system must be able to provide proof of the integrity of archives and validity of the signatures in a distant future. A third party should be able to verify the integrity of a file or an entire archive. The design consists of the following components:

1. The archive process X running in the Dom-U (user domain) of a trusted machine H1 which is used to archive file systems from one device to another device.
2. The archive manager process M running in a trusted machine H2 which has access to the attestations generated by the archive process X.
3. The verifier process V running in a remote machine H3 which is interested in verifying the integrity of a particular archive or of a particular file contained in the archive.
4. A Trusted Time Server process running in a trusted Machine TS with a TPM.

The requirement of long-term archiving drives the assumption that the archive process X and/or the machine H1 may not be accessible or even present at the time when the verifier needs to verify the integrity of the archived data. The attestations generated by process X using the vTPM in machine H1 are securely stored in the trusted machine H2. At a later point of time in the future, the verifier process V which has access to the archived content, challenges the manager M with the unique ID of the archive whose integrity it wants to verify. The archive manager M retrieves the stored attestations corresponding to the requested archive ID and generates succinct proofs in response to the challenge from the remote verifier process V.

Trusted Time Server

The architecture depends on a trusted Time Server which issues a digitally signed current time that can be remotely verified by any third party. This verifiable timestamp is used along with the attestation of the root file system to provide a proof of the existence of the signed data at that moment without the possibility of post-dating or pre-dating. The timestamp is used to provide evidence of the archival time which is the time when the backup was taken. It can also be used to verify and refresh the validity of signatures later in time. Logical and physical migration of the archive, re-timestamping the archive will benefit in determining the connection of an archive to its creation time.

Design Architecture

A host machine 'H1' is assumed to run with a TPM_{H1}, TrustedGrub, IMA and Xen Virtual Machine Monitor. The measurements of the BIOS, MBR, bootloader, Linux kernel, initrd, modules and loaded files are assumed to be in the file /sys/kernel/security/ima/ascii_runtime_ measurements and are extended in the respective PCRs in the TPM_{H1}. Thus the Dom-0 of the host machine 'H1' follows the Core Root of Trust Measurement as explained in Figure 3-2 and in Section 3-1. A process X is assumed to run in the user domain of Xen Virtual Machine Monitor with a virtual Trusted Platform Module, $vTPM_{H1}$. A lower set of PCRs of the $vTPM_{H1}$ has the measurements from the hardware TPM_{H1} and the upper PCRs contains the measurements for the VM, as shown in the Figure 3-5. The extension of the lower set of PCRs of the $vTPM_{H1}$ ensures that the dom-0 Core Root of Trust Measurement is chained to the trust measurement of the VM. The process X is used to archive data or the file system from one device to another. The process X generates vTPM Quotes, as explained later in this document, for each of the archive operation

it does. The machine H1 and the process X is assumed to have existed years ago and they may not be present or accessible to the user currently. Only the XML format of security information of archived data like the hash trees, vTPM Quotes and the measurement lists of H1 are signed and saved securely in machine H2 for later use. An architecture is proposed for remote verification by a third party of the integrity of a archive process X and the integrity of the archived data. The machine H2 is assumed to run with a TPM_{H2}, TrustedGrub and IMA. The archive manager process M manages the security information of archives that was generated by archive process. The manager will respond to challenge requests from a remote verifier. The verifier process V runs in the remote machine and challenges the manager process M for the integrity of a particular file or the entire file system that was copied by process X in host H1.

Protocol Conventions

The function h(d) denotes a cryptographic hash using SHA1 algorithm over some data d, and concatenation of different data elements is denoted as |. The quoting hosts are denoted H1 for the archive process, H2 for the manager process and H3 for the verifier process. PCR_i denotes the integrity state of host i. A TPM quote is denoted Quote(h, s, c), where h is the host identity performing the quote, s is the PCR state, and c is the quote challenge. The archive tool is assumed to copy the file system FSYS from one device to the other. The files in the file system are represented as f_i. Ttime is a time epoch returned from a hardware clock on the trusted time server. The protocols described below hashes the time quote t1 along with the data to be attested, so the hash h1 is constructed as h(time quote t1 | h(d)). The resulting hash h1 is then used a the quote challenge c. Thus the TPM quote Quote(i, s, c) of host i effectively ties the data to both the integrity state of host i and time t1. The Linux Integrity Measurement Architecture (IMA) is used for measuring the state of the code loaded in the system as mentioned in [6].

Asynchronous attestation protocol of the archiver

1. Each archive operation is considered as a transaction done by the process. After copying a file system 'FSYS' successfully, the process X creates a unique Transaction ID, Tid, representing the successful archive operation done.
2. Process X requests a quote from the $vTPM_{H1}$ with Tid as the nonce.
3. Q1 = Quote($vTPM_{H1}$, PCR_{H1}, Tid) is generated by the archive process X.
4. The process X requests the current time quote from a trusted Time Server 'TS'.
5. The Q2 = Quote(TPM_{TS}, PCR_{TS}, h(Ttime)) and Ttime is returned to the process X by the Time Server.
6. The content of (Q1 | Tid | Q2 | Ttime) is written in XML format to the root of the file system in a file named archive_info.xml and signed with h(Q1 | Tid | Q2 | Ttime).
7. Process X calculates the root hash of the file system that is archived MHT_{FSYS} = SHA1(FSYS). M1 is calculated as h(MHT_{FSYS} | Tid | Q2).
8. The Process X requests a quote from the $vTPM_{H1}$ with M1 as the nonce.
9. Q3 = Quote($vTPM_{H1}$, PCR_{H1}, M1) is returned to the process X, as shown in Figure 4-3.
10. An entry of (Tid, Q1, Q2, Q3) in XML format is appended to the log file Integrity_X.log
11. XML Format of the complete proof and related credentials is generated by archive process X for transaction Tid as shown in Figure 4-5. This XML is written to the Tid.dump file in the root of the archived file system. Quote(vTPMH1, PCRH1, h(tid | all entries in Tid.dump file)) is appended to the Tid.dump file. The Tid.dump file is called as the archive transaction dump file or the content proof file.
12. The steps from 1 to 11 are followed in sequence for each archive operation.

Depending on the interval set by the user, the archive process dumps all the files created till then to a secure storage. The archive transaction dump files(Tid.dump files) created for each archive operation are saved in a secure storage that will be accessible to the archive manager process M at a later point in time. The trusted, secure storage maintains a time line of the set of the Tid.dump files.

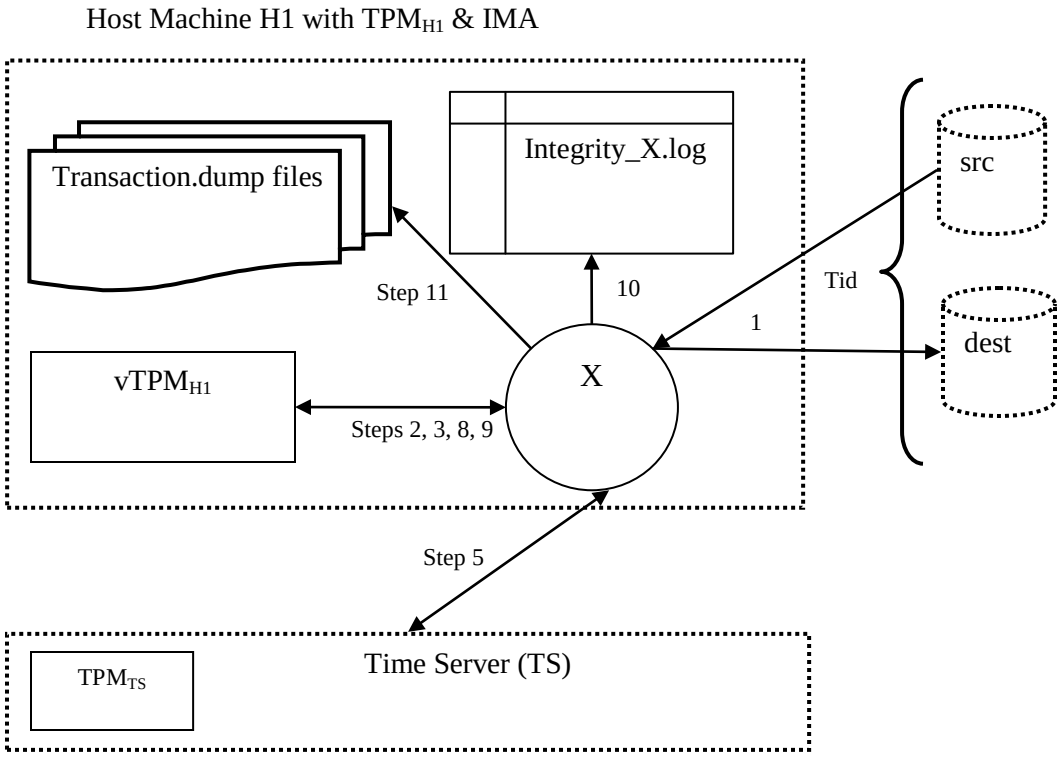


Figure **4-1**: Asynchronous attestation protocol of the archive process X.

The Merkle hash tree of the archived file system, FSYS, is generated as shown in the Figure 4-2. The files contained in the source file system are lexicographically sorted and a balanced complete binary tree is created with all the files as the leaf nodes of the tree. A SHA1(NULL) is taken as the hash of the null entries in the complete binary tree. The SHA1 hash of each file is created with the contents of the file and will include metadata information such as the inode number, timestamp, creation time, etc., depending on the custom user settings. The hash stored in the

binary tree leaves is calculated as h_i = SHA1(contents of file f_i | metadata of file f_i). The hash in the intermediate nodes of the binary tree is calculated as h_k = SHA1(h_i | h_j), where i, j are child nodes of node k. After the copy operation it is ensured by process X that h8 is equal to h'8. h'8 is taken to be the SHA1(FSYS) or the root hash of the file system MHT_{FSYS}.

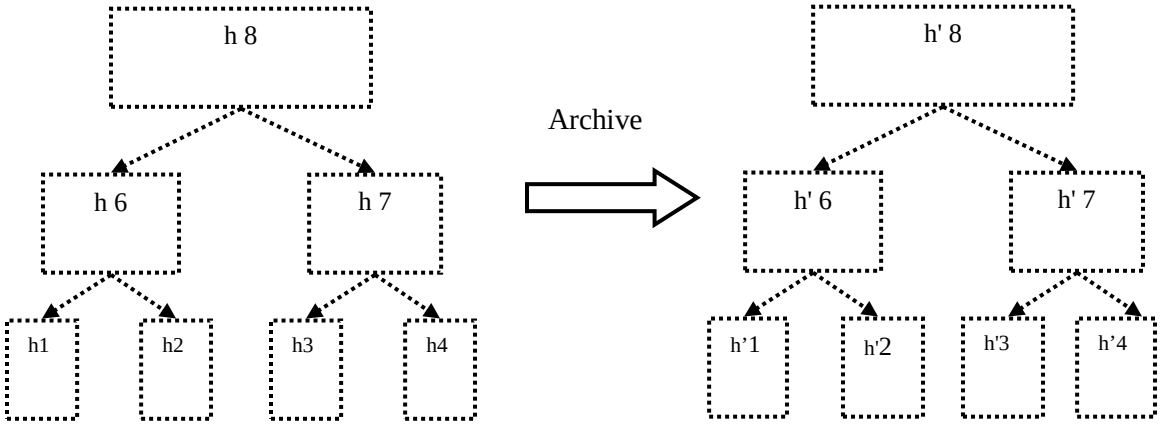


Figure **4-2**: Merkle Hash Tree of the archived File System with transaction ID, Tid.

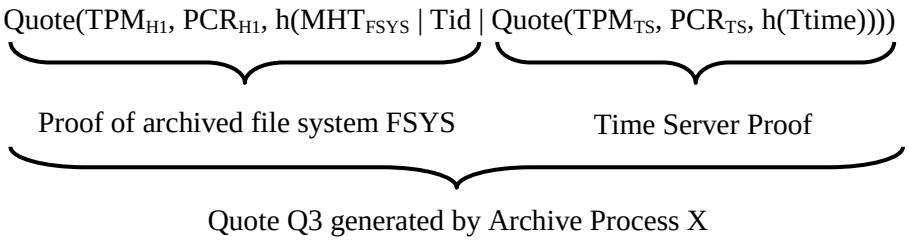


Figure **4-3**: Quote Q3 generated by the archive process X for transaction ID, Tid.

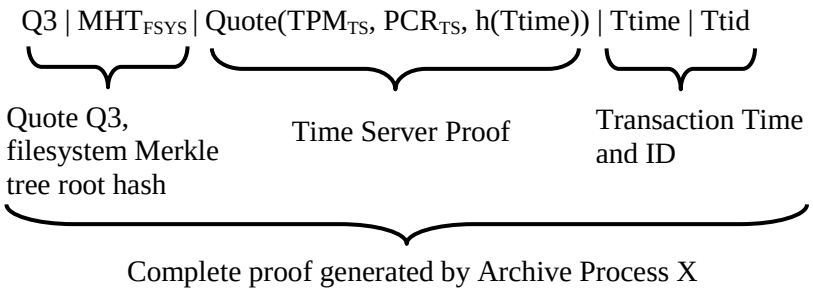


Figure **4-4**: Complete Proof generated by archive process X for transaction ID, Tid.

An archive transaction dump file identified by Tid as the transaction ID

<archive id='Tid' time='Ttime' algorithm='RSA'>
<integrity metadata='list of attributes hashed' type='MerkleTree'>
<roothash> MHT$_{FSYS}$ </roothash>
<dump> dump of MHT created in Figure 4-2 </dump>
<quote> Q3 </quote>
<pubkey> PUBKEY3 = public key used for Q3 </pubkey>
<certificate> certificate of PUBKEY3 </certificate>
<measurements> IMA list of H1 & PCRs of TPM$_{H1}$ </measurements>
</integrity>
<timequote> Qts3 = Quote(TPM$_{TS}$, PCR$_{TS}$, h(Ttime)) </timequote>
<measurementsTS> IMA list of TS & PCRs of TPM$_{TS}$</measurementsTS>
<settings> list of custom user settings during archive </settings>
</archive>

Figure **4-5**: XML Format of the complete proof and related credentials generated by archive process X for transaction Tid. This XML is written to the Tid.dump file in the root of the archived file system. Quote(vTPM$_{H1}$, PCR$_{H1}$, h(tid | all entries in Tid.dump file)) is appended to Tid.dump.

Periodic re-keying of archive manager

The archive manager manages the security information generated by the archiver process. It is assumed that the archive manager runs in the machine H2 and has access to the time-lined secure storage of the archive transaction dump files. The archive transaction dump files contain security information of each archive operation done by archive process in XML format as shown in Figure 4-5. The archiver process created the archive transaction dump files with signatures using the Attestation Identity Key (AIK) of $vTPM_{H1}$. The expiration or revocation of the certificate of a signature public key and any compromise of a key pair necessitates the need for an automatic and transparent update of the signature key pairs by re-keying with a new signature key pair. The archive manager should choose the frequency of re-keying operation according to security policy of the organization maintaining the security information of the archives. Before the re-keying of the archive content proof, the archive manager checks the validity of the quote present in the content proof and the validity of the certificate of the public key of the quote. For example, the manager will verify the quote Q3 and the certificate of PUBKEY3 while re-keying the content proof shown in Figure 4-5.

The re-keying process involves the creation of a new key pair with the public key PUBKEY4 and loading it as the Attestation Identity key (AIK) of TPM_{H2}. It then updates the Tid.dump file with the quote of $h(MHT_{FSYS})$ from the TPM_{H2} with the new AIK, as shown in Figure 4-6. The resulting Tid.dump file is as shown in Figure 4.7. The re-key operation provides an automated mechanism for restricting the amount of data which might be exposed when the private key is compromised. Updating the signing key pairs is a fundamental component of preserving the integrity information of the long term archives. The re-keying operation also provides a transparent way to change algorithms and/or key lengths (for example, it is possible to

change from 1024-bit RSA to 2048-bit RSA). The archive manager can employ the state of the art algorithms during a re-keying and specify the algorithm used as an attribute in the XML created as shown in Figure 4-7 and the Tid.dump file is updated with the modified XML as shown in Figure 4-8. The archive manager creates the new key pair's public key certificate of PUBKEY4 and adds the certificate to the modified Tid.dump XML file. The manager can also use the state of the art format to dump the integrity information rather than as the XML file. The decisions of the periodicity of re-keying, factors that should automatically trigger re-keying and the format of the dump of transaction files should be made according to the technologies, requirements and security challenges that exist at the point of time of managing the archive security information. The forward secure signature scheme [28] can be used to derive the new private key which will help to maintain the same public key with the corresponding certificate but with the different private keys.

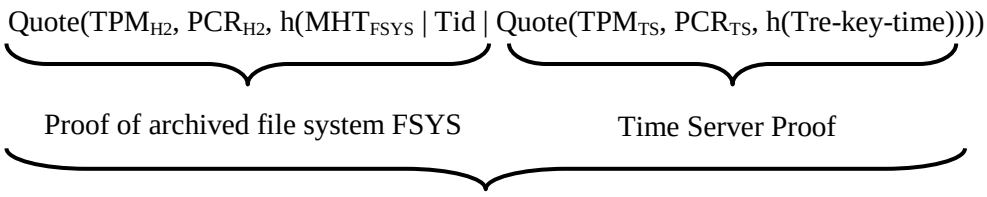


Figure **4-6**: Quote Q4 generated by the archive manager M for transaction ID, Tid.

An updated archive transaction dump file after a key-update

<archive id='Tid' time='Tre-key-time' algorithm='RSA'>
<integrity metadata='list of attributes hashed' type='MerkleTree'>
<roothash> MHT_{FSYS} </roothash>
<dump> dump of MHT created in Figure 4-2 </dump>
<quote> Q4 </quote>
<pubkey> PUBKEY4 = public key used for Q4 </pubkey>
<certificate> certificate of PUBKEY4 </certificate>
<measurements> IMA list of H2 & PCRs of TPM_{H2} </measurements>
</integrity>
<timequote> Qts4 = Quote(TPM_{TS}, PCR_{TS}, h(Tre-key-time)) </timequote>
<measurementsTS> IMA list of TS & PCRs of TPM_{TS}</measurementsTS>
<settings> list of custom user settings during archive </settings>
</archive>

Figure **4-7**: Updated XML Format of the archive dump file after re-keying operation.

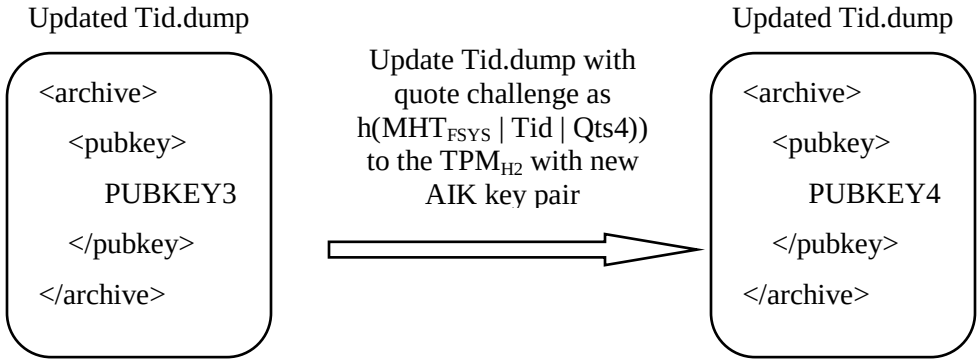


Figure **4-8**: Re-keying process of archive manager.

Challenge protocol of the verifier

Verifier ‘V’ reads the Tid and Ttime contained in the archive_info.xml file in the root node of the file system that was archived using process X. It validates the quotes Q1, Q2 and ensures that Tid and Ttime correspond to the challenge contained in the quotes Q1 and Q2. Verifier ‘V’ challenges the archive manager ‘M’ using OpenSSL with a randomly generated 160 bit nonce ‘N’ (to prevent replay attacks), Tid, Ttime and the name of the file ‘fi’ to be verified.

Response protocol of the archive manager

The archive manager retrieves the Tid.dump file corresponding to the Tid. The archive manager reads the XML file Tid.dump that was created by archive process and updated by archive manager. The integrity of Tid.dump is verified using the signature contained in it. The attribute ‘algorithm’ is referred to identify the algorithm with which the digital signature was generated and the verification proceeds accordingly. The validity of the certificate of the public key is also verified. After successful verification of the signature, the archive manager reads the Merkle hash tree of the file system from Tid.dump and searches for file f1 in the hash tree. It retrieves the hash of the root node as MHT_{FSYS}. It then generates a succinct proof for file fi, denoted as Pf(fi), consisting of the root node and all of the siblings on the path to the root, as shown in Figure 4-10.

The manager computes M4 = h(N | MHT_{FSYS} | Tid | Quote(TPM_{TS}, PCR_{TS}, h(Ttime)) and then requests a quote from the TPM_{H2} with M4 as the challenge. The quote Q8 = Quote(TPM_{H2}, PCR_{H2}, M4) is returned by TPM_{H2} to the archive manager. The quote Q8 is as shown in Figure 4-11. The archive manager sends the following to the verifier ‘V’ as shown in Figure 4-12.

1. Tid, Ttime - Transaction ID and time of corresponding to the archive operation
2. Q8 = Quote(TPM_{H2}, PCR_{H2}, M4)
3. MHT_{FSYS} - The root hash of the file system whose integrity is to be verified
4. Pf(fi) - Succinct proof for file fi belonging to transaction Tid
5. Tid.dump file containing the integrity of transaction Tid generated by Archive process X

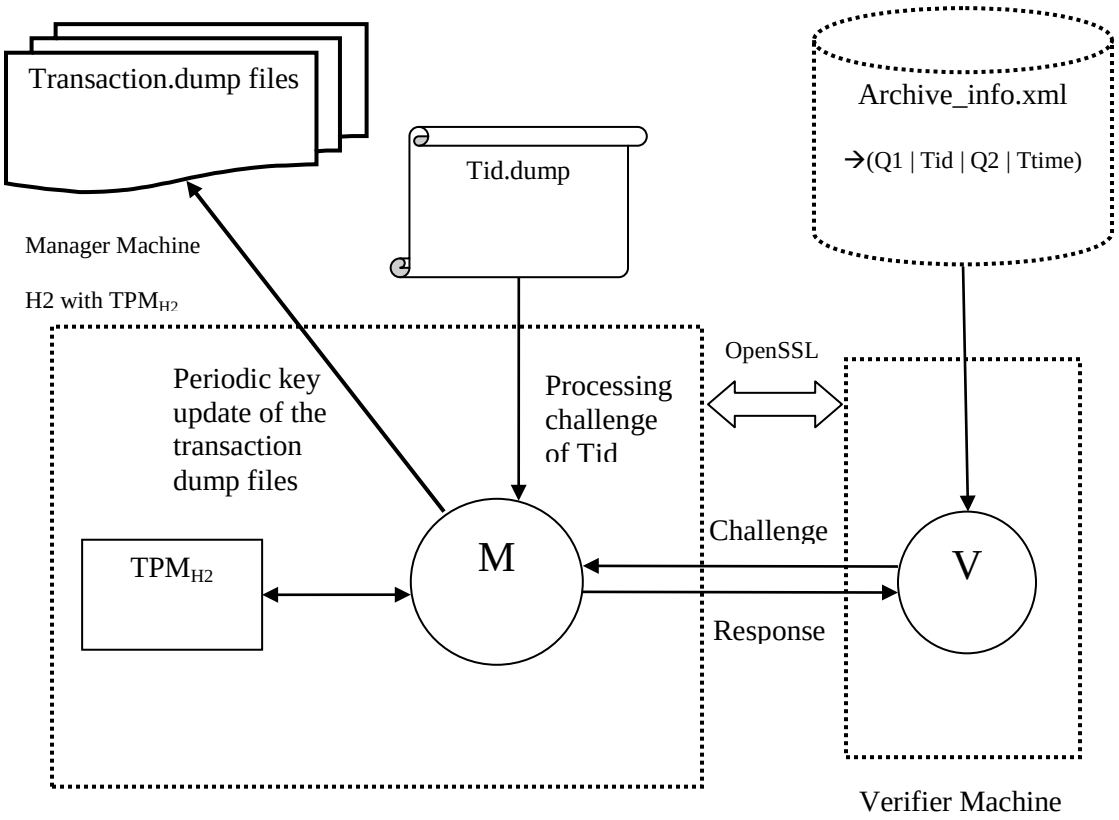
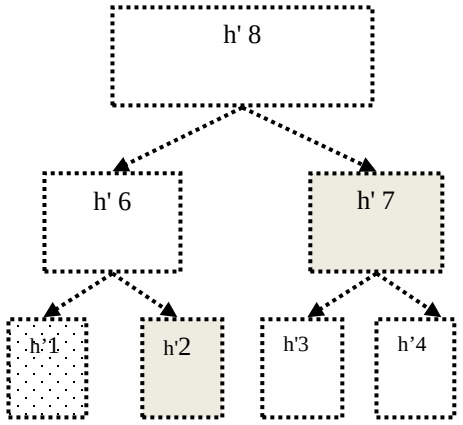


Figure **4-9**: Response protocol of the archive manager process M.



h'1 = SHA1(f1), h'2 = SHA1(f2)
h'3 = SHA1(f3), h'4 = SHA1(f4)
h'6 = SHA1(h'1 | h'2)
h'7 = SHA1(h'3 | h'4)
h'8 = SHA1(h'6 | h'7)
Succinct Proof for file f1, Pf(f1), consists of all of the siblings on the path to the root.
Pf(f1) = h'2, h'7

Figure **4-10**: Succinct Proof generation for file f1.

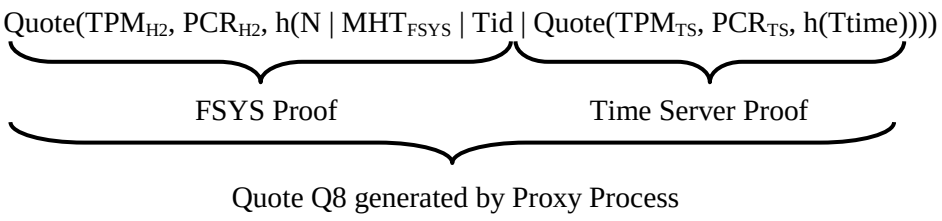


Figure **4-11**: Quote Q8 generated by the archive manager M.

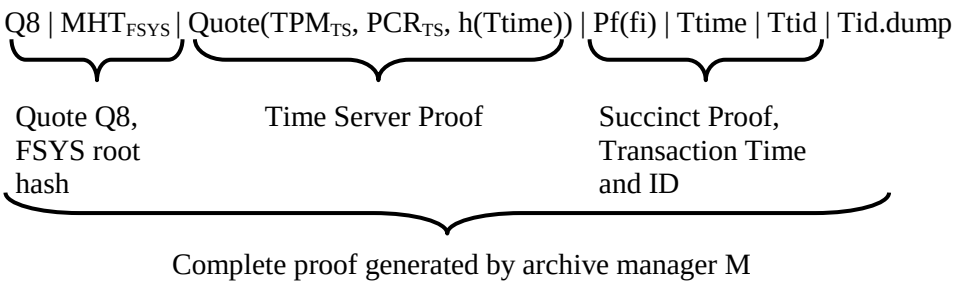


Figure **4-12**: Complete Proof provided by the archive manager M to the verifier V.

Verification protocol of the verifier

The verifier reads the XML file Tid.dump that was created by archive process and updated by archive manager. The integrity of Tid.dump is verified using the signature contained in it. The attribute 'algorithm' is referred to identify the algorithm with which the digital signature was generated and the verification proceeds accordingly. The validity of the certificate of the public key is also verified. After successful verification of the signature, the verifier computes M7 = $h(MHT_{FSYS} \mid Tid \mid Quote(TPM_{TS}, PCR_{TS}, h(Ttime)))$ and M8 = h(N | M7) as shown in Figure 4-13. The verifier validates Q8 and checks that M8 is equal to M4. The verifier thus ensures that the quote Q8 can only be created by TPM_{H2} which verifies the integrity of the environment of the archive manager M in machine H2. The Verifier process V computes the hash of the root node of the file system as M7 by using the succinct proof $Pf(f_i)$ and the hash of the file f_i as shown in figure 4-13. The Verifier process verifies if M7 is equal to MHT_{FSYS}. This step will ensure the integrity of the content of the file fi that was archived at time Ttime with the Transaction ID Tid and the validity of the quote Q3.

$M7 = h(h(f_i) \mid Pf(f_i))$
$M8 = h(MHT_{FSYS} \mid Tid \mid Quote(TPM_{TS}, PCR_{TS}, h(Ttime)))$
$M9 = h(N \mid M8)$

Figure **4-13**: Regenerated hashes at the verifier process V.

The successful verification of the signature contained in Tid.dump file and the successful validation of the certificate of the public signature key will ensure that the integrity of the archive content proof was maintained till the current time by the archive manager.

Chapter 5

Evaluation

The design consists of four main components, namely, the archiver, the manager, the verifier and the trusted time server. The design is implemented in Python and C. The libtpm library in C is used for the TPM interface operations such as the TPM_CreateKey, TPM_LoadKey, TPM_Extend and TPM_Quote operations. The Python-C extensions were written for the TPM operations. The trusted Time Server is implemented in Python as a TCP server. Whenever a client requests the time server for a quote, the time server reads the clock in the machine TS and computes the hash of the current time t1 as a challenge to the TPM_{TS} to generate a time quote as Quote(TPM_{TS}, PCR_{TS}, h(t1)). The archive process X uses the python library function 'shutil.copytree' to archive the file system along with the file attributes from one device to the other. It then generates the content proof with the asynchronous time attestation as described in the previous section. Debian Linux 2.6.18 patched with IMA and Xen is used for the Dom-0 and Debian Linux 2.6.24 patched with IMA and compiled with vTPM support is used for the Dom-U where the archive process X runs.

The manager process M periodically makes a re-key operation to the generated content proofs or the transaction dump files as shown in Figure 4-7. Before any re-key of a particular content proof, the manager checks the validity of the attestation already contained in the content proof. It also has a thread running as a TCP server which listens to any incoming challenge requests from the verifier. In response to a challenge, the manager TCP server thread generates the succinct proof as shown in Figure 4-9 and sends the entire proof as shown in Figure 4-12 to

the verifier process. The manager process M and the verifier process V are run in a host hosting Debian Linux 2.6.24 patched with IMA.

Table **5-1**: Time taken for basic TPM commands.

TPM Operation	Time in Seconds (s)
TPM_Quote	1.48
TPM_Verify	0.06
TPM_LoadKey	1.10
TPM_EvictKey (a single key)	0.18
TPM_ListKeys	0.06

The time taken for the basic TPM operations is as shown in Table 5-1. The archiver is evaluated with file systems of size ranging from 3GB to 16GB. The content proof size and the total time to generate the archive content proof is shown in Table 5-2. The content proof size depends on the number of files present in the archive as shown in Figure 5-1.

Table **5-2**: Comparison of content proof size and total processing time to archive size.

Archive ID	Archive Size (GB)	Content Proof Size (MB)	Total processing time to generate content proof (s)	Number of files in the archive
8	3.09	0.56	75.84	2324
9	4.25	0.90	112.55	4618
10	5.57	1.04	120.69	5240
11	5.81	0.86	142.85	4149
12	7.35	1.30	184.90	6924
13	8.66	1.39	212.27	7564
14	9.82	1.79	234.97	9858
15	11.13	1.90	264.89	10480
16	11.78	1.99	279.16	11094
17	12.91	2.17	300.30	12182
18	14.22	2.27	330.50	12804
19	14.87	2.37	344.27	13418
20	16.19	2.46	440.94	14040

The processing time break-up for the content proof generation of the archiver is shown in Table 5-3. The total processing time depends on the archive size as shown in Figure 5-2. The time to generate the Merkle file hash tree dominates the total processing time to generate the content proof for the archives.

Table **5-3**: Processing time break-up for content proof generation.

Archive ID	Archive Size (GB)	Time to generate file hash tree (s)	Time to generate TPM quotes (s)	Number of files in the archive
8	3.09	72.16	3.66	2324
9	4.25	108.70	3.85	4618
10	5.57	116.82	3.85	5240
11	5.81	139.09	3.76	4149
12	7.35	180.95	3.94	6924
13	8.66	208.27	3.98	7564
14	9.82	230.74	4.18	9858
15	11.13	260.68	4.17	10480
16	11.78	274.93	4.22	11094
17	12.91	295.97	4.31	12182
18	14.22	326.03	4.44	12804
19	14.87	339.74	4.48	13418
20	16.19	436.48	4.45	14040

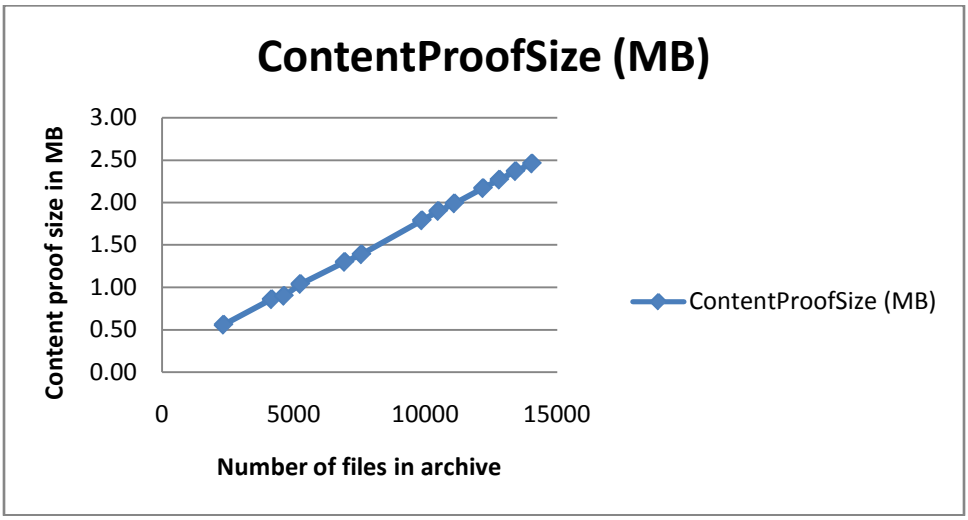


Figure **5-1**: Comparison of content proof size to number of files in archive.

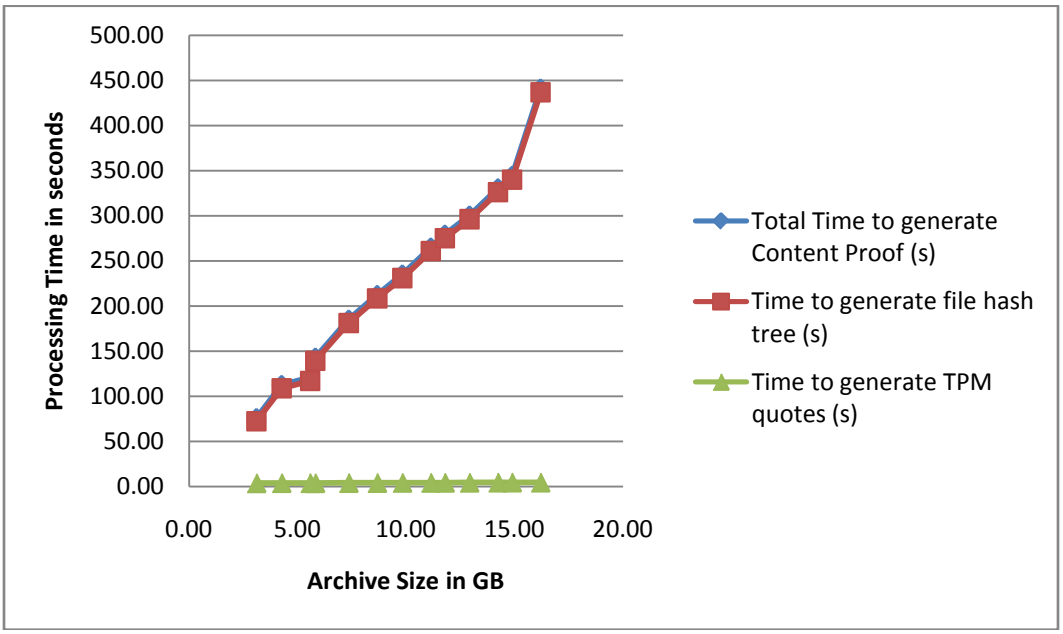


Figure **5-2**: Processing time break-up for content proof generation.

The Table 5-4 and Figure 5-3 compare the size of the content proofs and re-keying processing time to the archive size. The size of content proof after every re-keying remains the same, thus ensuring that the size of the content proofs is independent of the number of re-key operations. The total processing time increases only linearly when compared to the size of the content proofs as shown in Figure 5-3, mainly due to the size of the XML content proof file that has to be processed. Thus the re-key processing time is not affected much by the content proof size, which in effect is dependent on the number of files in the archive.

Table **5-4**: Comparison of content proof size and total processing time for re-keying.

Archive ID	Size of Content Proof (MB)	Total Time for re-keying operation (s)
8	0.56	3.51
9	0.90	3.64
10	1.04	3.69
11	0.86	3.65
12	1.30	3.82
13	1.39	3.85
14	1.79	3.99
15	1.90	4.05
16	1.99	4.09
17	2.17	4.35
18	2.27	4.20
19	2.37	4.24
20	2.46	4.29

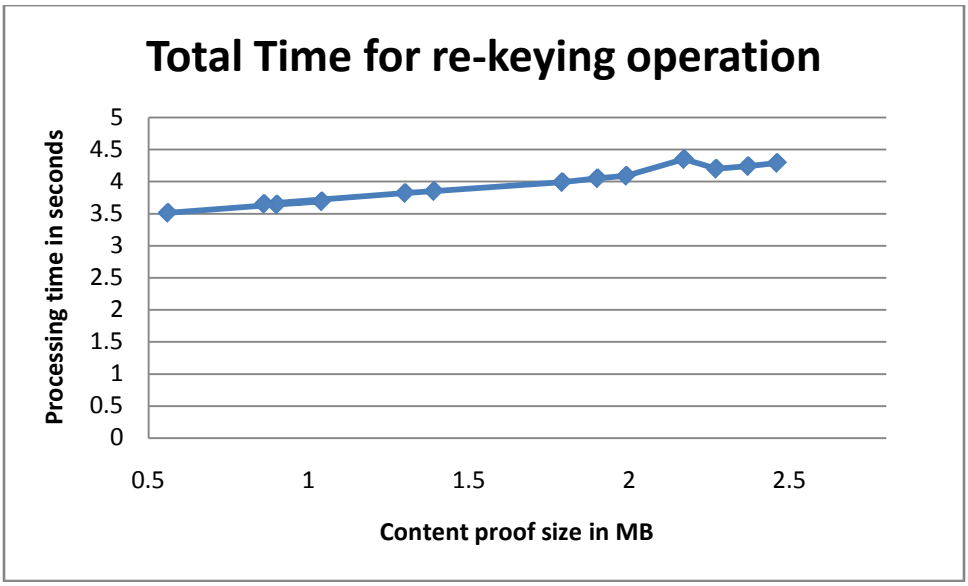


Figure **5-3**: Total processing time for the re-keying operation.

The TPM quote operations for the generation of the new attestation dominates the XML updating time for the re-keying operation. The succinct proof generation for a particular file depends on the time to re-generate the file hash tree, which in turn is dependent on the number of files present in the archive, as shown in Table 5-5.

Table **5-5**: Processing time for succinct proof generation.

Archive ID	Number of files in the archive content proof	Total time to verify archive content proof (s)	Breakup of time to recreate file tree hashes (s)	Total time to generate succinct proof (s)
8	2324	0.14	0.07	0.14
9	4618	0.22	0.15	0.22
10	5240	0.22	0.16	0.23
11	4149	0.21	0.14	0.21
12	6924	0.26	0.20	0.27
13	7564	0.27	0.21	0.28
14	9858	0.38	0.31	0.40
15	10480	0.39	0.33	0.40
16	11094	0.41	0.35	0.42
17	12182	0.43	0.36	0.45
18	12804	0.44	0.37	0.47
19	13418	0.45	0.38	0.49
20	14040	0.47	0.40	0.52

The protocol ensures freshness of the challenge-response protocol through the nonce ‘N’ used by the verifier. This prevents the replay attacks. The use of the Time Server TS will enable the verifier to verify the time at which the archive operation was done by the archive process ‘X’. The hash tree of the entire file system that was archived is signed and the transaction ID of each archive operation along with its time quote and content proof quote is also saved in the content proof file by process X for future use. The use of the Merkle hash tree effectively uses the single time-stamped attestation for the entire archive. The archive manager ‘M’ is also tied to the state of TPM_{H2}, so that the integrity of the manager machine H2 can also be verified by the verifier V.

The re-keying operation of the archive manager process addresses the weakening of cryptography over time and the need to re-validate signed archives and the certificates of the signing keys.

Chapter 6

Conclusion

A protocol is proposed to prove far in the future the integrity of an archive by the use of digital signatures, verifiable timestamps, Trusted Platform Module and Xen Virtual Machine Monitor. The asynchronous attestation protocol relies on a verifiable timestamp that is used along with the attestation of the root file system to provide a proof of the integrity of the file system at the time of archive without the possibility of post-dating or pre-dating. The issues of the weakening of cryptography over time and the need to re-validate signed archive content proofs and the certificates of the signing keys are addressed by the re-keying protocol. The re-keying of the content proofs is tied to the time of the content proof update and hence a third party verifier will be able to attest that the signing key was not compromised at the time of signing the archive or at the time of a re-keying operation. The re-keying process also follows the asynchronous attestation protocol. The XML format used by the re-keying process enables the option of using state of the art algorithm for the digital signature existing at that point in time. The XML format used in this protocol can also be changed to a compatible format at a later point in time in the future by the archive manager.

However the migration of archives from one media or logical format to the other and software and hardware unreliability has to be taken into more detailed analysis to build a comprehensive archival system that can ensure data reliability over a very long period of time of more than 50 or 100 years. The proposed protocol can be integrated with other proposed solutions to account for migration problems, data availability and data readability.

Bibliography

[1] 100 Year Archive Requirements Survey, January 2007
http://www.snia.org/forums/dmf/programs/ltacsi/forums/dmf/programs/ltacsi/100_year/100YrATF_Archive-Requirements-Survey_20070619.pdf

[2] J. McKnight, T. Asaro, and B. Babineau, ‘‘Research Report: Digital Archiving: End-User Survey & Market Forecast 2006–2010,’’ Enterprise Strategy Group, Milford, MA, January 2006.

[3] R. A. Lorie, ‘‘Long Term Preservation of Digital Information,’’ Proceedings of the First ACM/IEEE-CS Joint Conference on Digital Libraries, Roanoke, VA, 2001, pp. 346–352.

[4] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In Proceedings of USENIX Security Symposium, pages 223–238, 2004.

[5] T. Moyer, K. Butler, J. Schiffman, P. McDaniel, and T. Jaeger. Scalable Asynchronous Web Content Attestations.

[6] L. St.Clair, J. Schiffman, T. Jaeger, and P. McDaniel. Establishing and Sustaining System Integrity via Root of Trust Installation. In 23rd Annual Computer Security Applications Conference (ACSAC), pages 19–29, Miami, FL, December 2007.

[7] T. Jaeger, R. Sailer, and U. Shankar. PRIMA: Policy-Reduced Integrity Measurement Architecture. In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT 2006), Lake Tahoe, CA, June 2006.

[8] J. M. McCune, T. Jaeger, S. Berger, R. Caceres, and R. Sailer. Shamon: A system for distributed mandatory access control. In ACSAC ’06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference, pages 23–32, Washington, DC, USA, 2006. IEEE Computer Society.

[9] R. Merkle. Protocols for public key cryptosystems. In Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, Apr. 1980.

[10] D. Eastlake 3rd, J. Reagle, and D. Solo. (Extensible Markup Language) XML-Signature Syntax and Processing. RFC 3275 (Draft Standard), Mar. 2002.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization," in SOSP '03: Proceedings of the nine-teenth ACM symposium on Operating systems principles.

[12] Berger, S., Caceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: Virtualizing the Trusted Platform Module. In: Proceedings of the 15th USENIX Security Symposium, USENIX, August 2006, pp. 305–320 (2006)

[13] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In Proceedings of ACM Symposium on Operating Systems Principles (SOSP), 2003.

[14] R. Kotla, M. Dahlin, and L. Alvisi. SafeStore: A durable and practical storage system. In Proceedings of the 2007 USENIX Annual Technical Conference, pages 129–142. USENIX, June 2007.

[15] P. Maniatis and M. Baker. Secure History Preservation Through Timeline Entanglement. In Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 2002.

[16] P. Maniatis, M. Roussopoulos, T. J. Giuli, D. S. H. Rosenthal, and M. Baker. The LOCKSS peer-to-peer digital preservation system. ACM Trans. Comput. Syst., 23(1):2–50, 2005.

[17] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. POTSHARDS: Secure long-term storage without encryption. In Proceedings of the 2007 USENIX Annual Technical Conference, pages 143″U–156. USENIX, June 2007.

[18] Sproull, R. F., and Eisenberg, J. Building an Electronic Records Archive at the National Archives and Records Administration: Recommendations for a Long-Term Strategy. <http://www.nap.edu/catalog/11332.html>, June 2005.

[19] A. Jerman Blazic. Long term trusted archive services. In First International Conference on the Digital Society ICDS, page 29. IEEE Computer Society, Jan 2007.

[20] Maniatis, P., Giuli, T., and Baker, M. Enabling the Long-Term Archival of Signed Documents through Time Stamping. Technical report, Computer Science Department, Stanford University, Stanford, CA, USA, June 2001.

[21] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21(2):120–126, 1978.

[22] A. Shamir. How to share a secret. Commun. ACM, 22(11):612–613, 1979.

[23] SHA-1 Standard. http://www.itl.nist.gov/fipspubs/fip180-1.htm

[24] Trusted Computing Group. TPMWorking Group.
https: //www.trustedcomputinggroup.org/groups/tpm/.

[25] Trusted Computing Group. Trusted Platform Module Specifications.
https://www.trustedcomputinggroup.org/specs/TPM/.

[25] C. Wallace, R. Brandner, and U. Pordesch. Long-term archive service requirements. RFC 4810, Internet Engineering Task Force, March 2007.

[26] A. Jerman Blazic. Long term trusted archive services. In First International Conference on the Digital Society ICDS, page 29. IEEE Computer Society, Jan 2007.

[27] G. R. Ganger, P. K. Khosla, M. Bakkaloglu, M. W. Bigrigg, G. R. Goodson, S. Oguz, V. Pandurangan, C. A. N. Soules, J. D. Strunk, and J. J. Wylie. Survivable storage systems.

[28] M. Bellare and S. Miner, “A forward-secure digital signature scheme”

[29] D. Boneh, C. Gentry, B. Lynn and H. Shacham, "A survey of two signature aggregation techniques"

[30] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. “Sequential aggregate signatures and multi-signatures without random oracles”

[31] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. “Aggregate and verifiably encrypted signatures”

[32] P. Ting and F. Chu, "Enhancing the Security Promise of a Digital Time-Stamp"

[33] D. Song, "Practical Forward Secure Group Signature Schemes"

Appendix A

Glossary

Asymmetric key encryption - In this encryption method, key exchange occurs with the existence and use of both a public and private key, in which anyone can encrypt a message using the public key, but only the holder of the private key is able to decrypt messages. Page 18.

AIK Attestation – an asymmetric key, the private portion of which is non-migratable and protected by the TPM. The public portion of an AIK is part of the AIK Credential, issued using either the Privacy CA. An AIK can only be created by the TPM Owner. The AIK can be used for platform authentication, platform attestation and certification of keys. Page 11.

AIK Credential - A credential issued by a Privacy CA that contains the public portion of an AIK key signed by a Privacy CA. Page 11.

Attestation of the Platform - An operation that provides proof of a set of the platform's integrity measurements. This is done by digitally signing a set of PCRs using an AIK in the TPM. Page 11.

Authenticated Boot - A boot after which the platform's Root-of-Trust-for-Reporting (RTR) can report an accurate record of the way that the platform booted. Page 10.

Challenger - An entity that requests and has the ability to interpret integrity metrics. Page 11.

Endorsement Key - EK; an RSA Key pair composed of a public key (EKpu) and private (EKpr). The EK is used to recognize a genuine TPM. The EK is used to decrypt information sent to a TPM in the Privacy CA and during the installation of an Owner in the TPM. Page 8.

Integrity challenge - A process used to send accurate integrity measurements and PCR values to a challenger. Page 11.

Privacy CA - An entity trusted by both the Owner and the Verifier, that will issue AIK Credentials. Page 11.

Private key - Key owned by recipient of encrypted message in private key cryptography. Private key is used to decrypt the message. Page 17.

Private key encryption - See asymmetric key encryption. Page 18.

Public key - Key owned by sender of encrypted message in private key cryptography (used to encrypt messages, (or) Key owned by both sender and recipient of a message – the single public key is used to both encrypt and decrypt the message. Page 17.

Public key encryption - See symmetric key encryption. Page 18.

RSA - Algorithm developed by: Rivest, Shamir, Adleman. Their creation was named after the first letters of their surnames. Page 19.

Symmetric key encryption - The sender and the recipient each have a copy of the public key (which is to be kept secret between the two of them). The public key both encrypts and decrypts the message. Page 18.

SRK - Storage Root Key: the root key of a hierarchy of keys associated with a TPM's Protected Storage function; a non-migratable key generated within a TPM. Page 8.

TSS - TCG Software Stack: untrusted software services that facilitate the use of the TPM and do not require the protections afforded to the TPM. Page 8.

Trust - Trust is the expectation that a device will behave in a particular manner for a specific purpose. Page 10.

Trusted Computing Platform - A Trusted Computing Platform is a computing platform that can be trusted to report its properties. Page 8.