Security of Cloud Computing

Fabrizio Baiardi
f.baiardi@unipi.it

Syllabus

- Cloud Computing Introduction
  - Definitions
  - Economic Reasons
  - Service Model
  - Deployment Model
- Supporting Technologies
  - Virtualization Technology
  - Scalable Computing = Elasticity
- Security
  - New Threat Model
  - New Attack
  - Countermeasures ← Attestation

Securing Clouds

Goal: Some policies to detect if an attacker (or the provider) has updated the software layers of the cloud system (and it violated any SLA)

Papers

Santos et al., Towards Trusted Cloud Computing, HotCloud 2009

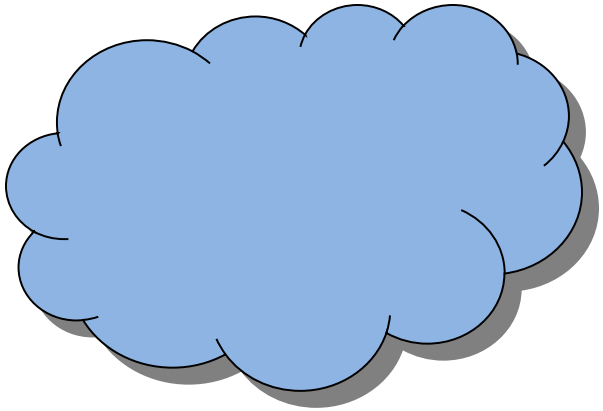
Krautheim, Private Virtual Infrastructure for Cloud Computing, HotCloud 2009

Wood et al., The Case for Enterprise-Ready Virtual Private Clouds, HotCloud 2009

Baiardi et al., Measuring Semantic Integrity for Remote Trust 2009
Baiardi et al. Attesting the Integrity of Overlay Networks,. Journal of Sys.Arch 2011.

The IaaS security problem

The cloud acts as a big black box, nothing inside the cloud is visible to the clients



Even if a SLA exists, clients have no control over the cloud software but the overall behaviour depens upon this software

Even an honest cloud provider can have malicious sys admins who can tamper with the VMs and violate confidentiality and integrity

How to discover if the cloud has been tampered with?

Naïve Approach 1: Just trust the cloud provider

Why won't work: Provider may be honest, sys admins may not be so

Naïve Approach 2: Ask the cloud provider to allow auditing of the cloud by the client

Why won't work: Providers are not willing to open their system to outside audits

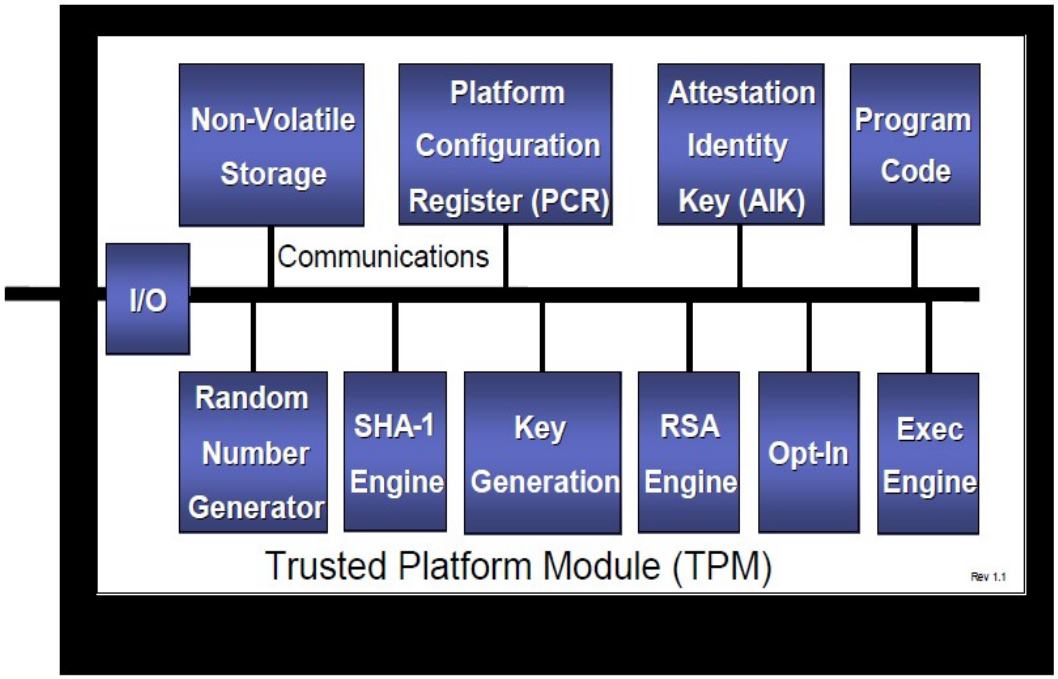
Workable Approach 3: Ask cloud provider for unforgeable proof/ attestation

Why may work: A third party proof not revealing other information may be enough for both client and provider SLA is fundamental

How to discover if the cloud has been tampered with?

- Any solution to attest the integrity of any object
  - measures the object
  - compares the measurements against the stored one
- This immediately poses the problem of trusting the measurements and the component that implements them
- The problem may not be solved by recursion because anyway we have the problem of the root of trust of the measurement system
- In computer science, one solution for the root of trust is
  - a trustable
  - anti tamper
  - hardware component

Trusted Platform Module (TPM)



TPMs are inexpensive chips now included in most laptops

Can be the building block for a trusted computing base

Can bootstrap trust in a system

Cannot (easily) be compromised to get the keys

The keys never leave the module

Endorsement Key: A private (RSA) key that identifies the chip

Platform Configuration Register (PCR) : Can contain hashes of system configurations

Background

TCG consortium. Founded in 1999 as TCPA.

Main players (promotors): (>200 members)

AMD, HP, IBM, Infineon, Intel, Lenovo, Microsoft, Sun

Goals:

Hardware protected (encrypted) storage:

Only “authorized” software can decrypt data

e.g.: protecting key for decrypting file system

Secure boot: method to “authorize” software

Attestation: Prove to remote server what software is running on my machine.

TPMs in the real world

Systems containing TPM chips:

- Lenovo (IBM) Thinkpads and desktops
- Fujitsu lifebook
- HP desktop and notebooks
- Acer, Toshiba, Panasonic, Gateway, Dell, …

Software using TPMs:

- File/disk encryption: Vista, IBM, HP, Softex
- Attestation for enterprise login: Cognizance, Wave
- Client-side single sign on: IBM, Utimaco, Wave

TPM module

Input/Output (I/O)

Manages information flow over the communications bus. It performs protocol encoding /decoding for communication over external and internal buses. It routes messages to appropriate components. It enforces access policies associated with the Opt-In component as well as other TPM functions requiring access control.

Non-Volatile Storage

It stores Endorsement Key (EK), Storage Root Key (SRK), owner authorization data and persistent flags.

At least 16 Platform Configuration Registers (PCR) have to be implemented in either volatile or non-volatile storage.

Registers 0-7 for TPM 8-15 available for operating system and application use.

They are reset at system start or whenever the platform loses power.

In the last version counters only operation increase/read

Attestation Identity Keys(AIK)

Must be persistent, but it is recommended that AIK keys be stored in persistent external storage (outside the TPM), rather than stored permanently inside TPM non-volatile storage to speed up the computation.

TPM module

Program Code

It contains firmware for measuring platform devices. Logically, this is the Core Root of Trust for Measurement (CRTM). Ideally, the CRTM is contained in the TPM, but implementation decisions may require it be located in other firmware.

Random Number Generator (RNG)

A true random-bit generator used to seed random number generation. The RNG is used for key generation, nonce creation and to strengthen pass phrase entropy.

Sha-1 Engine

This message digest engine is used for computing signatures, creating key Blobs and for general purpose use.

RSA Key Generation

TCG standardizes the RSA algorithm for use in TPM modules. Its recent release into the public domain combined with its long track record makes it a good candidate for TCG. The RSA key generation engine is use to create *signing keys* and *storage keys*. TCG requires a TPM to support RSA keys up to a 2048-bit modulus, and mandates that certain keys (the SRK and AIKs, for example) must have at least a 2048-bit modulus.

TPM module

RSA Engine

It is used for signing with *signing keys*, encryption/decryption with *storage keys*, and decryption with the EK. It is anticipated that TPM modules containing an RSA engine will not be subject to import/export restrictions.

Opt-In

This component implements TCG policy requiring TPM modules are shipped in the state the customer desires. This ranges from **disabled** and **deactivated** to fully **enabled;** ready for an owner to take possession. The Opt-In mechanism maintains logic and (if necessary) interfaces to determine physical presence state and ensure disabling operations are applied to other TPM components as needed.

Execution Engine

It runs program code, performs TPM initialization and measurement taking

Tamper-Protected Packaging

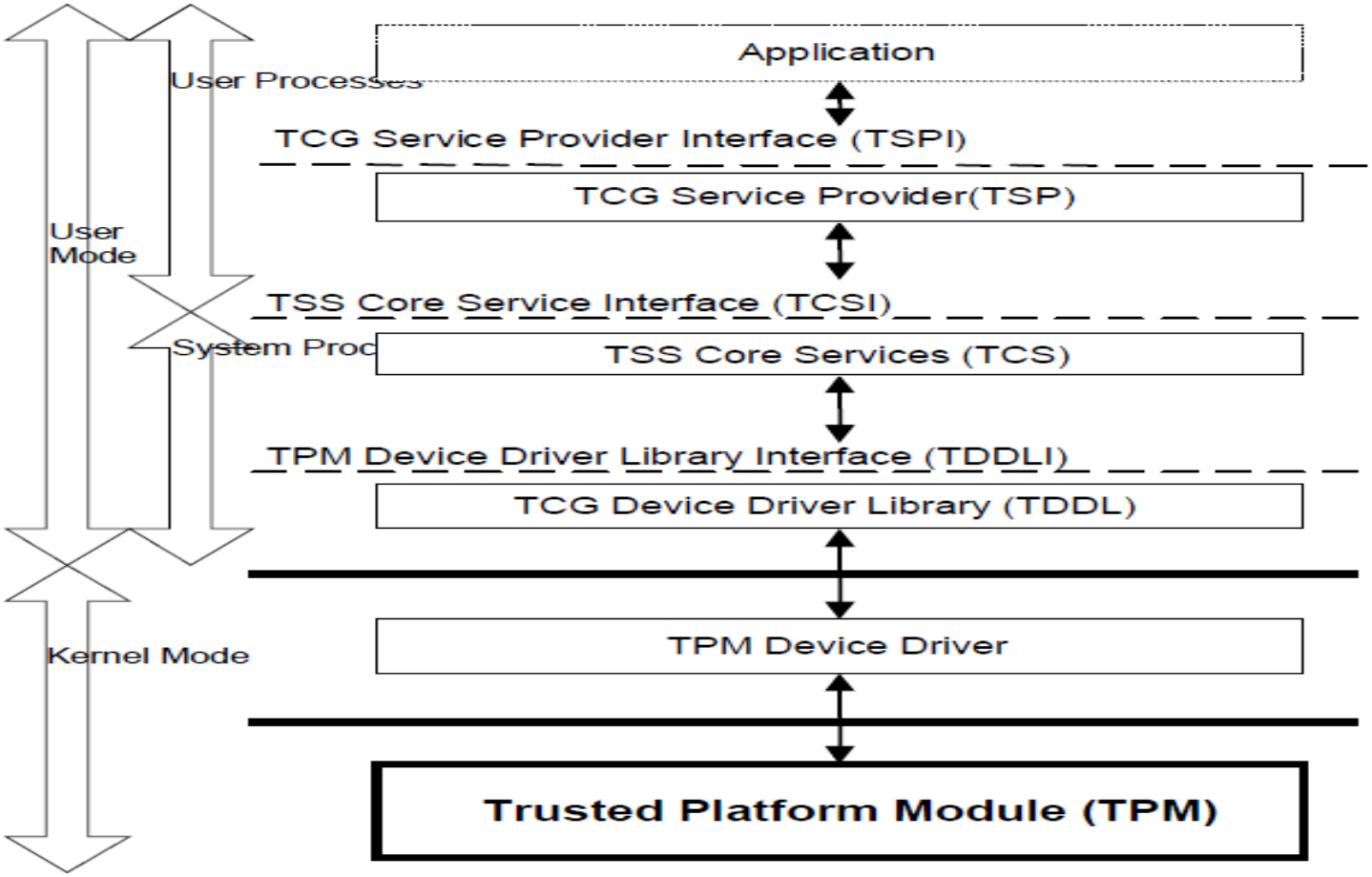
TCG requires the TPM be physically protected from tampering. This includes physically binding the TPM module (if it were physically a discrete part) to the other physical parts of the platform (e.g. motherboard)such that it cannot be easily disassembled and transferred to other platforms.

These mechanisms are intended to *resist* tampering but tamper evidence measures are to be employed to enable *detection* of tampering upon physical inspection.

TPM packaging must limit pin probing and EMR scanning. The TPM must be “glued” to the motherboard as a deterrent to removal procedures and removal of a TMP device is evident to visual inspection.

Software TPM implementations must justify a hardware-equivalent interpretation for tamper-protection. Such an interpretation should realize the desired security properties. Namely, that a particular TPM has cardinality of exactly one and that that TPM is bound to a particular platform.

TCG Software Layers



TPM commands

Component	*Area*	*Command Name*	*Description*
TPM / RTS	*Protected Storage Commands*	TPM_Seal TPM_Unseal TPM_UnBind TPM_CreateWrapKey	These commands use public-key cryptography to prepare arbitrary data and keys for private key operations at TPM endpoints, and to perform those private key operations. TPM endpoints may be explicitly refined via platform configuration register values.
	Key Management Commands	TPM_LoadKey TPM_EvictKey TPM_GetPubKey TPM_CertifyKey TPM_SaveKeyContext TPM_LoadKeyContext	These commands control which keys are available for use by the TPM and prepare keys for safe storage outside the TPM package.
	Migration Commands	TPM_CreateMigrationBlob TPM_ConvertMigrationBlob TPM_AuthorizeMigrationKey	These commands are used to transfer migratable objects from one TPM to another.
	Maintenance Commands (optional)	TPM_CreateMaintenanceArchive TPM_LoadMaintenanceArchive TPM_KillMaintenanceFeature TPM_LoadManuMaintPub TPM_ReadManuMaintPub	These commands are used to transfer non-migratable objects from one TPM to another. Transfer requires cooperation of both the TPM owner and an external entity, probably the platform OEM or their agent.
	Startup Commands	TPM_Reset TPM_Init TPM_SaveState TPM_Startup	These commands are used to reset and restart the TPM.

TPM commands

Component	Area	Command Name	Description
TPM / RTM	*Measurement Collection Commands*	TPM_Extend TPM_DirWriteAuth TPM_SHA1Start TPM_SHA1Update TPM_SHA1Complete TPM_SHA1CompleteExtend	These commands facilitate update of Platform Credential Register (PCR) and Data Integrity Register (DIR) values and for computing hash and extend values
TPM / RTR	*Measurement Reporting Commands*	TPM_PcrRead TPM_Quote TPM_DirRead TPM_DirReadSigned	These commands facilitate reporting of Platform Credential Register (PCR) and Data Integrity Register (DIR) values. These commands may use AIK keys.
	TPM Endorsement Key Commands	TPM_CreateEndorsementKeyPair TPM_ReadPubek TPM_DisablePubekRead TPM_OwnerReadPubek	These commands manipulate the platform Endorsement Key (EK) and manage access control policy.
	AIK Commands	TPM_MakeIdentity TPM_ActivateIdentity	These commands manage the creation, activation and recovery of Attestation Identity Keys (AIK).
TPM Support Services	*Authentication Protocols and Authorization Commands*	TPM_OIAP TPM_OSAP TPM_ChangeAuth TPM_ChangeAuthOwner TPM_ChangeAuthAsymStart TPM_ChangAuthAsymFinish TPM_SaveAuthContext TPM_LoadAuthContext	These commands establish authorized sessions for exchanging commands with the TPM. They also manage access controlled objects contained within the TPM.

TPM commands

TPM Misc. Services	*Cryptographic Commands*	TPM_Sign TPM_GetRandom TPM_StirRandom	These commands provide general purpose cryptographic services.
	Auditing Commands	TPM_GetAuditEvent TPM_GetAuditEventSigned TPM_SetOrdinalAuditStatus TPM_GetOrdinalAuditStatus	These commands are used to collect audit trail data and control auditing features.
	Capability Reporting Commands	TPM_GetCapability TPM_GetCapabilitySigned TPM_GetCapabilityOwner	These commands provide information about the TPM part and implemented functionality.
TPM Management	*TPM Ownership Commands*	TPM_TakeOwnership TPM_SetOwnerInstall TPM_OwnerSetDisable TPM_FieldUpgrade TPM_SetRedirection	These commands are used to initialize the TPM for deployment and for field maintenance.
	Operational Flags Commands	TPM_OwnerClear TPM_DisableOwnerClear TPM_ForceClear TPM_DisableForceClear TPM_PhysicalDisable TPM_PhysicalEnable TPM_PhysicalSetDeactivated TPM_SetTempDeactivated TSC_PhysicalPresence	These commands configure the operational modes of the TPM.
	Self-Test Commands	TPM_SelfTestFull TPM_CertifySelfTest TPM_ContinueSelfTest TPM_GetTestResult	These commands are used to detect and diagnose problems with TPM operation.

TPM Important Commands

Command	*Main inputs*	*Main outputs*	*Authorisation*
TPM_CreateWrapKey Creates a new TPM key. The new key is returned to the user, with the private part, the authdata, and key attributes encrypted with another TPM key, called the parent key	parent key handle; ADIP-encrypted new authdata; information about the key to be created	wrapped key (i.e. newly created key, encrypted with parent key)	parent key
TPM_LoadKey2 Given a wrapped key, loads it on to the TPM for usage	wrapped key	key handle	parent key
TPM_Seal Given some data, encrypts it with a TPM key. Some PCR values that should hold on unseal may also be specified	key handle; encrypted new authdata for the sealed blob; PCRs for unseal; data to be sealed	sealed blob	key
TPM_Unseal Given sealed data, decrypts it. Checks that the PCR values specified in the blob are indeed current	key handle; sealed blob	unsealed data	key; sealed blob

TPM Important Commands

Command	*Main inputs*	*Main outputs*	*Authorisation*
TPM_Extend Updates a PCR by "hashing in" a measurement value	PCR; measurement	(none)	(none)
TPM_Quote Obtains a signed report of the current PCR values	key handle; PCRs; external data	a signature on PCR values and the external data	(none)
TPM_MakeIdentity Create an *application identity key* (AIK)	new encrypted auth info about the identity and the privacy CA	new key blob	owner srk
TPM_ActivateIdentity Decrypt an AIK certificate obtained from a Privacy CA	AIK handle blob from Privacy CA	session key to decrypt the certificate	owner AIK auth

Non-volatile storage

1. Endorsement Key (EK) (2048-bit RSA)
   Created at manufacturing time. Cannot be changed.
   Used for "attestation" (described later)

2. Storage Root Key (SRK) (2048-bit RSA)
   Used to implement encrypted storage
   Created by TPM_TakeOwnership(OwnerPassword, …)
   Cleared by TPM_ForceClear from BIOS

3. OwnerPassword (160 bits) and persistent flags

Private EK, SRK, and OwnerPwd never leave the TPM

Endorsment Key

- The endorsement key is an encryption key that is permanently embedded in the Trusted Platform Module (TPM) security hardware, generally at the time of manufacture.
- This private portion of the endorsement key is never released outside of the TPM. The public portion of the endorsement key helps to recognize a genuine TPM.
- TPM operations that involve signing pieces of data can make use of the endorsement key to allow other components to verify that the data can be trusted.
- To sign a piece of data, a private key is used to encrypt a small piece of information. The signature can be verified by using the corresponding public key to decrypt that same piece of data. If it can be decrypted with the public key, then it must have been encrypted by the corresponding private key.

Storage Key

- The storage root key is embedded in the Trusted Platform Module (TPM) security hardware.
- It is used to protect TPM keys created by applications, so that these keys cannot be used without the TPM.
- Unlike the endorsement key (which is generally created when the TPM is manufactured), the storage root key is created when you take ownership of the TPM.
- This means that if you clear the TPM and a new user takes ownership, a new storage root key is created.

PCR: the heart of the matter

PCR: Platform Configuration Registers

Lots of PCR registers on chip (at least 16)

Register contents: 20-byte SHA-1 digest (+junk)

Updating PCR #n :

TPM_Extend(n,D): PCR[n] ← SHA-1 (PCR[n] || D)

TPM_PcrRead(n): returns value(PCR(n))

PCRs initialized to default value (e.g. 0) at boot time

TPM can be told to restore PCR values via

TPM_SaveState and TPM_Startup(ST_STATE)

Using PCRs: the TCG boot process

BIOS **boot block** executes

1. TPM_Startup (ST_CLEAR) to initialize PCRs to 0
2. PCR_Extend(n, <BIOS code>)
3. Load and runs BIOS post boot code

BIOS executes:

1. PCR_Extend(n, <MBR code>)
2. Runs MBR (master boot record), e.g. GRUB.

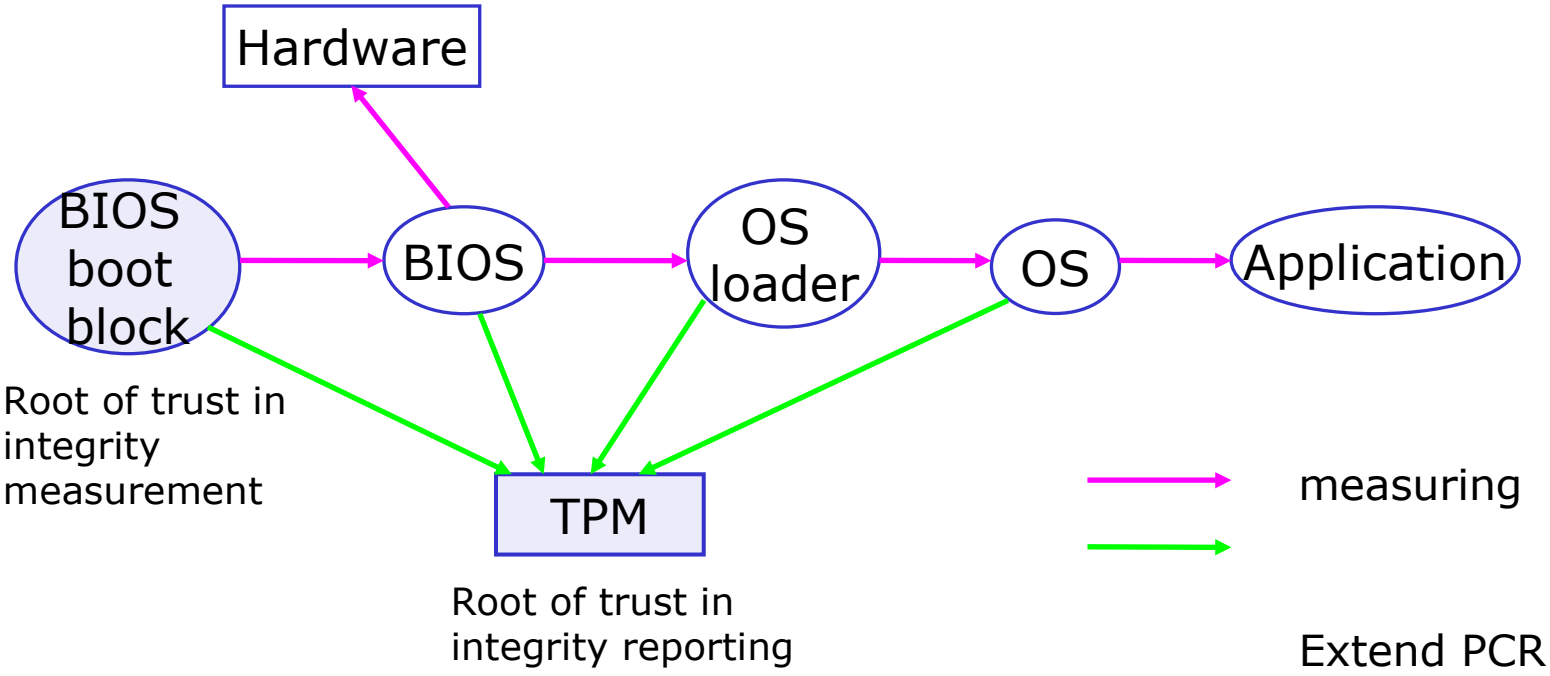
MBR executes:

1. PCR_Extend(n, <OS loader code, config>)
2. Runs OS loader

… and so on

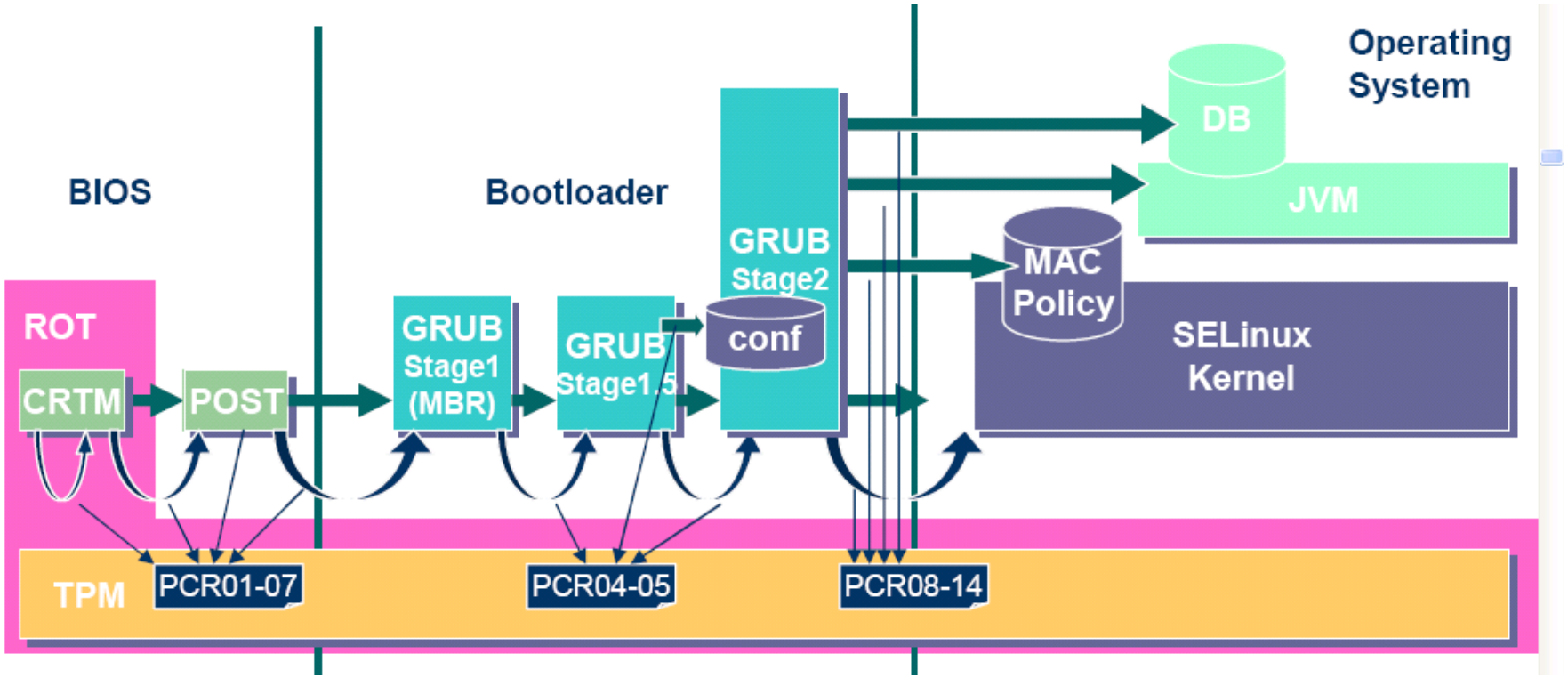
Each step computes the hash of the code it loads

In a diagram



- After boot, PCRs contain hash chain of booted software
- Collision resistance of SHA1 (?) ensures commitment

Example: Trusted GRUB (IBM'05)



What PCR # to use and what to measure specified in GRUB config file

Using PCR values after boot

Application 1: encrypted (a.k.a sealed) storage.

Step 1: TPM_TakeOwnership(OwnerPassword, …)

Creates 2048-bit RSA Storage Root Key (SRK) on TPM

Cannot run TPM_TakeOwnership again without OwnerPwd:

Ownership Enabled Flag ← False

Done once by IT department or laptop owner.

(optional) Step 2: TPM_CreateWrapKey / TPM_LoadKey

Create more RSA keys on TPM protected by SRK

Each key identified by 32-bit keyhandle

Protected Storage

Main Step: Encrypt data using RSA key on TPM

TPM_Seal (some) Arguments:

keyhandle: which TPM key to encrypt with

KeyAuth: Password for using key `keyhandle'

PcrValues: PCRs to embed in encrypted blob

data block: at most 256 bytes (2048 bits)

Used to encrypt symmetric key (e.g. AES)

Returns encrypted blob.

Main point: blob can only be decrypted with TPM_Unseal when PCR-reg-vals = PCR-vals in blob. TPM_Unseal will fail othrwise

Embedding PCR values in blob ensures that only some apps can decrypt data and messing with MBR or OS kernel will be detected as it changes PCR values.

Seal

- The Seal command takes a set of PCR indices as input, and encrypts the data provided using its Storage Root Key (Kroot), a key that never leaves the TPM
- It outputs the resulting ciphertext C, along with an integrity-protected list of the indicies provided and the values of the corresponding PCRs at the time Seal was invoked.
- For example, to seal a secret key Ksecret under the values stored in PCRs 1, 3, and 17, you would invoke:

Seal((1, 3, 17),Ksecret)
→ (C,MAC Kroot((1, PCR1), (3, PCR3), (17, PCR17))

- It is also possible to provide the Seal command not only with the PCR indices of interest, but also with the values those PCRs should have before Unseal will decrypt the data.

UnSeal

- The Unseal command takes in a ciphertext and PCR list created by the Seal command and verifies the integrity of the list of PCR values, and compares them against the current values of those PCRs
- If they match, it decrypts C and outputs the resulting data, otherwise the TPM simply returns an error.
- Unseal(C,MACKroot((1, PCR1), (3, PCR3), (17, PCR17)) will produce data if
  - PCR′ 1 = PCR1
  - PCR′ 3 = PCR3
  - PCR′ 17 = PCR17

  where PCR′ 1, PCR′ 3, and PCR′ 17 represent the current values of those PCRs.

Sealed storage: applications

- Lock software on machine:
  - OS and apps sealed with MBR's PCR.
  - Any changes to MBR (to load other OS) will prevent locked software from loading.
  - Prevents tampering and reverse engineering e.g. software integrity on voting terminals
- Web server: seal server's SSL private key

  Goal: only unmodified Apache can access SSL key

  Problem: updates to Apache or Apache config
- General problem with software patches because patch process must re-seal all blobs with new PCRs

Security?

1. Can attacker disable TPM until after boot, then extend PCRs with whatever he wants?

   Root of trust: BIOS boot block defeated with one byte change to boot block Oslo: Improving the Security of Trusted Computing Usenix 2007

1. Resetting TPM after boot (by sending TPM_Init on LPC bus) allows arbitrary values to be loaded onto PCR (requires physical access)

2. Other problems: roll-back attack on encrypted blobs to adopt a weak encryption scheme and undo security patches without being noticed.

   Can be mitigated using Data Integrity Regs (DIR) in the non volatile storage as it needs OwnerPassword to write DIR

Better root of trust

DRTM – Dynamic Root of Trust Measurement

AMD: **skinit** Intel: **senter**

To check the integrity of a VMM it atomically does:

Reset CPU. Reset PCR 17 to 0.
Load given Secure Loader (SL) code into I-cache
Extend PCR 17 with SL
Jump to SL

BIOS boot loader is no longer root of trust

Avoids TPM_Init attack: TPM_Init sets PCR 17 to -1

TPM Counters

TPM must support at least four hardware counters

Increment rate: every 5 seconds for 7 years.

Applications:

Provide time stamps on blobs.

Supports "music will pay for 30 days" policy.

Attestation

Attestation: what it does

Goal: prove to remote party what software is running on my machine.

Good applications:

1. Bank allows money transfer only if customer's machine runs "up-to-date" OS patches.
2. Enterprise allows laptop to connect to its network only if laptop runs "authorized" software
3. Quake players can join a Quake network only if their Quake client is unmodified.

DRM:

MusicStore sells content for authorized players only.

Attestation: how it works

Recall: EK private key on TPM.

Cert for EK public-key issued by TPM vendor

TPM_Quote . Obtains a signed report of the current PCR values	key handle; PCRs; external data	a signature on PCR values and the external data

Step 1: Create Attestation Identity Key (AIK)

Details not important here

AIK Private key known only to TPM

AIK public cert issued only if EK cert is valid

Attestation: how it works

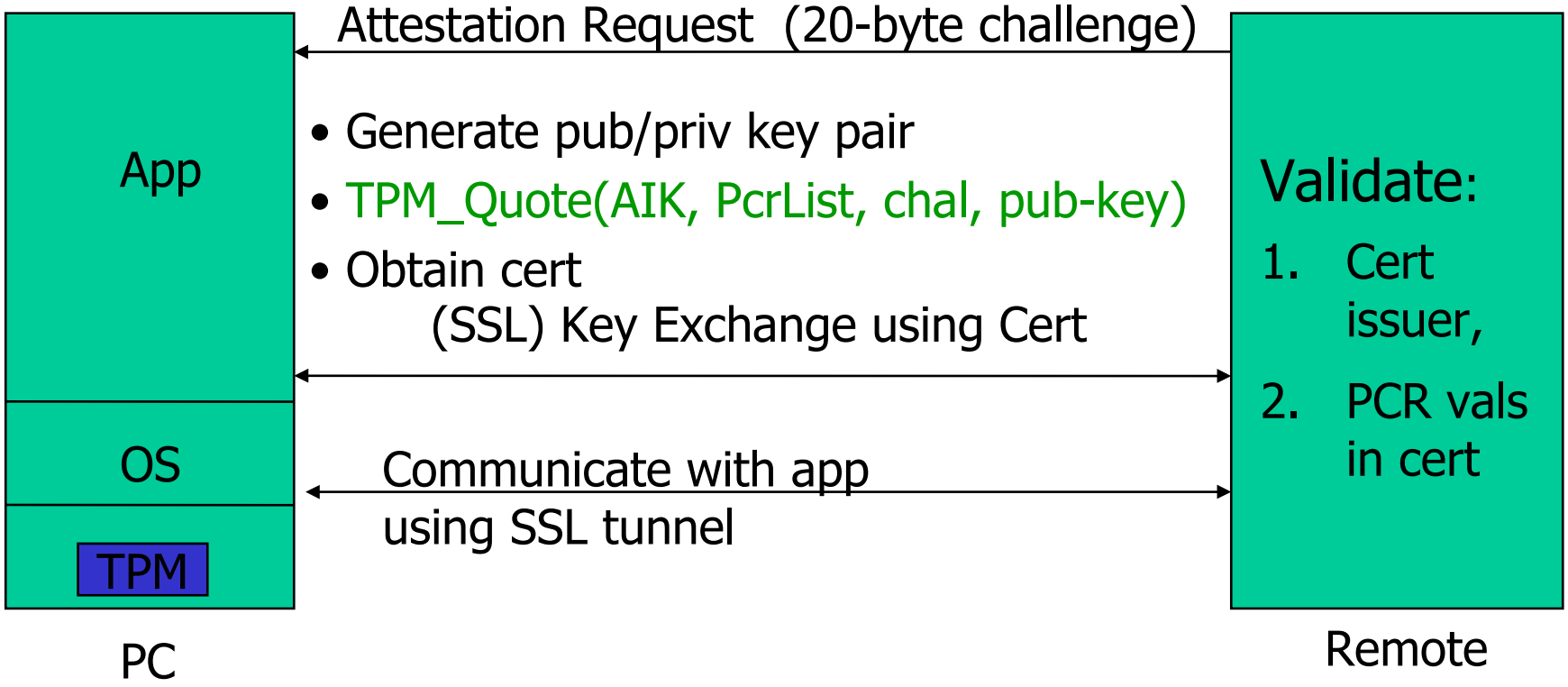
Step 2: sign PCR values (after boot)

Call TPM_Quote (some) Arguments:

- keyhandle: which AIK key to sign with
- KeyAuth: Password for using key `keyhandle'
- PCR List: Which PCRs to sign.
- Challenge: 20-byte challenge from remote server
  Prevents replay of old signatures.
- Userdata: additional data to include in sig.

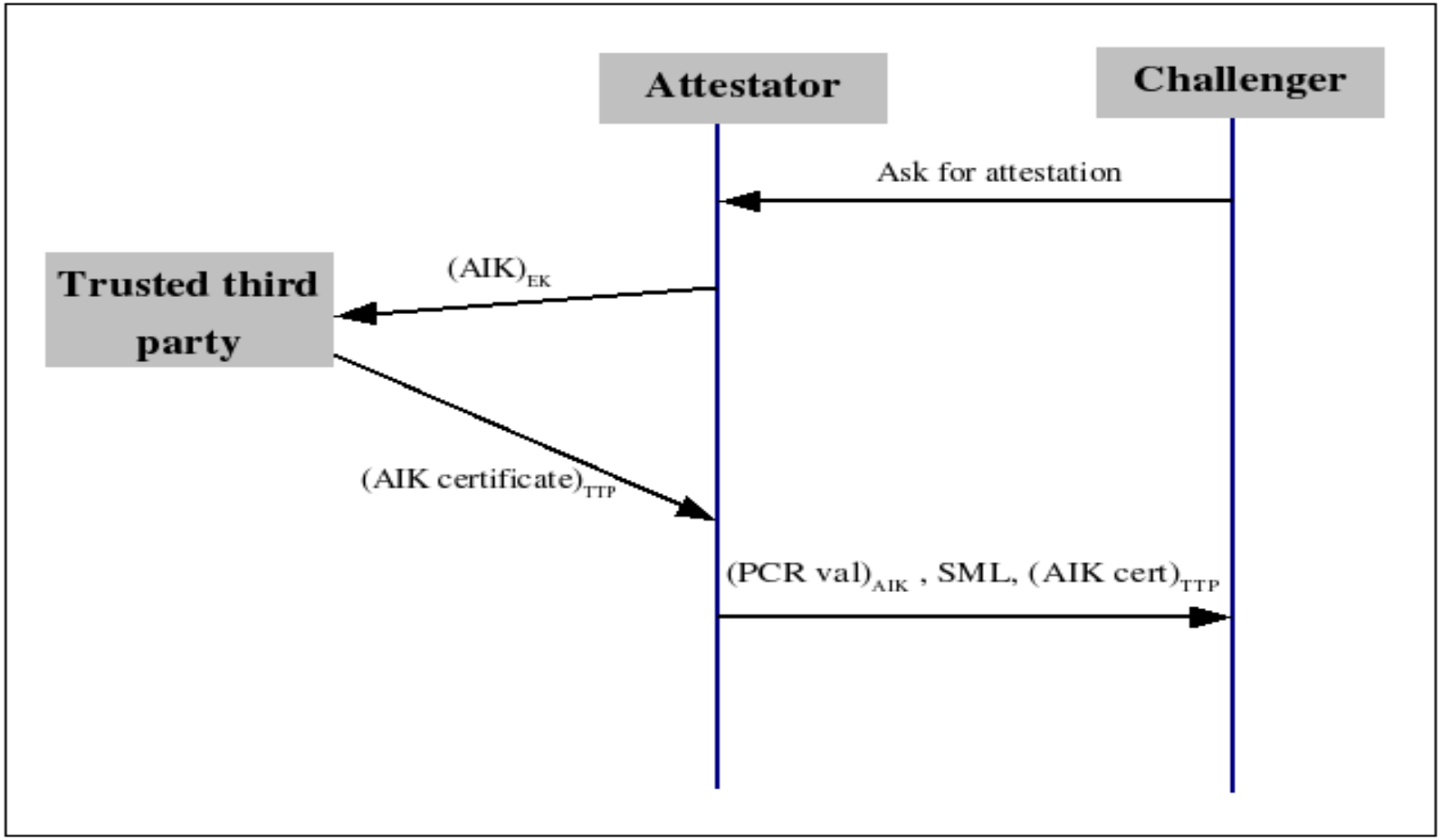
Returns signed data and signature.

Attestation: how it (should) work



- Attestation must include key-exchange
- App must be isolated from rest of system

Attestation



Attestation

- On receipt of a request for attestation,
  - the attestator generates a public/private key pair, the attestation identity key (AIK),
  - send the public part to a trusted third party (TTP = a *Privacy CA)* that generates a AIK certificate after validating the attestator's EK.
  - The certificate is sent back to the attestator that can now send its PCR values (signed with AIK) and the AIK certificate to the challenger
- The verification process in the challenger side:
  - Verify the AIK certificate with the TTP public key.
  - Use AIK to verify the signature on the PCR values.
  - Recalculate this value from the fingerprint list within SML (by applying the PCR's extend operation on these fingerprints).
- Compare the calculated value with PCR's value. A mismatch signals a tampering, and the verifier should not trust the attestator. Otherwise, the challenger can decide that the local party is trustworthy.

The security of the attestation report relies on the the AIK certified by the TTP on the basis of the EK. ⇒ the ***Root of Trust for Reporting (RTR)*** is the EK.

Attestation: Conditions

Three conditions must be met, to make a chain of hashes trustworthy:

- The first code running and extending PCRs after a platform reset (called SRTM) is trustworthy and cannot be replaced.
- The PCRs are not resetable, without passing control to trusted code.
- The chain is contiguous. There is no code inbetween that is executed but not hashed

Stored Measurement Log

- Include in the reply from the attestor to the challenger a list of SML Stored Measurement Log

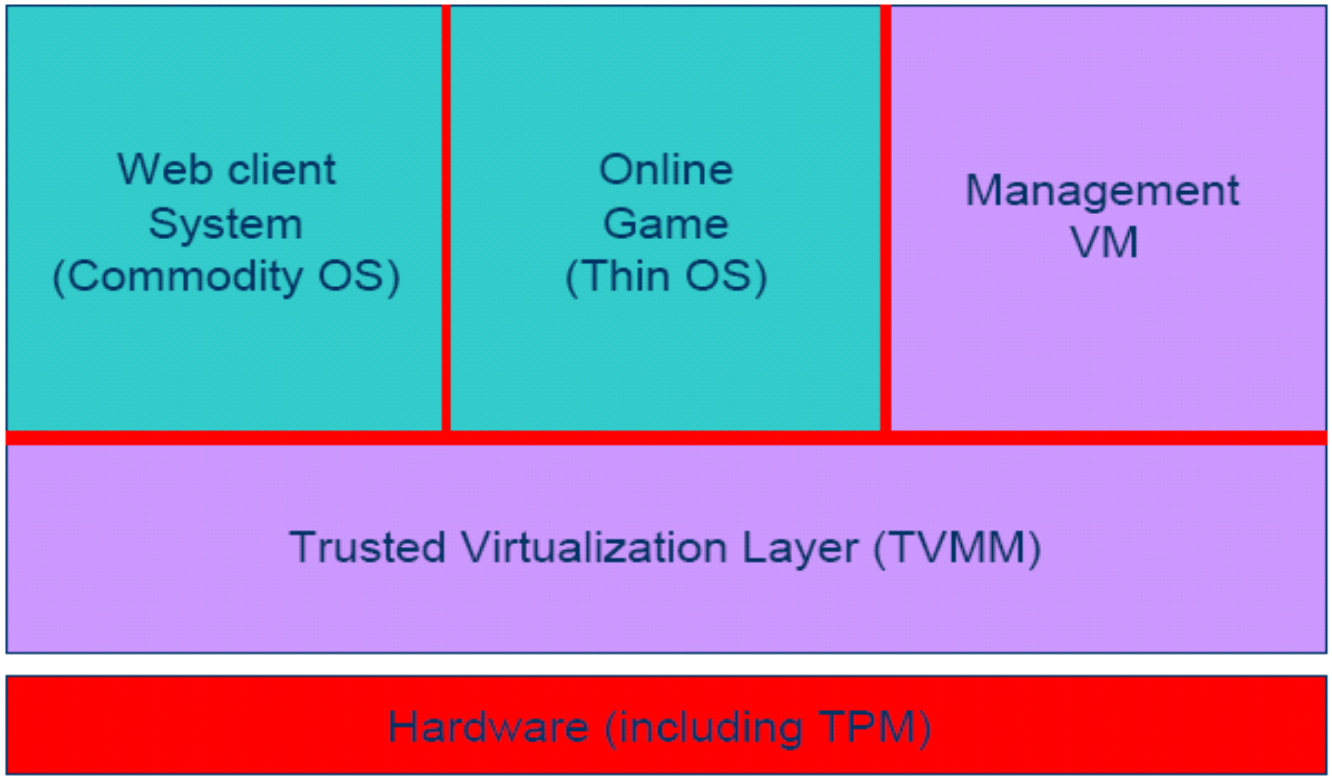
- Measuring is done by hashing the entity with a hash function. The result will be the *measured value* of that entity. An entity in a PC platform could be an application executable, a configuration file or a data file.

Considering two entities A and B:

- A measures entity B (could be executable or other files ...).
- Result is a B's "fingerprint“ that fingerprint is stored in SMlog which resides in the hard drive (outside, and not protected by a TPM).
- A then inserts B's fingerprint into a PCR (via the PCR's *extend* operation).
- Control is passed to B.

Using Attestation

Attesting to VMs: Terra [SOSP'03]



TVMM Provides isolation between attested applications

Nexus OS (Sirer et al. '06)

Problem: attesting to hashed application/kernel code

Too many possible software configurations

Better approach: attesting to properties

Example: "application never writes to disk"

Supported in Nexus OS (Sierer et al. '06)

General attestation statements:

"TPM says that it booted Nexus,
Nexus says that it ran checker with hash X,
checker says that IPD A has property P"

IBM **Integrity Measurement Architecture (IMA)**

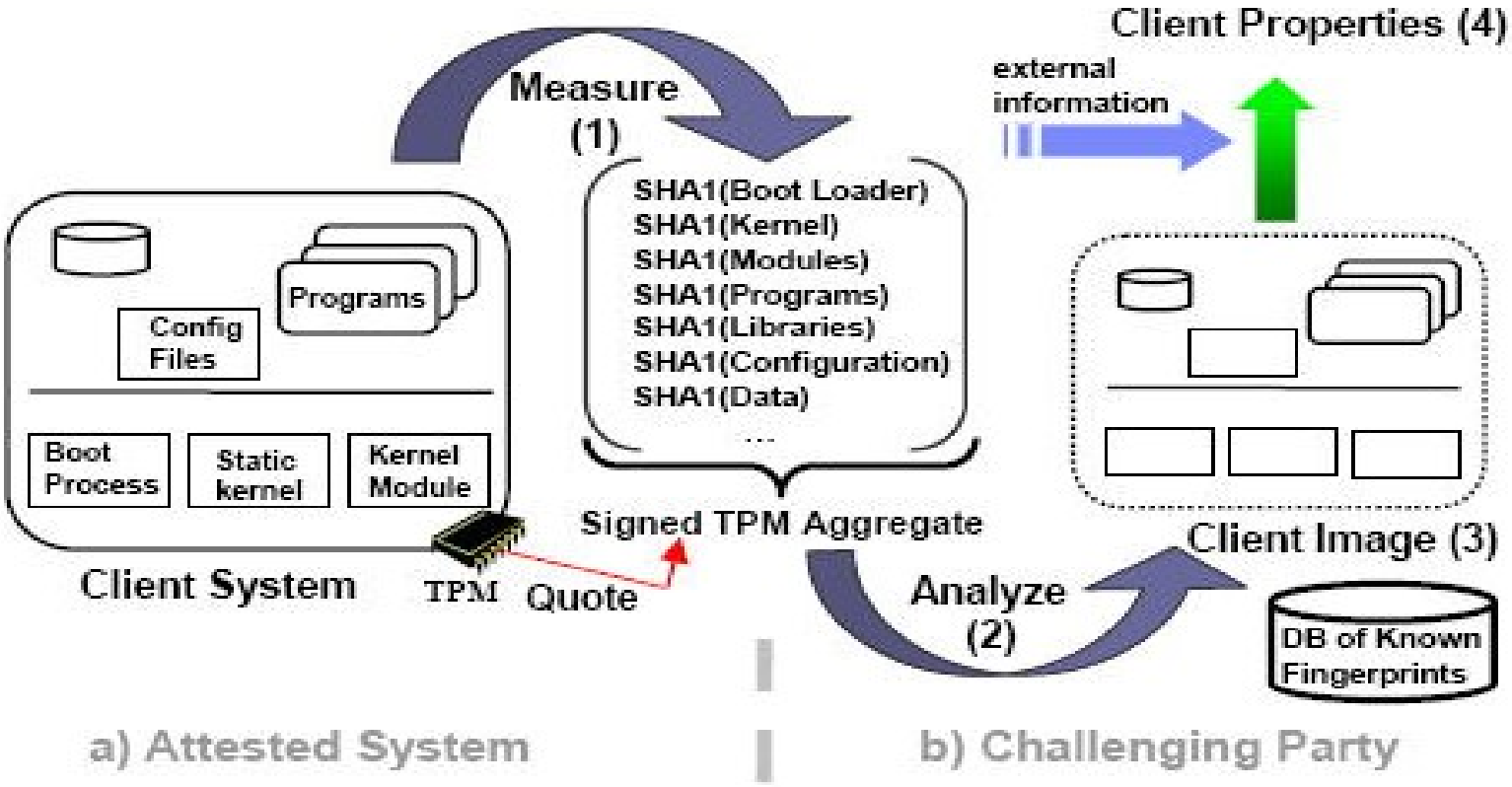


Figure 1: Attestation Architecture Overview

Elettronic Frontier Foundation: Owner Override

TCG attestation:

The good: enables user to prove to remote bank that machine is up-to-date

The bad: content owners can release decryption key only to machines running "authorized" software.

Stifles innovation in player design

EFF proposal: allow users to inject chosen values into PCRs.

Enables users to conceal changes to their computing environment

Defeats malicious changes to computing platform

TCG Alternatives

IBM 4758: Supports all TCG functionality and more.

Tamper resistant 486 100MhZ PCI co-processor.

Programmable.

… but expensive ~ $2000. TPM ~ $7.

AEGIS System: Arbaugh, Farber, Smith ’97:

Secure boot with BIOS changes only.

Cannot support sealed storage.

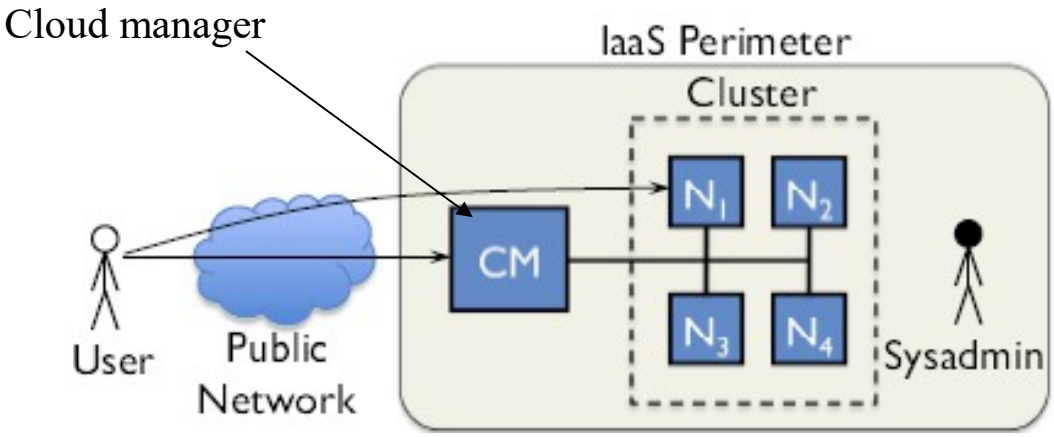
Phoenix TrustConnector 2

SWATT: Seshadri et al., 2004

Attestation w/o extra hardware

Server must know precise HW configuration

Trusted Cloud Computing Platform by Santos et al., HotCloud 09



Problem Insiders with root access can compromise confidentiality of client virtual machines

Possible solution?: Encrypt virtual machines, but sooner or later, it has to be decrypted to run

Possible Solution?: Only allow nodes running trustworthy software to decrypt the VM

Threat Model

Attacker

- A malicious insider with root level access to cloud nodes (i.e., can install new software, modify existing software etc., inspect VMs running on a node)
- Does not have physical access to machine

How determine if a node is trustworthy?

Major events that causes changes

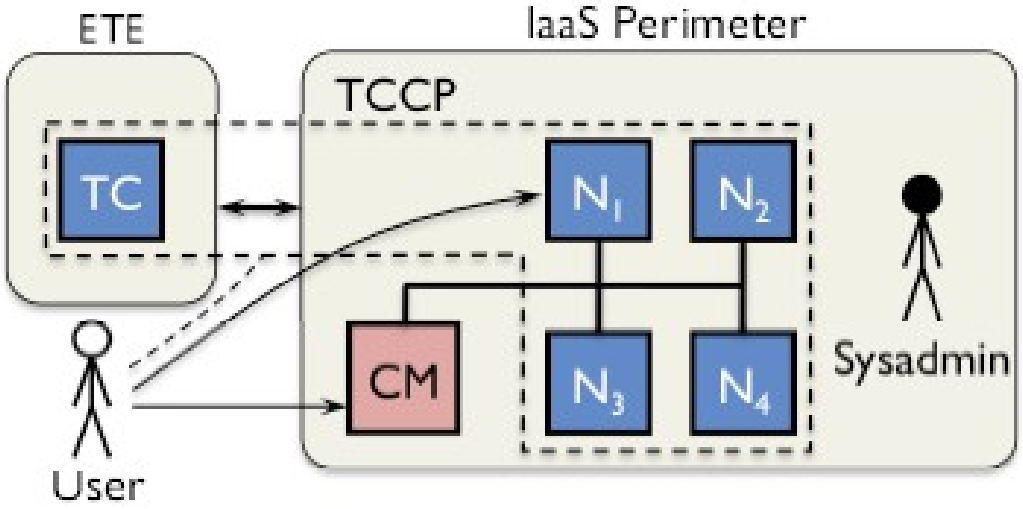
Node start, VM Launch, VM migration

How to determine trustworthiness?

A node is trustworthy if it has a trustworthy configuration (e.g., h/w, software etc.)

Remote attestation can help in verifying configurations

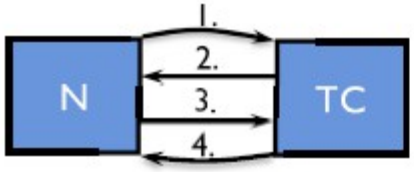
TCCP Architecture



Nodes run Trusted Virtual Machine Monitors (TVMM), and TVVM configuration can be certified by TPMs

External Trusted Entity (ETE) or the Trusted Coordinator (TC) is the trusted third party that verifies the node

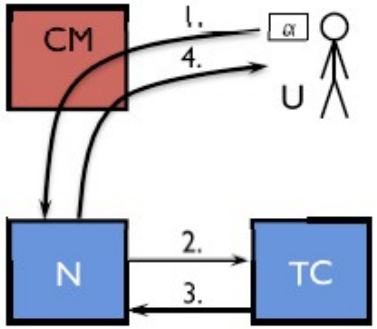
TCCP Protocols



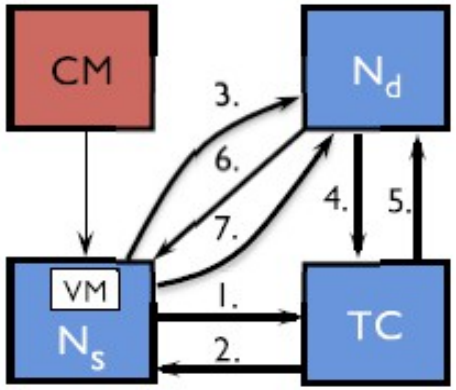
1. n_N
2. $\{ML_{TC}, n_N\}_{EK^P_{TC}}, n_{TC}$
3. $\{\{ML_N, n_{TC}\}_{EK^P_N}, TK^P_N\}_{TK^P_{TC}}$
4. $\{accepted\}_{TK^P_N}$

Node registration

VM Launch



1. $\{\alpha, \#\alpha\}_{K_{VM}}\{n_U, K_{VM}\}_{TK^P_{TC}}$
2. $\{\{\{n_U, K_{VM}\}_{TK^P_{TC}}, n_N\}_{TK^P_N}, N\}_{TK^P_{TC}}$
3. $\{\{n_N, n_U, K_{VM}\}_{TK^P_N}\}_{TK^P_{TC}}$
4. $\{n_U, N\}_{K_{VM}}$



1. $\{\{N_d, n_{s1}\}_{TK^P_N}, N_s\}_{TK^P_{TC}}$
2. $\{\{n_{s1}, TK^P_{N_d}\}_{TK^P_{N_s}}\}_{TK^P_{TC}}$
3. $\{\{K_S, n_{s2}\}_{TK^P_{N_s}}, N_s\}_{TK^P_{N_d}}$
4. $\{\{N_s, n_d\}_{TK^P_{N_d}}, N_d\}_{TK^P_{TC}}$
5. $\{\{n_d, TK^P_{N_s}\}_{TK^P_{N_d}}\}_{TK^P_{TC}}$
6. $\{n_d\}_{K_S}$
7. $\{VM_{id}, \#VM_{id}\}_{K_S}$

VM to be migrated is encrypted by a key that the TC reveals to a trusted machine only

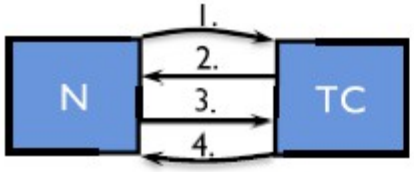
VM Migrate

Registration

1. N sends a challenge n_N to the TC
2. TC replies with its bootstrap measurements ML_{TC} encrypted with EK^{p}_{TC} to guarantee the authenticity of the TC. If the ML_{TC} matches the expected configuration, the TC is trusted
3. The node generates a keypair (TK^{pub}_{N}, TK^{Priv}_{N}), and sends its public key to the TC together with its list of measurements. If both peers mutually attest successfully, the TC adds TK^{pub}_{N} to its node database, and sends message 4
4. confirm that the node is trusted. Key TK^{pub}_{N} certifies that node N is trusted.

A node should keeps its private key TK^{Priv}_{N} in memory so that it lost when the node reboots

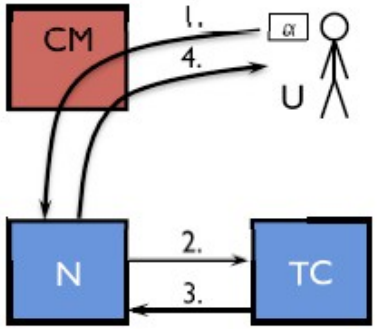
TCCP Protocols



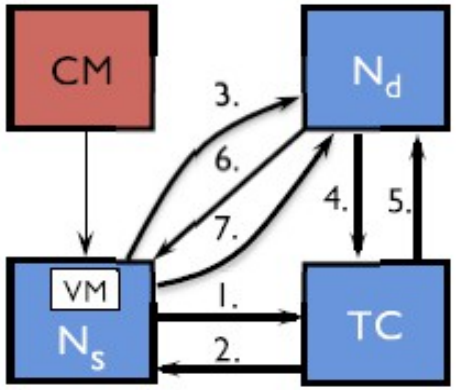
1. n_N
2. $\{ML_{TC}, n_N\}_{EK^p_{TC}}, n_{TC}$
3. $\{\{ML_N, n_{TC}\}_{EK^p_N}, TK^P_N\}_{TK^P_{TC}}$
4. $\{accepted\}_{TK^P_N}$

Node registration

VM Launch



1. $\{\alpha, \#\alpha\}_{K_{VM}}\{n_U, K_{VM}\}_{TK^P_{TC}}$
2. $\{\{\{n_U, K_{VM}\}_{TK^P_{TC}}, n_N\}_{TK^P_N}, N\}_{TK^P_{TC}}$
3. $\{\{n_N, n_U, K_{VM}\}_{TK^P_N}\}_{TK^P_{TC}}$
4. $\{n_U, N\}_{K_{VM}}$



1. $\{\{N_d, n_{s1}\}_{TK^P_N}, N_s\}_{TK^P_{TC}}$
2. $\{\{n_{s1}, TK^P_{N_d}\}_{TK^P_{N_s}}\}_{TK^P_{TC}}$
3. $\{\{K_S, n_{s2}\}_{TK^P_{N_s}}, N_s\}_{TK^P_{N_d}}$
4. $\{\{N_s, n_d\}_{TK^P_{N_d}}, N_d\}_{TK^P_{TC}}$
5. $\{\{n_d, TK^P_{N_s}\}_{TK^P_{N_d}}\}_{TK^P_{TC}}$
6. $\{n_d\}_{K_S}$
7. $\{VM_{id}, \#VM_{id}\}_{K_S}$

VM Migrate

VM launch - I

- The user generates a session key K_{VM}, and sends message 1 to the CM $\alpha = \text{VM Image}$ and α 's hash encrypted with K_{VM} (to protect the confidentiality/integrity of the image), and K_{VM} encrypted with TK^{P}_{TC}.
- Hence, only the TC can authorize someone to access α and the TC only authorizes trusted nodes.
- Upon receiving the request , the CM designates a node N from the cluster to host the VM, and forwards the request to N.
- Since N has to access α to boot the VM, it sends message 2 to TC which decrypts K_{VM} on N's behalf.
- The message is encrypted with TK^{P}_{N} so that TC can verify whether N is trusted. If the corresponding public key is not found in the TC's trusted node database, the request is denied.

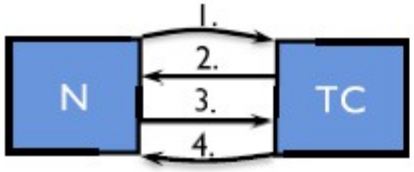
VM launch -II

- If N is trusted; the TC decrypts the session key, and sends it to the node in message 3. Only N can read the key, decrypt α and boot the VM.
- N sends message 4 to the user containing the identity of the node running the VM.

VM migration

- To secure the transfer of the state of an executing VM from Ns to Nd,
  - both nodes must be trusted,
  - the transit over the network should be confidential and secure.
- In steps 1 and 2, Ns asks TC to check whether Nd is trusted.
- Ns negotiates a session key KS with Nd (message 3)
- Before accepting the key, Nd verifies that Ns is trusted (steps 4 and 5).
- If both nodes mutually authenticate successfully,
  - Nd acknowledges the acceptance of the session key KS (step 6),
  - Ns finally transfers the encrypted and hashed VM state to the Nd (step 7), guaranteeing the confidentiality and integrity of the VM.

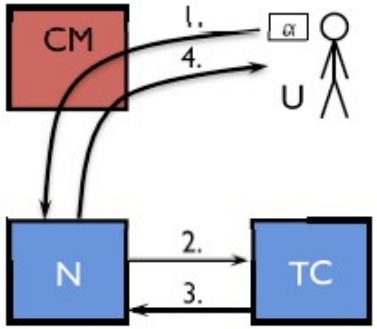
TCCP Protocols



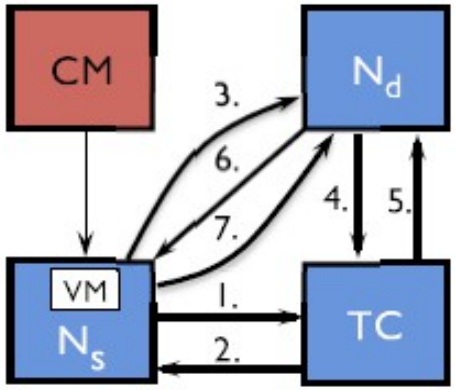
1. n_N
2. $\{ML_{TC}, n_N\}_{EK^P_{TC}}, n_{TC}$
3. $\{\{ML_N, n_{TC}\}_{EK^P_N}, TK^P_N\}_{TK^P_{TC}}$
4. $\{accepted\}_{TK^P_N}$

Node registration

VM Launch



1. $\{\alpha, \#\alpha\}_{K_{VM}}\{n_U, K_{VM}\}_{TK^P_{TC}}$
2. $\{\{\{n_U, K_{VM}\}_{TK^P_{TC}}, n_N\}_{TK^P_N}, N\}_{TK^P_{TC}}$
3. $\{\{n_N, n_U, K_{VM}\}_{TK^P_N}\}_{TK^P_{TC}}$
4. $\{n_U, N\}_{K_{VM}}$



1. $\{\{N_d, n_{s1}\}_{TK^P_N}, N_s\}_{TK^P_{TC}}$
2. $\{\{n_{s1}, TK^P_{N_d}\}_{TK^P_{N_s}}\}_{TK^P_{TC}}$
3. $\{\{K_S, n_{s2}\}_{TK^P_{N_s}}, N_s\}_{TK^P_{N_d}}$
4. $\{\{N_s, n_d\}_{TK^P_{N_d}}, N_d\}_{TK^P_{TC}}$
5. $\{\{n_d, TK^P_{N_s}\}_{TK^P_{N_d}}\}_{TK^P_{TC}}$
6. $\{n_d\}_{K_S}$
7. $\{VM_{id}, \#VM_{id}\}_{K_S}$

VM Migrate

TCCP limitations

Any single point of failure?

Will it increase the attack surface?

How about cost-effectiveness?

Private Virtual Infrastructure Krautheim, HotCloud09

Problem The abstraction of a cloud hides the internal security details from clients, which in turn causes them to mistrust the cloud.

Idea

- Cloud provider and client collaborate to create a trusted system.
- Separate the different clients through their exclusive private virtual infrastructures
- Give more control to the cloud clients

Separating the Fabric from the Cloud

- The IaaS fabric layer provides computation resources managed by the service provider, while the PVI layer provides a virtual datacenter managed by the client.

- The service provider assumes responsibility for providing the physical security and the logical security of the service platform required for the PVI layer

- Each client is responsible for securely provisioning their virtual infrastructure with appropriate firewalls, intrusion detection systems, monitoring and logging to ensure that data is kept confidential. PVI enables the client to build a virtual infrastructure that meets these requirements

Separating the Fabric from the Cloud

- The PVI Factory is the most sensitive component of the PVI.
- The factory is where all components of the PVI are provisioned and it is the root authority for
  - provisioning,
  - VTPM key generation
  - certificate generation
  - management within the PVI.
- The factory also maintains master images for application servers, and handles data transfers to the PVI through the VPN configuration and management

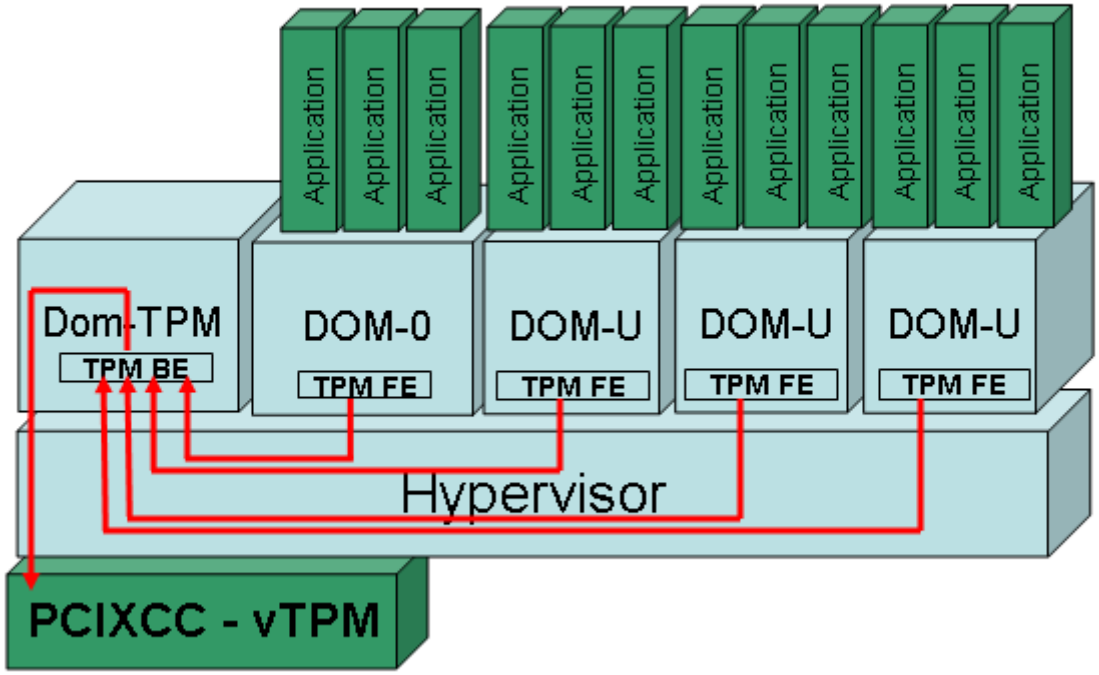
5 Tenets of Cloud Computing Security

- Provide a trusted foundation on which to build PVI. This is accomplished through the service level agreement with the service provider assuring they will provide the security services to protect the information with PVI.
- Provide a secure factory to provision PVI. The factory also serves as a policy decision point and root authority for PVI.
- Provide a measurement mechanism to validate the security of the fabric prior to provisioning of PVI.
- Provide secure methods for shutdown and destruction of virtual devices in PVI to prevent object reuse attacks.
- Provide continuous monitoring and auditing from within PVI as well as from outside of PVI with intrusion detection systems and other devices.

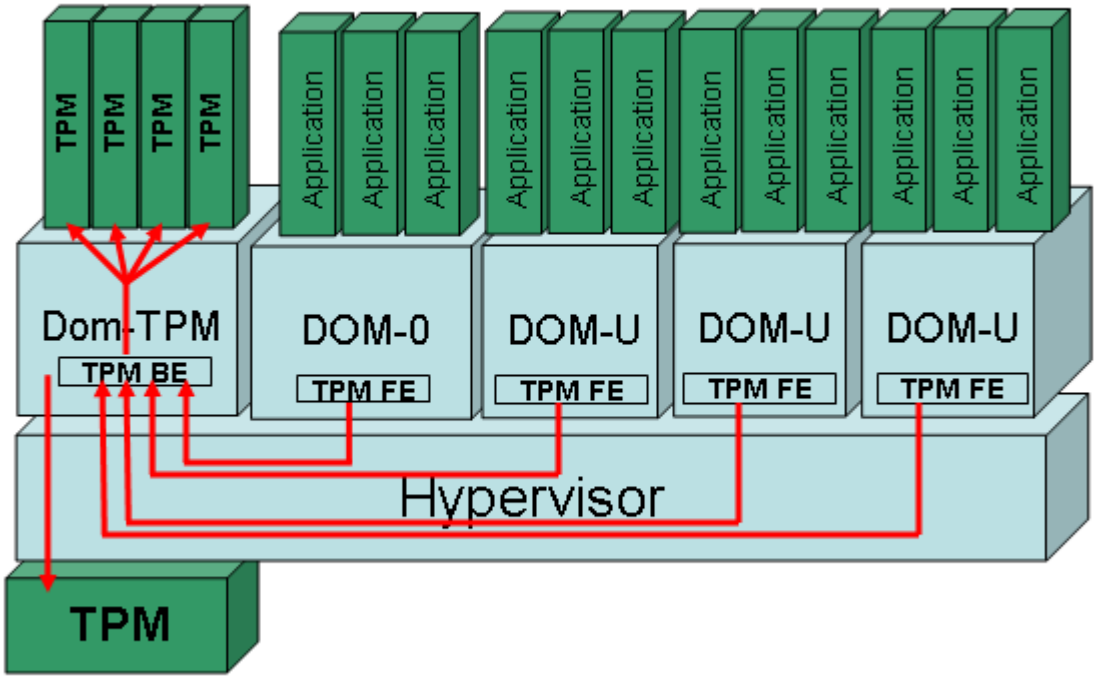
Trusted Infrastructure and TPM

- Trusted computing provides mechanisms to control the behavior of computer systems through enforcement of security policies
- By requiring providers to use TC technology, organizations can verify their security posture and control their information, allowing them to achieve the economies of scale, availability, and agility of the cloud.
- One problem associated with the TPM is that it only works for non-virtualized environments. If virtualization is used, which is a common occurrence in cloud services, the TPM also needs to be virtualized.
- Specifications have been developed for a virtual TPM implemented by providing software instances of TPMs for each VM on a trusted platform
- PVI uses TPMs as the basis for trust in the cloud. Individual cloud nodes each have a TPM owned by the service provider. VTPMs are linked to the physical TPM and used to secure each VM in the cloud.

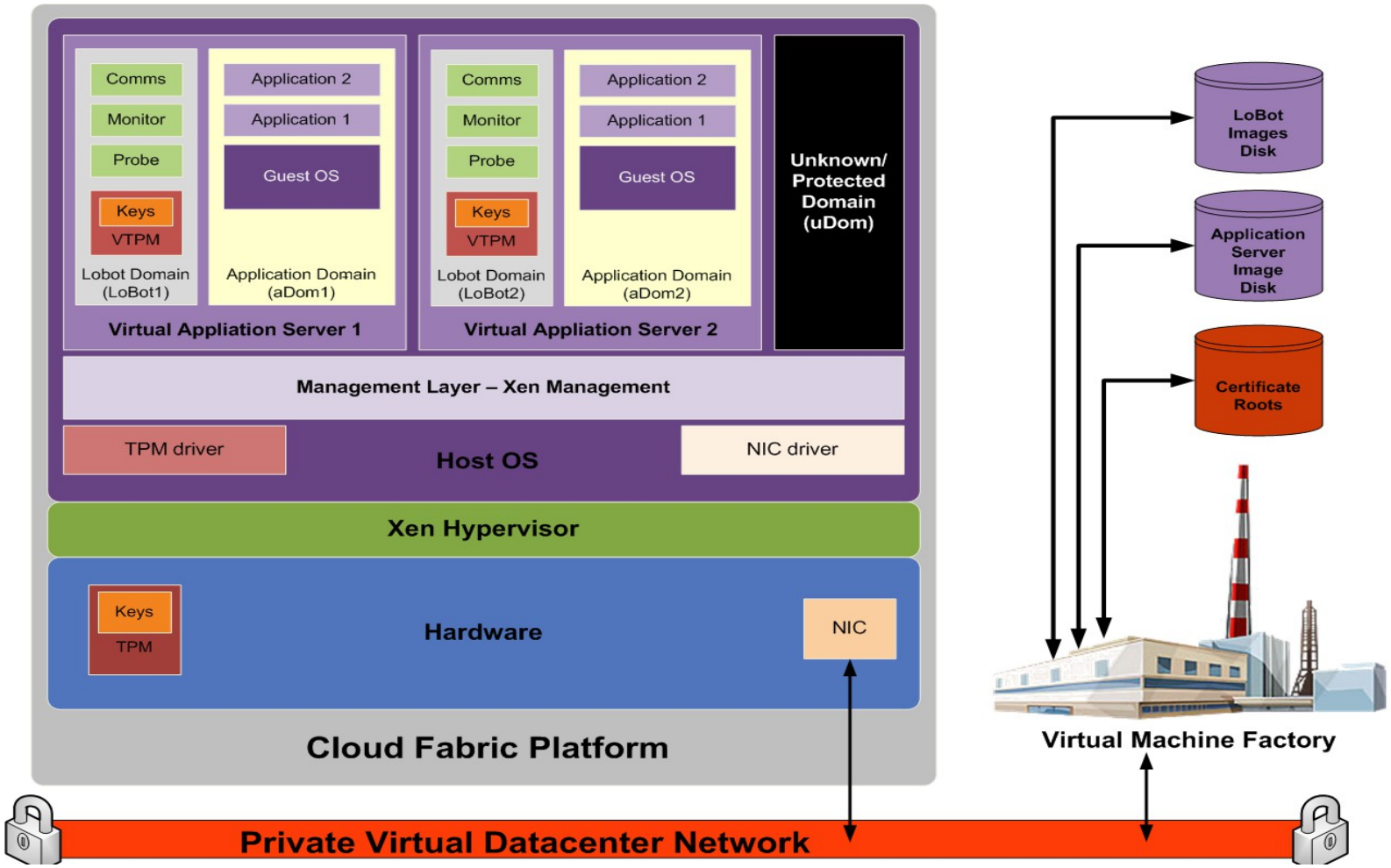
Virtual TPM


Application
Application
Application
Application
Application
Application
Application
Application
Application
Application
Application
Application
Dom-TPM
TPM BE
DOM-0
TPM FE
DOM-U
TPM FE
DOM-U
TPM FE
DOM-U
TPM FE
Hypervisor
PCIXCC - vTPM

Virtual TPM


TPM
TPM
TPM
TPM
Application
Application
Application
Application
Application
Application
Application
Application
Application
Application
Application
Application
Dom-TPM
TPM BE
DOM-0
TPM FE
DOM-U
TPM FE
DOM-U
TPM FE
DOM-U
TPM FE
Hypervisor
TPM

PVI Model


Comms
Monitor
Probe
Keys
VTPM
Lobot Domain (LoBot1)
Application 2
Application 1
Guest OS
Application Domain (aDom1)
Virtual Appliation Server 1
Comms
Monitor
Probe
Keys
VTPM
Lobot Domain (LoBot2)
Application 2
Application 1
Guest OS
Application Domain (aDom2)
Virtual Appliation Server 2
Unknown/ Protected Domain (uDom)
Management Layer – Xen Management
TPM driver
Host OS
NIC driver
Xen Hypervisor
Keys
TPM
Hardware
NIC
Cloud Fabric Platform
LoBot Images Disk
Application Server Image Disk
Certificate Roots
Virtual Machine Factory
Private Virtual Datacenter Network