Trusted Computing: Introduction & Applications

Lecture 3: The CRTM and authenticated boot process

Dr. Andreas U. Schmidt

Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany

Literature

1. Trusted Computing Group: TPM Main Part 1 Design Principles, Version 1.2, 2006.
2. C. J. Mitchell (ed.), *Trusted Computing*. IEE Press, 2005.
3. D. C. Blight. Trusted Computing. Blackhat Windows 2004. https://www.blackhat.com/presentations/win-usa-04/bh-win-04-blight/bh-win-04-blight.pdf
4. Eimear Gallery, Graeme Proudler Lecture at Royal Holloway: http://www.isg.rhul.ac.uk/files/IY5608_-_Lecture_2_Roots_of_Trust.pdf http://www.isg.rhul.ac.uk/files/IY5608_-_Lecture_3_Roots_of_Trust.pdf
5. David Grawrock, The Intel Safer Computing Initiative: Building Blocks for Trusted Computing, Intel Press, 2006)
6. How To Build Hardware Support For Secure Startup. Steve Heil & Mark Williams (Microsoft), Manny Novoa (Hewlett-Packard) at WinHEC05
7. Trent Jaeger, Reiner Sailer, Umesh Shankar PRIMA: Policy Reduced Integrity Measurement Architecture. *SACMAT'06*
8. Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA, pages 223-238, 2004. http://domino.research.ibm.com/comm/research_people.nsf/pages/sailer.ima.html http://domino.research.ibm.com/comm/research_projects.nsf/pages/ssd_ima.index.html/$FILE/sailer_usenix_security_2004_slides.pdf
9. Clark-Wilson integrity model http://www.computing.dcu.ie/~davids/courses/CA548/C_I_Policies.pdf
10. William A. Arbaugh, David J. Farbert, Jonathan M. Smith: A Secure and Reliable Bootstrap Architecture (AEGIS) http://www.cs.umd.edu/~waa/pubs/oakland97.pdf
11. Megumi Nakamura Seiji Munetoh: Designing a Trust Chain for a Thin Client on a Live Linux CD SAC'07 http://unit.aist.go.jp/itri/knoppix/http-fuse/LinuxKongress2006-suzaki.pdf http://delivery.acm.org/10.1145/1250000/1244343/p1605-nakamura.pdf http://www.trl.ibm.com/projects/watc/20061130d-WATC-Munetoh-Paper.pdf

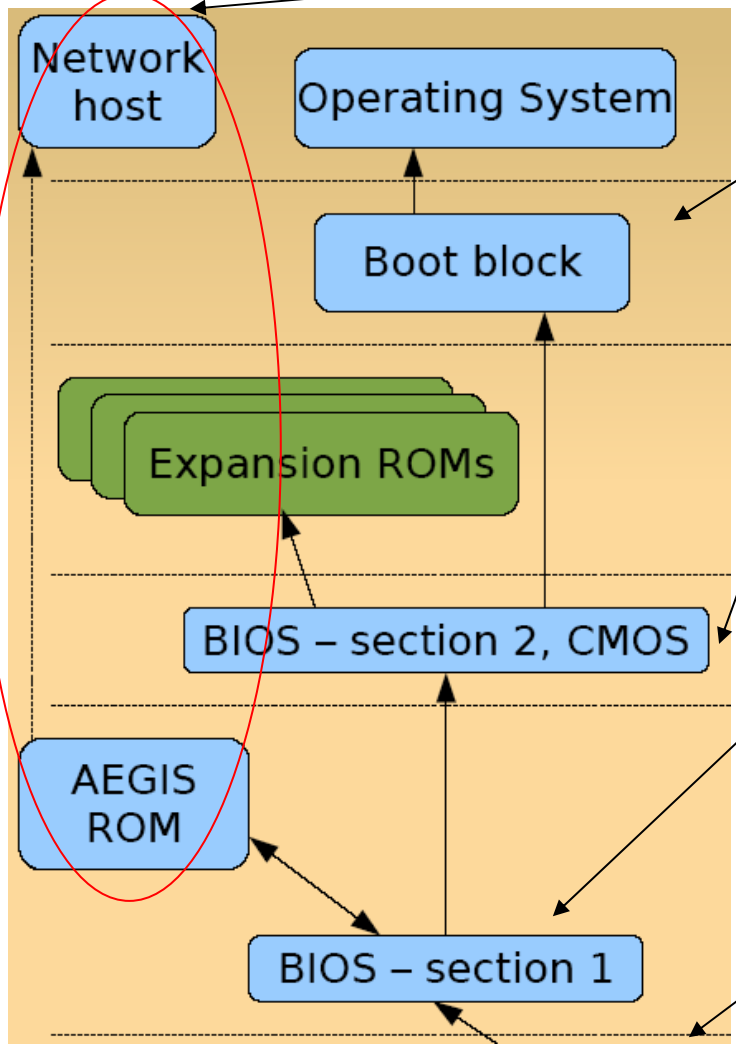
Clark-Wilson integrity model

- Verification Scope
  - Usually all code
  - May also be limited under some circumstances (policy)
- Executable Content measured
  - Independent of the method of loading
  - Modules, libraries, application loaded code
- Structured Data
  - The data is “known”
  - Handled similarly as executable content
- Unstructured Data
  - Integrity depends on the code that has modified it
  - Can challenger trust application that loads low integrity data?
- In addition, measurement list must be
  - fresh and complete
  - Unchanged

AEGIS system (1997)

- AEGIS (1997)
  - Ensure integrity of the bootstrap code, fom power-on until control goes to OS
  - Recovery mechanism
    - ROM
    - Trusted network host
- Assumptions
  - **Motherboard**, **processor**, parts of **BIOS** are not compromised
    - expansion card trusted, containing
  - Copies of essential parts of boot process
  - Optionally, a small OS for retrieving components from a trusted host
  - Certificate authorization infrastructure
  - Vendor keys
  - A trusted host as a source for recovery
- Integrity verification: signed hashes
  - Values stored on ROM, signed by certified vendor
  - RSA keys (-> later X.509v3, ISAKMP)
- No code is executed unless
  - It is trusted
  - Its integrity is verified before use
- Detect **integrity failure**
  - the failed code has to be replaced

AEGIS architecture



Fallback recovery

If secondary boot block needed, it is Verified. Kernel code is verified, and if it is OK, control is passed to it.

Expansion ROMs crypto hash calculated and verified against stored values.Bootstrap code finds the bootable device, verifies the boot block.

Checksum calculation: possible ROM Failures. Next section cryptographic hash is calculated (BIOS sect. 2 and CMOS).
If OK, control is passed to it.

Power on
- Hardware reset
- Warm boot (ctrl-alt-del under DOS)
- Software programs, if permitted by the OS

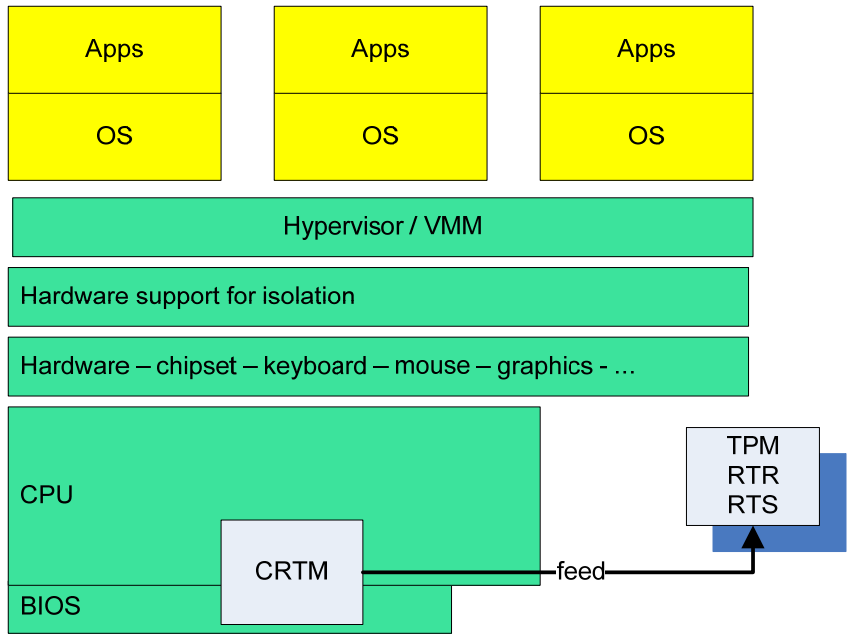
Power on self-test

AEGIS performance

- Performance
  - Signature verifications ~0.003s – 0.03s
  - Not bad when compared to total boot time
  - With “slow” computer: Pentium 90Mhz
- Problems
  - Floppy disk boot problem
  - tampered floppy disks
- no firmware is verified before boot (e.g. unauthorized expansion cards)

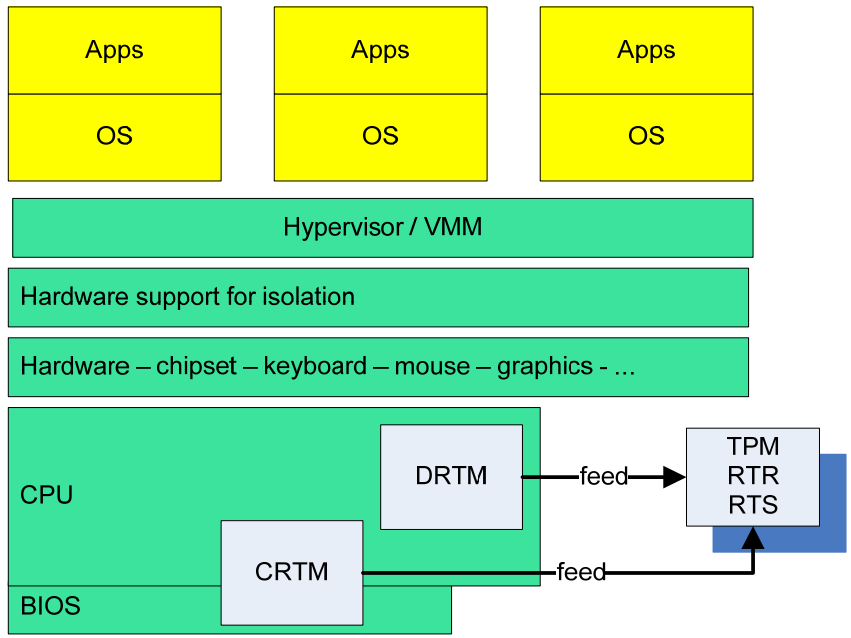
CRTM / DRTM

- The **CRTM** is the static root of trust for measurement.
  - Generically: Static RTM is CPU after platform reset, executing BIOS (extensions)
  - The CRTM is a genuine platform feature
- A RTM is a function that executes on the platform when the previous history of the platform cannot affect the future of the platform.
- Trusted to properly measure the first software/firmware that executes after some sort of reset and report this integrity measurement to the TPM.
- In a traditional system architecture –this function executes after a platform reset.
- With the advent of system partitioning/compartment isolation –boot process is no longer linear.



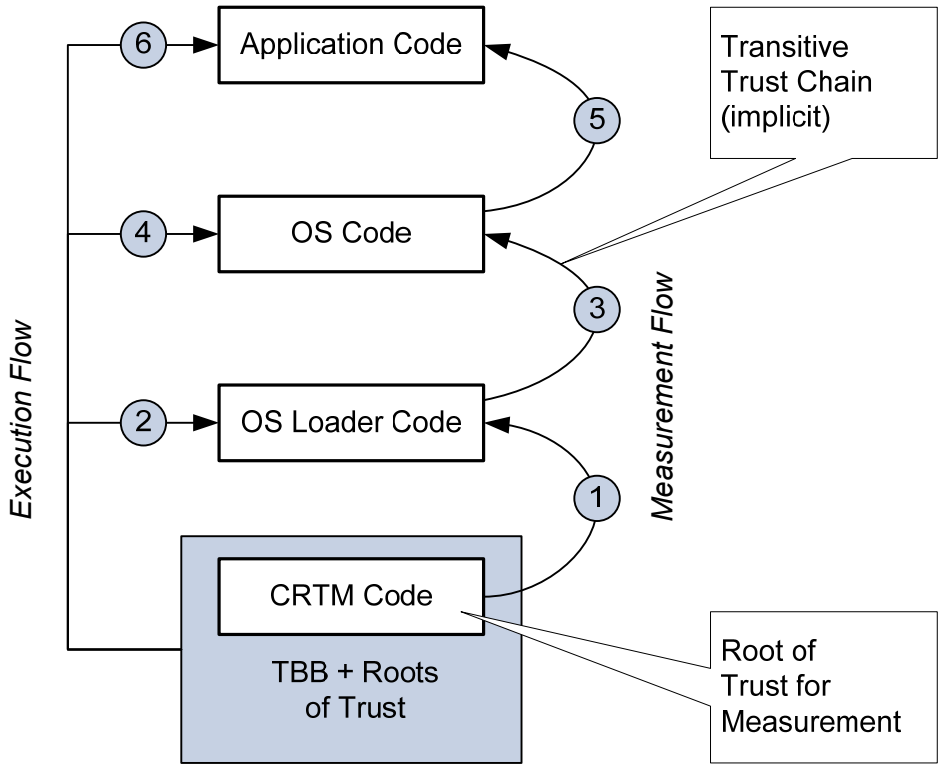
DRTM

- Isolated execution environments/system partitions may be brought up and taken down:
- The history of a the platform prior to launch of the isolation layer cannot effect the future of the isolation layer; or, indeed,
- Any previous compartment/ isolated execution environment cannot effect the future of any isolated compartments/ execution environments.
- With this, the concept of a dynamic RTM was introduced.
- Dynamic RTM is CPU after compartment reset.
- For instance:
- DRTM –Measures the VMM prior to its launch.
- Events occurring prior to the VMM launch do not effect the launch.



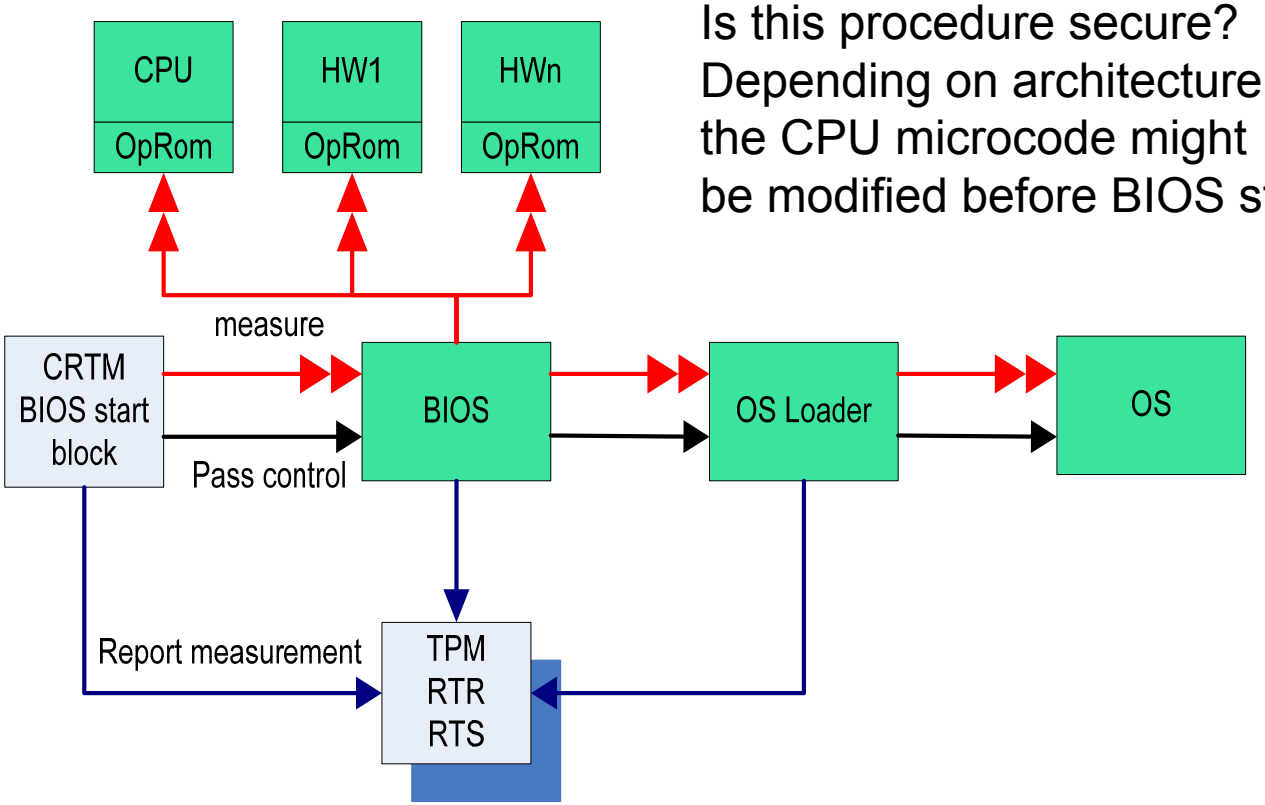
Transitive Trust

- Enables extension of the trust boundary
  - Each layer is responsible for determining the trustworthiness of the following
  - Reports this to the TPM



Authenticated boot overview

- The process by which measurements about the firmware/software state of the platform (integrity measurements) are reliably **measured** and **stored** (but not checked).



Is this procedure secure?
Depending on architecture,
the CPU microcode might
be modified before BIOS starts

PCRs

- Platform configuration registers (PCRs)
  - Used to store condensed software integrity measurements of a platform (called integrity metrics).
  - A TP must have a minimum of sixteen PCRs (PCR0 – PCR15).
  - Each storage register has a length equal to the SHA-1
  - digest: 20 bytes.
  - Each PCR holds a summary value of all the measurements
  - presented to it:
    - Less expensive than holding all individual measurements in the TPM.
    - This means that an unlimited number of results can be stored.
    - The fewer sequences/PCRs there are, the more difficult it is to determine the meaning of the sequence.
    - The more there are, the more costly it is to store sequences in the TPM.
  - A PCR must be a TPM shielded location, protected from interference and prying.

PCR usage and update

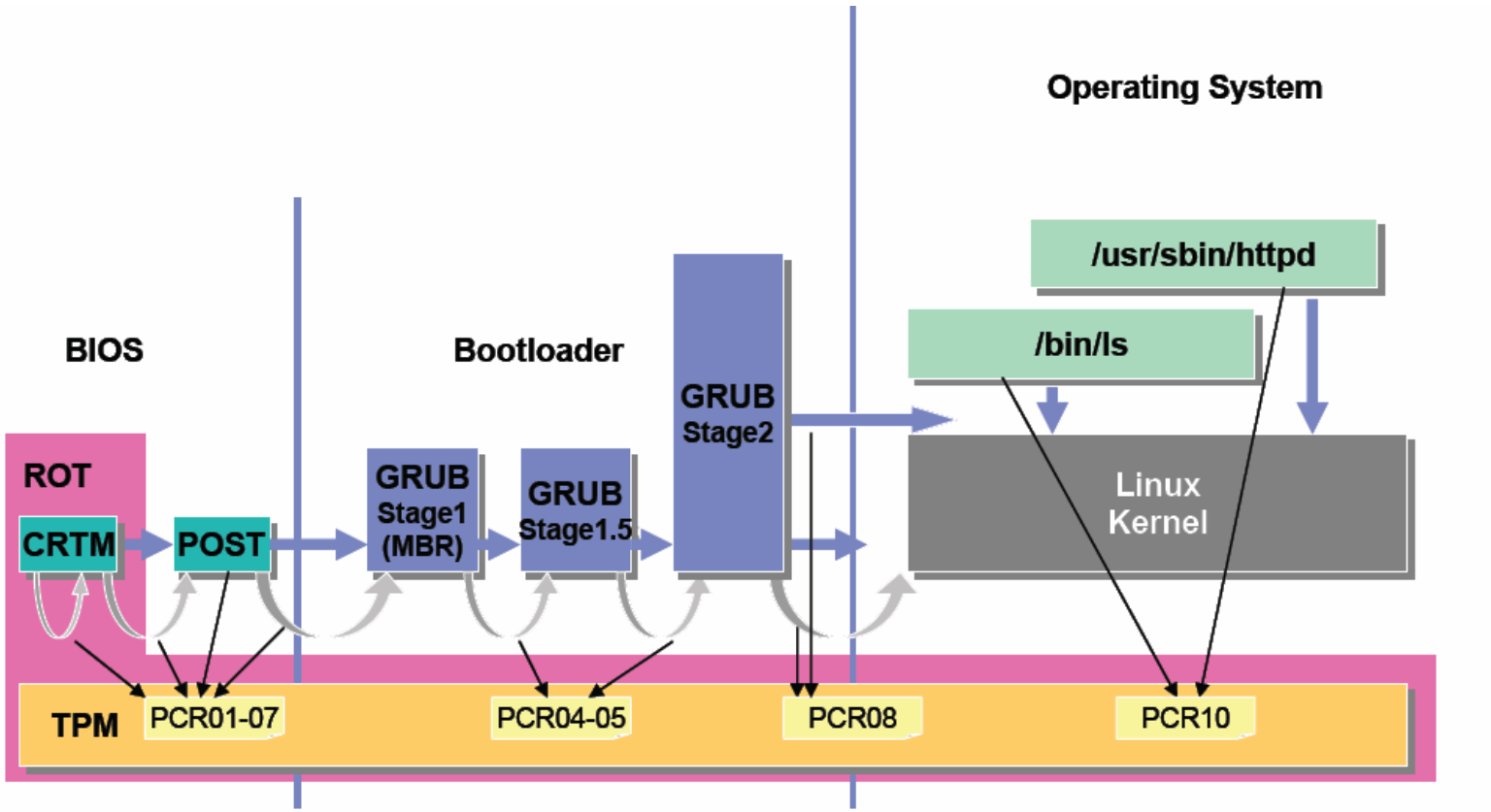
- Updating PCR #n :
  - Incorporates a new integrity measurement into a PCR.
  - TPM_Extend(n,D): PCR[n] ← SHA-1 (PCR[n] || D)
  - TPM_PcrRead(n): returns value(PCR(n))
  - TPM_SHA1CompleteExtend: Completes a SHA-1 digest process on the component to be measured and then incorporates a new integrity measurement into a PCR.
- 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)
- Values recorded to PCRs cannot be removed or deleted until a reset.
- Details of the measuring process, namely the measured value, are recorded to the Stored Measurement Log (SML) outside the TPM.

PCR designation (PC Client spec)

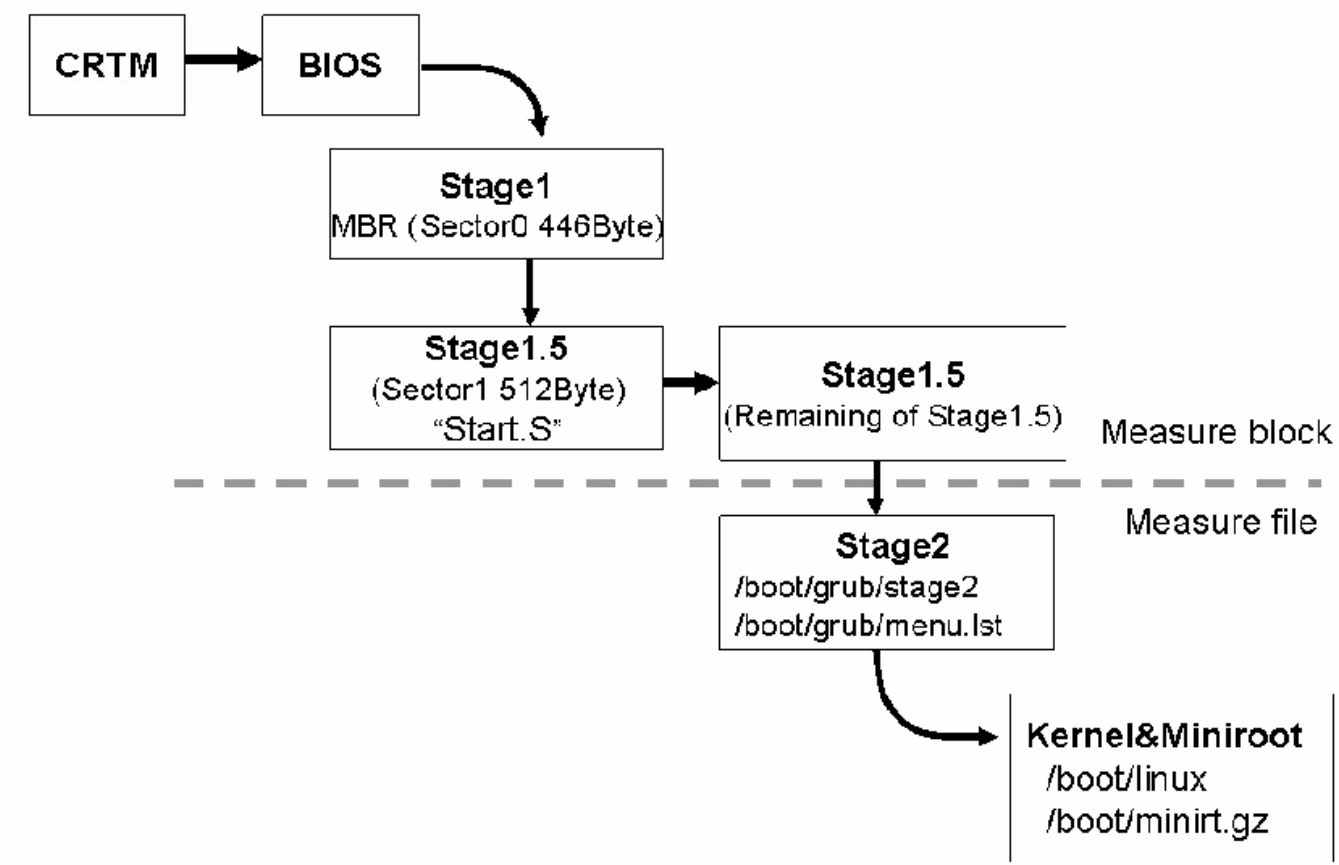
PCR	Function
0	CRTM, BIOS, and Platform Extensions
1	Platform Configuration
2	Option ROM Code
3	Optional ROM Configurations and Data
4	IPL Code (Usually the MBR)
5	IPL Code Configuration and DATA (for use by the IPL code)
6	State Transition and Wake Events
7	Reserved for future usage. Don't use.
8-15	Flexible use

Example – authenticated boot of Linux with Trusted GRUB

- (sourceforge, initially IBM) authenticated boot extension of the GRUB bootloader



Trusted GRUB



Example: Extend to runtime meaurements in IMA

- IBM's Integrity Measurement architecture
  - IMA has Linux measure code loaded and static data files (e.g., confi-gurations) used, such that a remote party can verify that a Linux system contains no low integrity components
- Developed 2005 submitted to lkml but rejected, still available on sourceforge
- Extend TPM-based attestation into the system runtime
  - Attest the Software Stack
- IMA-Guarantees
  - Non-intrusive (not changing system behavior)
  - Load-guarantees for code loaded into the system run-time
  - Detects systems cheating with the measurement list
- Goals
  - Negligible overhead on attested system
  - Usability
- Stored measurement log (SML) is held in kernel space

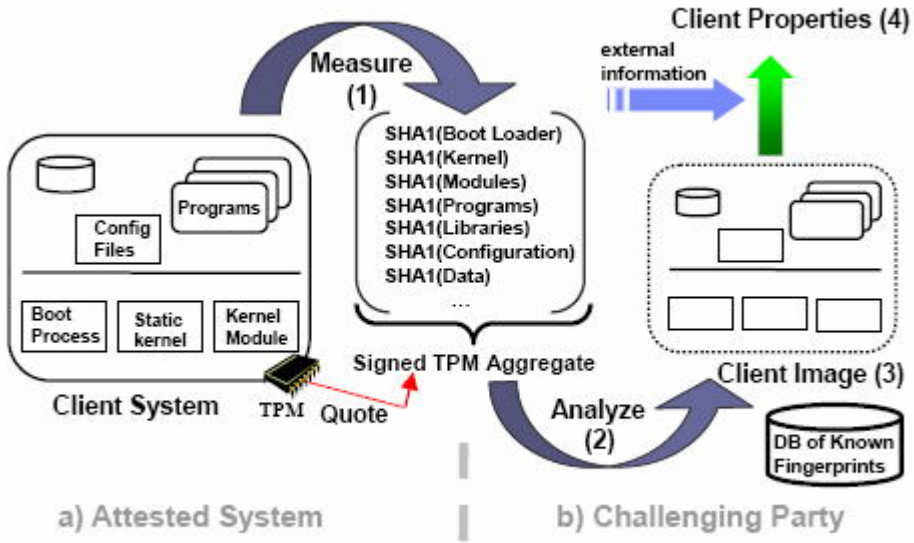


Figure 1: Attestation Architecture Overview

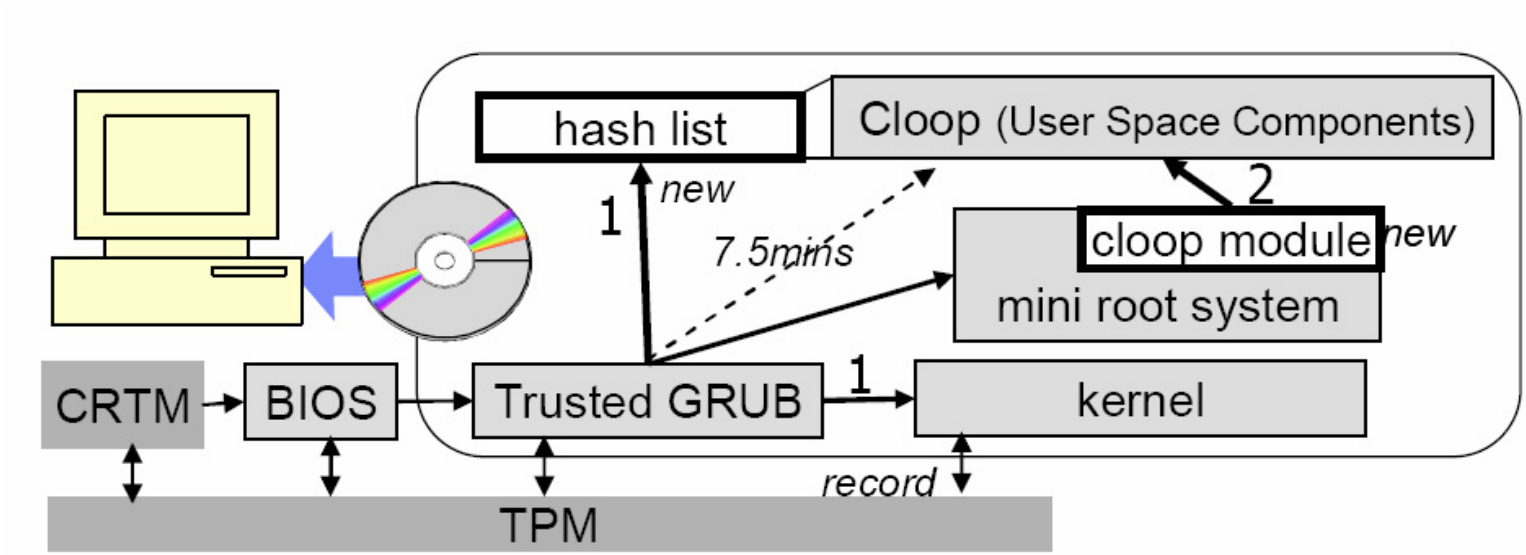
IMA performance

- **Attested System:**
  - Implementation: ~ 5000 lines of code (LSM kernel module)
  - About 400-600 measurements for Fedora C2, Apache, Jakarta Tomcat, etc.
  - Measurement Overhead
- **Attestation service:**
  - Known Fingerprint DB ~ 20 000 Fingerprints (RedHat 9.0, Fedora, ES3)
  - Attestation: 1-2 second “latency” (unoptimized demonstration)

	Kernel	Application	Likelihood
Clean Hit	~ 0.1 µs	~ 5 µs	>> 99 %
New (TPM) Measurement	~ 5 ms + SHA1 (~80MB/s)	~ 5ms + SHA1 (~80MB/s)	<< 1 %

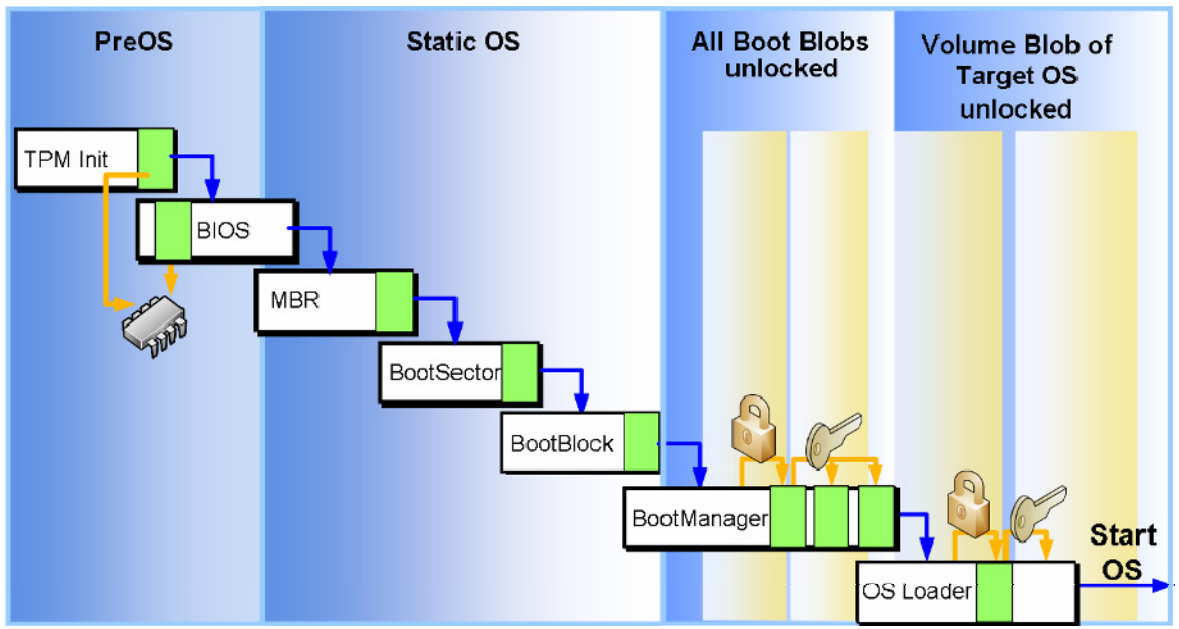
Thin clients started from Live Linux CD

- Trusted CD-boot for Knoppix Linux
- uses a compressed loop-back block device (CLOOP) to read the file system on the CD-ROM
- Uses optimistic strategy on file load
  - calculate the hash value of each block beforehand, and record them in a list in a file.
  - The kernel can verify the integrity of each block using this list file.
  - list file must be protected from unauthorized modification.
  - Trusted GRUB checks the list file and records the hash value in a PCR
- Significant performance increase to plain trusted GRUB



Secure Startup support envisaged to work with Longhorn [6.]

- Builds on firmware extensions (EFI)
  - Secure Startup code runs in the pre-OS environment that is controlled by firmware
  - Secure Startup code must be able to use firmware to access the TPM



BIOS Requirements & EFI

- Requirements for BIOS usage of TPM 1.2 PCR
  - The BIOS MUST measure into PCR each IPL that is attempted and executed; if IPL code returns control back to BIOS then each initial program load MUST subsequently be measured
  - The BIOS MUST NOT measure portions of the IPL pertaining to the specific configuration of the platform into PCR
    - For example, the disk geometry data in the MBR would not be measured into PCR
    - To measure the content of an MBR style disk, the BIOS would measure 0000-01B7h into PCR and 01B8-01FFh into PCR
- Security-enhanced firmware MAY be conventional BIOS, EFI, or a combination of BIOS and EFI
- TCG currently drafting two industry-standard EFI specs
  - EFI Protocol Spec common to PC Clients and Servers
  - EFI Implementation Spec for PC Clients
    - Includes mapping of TPM PCR event measurements to EFI boot components
  - Planned support for EFI support in Longhorn OS loader

Platform builder requirements for secure startup

- After system builder has:
  - Chosen a TPM 1.2 vendor
  - Committed a BIOS team to working on the extensions
- What else is needed?
  - Build a TCG-defined “Host Platform” which includes
    - Motherboard
    - Host processor(s)
    - TPM
    - Immutable part of firmware called the Static Core Root of Trust for Measurement (S-CRTM)
    - Other devices that connect directly to the CPU and interact directly with the CPU
- The platform MUST perform a “Host Platform Reset” which may be:
  - Cold Boot Host Platform Reset,
  - Hardware Host Platform Reset, or
  - Warm Boot Host Platform Reset
- Boot Strap Host processor MUST be reset & begin execution with the S-CRTM
  - All remaining Host Processors MUST be reset
- The TPM MUST be reset
  - Execution of TPM_Init signal
  - TPM MUST NOT be reset without a Host Platform Reset

BIOS to CRTM relationship

- Two CRTM options for PC Architecture
  - Boot Block as CRTM
    - Immutable (fixed) code per TCG Specification

    or…

    - Prove secure update process in “conformance” security target
  - Entire BIOS as CRTM
    - Prove secure update process in “conformance” security target
    - Challenge for most flash mechanisms in the runtime state!
- S-CRTM TPM interface code adds 3KB to 6KB to boot block
- F000 segment size limitation requires creative mapping of BIOS core
- BIOS Setup must include TPM functions including enable/disable and factory reset (ForceClear)
- RTM TPM interface code is now 32-bit
  - Mechanism required to transition from natural BIOS state to 32-bit mode

Architecture options in the PC environment

- Where to place the CRTM?
  - S-CRTM as BIOS Boot block + EFI to have access to TPM function (envisaged for Longhorn)
  - CPU (Itanium architecture)
- Needs access to TPM functionality
- Needs to support
  - Recovery
  - Updates
- Authorisation and physical presence

Authenticated boot caveats

- Physical presence
  - Conduit to BIOS for command sequences requiring physical presence
    - S-CRTM must detect user presence (i.e. button press, etc.), otherwise physical presence is locked
      - e.g. BIOS must distinguish a SW initiated warm/cold boot from a physical pressing of the power button
- In any case
  - CRTM is orthogonal to changes in BIOS/firmware/microcode between boot periods
  - CRTM needs to have update mechanisms for the latter
- Load-time verification is resource consuming
  - Optimistic strategies can help