Trusted Platform Module

Integrity Measurement, Reporting, and Evaluation

Motivation

- Reliance on remote clients/servers
  - Financial records and e-commerce
  - Electronic medical records
  - Cloud computing
- Threats to clients from remote servers
  - Malicious servers masquerade as legitimate ones
  - Legitimate servers subject to attack
    - Malware
    - Viruses
    - Rootkits
- Threats to servers from corrupted remote clients
  - Penetrating firewalls
  - Release of confidential data

Motivation

- Need: mechanisms to verify the integrity of remote clients/servers
  - Correct patches installed
  - Advertised/expected services exist
  - System not compromised
- Solution
  - Provision of critical services by a trusted platform module (TPM) on the local host
  - Capability of host to measure integrity of host software
  - Protocol to communicate the integrity measurements from the host to a remote party
  - Means for remote party to assess the integrity measurements and determine level of trust in the host

Trusted Platform Module (TPM)

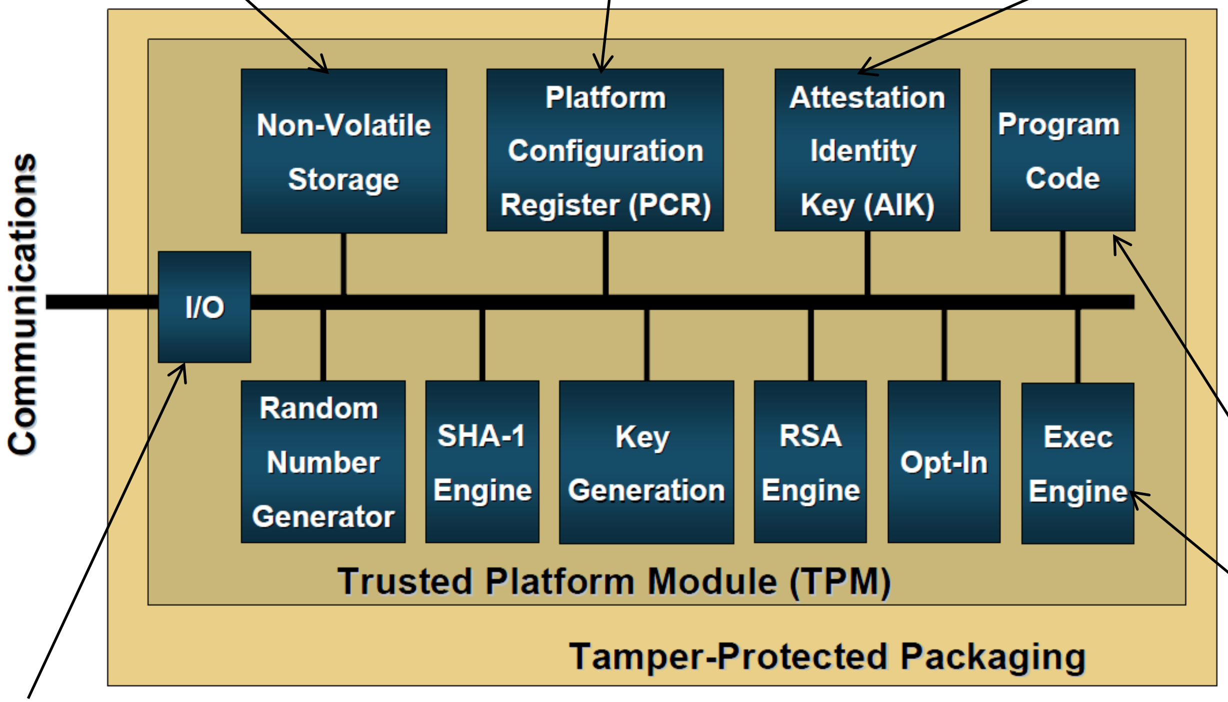
- Standard defined by the Trusted Computing Group
- Availability
  - Hardware chip currently in 100M laptops
  - HP, Dell, Sony, Lenovo, Toshiba,…
  - HP alone ships 1M TPM-enabled laptops each month
- Core functionality
  - Secure storage
  - Platform integrity reporting
  - Platform authentication

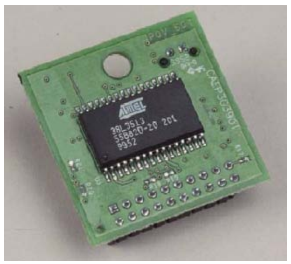
TPM Architecture

keys, owner
authorization data

integrity measures

signing keys
when in use

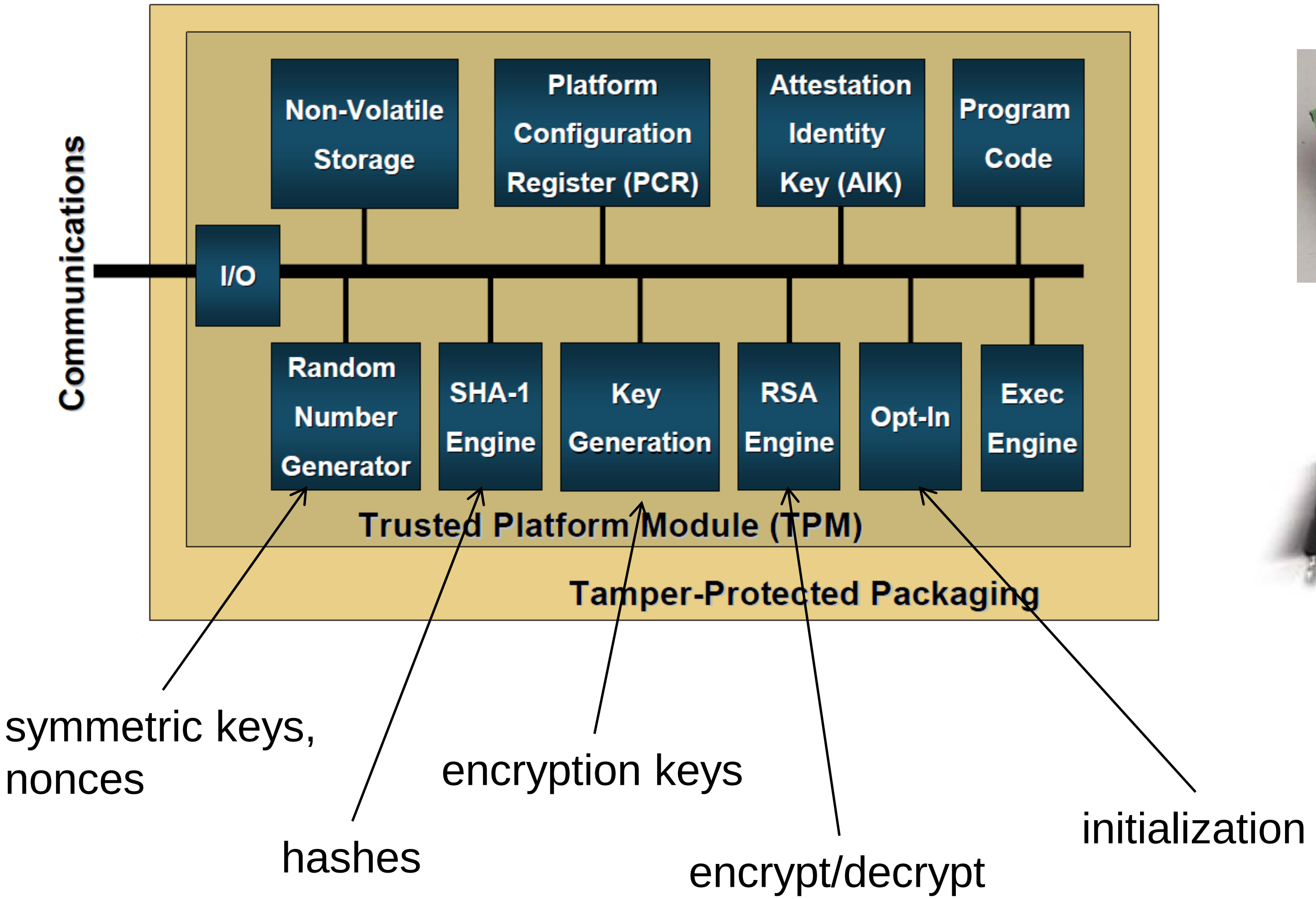


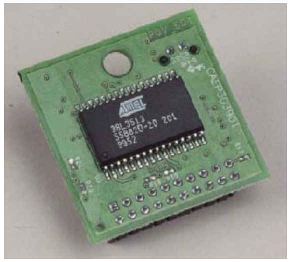


external
interaction

TPM control

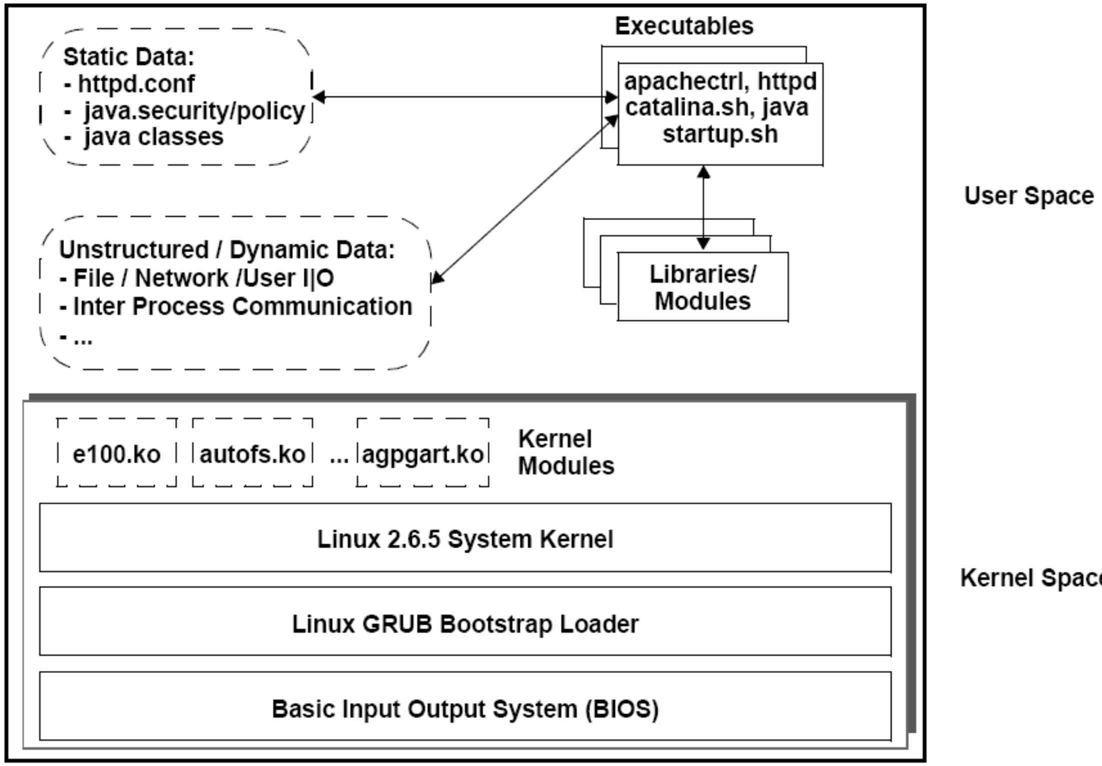
TPM Architecture





Execution Environment

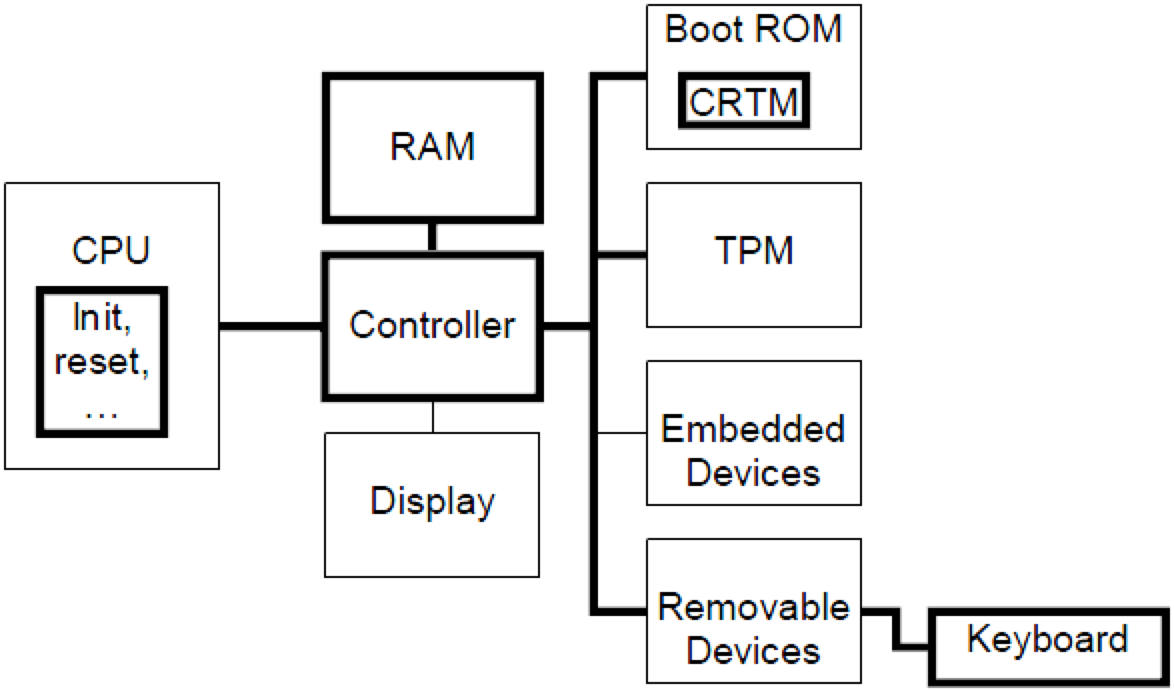
- Executable content
  - Types
    - programs
    - libraries
    - scripts
  - Loaded by
    - kernel
    - application
- Structured data
  - class files
  - configuration files
- Unstructured data
  - databases



Pragmatics

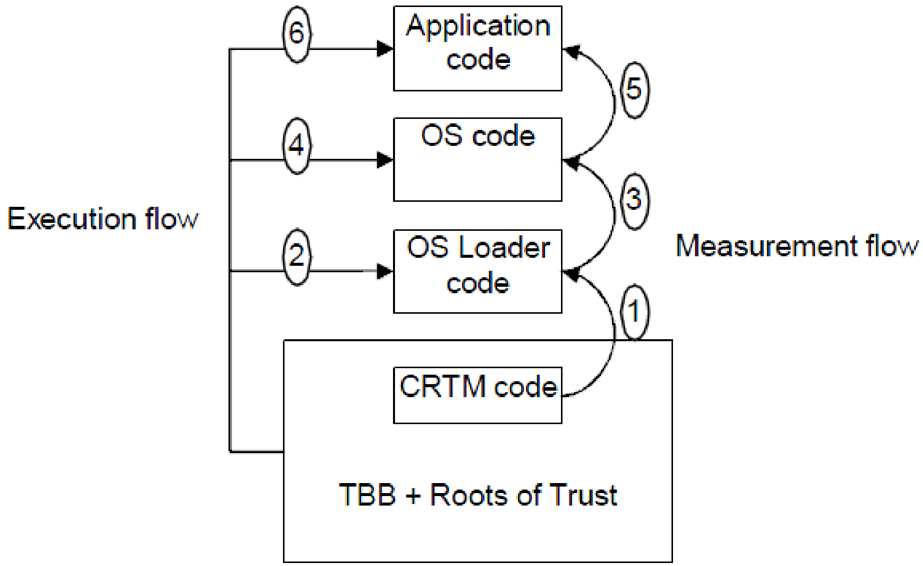
- Feasibility
  - Manageable number of components to measure for typical systems
    - 500 for a workstation configured for general technical work (document authoring, programming, browsing, etc.)
    - 250 for a typical web server
- Approach
  - Extensible architecture
  - Provides essential measurement structures
  - Allows future additions

Trusted Building Blocks



- TBB do no have shielded locations or protected capabilities (as does TPM)
- CRTM: core root of trust for measurement
- Keyboard: showing physical presence when needed

Integrity Measurement



- Measure a component before executing it
- Record the measurement as a hash value of the code/data (aka, *fingerprint*)
- Produces a hash chain by combining individual hash values
- Changes in the executing code can be detected by comparing measurement of executing code against recorded value
- The measurements themselves must be protected from *undetected* manipulation

Detecting Malware Attacks

```
#000: D6DC07881A7EFD58EB8E9184CCA723AF4212D3DB boot_aggregate
#001: CD554B285123353BDA1794D9ABA48D69B2F74D73 linuxrc
#002: 9F860256709F1CD35037563DCDF798054F878705 nash
#003: 84ABD2960414CA4A448E0D2C9364B4E1725BDA4F init
#004: 194D956F288B36FB46E46A124E59D466DE7C73B6 ld-2.3.2.so
#005: 7DF33561E2A467A87CDD4BB8F68880517D3CAECB libc-2.3.2.so
...

#110: F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 syslogd

...
```

initial

---

attack

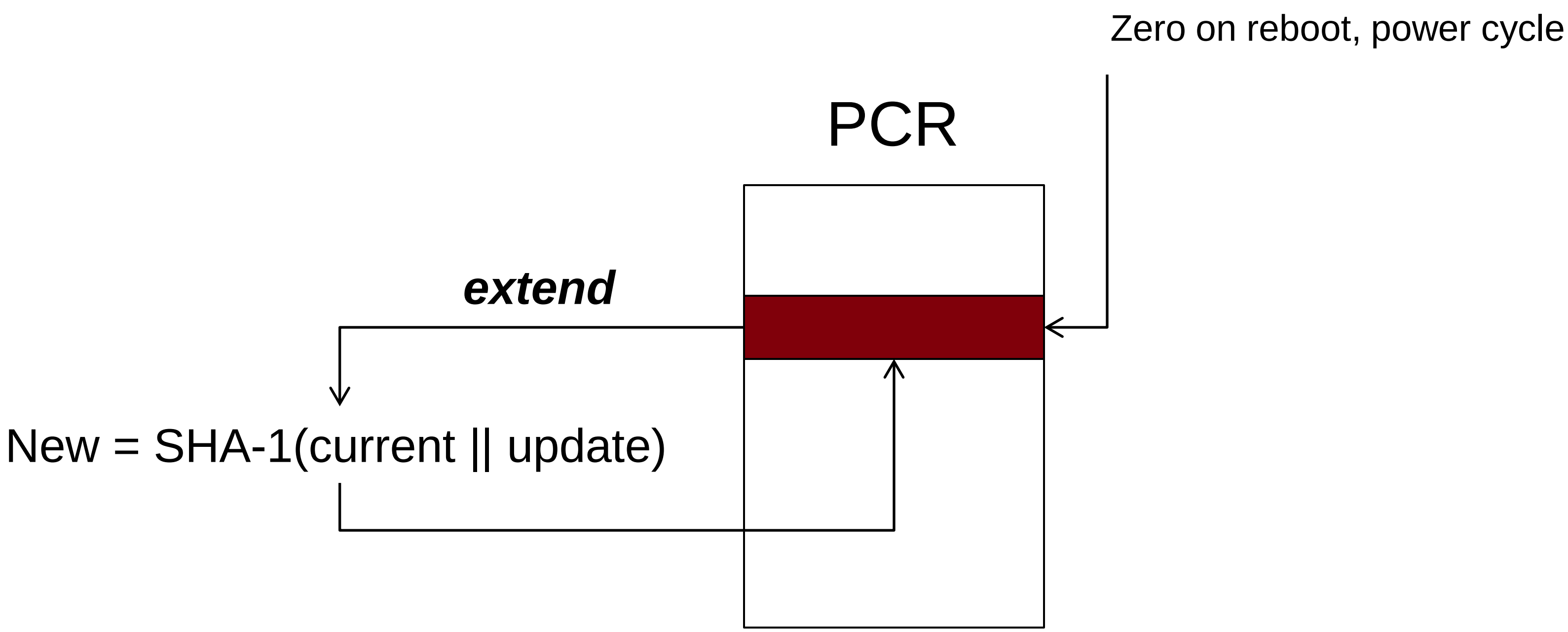
```
...
#110: F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 syslogd
...

#525: 4CA3918834E48694187F5A4DAB4EECD540AA8EA2 syslogd
```

Measurement before rootkit attack

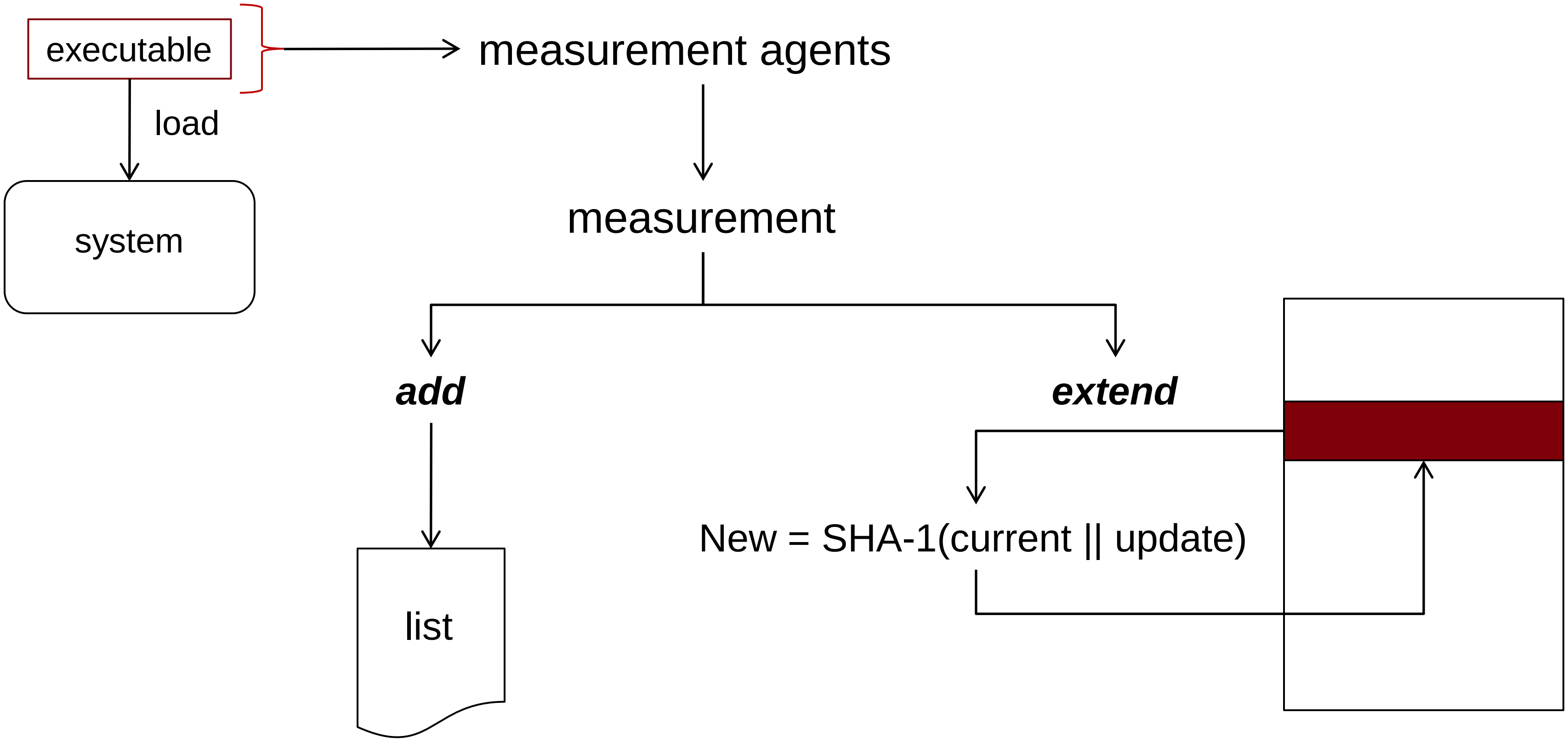
Measurement after rootkit attack

Platform Configuration Registers



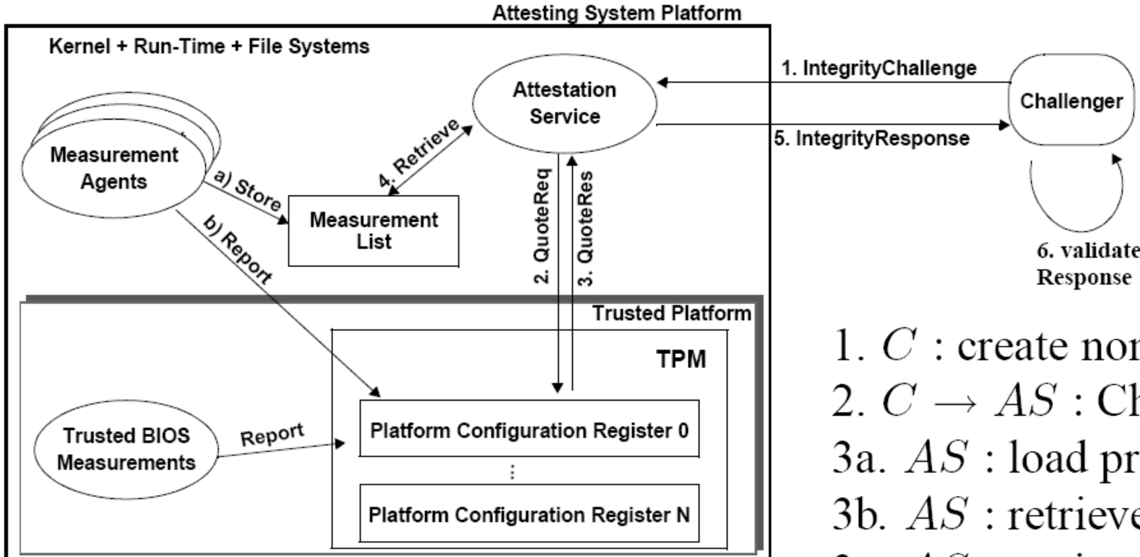
- At least 16 PCR registers, each register stores 20 bytes

Maintaining a Measurement List



- PCR contains the linked hash of all measurements in the list
- Alterations to the list values can be detected

Reporting a Measurement List



1. C : create non-predictable 160bit $nonce$
2. $C \rightarrow AS$: ChReq($nonce$)
3a. AS : load protected AIK_{priv} into TPM
3b. AS : retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c. AS : retrieve Measurement List ML
4. $AS \rightarrow C$: ChRes($Quote$, ML)
5a. C : determine trusted $cert(AIK_{pub})$
5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c. C : validate $nonce$ and ML using PCR

Questions

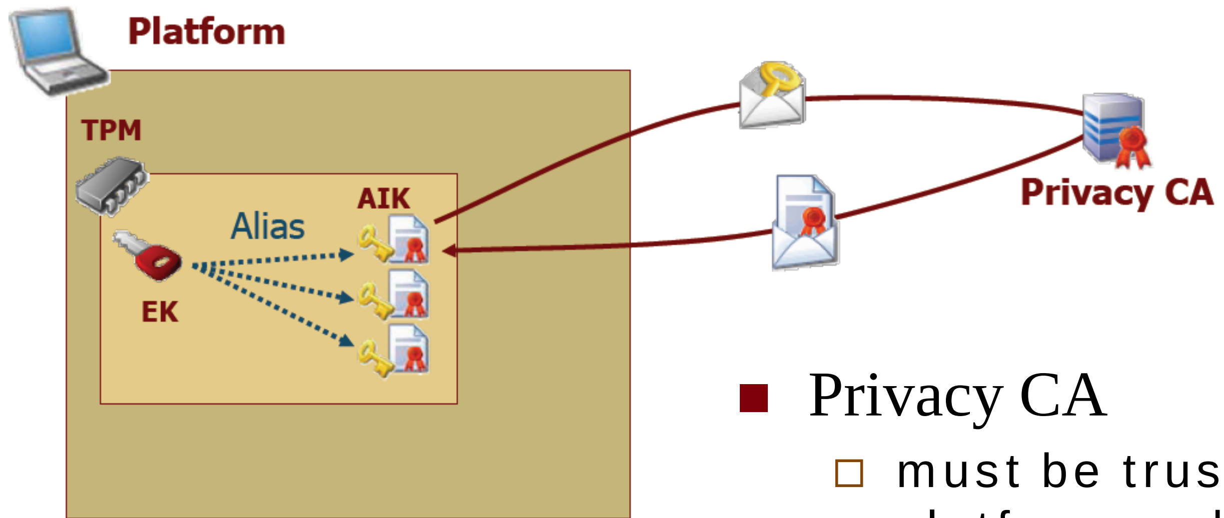
- How is the AIK generated?
- Where is it stored?
- How does the challenger validate the measurement list (*ML*)?

C: challenger
AS: attesting system
AIK: attestation identity key

Long-term Keys

- The TPM has two long-term key pairs stored in non-volatile memory on the TPM
  - Endorsement Key (EK)
  - Storage Root Key (SRK)
- Endorsement Key
  - Private key never leaves the TPM
  - Limited use to minimize vulnerability
  - Identifies individual platform: potential privacy risk
  - Public part contained in endorsement credential
  - EK and endorsement credential loaded by manufacturer
- Storage Root Key
  - Basis for a key hierarchy that manages secure storage
  - More on this later…

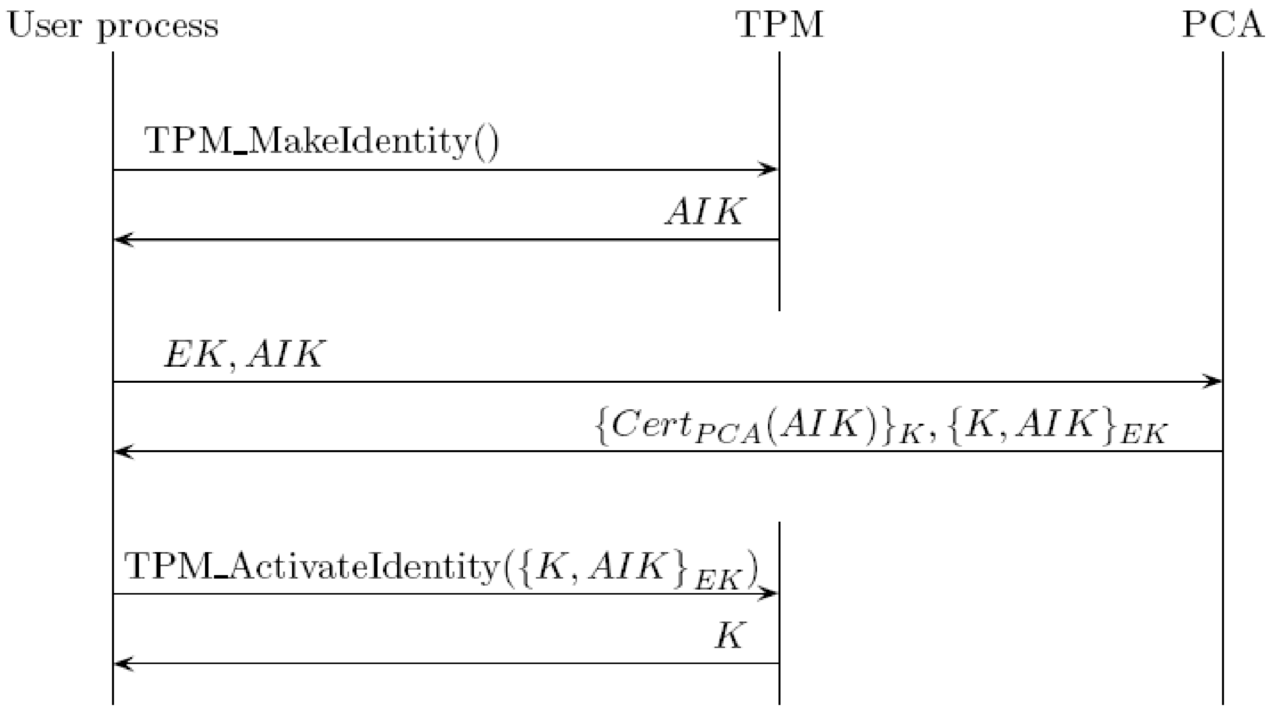
Attestation Identity Keys (AIKs)



- Privacy CA
  - must be trusted by platform and challenger

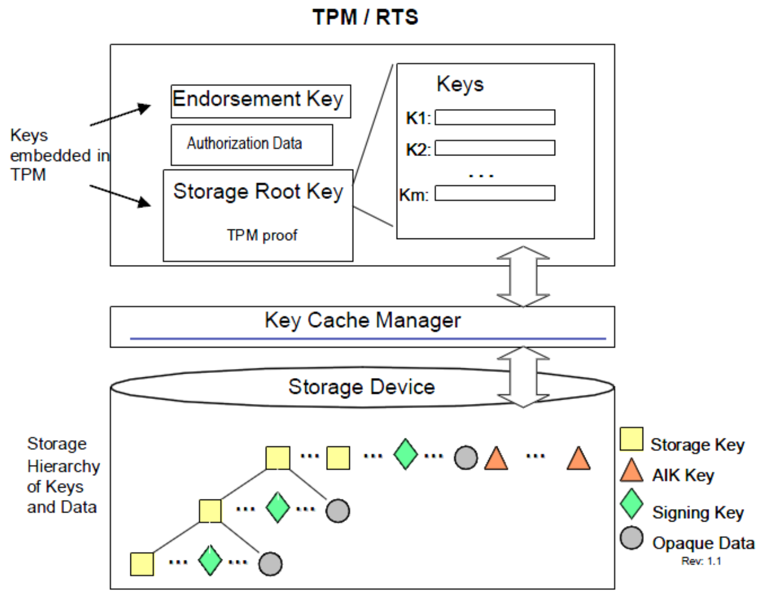
- AIK
  - serves as alias for EK
  - platform may have many AIKs to allow a number of unlinkable interactions
  - held in secure storage (see later)
  - guarantees that platform has a valid TPM (but does not identify platform)

Creating AIKs



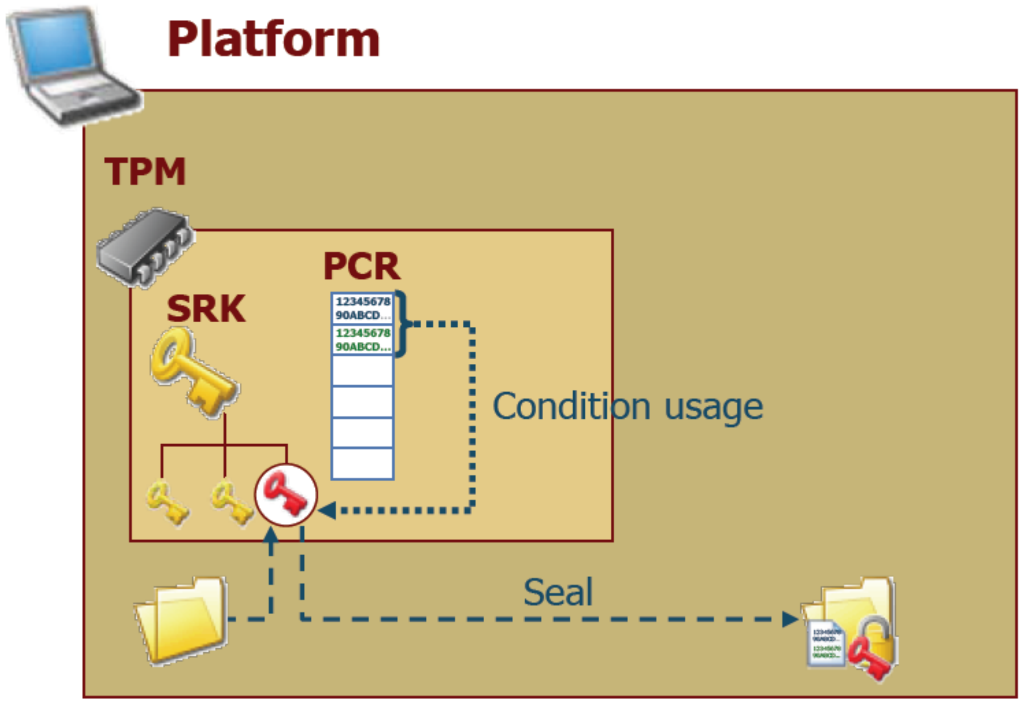
- AIK cryptographically bound to TPM with specific EK

Secure Key Storage



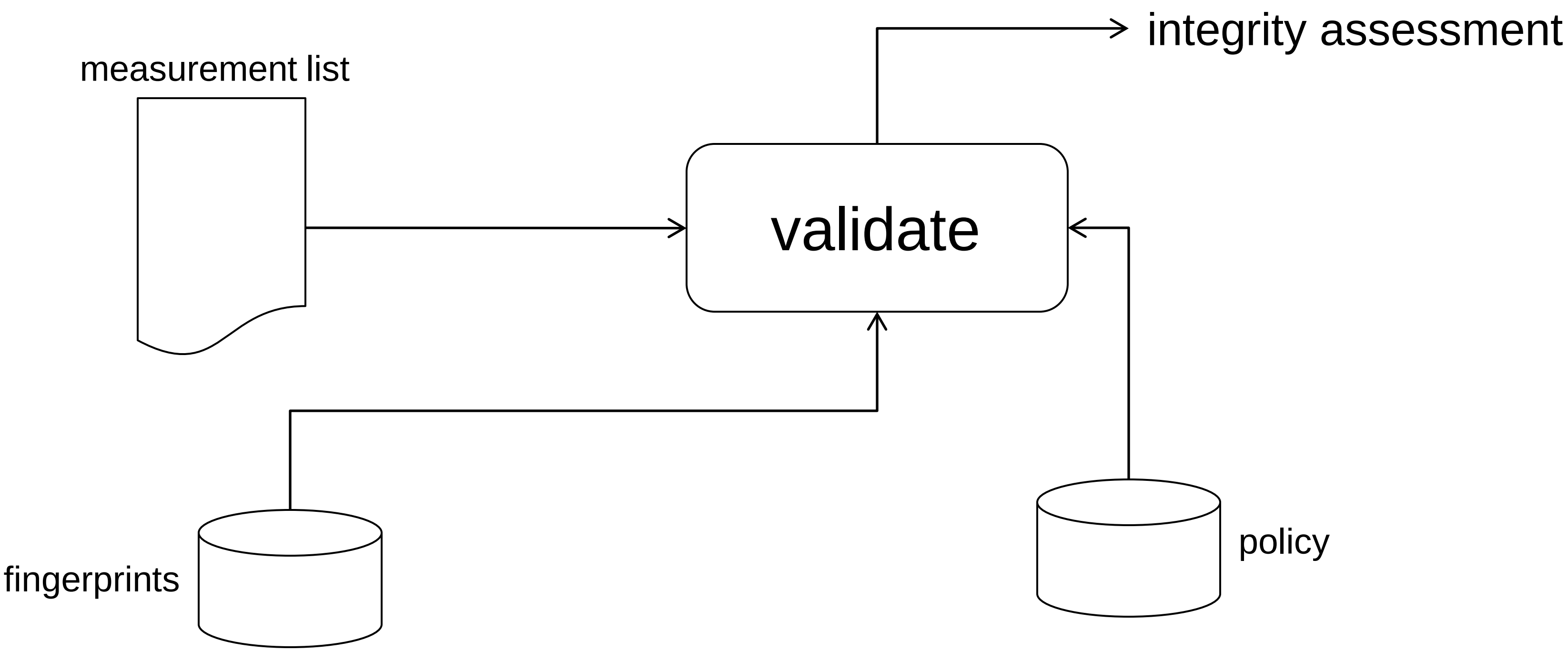
- The TPM uses/manages many keys, but has limited storage
- Keys (except for the EK and SRK) may be placed in secure storage
- Secure storage may be on flash drive, file server, etc.
- Authdata (password) is associated with each key
- Key and authdata encrypted with storage key (creating a blob)
- Two forms: bind (normal encryption) and seal (bound to PCR state)

Sealed Storage



- Goal: ensure that information is accessible only when the system is in a known/acceptable state
- System state determined by PCR value

Assessing Integrity



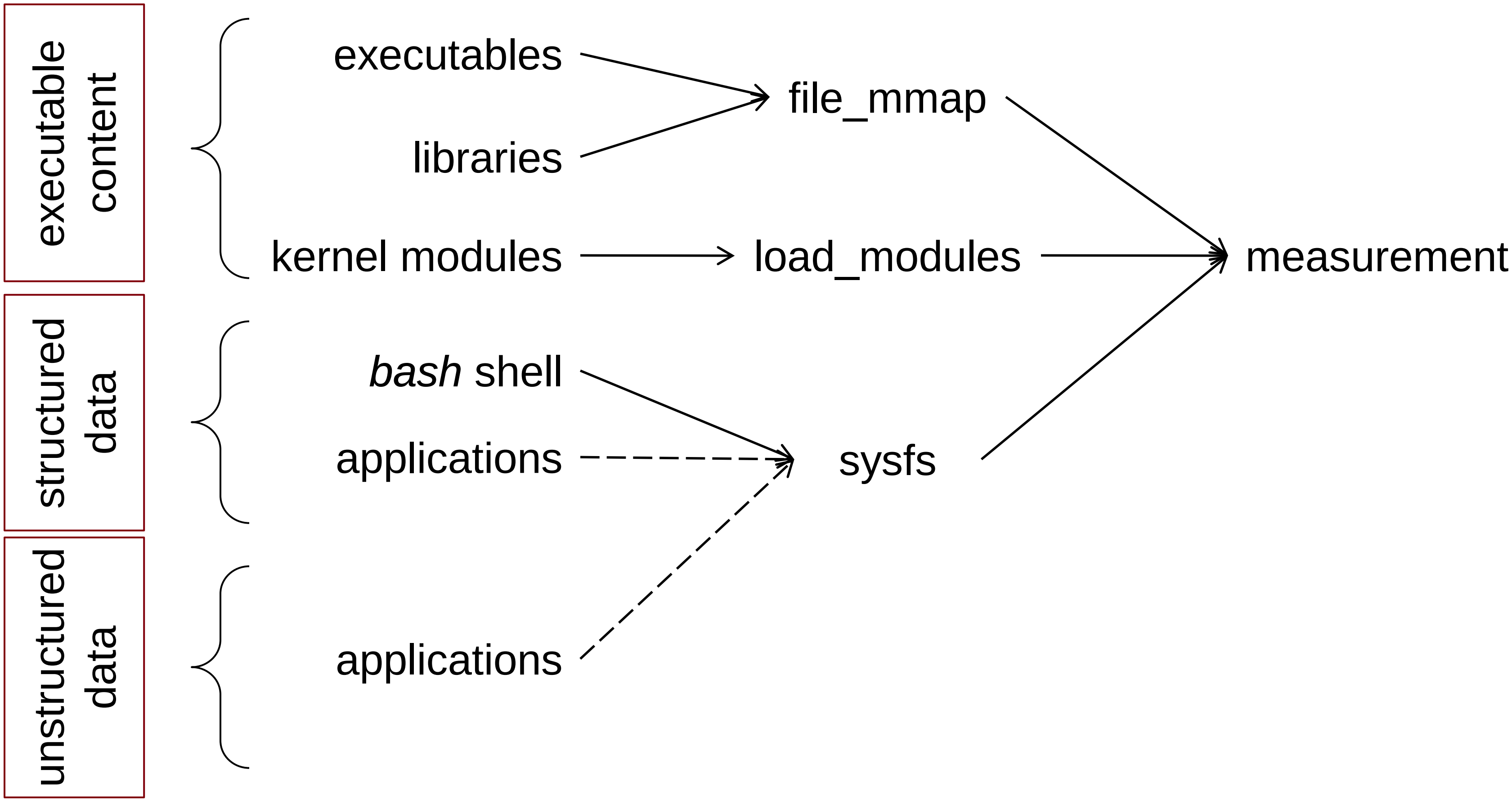
- acceptable
- malicious
- vulnerable-remote
- vulnerable-local
- unknown/uncontrolled

$$client \in Distrusted \leftarrow \exists e \in E(client) : \neg(e \in Known) \vee (e \in (Malicious \cup Uncontrolled \cup Remote)) \quad (1)$$

$$client \in IntHigh \leftarrow \forall e \in E(client) : (e \in Acceptable) \quad (2)$$

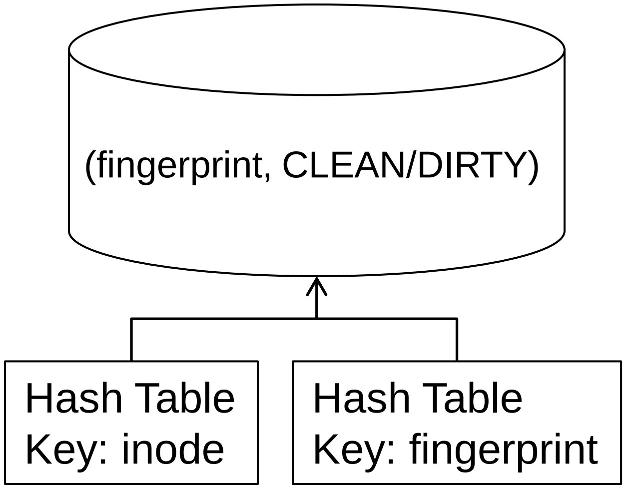
$$client \in IntMedium \leftarrow \neg(client \in IntHigh) \wedge \forall e \in E(client) : e \in (Acceptable \cup Local) \quad (3)$$

Adding Measurement Instrumentation



Measuring New Files

```
if (found via inode HT) {
   if (CLEAN) exit;
   if (DIRTY) {
      compute fingerprint;
      if (same as stored) {
         set CLEAR;
         exit;
      }
      else {
         search fingerprint HT;
         if (found) {
            exit;
         }
         else {
           UPDATE();
         }
      }
   }
}
if(not found) {
   UPDATE();
}
```




```
UPDATE() {
   add to database;
   update HTs;
   extend PCR;
}
```

Performance

mmap type	mmap latency (stdev)	file_mmap LSM
no_SHA1	1.73 μs (0.0)	0.08 μs
SHA1	4.21 μs (0.0)	2.56 μs
SHA1+extend	5430 μs (1.3)	5430 μs
reference	1.65 μs (0.0)	n/a

Measurements via sysfs		Overhead (stdev)
measure	no_SHA1	4.32 μs (0.0)
	SHA1	7.50 μs (0.0)
	SHA1+extend	5430 μs (1.6)
reference	sys fs open/write/close	4.32 μs (0.0)

- vast majority of cases does not require +extend

Performance

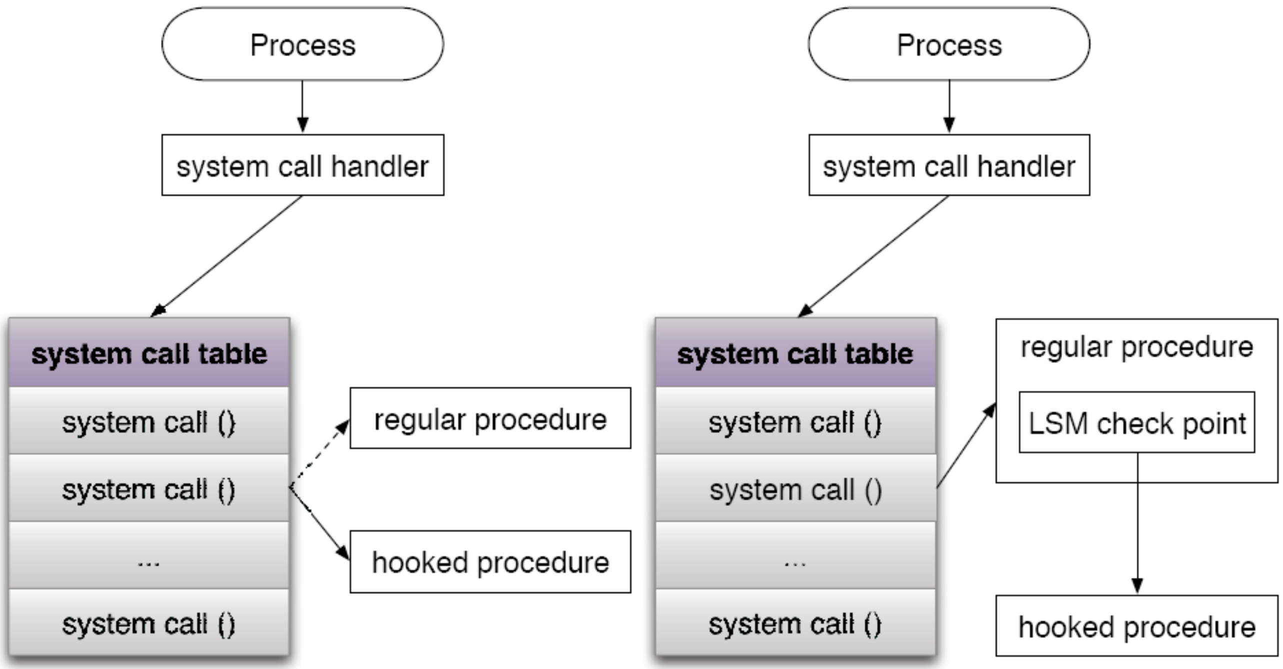
File Size (Bytes)	Overhead (stdev)
2	4.21 μs (0.0)
512	10.3 μs (0.0)
1K	16.3 μs (0.0)
16K	197 μs (0.1)
128K	1550 μs (1.1)
1M	12700 μs (16)

- increase in overhead for computing fingerprint

Secure Monitoring

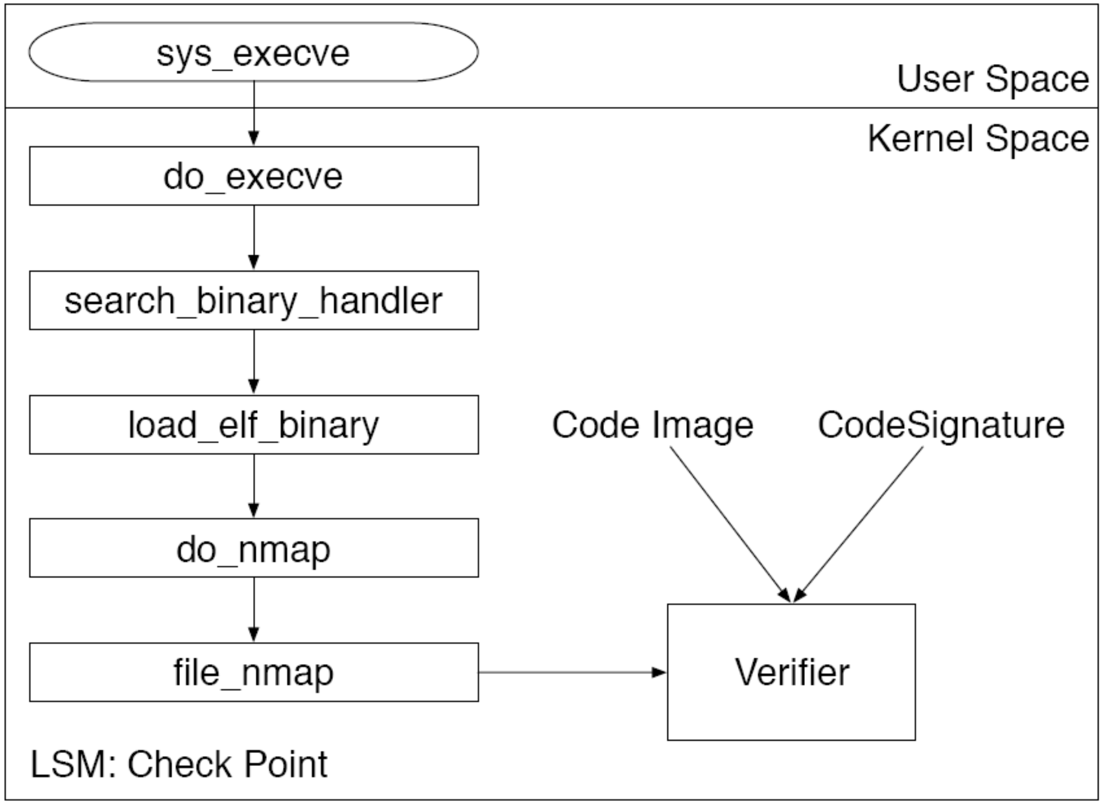
- Monitoring of system activity is important
  - Detect information leakage
  - Warn of intrusions
  - Indicate presence of malware activity
- Approach
  - Security of monitoring module
    - Implemented using LSM hooks
    - Secured by SecVisor
  - Monitoring result guaranteed to be secure
    - LSM-base mandatory access control (MAC)
    - DigSig (application integrity and invocation)

Linux Security Module (LSM)



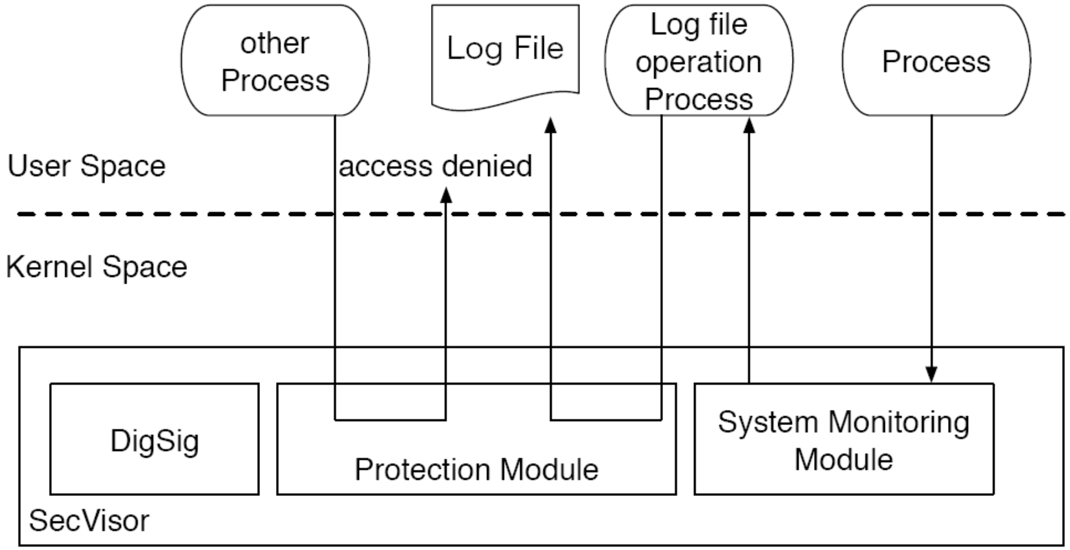
(a) sys_call_table export (b) LSM

DigSig Verifier



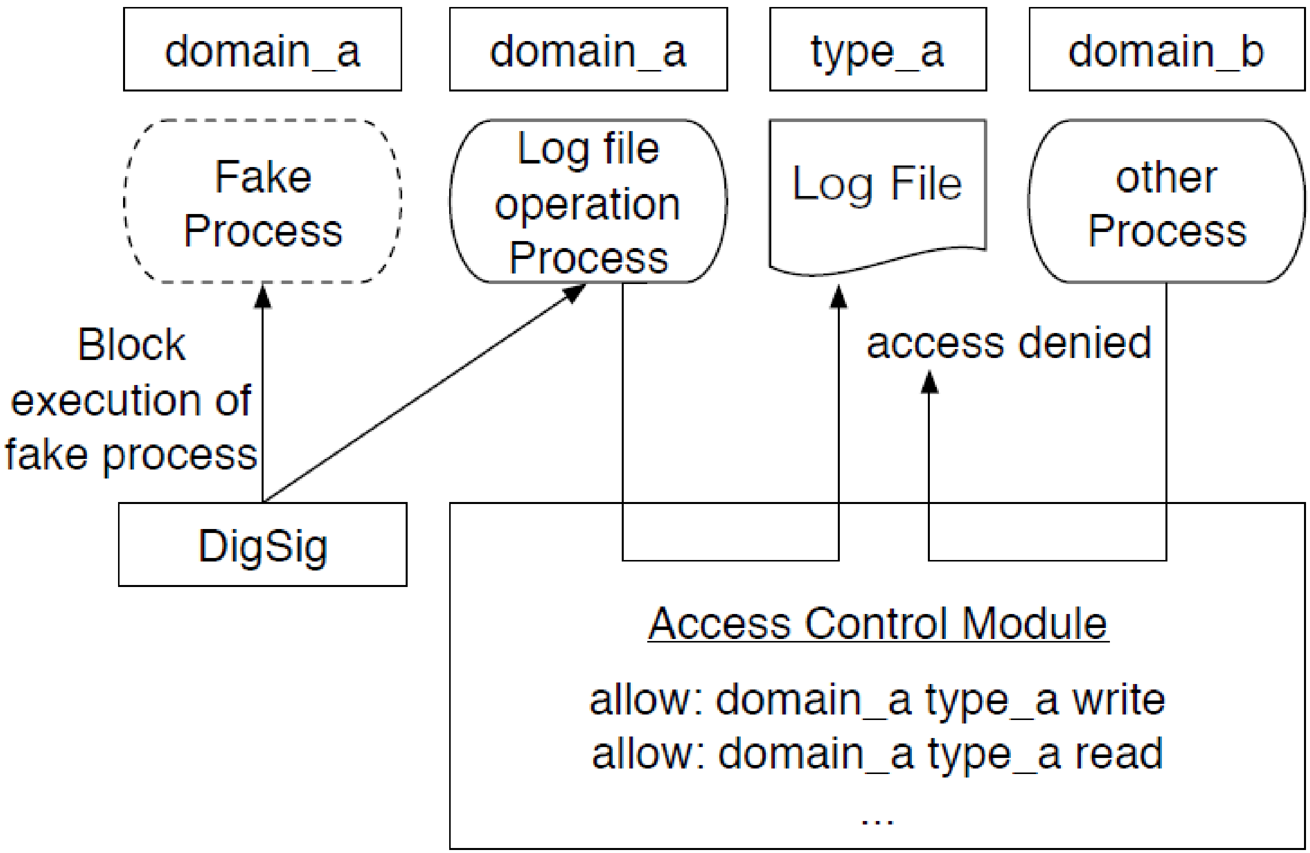
- Verifies that load code conforms to signature
- Ensures that trusted applications are running

SecVisor



- Small hypervisor creating
  - Trusted boot
    - Boots SecVisor and records SecVisor fingerprint in TPM
    - Boots Linux kernel and records kernel fingerprint in TPM
  - Memory protection
    - During boot processes and kernel execution
  - Provides run-time protection of kernel against rootkit attacks

Protection Module



Performance

Evaluation System	Components
(i)	Linux kernel 2.6.20.14
(ii)	Linux kernel 2.6.20.14 + SecVisor
(iii)	Linux kernel 2.6.20.14 + System monitoring mechanism
(iv)	Linux kernel 2.6.20.14 + SecVisor + System monitoring mechanism

System	Null Call	Process		File	
		Fork	Exec	Create	Delete
(i)	0.09	117	353	13.4	12.1
(ii)	4.84	1398	3434	21.5	16.9
(iii)	0.13	584	1267	256.3	691.6
(iv)	4.81	4709	7771	484.3	1280.4