Trusted Computing

Numerous computing devices

Numerous software

How should we trust these devices or software?

- **Trust** (Dictionary.com): firm **reliance** on the integrity, ability, or character of a person or thing.
- **Security**: the state of being free from danger or threat

Trust vs. Security

- **Trust** implies **reliance**
- Ideally, only trust secure systems
- All trust relationships should be explicit

- **Security** is a judgment of effectiveness of a particular mechanism
- Judge based on specified policy
- Security depends on trust relationships

Trusted computing - Definition

- “a system that you are forced to trust because you have no choice” -- US DoD

- “A ‘trusted’ computer does not mean a computer is trustworthy” -- B. Schneier

Trusted computing base

- Trusted Computing Base (TCB)
  - Hardware
  - Firmware
  - Operating System
  - …

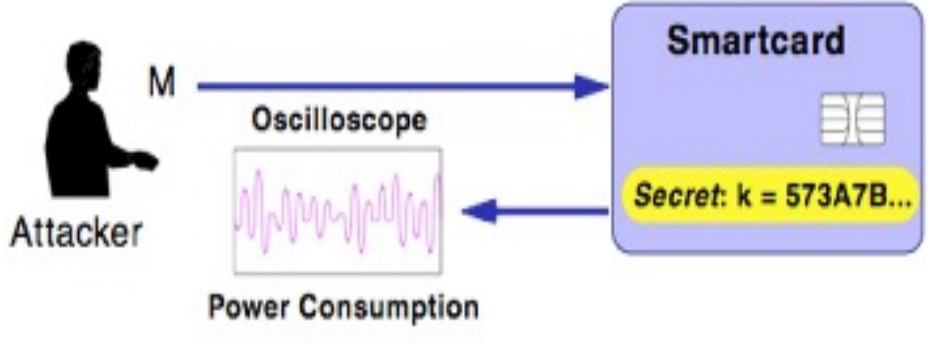
- There is always a level at which we must rely on trust

Trusted computing is difficult

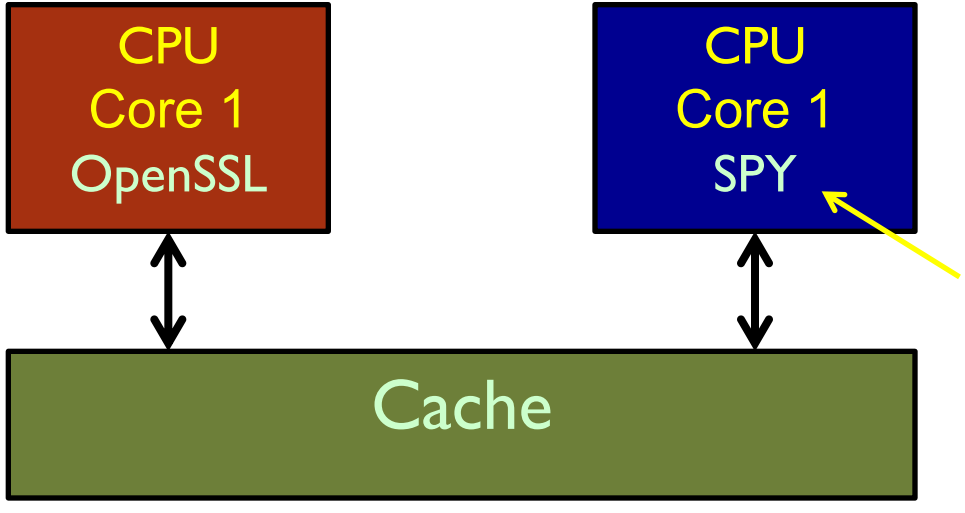
- **Software vulnerabilities**
  - Buffer overflow
  - Incomplete mediation
  - Race condition
- **Hardware exploitation**
  - Invasive probing
  - Non-invasive measurement

Side channel attacks: Unexpected attacks on smartcards

- Khokar et al., June 1998: Measure instantaneous power consumption of a device while it runs a cryptographic algorithm
- Different **power consumption** when operating on logical ones vs. logical zeros



Side channel attacks – multicore CPU



Use side information

- Timing
- Cache misses

To reveal the key

Trusted Systems: Hardware vs. Software

- **Hardware-based** trusted computing
  - Pros: High performance possible
  - Cons: Hard/costly to change

- **Software-based** trusted computing
  - Pros: Easy to change
  - Cons: Difficult to hold private keys

Software-Based Trusted Computing

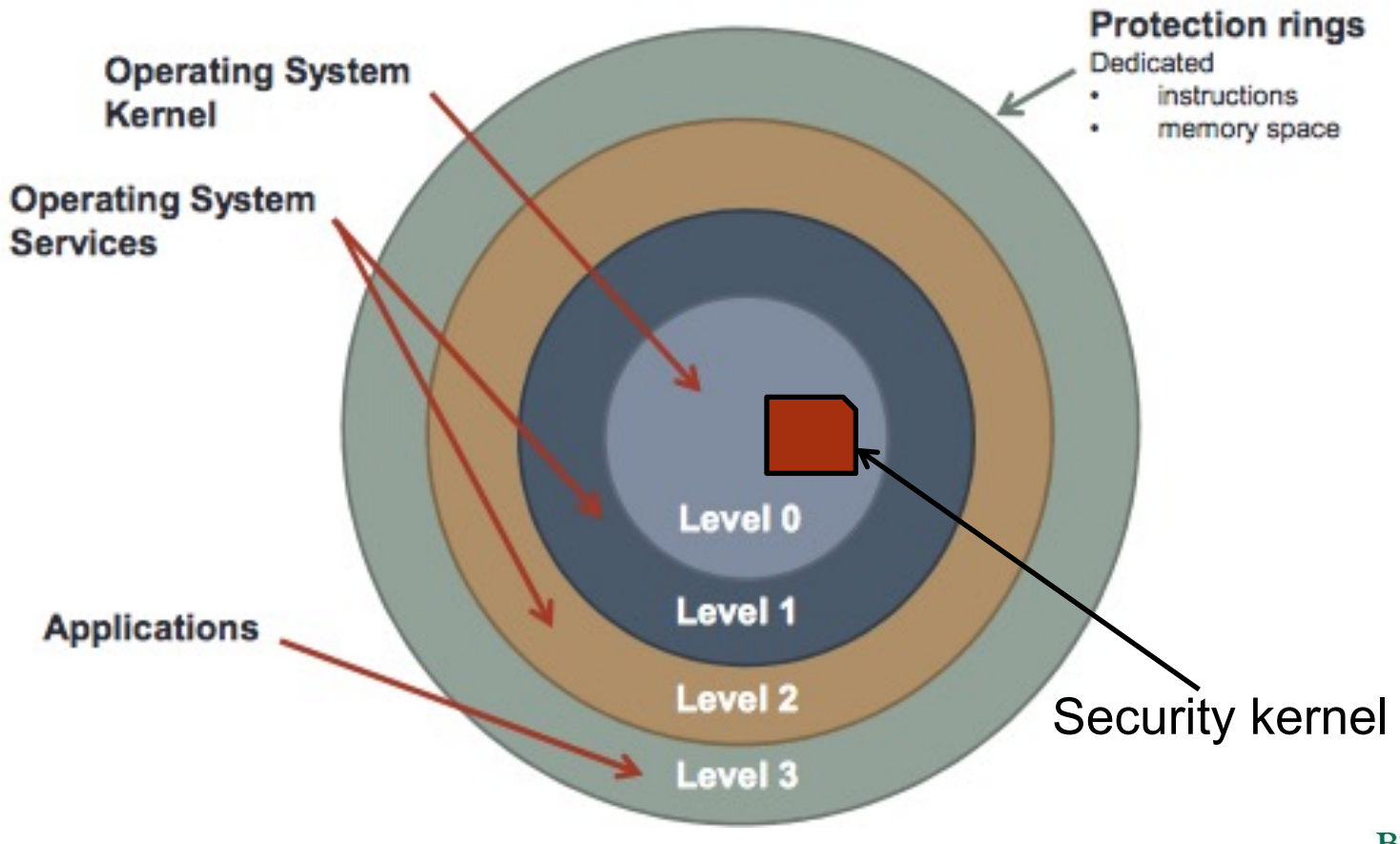
- Goal of software-based trusted computing secure isolation
  - **Trusted OS** (e.g. Trusted Solaris, SE-Linux)
    - SE-Linux
  - **Virtualization**
    - Java virtual machine

SELinux: Trusted OS

Security Kernel in OS

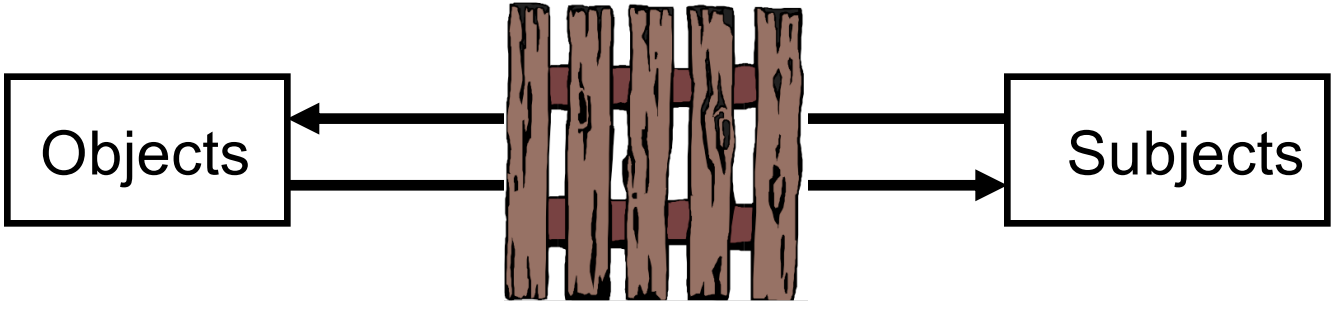
- **Kernel** is the lowest-level part of the OS
- Kernel is responsible for
  - Synchronization
  - Inter-process communication
  - Message passing
  - Interrupt handling
- The **security kernel** is the part of the kernel that deals with security
- Security kernel contained within the kernel

Security ring architecture



Reference Monitor

- The part of the security kernel that deals with access control
  - Mediates **access** of subjects to objects
  - Tamper-resistant(isolated from other components)
  - Analyzable (small, simple, etc.)



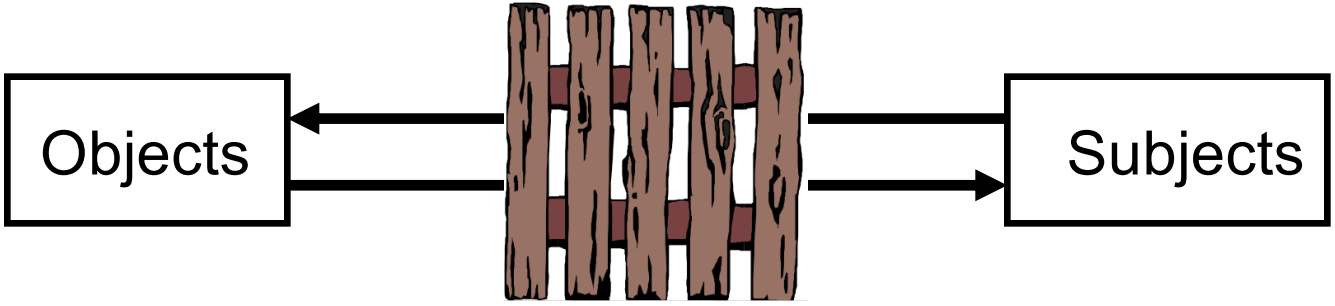
Reference monitor

Mainstream OS (e.g., Windows & Linux)

- Large and complex code base

- Optimized for ease of use, performance and reliability

Access control: who is allowed to access what?



Reference monitor

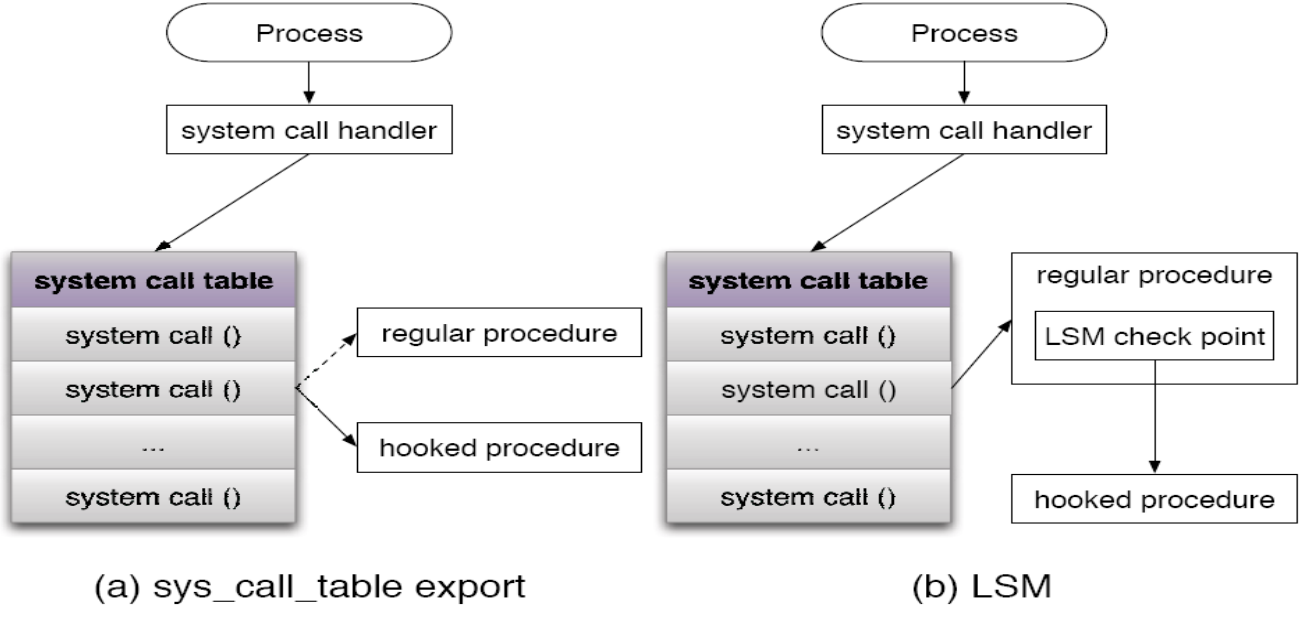
Discretionary vs. Mandatory Access Control

- **Discretionary Access Control (DAC)** in Linux allows **owner** of the data object to control its access permissions.

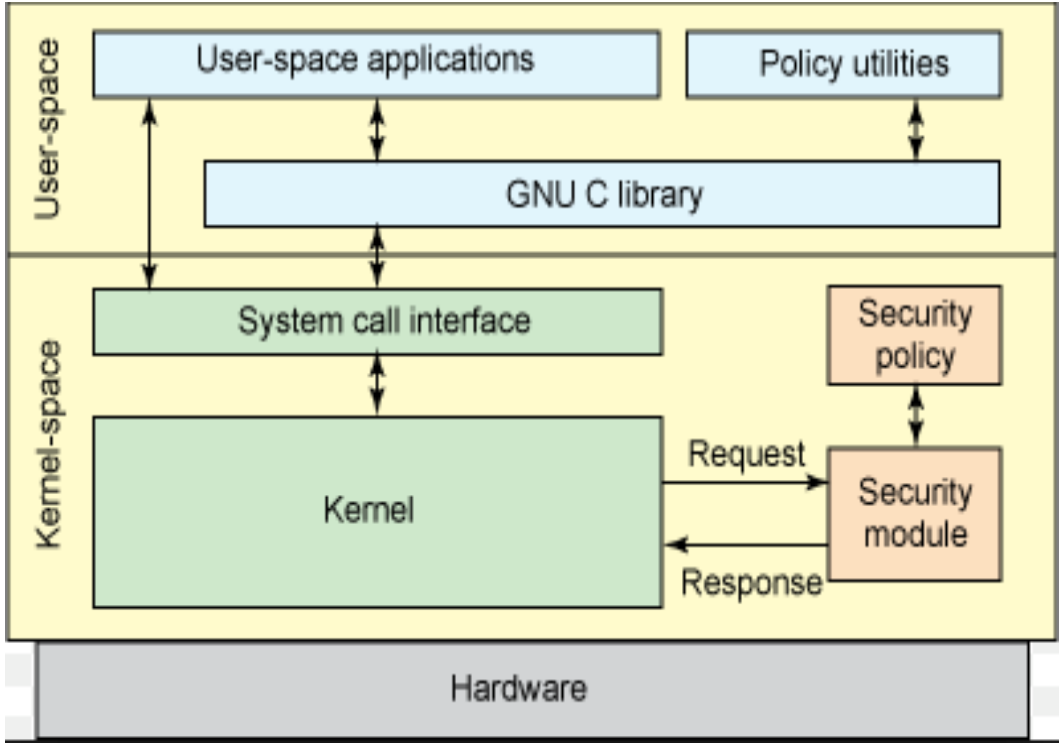
- **Mandatory Access Control (MAC)** allows you to define permissions for how **all processes** (called *subjects*) **interact** with **other parts** of the system such as files, devices, sockets, ports, and other processes (called *objects* in SELinux).

Reference Monitor for Linux

- **LSM** (Linux Security Module) provides a reference monitor interface for Linux (complete mediation)



SELinux Architecture builds on LSM



Initial developer: NSA

"NSA Security-enhanced Linux is a set of patches to the Linux kernel and some utilities to incorporate a strong, flexible mandatory access control (MAC) architecture into the major subsystems of the kernel. It provides an enhanced mechanism to enforce the separation of information based on confidentiality and integrity requirements, which allows threats of tampering and bypassing of application security mechanisms to be addressed and enables the confinement of damage that can be caused by malicious or flawed applications. It includes a set of sample security policy configuration files designed to meet common, general-purpose security goals."

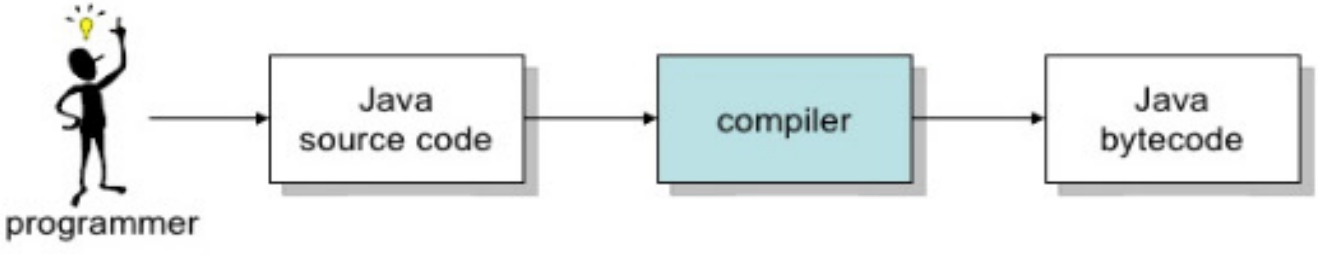
- NSA Security-enhanced Linux Team

Policy in MAC system (SELinux)

- The behavior or control what is allowed or not is handled through **a policy file**
- In SELinux where MAC is implemented on top of a DAC system
- **The kernel checks and enforces DAC policy rules before MAC rules, so it does not check SELinux policy rules if DAC rules have already denied access to a resource**
- The specification of the policy (file) is the heart of the MAC system
- Reference selinux policies: can be downloaded at *https://github.com/TresysTechnology/refpolicy/wiki*

JAVA As Trusted Execution Environment

Java Language



Java characteristics

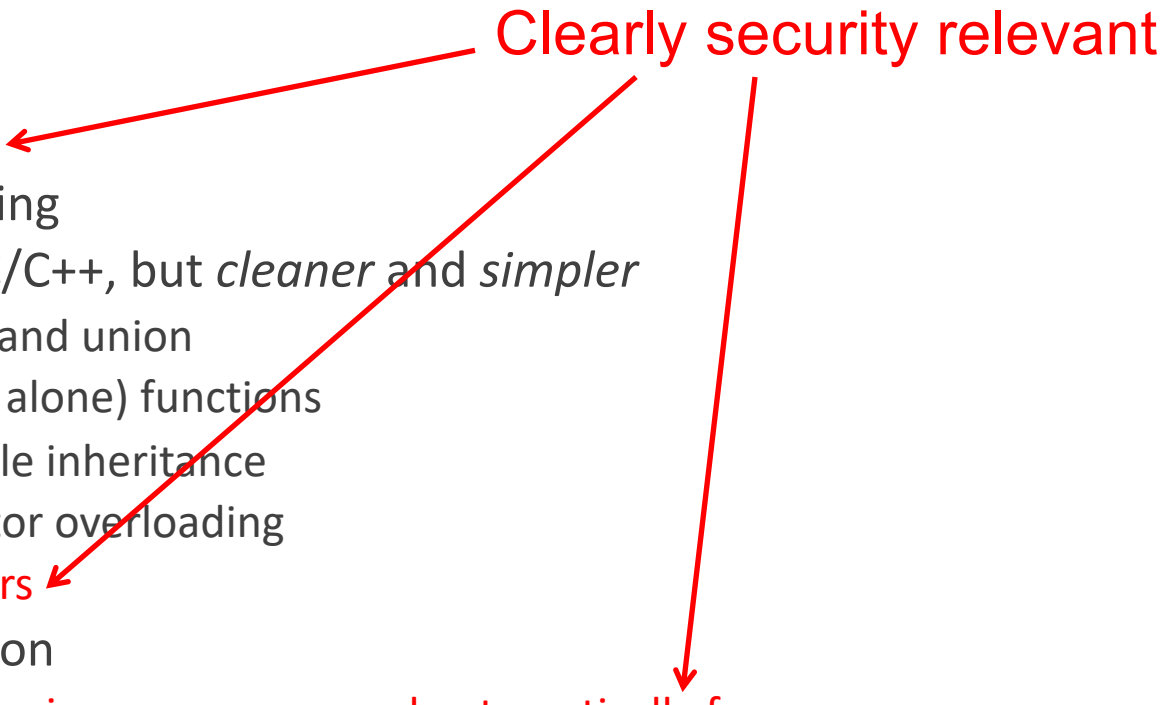
- Java is a high-level programming language

- Java is compiled and interpreted
  - Source code is compiled into bytecode (low-level, platform independent code)
  - Bytecode is interpreted (real machine code produced at run time)
  - Fast and portable (“write once run anyway”)

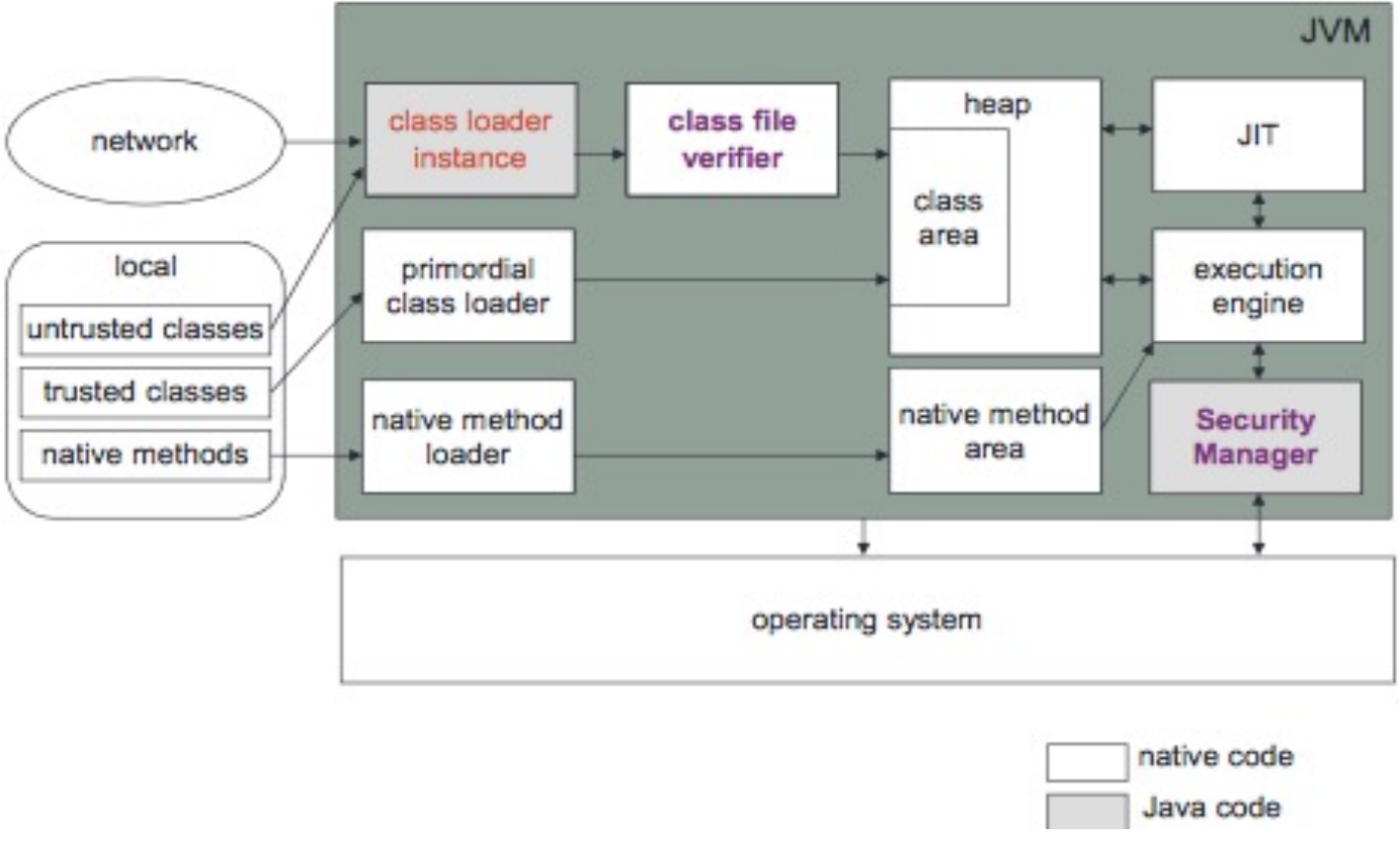
- Dynamic linking (no link phase at compile time)
  - Program consists of class definitions
  - Each class is compiled into a separate class file
  - Classes may refer to each other, references are resolved at run-time

Java language features

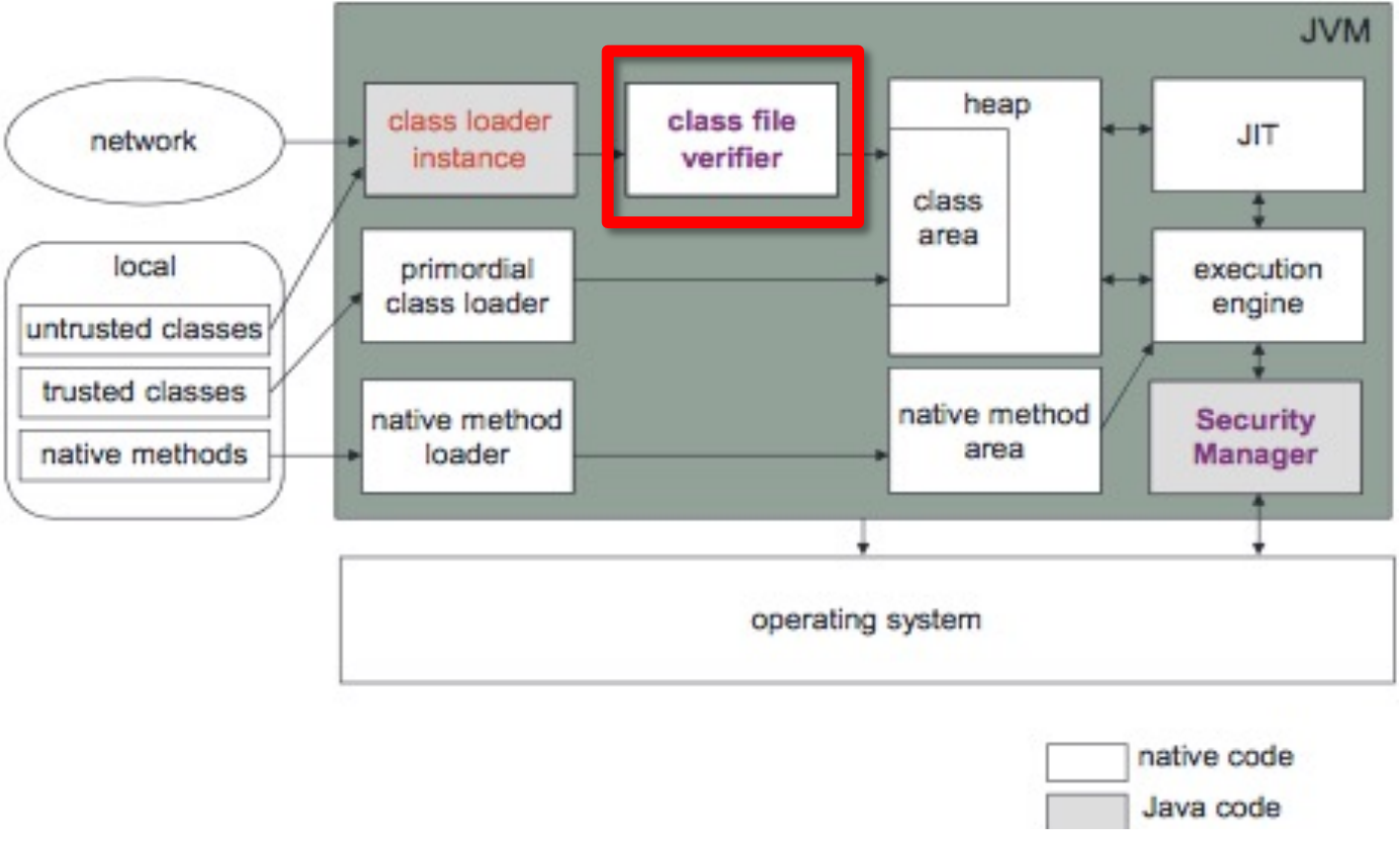
- Object-oriented
- Multi-threaded
- Strongly typed
- Exception handling
- Very similar to C/C++, but *cleaner* and *simpler*
  - no more struct and union
  - no more (stand alone) functions
  - no more multiple inheritance
  - no more operator overloading
  - no more pointers
- Garbage collection
  - objects no longer in use are removed automatically from memory



Java Virtual Machine



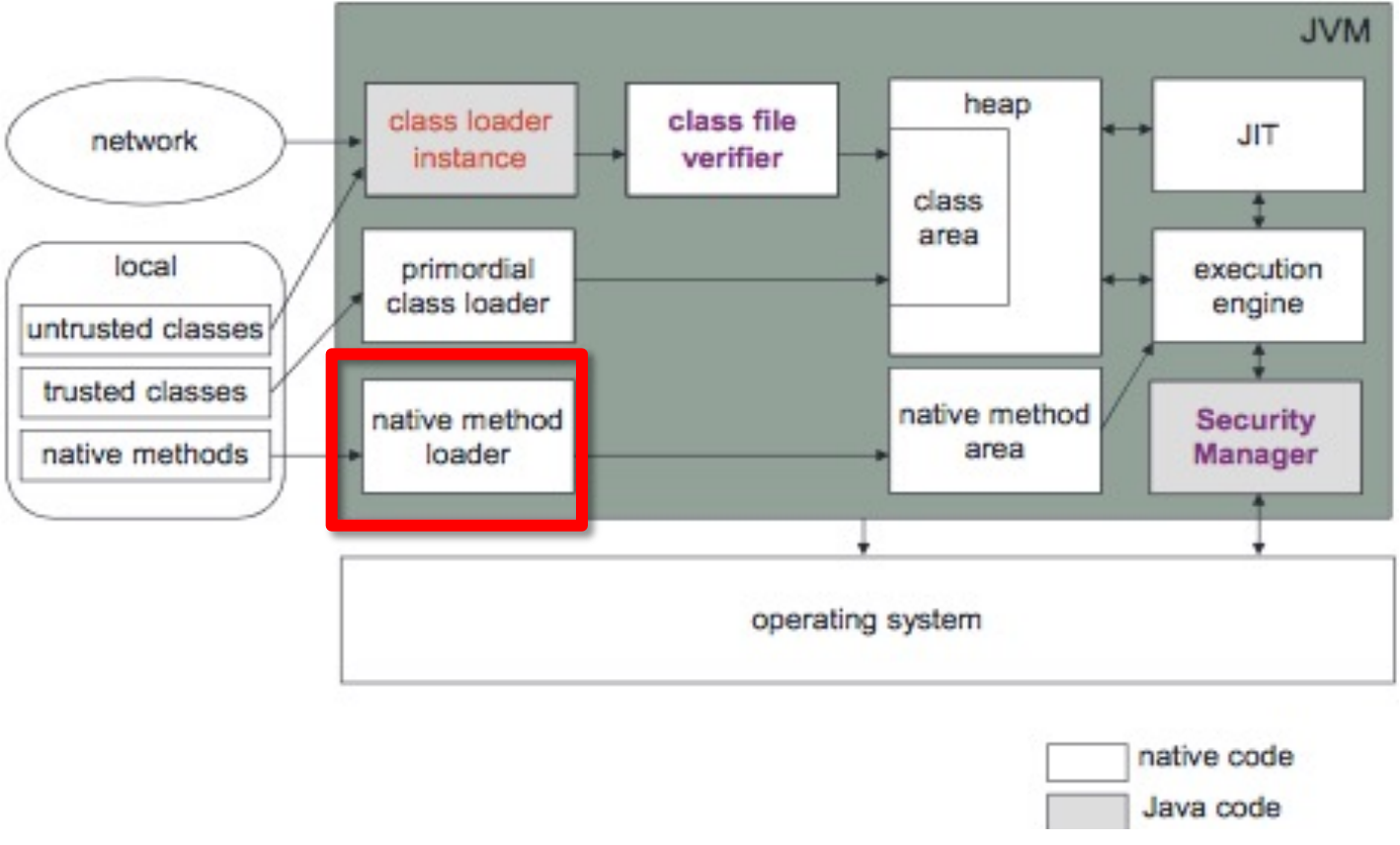
Java Virtual Machine



JVM – Class File Verifier

- **Class file verifier**
  - Checks untrusted class files
    - Size and structure of the class file
    - Bytecode integrity (references, illegal operations, ...)
    - Some run-time characteristics (e.g., stack overflow)
  - A class is accepted only if it passes the test

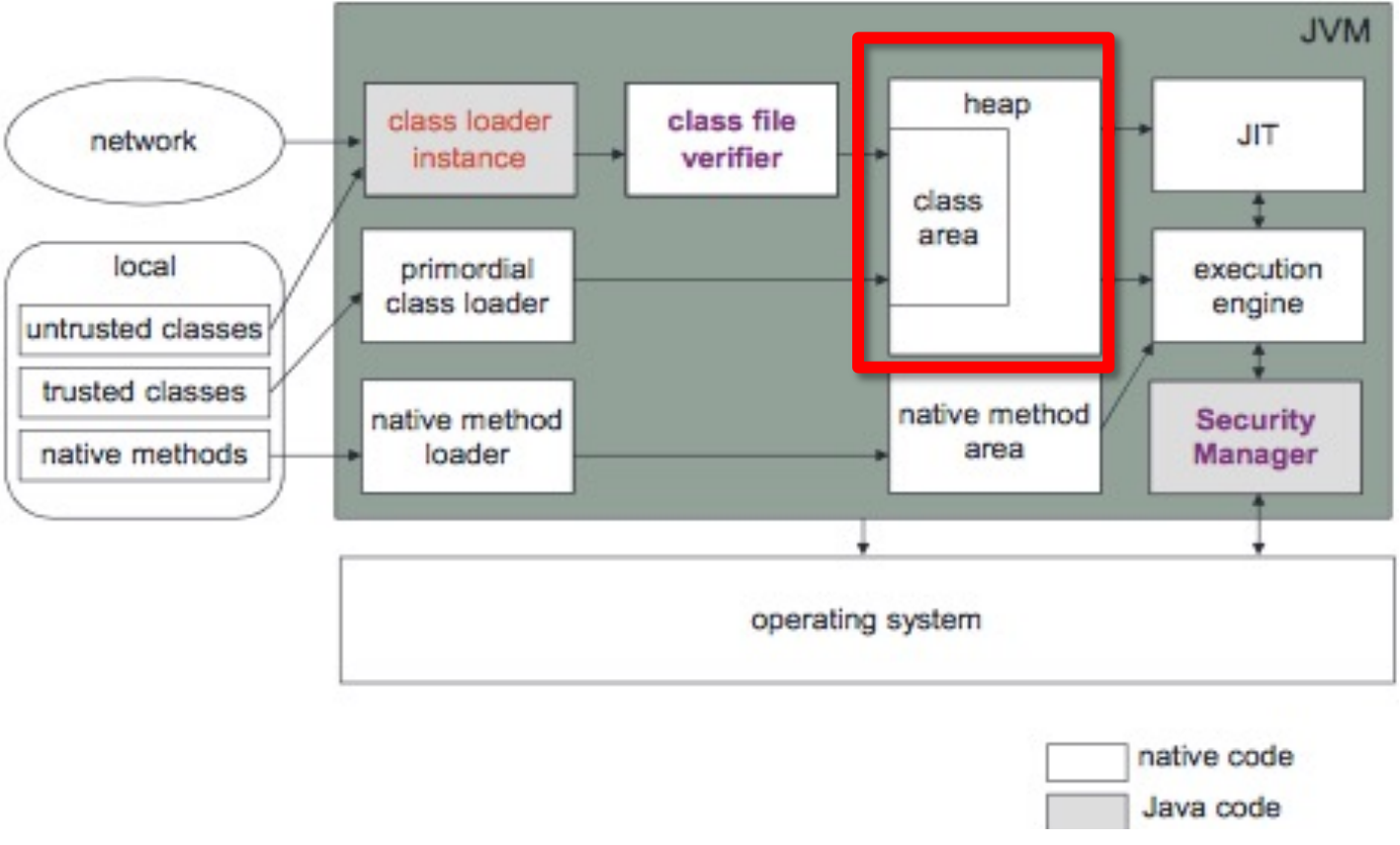
JVM



JVM – Native method loader

- **Native method loader**
  - Native methods are needed to access some of the underlying operating system functions (e.g., graphics and networking features)
  - Once loaded, native code is stored in the native method area for easy access

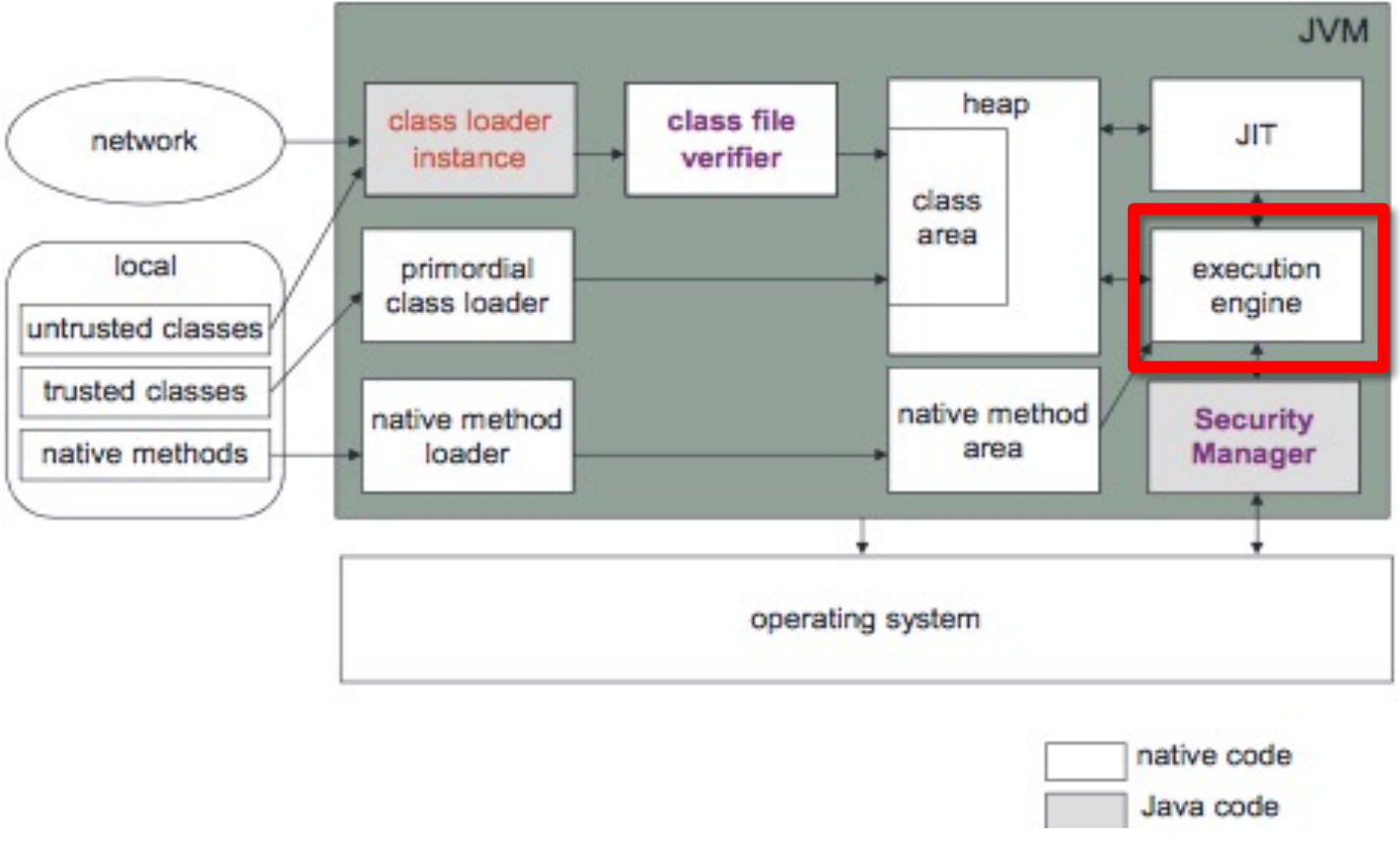
Java Virtual Machine



JVM - Heap

- The heap
  - Memory used to store objects during execution
  - How objects are stored is implementation specific

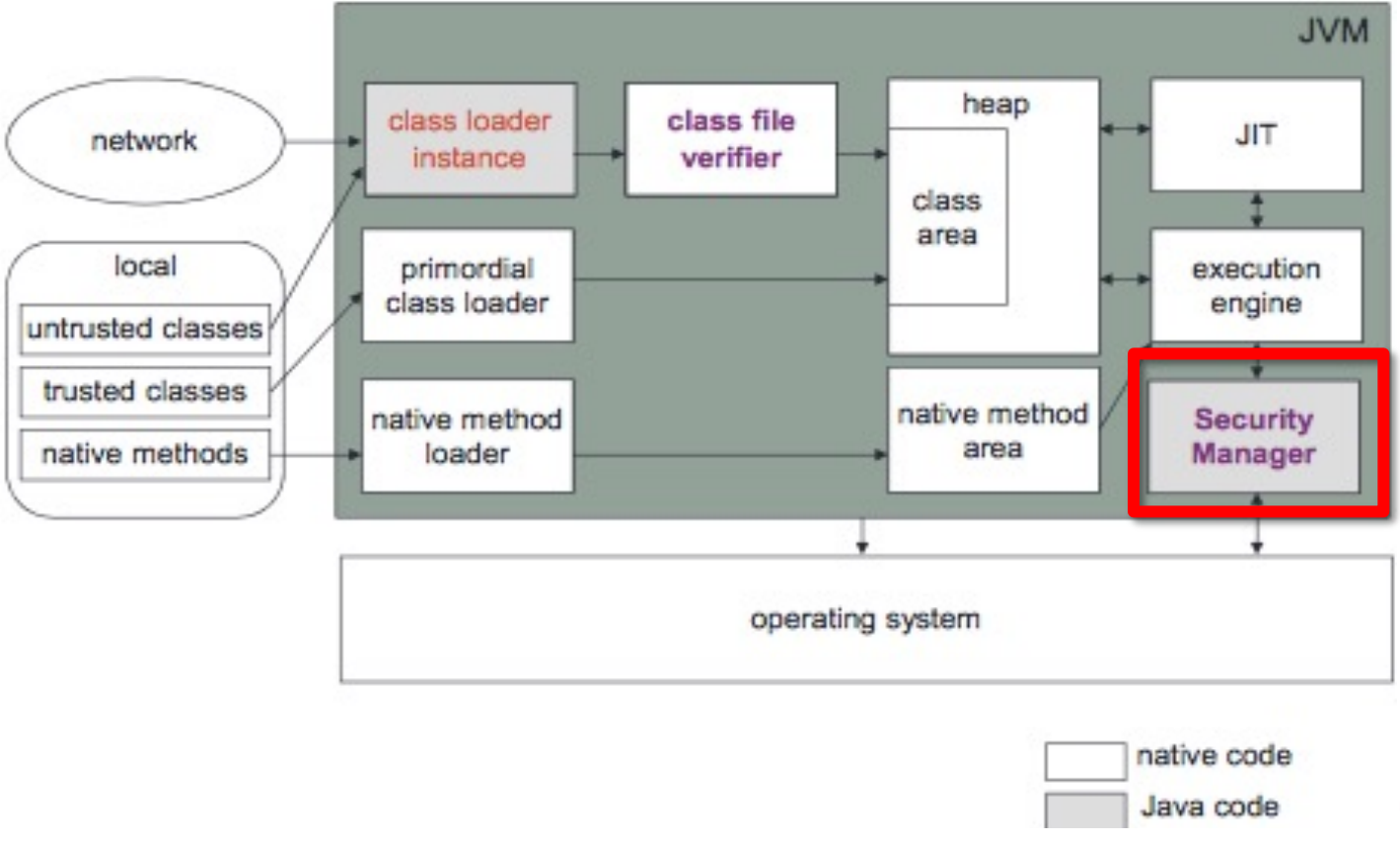
Java Virtual Machine



JVM – Execution Engine

- **Execution engine**
  - a virtual processor that executes bytecode
  - has virtual registers, stack, etc.
  - performs memory management, thread management, calls to native methods, etc.

Java Virtual Machine



JVM – Security Manager

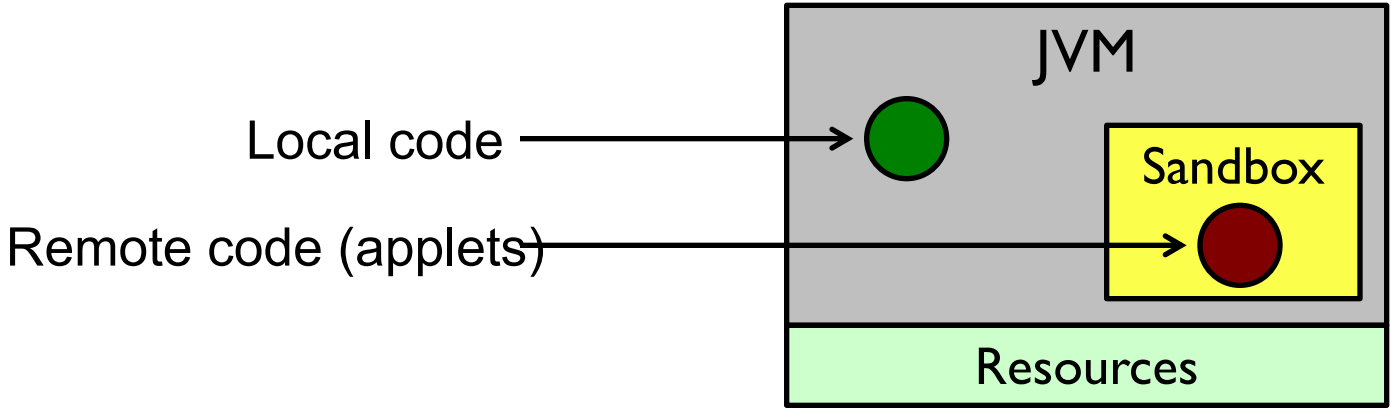
- **Security Manager**
  - Enforces **access control** at run-time (e.g., prevents applets from reading or writing to the file system, accessing the network, printing, ...)
  - Application developers can implement their own Security Manager
  - Or use the policy based SM implementation provided by the “standard” JDK
    - Extend `checkPermission` to add new rules
    - `-Djava.security.manager` to provide custom SM
  - Diminished in server-side environments and with the deprecation of applets
    - For applets and client-side Java applications, the role of the Security Manager has

Java security models

- The **sandbox** (Java 1.0)
- The concept of **trusted code** (Java 1.1)
- Fine grained **access control** (Java 2)

Java 1.0: The sandbox

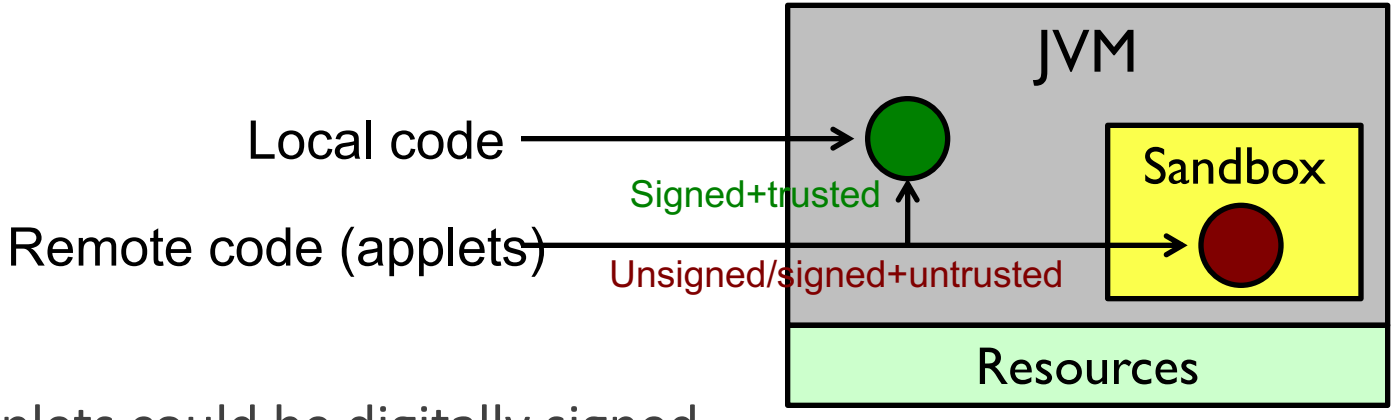
- **Idea**: limit the resources that can be accessed by **applets** (this creates an execution sandbox)



- Local code had unrestricted access to resources
- **Downloaded code (applet)** was restricted to the sandbox
  - cannot access the local file system
  - cannot access system resources
  - can establish a network connection only with its originating web server

Java 1.1: The concept of trusted code

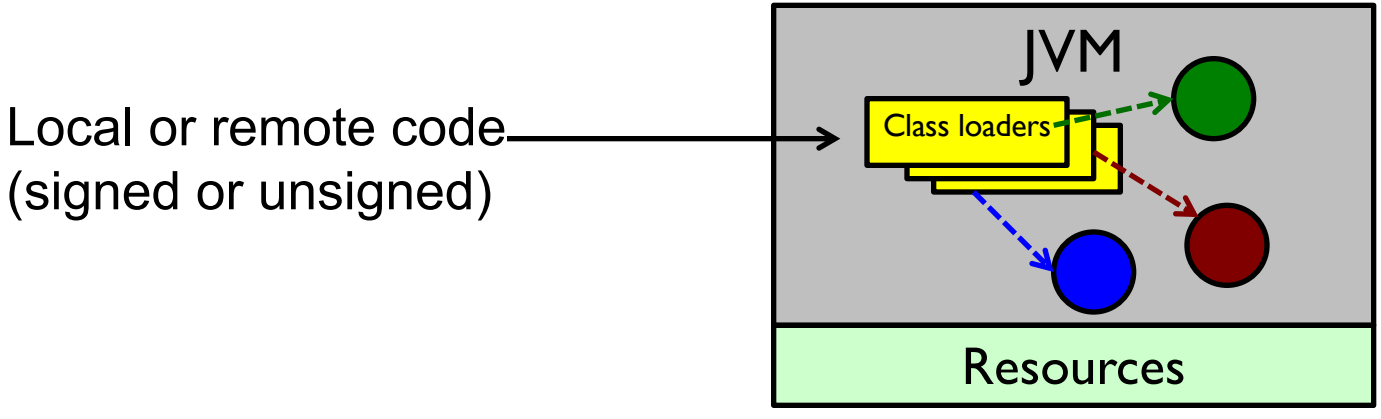
- **Idea**: applets that originate from a **trusted source** could be trusted



- Applets could be digitally signed
- Unsigned applets and applets signed by an untrusted principal were restricted to the sandbox
- Local applications and applets signed by a trusted principal had unrestricted access to resources

Java 2: Fine grained access control

- **Idea**: every code (remote or local) has access to the system resources based on what is defined in a **policy file**



- A **protection domain** is an association of a code source and granted permissions
- The code source consists of a **URL** and an optional signature
- **Permissions** granted to a code source are specified in **policy file**

Java's impact

- The Java system has been an example for many other languages, execution environments, systems

- E.g.
  - .Net
  - Android

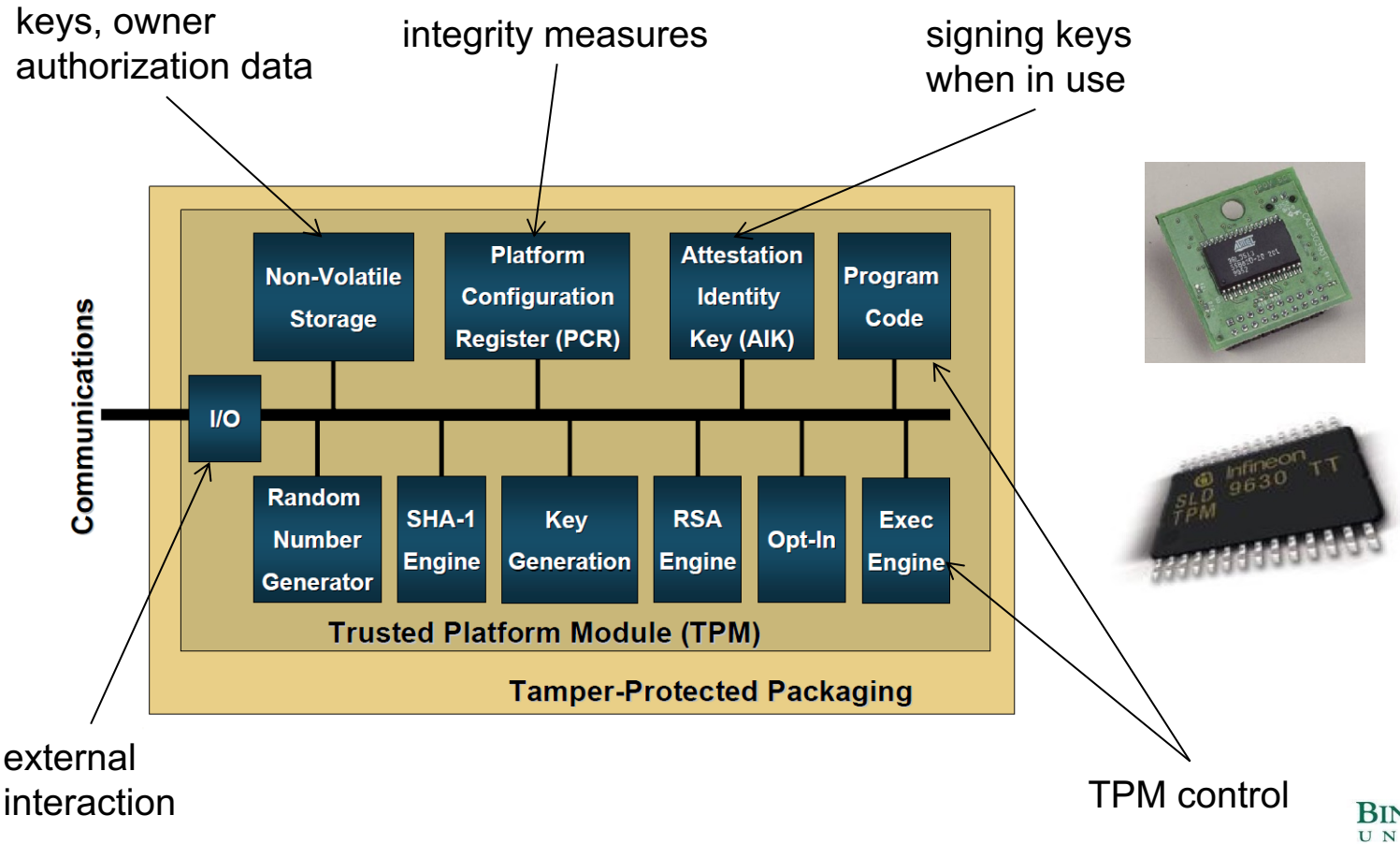
Hardware TCB:
Trusted Platform Module

Trusted Platform Module (TPM)

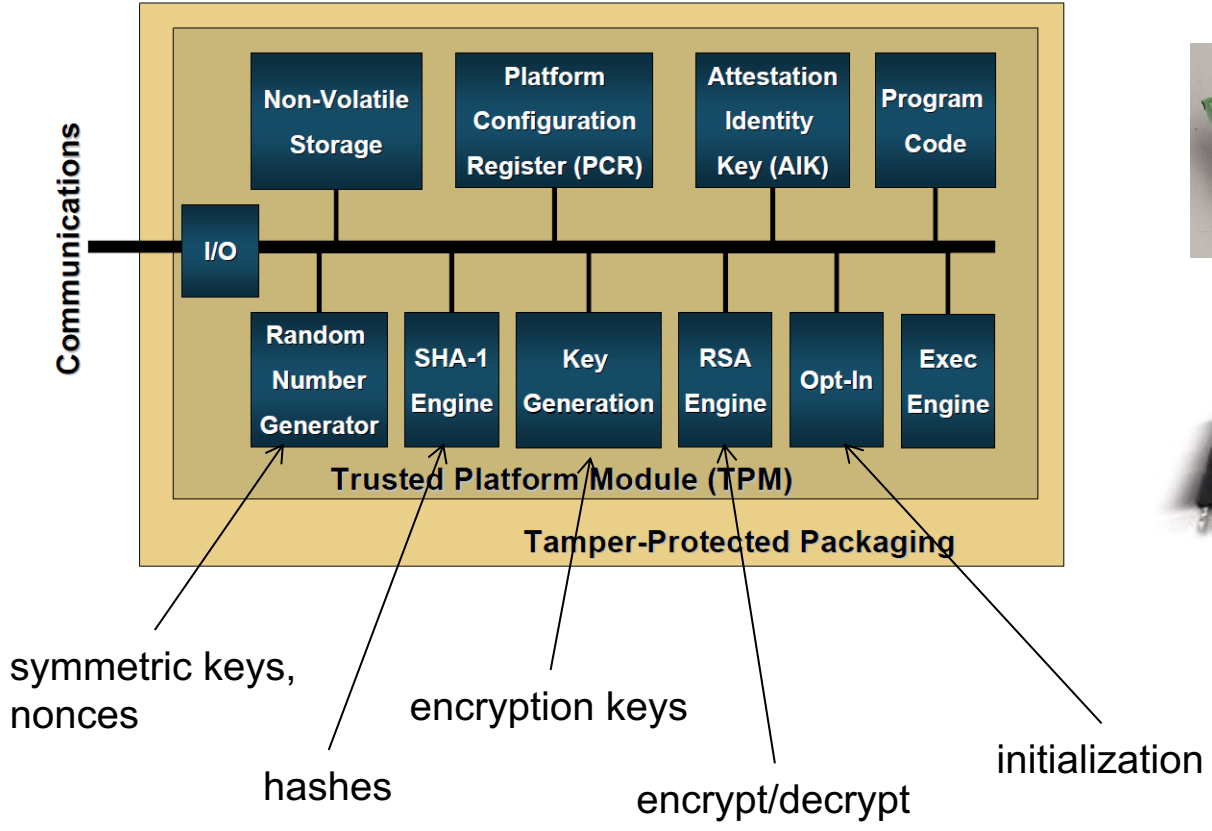
- Standard defined by the Trusted Computing Group
- Availability
  - Hardware chip in 100M laptops
  - HP, Dell, Sony, Lenovo, Toshiba,…
  - HP alone ships 1M TPM-enabled laptops each month
- Core functionality
  - Secure storage
  - Platform integrity reporting
  - Platform authentication

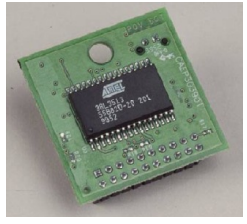
Example: Infineon TPM

TPM Architecture



TPM Architecture

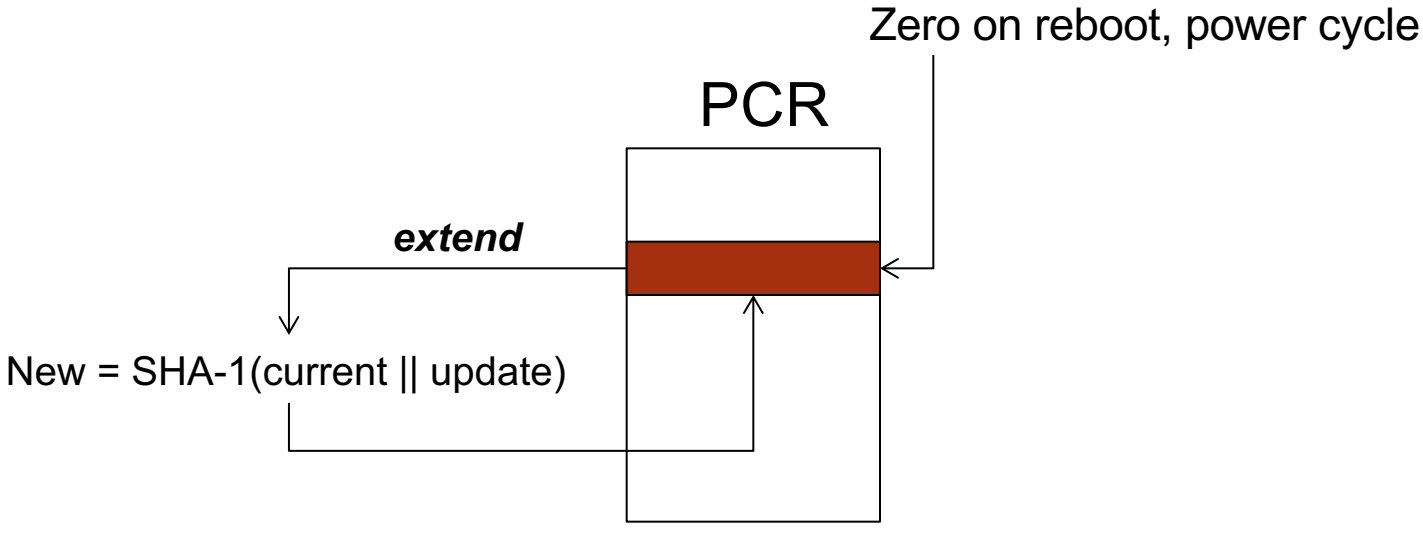




TPM Functions

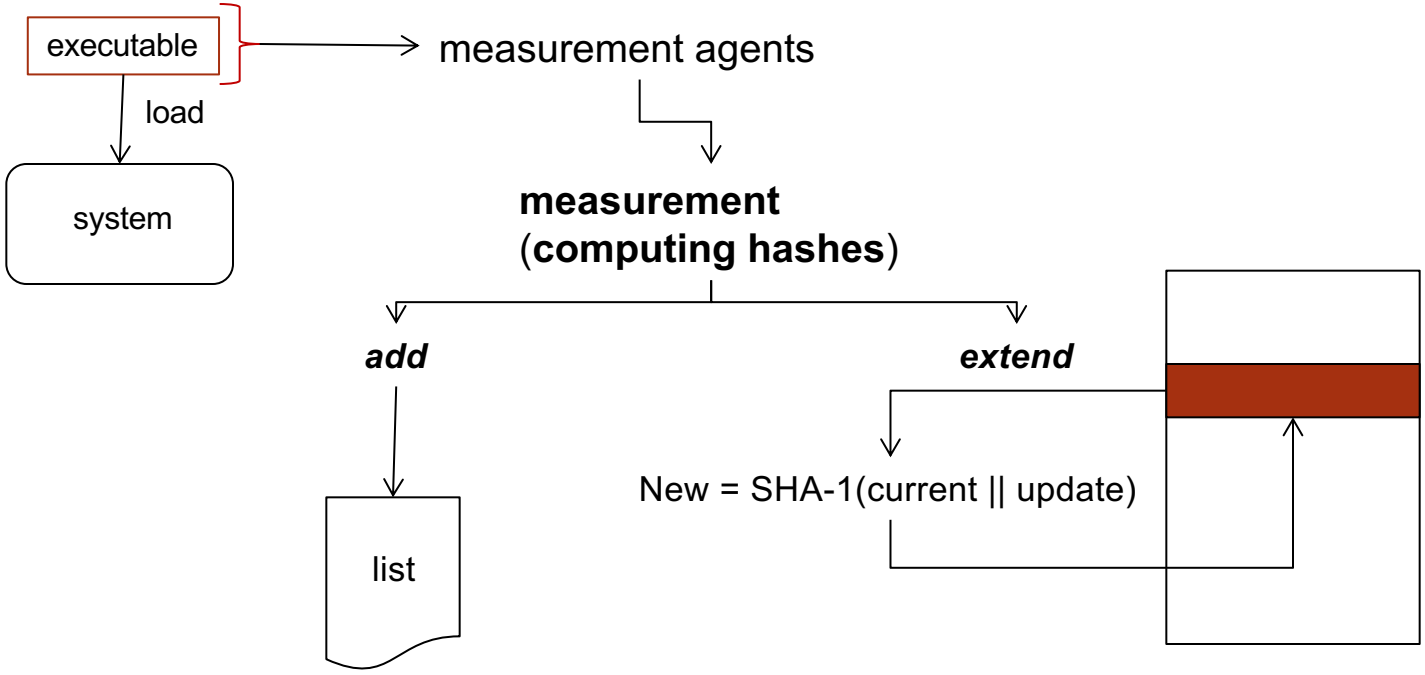
- Integrity measurement
- Attestation
- Sealed Storage

Platform Configuration Registers (PCR)



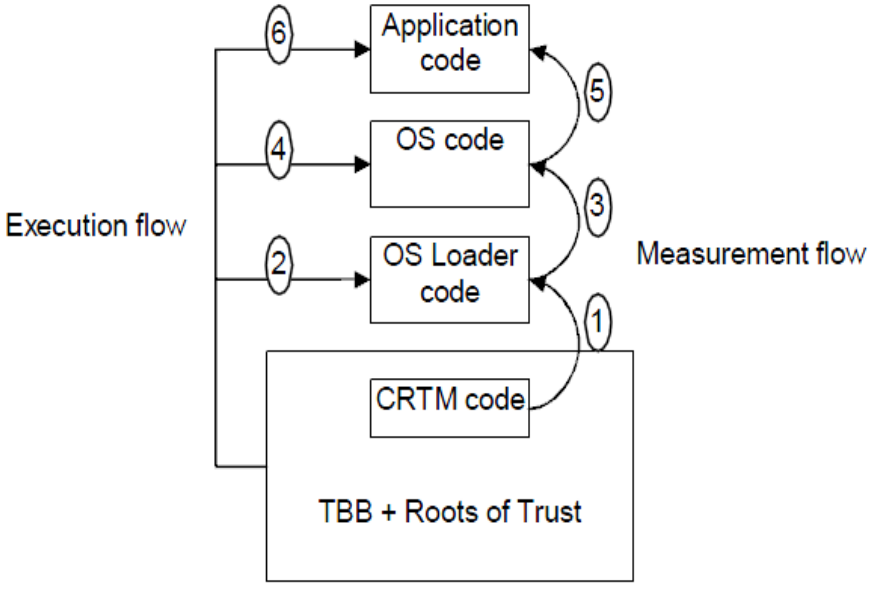
- At least 16 PCR registers, each register stores 20 bytes
- Store the calculated hashes of system components for each boot

Maintaining a measurement List



- Measurement List
  - **Hashes** of system components
- PCR contains the **linked hash** of all measurements in the list
- **Verification** against known good values
  - Alterations to the list values can be detected

Example: Integrity Measurement of Secure Boot



- Measure a component before executing it
- Record the measurement as a **hash value** of the code/data
- 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 attack

```
#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

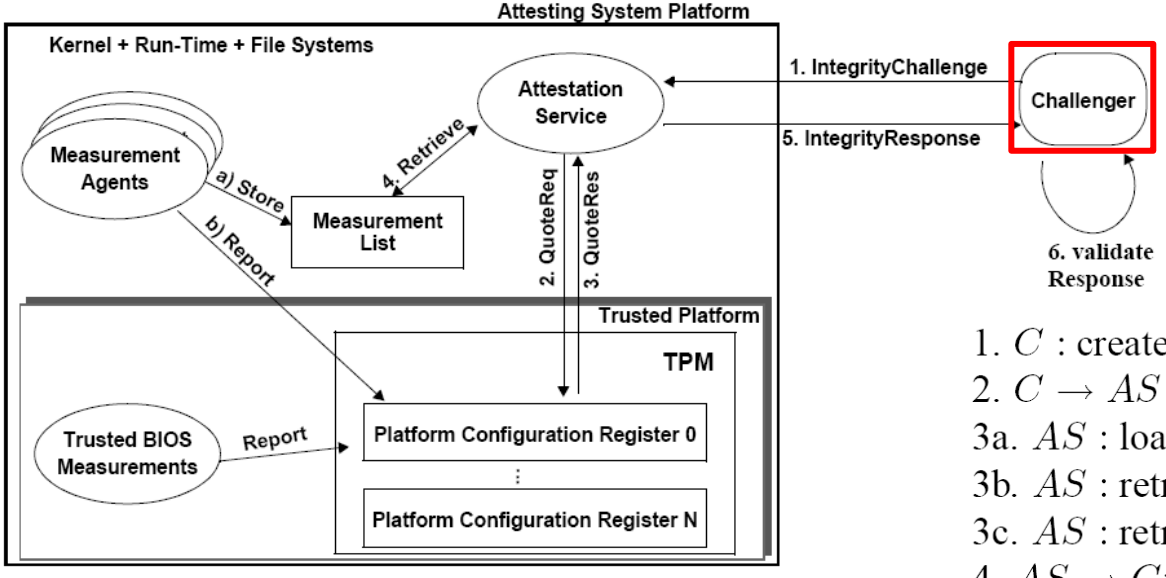
Long-term Keys

- 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 (EK)**
  - **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 (SRK)**
  - Basis for a key hierarchy that manages secure storage

Attestation Identity Key(AIK)

- AIK
  - Created by TPM to attest the identity of the device
  - Based on EK
  - Without revealing the EK
- Device use AIK to sign a quote
  - Quote is integrity measurement(hash) + nouce
- The remote party verifies the AIK-signed quote use corresponding AIK public key
  - E.g., Positive decision can let the device join a cluster; let the device use certain remote resources(software)

Remote Attestation

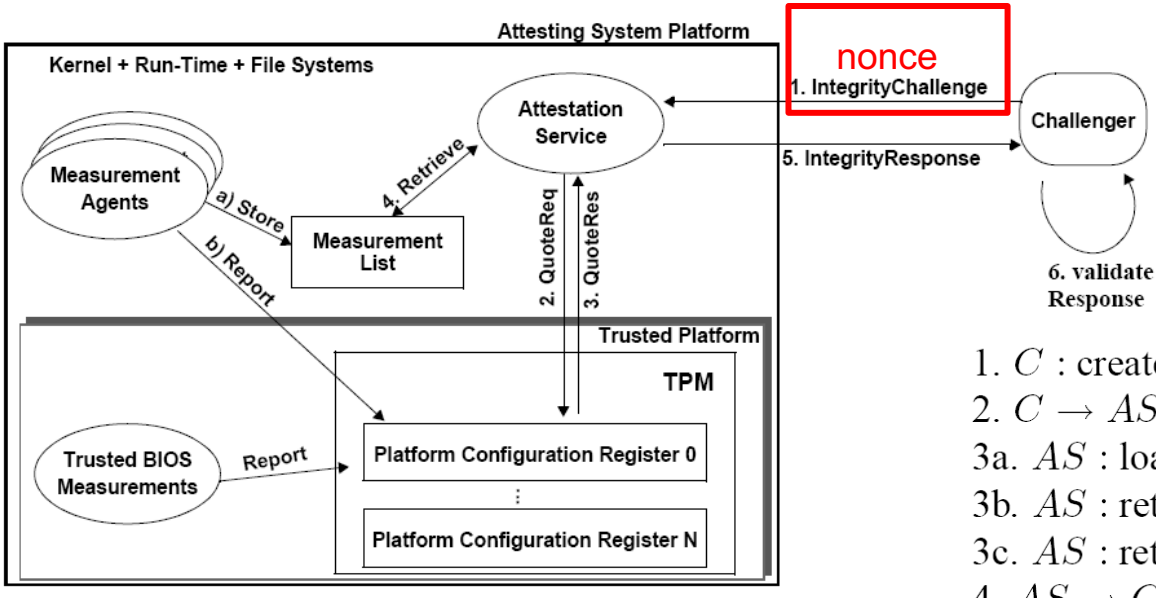


Remote Challenger
Challenge integrity of device

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

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

Remote Attestation



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)

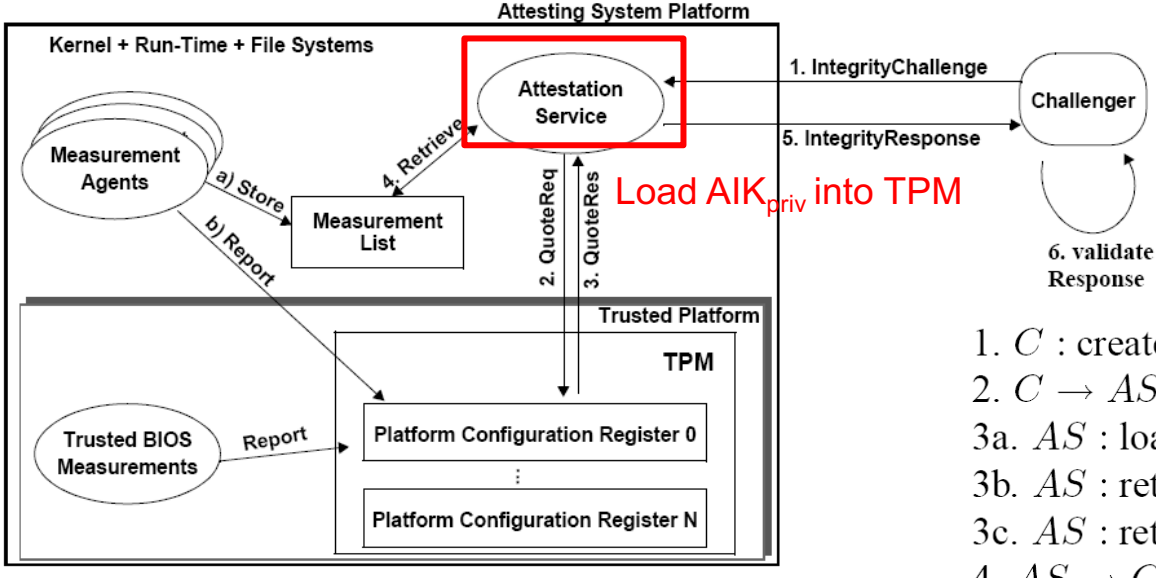
5a. C : determine trusted $cert(AIK_{pub})$

5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$

5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

Remote Attestation



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)

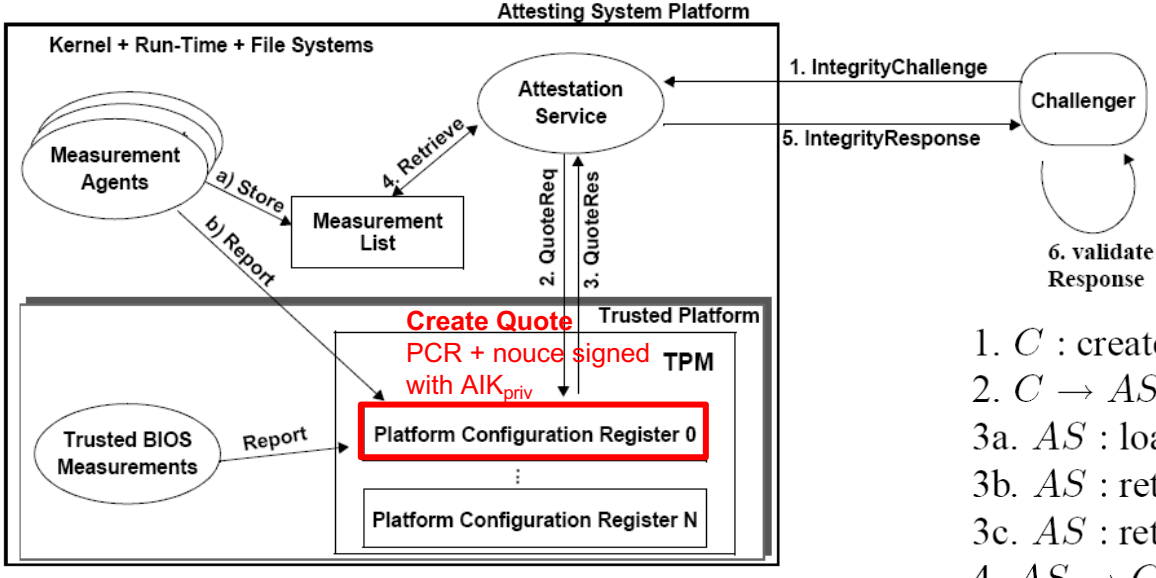
5a. C : determine trusted $cert(AIK_{pub})$

5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$

5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

Remote Attestation



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)

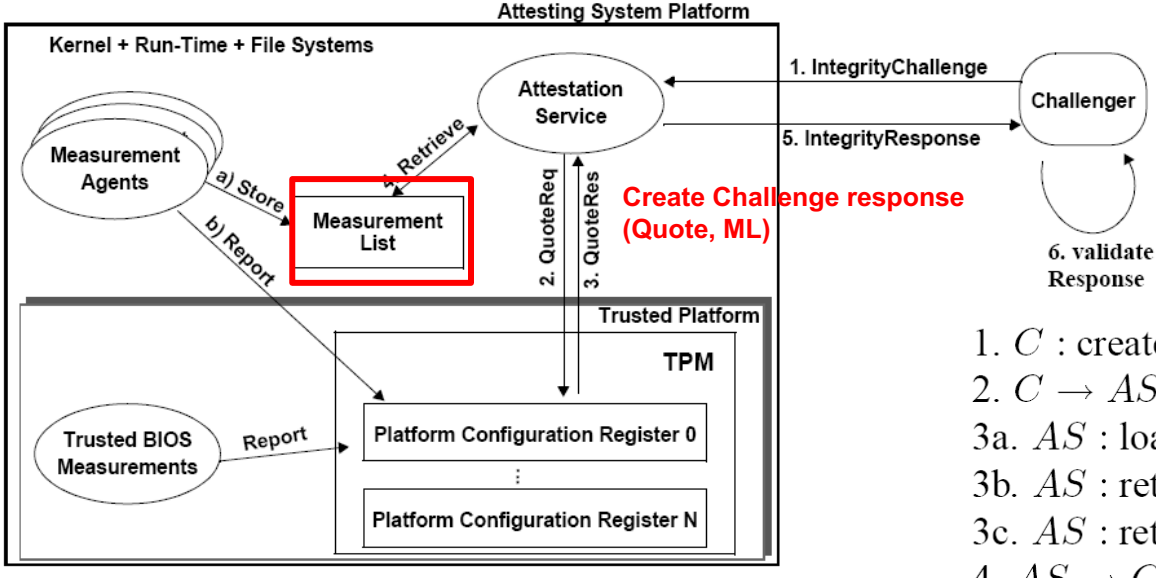
5a. C : determine trusted $cert(AIK_{pub})$

5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$

5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

Remote Attestation



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)

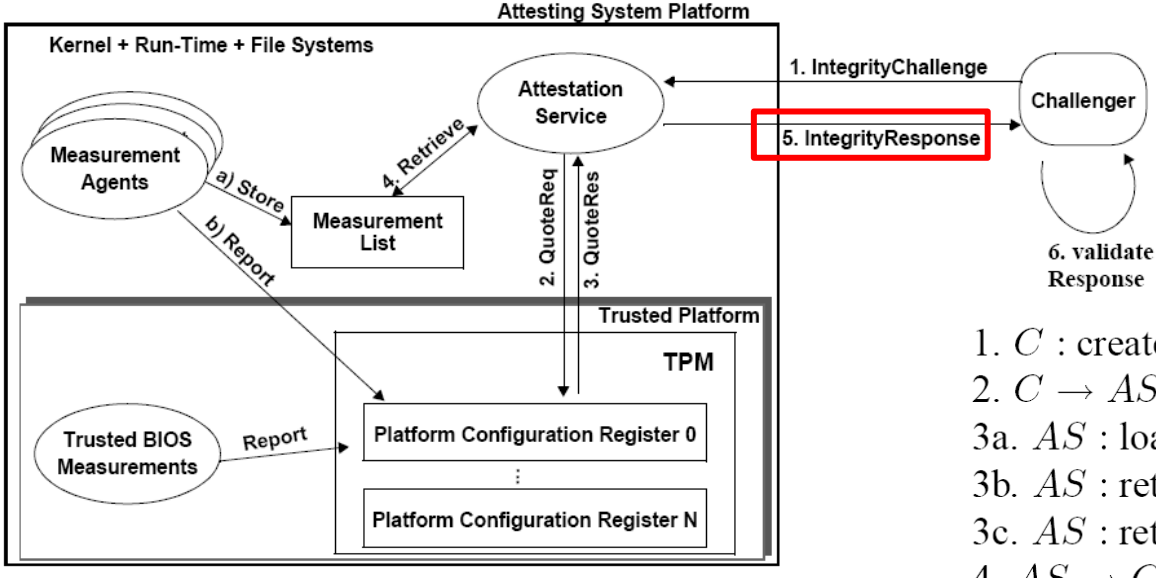
5a. C : determine trusted $cert(AIK_{pub})$

5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$

5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

Remote Attestation



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)

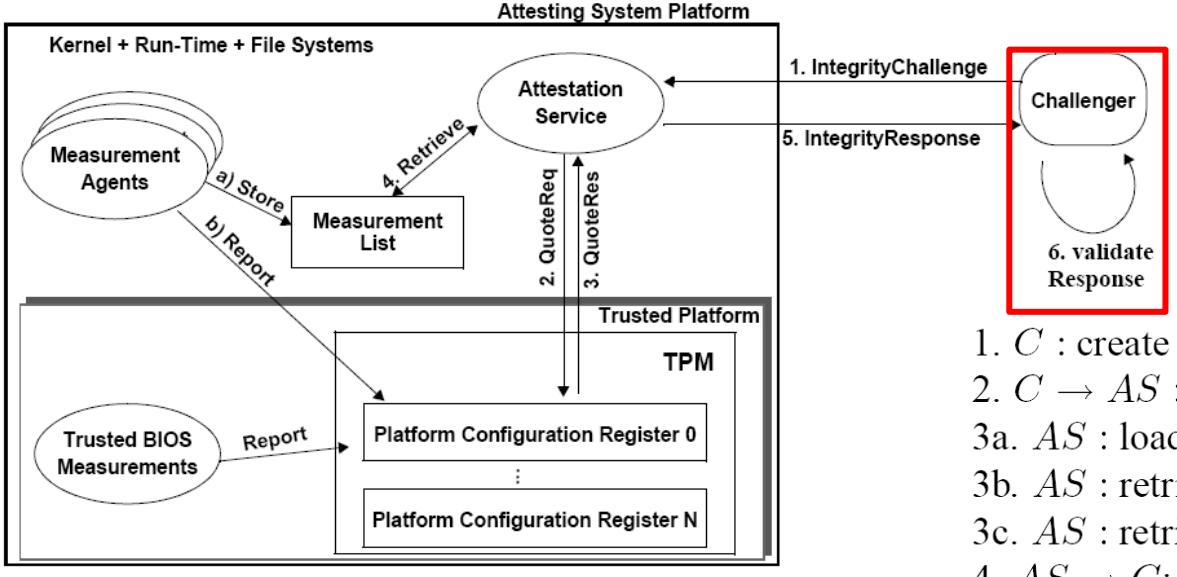
5a. C : determine trusted $cert(AIK_{pub})$

5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$

5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

Remote Attestation



Verify identity using AIK_{pub}
Validate nonce and ML

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

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

Sealed Storage

- Goal: ensure that information is accessible only when the system is in a known/acceptable state
- System state determined by PCR value
- Encryption/Decryption
  - Encrypt/decrypt data only when system in ”good” state
  - Key is managed by TPM
- **Usage: Full Disk Encryption**
  - Sealed storage can be used for full disk encryption systems where the decryption key is sealed by the TPM. Only if the system boots in the expected, secure state can the key be accessed to decrypt the disk

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)