SYSTEM SECURITY II: TRUSTED COMPUTING

TDDD17 Informationssäkerhet

Ben Smeets

Ericsson Research Security / Lund University

During the lecture we will not use all the slides of this deck and leave these additional slides for own reading and self-study material

Goal of this lecture

- Understand trusted computing and its purpose
- Threats to compute HW
- Get a basic insight in technologies to achieve trusted computing in devices, servers, and cloud infrastructure
- Meet technical approaches to build trustworthy ICT systems
  - In the first part you already saw approached used in operating systems and VMs with access control and the use of memory protection

Overview

- Why trusted computing
  - Intuitive model for trusted computing
  - Roots of trust
  - Hardware versus software
- Software secured execution environment:
  - Java, SELinux
  - Virtualization
- CPU secured execution environment: TrustZone, SGX
- Trusted Computing Group (TCG):
  - What is a TPM?
  - Use of TPM : Secure boot, Trust pools in Openstack
- Special trusted computing devices: HSM, smart cards
- Study questions

Complimentary reading

Trustzone:

Web page: https://genode.org/documentation/articles/trustzone

SGX:

Article: *SGX explained*, Victor Costan and Srinivas Devadas, : https://eprint.iacr.org/2016/086.pdf

TCG & TPM:

Ebook: *A Practical Guide to TPM 2.0, Using the Trusted Platform Module in the New Age of Security***.** Arthur, Will, Challener, David

HSM

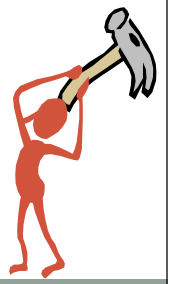
Web page SANS: https://www.sans.org/reading-room/whitepapers/vpns/overview-hardware-security-modules-757

Smartcard:

Book (expensive but good): *Smart Card Handbook, 4th Edition,* Wolfgang Rankl, Wolfgang Effing

New Security Challenges

- Computing devices are becoming distributed, unsupervised, and physically exposed
  - Computers on the Internet (with untrusted owners)
  - Embedded devices (cars, home appliances)
  - Mobile devices (cell phones, PDAs, laptops)
  - Base stations and wireless access points
- Attackers may physically tamper with devices
  - Invasive probing
  - Non-invasive measurement
  - Install malicious software



Why is Trusted Computing so hard?

Security

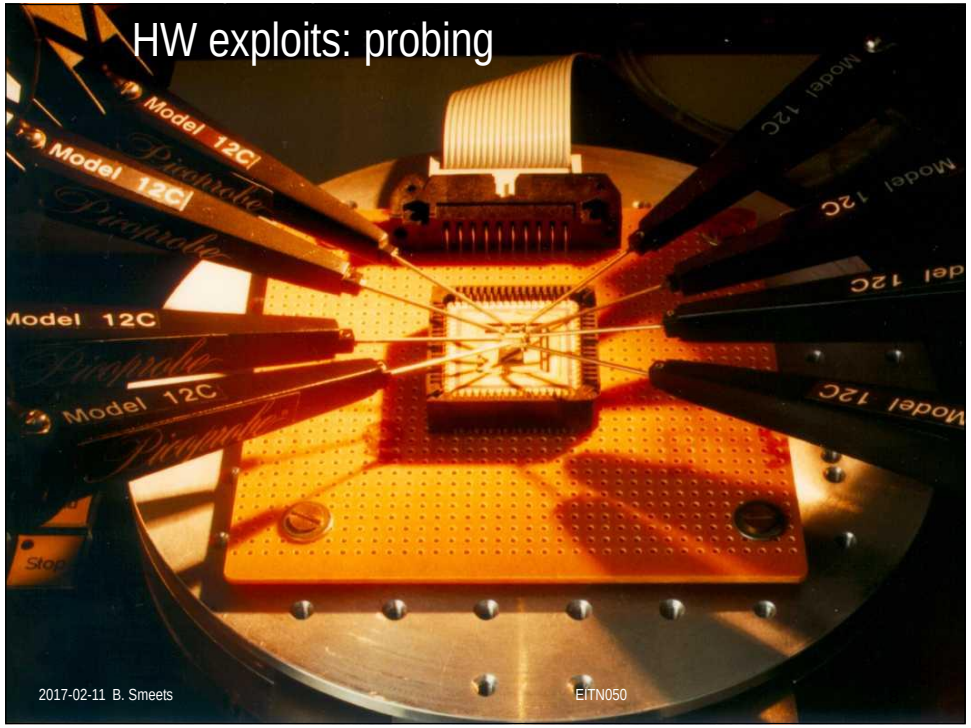
Attacks

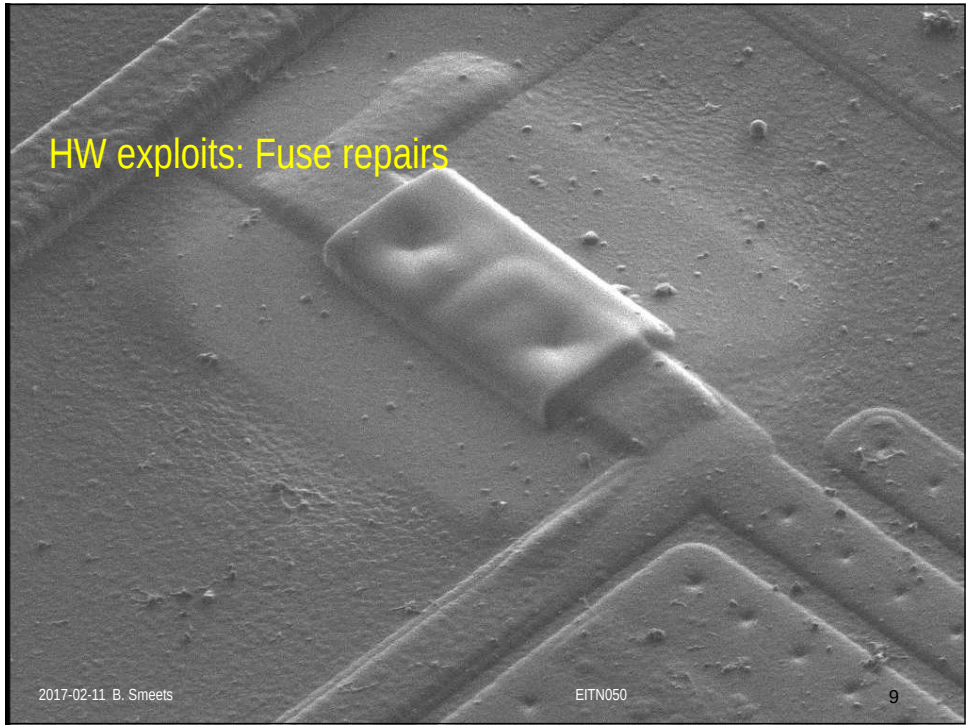
- SW only based attacks
- HW based attacks
- Other (unexpected) ways

} Hybrid forms

- The opponent can use the attack tools he/she wants. You cannot say ‘this attack is not allowed’
- The unexpected

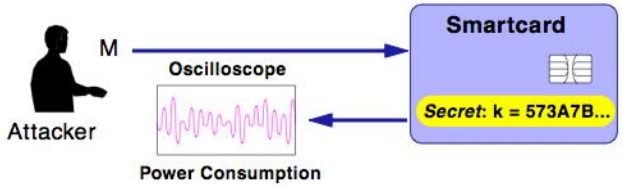
HW exploits: probing





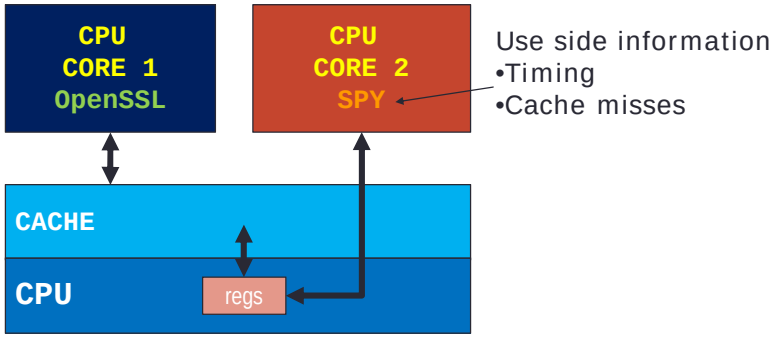
Example of the unexpected: Attacks on smartcards

- 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 zeroes.



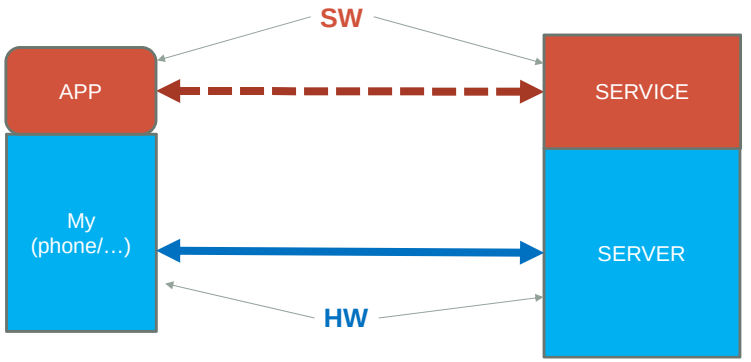
Sidechannel attacks – example in multicore CPU

SPY process gets the key from the program using OpenSSL



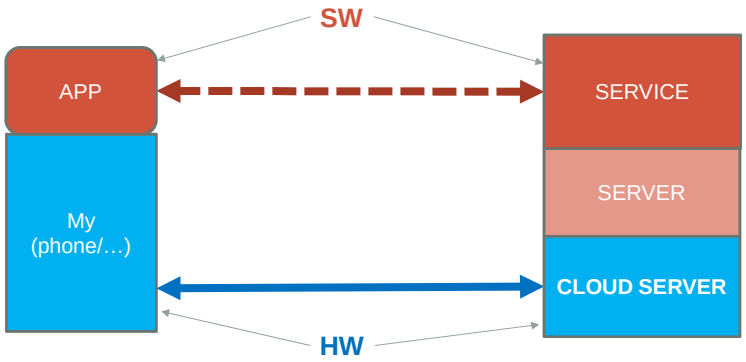
Use side information
•Timing
•Cache misses

How to trust a (remote) service?

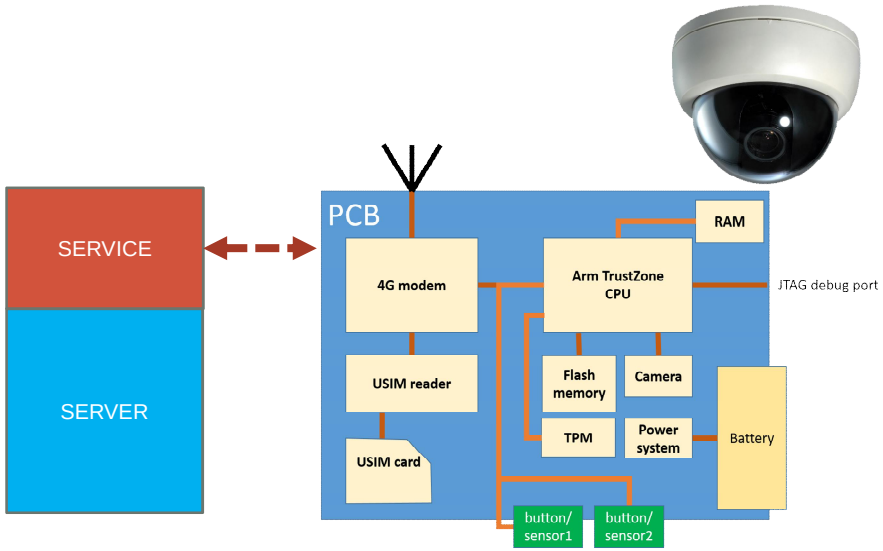


Is the division in SW and HW meaningful?

Or how to trust a service in the cloud?



Or how to build a trusted camera?



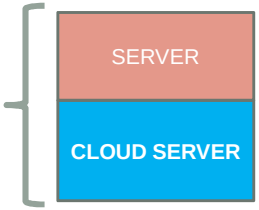
From trusted computing to trusted platform

- Trusted Computing
  - ´Requires that the (application) software can be trusted

SW security: Not the subject of this lecture

  - Requires that the underlying system can be trusted

Trusted Platforms



Useful Benefits of a trusted platform

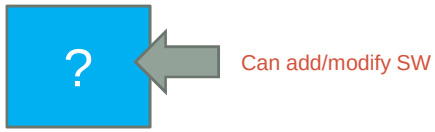
When a platform can *prove* that it is running the expected executable

- Third-party (grid) computing
  - Produce correct results
- Supporting Bring Your Own Device (BYOD) policy
  - Data of different stakeholders is kept safe on the device
- Electronic payments
  - Correct amount, anonymous (to some extend), Trusted UI
- Digital Rights Management (DRM)
  - Enforce copyright on content (music, video, programs, etc)
- Sensor and surveillance
  - Can rely on the data that is received
- AND …

Intuitive platform models

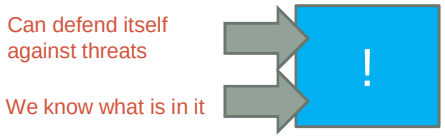
- **Open platform**

  (e.g. PC, PDA, iOS, Android, Linux device)
  - General purpose computing platform



- **Closed platform**

  (e.g. ATM, set-top box, game console, satellite receiver, most older (pre iOS, Android) mobile phones)
  - Special purpose computing device



Intuitive models cont'd

Trusted computing combines best properties of

- **Open:** allow applications from many different sources to run on same platform
- **Closed:**
  - Only "known" software can execute
  - Isolation of sw components to limit propagation of malicious code
  - remote parties can determine what software is running and whether to expect the platform to be well behaved

What do we want of a trusted platform?

A semi-open platform that is both

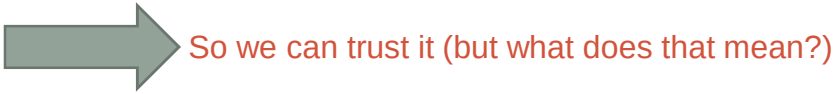
OPEN:

we can add/modify with good sw components

CLOSED:

platform can defend itself against threats

we want to know what is running in it



Trusted vs Trustworthy

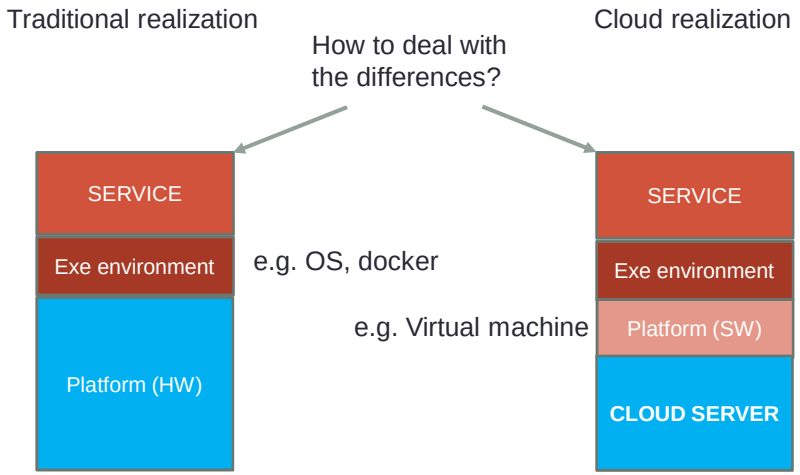
What are we after, a trusted or trustworthy platform?

Trusted: A system can be trusted but is it trustworthy?

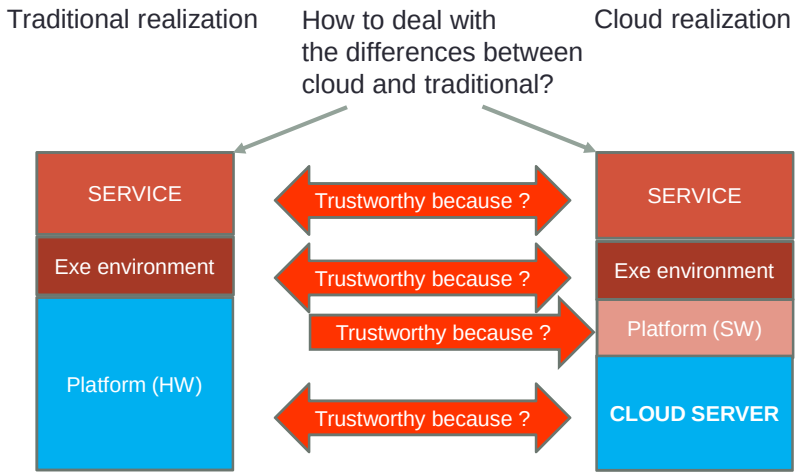
Trustworthy: The system can fullfill the requirements defined by a methodology. Is the methodology then trustworthy (and we get a recurssion) or we just trust the methdology.

Recall: Using Common Criteria a system that is successfully evaluated at level EALx is trustworthy.

How to obtain trustworthiness ?

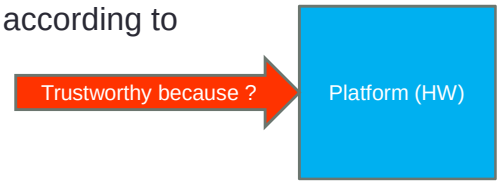


How to obtain trustworthiness ?



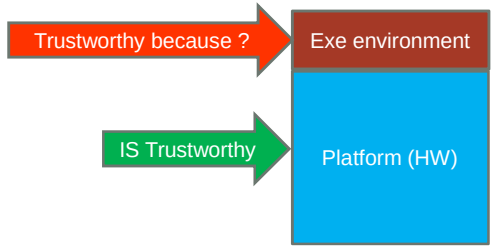
How & why trust HW

- Trust by reputation
- Trust by relying on a third party
- Assurance of design
  - Review
  - Proofs (by modeling of HW)
- Assurance of production
  - HW is produced according to design

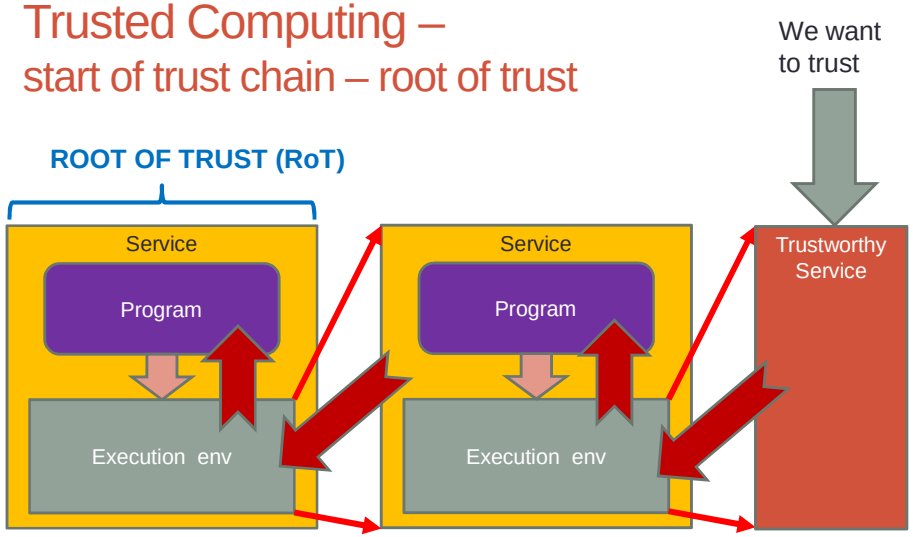


How & why trust execution environment

- Trust by reputation
- Because we checked the code and put it there
- Because HW will only start code we approved (and checked to be OK)



Trusted Computing – start of trust chain – root of trust



Recurssion must stop at a service we trust/have to trust, e.g. Intel HW.

Note: RoT is not only data (e.g. keys) but also logic, therefore we say that a RoT is an engine.

Root of Trust

- So the RoT(Root of Trust) is that part of the system we consider trustworthy.
- **Why can it be trustworthy ?**
  - We did a very carefull analys of the design and implementation
    - Recall Common Criteria (from Computer Security course)
  - We can verify that some one else (a third party) considers the part of the system trustworthy
    - Compliance statement, certificate (e.g. according to US FIPS 140-2)

The different Roots of Trust

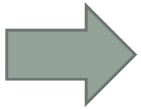
- It is useful to think a RoT to consist of different RoTs each with a special task
- The RTS (RoT for Storage):
  - A compute engine that protects use and access to data/keys
- The RTM (RoT for Measurement):
  - A computing engine capable of making reliable integrity measurements.
- The RTR (RoT for Reporting):
  - A computing engine capable of reliably reporting information held by the RTS

Industry specifications/requirements on RoTs

- Trusted Computing Group (TCG)
- GlobalPlatform
- NIST (only as draft)

Trust: Hardware vs Software

- Functionality in Hardware
  - hard/costly to change
  - high performance possible
- Functionality in Software
  - Easy to change
  - Difficult to hold private keys



The general view is that HW is more trustworthy than SW realizations

Trustworthy Systems in Software

- **Possible to do** but we have limitations
  - owner of the device on which software runs should not be an attacker
    (he/she and the device ”work together”/”have the same interests”)
  - Does not work when the device in the ”enemy’s territory”
- But ”software only” is sometimes the only implementation option: e.g. virtual platforms

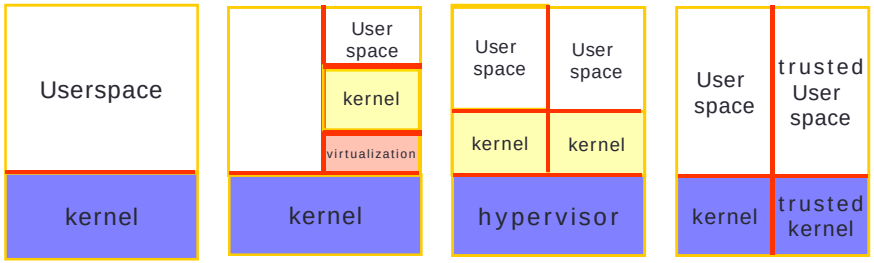
Execution platforms: OSes

- OSes come in many different versions; Linux, Windows X, Android, MeeGo, Symbian, Palm-OS, iOS, etc
- Most simple OSes have no means to securely enforce (multi-user) access control. (relevant for small embedded systems)
- Even if so
  - Correct configuration OS environment is not trivial
  - Without protection user can attack system from ”below”
- SEE: System Security I: Introduction

Execution environments for trusted computing

- Solutions to have best of both, using soft- and hardware protection mechanisms
  - Hypervisor (also called Virtual Machine Monitor (VMM))
    - attestation through virtual device
  - Modify OS
    - try to create isolation (VMs or OS features)
    - SE Linux
  - Change existing hardware
    - attestation done by hardware module
    - add secure execution mode to CPU

OS environment setup for a trustworthy platform



Normal OS
Windows, Linux
SE Linux,
Android
iOS

Virtual Machine
VMWare, KVM,
Virtualbox,
Java VM

Hypervisor
Xen, VMware ESXi,
Microsoft Hyper-V
(L4)

CPU with
trusted mode
e.g.TrustZone
and Intel SGX

partly based on slide material from Dries Schellekens
http://www.esat.kuleuven.be/cosic/seminars/slides/Trusted_Platforms.ppt

We take a closer look at

- Virtualization - brief
- Java - brief
- SELinux - brief
- ARM TrustZone and SGX

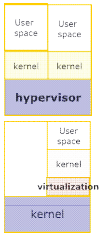
Virtualization

- Abstraction of computer resources
- Pioneered by IBM to keep using legacy system solutions on new hardware without rewriting code
- Turned up to have stability and security benefits (isolation)
  (at the expense of performance)
- There are many ways to do this and there exist therefore many different types of approaches to virtualization

Virtualization approaches

Type 1 and Type 2 virtualization

- Type 1: runs on ”bare metal”
- Type 2: runs on host

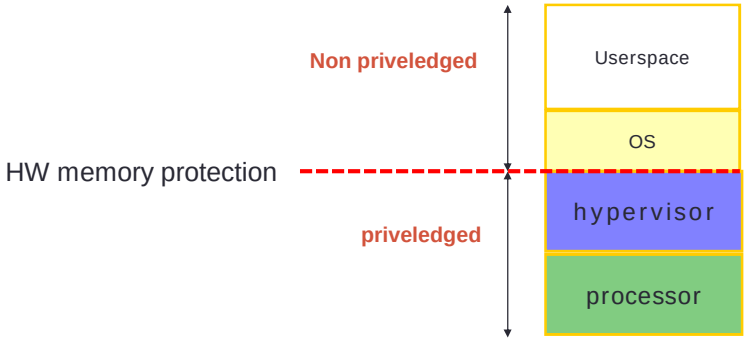


Full/Pure vs impure/para virtualization

- **Full/pure** virtualization: ensure that sensitive instructions are not executable within the virtual machine, but instead invoke the hypervisor: needs hardware support
- **Impure** virtualization: remove sensitive instructions from the virtual machine and replace them with virtualization code.

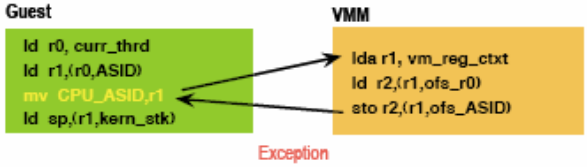
Hypervisors (and micro Kernels)

- Execute in priveledge mode
- Schedule the systems(OSs) that execute on it



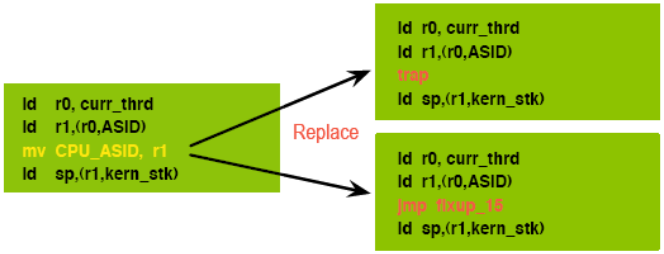
Pure Virtualization

- Most instructions are executed directly on the hardware.
- All senstive instructions are priveledged. They are trapped and instead executed by the hypervisor that runs in priveledged space (kernel mode, superuser mode, etc).
- Needs hardware support (Modern main CPUs have this, X86: AMD-V/Intel-VT)



Impure Virtualization

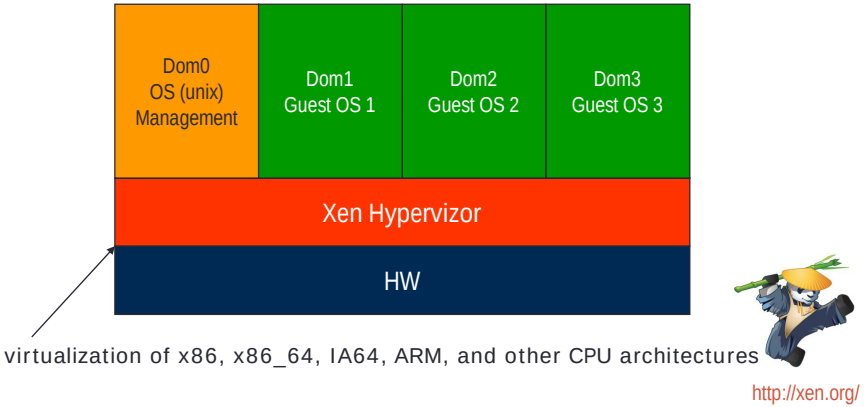
- Most instructions are executed directly on the hardware.
- All senstive instructions rewritten (e.g. during load time or during porting (para virtualization)): either trap to hypervisor or jump to a user-level emulation code



Hypervisors and micro-kernels

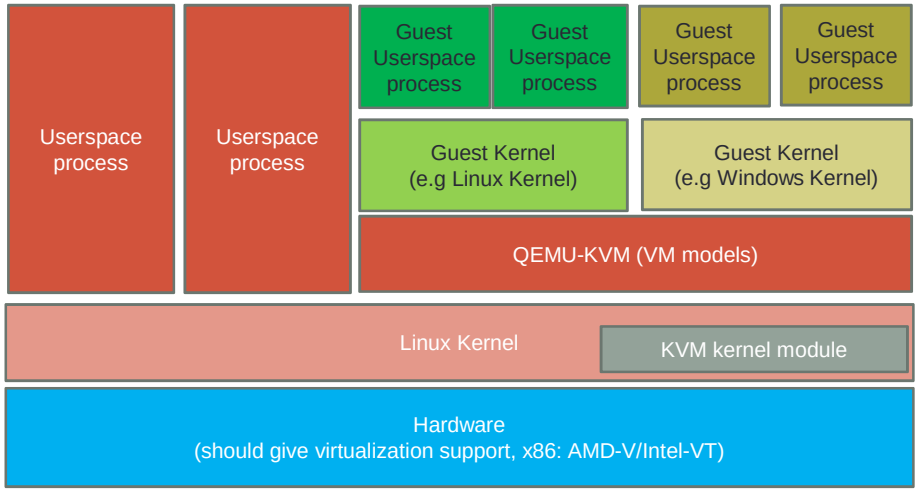
- Virtualization is done often through a hypervisors
  - Thin layer: e.g. Xen
  - Modifying OS kernel + HW emulation: e.g. KVM/qemu
- But can also done via micro-kernels
  - Interesting for small systems ? (However micro-kernels are common also as core part in large Oses)
  - E.g. L4

Xen



KVM/qemu

Kernel-based Virtual Machine



Example of a trusted microkernel: L4

- Used (with modifications) in various Qualcomm platform based phones. Available for Linux and Symbian. Used in some smart cards
- Academic work ongoing: to proof the kernel is correct
- Free to use
- Commercial: e.g. OpenKernel Labs

JAVA AS TRUSTED EXECUTION ENVIRONMENT

Example of Trusted Computing in SW

Outline

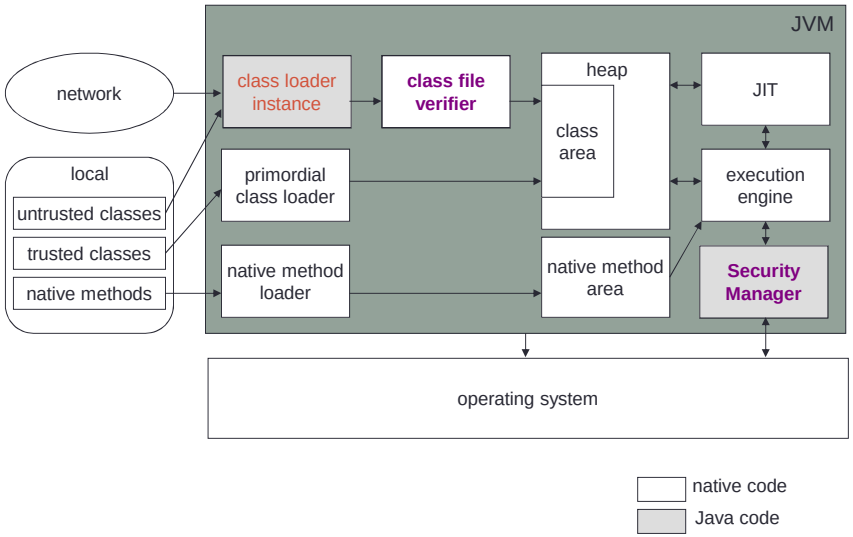
- components of Java
- Java security models
- main components of the Java security architecture
  - class loaders
  - byte code verification
  - the Security Manager

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

=Clearly Security relevant

The Java Virtual Machine (JVM)



JVM cont'd

- 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 cont'd

- 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
- the heap
  - memory used to store objects during execution
  - how objects are stored is implementation specific

JVM cont'd

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

JVM cont'd

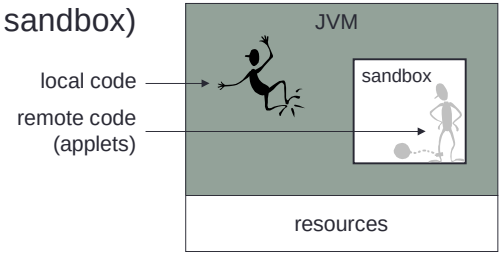
- Security Manager
  - enforces access control at run-time (e.g., prevents applets from reading or writing to the file system, accessing the network, printing, ...)
  - application developers can implement their own Security Manager
  - or use the policy based SM implementation provided by the "standard" JDK

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

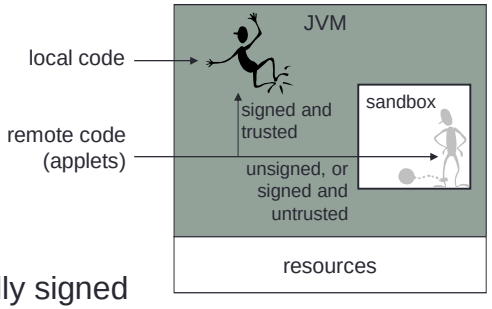
idea: limit the resources that can be accessed by applets (this creates an execution sandbox)



- 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

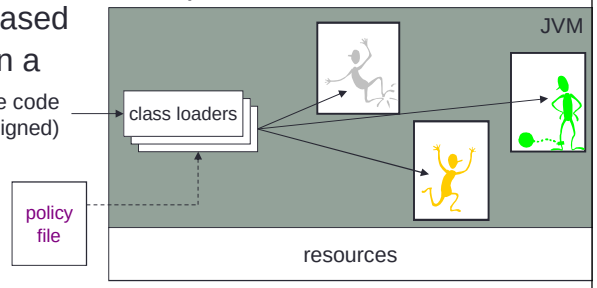
idea: applets that originate from a trusted source could be trusted



- 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 the policy file

```
grant CodeBase “http://java.sun.com”, SignedBy “Sun” {
    permission java.io.FilePermission “${user.home}${/}*”, “read, write”;
    permission java.net.SocketPermission “localhost:1024-”, “listen”;};
```


Java’s impact

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

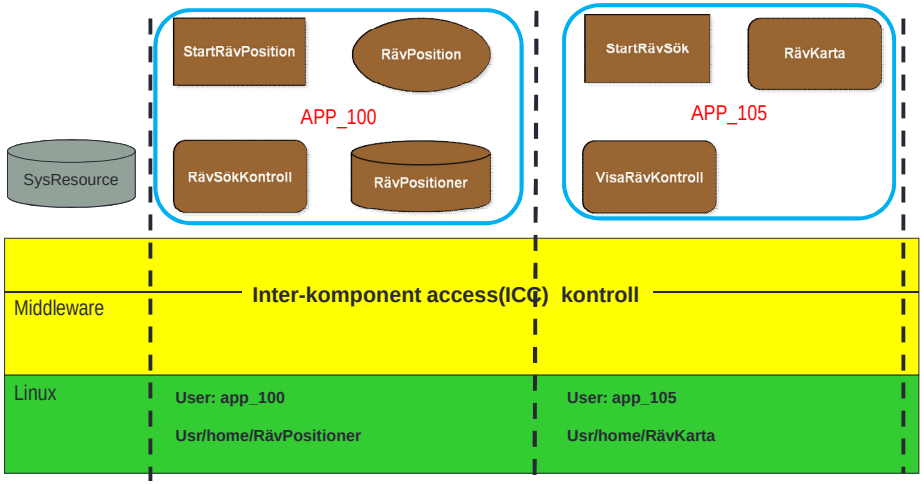
  E.g.

  - ActiveX
  - .Net
  - STIP
  - Android

Android

- Android exe env borrows many ideas from ordinary Java but in Android one has a totally different security architecture
  - E.g. digital signatures of apps are not used to verify origin but to have a proof that they originate from same issuer or not which affects how apps are isolated

Android: Linux access control also in ”Java” env



SELINUX

Trusted systems: trusted OS

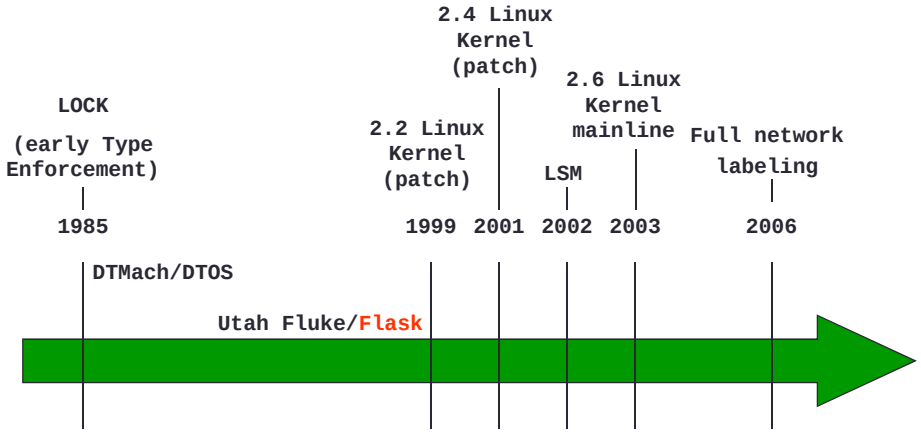
Example of a trusted OS: SELinux

Motivation

- Discretionary Access Control (DAC) in Linux provides not enough choices for controlling objects.
- 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).

SELinux Development History

Primarily developed by the US National Security Agency
SELinux implements Flask



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 DAC control goes first and then the policy is enforced.

➡ The specification of the policy (file) is the heart of the MAC system

Reference Policies

- Maintained by NSA and FC Mailing Lists
- Compiles into three versions
  - Strict, Targeted, MLS (Multi Level Security)
- Statistic:
  - Version .18
  - Object Classes 55
  - Common Permissions 3, Permission 205
  - Types 1589
  - allow 372755, auditallow 12, dontaudit 238663
  - type_transition 2657, type_change 68
  - roles 6, RBAC allow 6, role_transition 97, users 3
  - bools 70

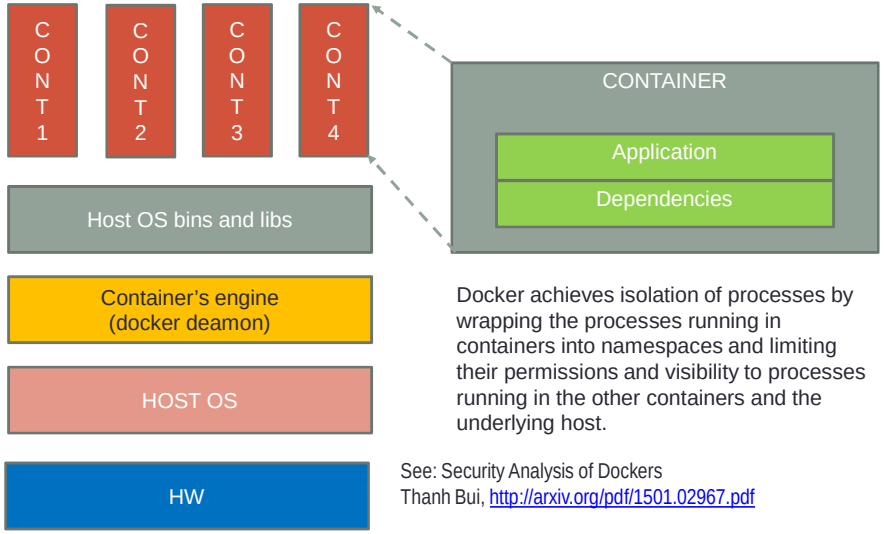
Time to play: SELinux distributions

- Fedora Core 3 and later
- Debian
- Gentoo
- SuSe
- SE-BSD
- SE-MACH

SELinux and Android

- Android 4.4 is adopting SELinux components
  - Android sandbox reinforced with SELinux. Android now uses SELinux in enforcing mode.

Container based compartmentalization



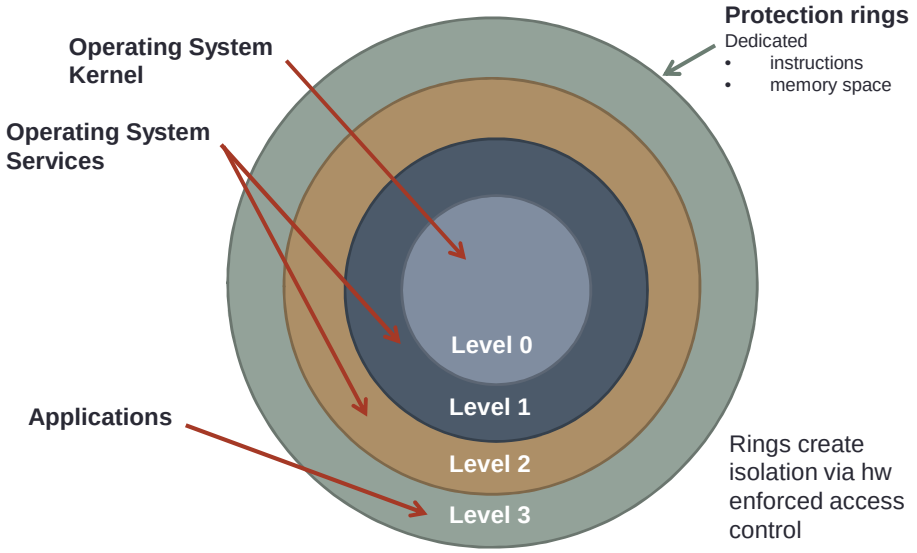
Docker achieves isolation of processes by wrapping the processes running in containers into namespaces and limiting their permissions and visibility to processes running in the other containers and the underlying host.

See: Security Analysis of Dockers
Thanh Bui, http://arxiv.org/pdf/1501.02967.pdf

THREE APPROACHES TO CPU SUPPORTED TRUSTED COMPUTING

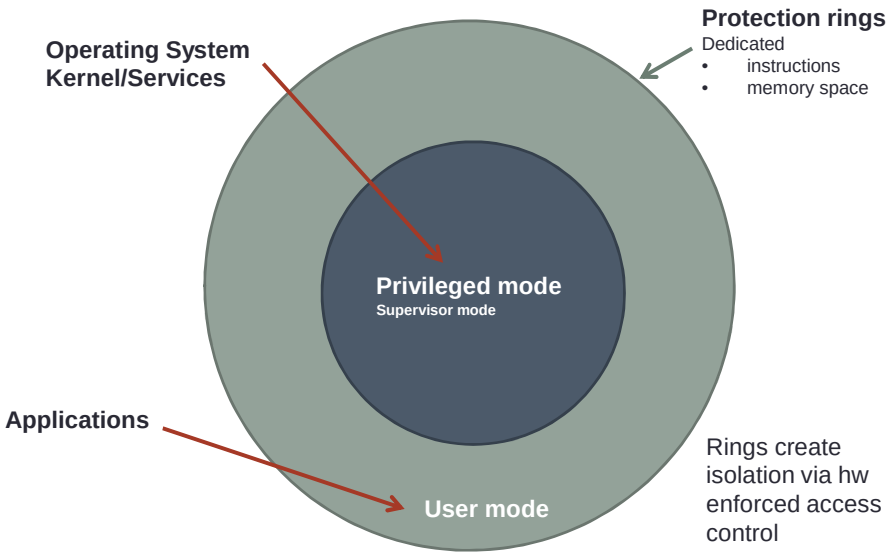
- ARM TRUSTZONE
- Intel SGX
- TCG TPM

Recall security ring architecture



Rings create isolation via hw enforced access control

ARM standard approach

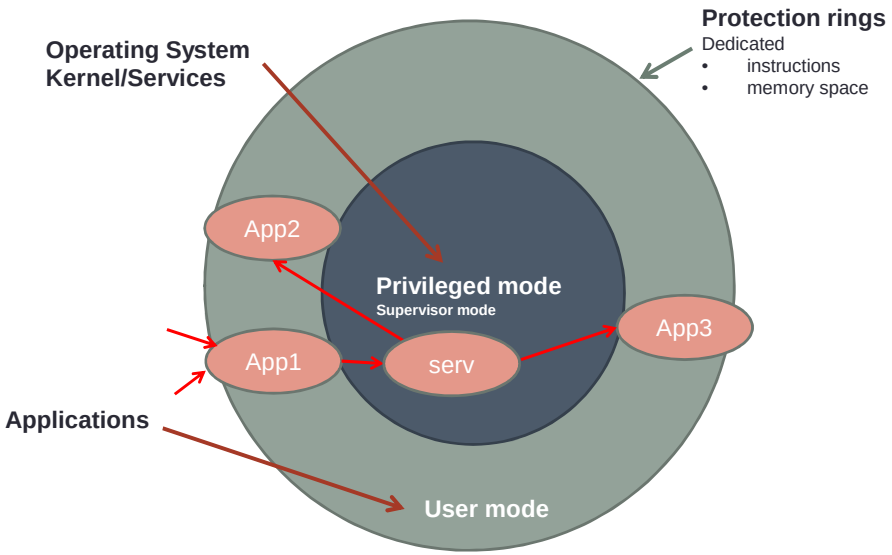


Separation of sensitive ops and data

- Since too much code in running in user space and even in the privileged space:

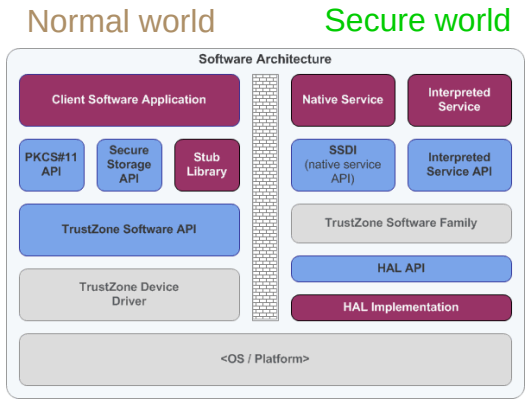
  Sensitive applications and data cannot be given good guarantees that other running code cannot tamper or get access.

Security problem for applications



ARM TrustZone

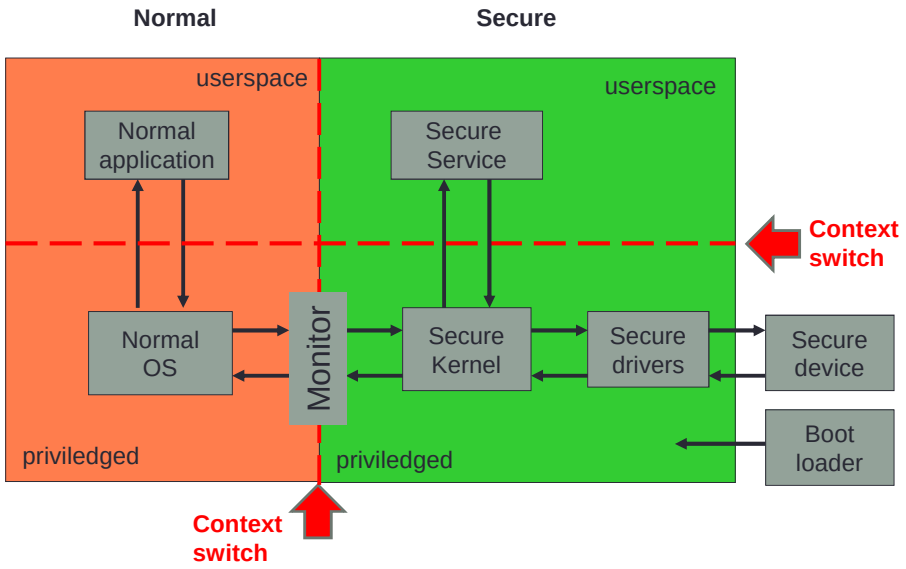
- A special mode of operation for the ARM11 processor
- Divides the SoC into “normal world” and “secure world”



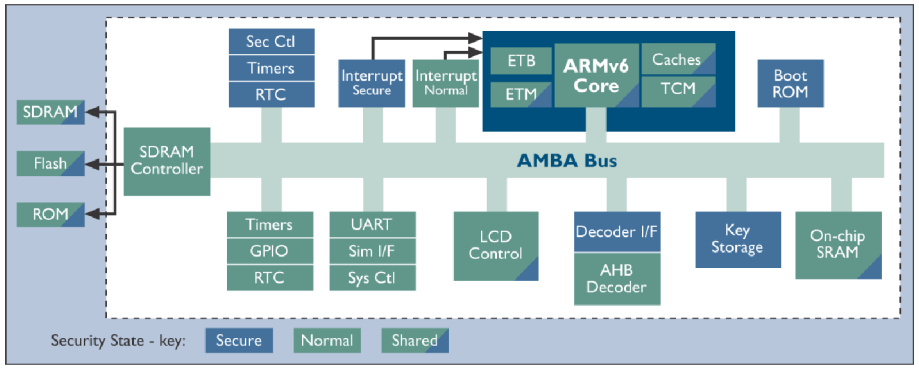
Basic idea

- Introduce an NS-bit
  - use this bit to ***tag*** secure data throughout system
    - Buses
    - cache
    - pages
- Monitor
  - manages the NS-bit
  - manages transition in & out of security mode
  - Small fixed API (so we can better check/verify the code)

Switching from Normal to Secure



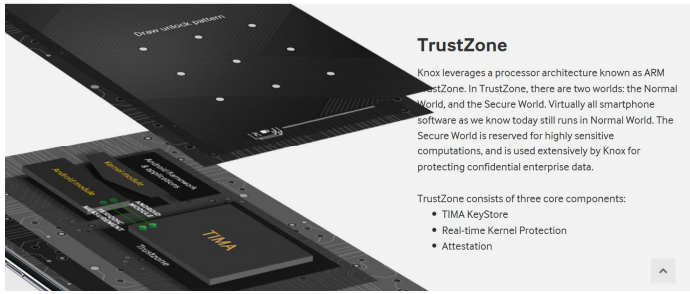
TrustZone uses Hardware features - Example System



TRustZone is used in many Android smartphone products:

TrustZone use

- Widespread in use in smartphones using Qualcomm and Samsung chipsets
- Forms a core of Samsung's KNOX solution
- https://www.samsungknox.com/en



Shortcomings of Trustzone

- Since the TZ system is not an isolated part on the ASIC it is practically impossible to get high EAL levels in the Common critera framework nor in the US NIST security levels for HW , FIPS 184-2, Security Requirements For Cryptographic Modules
- Isolation of multiple apps in secure world and handling of multiple threads ???
- Secure boot of system and thus the setup of the TZ system is not part of the TZ solution and must be addressed by the chip maker that used TZ in his ASICS and the final device vendor (e.g. Samsung, Sony)

SGX - ENCLAVES

Software Guard eXtensions

From:
Innovative Instructions and Software Model for Isolated Execution, Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos Rozas, Hisham Shafi, Vedvyas Shanbhogue and Uday Savagaonkar, Intel Corporation

Overview - SGX characterisics

- SGX in a new technology introduced in Intel chipsets
- SGX architecture includes 17 new instructions, new processor structures and a new mode of execution (additional extensions for servers are upcoming).
- These include loading an enclave into protected memory, access to resources via page table mappings, and scheduling the execution of enclave enabled application. Thus, system software still maintains control as to what resources an enclave can access.
- An application can be encapsulated by a single enclave or can be decomposed into smaller components, such that only security critical components are placed into an enclave.

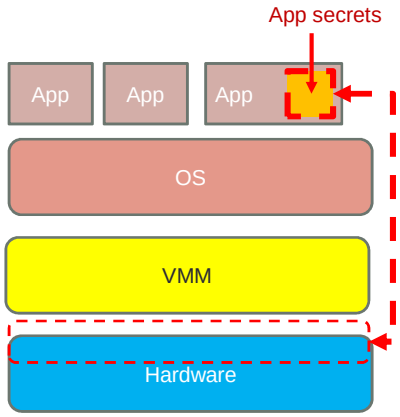
Enclaves

- Enclaves are isolated memory regions of code and data
- One part of physical memory (RAM) is reserved for enclaves and is called Enclave Page Cache (EPC)
- EPC memory is encrypted in the main memory (RAM)
- EPC is managed by OS/VMM
- Trusted hardware consists of the CPU Die only

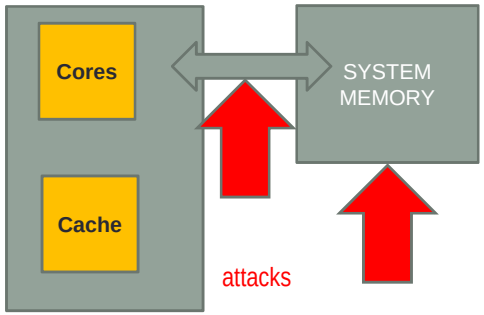
More info see this good overview paper:
Victor Costan and Srinivas Devadas, SGX explained:
https://eprint.iacr.org/2016/086.pdf

Reduced attack surface with SGX

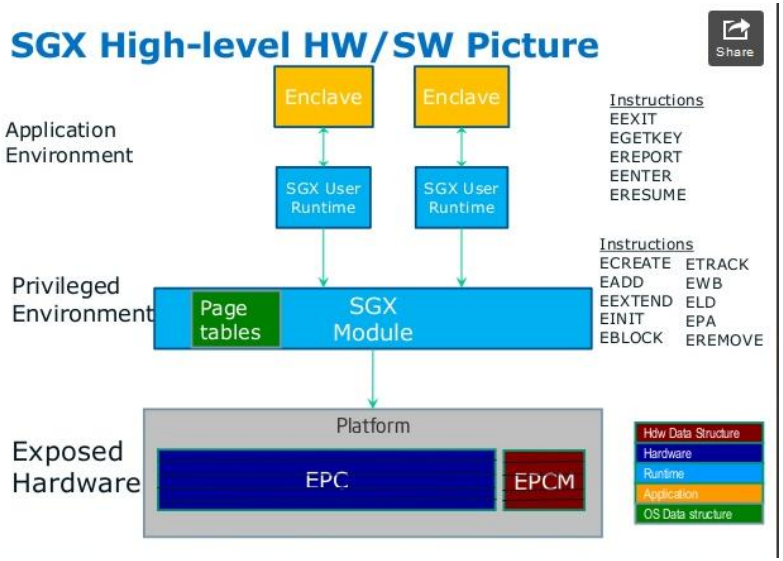
- Application gains ability to defend is own secrets
  - Smaller attack surface (App enclave+processor)
  - Malware that subverts OS or VMM, BIOS, drivers cannot steal app secrets



Protection against Memory Snooping

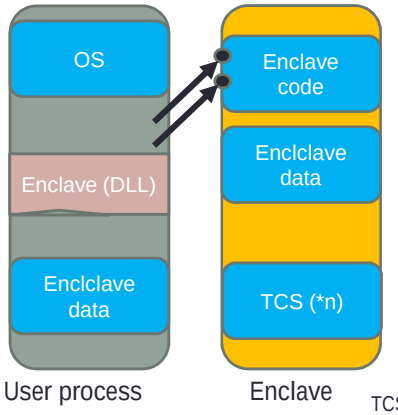


1. **Security perimeter is the CPU package boundary**
2. **Data and code unencrypted inside CPU package**
3. **Data and code outside CPU package is encrypted/integrity protected,**
4. **External memory reads and bus snoops tapping gives access to encrypted**



The Enclave Page Cache (EPC) is protected memory used to store enclave pages and SGX structures. The EPC is divided into 4KB chunks called an EPC page. The Enclave Page Cache Map (EPCM) is a protected structure used by the processor to track the contents of the EPC.

SGX Programming Environment



Protected execution environment embedded in a process

- With its own code and data
- Provide confidentiality and integrity protection
- With controlled entry points
- Support for multiple threads
- With full access to app memory
- Dedicated entry (call) points into enclave

TCS= Thread Control Structure

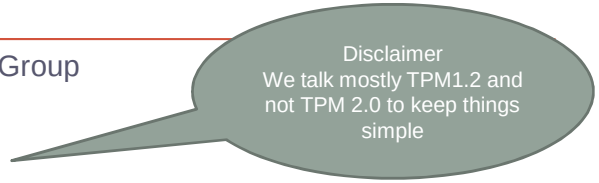
Attestation and Sealing

- SGX supports also attestation of enclaves and sealing of data to an enclave

  What this is will be explained in the TCG slides

TCG

Trusted Computing Group



- TCG Goals
- TPM Theory
- TPM practice
- TPM software stacks
- Intel TXT and Trustpools in OpenStack
- TPM and UEFI boot

TCG (http://www.trustedcomputinggroup.org/)

- Founded in 1999 by Compaq, HP, IBM, Intel and Microsoft
- Currently more than 200 members
- Implies changes to the hw platform
  - Extra for advanced devices: Trusted Platform Module (TPM)
  - Extra for mobile devices: TPM Mobile
  - Software changes: BIOS + OS

The three basic functions in a TCG trusted platform

- ***Protected Capabilities***

  Protected capabilities is a set of commands that grant the user issuing the command access to protected locations, memory (storage), registers, etc.

- ***Attestation***

  Attestation is the process of verifying the accuracy of information and the characteristics of the TPMs current state.

- ***Integrity (Measurement and Reporting)***

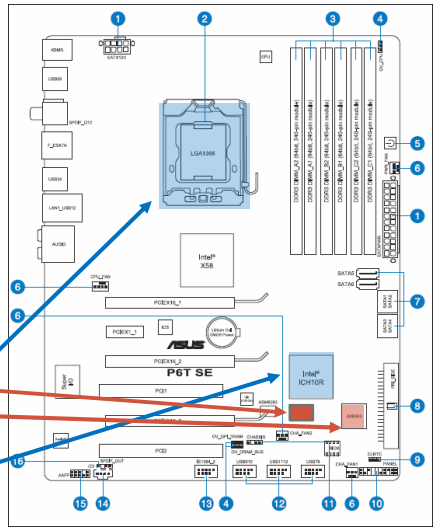
  Integrity measurement is the process of obtaining metrics of the platform characteristics and storing the information digest in a *Platform Configuration Register* (PCR). Integrity reporting is to attest the integrity measurements that are recorded in the PCR register.

TCG goal and impact on HW

- On a high level TCG wants to foster technology that promotes and defines and promote hardware-based root of trust, **a RoT**.
- How does this affect the HW then??

At least:
- We add TPM chip
- We modify BIOS

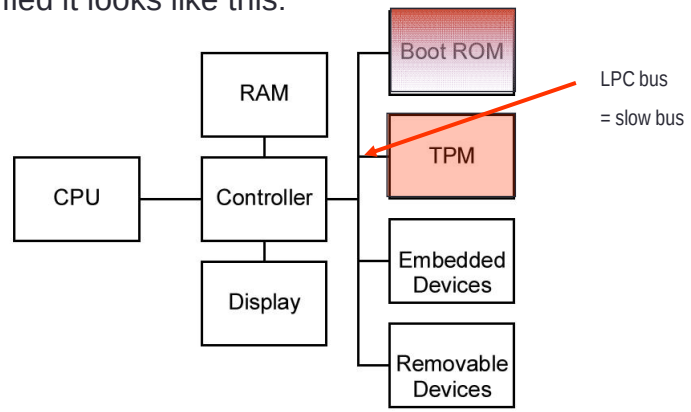
Later also
- Modify CPU
- Southbridge



TPM = Trusted Platform Module

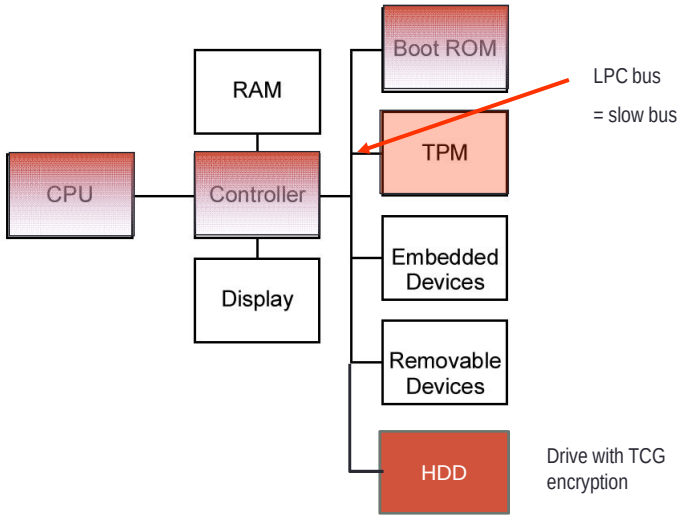
TCG Architecture (typically PC or Server)

So simplified it looks like this:

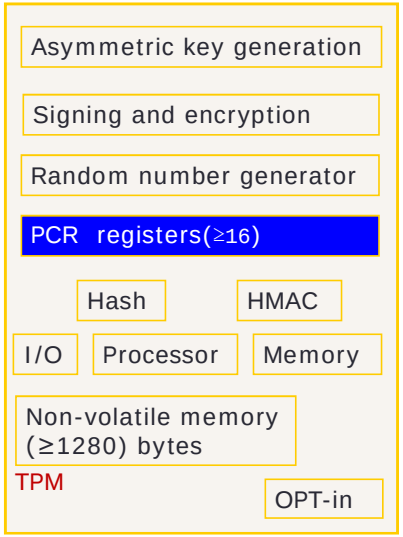


TCG Architecture (typically PC or Server)

And in a more complex system like this:



Trusted Platform Module (TPM 1.2)

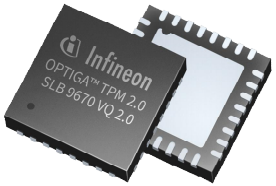


- Cryptographic operations
  - Hashing: SHA-1, HMAC
  - Random number generator
  - Asymmetric key generation: RSA (512, 1024, 2048)
  - Asymmetric encryption/ decryption: RSA
  - *Symmetric encryption/ decryption: DES, 3DES (AES)*
- PCR Registers (≥16)
- Tamper resistant (hash and key) storage
- Slave device (i.e. must be driven from outside)
- Opt-in: TPM state

Commercial TPM example - Infineon

The SLB9670 is a Trusted Platform Module and is based on advanced hardware security technology. This TPM implementation has achieved CC EAL4+ certification and serves as a basis for other TPM products and firmware upgrades. It is available in PG-VQFN-32-13 package. It supports an SPI interface with a transfer rate of up to 43 MHz.

- Compliant to TPM Main Specification, Family "2.0", Level 00, Revision 01.16
- SPI interface
- Meeting Intel TXT, Microsoft Windows and Google Chromebook certification criteria for successful platform qualification
- True Random Number Generator (TRNG)
- Full personalization with Endorsement Key (EK) and EK certificate
- 24 PCRs (SHA-1 or SHA-256)
- **7206 Byte free NV memory**
- Up to 3 loaded sessions (TPM_PT_HR_LOADED_MIN)
- Up to 64 active sessions (TPM_PT_ACTIVE_SESSIONS_MAX)
- Up to 3 loaded transient Objects (TPM_PT_HR_TRANSIENT_MIN)
- Up to 7 loaded persistent Objects (TPM_PT_HR_PERSISTENT_MIN)
- Up to 8 NV counters
- Up to 1 kByte for command parameters and response parameters
- Up to 768 Byte for NV read or NV write
- 1280 Byte I/O buffer
- Built-in support by Linux Kernel

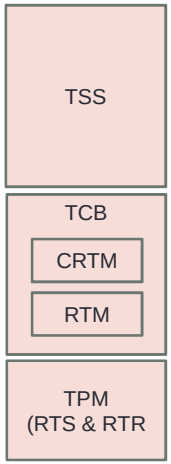


16 bit CPU

Asymm crypto	Symm. Crypto
ECC	AES
ECC BN-256	HMAC
ECC NIST P-256	SHA-1
ECC256	SHA-256
ECDH	
RSA1024	
RSA2048	

Main system components in a PC
with a TPM

- Trusted Compute Base (TCB)
  - Root of Trust for Measurement (RTM)
    - The Core Root of Trust for Measurement (CRTM)
    - Static and Dynamic RTM
- Root of Trust for Storage (RTS)
- Root of Trust for Reporting (RTR)
- The TCG SW Stack (TSS)



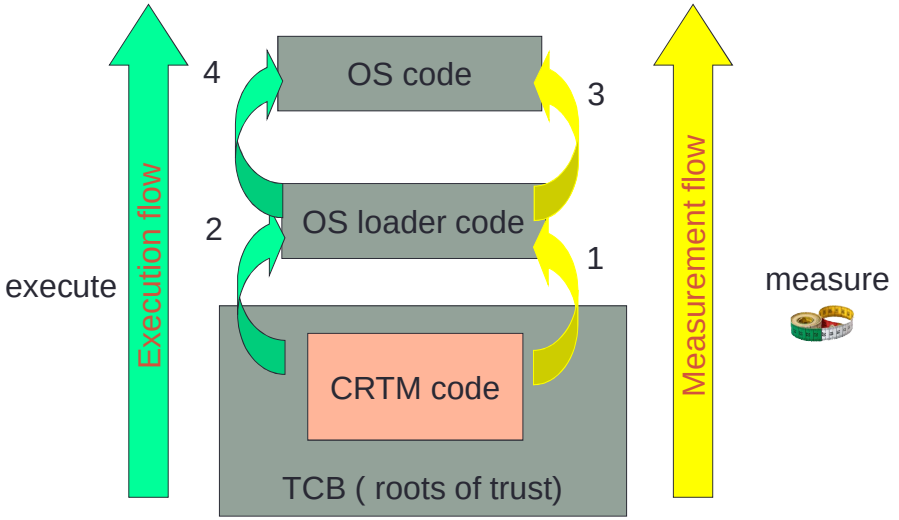
The CRTM, S-RTM and D-RTM

- The **CRTM** is the *a priori trusted code* that is part of the platform credential. On PC platform the CRTM = BIOS boot block
- In the **Static RTM Model**, this MUST be the very first piece of code executed on power on or upon reset of the server or complete physical hardware environment.
  - Note: at startup the CRTM will check for physical presence of the TPM
  - REMEMBER: TPM is not the root-of-trust but trust starts with the CRTM
- In the **Dynamic RTM model** the hardware is designed to support that while running a trusted execution thread can be started:
  - Intel call their implementation (Intel) Trusted eXecution Technology
  - AMD: DRTM instruction, SKINIT

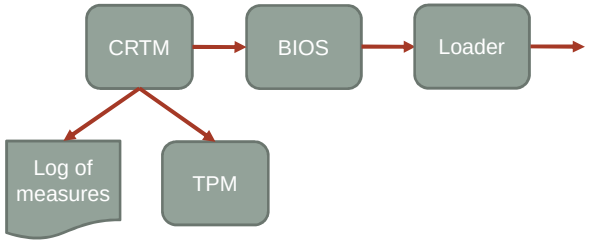
Trusted Compute Base (TCB)

- The trusted computing base of a computer system is the set of all hardware, firmware, and/or software components that are critical to the systems security. Any malfunction in the TCB due to bugs and/or vulnerabilities occurring might compromise the security of the entire system.
- The RTM will be part of the TCB but usually the TCB is much larger in an OS. Yet one strives at keeping the TCB small. Why ?

The basic idea for secure boot



Secure bootstrap in PC's



TPM theory- overview

We will look at

- The TPM functions themselves
  - TPM key types
  - TPM Key hierarchy
  - TPM command sessions
  - Attestation and AK keys
  - Binding and Sealing
- TPM secure storage
- Secure boot
- Software stack for TPM
- TPM in windows

TPM Versions

- Most TPM in use are TPM version 1.2
  - There was a TPM version 1.0 but we can today forget about that version.
- There is a new TPM version 2.0 rapidly being deployed
  - TPM2.0 Differs in many aspects from the TPM1.2
  - TPM2.0 has better algorithm agility
  - TPM2.0 allows flexible definition of policies (rules) to use objects (e.g. keys)
  - It has three key hierarchies:
    - platform, endorsement, and storage
    - The flexibility makes the TPM2.0 more useful but could also lead to fragmentation in its use.

Opt-In

- **The TCG policy is that the TPM should be shipped "in the state that the customer desires".**
- Thus, **it is up to the user to opt-in** to use the TPM. Users are not forced to use trusted computing, they opt-in if they choose to do so by taking ownership of the device.
- The function of the Opt-In component is to provide mechanisms and protection to maintain the TPM state via the state of the corresponding flags

TPM Counters

TPM has monotonic counters (at least 4)

- Increment rate: Every 5 secs for at least 7 years (so at least 26 bit counter needed)
- Can be used for anti-roll back protection
  - (old versions can be blocked from loading)

TPM 1.2 Key types

- Endorsement Key (EK)
- Storage Root Key (SRK)
- Attestation Identity Keys (AIK)
  - sign data from the TPM. A TPM can have many identities!
- Storage: encrypt data, including other keys, (SRK is a special storage key)
- Signing: key only for signing
- Binding: decrypt data (usually from remote platforms)
- Certified Migration Key (CMK), is of one above type but taged as migratable
- Legacy: signing or encryption (compatible with TPM v1)

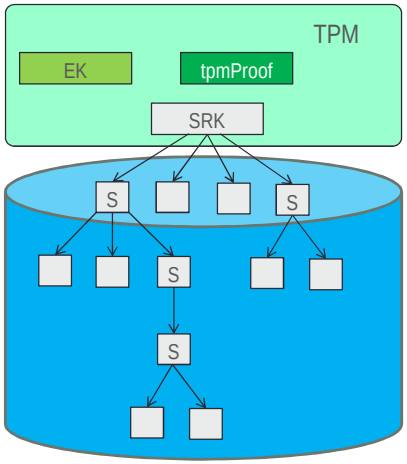
Main keys – always remain in TPM

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 for implementing encrypted storage
   - Created after running
     TPM_TakeOwnership(OwnerPassword, …)
   - Can be cleared later with TPM_ForceClear from BIOS
3. OwnerPwd(password) (160 bits) and persistent flags

Private EK, SRK, and OwnerPwd never leave the TPM

Key Hierarchy in TPM 1.2

- tpmProof can be used to tie keys to the TPM
- Each key has 160-bit authData
- EK is static throughout the lifetime of the TPM
  - Inserted by manufacturer (never leaves the TPM (fixed handle))
  - Comes with EK certificate stating it is a genuine EK
- **Intermediate keys** are storage keys
  - SRK is the root (never leaves the TPM (fixed handle))
- **Leaves** are special purpose keys
  - Binding keys
  - Sealing keys
  - Attestation identity keys (AIK)
  - Signing keys

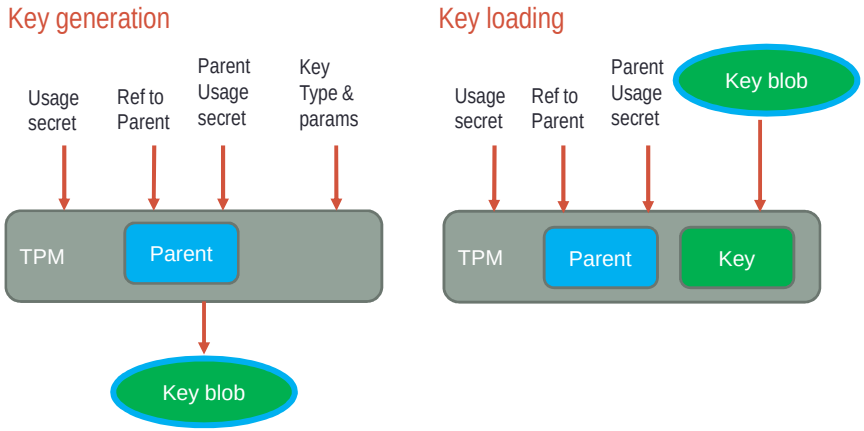


Stored outside TPM, e.g., hard-disk

Question: How to prove that SRK inside TPM ?

Key generation & loading

- When a key (except EK, SRK) is generated we get a key blob



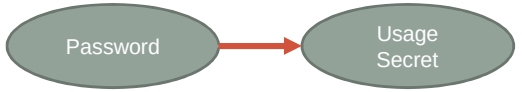
TakeOwnership

- The TakeOwnership results in
  - a (re)computation of the SRK private and public key
  - The usage secret for SRK is set
  - The owner secret is set
  - A new **tpmProof** value is set which is a random value kept secret inside the tpm
  - Future reading of pubEK will require knowledge of owner secret

The tpmProof is used in several TPM functions to give a binding which is a) unique for the given TPM and b) unique for the current active TPM. We will example of this later.

Passwords and Secrets

- When taking ownership an owner(ship) secret is set that is needed later for certain TPM commands
- Each key except EK has a usage secret which must be presented when certain operations with the key is to be performed. (one could regard the owner secret as the usage secret of EK).
- To each secret is connected a password from which it could be derived.

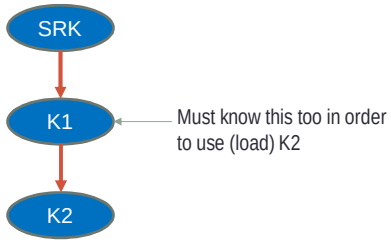


Endorsement key

- Is a very special key since it stays the same during the lifetime. This gives privacy concern due to linkabilty to the user when the EK is used.
- To reduce the risk of EK being used in an improper way even the use of EK is limited
  - Basically it allows only EK for encryption. So signing type of operations are not allowed.
  - We come back to the use of EK later

Warning

- All keys “below” SRK are lost for ever if a new SRK is generated by a re-takeOwnership command.
- To be able to load a key into the TPM by importing its key blob one must have posses the parent keys between SRK and the key



TPM Protected storage - data

- The TPM
  - can wrap data
  - “Sealing”: binds data to a certain value of the PCR. Then the TPM can only decrypt (unseal) if the PCR value(s) is the same as when encryption happened (seal)
- Management: migration (=duplication in TPM2.0), backup

TPM sessions

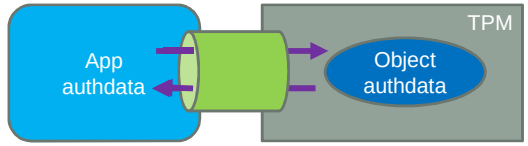
In order to protect the communication between the application and the TPM most commands support protection mechanisms.

- Use of the authdata of an object

Since the interaction of an application with the TPM may involve several commands that consecutively have to be performed the TPM supports sessions

- TPM1.2 supports three types of session
  - **OIAP**: Object Independent Authorization Protocol which creates a session that can manipulate any object, but works only for certain command
  - **OSAP**: Object Specific Authorization Protocol which creates a session that manipulates a specific object specified when the session is set up.
  - **DSAP**: Delegate-specific Authorization Protocol. Similarly to OSAP sessions, DSAP sessions are restricted to a single object.

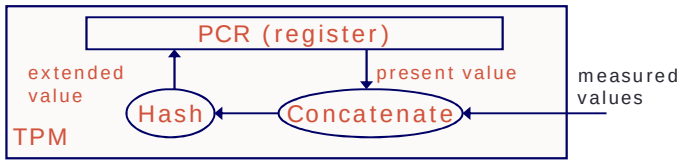
TPM2.0 will do this differently



PCRs & PCR Extend

- PCR is a register that contains a SHA1 hash and is used to accumulate “measurements”
  - Can be read from the outside
  - Are reset to zero at power up

We return to this later



PCR[n]←SHA1(PCR[n] || data(20 bytes))

In TPM1.2, other hashes in TPM2.0

Example: Intel TXT use of PCRs

The static core root of trust measurement (CRTM) as well as the measurement of the BIOS Trusted Computing Base (TCB).

- PCR0 – CRTM, BIOS code, and Host Platform Extensions[a]
- PCR1 – Host Platform Configuration
- PCR2 – Option ROM Code
- PCR3 – Option ROM Configuration and Data
- PCR4 – IPL (Initial Program Loader) Code (usually the Master Boot Record – MBR)
- PCR5 – IPL Code Configuration and Data (for use by the IPL Code)
- PCR6 – State Transition and Wake Events
- PCR7 – Host Platform Manufacturer Control

The dynamic PCR measurements

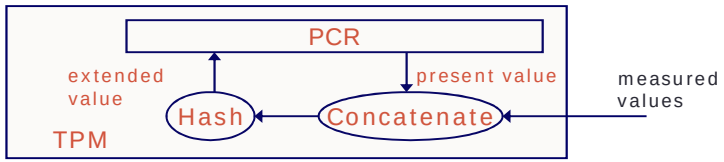
- PCR17 – DRTM and launch control policy
- PCR18 – Trusted OS start-up code (MLE)
- PCR19 – Trusted OS (for example OS configuration)
- PCR20 – Trusted OS (for example OS Kernel and other code)
- PCR21 – as defined by the Trusted OS
- PCR22 – as defined by the Trusted OS

Secure bootstrap: PCRs and measuring

- Extending a PCR

  PCR_Extend(n,data): PCR[n]←SHA1(PCR[n] || data*)
- Use when booting:
  1. Reset PCRs
  2. PCR_Extend(n,<Bios Code>)
  3. PCR_Extend(n,<MBR>)
  4. etc

* data must be 20 bytes: Hash(data)



Protected Storage: SEAL

Main Step: Encrypt data using RSA key on TPM

TPM_Seal

- INPUT:
  - KeyHandle: which TPM key to encrypt with
  - KeyAuth: password for using key with id `KeyHandle’
  - PcrList: list with indices J and PCR[i] $i \in J$ to be embedded in output sealedData
  - data: at most 256 bytes (2048 bits)
    (typically used to encrypt symmetric key (e.g. AES))
- OUTPUT:
  - sealedData RSA encrypted data (and PcrList)

MAC protected list by TPM_Seal

- SEAL to set of PCRs J=(i1,…,in) using key
- TPM_Seal(J , data) →
  (C,MAC(SRK,((i1, PCR[i1]), (i2, PCR[i2]), ...))

Ex: C=RSA(SK,data) and SK is storage key

Protected Storage: UNSEAL

Main Step: Decrypt data using RSA key on TPM

TPM_UnSeal

- INPUT:
  - KeyHandle: which TPM key to decrypt with
  - KeyAuth: password for using key with id `KeyHandle'
  - sealedData: RSA decrypted data and PcrList
- OUTPUT: IF and Only IF
  $\forall i \in J$ current PCR[i] = PCR in MAC protected list
  - data:

Example: Use case of a sealed key

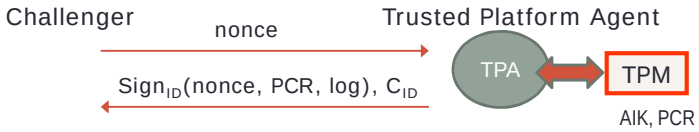
- **Problem**: We want at machine start read out a secret that decrypts some site specific TLS client certificates (containing secret PKI keys) and puts it into RAM but when the OS kicks-in the secret should not be recoverable anymore.
- **Solution:** Seal the secret to a PCR that "measures" the machine state during boot. When the boot comes to the correct point its TPM can do unseal and we load the RAM with our client cert. Then we erase the secret, verify the OS code, update the PCR and start the OS. Now we no longer can unseal the secret and is protected against wrong doings by the OS and the software that is running on the OS.

Remote measurement of TPM state

- This is called attestation
- The goal is to securely determine remotely the state of a machine. The approach is to record the state of the machine in the PCRs and to have a mechanism by which one can interrogate a TPM to get the PCR values.

TCG: (remote) Attestation

- Integrity reporting: report the value of the PCR
- Challenge-response protocol



- AIK keys used for signing are TPM Identities (pseudonyms)
  - Use different identity (AIK) for every challenger
  - CID is a certificate proving the AIK is trustworthy

TPM_Quote (prepare)

The TPM_Quote is the command that does the attestation

CreateIdentity

- A new Attestation Identity Key (AIK) is generated
- The AIK is linked to the TPM/EK by issuing a certificate that is has no link to EK (for privacy reasons)
- This certificate is issues by a special procedure in which the AIK certificate issuer uses the knowledge of genuine EKs. (we return to that shortly)

TPM_Quote

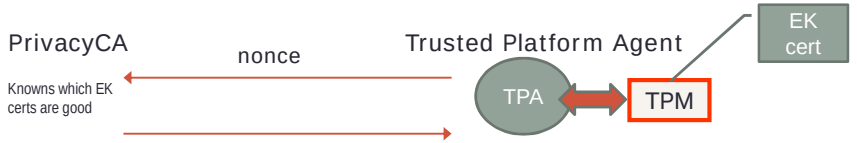
The TPM_Quote is the command that does the attestation

TPM_Quote

- INPUT:
  - KeyHandle: which TPM key to decrypt with
  - KeyAuth: password for using key with id `KeyHandle'
  - PCR list: the PCR to quote
  - ExternalData: 20 byte value
    - challenge to prevent replay attacks
    - Hash of a challenge and userdata to be included in the quote signature
- OUTPUT:
  - The signature of the quoted data.

Trusting the AIK – privacy CA

- How can we trust the AIK? Is it really an AIK inside a real TPM or are do we deal with an emulator.
- We cannot sign the (pub)AIK with EK´. There is no command for that

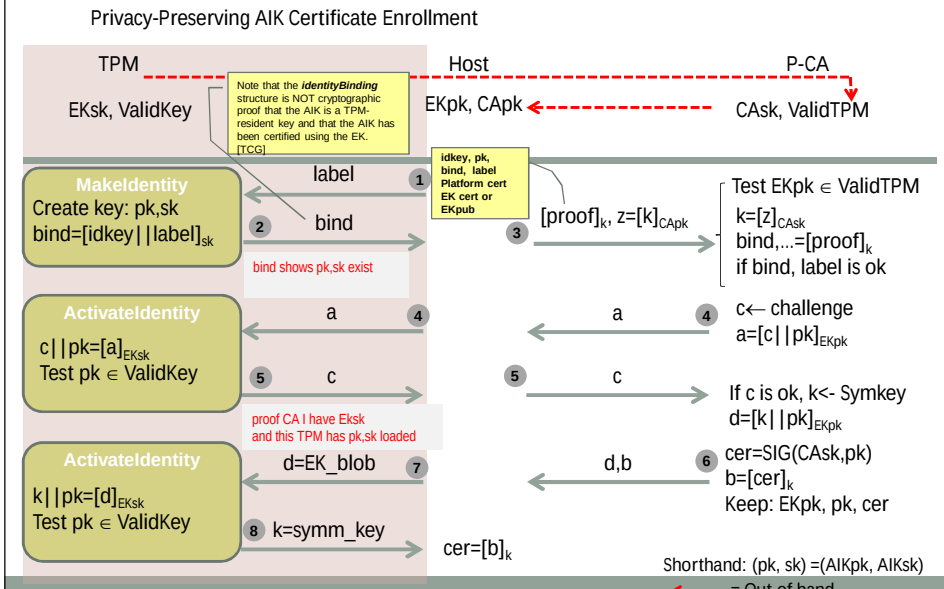


- We extract the pubAIK and then ask for a certificate
- The issuer encrypts the certificate with a key that is encrypted by pubEK so only the TPM with the right EK can recover the certificate. The TPA sends a proof that it had the correct TPM (the one with the correct EK)

AIKs: privacyCA and concerns

- **Initially a privacyCA was architectured through which AIKs could be bind to a specific TPM.**
  - Yet there were concerns that the binding compromises anonymity and therefore TCG has implemented a more advanced attestation method based on zero-knowledge techniques: Direct Anonymous Attestation (DAA).
  - DAA is a cryptographic protocol which enables the remote authentication of a trusted platform yet preserving the user's privacy.

TCG Privacy-CA (sketch)

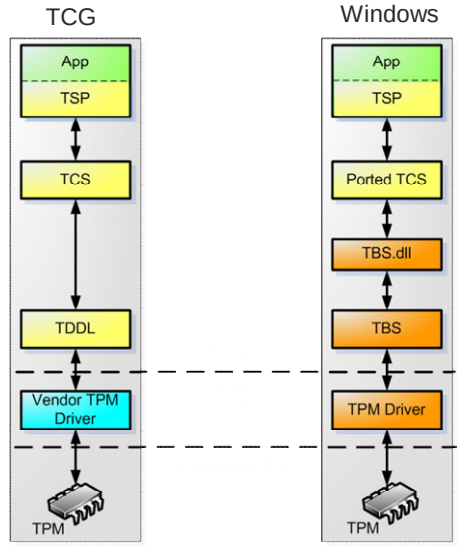


TPM use in practice

- TPM software stacks (skip)
- Secure boot
- Intel TXT and Trustpools in OpenStack
- TPM in MS Windows (for own reading)
- TPM and UEFI boot

TCG Stack vs. TPM Services Stack

- TPM applications use the TCG Service Provider (TSP) interfaces
- The TCG Core Services component (TCS) is ported to communicate with the TBS instead of the TCG Device Driver Layer (TDDL)



TPM1.2 experimenting

- You can experiment with TPM by using a TPM emulator and the TrouSerS (TSS) software stack
- TPM Emulator:
  - http://tpm-emulator.berlios.de/
- TSS Stack:
  - http://trousers.sourceforge.net/
  - Above link contains many other useful code
- PrivacyCA: (for use with AIKs)
  - http://www.privacyca.com/

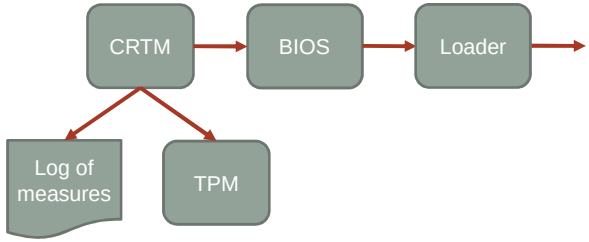
TPM2.0 experimenting

- You can experiment with TPM 2.0 by using a TPM emulator (from Microsoft or IBM) and the IBM software stack
- http://sourceforge.net/projects/ibmtpm20tss/

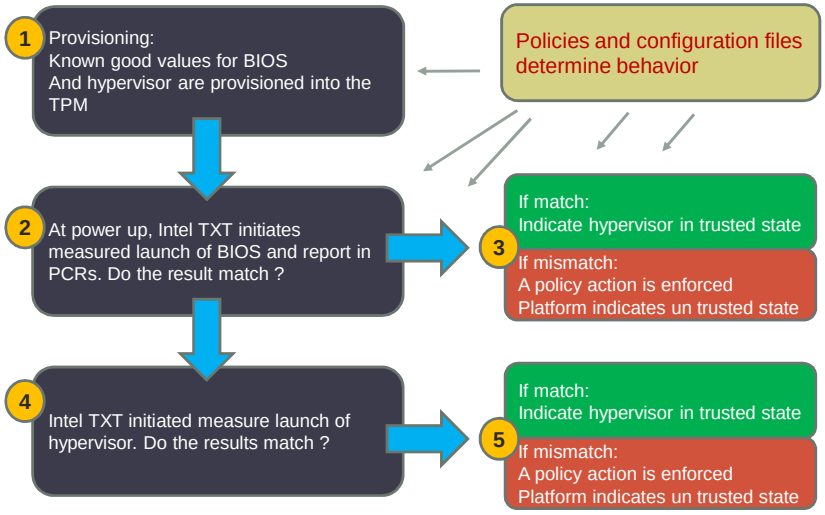
Secure bootstrap: secure vs authenticated boot

Two methods of booting

- **Secure Boot**: boot can be halted when check fails
- **Authenticated** : just reporting
  Synonyms in use : measured boot or trusted boot



Intel TXT for system with hypervisor



Software measured and verified

Platform trustworthiness is reported

Intel TXT components

- The CPU extensions
  - New instruction: GETSEC
    - with leave functions SENTER and ENTERACCS
    - Halt execution of cores and calls SINT ACM,
- The HW
  - ACM modules.

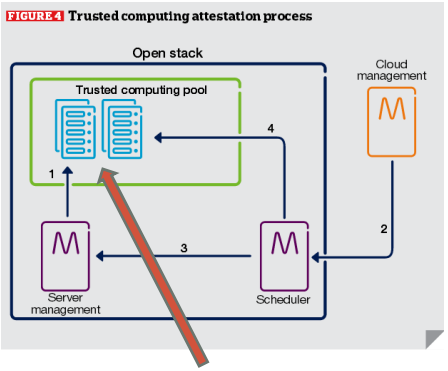
  Special signed sw modules by HW manufacturer that execute at highest security level and execute in special separate secure memory

  BIOS ACM: code that measures BIOS +init

  SINT ACM: code that is part of the DRTM for the secure init/launch

Trusted Computing Pools -idea

1. Server checks via remote attestation the compute resources and marks those that ok as trusted
2. Cloud manager initiated Virtual Machine (VM)
3. OpenStack scheduler checks server for trusted compute resource
4. VM is launched on trusted resource



See
Trusted computing for infrastructures
Trusted Computing Pools

Intel TXT warning

- It is very complex
  - So likely we can see that people find design errors and implementation flaws
- The first attack was the reset attack which is mitigated in the current releases of TXT

TPM in windows

- TPM support in Windows 7
- Increased TPM support in Windows 8 & 10
- Use case
  - Bitlocker (file encryption)
  - Secure boot (UEFI boot)

Bitlocker (1/3)

- AES in CBC mode with Elephant diffuser
- Key escrow via Active Directory
- Three different modes are supported
  - **Transparent operation mode (with TPM)**: The key used for the disk encryption is sealed by the TPM and will only be released to the OS loader code if the early boot successfully verified. The boot components of BitLocker do a Static Root of Trust Measurement.
  - **User authentication mode (with TPM)**: This mode requires that the user provide some authentication to the pre-boot environment in order to be able to boot the OS. Two authentication modes are supported: a pre-boot *PIN* entered by the user, or a *USB key*.
  - **USB Key Mode**: The user must insert a USB device that contains a startup key into the computer to be able to boot the protected OS.

Bitlocker (2/3)

- Combinations
  - TPM only
  - TPM + PIN
  - TPM + PIN + USB Key
  - TPM + USB Key
  - USB Key
- At least two NTFS volumes needed: one for the OS and another unencrypted to boot the OS from.

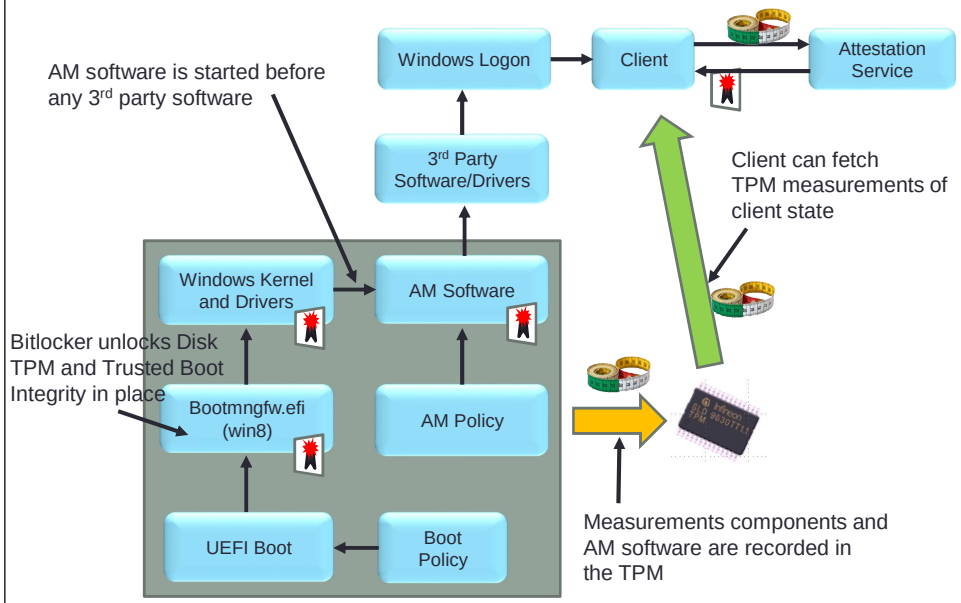
Bitlocker (3/3)

- BitLocker encryption is transparent to OS
  - Bitlocker decrypts on-disk files before the OS has loaded. Therefore, all file operations occur from the OS perspective as if there is no encryption.
  - Protection of the files from processes/users within the operating system can only be performed using encryption software that operates within Windows, such as Encrypting File System.

UEFI secure boot

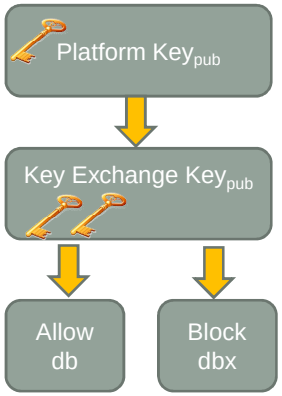
- UEFI secure boot and role of TPM
- OEM (manufacturer's) role

UEFi Trusted Boot Architecture



UEFi Secure Boot Keys

- Platform Key (PK)
  - Only one
  - Allows modification of KEKs
- Key Exchange Key(KEK)
  - Can be multiple
  - Allows modification of db and dbx
- Authorized Database(db)
  - CA, key, or image hash to allow
- Forbidden Database(dbx)
  - CA, key, or image hash to block



Keys required for Secure Boot

Key/db name	variable	Owner	Info
PKpub	PK	OEM	Must be RSA 2048 or stronger
Microsoft KEK CA	KEK	Microsoft	Allows updates to db and dbx
Microsoft Windows Production CA	db	Microsoft	This CA in the allowed signature database (db) allows Windows 8 to boot
Forbidden Signature database	dbx	Microsoft	List of bad/compromised keys, CAs images from Microsoft

+ Required for secure firmware updates

Key/db name	Owner	Info
Secure firmware update key	OEM	Should differ from PK, Must be RSA 2048 or stronger

Optional Keys for Secure Boot

+ Recommended for non WinRT Systems

Key/db name	variable	Owner	Info
Microsoft UEFI driver signing CA	db	Microsoft	Microsoft signer of 3rd party UEFI binaries via DevCenter program

+ Optional for Customization

Key/db name	variable	Owner	Info
OEM or 3rd party KEKpub	KEK	OEM/3rdP	Allows db/dbx updates, e.g. for alternate OS or Trusted 3rd party
OEM or 3rd party CA	db	OEM/3rdP	Allows 3rd party OS or drivers signed by Trusted 3rd party
Image Hashes	db	OEM	Hashes of images on PC that are allowed to execute even if not signed,
Forbidden Signature Database	dbx	OEM/3rdP	List of bad/compromised keys, CAs images from OEM or partner

UEFi secure boot and TPM

- Observe that actually the TPM is not needed for secure boot if one skips the requirement to support attestation. (one basically has no secrets to protect then).

http://technet.microsoft.com/en-us/library/hh824987.aspx

▲ Manufacturing Requirements

Secure Boot requires a computer that meets the UEFI Specifications Version 2.3.1, Errata C or higher.

Secure Boot is supported for UEFI Class 2 and Class 3 computers. For UEFI Class 2 computers, when Secure Boot is enabled, the compatibility support module (CSM) must be disabled so that the computer can only boot authorized, UEFI-based operating systems.

Secure Boot does not require a Trusted Platform Module (TPM).

To enable kernel-mode debugging, enable TESTSIGNING, or to disable NX, you must disable Secure Boot. For more info, see Windows 8 Secure Boot Key Creation and Management Guidance.

OEM's role

- Thus the OEM generates and own the PK secret key.
  - Basically the OEM can decide what can be loaded/booted is defined in the boot policy
- However, Microsoft, can demand the OEM boot policy to comply with Microsoft requirements if the OEM want to run Microsoft software (say Windows 8)

UEFi checklist for OEMs

As explained by Microsoft

- ❑ Define your security strategy
  - ❑ Identify roles
  - ❑ Procure server and hardware for key management
    - Recommended solution – network or standalone HSM
    - Consider whether you will need one or several HSM's for high availability and also your key back up strategy
  - ❑ Set policy for how frequently will you be rekeying keys
  - ❑ Have a contingency plan for Secure Boot Key compromise
  - ❑ Identify how many PK and other keys will you be generating
- ❑ Use HSM to pre-generate secure boot related keys and certificates
- ❑ Get the Microsoft KEK and other Secure Boot related keys and certificates
- ❑ Sign UEFI drivers
- ❑ Update firmware with Secure Boot keys based on the system type
- ❑ Run tests including WHCK Secure Boot tests
- ❑ Deploy > Refine > Deploy > Refine…

Criticism

References

- Microsoft Connect http://connect.microsoft.com/
- MSDN: http://msdn.microsoft.com/
  - Search on keyword “Secure Boot”
- http://www.microsoft.com/security
- UEFI 2.3.1. Specification errata C: http://www.uefi.org
- Tianocore: http://www.tianocore.sourceforge.net
- UEFI and Windows: http://msdn.microsoft.com/en-us/windows/hardware/gg463149
- Beyond BIOS: http://click.intel.com/beyond-bios-2nd-edition.html

TCG/TPM Issues

- Current TPM 1.2 cannot be remotely managed.
  - IT-departments do not like this. Hard to use
- Integration of TPM in a product is not the same for different computer vendors (even true for PC/laptops)
- Why trust a US standard ?

Chinese TPM

- Chinese authorities: import control on equipment which uses crypto. Permission is needed.
- There is a Chinese TPM called TCM which has certain approaches in common with TPM v 2.

Special trusted computing devices

- Secure Cryptoprocessors
  - Dedicated microprocessor system with physical protection features
    - Tamper-detecting and tamper-evident containment.
    - Automatic zeroization of secrets in the event of tampering.
    - Chain of trust boot-loader which authenticates the operating system before loading it.
    - Chain of trust operating system which authenticates application software before loading it.
    - Hardware-based capability registers, implementing a one-way privilege separation model.
    - Possibly battery backup
  - Smart cards
  - **History:** IBM 4758 PCI Cryptographic processor (PKCS#11 interface)

    Attacked 2001 by PhD students at Cambridge University

    replaced by IBM 4764
- Security Modules:
  - like Secure cryptoprocessors but more capable as system.

HSM
(Hardware (or Host) Security Modules)

- Special Computers with high-grade protection with purpose to to store critical information and keys

- Some can be small – pci card/smartcard like
- Some can be large – desktop box like

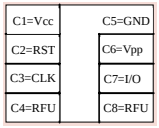
- Interfaces
  - often non-standard or
  - PKCS#11
  - Cloud use: e.g. Barbican

SMARTCARDS

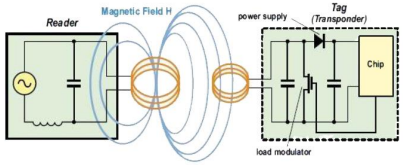
Parts of this material has been compiled from various open sources

I/O (Input/Output)

- Contact Interface
  - Vcc = 5 Volt (3 Volt)
  - Vpp not used anymore
  - CLK (3.5712, 4.9152, 10 MHz.)
  - UART for I/O

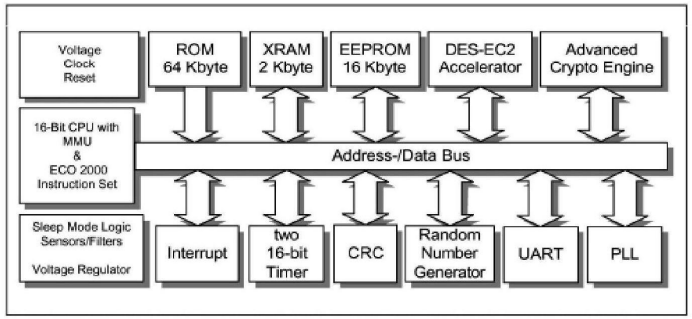


- Contactless Interface (125 kHz & 13.56 MHz)
  - Close coupled, a few millimeters
  - Proximity, less than 10 centimeter
  - Vicinity, more than 10 centimeter

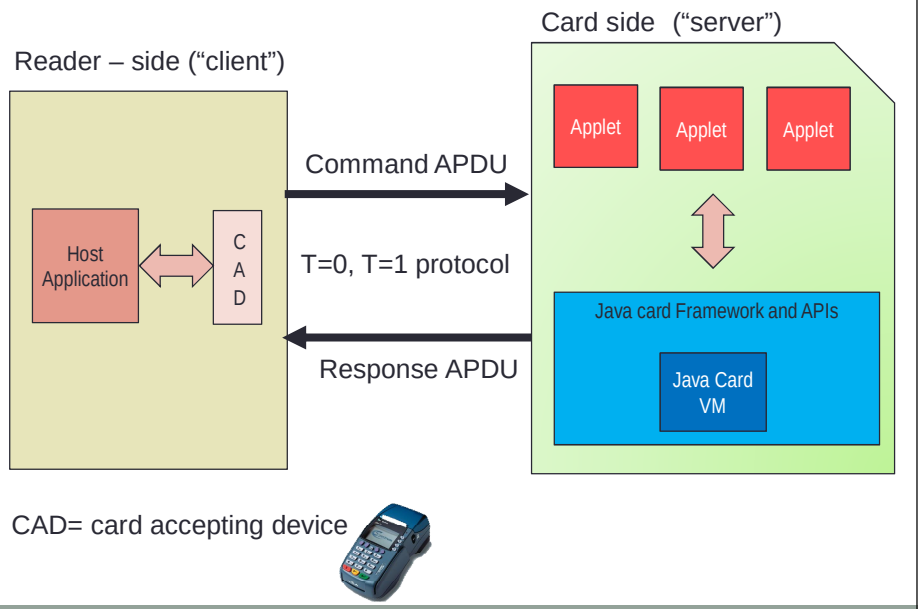


Smartcard architecture
(example Infineon SLE66)

- **Smart card IC processor product with advanced security mechanisms (cryptographic engine, physical protection)**
- **Certifed according to EAL5**



The Message-Passing Model



CAD= card accepting device

Data Transmission T=0 protocol

- Card acts as server
- Byte oriented
- TPDU (Transmission Protocol Data Unit) ≈ APDU
  - CAD transmits CLA, INS, P1, P2, P3
  - Card transmits procedure byte ACK
  - Following communication depends on Command
  - Communications end with status bytes SW1, SW2
- Transmission errors detected via parity bit and corrected via second time transmission
- Poor separation of application and data link layer

IN → OUT

APDU for Commands							APDU for Response			Compulsory
cla	ins	P1	P2	K	data	le	data	sw1	sw2	Optional

Data Transmission T=1 protocol

- Card acts as server
- Block oriented

Prologue			Information	Epilogue
NAD	PCB	LEN	APDU	EDC
1 Byte	1 Byte	1Byte	0 - 254 Bytes	1-2 Bytes

  - Block types:
    - I - application data
    - R - receive confirmation
    - S - protocol control data
- Good separation of application and data link layer which is good for multi application cards (Yet T=0 protocol is the one that is most often used.)
- Transmission errors detected with EDC: LRC (XOR byte) or CRC ($x^{16}+x^{12}+x^5+1$), correction via S-block + PCB

Study questions

- Why not only depend on closed platforms for building mission critical systems?
- Is it possible to implement a secure system using only sw?
- What is the security relevance of virtualization?
- What are the main security roles of a hypervisor?
- Why runs the guest OS kernel not in privileged mode?
- Explain the role of the app signatures and compare it with signed applets in Java
- What makes SELinux a special OS? It's pros and cons.
- Explain the three main use cases for a TPM.
- Which keys are permanently stored in a TPM?
- In connection to TPM use what is a sealed key?
- What is RTS, RTR and RTM?
- What is the difference between SRTM and DRTM?

- What is attestation?
- What keys are used for attestation?
- What are enclaves in the SGX Technology
- Secure boot vs authenticated boot
- Explain the concept behind OpenStack Trustpools
- Explain main security features of UEFI
- Describe ARM TrustZone
- What is an HSM and give two use cases.
- How do applications communicate with a smart card (messages, protocols?)
- By what mechanisms are applications sandboxed in Android and in Java?

SOME THINGS OF INTEREST

Alternative to trusted computing

- If we can do use (almost) homomorphic encryption techniques then we can outsource processing (certain types) of sensitive data to an untrusted compute pool !

- For example CryptDB from MIT.

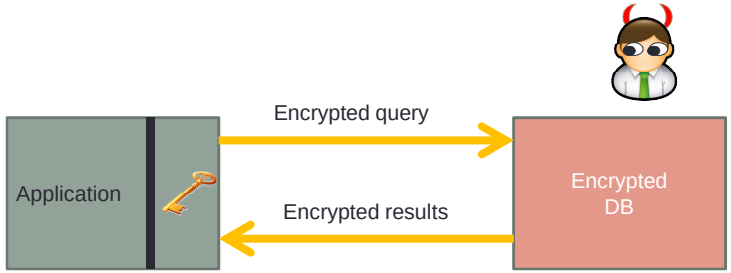
  (in cryptDB information on stored data still may leak during processing, but the idea is very nice, and it works pretty efficient)

  css.csail.mit.edu/cryptdb/

  Unfortunately Homomorphic encryption is still not practical except for some very special (use) cases such as voting.

Database operations on encrypted data

- See http://css.csail.mit.edu/cryptdb/



Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), Cascais, Portugal, October 2011.