PV204 Security technologies

Trusted boot

Petr Švenda svenda@fi.muni.cz
Faculty of Informatics, Masaryk University

CROCS
Centre for Research on Cryptography and Security

Overview

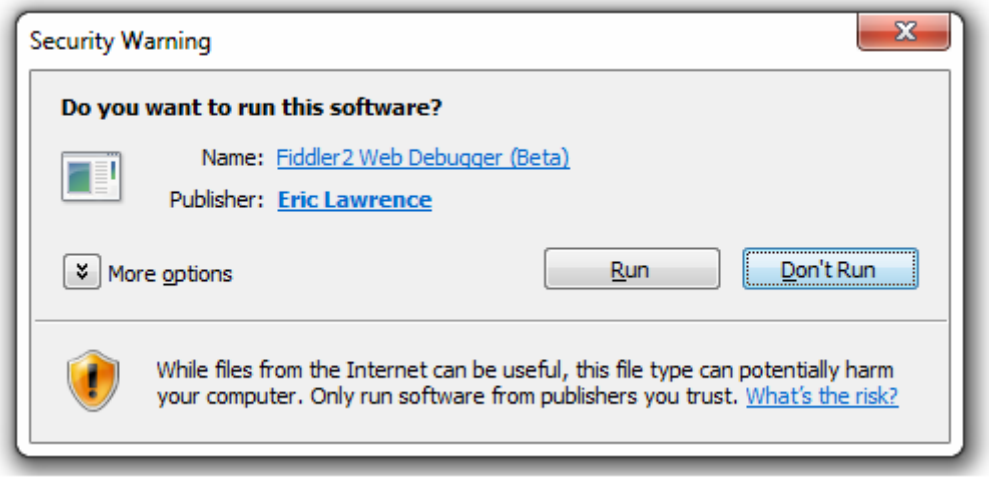
- Booting chain of programs
- BIOS as root of trust
- Verified and Measured boot
- Trusted boot in the wild
  - Trusted Platform Module
  - Chromium, Windows 8/10, UEFI…
- Dynamic root of trust
  - Intel's TXT, SGX

Motivation – untrusted host platform

- Traditional role of operating system
  - Isolate processed
  - Manage privileges, authorize operations
- But how to deal with
  - Debugger, disassembler
  - Intercepted multimedia output
  - Malware run along with banking app
  - Keyloggers
  - System administrators
  - Service providers
  - Evil maid…

Solution?

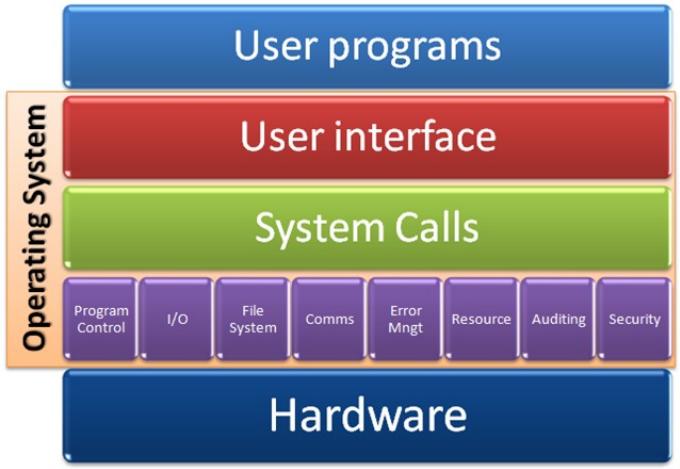
- Code signing (e.g., Microsoft AuthentiCode)
  - Application binary is signed, PKI used to verify certificate
  - If not signed, user is notified
  - Mandatory signing for selected applications (drivers…)



Signed == Secure?

Trust in program's functionality

- Trust in a program's code?
  - Signed code can still contain bugs and vulnerabilities
- Trust only in a program's code?
  - Underlying OS layers
  - Underlying firmware
  - Underlying hardware
  - Memory used by the program
  - Other code with access to the program's memory/code
  - …
- The program is almost never executed "alone"



Problem statement

- How to make sure that valid programs run only within valid environment?

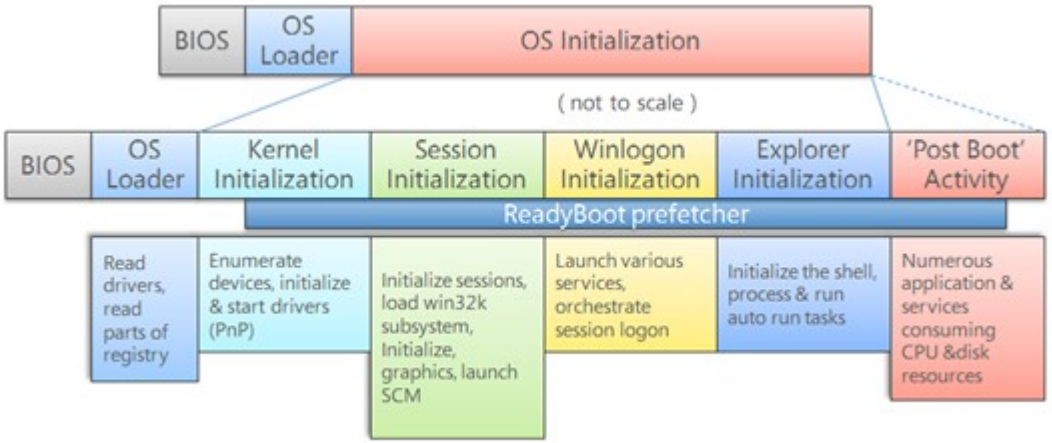
1. Is possible to start valid environment on previously compromised machine?
2. Is possible to prevent tampering of apps against attacker with physical access?
3. How to prove to remote party what apps are running on local machine?

Classical boot chain

Linux

BIOS	Basic Input/Output System executes MBR
MBR	Master Boot Record executes GRUB
GRUB	Grand Unified Bootloader executes Kernel
Kernel	Kernel executes /sbin/init
Init	Init executes runlevel programs
Runlevel	Runlevel programs are executed from /etc/rc.d/rc*.d/

Windows (7)



How to detect that BIOS or OS Loader was modified? (evil maid, bootkit…)

http://www.thegeekstuff.com/2011/02/linux-boot-process/
http://social.technet.microsoft.com/wiki/contents/articles/11341.the-windows-7-boot-process-sbsl.aspx

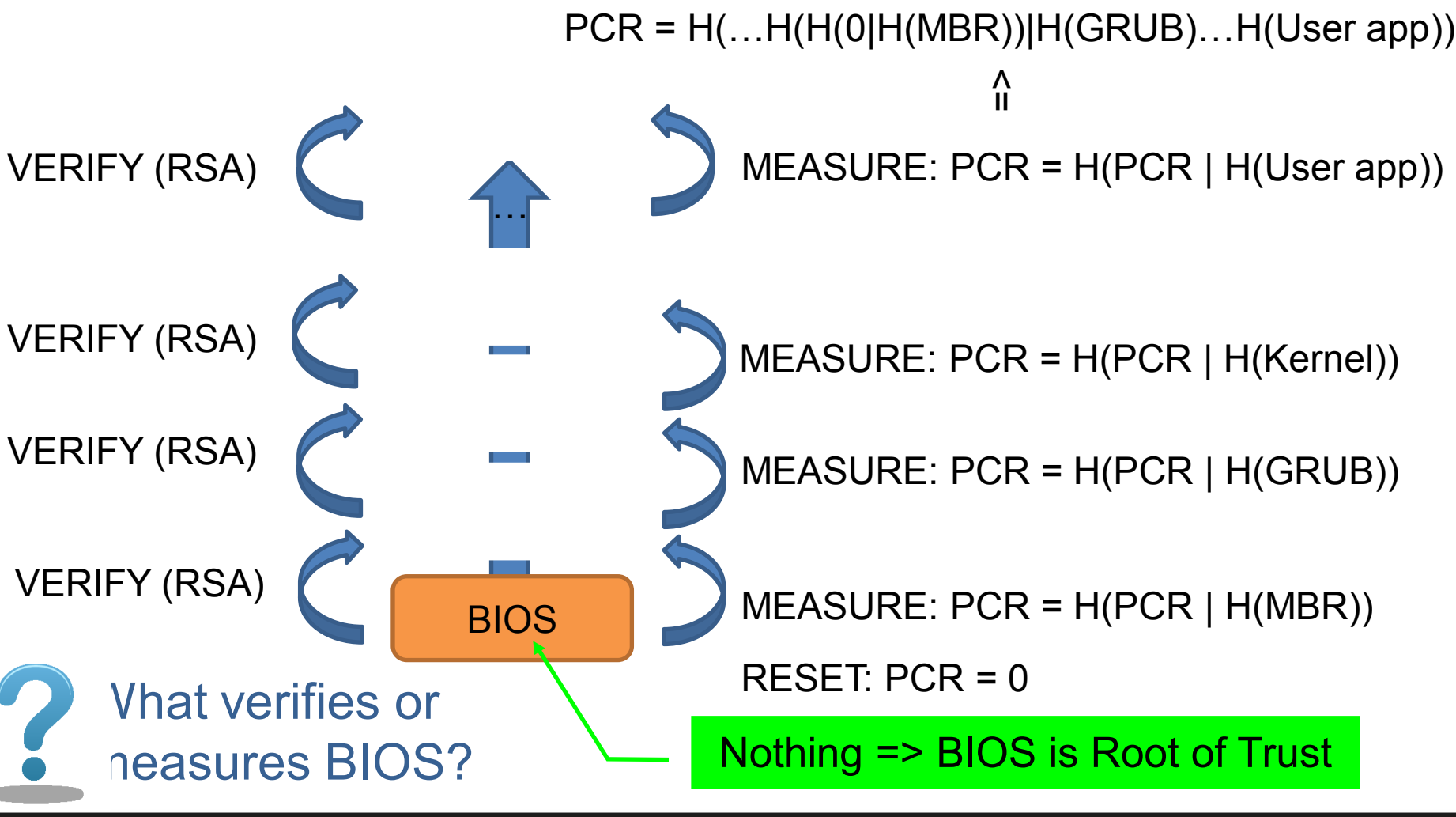
How to arrive at expected chain of apps?

1. Just trust the whole boot process
2. Signature-based approach: “Verified boot”
   - Before next app is executed, its signature is verified
   - Requires valid (unforged) public key (integrity)
   - Requires trust to owner of private key (signs only valid applications)
3. Create un-spoofable log what executed: “Measured” boot
   - Before next app is executed, its hash (“measurement”) is computed added to un-spoofable log (TPM’s PCR)
   - Will NOT prevent run of unwanted app, but environment cannot lie about what was executed (after-the-fact examination)
   - Requires (protected) log storage (Trusted Platform Module)
   - May require authentication of log (Remote attestation)

Trusted boot

“Verified” boot

“Measured” boot



Root of trust (for verified/measured boot)

- Verified and Measured boot need some root of trust
  - Initial piece of code that nobody verifies/measures
- Static root of trust
  - Start building trusted chain after reset of whole device
- Dynamic root of trust
  - Start building trusted chain without reset of device (faster)
- What can be root of trust?
  - static root of trust: BIOS, UEFI firmware, Intel Boot Guard
  - dynamic root of trust: Intel TXT, Intel SGX
- Root of trust requires special protection
  - As nobody verifies than nobody will detect eventual modification

BIOS as root of trust

- First code executed on CPU of target machine
- Privileged access to hardware
  - E.g., can write into memory of OS code via DMA
- Provides code for System Management Mode (SMM)
  - Routines executed during the whole platform runtime
  - x86 feature since 386, all normal execution is suspended
  - Used for power management, memory errors, hardware-assisted debugger…
  - Very powerful mode (=> also target of “ring -2” rootkits)

BIOS – security considerations

- How BIOS verifies integrity of next module to run?
- Where public key(s) for verification are stored?
- How to handle updates of signing keys?
- How BIOS checks signatures on its own updates?
- How BIOS can be compromised?

How BIOS can be compromised?

1. Maliciously written by BIOS vendor (backdoor)
2. Replacement of genuine BIOS by malicious one
   - By physical flash (SPI programmer) of BIOS code
   - By lack of flashing protection mechanism by original BIOS
   - By code logic flaws in BIOS locking mechanisms
3. Modification of other code/data used by BIOS
   - Bug in parsing unsigned data…

- Currently used protections:
  - Chipset-enforced protection of flash memory with BIOS
  - BIOS signature verification before new version is written
  - Hardware-aided check of executed code (TPM, TXT, SGX)
  - Check of BIOS signature before execution by CPU (IBG)

Attacks against BIOS locks

1. Attacks typically via BIOS code vulnerability
   - BIOS usually does not takes (much) user input, but parsing of BIOS update blob, parts are unsigned (logo)
   - Buffer overflow in logo parsing => Locks are not locked yet => write own BIOS
   - http://invisiblethingslab.com/resources/bh09usa/Attacking%20Intel%20BIOS.pdf
2. Write into flash memory by SPI programmer

Which one is more serious? Different attacker models
1. Is remote, but patchable
2. Is local attacker, but requires design changes to prevent

Impact: Attack against Tails live-CD distro

- Tails is live-CD Linux distribution
- Designed to provide security even on previously compromised computer
  - Boot complete fresh OS from live-CD + security tools
- Attack 1: Physical BIOS modification
  - Modified BIOS inserts malicious code into Tails during boot time
  - Known thread, physical access to computer assumed
- Attack 2: SMM rootkit (LightEater)
  - Bug in BIOS exploited by remote party to modify SMM routines
- Main issue: Tails tries to start with clean erased computer, but some elements still persists erase (BIOS modification)

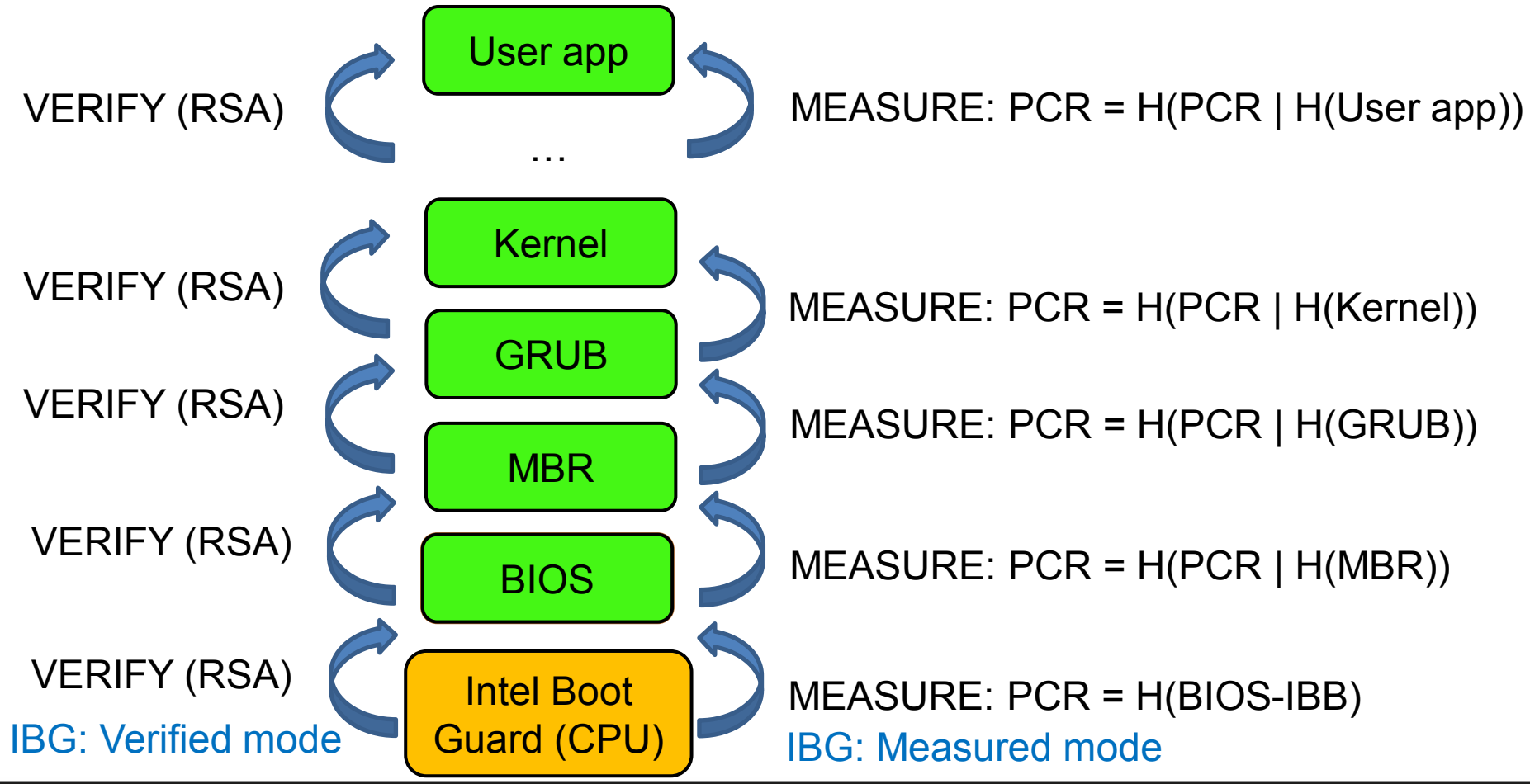
INTEL BOOT GUARD (IBG)

Intel Boot Guard (IBG)

- Recently (2014) introduced feature to protect BIOS
  - Piece of trusted processor-provided, ROM-based code
  - Runs first after reset, verifies *Initial Boot Block (IBB)*

1. "Measured" boot mode (TPM-based)
   - Passively extends TPM's PCRs by hash of IBB
2. "Verified" boot mode (digital signature)
   - OEM vendor hardcodes public key via fuses into CPU
   - Intel Boot Guard checks signature of IBB by OEM's key
   - Only vendor-approved IBB=>BIOS=>OS is executed
3. Combination of measured and verified mode

Intel Boot Guard – new root of trust



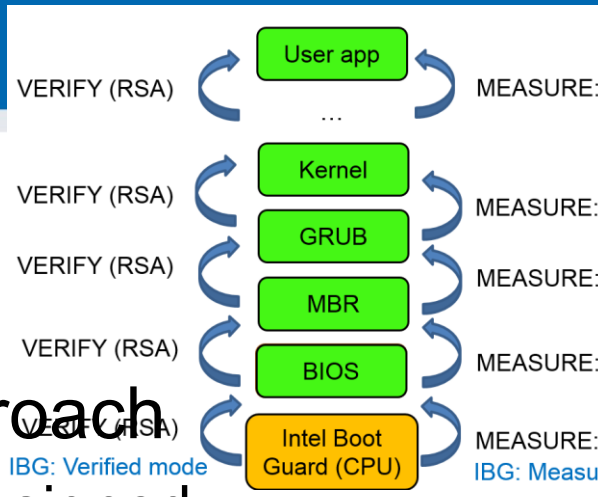
Intel Boot Guard – security improvements

- What attacks are mitigated by Intel Boot Guard?
- Direct BIOS flash by SPI programmer
  - Mitigated, signature/measurement mismatch
- Remote change of BIOS / BIOS data
  - Mitigated, signature/measurement mismatch
- Other bug in BIOS code
  - Not mitigated, signed code still contains bug

- Any new attacks opened by IBG?

How hard is to incorporate backdoor?

- OEM vendor can sign backdoored BIOS
  - But multiple OEM vendors exist, open-source coreboot
- Intel Boot Guard is written by Intel only
  - But OEM fuses own verification public key, right?
  - But it is the IBG code that actually verifies
- Trivial backdoor (inside IBG code inside CPU)
  - if (IBB[SOME_OFFSET] == BACKDOOR_MAGIC) then always load provided BIOS (no signature check)
  - Or possibly verify by some other public key (secure even when BACKDOOR_MAGIC is leaked)



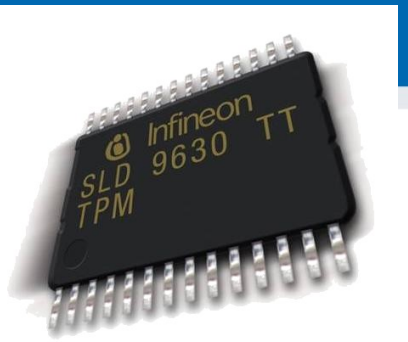
Short (intermediate) summary

- Signature-based “verified” boot approach
  - Whitelisting approach – run only what is signed
  - Robust signature process needed (trust in private key owner)
  - Integrity of verification public key is critical
  - Key management is necessary (multiple keys, key updates)
- “Measured” boot approach
  - Un-spoofable log of hashes of executed code
  - Can be remotely verified (remote attestation, explained later)
- Root of trust needs to be protected
  - Historically was BIOS (+ update signatures + write locks)
  - Recently Intel Boot Guard inside CPU (signature of BIOS)

TRUSTED PLATFORM MODULE

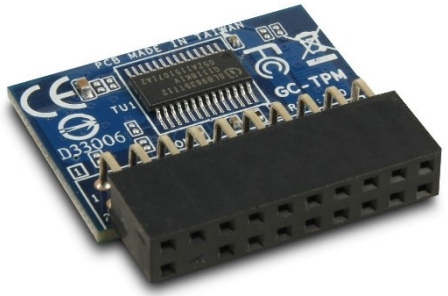
Trusted Platform Module (TPM)

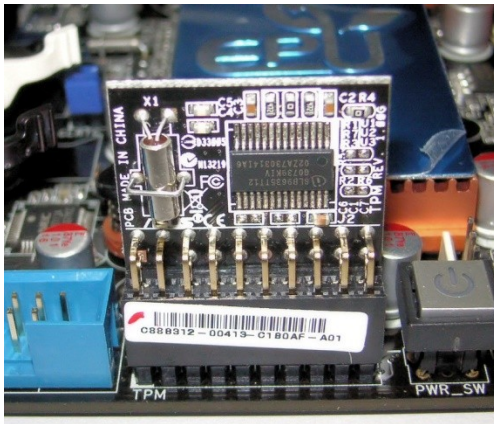
- Standard for secure crypto-processor
  - ISO/IEC 11889 (2009)
- Additional versions published by TCG
  - Trusted Computing Group (TCG)
  - TPM 1.2 (2011)
  - TPM 2.0 (2014, draft, not compatible with 1.2)
    - http://www.trustedcomputinggroup.org/resources/tpm_library_specification



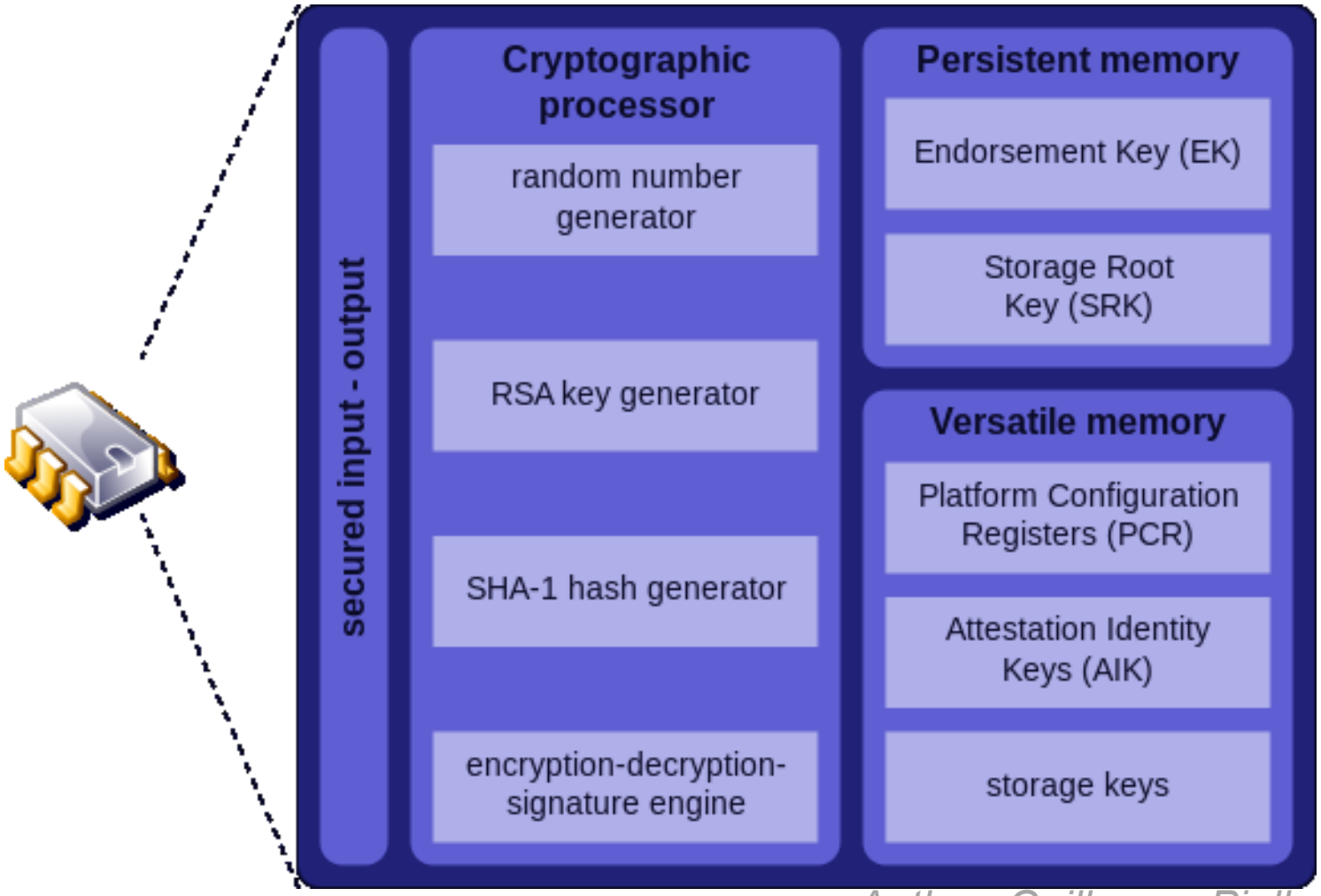
TPM hardware

- Cryptographic smart card connected/inside to device
  - Secure storage, secure crypto environment…
  - (But not programmable JavaCard ☺)
- Physical placement
  1. Additional chip on motherboard
  2. Incorporated inside CPU
  3. Incorporated in peripheral (Ethernet card)
- Accessed during boot time
  - “Measured” boot (TPM’s PCR registers)
  - Bitlocker encrypted drive keys
- Accessed later (private key operation)





Trusted platform module



Author: Guillaume Piolle

TPM 1.2 vs. TPM 2.0

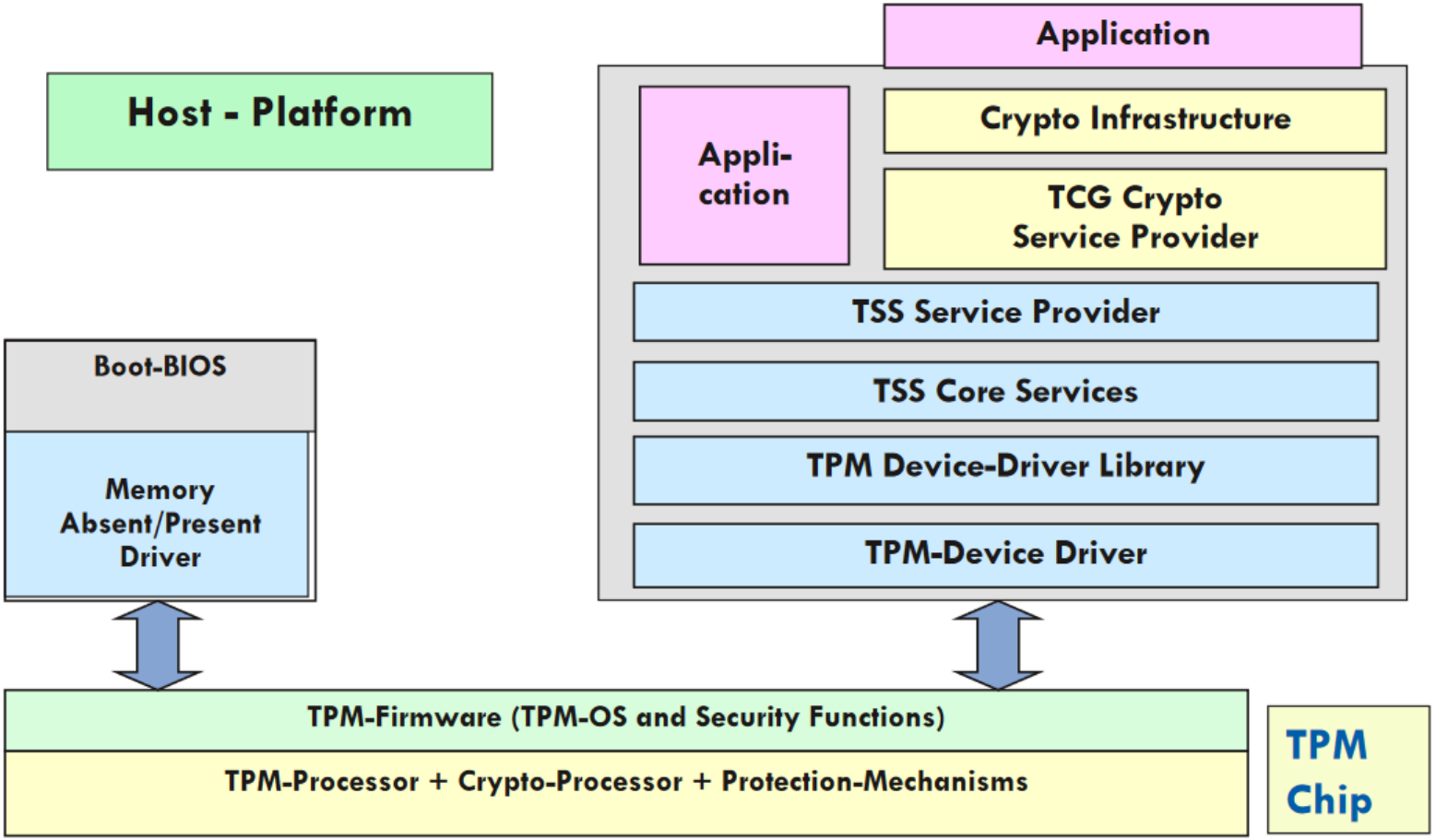
	TPM 1.2	TPM 2.0
Algorithms	SHA-1, RSA	Agile (such as SHA-1, SHA-256, RSA and Elliptic curve cryptography P256)
Crypto Primitives	RNG, SHA-1	RNG, RSA, SHA-1, SHA-256
Hierarchy	One (storage)	Three (platform, storage and endorsement)
Root Keys	One (SRK RSA-2048)	Multiple keys and algorithms per hierarchy
Authorization	HMAC, PCR, locality, physical presence	Password, HMAC, and policy (which covers HMAC, PRC, locality, and physical presence).
NV RAM	Unstructured data	Unstructured data, Counter, Bitmap, Extend

https://en.wikipedia.org/wiki/Trusted_Platform_Module

Provided security functions

I. "Measured" boot with remote attestation
   – Provide signed log of what executed on platform (PCR)
II. Storage of keys (disk encryption, private keys…)
   – Can be additionally password protected
III. Binding and Sealing of data
   – Encryption key wrapped by concrete TPM's public key
IV. Platform integrity
   – Software will not start if current PCR value is not right

TPM Trusted Software Stack stack



Infineon, http://www.cs.unh.edu/~it666/reading_list/Hardware/tpm_fundamentals.pdf

TPM PCR

- Platform Configuration Register (PCR)
- Measurement cumulatively stored in PCR
  - measurement = SHA1(next block to execute)
  - PCR[i] = SHA1(PCR[i] | new_measurement)
  - Current block measure & store next before passing control
- PCR cannot be erased until reboot
  - Every part that was executed is stored
  - After-the-fact verification what happened
- Idea: boot what you want, but PCR will hold trace

Remote attestation

- Multiple PCRs to support finer grained reporting
  - not just single cumulative value
- Multiple PCRs available
  - BIOS, ROM, Memory Block Register [index 0-4]
  - OS loaders [5-7], Operating System [8-15]
  - Debug [16], Localities, Trusted OS [17-22]
  - Application specific [23]
- What is PCR measurement good for?
  - PCR content can be signed by TPM's private key and exported
  - List of applications claimed to be executed (=> PCR expected value can be recomputed by remote party)
  - => Remote attestation

Platform attestation – PCR registers

```
<PlatformAttestation size="30591">
  <Magic>PADS<!-- 0x53444150 --></Magic>
  <Platform>TPM_VERSION_12</Platform>
  <HeaderSize>28</HeaderSize>
  <PcrValues size="480">
    <PCR Index="0">8cb1a2e093cf41c1a726bab3e10bc1750180bbc5</PCR>
    <PCR Index="1">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
    <PCR Index="2">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
    <PCR Index="3">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
    <PCR Index="4">68fffb7e5c5f6e6461b3527a0694f41ebd07e4e1</PCR>
    <PCR Index="5">8e33d52190def152c9939e9dd9b0ea84da25d29b</PCR>
    <PCR Index="6">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
    <PCR Index="7">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
    <PCR Index="8">0000000000000000000000000000000000000000</PCR>
    <PCR Index="9">0000000000000000000000000000000000000000</PCR>
    <PCR Index="10">0000000000000000000000000000000000000000</PCR>
    <PCR Index="11">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
    <PCR Index="12">7c84e69cd581eefd7ebe1406666711fd4fda8aa8</PCR>
    <PCR Index="13">01788a8a31f2dafcd9fe58c5a11701e187687d49</PCR>
    <PCR Index="14">26cda47f1db41bedc2c2b1e6c91311c98b4e2246</PCR>
    <PCR Index="15">0000000000000000000000000000000000000000</PCR>
    <PCR Index="16">0000000000000000000000000000000000000000</PCR>
    <PCR Index="17">ffffffffffffffffffffffffffffffffffffffff</PCR>
    <PCR Index="18">ffffffffffffffffffffffffffffffffffffffff</PCR>
    <PCR Index="19">ffffffffffffffffffffffffffffffffffffffff</PCR>
    <PCR Index="20">ffffffffffffffffffffffffffffffffffffffff</PCR>
    <PCR Index="21">ffffffffffffffffffffffffffffffffffffffff</PCR>
    <PCR Index="22">ffffffffffffffffffffffffffffffffffffffff</PCR>
    <PCR Index="23">0000000000000000000000000000000000000000</PCR>
  </PcrValues>
...
```

TPM platform info

- Provides information about your platform state

```
<PlatformCounters>
  <OsBootCount>44</OsBootCount>
  <OsResumeCount>2</OsResumeCount>
  <CurrentBootCount>0</CurrentBootCount>
  <CurrentEventCount>66</CurrentEventCount>
  <CurrentCounterId>179136858</CurrentCounterId>
  <InitialBootCount>0</InitialBootCount>
  <InitialEventCount>64</InitialEventCount>
  <InitialCounterId>179136858</InitialCounterId>
</PlatformAttestation>
```

Reboot =>

```
<PlatformCounters>
  <OsBootCount>45</OsBootCo
  <OsResumeCount>0</OsResu
  <CurrentBootCount>0</Curre
  <CurrentEventCount>67</Cur
  <CurrentCounterId>17913685
  <InitialBootCount>0</InitialBo
  <InitialEventCount>67</Initial
  <InitialCounterId>179136858<
</PlatformAttestation>
```

TRUSTED BOOT – REAL IMPLEMENTATIONS

Verified boot - Chromium OS

- Starts with read-only part of firmware/BIOS (root of trust)
  - Cannot be forged, but also cannot be not updated
  - Contains permanently stored root RSA public key
- “Verified” boot strategy is used
  - Verifies that all executed code is from Chromium OS source tree
  - Code signatures verified by (shorter) keys signed by root key
    - speed tradeoff + possibility to update compromised keys
- Does not completely prevent user to boot other OSes
  - Developer mode turned on => signature on kernel not checked
  - TPM is used to provide mode reporting (normal/devel/recovery)
- https://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot
- https://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot-crypto

Chromium OS uses of TPM

- Limited remote attestation (PCR[0] used)
  - to store developer and recovery mode switches
- Prevent rollback attack
  - Prevented by strictly increasing version of key & firmware
  - Version is written in TPM's NV RAM location, only read-only firmware can update
  - Key version prevents update to older compromised key
  - Firmware version prevents update to vulnerable firmware
- Store selected user's private keys
- Wrap selected disk encryption keys by TPM's system key
- https://www.chromium.org/developers/design-documents/tpm-usage

Secured and Trusted Boot

UEFI SECURE BOOT

UEFI secure boot principles

- Platform key (RSA 2048b, PK) for authentication of platform owner
- Key exchange keys (KEKs) for authentication of other components (drivers, OS components…)

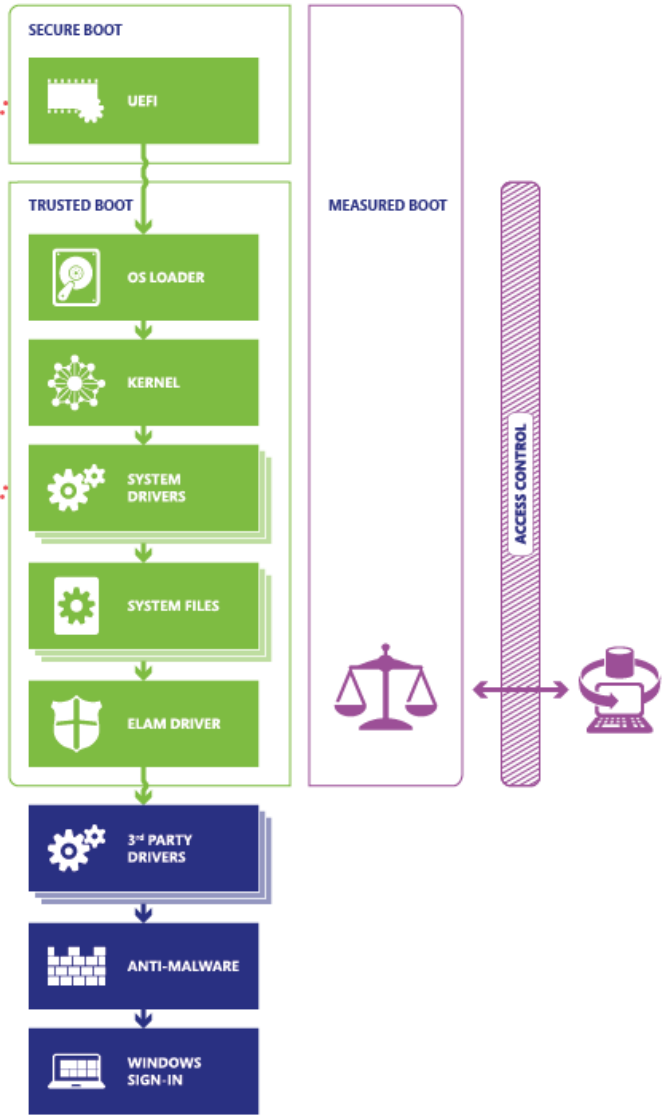
1. “Setup" mode – platform key (PK) is not loaded yet
   - Everybody can write its own platform key
   - Once PK is written, switch to “user” mode
2. “User” mode
   - New keys (PKs, KEKs) can be written only if signed by PK
   - New software components loaded only if signed by KEKs

Secured and Trusted Boot

WINDOWS 8/10 TRUSTED BOOT

Windows 8/10 trusted boot

- Certified Windows 8/10 devices have trusted boot by default
  - "Verified" boot used (UEFI+OS sign)
  - "Measured" boot used (TPM)
- TPM PCRs used for measurements
- TPM used for keys protection
  - Bitlocker disk encryption key



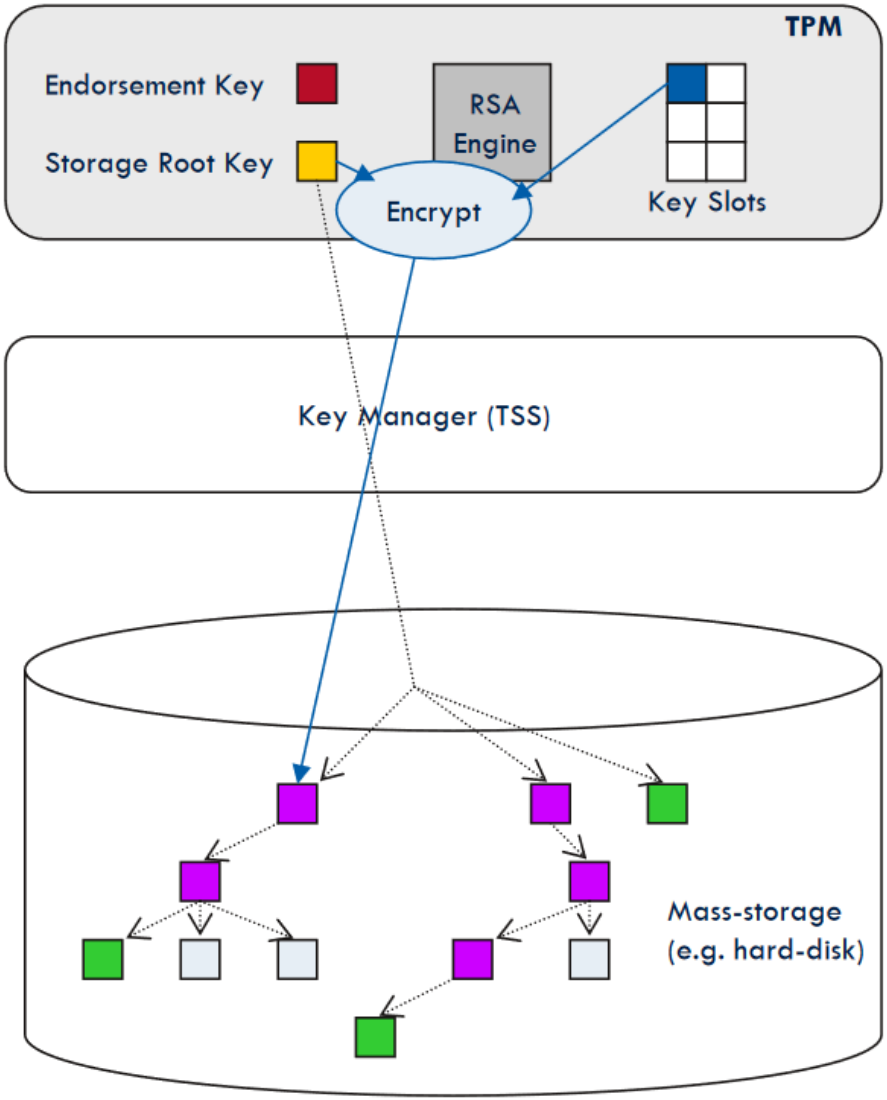
BASIC COMPONENTS

TPM keys

- Endorsement key (EK)
  - Generated during manufacturing, permanent
  - Remain in TPM device during whole chip lifetime
- TPM Storage Root Key (SRK)
  - Generated by use after taking ownership
  - New Storage root key can be generated after TPM clear
  - Used to protect TPM keys created by application
- Various delegate keys
  - Separate keys signed/wrapped by EK, SRK…
  - Application can generate and store own keys
  - Good practice: not to have single key for everything

TPM storage keys

- Application keys encrypted under SRK
- Exported as protected blob
- Stored on mass-storage
- If needed, decrypted back and placed into slot
- Key usable until removed



http://www.cs.unh.edu/~it666/reading_list/Hardware/tpm_fundamentals.pdf

TPM policy

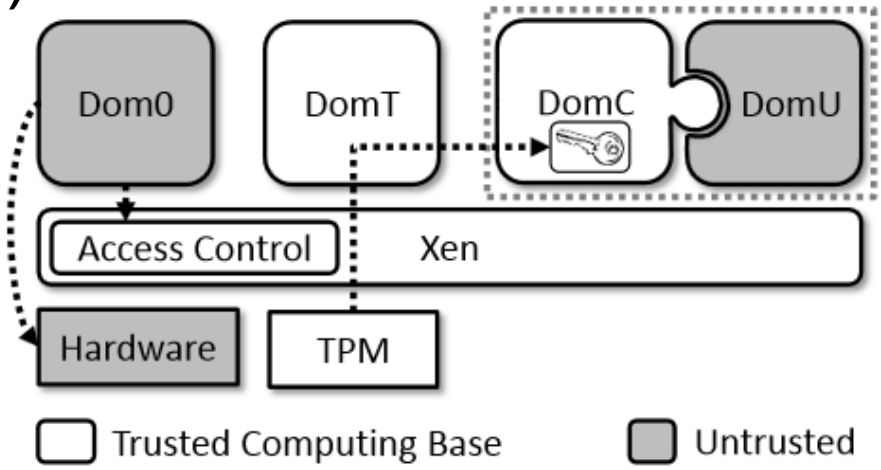
- TPM releases secret only when PCR contains particular value
- Enforcement even in measured-only mode
  - Key is not released if unexpected component was started (started => is included in measurements)
- Conditions can use ANDs and ORs
- How to handle policy updates?
  - Change policy of state only from already valid state

Programming with TPM

- TPM Platform Crypto-Provider Toolkit
  - https://research.microsoft.com/en-us/downloads/74c45746-24ad-4cb7-ba4b-0c6df2f92d5d/
  - Source code examples for Attestation and trusted boot
- Measured Boot Tool
  - https://mbt.codeplex.com/
  - Demonstrates TPM secure boot and remote attestation on Windows 8/10
  - List of most important functions, example code
- TrouSerS: open-source stack for TCG
  - Linux version (2008)
    - http://trousers.sourceforge.net/
    - https://sourceforge.net/projects/trousers/files/tpm-tools/
  - Windows port (2010)
    - http://security.polito.it/trusted-computing/trousers-for-windows/

Usage of TPM in cloud-computing

- Combination of virtualization and trusted computing
- Modified Xen hypervisor used to make standard TPM available for secret-less virtual machine
- Results in significant decrease in the size of trusted computational base (TCB)



http://bleikertz.com/research/acns2013.pdf

DYNAMIC ROOT OF TRUST

Static Root of Trust Measurement (SRTM)

- Start trusted immutable piece of firmware
  - E.g., BIOS loader or Intel Boot Guard
- Initiates measurement process
  - Integrity of every next component is added to TPM's PCRs
  - Start → BIOS → PCI EEPROM → MBR → OS …
- But do we need to start only after reboot?
  - Takes relatively long time
  - Can we execute same process, but dynamically?
  - Can we exclude long chain (BIOS, PCI…)?

Dynamic Root Trust Measurement (DRTM)

- Launch of measured environment at any time
  - "Late lunch" option
  - No need to reset whole platform
  - Can be also terminated after some time
- Measurement process similar to static root of trust
  - Application trust chain executed from dynamic root
- Implementation of DRTM
  - Intel's TXT
  - Intel's SGX

Intel's Trusted Execution Technology

- Intel's TXT uses a processor-based root of trust
  - Option given in TCG specifications
- Goal: shorten chain of trust
  - Run specific program in verified/trusted chain without restart
- Goal: provide independent root of trust (CPU-based)
  - Processor isolates memory of Measured Launched Environment (MLE) from other processes
- Intel's TXT still uses TPM to store measurements
- http://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf

Intel's TXT issues

1. TXT still relies on BIOS provided code (SMM)
   - TXT-started chain can be compromised by forged BIOS
   - Hard to patch (design decision, not implementation bug)
   - Proposed defence by hardening and sandboxing SMM
2. Bugs in TXT implementation
   - Memory corruption, misconfiguring VT-d …
   - Can be fixed after discovery
3. Bugs in processing residual state of pre-TXT lunch
   - Maliciously modified ACPI tables
   - Can be fixed after discovery

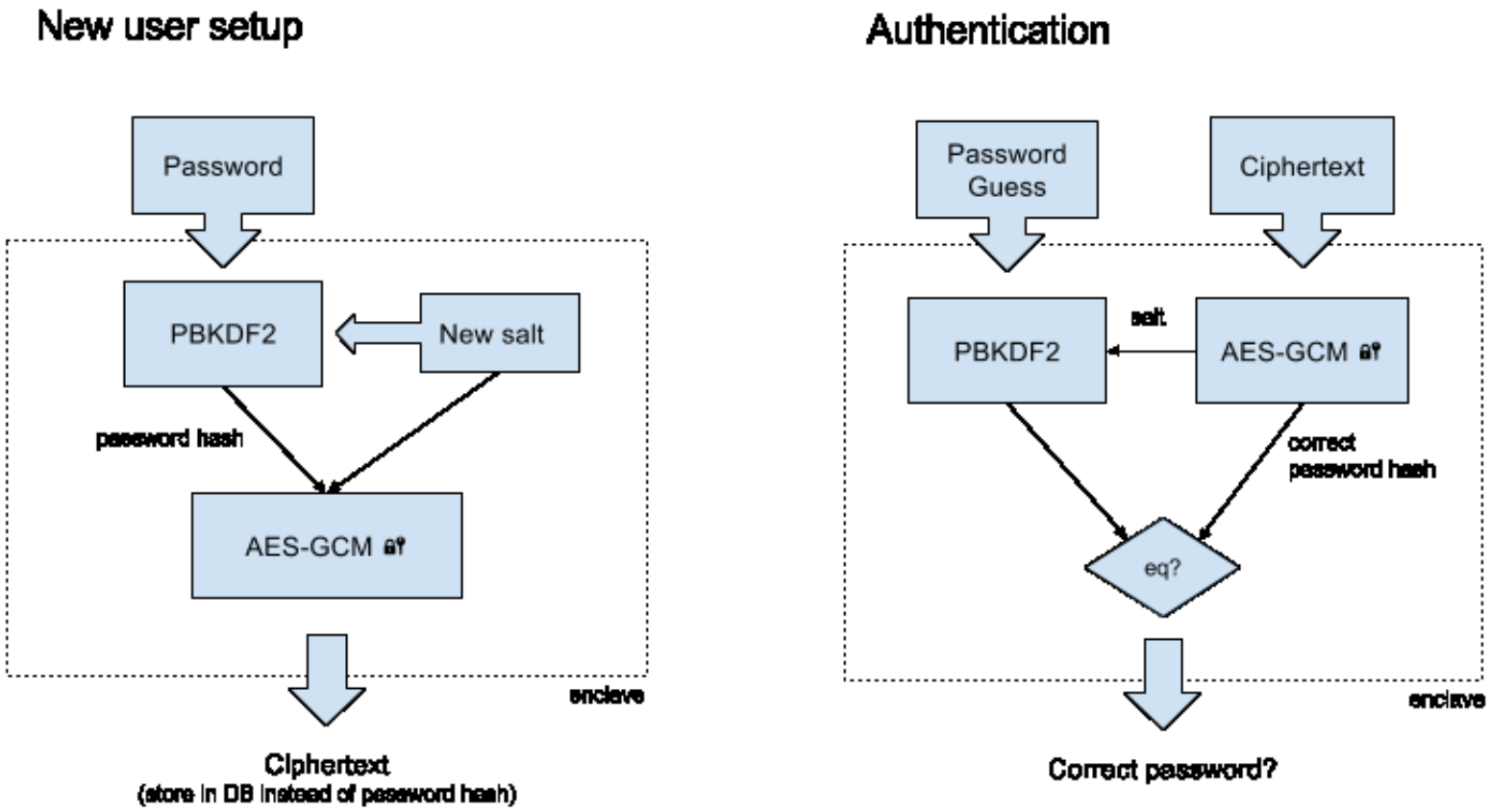
Intel's SGX : Security enclave

- Intel's Software Guard Extension (SGX)
  - New set of CPU instructions intended for future cloud server CPUs
- Protection against privileged attacker
  - Server admin with physical access, privileged malware
- Application requests private region of code and data
  - Security enclave (4KB for heap, stack, code)
  - Encrypted enclave is stored in main RAM memory, decrypted only inside CPU
  - Access from outside enclave is prevented on CPU level
  - Code for enclave is distributed as part of application

Intel's SGX – some details

- EGETKEY instruction generates new enclave key
  - SGX security version numbers
  - Device ID (unique number of CPU)
  - Owner epoch – additional entropy from user
- EREPORT instruction generates signed report
  - Local/remote attestation of target platform
- Debugging possible if application opt in
- Enclave cannot be emulated by VM

SGX hardened password verification



https://jpp.io/2016/01/17/using-sgx-to-hash-passwords/

Programming with Intel's SGX

- Intel SGX SDK
  - https://software.intel.com/en-us/sgx-sdk
  - 6th generation core processor (or later) based platform with SGX enabled BIOS support
- Example: Hardened password hashing
  - https://jbp.io/2016/01/17/using-sgx-to-hash-passwords/
  - https://github.com/ctz/sgx-pwenclave
- More SGX info
  - http://theinvisiblethings.blogspot.cz/2013/08/thoughts-on-intels-upcoming-software.html
  - http://theinvisiblethings.blogspot.cz/2013/09/thoughts-on-intels-upcoming-software.html

TRUSTED COMPUTING - CRITIQUE

Trusted computing - controversy

- For whom is your computed trusted?
  - Secure against you as an owner?
- Is TC preventing users to run code of their choice?
  - Custom OS distribution?
  - Open OEM system – locked on first installation
  - Physical switch to unlock later
- Why some people from *Trusted Computing* consortium think that Trustworthy Computing might be better title?

Trusted computing - controversy

- R. Anderson, `Trusted Computing' FAQ (2003)
  - http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html
- J. Edge, UEFI and "secure boot“
  - http://lwn.net/Articles/447381/
- R. Stallman, Can You Trust Your Computer?
  - https://www.gnu.org/philosophy/can-you-trust.html
- Selected problems addressed in current designs

Summary

- Two principal solutions for trusted boot
  - Verified boot (signatures) and Measured boot (PCR)
- Start from clean (and trusted) point
  - Allow only intended software to run
  - Or prove what actually executed
- Additional hardware inside motherboard / CPU provides wide range of new possibilities (TPM)
- Controversy about implication of trusted boot
  - Who owns and control target platform