PV204 Security technologies

Trusted Boot, TPM, SGX

Petr Švenda ***svenda@fi.muni.cz*** ***@rngsec***
Centre for Research on Cryptography and Security, Masaryk University

CROCS
Centre for Research on Cryptography and Security

Please comment on slides with anything unclear, incorrect or suggestions for improvement
https://drive.google.com/file/d/1i8K1d8JpIesLnMbf8S4QUNs3UEXhLbUr/view?usp=sharing

Overview

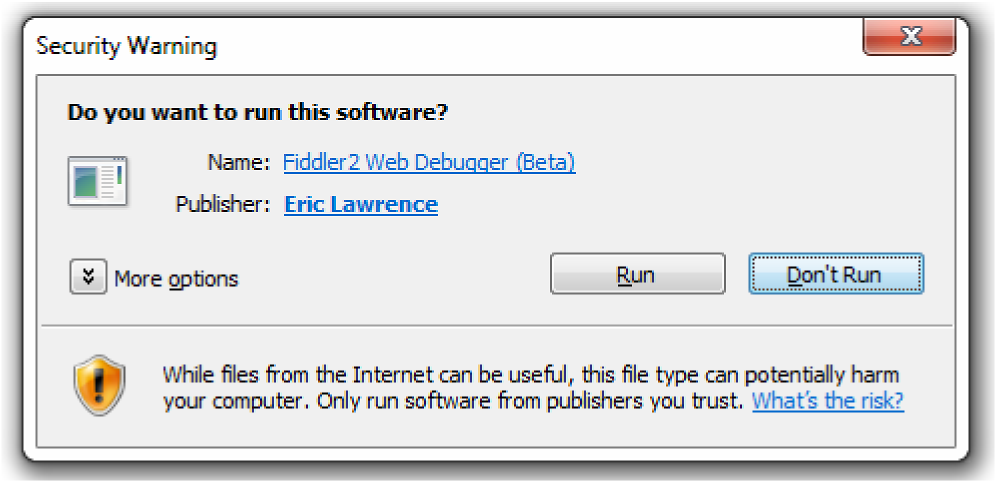
- Booting chain of programs
- BIOS as root of trust
- Verified and Measured boot
- Trusted boot in the wild
  - Trusted Platform Module
  - Chromium, Windows 8/10/11, UEFI…
- Dynamic root of trust
  - Intel's TXT, SGX

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, Evil maid
  - System administrators, Service providers
  - …

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…)



Security Warning

Do you want to run this software?

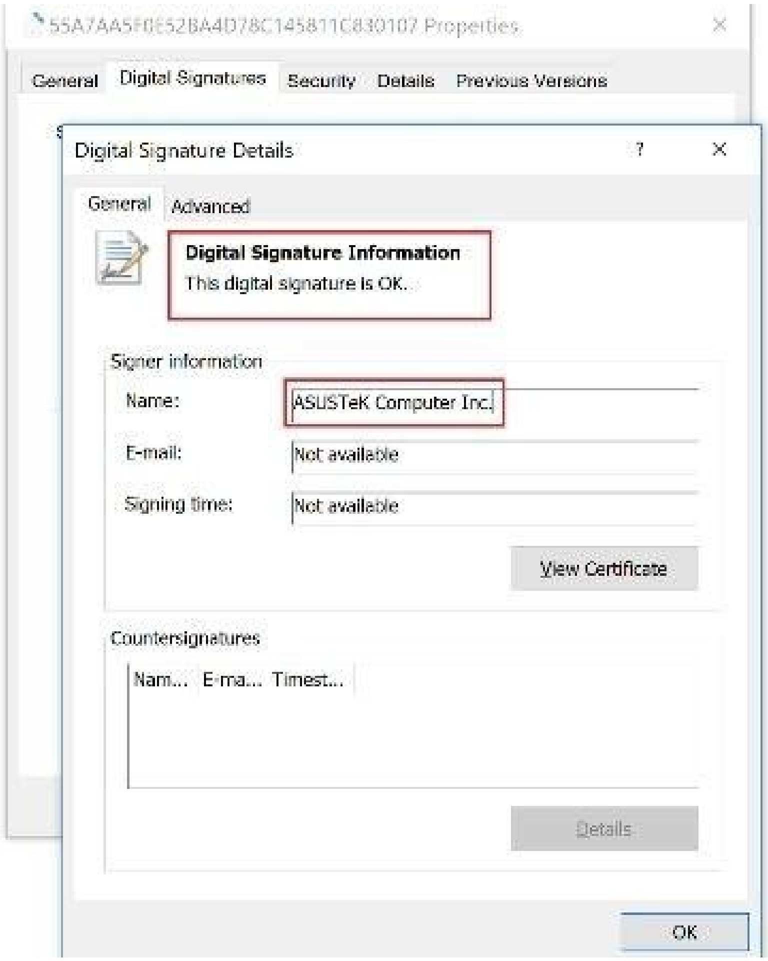
Name: Fiddler2 Web Debugger (Beta)

Publisher: Eric Lawrence

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Signed == Secure?

Signed == Secure?



55A7AA5F0E52BA4D78C145811C830107 Properties

General | Digital Signatures | Security | Details | Previous Versions

Digital Signature Details

General | Advanced

Digital Signature Information
This digital signature is OK.

Signer information

Name: ASUSTeK Computer Inc.
E-mail: Not available
Signing time: Not available

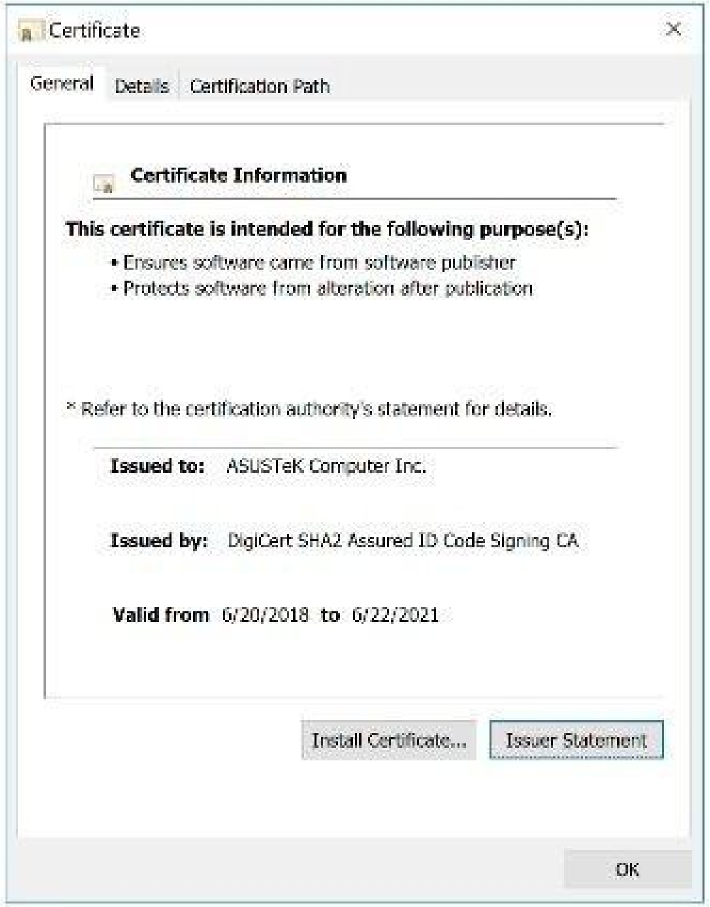
View Certificate

Countersignatures

Nam... E-ma... Timest...

Details

OK



Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):
- Ensures software came from software publisher
- Protects software from alteration after publication

* Refer to the certification authority's statement for details.

Issued to: ASUSTeK Computer Inc.

Issued by: DigiCert SHA2 Assured ID Code Signing CA

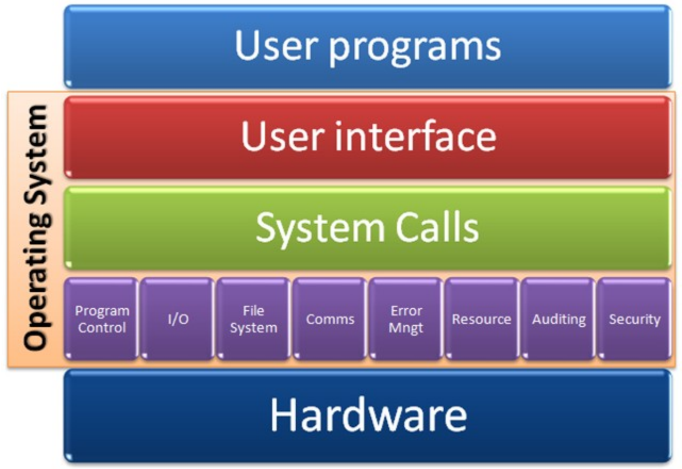
Valid from 6/20/2018 **to** 6/22/2021

Install Certificate... | Issuer Statement

OK

Trust in program's functionality

- Trust in a program code?
  - Signed code may still contain bugs and vulnerabilities
- Trust only in a program 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"



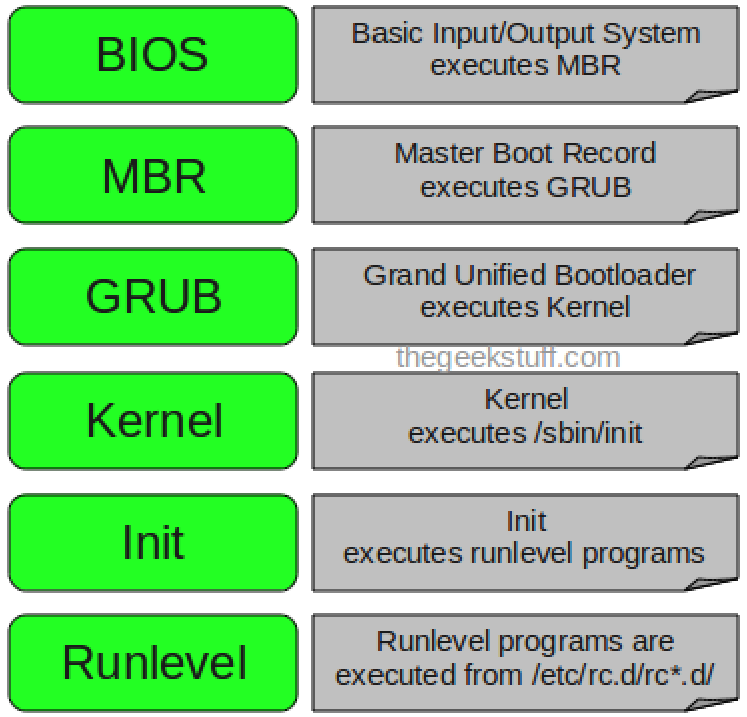
Problem statement

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

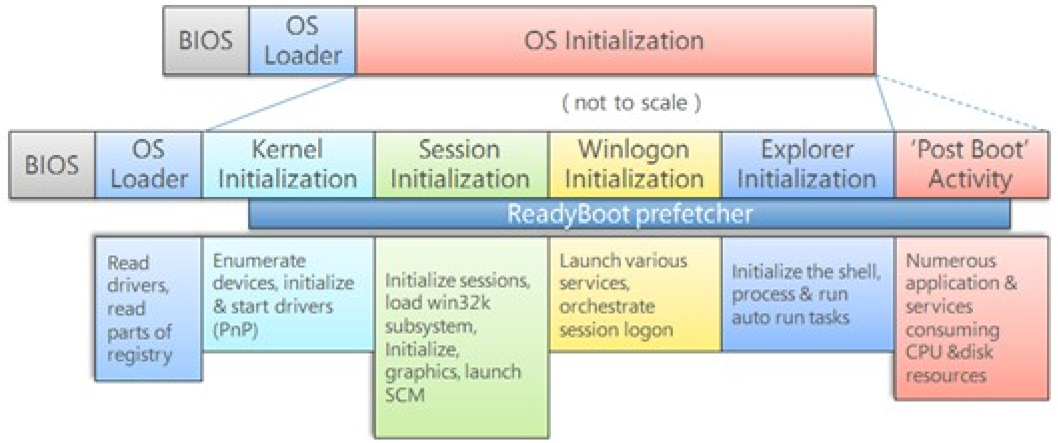
1. Is it possible to start valid “clean” environment on previously compromised machine?
2. Is it possible to prevent tampering of apps against an attacker with physical access?
3. How to prove what apps are running on local machine to a remote party?

Classical boot chain

Linux


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

Windows


BIOS
OS Loader
OS Initialization
(not to scale)
BIOS
OS Loader
Kernel Initialization
Session Initialization
Winlogon Initialization
Explorer Initialization
'Post Boot' Activity
ReadyBoot prefetcher
Read drivers, read parts of registry
Enumerate devices, initialize & start drivers (PnP)
Initialize sessions, load win32k subsystem, Initialize, graphics, launch SCM
Launch various services, orchestrate session logon
Initialize the shell, process & run auto run tasks
Numerous application & services consuming CPU &disk resources

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

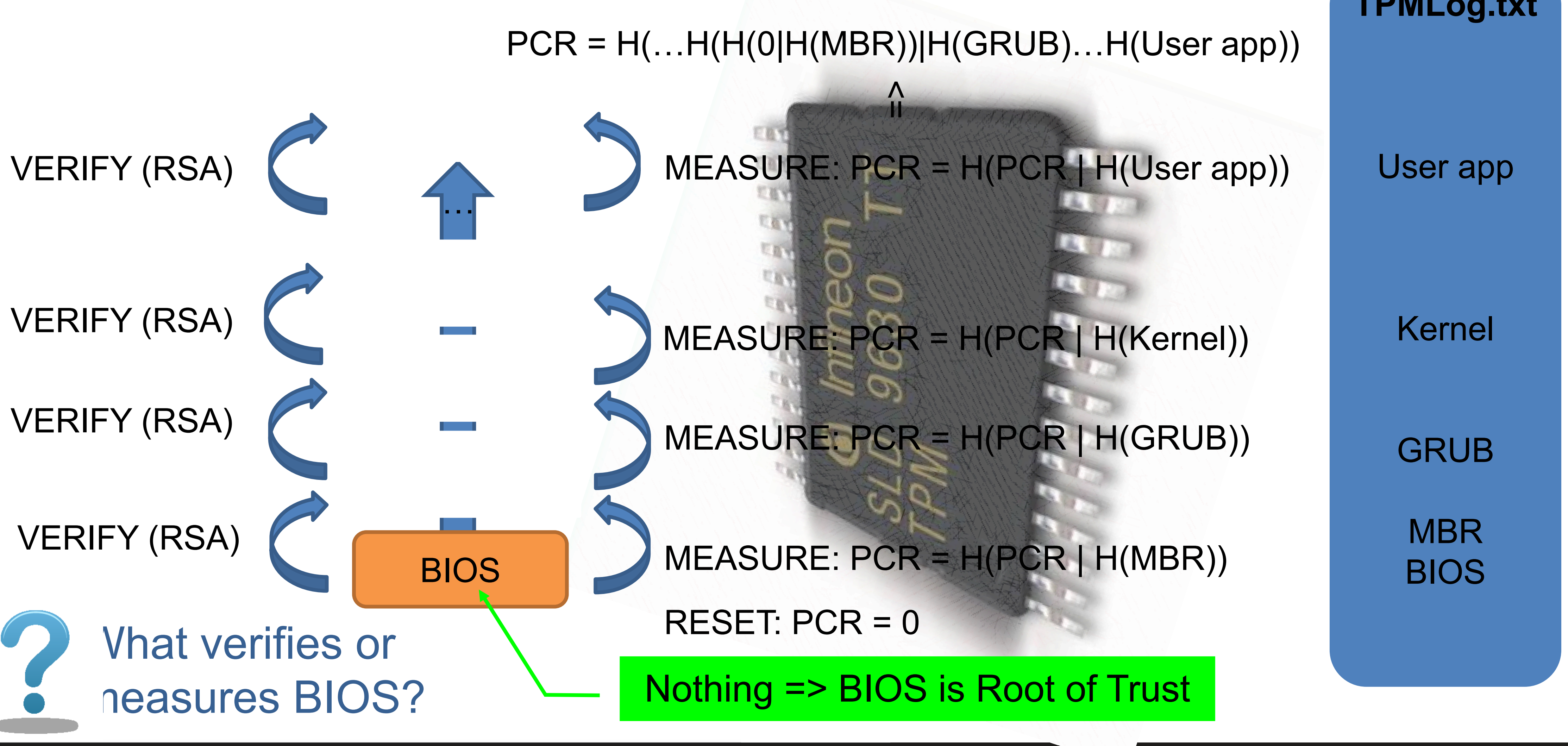
How to arrive at the expected chain of apps?

1. Just trust the whole boot process
2. Make all applications in protected read-only memory
   - If read-only => cannot be (maliciously) modified. But is it really what is running?
3. 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)
   - (but which particular apps were executed is not known, only that they were signed)
4. Create un-spoofable log what executed: Measured boot
   - Before next app is executed, its hash (“measurement”) is added to un-spoofable log (TPM’s PCR)
   - Will NOT prevent run of unwanted app, but environment cannot lie about what was executed
   - Requires (protected) log storage (Trusted Platform Module)
   - May require authentication of log (Remote attestation)

Trusted boot

Verified and measured approaches can be combined

"Verified" boot   "Measured" boot


PCR = H(…H(H(0|H(MBR))|H(GRUB)…H(User app))
VERIFY (RSA)
MEASURE: PCR = H(PCR | H(User app))
…
VERIFY (RSA)
MEASURE: PCR = H(PCR | H(Kernel))
VERIFY (RSA)
MEASURE: PCR = H(PCR | H(GRUB))
VERIFY (RSA)
BIOS
MEASURE: PCR = H(PCR | H(MBR))
RESET: PCR = 0
TPMLog.txt
User app
Kernel
GRUB
MBR
BIOS
Nothing => BIOS is Root of Trust

Vhat verifies or
neasures BIOS?

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, AMD Platform Security Processor
  - dynamic root of trust: Intel TXT, Intel SGX, Pluton
- Root of trust requires special protection
  - As nobody verifies than nobody will detect eventual modification of it

INTEL BOOT GUARD (IBG)

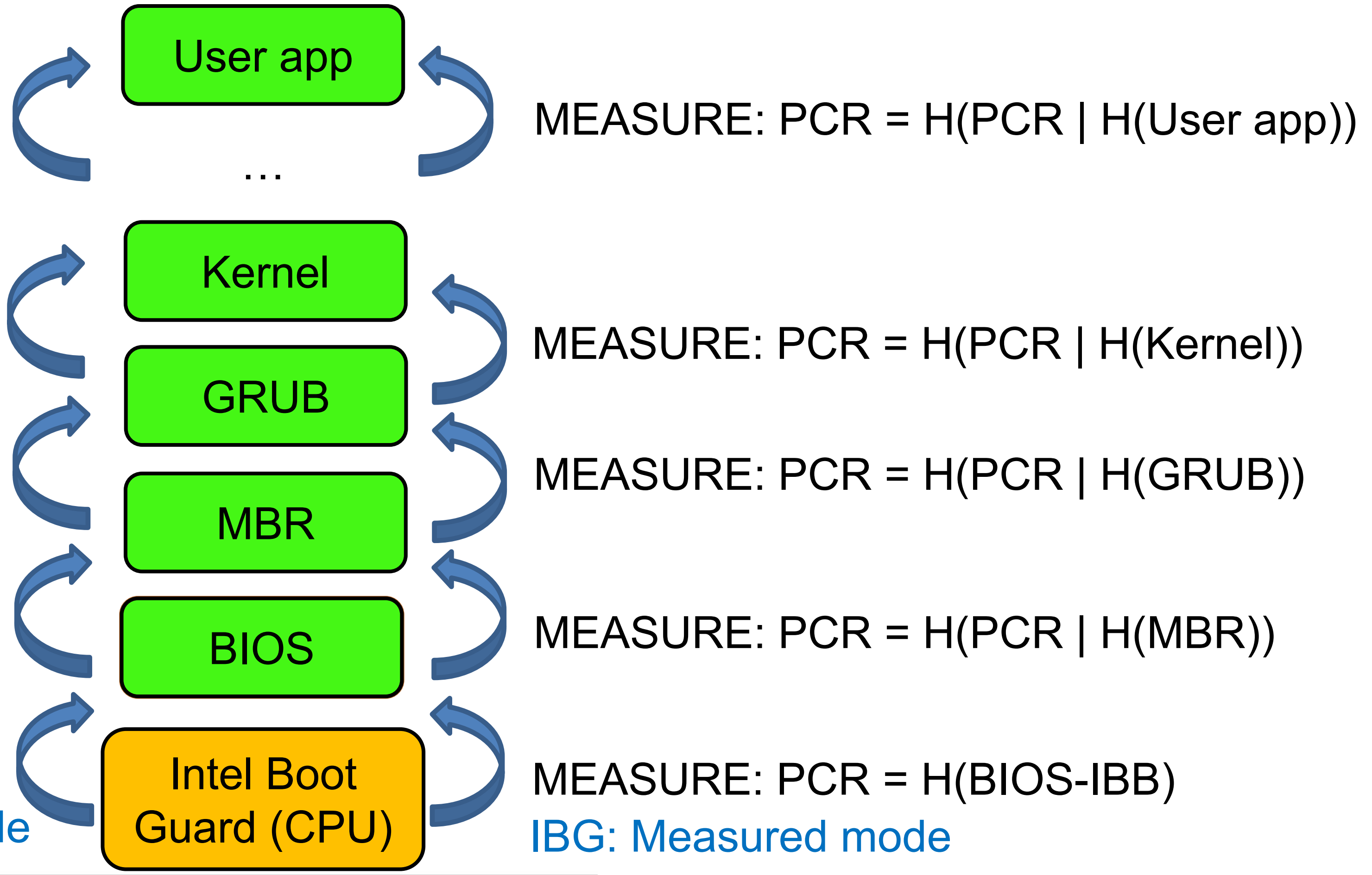
Intel Boot Guard (IBG)

- 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

AMD Platform Security Processor (PSP) provides same functionality as IBG



VERIFY (RSA)

VERIFY (RSA)

VERIFY (RSA)

VERIFY (RSA)

VERIFY (RSA)

IBG: Verified mode

MEASURE: PCR = H(PCR | H(User app))

MEASURE: PCR = H(PCR | H(Kernel))

MEASURE: PCR = H(PCR | H(GRUB))

MEASURE: PCR = H(PCR | H(MBR))

MEASURE: PCR = H(BIOS-IBB)

IBG: Measured mode

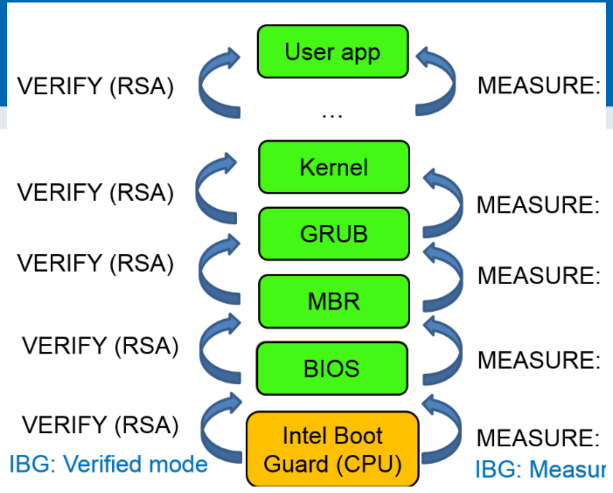
Intel Boot Guard – security improvements

- 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(s) 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 bootloaders (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 a signature!
- Trivial (potential) 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)



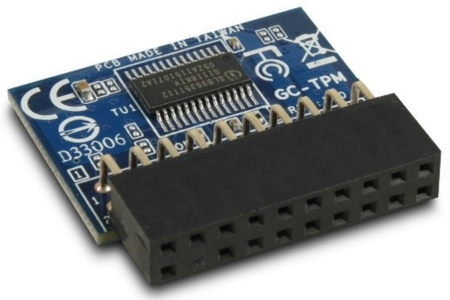
Short summary

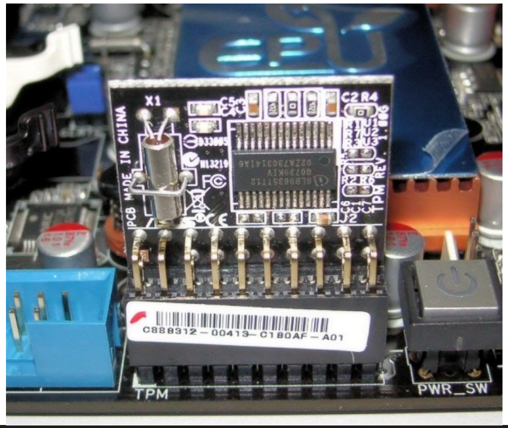
- Signature-based "verified" boot approach
  - Whitelisting approach – run only what is signed
  - Robust signature process needed (trust in private key owner)
  - Integrity of verification public key is critical
  - Key management is necessary (multiple keys, key updates)
- "Measured" boot approach
  - 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)
  - Intel Boot Guard/AMD Platform Security Processor inside CPU (signature of BIOS)

TRUSTED PLATFORM MODULE

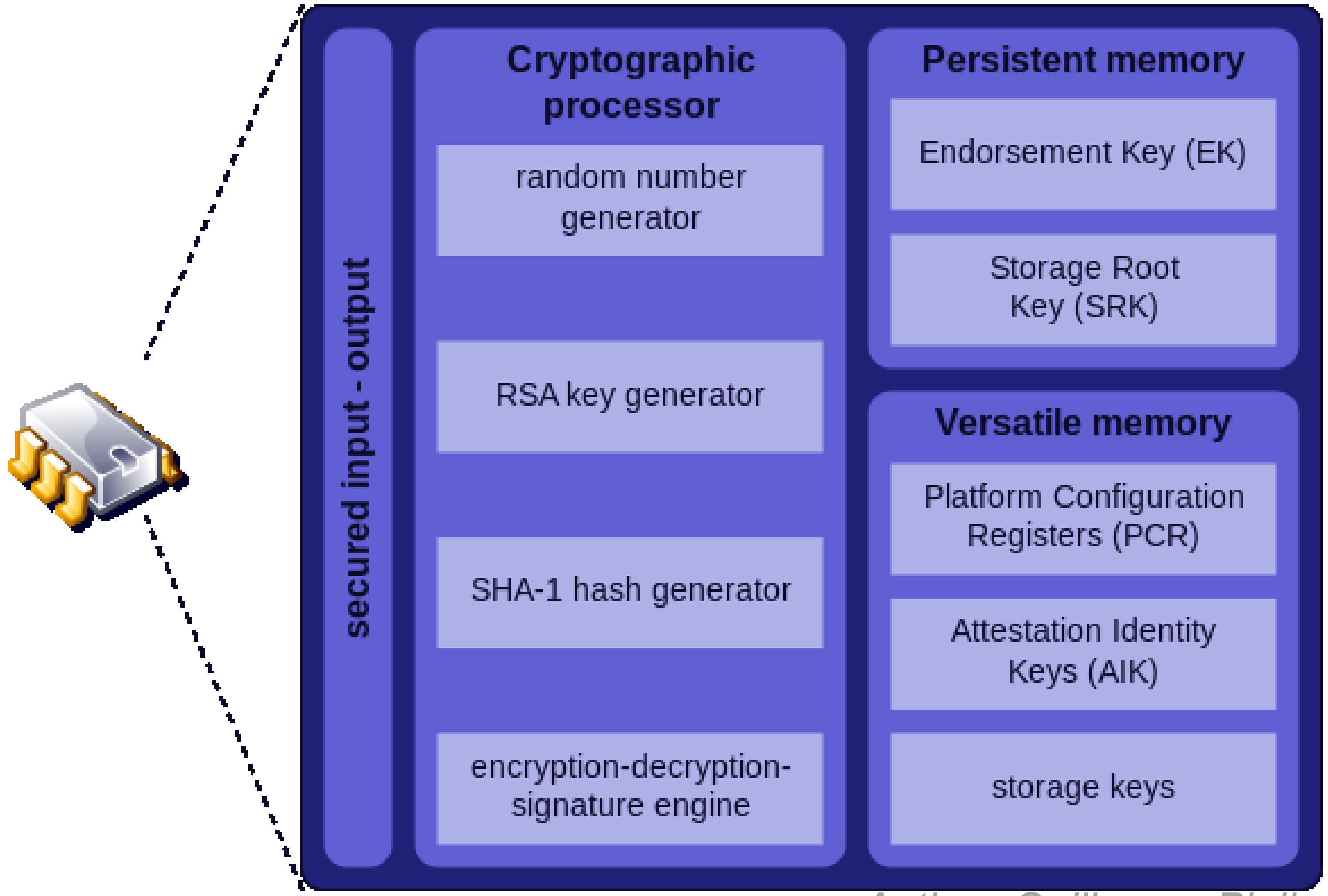
TPM hardware

- Cryptographic smart card connected/inside to device
  - Secure storage, cryptographic operations…
  - (But not programmable JavaCard ☺)
- Physical placement
  1. Additional chip on motherboard (discrete dTPM: Infineon, STM, Nuvoton)
  2. Firmware module inside CPU (firmware fTPM: Intel, AMD)
  3. Incorporated in CPU/peripheral (integrated iTPM: Pluton)
  4. (Software TPM – for development and debugging)
- Accessed during boot time
  - “Measured” boot (TPM’s PCR registers)
  - BitLocker encrypted drive keys
- Accessed later (private key operation)





Trusted platform module

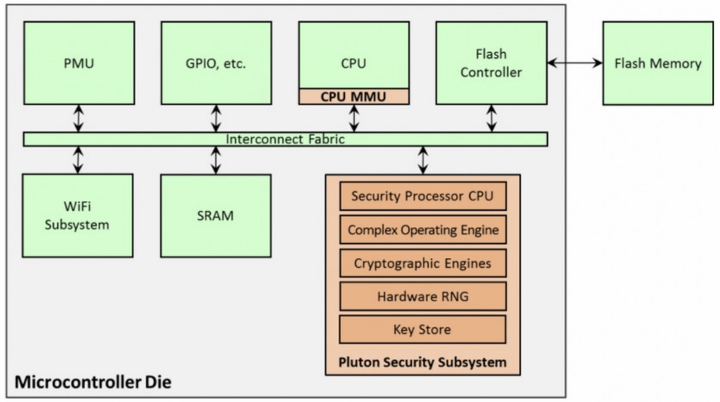


Author: Guillaume Piolle

Trusted Platform Module (TPM)

- ISO/IEC 11889 standard for secure crypto-processor
- Versions published by Trusted Computing Group
  - https://trustedcomputinggroup.org
  - TPM 1.2 (2003-2011)
  - TPM 2.0 (2013-now, not compatible with 1.2, but downgrade switch in BIOS)
- Tools to communicate with TPM
  - Windows: Microsoft PCPTool, TSS.MSR, Windows API
  - Linux: tpm_tools, tpm2_tools, GUI TPMManager

https://www.microsoft.com/en-us/research/wp-content/uploads/2017/03/SevenPropertiesofHighlySecureDevices.pdf



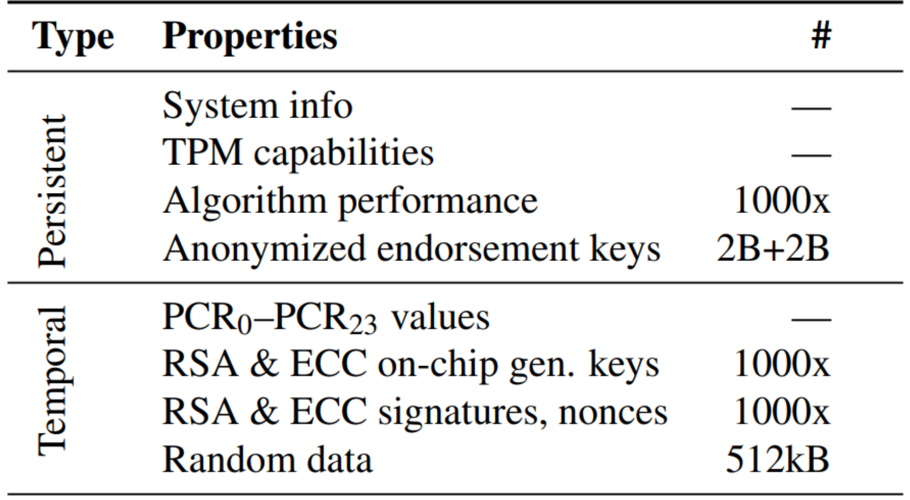
Pluton chip (iTPM)

- Hardware chip inside AMD and Qualcomm CPU/SoC silicon die
  - Co-developed by Microsoft, AMD and Qualcomm (Intel not yet)
  - Similar functionality like Secure Enclave or ARM TrustZone
    - own on-chip RAM, ROM, RNG, cryptographic co-processors…
    - Only Microsoft signed firmware (Windows Update), downgrade protection
  - On non-Windows systems provides only generic TPM 2.0 (iTPM)
- Used to implement TPM 2.0 functionality (integrated TPM => iTPM)
  - But also more, design originally from Microsoft Xbox (DRM) and Azure Sphere
  - SHACK (Secure Hardware Cryptography Key) implementation
  - DICE (Device Identifier Composition Engine) implementation
  - Robust Internet of Things (RIoT) specification compliance

TPMALGTEST PROJECT

CROCS

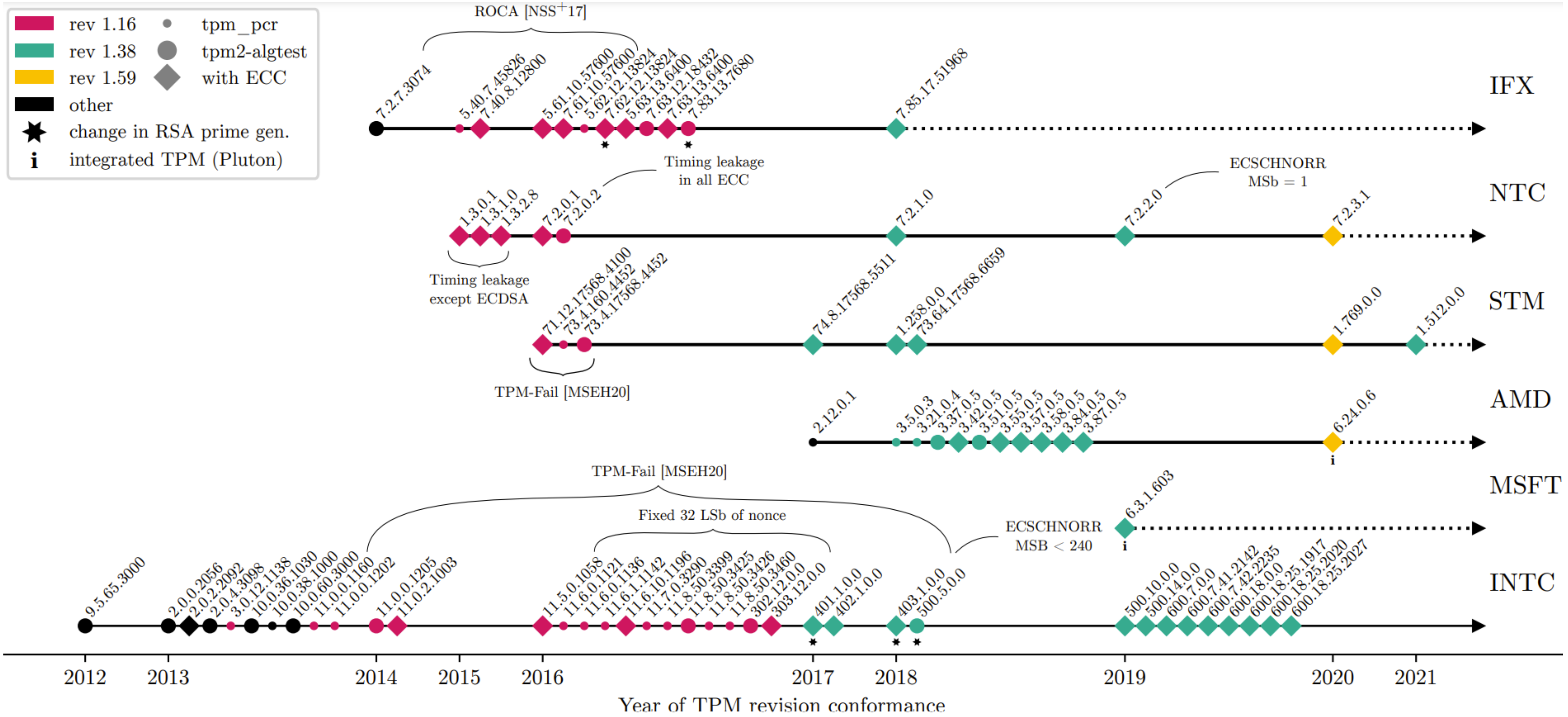
TPM analysis (TPMAlgTest project)



Type	Properties	#
Persistent	System info	—
	TPM capabilities	—
	Algorithm performance	1000x
	Anonymized endorsement keys	2B+2B
Temporal	PCR_0–PCR_{23} values	—
	RSA & ECC on-chip gen. keys	1000x
	RSA & ECC signatures, nonces	1000x
	Random data	512kB

- TPMAlgTest data collection tools
  - AMD, Intel, Infineon, Nuvoton, STM (total 80 TPM versions)
  - FI MU computers, compatibility testing cluster, community submissions
  - https://github.com/crocs-muni/tpm2-algtest

1. Algorithmic and performance support
2. Properties of cryptographic material (RSA and ECC keypairs)
   - Frequency of changes in cryptographic library
3. Properties of Endorsement keys (on-chip or injected)
4. Analysis of randomness data (GetRandom(), ECC keys and nonces…)



rev 1.16
rev 1.38
rev 1.59
other
change in RSA prime gen.
integrated TPM (Pluton)
tpm_pcr
tpm2-algtest
with ECC
ROCA [NSS+17]
7.2.7.3074
5.40.7.45826
7.40.8.12800
5.61.10.57600
7.61.10.57600
5.62.12.13824
7.62.12.13824
5.63.13.6400
7.63.12.18432
7.63.13.6400
7.83.13.7680
7.85.17.51968
IFX
Timing leakage in all ECC
ECSCHNORR MSb = 1
1.3.0.1
1.3.1.0
1.3.2.8
7.2.0.1
7.2.0.2
7.2.1.0
7.2.2.0
7.2.3.1
NTC
Timing leakage except ECDSA
71.12.17568.4100
73.4.160.4452
73.4.17568.4452
74.8.17568.5511
1.258.0.0
73.64.17568.6659
1.769.0.0
1.512.0.0
STM
TPM-Fail [MSEH20]
2.12.0.1
3.5.0.3
3.21.0.4
3.37.0.5
3.42.0.5
3.51.0.5
3.55.0.5
3.57.0.5
3.58.0.5
3.84.0.5
3.87.0.5
6.24.0.6
AMD
i
TPM-Fail [MSEH20]
Fixed 32 LSb of nonce
ECSCHNORR MSB < 240
6.3.1.603
i
MSFT
9.5.65.3000
2.0.0.2056
2.0.2.2092
2.0.4.3098
3.0.12.1138
10.0.36.1030
10.0.38.1000
10.0.60.3000
11.0.0.1160
11.0.0.1202
11.0.0.1205
11.0.2.1003
11.5.0.1058
11.6.0.1121
11.6.0.1136
11.6.1.1142
11.6.10.1196
11.7.0.3290
11.8.50.3399
11.8.50.3425
11.8.50.3426
11.8.50.3460
302.12.0.0
303.12.0.0
401.1.0.0
402.1.0.0
403.1.0.0
500.5.0.0
500.10.0.0
500.14.0.0
600.7.0.0
600.7.41.2142
600.7.42.2235
600.18.0.0
600.18.25.1917
600.18.25.2020
600.18.25.2027
INTC
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
Year of TPM revision conformance

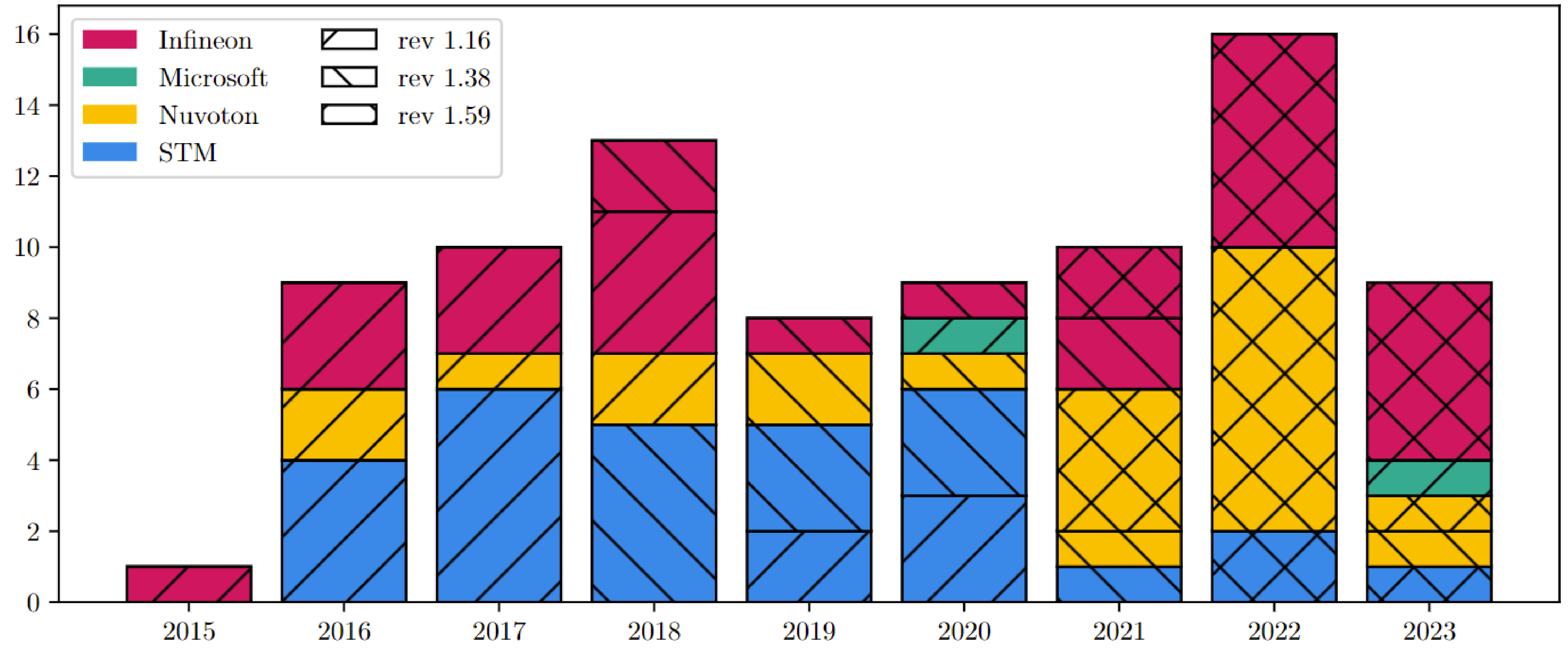


Figure 1: Number of TPM 2.0 certificates issued to vendors by year. The specification revision the certified TPM complies with is shown with a bar pattern.

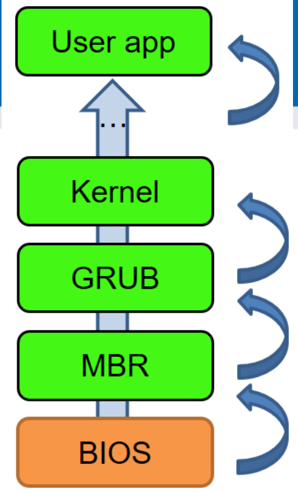
Security functions provided by TPM-based systems

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



Platform attestation – PCR registers

- **W:** PCPTool.exe GetPCRs
- **L:** cat `find /sys/class/ -name "tpm0"`/device/pcrs

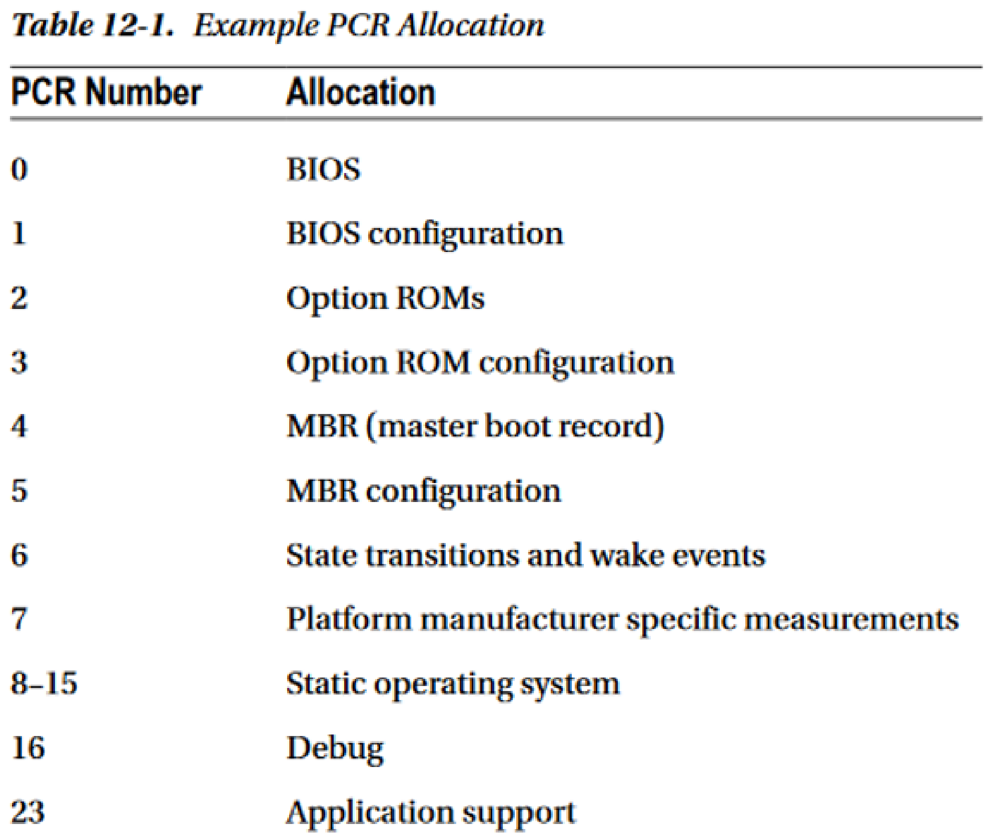
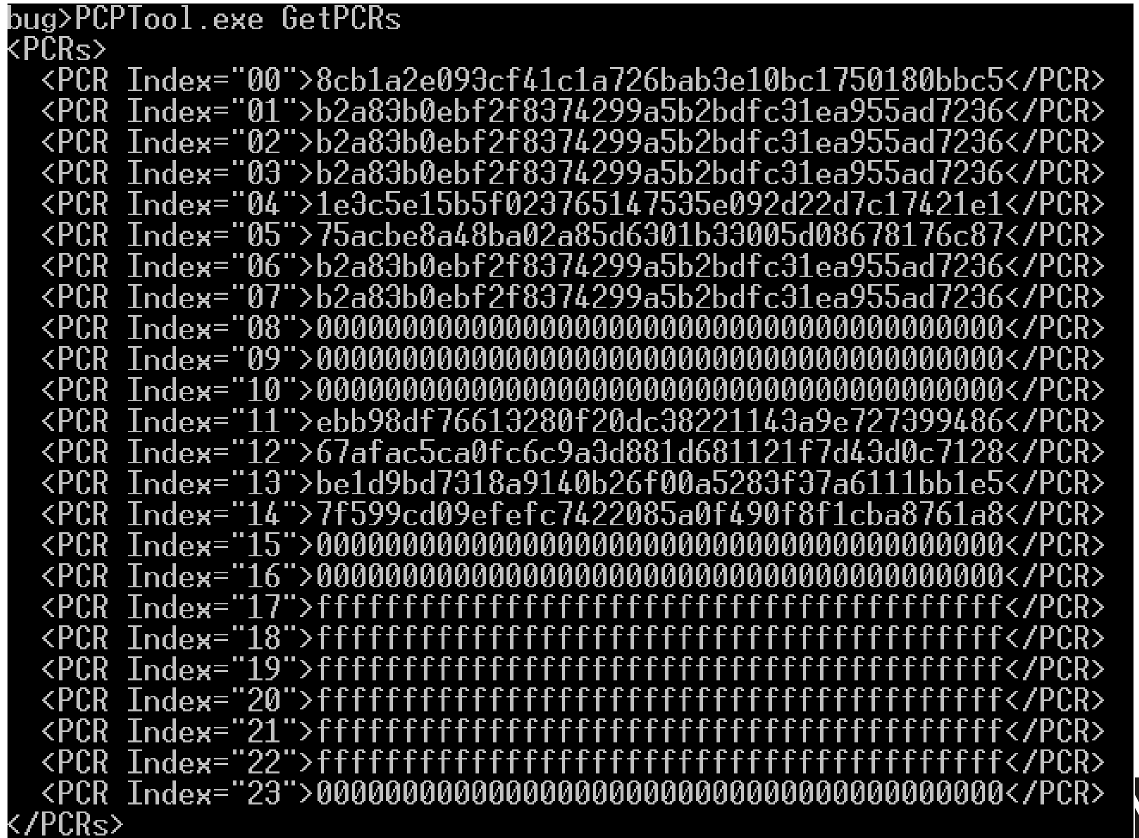


Table 12-1. Example PCR Allocation

PCR Number	Allocation
0	BIOS
1	BIOS configuration
2	Option ROMs
3	Option ROM configuration
4	MBR (master boot record)
5	MBR configuration
6	State transitions and wake events
7	Platform manufacturer specific measurements
8–15	Static operating system
16	Debug
23	Application support



```
bug>PCPTool.exe GetPCRs
<PCRs>
  <PCR Index="00">8cb1a2e093cf41c1a726bab3e10bc1750180bbc5</PCR>
  <PCR Index="01">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
  <PCR Index="02">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
  <PCR Index="03">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
  <PCR Index="04">1e3c5e15b5f023765147535e092d22d7c17421e1</PCR>
  <PCR Index="05">75acbe8a48ba02a85d6301b33005d08678176c87</PCR>
  <PCR Index="06">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
  <PCR Index="07">b2a83b0ebf2f8374299a5b2bdfc31ea955ad7236</PCR>
  <PCR Index="08">0000000000000000000000000000000000000000</PCR>
  <PCR Index="09">0000000000000000000000000000000000000000</PCR>
  <PCR Index="10">0000000000000000000000000000000000000000</PCR>
  <PCR Index="11">ebb98df76613280f20dc38221143a9e727399486</PCR>
  <PCR Index="12">67afac5ca0fc6c9a3d881d681121f7d43d0c7128</PCR>
  <PCR Index="13">be1d9bd7318a9140b26f00a5283f37a6111bb1e5</PCR>
  <PCR Index="14">7f599cd09efefc7422085a0f490f8f1cba8761a8</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>
</PCRs>
```

CROCS

Remote attestation of platform state

- So you measured your boot. How to prove your sta ote party?

- Idea:
  1. Take current PCR values (stored inside TPM)
  2. Sign it by TPM’s attestation private key (AIK), (all inside TPM)
  3. Send signed PCR values + TPMLog from computer to remote party
  4. Remote party holds public key and can verify signature => trust in authenticity of PCR values

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 this location
  - Key version prevents update to older (compromised) key
  - Firmware version prevents update to vulnerable firmware
- Store selected user’s private keys (secure storage)
- 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…)

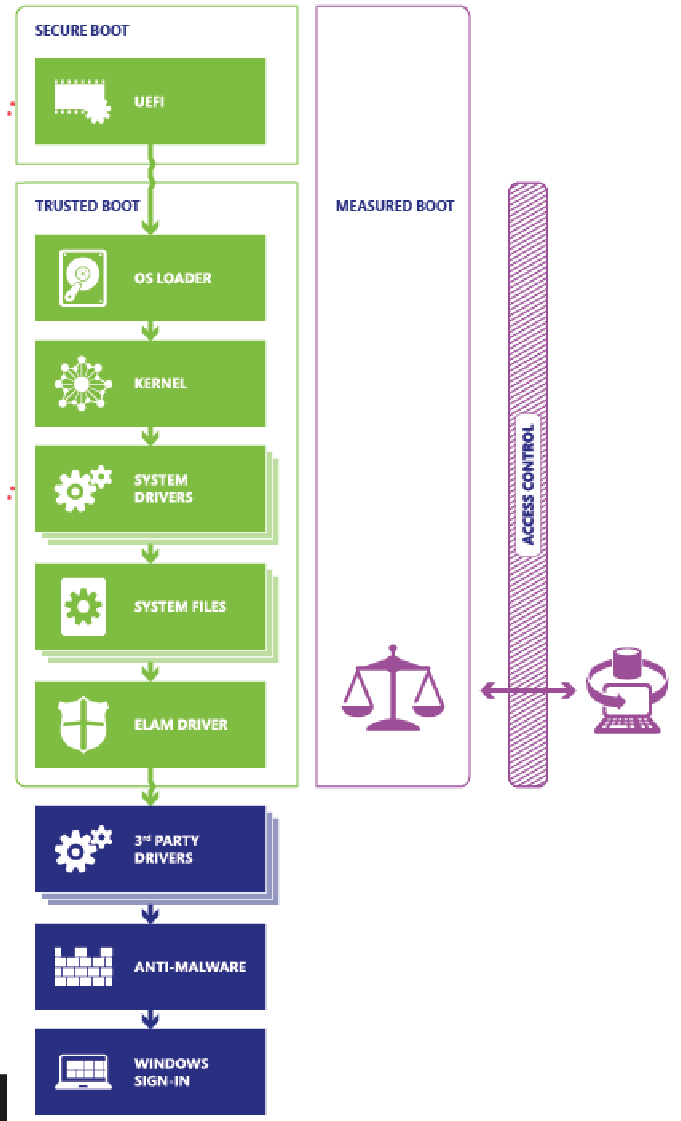
1. “Setup" mode – platform key (PK) is not loaded yet
   - Everybody can write its own platform key (become owner)
   - Once PK is written, switch to “user” mode
2. “User” mode
   - New keys (PKs, KEKs) can be written only if signed by PK
   - New software components loaded only if signed by KEKs

Secured and Trusted Boot

WINDOWS 8/10/11 TRUSTED BOOT

Windows 8/10 trusted boot

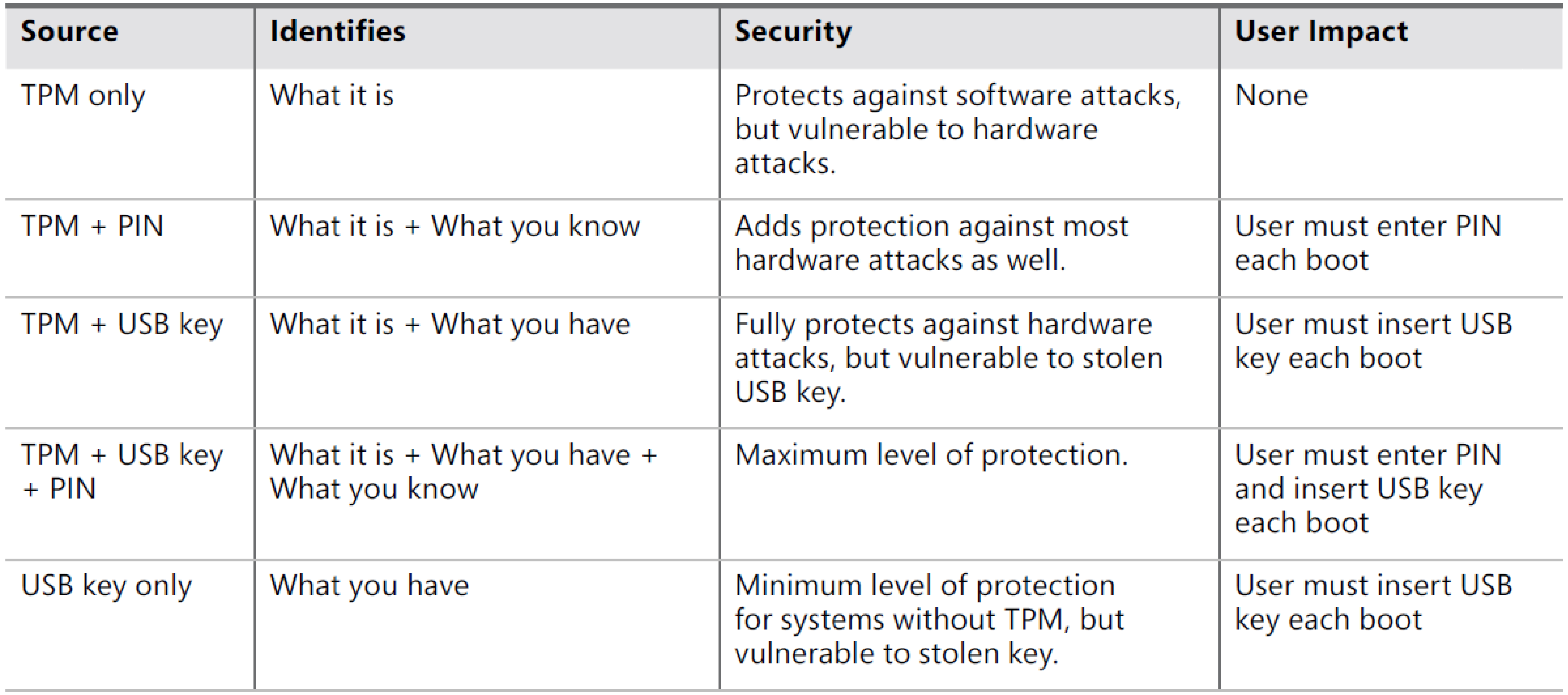
- Certified Windows 8/10/11 devices have trusted boot by default
  - "Verified" boot used (UEFI+OS sign)
  - "Measured" boot used (TPM)
- TPM PCRs used for measurements
- TPM used for keys protection
  - BitLocker disk encryption key
  - ROCA CVE-2017-15361 is relevant
    - If Infineon TPM used, patch!



http://technet.microsoft.com/en-US/windows/dn168167.aspx

Usage of TPM in BitLocker (disk encryption)

- Source of Volume Master Key (VMK)



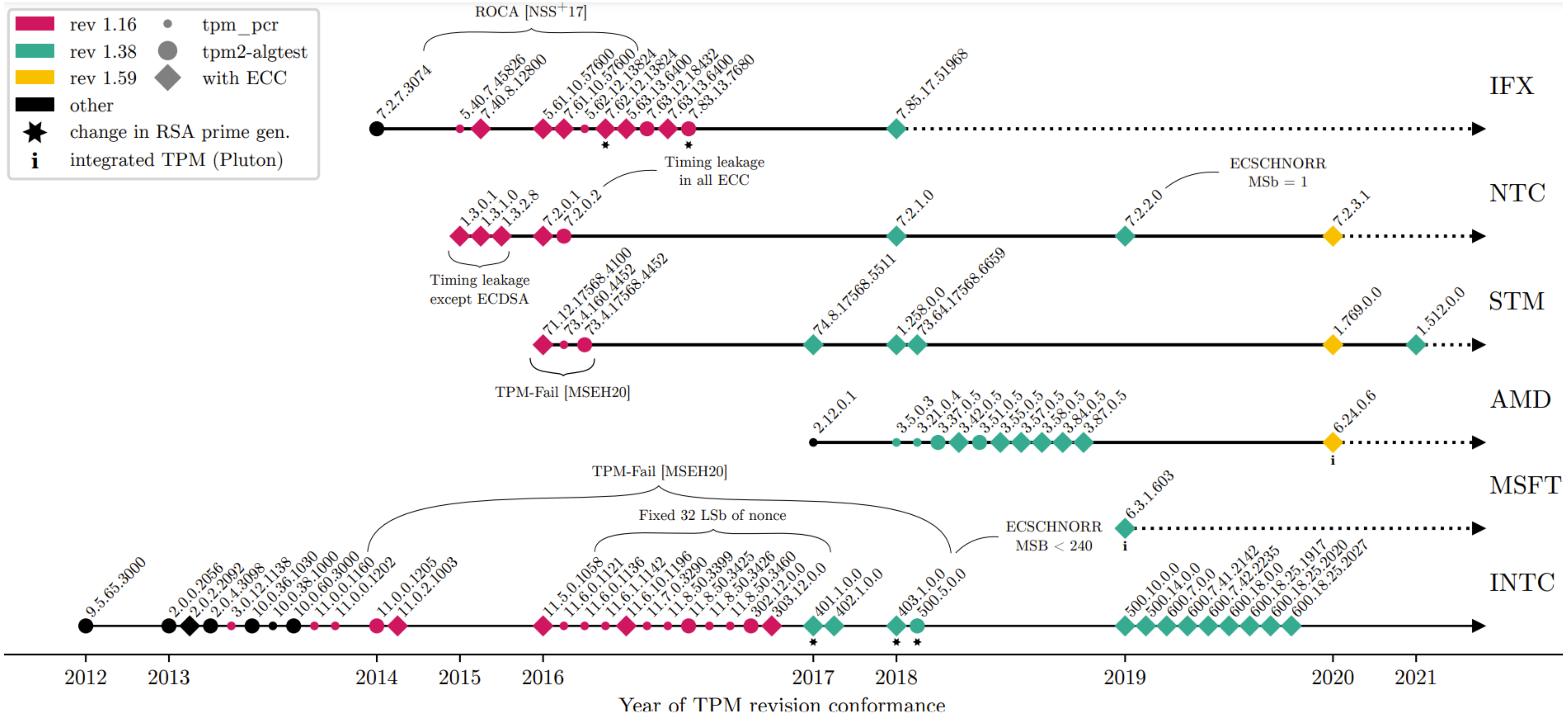
Source	Identifies	Security	User Impact
TPM only	What it is	Protects against software attacks, but vulnerable to hardware attacks.	None
TPM + PIN	What it is + What you know	Adds protection against most hardware attacks as well.	User must enter PIN each boot
TPM + USB key	What it is + What you have	Fully protects against hardware attacks, but vulnerable to stolen USB key.	User must insert USB key each boot
TPM + USB key + PIN	What it is + What you have + What you know	Maximum level of protection.	User must enter PIN and insert USB key each boot
USB key only	What you have	Minimum level of protection for systems without TPM, but vulnerable to stolen key.	User must insert USB key each boot

M. Russinovich et. al., Windows Internals Part 2, 6th Edition

ATTACKS AGAINST TPM

Attacks against systems with TPM

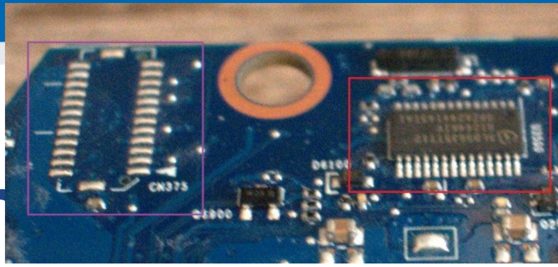
- Physical attacks
  - Sniffing, side-channels, fault induction…
- Design/reference implementation weaknesses
  - Buffer overflow in packet handling [2023], updated specification January 2024
    - *“Revision 98 Added parameter to MemoryMove(), MemoryCopy(), and MemoryConcat() to make sure that the data being moved will fix into the receiving buffer.”*
- Attacks against cryptographic implementations
  - ROCA [2017, CRoCS], RSA factorization (Infineon)
  - TPM-Fail vulnerability [2020], ECDSA nonce timing dependency (STM, Intel)
  - TPMScan vulnerabilities [2024, CRoCS]
    - Fixed low 4 bytes of ECDSA nonce, (older Intel fTPM)
    - TPM-Fail-like nonce timing in other algorithm than ECDSA (Nuvoton)


🟥 rev 1.16
🟩 rev 1.38
🟨 rev 1.59
⬛ other
● tpm_pcr
⬤ tpm2-algtest
◆ with ECC
★ change in RSA prime gen.
i integrated TPM (Pluton)
ROCA [NSS+17]
7.2.7.3074
5.40.7.45826
7.40.8.12800
5.61.10.57600
7.61.10.57600
5.62.12.13824
7.62.12.13824
5.63.13.6400
7.63.12.18432
7.63.13.6400
7.83.13.7680
7.85.17.51968
IFX
Timing leakage in all ECC
ECSCHNORR MSb = 1
1.3.0.1
1.3.1.0
1.3.2.8
7.2.0.1
7.2.0.2
7.2.1.0
7.2.2.0
7.2.3.1
NTC
Timing leakage except ECDSA
71.12.17568.4100
73.4.160.4452
73.4.17568.4452
74.8.17568.5511
1.258.0.0
73.64.17568.6659
1.769.0.0
1.512.0.0
STM
TPM-Fail [MSEH20]
2.12.0.1
3.5.0.3
3.21.0.4
3.37.0.5
3.42.0.5
3.51.0.5
3.55.0.5
3.57.0.5
3.58.0.5
3.84.0.5
3.87.0.5
6.24.0.6
AMD
i
TPM-Fail [MSEH20]
Fixed 32 LSb of nonce
ECSCHNORR MSB < 240
6.3.1.603
i
MSFT
9.5.65.3000
2.0.0.2056
2.0.2.2092
2.0.4.3098
3.0.12.1138
10.0.36.1030
10.0.38.1000
10.0.60.3000
11.0.0.1160
11.0.0.1202
11.0.0.1205
11.0.2.1003
11.5.0.1058
11.6.0.1121
11.6.0.1136
11.6.1.1142
11.6.10.1196
11.7.0.3290
11.8.50.3399
11.8.50.3425
11.8.50.3426
11.8.50.3460
302.12.0.0
303.12.0.0
401.1.0.0
402.1.0.0
403.1.0.0
500.5.0.0
500.10.0.0
500.14.0.0
600.7.0.0
600.7.41.2142
600.7.42.2235
600.18.0.0
600.18.25.1917
600.18.25.2020
600.18.25.2027
INTC
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
Year of TPM revision conformance

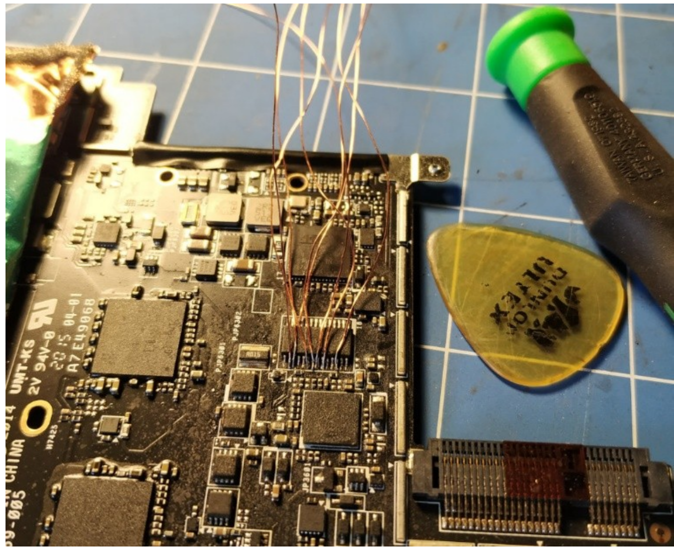
Research paper

- Paper written by CROCS and NUKIB [CHES'24]
  - https://crocs.fi.muni.cz/papers/tpm_ches2024
- Several ECC nonce-related vulnerabilities discovered
  - Known vulnerabilities by TPM-Fail (2019) – Intel, STM, Nuvoton
    - Few topmost bits leaked via timing, ~1000s signatures to recover key
  - Previously unreported vulnerabilities in ECSCHNORR and ECDAA
    - inconsistent testing and reporting
- New serious vulnerability in older Intel fTPMs 11.5.0.1058-303.12.0.0
  - Lowest bytes of nonces of ECDSA and ECSCHNORR fixed to 0x00000001
  - Only nine signatures required to extract private key, no need for active observation
  - Fixed in 400.x versions, but not publicly disclosed

Attack: Sniffing commands/keys for BitLocker



- Nice writeup how to sniff BitLocker key when send from TPM to OS, then decrypt disk image
  - https://pulsesecurity.co.nz/articles/TPM-sniffing



- fTPM and iTPM does not have exposed bus

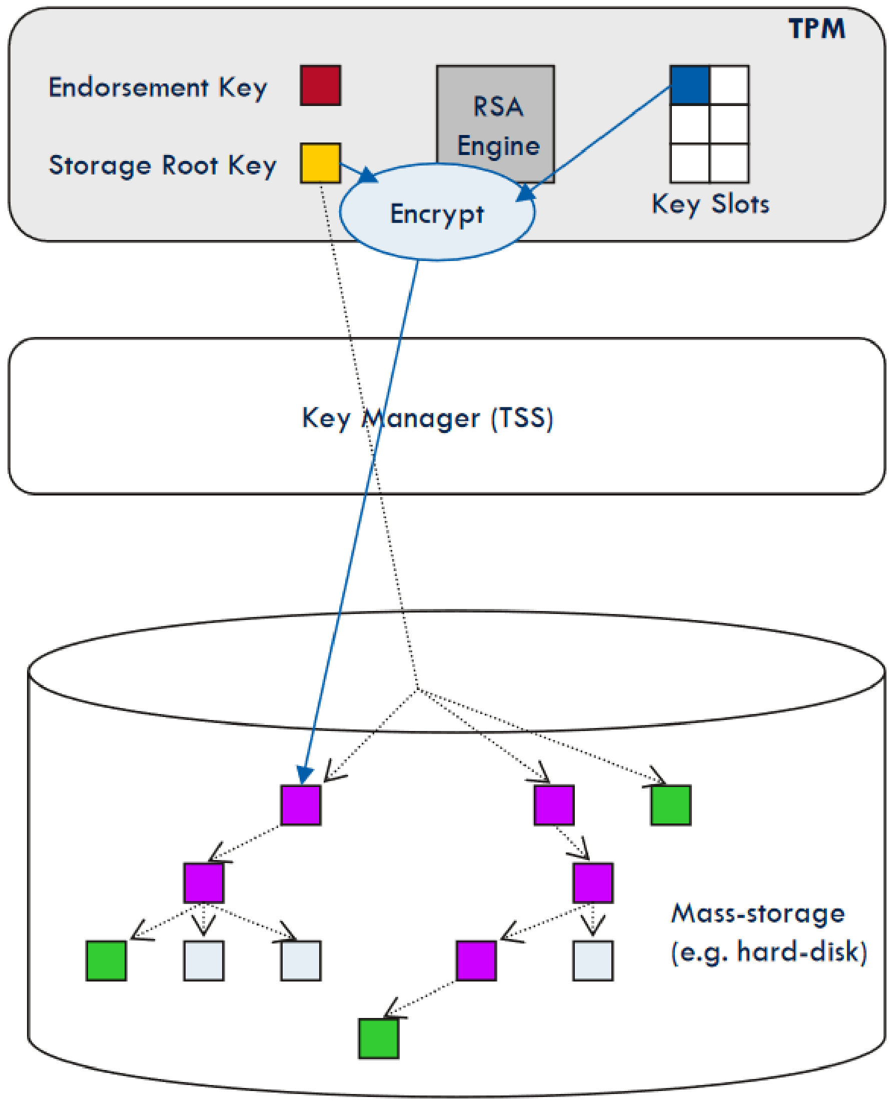
BASIC COMPONENTS

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: do not 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

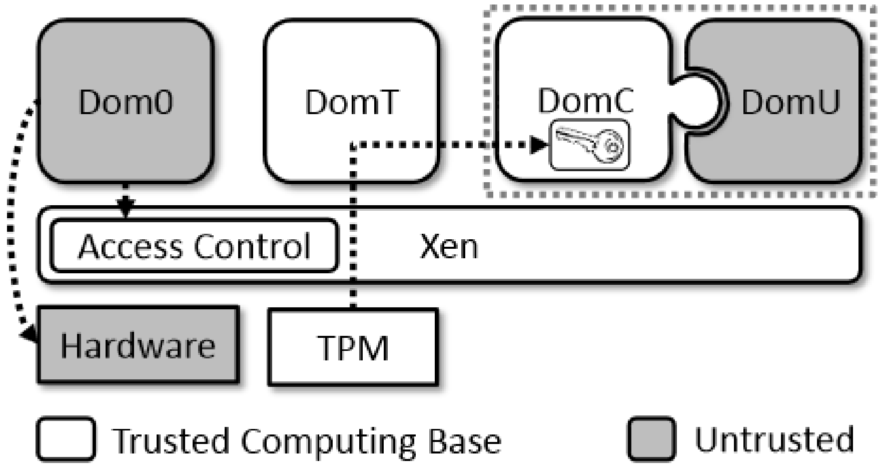
- TPM releases secret only when PCR contains particular value
- Enforcement even in measured-only mode
  - Key is not released if unexpected component was started (started => is included in measurements)
- Conditions can use ANDs and ORs
- How to handle policy updates?
  - Change policy of state only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

Programming with TPM

- The TPM Software Stack from Microsoft Research (C++, Java, C#, Python)
  - https://github.com/Microsoft/TSS.MSR
- tpm2-tools
  - Open-source TPM stack for Linux and Windows
  - https://github.com/tpm2-software/tpm2-tools

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)
- Several different implementations
  - E.g., Red Hat keylime https://github.com/keylime/



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 (trusted boot) only after reboot?
  - Takes relatively long time
  - Can we execute the same process, but dynamically?
  - Can we exclude long chain (BIOS, PCI…)?
    - Long chain => large Trusted Computing Base (TCB)!

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 (not used much in practice, server CPUs typically)
  - Intel’s SGX (all Skylake processors and newer, from 2015)

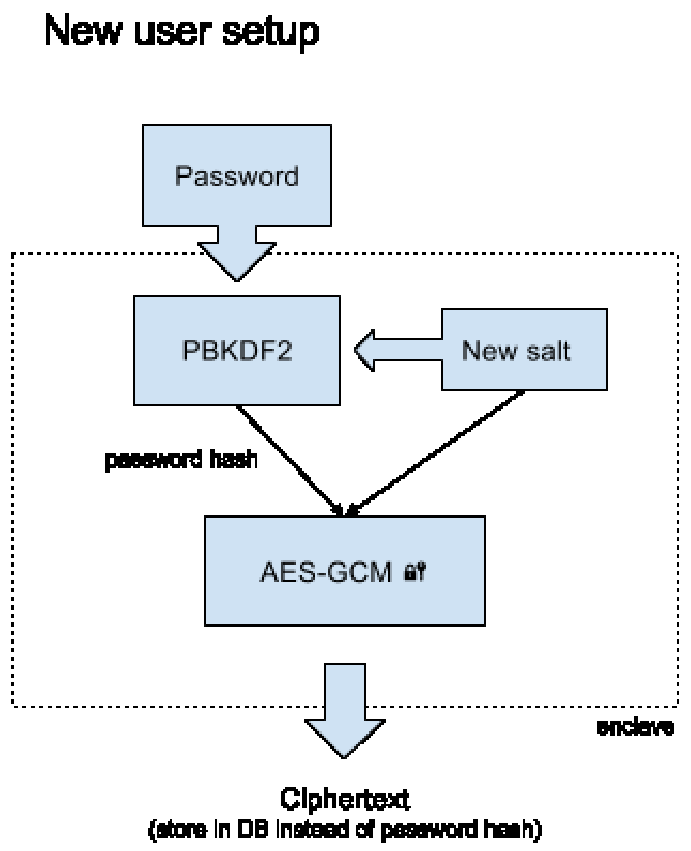
Intel’s SGX : Security enclave

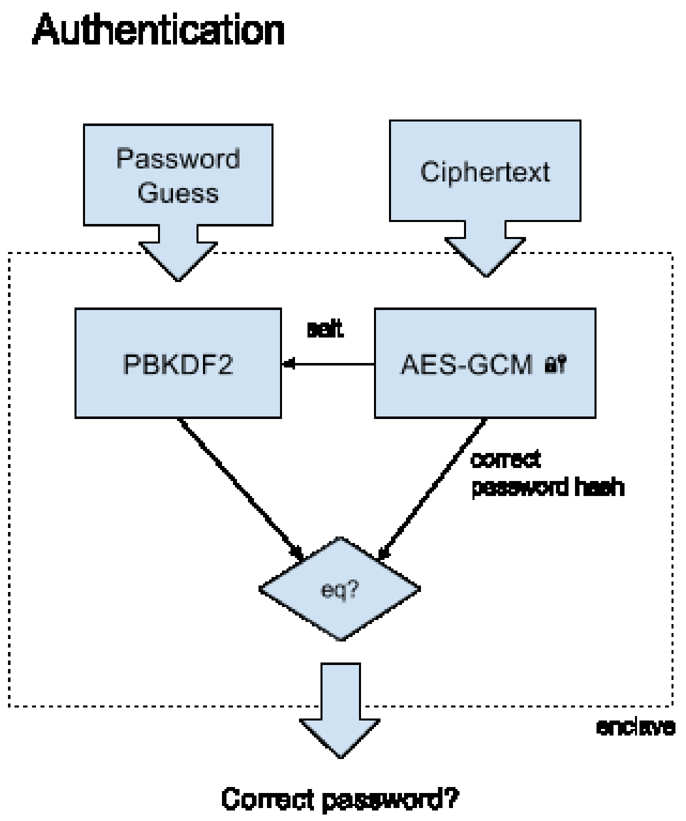
- Intel’s Software Guard Extension (SGX)
  - New set of CPU instructions intended for future cloud server CPUs
- Protection against privileged attacker
  - Server admin with physical access, privileged malware
- Application requests private region of code and data
  - Security enclave (4KB for heap, 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
- Trusted Computing Base significantly limited! ☺
  - But proprietary Intel code inside CPU ☹

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

Recent attacks against SGX

- SGX is not a silver bullet
- Vulnerable to side-channels
  - Attacker with physical access explicitly excluded from attacker model
  - Impacted by Spectre attack (2017)
    - https://github.com/lsds/spectre-attack-sgx
    - https://github.com/osusecLab/SgxPectre
  - Impacted by Foreshadow attack (CVE-2018-3615) https://foreshadowattack.eu/
    - Reading out attestation private key
- Bugs of enclave code are still problem (developer)
- Not everything is running inside enclave (other code, user input…)

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

Intel SGX deprecated on non-server CPUs (end 2021)

- Intel deprecated technology for the newest non-server CPUs
  - Still present in server CPUs, utilized by Azure confidential computing…
- Not completely clear reasons so far
  - Possibly mix of many past attacks which cannot be fixed without changing the architecture significantly (and breaking compatibility)
- https://community.intel.com/t5/Intel-Software-Guard-Extensions/Intel-SGX-deprecated-in-11th-Gen-processors/m-p/1351848
- https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/001/deprecated-technologies/

TRUSTED COMPUTING - CRITIQUE

Trusted Computing (TC) - 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

Quo Vadis, TPM?

- ~2004: Started with primarily aim at DRM enforcement (TPM 1.2)
  - Some adoption, but also controversy, unclear future
- ~2013: TPM 2.0 significantly renewed interest and scope of use
  - Wide hardware support via certified dTPMs (Infineon, Nuvoton, STM) and non-certified fTPMs (Intel, AMD)
  - Microsoft Windows 11 requires TPM presence (measured boot, Bitlocker)
  - Linux systemd rapidly adds measured boot https://systemd.io/TPM2_PCR_MEASUREMENTS/
- ~2017: Support for TPM-based functions more common
- ~2022: Pluton chip (Microsoft + AMD & Qualcomm), iTPM
  - iTPM implementation (certification in progress), difficult to sniff TPM bus
  - Directly updatable via Windows Update

Summary

- Two principal solutions for trusted boot
  - Verified boot (signatures) and Measured boot (PCR+RA)
- 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)
- Size of Trusted Computing Base matters (TPM/SGX)
- Controversy about implication of trusted boot
  - Who owns and control target platform