CIS 4360
Secure Computer Systems

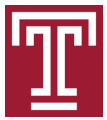
Secured System Boot

Professor Qiang Zeng

Spring 2017

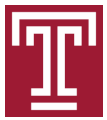
Previous Class

- Attacks against System Boot
  - Bootkit
  - Evil Maid Attack
  - Bios-kit
- Attacks against RAM
  - DMA Attack
  - Cold Boot Attack



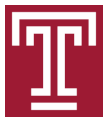
Outline

- UEFI Secure Boot
- Windows's Trusted Boot
- Intel's Trusted Boot



UEFI Secure Boot

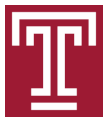
- Part of UEFI Spec.
  - UEFI Version 2.3.1, Errata C, 2012
- Secure Boot is a technology where the System Firmware checks that all other pre-OS code is **signed** with an authorized key and prevents the execution of unsigned code
  - A type of Boot Path Validation technologies
  - Pre-OS code: f/w drivers, option ROMs, UEFI drivers on disk, UEFI applications, or UEFI boot loaders
- PKI is at the core of the security for Secure Boot



Question

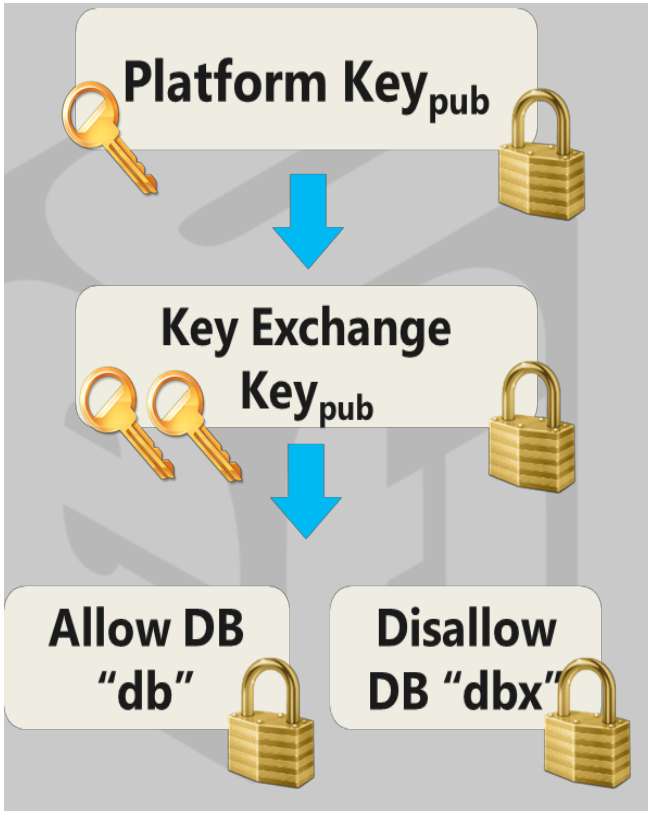
Does UEFI Secure Boot rely on TPM?

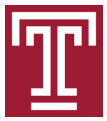
No, UEFI Secure Boot does not rely on TPM. It makes use of several public keys stored in non-volatile memory and these keys are protected against remote attacks



Keys in UEFI Secure Boot

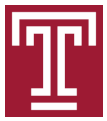
- Platform Key (PK)
  - Allows modification of KEK
  - Initialized by the h/w vendor
- Key Exchange Key (KEK)
  - Allows modification of db and dbx
  - Can be multiple
- Authorized Database (db)
  - Legal CAs, signatures and hashes
- Forbidden Database (dbx)
  - Illegal CAs, signatures and hashes





Authorized Code Only

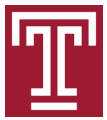
- If Secure Boot is enabled, it will only execute binaries which
  - has a sha-256 hash in db and not in dbx or
  - has a signature and that signature is in db but not in dbx or
  - has a signature signed by either a key in KEK or db and the key does not appear in dbx



Question

Examine Bios-kits and Bootkits under Secure Boot

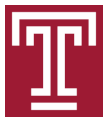
Neither Bios-kits nor Bootkits will succeed under Secure Boot, as the System Firmware will notice that the firmware or boot loader code is unauthorized.



Question

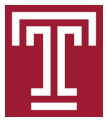
What should you do if you want your system to execute a custom boot loader

You can
(1) add a hash value to db, or
(2) add a key to db and use that key- to sign your binary, or
(3) add a KEK key and use that key- to sign your binary



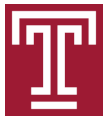
Two Modes

- In Setup Mode, all the keys can be altered arbitrarily and Secure Boot is off in this mode
- In User Mode, all key changes are authorized
  - E.g., an update of KEK should be singed by PK^{-}
  - E.g., a change of db should be signed by a key in KEK
- Only firmware administrator can enter the Setup mode; remote attackers cannot enter this mode



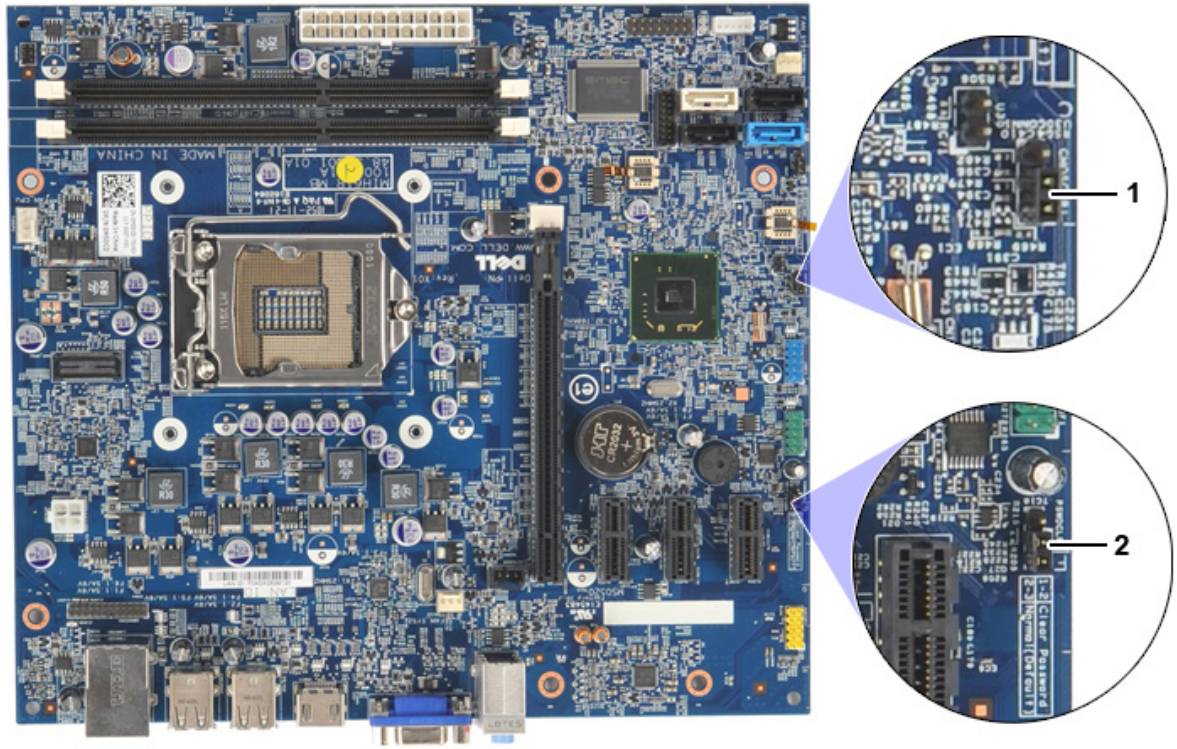
How Secure is "Secure Boot"

- Secure Boot is indeed very effective for fighting firmware rootkits and bootkits. But how secure is it?
- The integrity of System Firmware is obviously critical. It relies on an Authenticated BIOS Update mechanism
  - NIST 800-147 (a well-written document; highly recommended!)
  - A public key is contained and protected as part of the firmware
  - The firmware on the system flash checks the signature of the new firmware to be installed. So this is not a problem nowadays.
- How about the Firmware Administrator Password (recall that the admin can disable Secure Boot or change the keys)?
  - It is largely a joke, as most vendors allow you to use the jumper trick to clear Firmware Passwords easily. A physical lock may be needed to provide some
  - But Mac is an exception. Since 2011, it uses a dedicated chip (Atmel) to store and protect firmware passwords



How to Clear Firmware Passwords (on most modern computers)

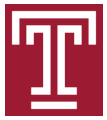
- Dell, as an example, gives detailed instructions on its website about how to reset Firmware Passwords



Question

How will you attack your roomate's PCs?

Install a Hardware-based Keylogger
DMA Attack
Cold Boot Attack
Evil Maid Attack
…

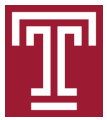


Question

How to launch the Evil Maid Attack against a computer protected by Secure Boot?

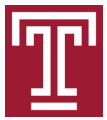
If you simply overwrite the disk with a malicious boot loader, due to Secure Boot, the System Firmware will simply refuse to execute the malicious boot loader. So you need to

(1) Clear the firmware password and enter the Setup Mode of Secure Boot
(2) Add a hash of the malicious boot loader to db
(3) Enter the User Mode of Secure Boot



Outline

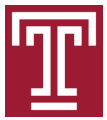
- UEFI Secure Boot
- Windows's Trusted Boot
- Intel's Trusted Boot



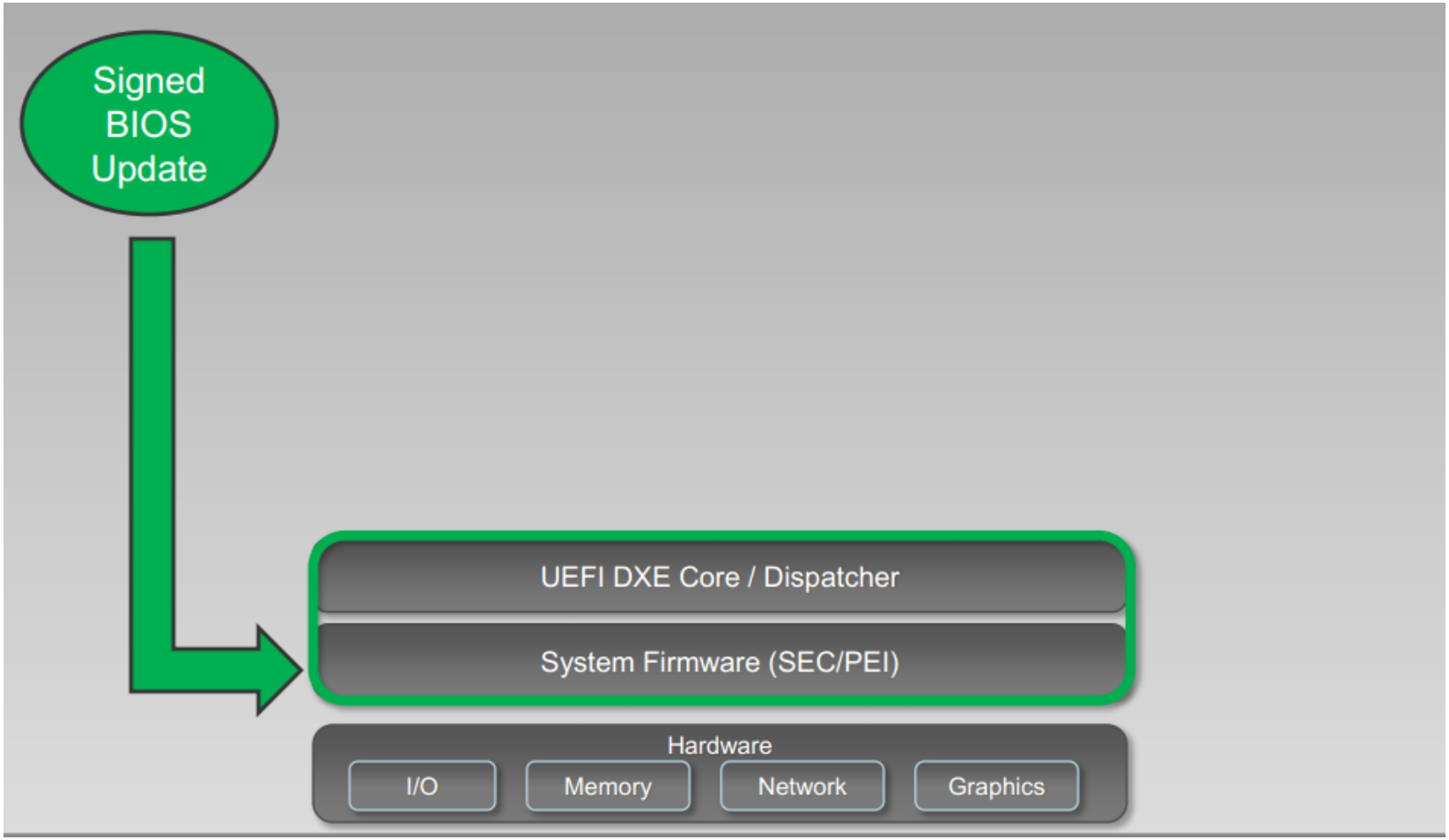
With Secure Boot, Firmware and Boot Loaders now can be trusted (at least against remote attacks), but how about drivers in the kernel mode?

Windows 8 Boot Process

- It is secured in three phases
  - Secure Boot: it ensures only authorized firmware and boot loaders can be executed
  - Trusted Boot: the boot loader loads a trusted Windows 8 OS loader, which loads a trusted kernel, and the kernel in turn loads authenticated Windows components including the ELAM drive
  - ELAM (Early Launch Anti Malware): before any third-party driver is loaded, a trusted kernel-mode anti-malware program is loaded and runs first; it checks all non-Microsoft boots drivers
    - Starting with Windows 10, version 1607, Windows will not load any new kernel mode drivers which are not signed by Microsoft
- Windows 8 also supports Measured Boot and Remote Attestation



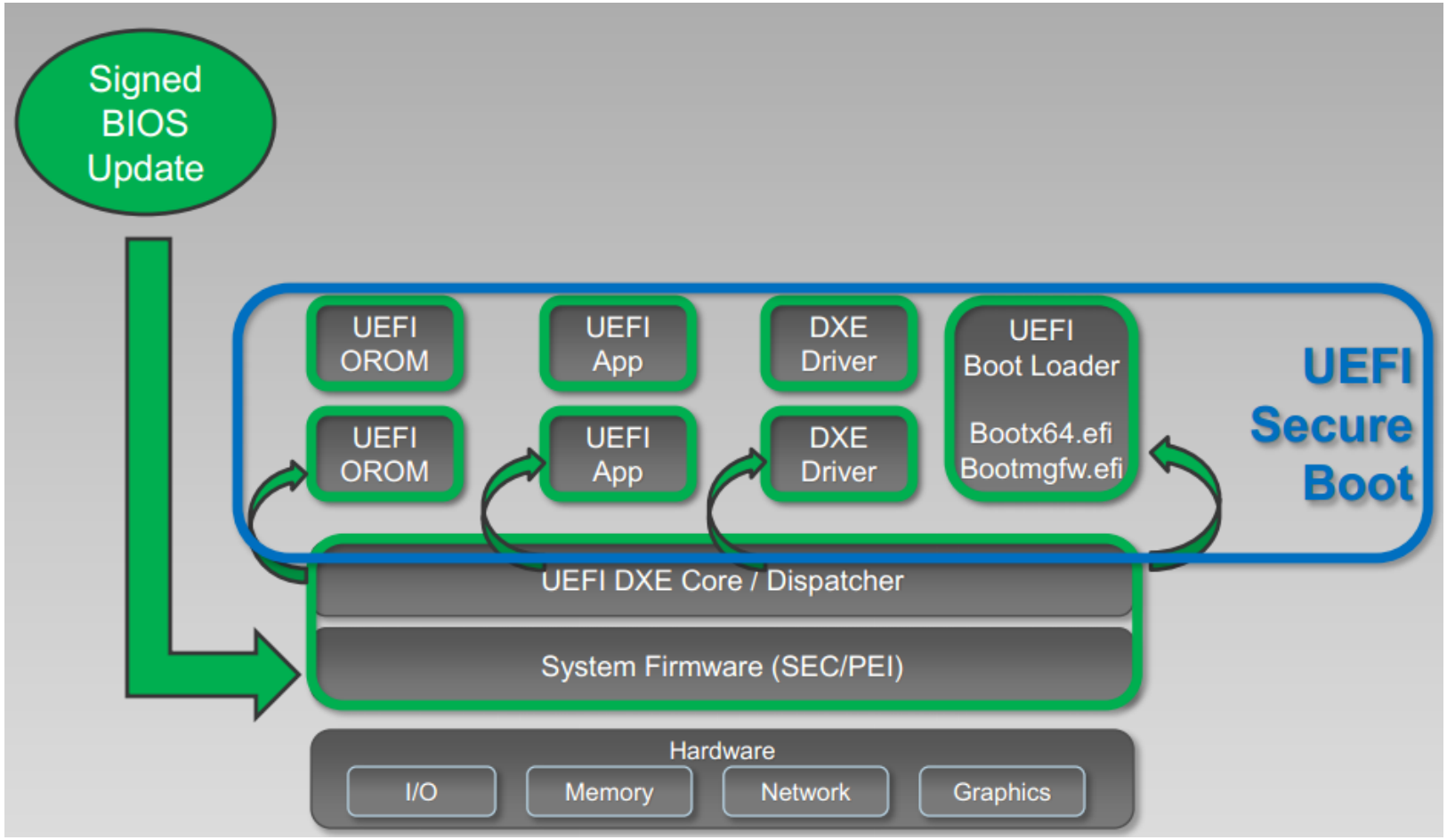
Firmware Signing



- Flash-based UEFI components are verified only during the update process when the whole BIOS image has its signature verified

http://c7zero.info/stuff/Windows8SecureBoot_Bulygin-Furtak-Bazhniuk_BHUSA2013.pdf

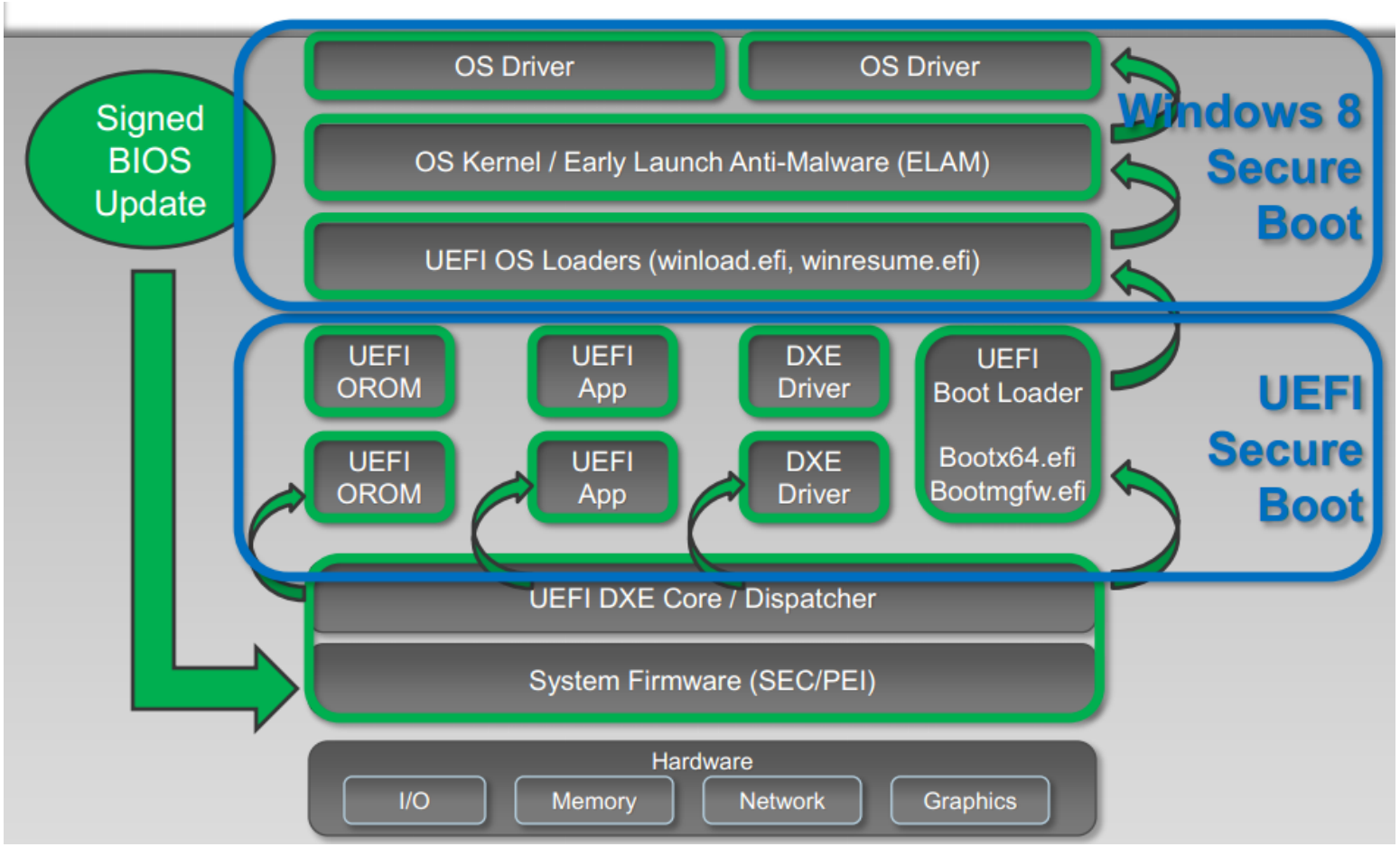
UEFI Secure Boot



- DXE verifies non-embedded XROMs, DXE drivers, UEFI applications and boot loader(s)
- This is the UEFI Secure Boot process

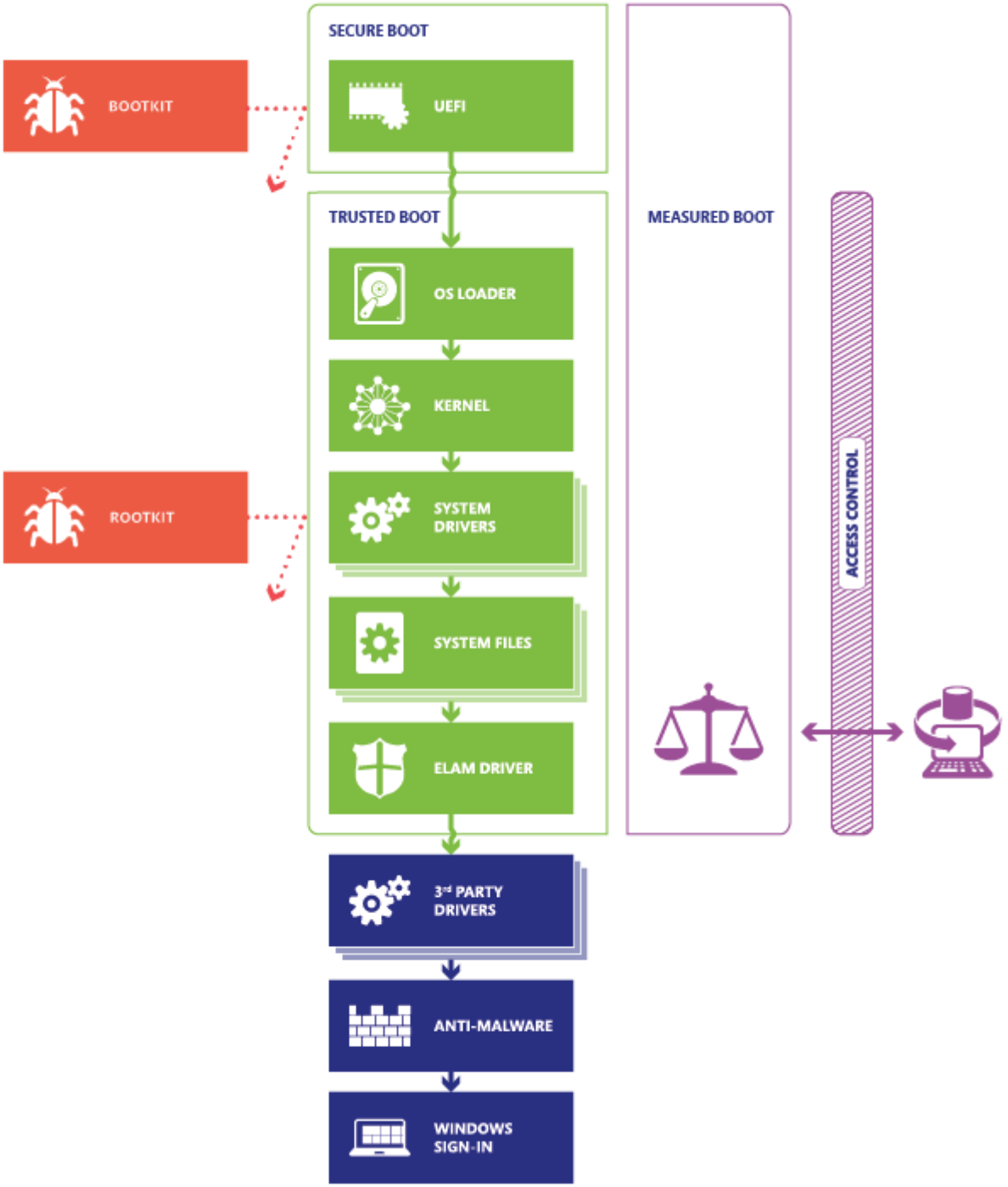
http://c7zero.info/stuff/Windows8SecureBoot_Bulygin-Furtak-Bazhniuk_BHUSA2013.pdf

Windows 8 Secure Boot



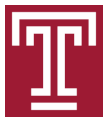
- Microsoft Windows 8 adds to the UEFI secure boot process
- Establishes a chain of verification
- UEFI Boot Loader -> OS Loader -> OS Kernel -> OS Drivers

http://c7zero.info/stuff/Windows8SecureBoot_Bulygin-Furtak-Bazhniuk_BHUSA2013.pdf


SECURE BOOT
BOOTKIT
UEFI
TRUSTED BOOT
MEASURED BOOT
OS LOADER
KERNEL
ROOTKIT
SYSTEM DRIVERS
SYSTEM FILES
ELAM DRIVER
ACCESS CONTROL
3rd PARTY DRIVERS
ANTI-MALWARE
WINDOWS SIGN-IN

All X-86 PCs with Windows Logo

- They must have Secure Boot enabled by default
- They must trust Microsoft's certificate (and thus any bootloader Microsoft has signed).
- They must allow the user to configure Secure Boot to trust other bootloaders.
- They must allow the user to completely disable Secure Boot
  - Windows RT devices based on ARM cannot disable Secure Boot, though

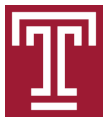


Keys in Windows' Certified PCs

Key/db Name	Variable	Owner	Notes
PKpub	PK (platform key)	OEM	
Microsoft Corporation KEK CA 2011	KEK (Key Exchange Key)	Microsoft	Allows updates to db and dbx:
Microsoft Windows Production CA 2011	db	Microsoft	This CA in the Signature Database (db) allows Windows to boot:
Microsoft UEFI driver signing CA	db	Microsoft	Microsoft signer for 3rd party UEFI drivers and Apps signed through the DevCenter program

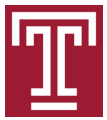
Sadly, for Linux Boot Loaders

- On the other hand, nowadays almost all PCs contain Microsoft's keys
- The PCs would simply refuse to boot Linux if its boot loader is not accepted by Microsoft's key
- Although the UEFI spec. allows users to add/delete keys in PK/KEK/db, it is unrealistic to expect all users to have the skills
- So what the Linux distros do is to request Microsoft (its DevCenter program) to sign their boot loaders
  - Sigh…

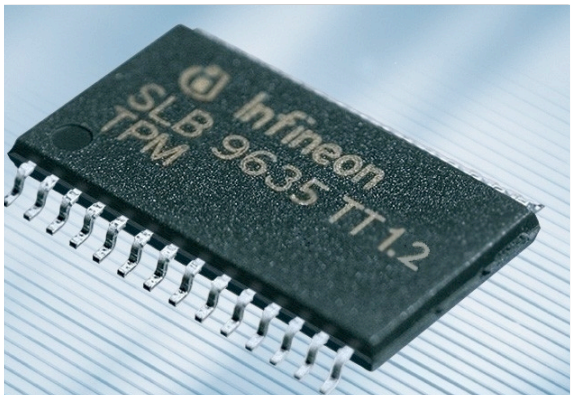


Outline

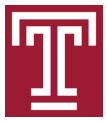
- UEFI Secure Boot
- Windows's Trusted Boot
- Intel's Trusted Boot



Recall TPM

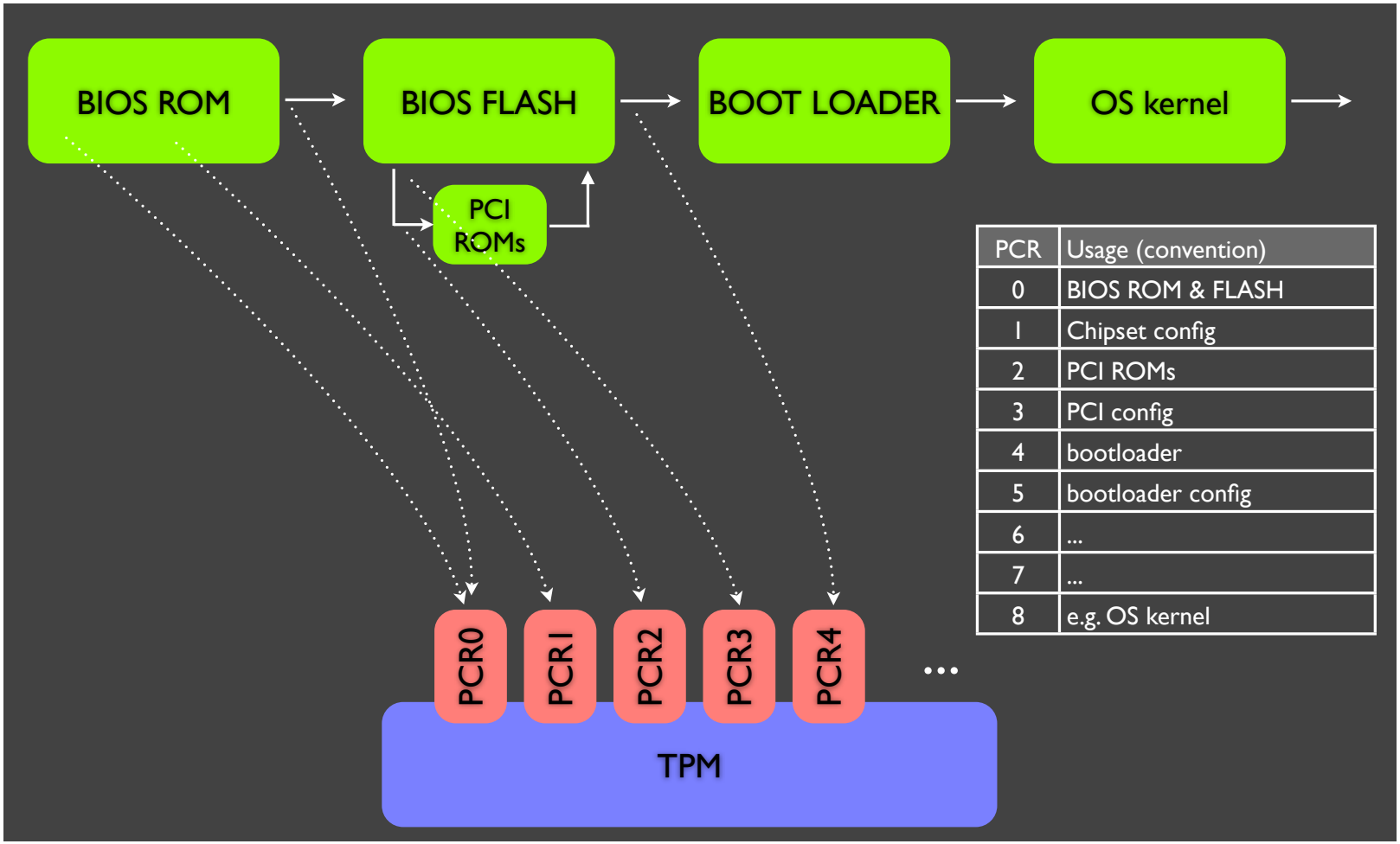


- Passive I/O device
- Special Registers: PCR
- Operations supported:
  - Seal/Unseal
  - Remote Attestation (Quote)
  - Primitive crypto services: RSA, HMAC, PRNG

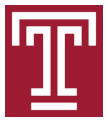


Static Root of Trust Measurement (SRTM)

- It refers to the measurement upon reboot

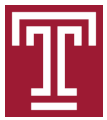


PCR	Usage (convention)
0	BIOS ROM & FLASH
1	Chipset config
2	PCI ROMs
3	PCI config
4	bootloader
5	bootloader config
6	...
7	...
8	e.g. OS kernel



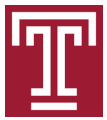
Applications and Limitations of SRTM

- Example Applications:
  - BitLocker
  - Remote Attestation
- Limitations of SRTM
  - It only measures until the kernel. It does not measure code that is running after the boot
  - It requires reboot
  - The measurement is not stable due to any change in the boot path



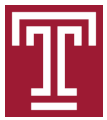
Dynamic Root of Trust Measurement (DRTM)

- It starts when a special security instruction is invoked
- It first resets PCRs 17-22 and then measures the code and configuration data according to the predefined policy
- It does not require reboot
- Designed to launch VMM with the confidence of the execution environment and the code integrity of VMM



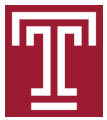
DRTM

- The dynamic PCRs contain measurement of:
  - 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

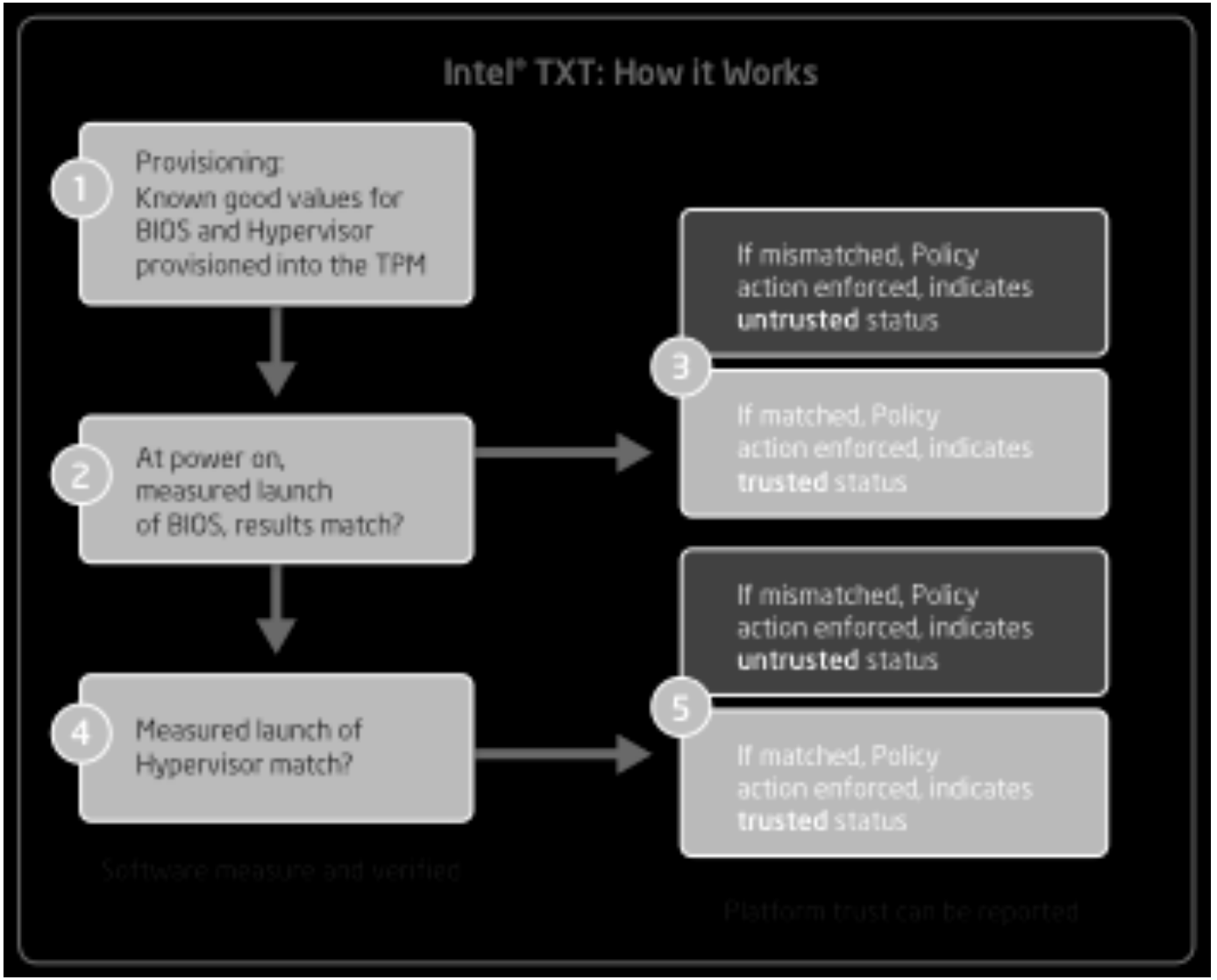


DRTM Implementations

- Intel's Trusted Execution Technology (TXT)
  - Formally known as LaGrande
  - A set of h/w extensions and primitives
  - Measurement upon SENTER instruction
  - Root of trust: the processor microcode first verifies the SINIT ACM (Authenticated Code Module)
- AMD's Secure Virtual Machine (SVM)
  - SKINIT (Secure Kernel Initialization) instruction
  - Chipset extensions and support

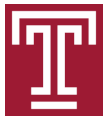


How TXT works

- 

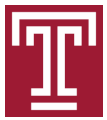
Trusted Boot (tboot)

- An open source project by Intel
- Trusted Boot (boot) is the name of an open source, pre-kernel/VMM module for Linux, that uses Intel TXT to perform a measured and verified launch of an OS kernel/VMM
- Intel TXT provides a hardware-based root of trust to ensure that a platform boots with a known good configuration of firmware, BIOS, virtual machine monitor, and operating system.



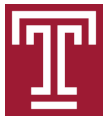
Summary

- UEFI Secure Boot: firmware and boot loader
- Windows's Trusted Boot: kernel and drivers
- Intel's Trusted Boot: measure the boot and enforce policies upon good/bad boot



References

- UEFI Secure Boot in Windows 8.1
  - https://answers.microsoft.com/en-us/windows/forum/windows8_1-security/uefi-secure-boot-in-windows-81
- Secure the Windows 8.1 boot process
  - https://technet.microsoft.com/en-us/windows/dn168167.aspx
- Advanced x86: BIOS and System Management Mode Internals. *UEFI SecureBoot*
- Attacking Intel TXT
  - http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20slides.pdf
- Intel TXT
  - https://en.wikipedia.org/wiki/Trusted_Execution_Technology
  - https://www.kernel.org/doc/Documentation/intel_txt.txt
  - https://www.slideshare.net/slkevin/txt-introduction
- Comparison between secure boot, trusted boot, and tboot
  - https://firmwaresecurity.com/tag/verified-boot/



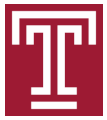
Backup

Is BitLocker + TPM vulnerable to Evil Maid Attack?

No! When disk is protected by BitLocker with TPM, the key is **sealed** by TPM, and TPM releases the key only when it finds the boot path (including the firmware and boot loader) is intact

But people have described attacks that exploit some implementation flaws: http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html

Another attack is the Fake Prompt attack described in next slide



How will you attack your roomate's PC whose disk is protected by BitLocker+TPM?

BitLocker + TPM means that if there is any change in the boot path, including the firmware and boot loader for BitLocker, TPM will not release the key sealed by TPM

But the trick is that you can install a modified BitLocker loader, which fools your roommate by showing a fake password input prompt

Later, when your roommate inputs the password the malware can log it. **Although your roommate will notice that her computer refuse to boot (since TPM refuse to release the key)**

So after logging the password, the malware should clear itself and reboot

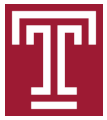
http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html

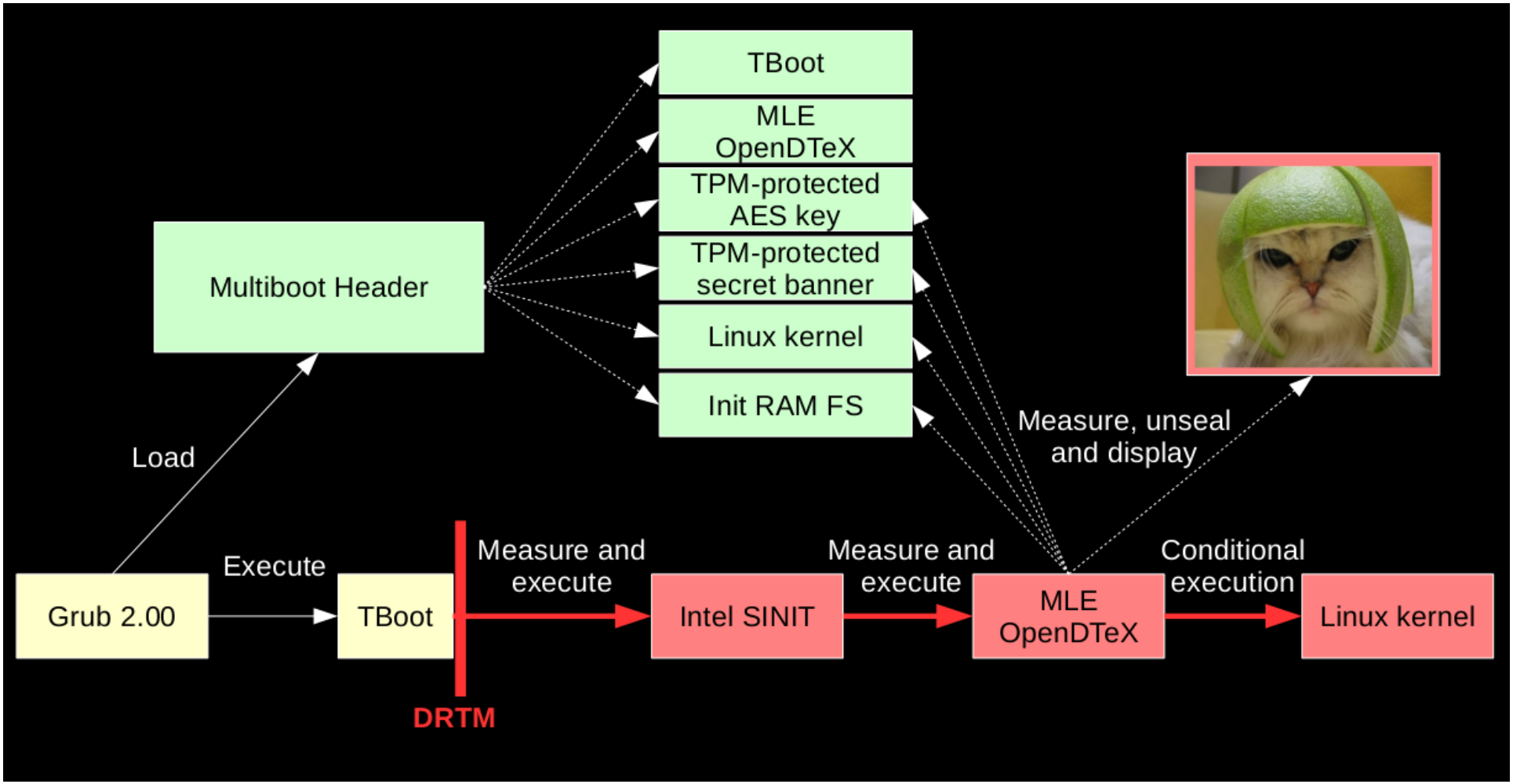
Backup

How about BitLocker + TPM + Secure Boot

The attacker has to disable Secure Boot first (e.g., by resetting the firmware password and then change the firmware setting).

Otherwise, I do not think the attacker would have any chance, as the system would refuse to execute malicious firmware/boot code




TBoot
MLE
OpenDTeX
TPM-protected
AES key
TPM-protected
secret banner
Linux kernel
Init RAM FS
Multiboot Header
Load
Measure, unseal
and display
Grub 2.00
Execute
TBoot
Measure and
execute
Intel SINIT
Measure and
execute
MLE
OpenDTeX
Conditional
execution
Linux kernel
DRTM