presented by

Strengthening the Chain of Trust

Kevin Lane – HP

Jeff Bobzin – Insyde Software

August 22, 2014

Agenda

- Quick Intro to UEFI
- UEFI Myths
- Using Linux + Secure Boot
- Continuing the Chain-of-Trust

UEFI Specification: What Is It?

- Defines a PC operating system and platform firmware interface
- Replaces Basic Input/Output System (BIOS) firmware interface
  - *Note: most UEFI images provide legacy support for BIOS services.*
- Consists of platform data tables, boot and runtime service calls
- Supports remote diagnostics and repair, even without an OS
- Standardizes booting of OSes and running pre-boot applications
- Evolved from Intel's EFI (Extensible Firmware Interface) specification
  - *Note: EFI was deprecated by UEFI in 2005.*
- Managed by the UEFI Forum

UEFI Specification: Milestones

- **Mid-90s:** First Intel–HP Itanium system development occurs
  - BIOS limitations (e.g., 16-bit processor mode, 1 MB addressable space and PC AT hardware) don't meet needs of larger server platforms
- **1998:** Intel creates EFI (originally "Intel Boot Initiative") to solve BIOS issue
- **2005:** Intel ceases EFI spec development at v1.10; contributes it to UEFI Forum
  - Original EFI specification remains owned and licensed by Intel
- **2007:** UEFI Forum releases UEFI specification v2.1
  - Adds cryptography, network authentication, user interface architecture (UEFI's Human Interface Infrastructure)
- **2008:** UEFI Forum releases UEFI specification v2.2
  - Adds Secure Boot
- **2013:** UEFI Forum releases current UEFI specification v2.4

Secure Boot: What Is It?

- Security protocol component of UEFI
- Secures the boot process by preventing loading of drivers or OSes that are not digitally signed in with acceptable markers
- Prohibits the ability of rootkits to install themselves into the computer system's boot loader
- Required WITH UEFI firmware by all manufactured PC's since advent of Windows 8 and 8.1
  - Secure Boot keys were developed by Microsoft and embedded in the UEFI firmware on the system.
- The UEFI *feature* causing the most Linux boot issues

Myth #1

I can't install Linux on a UEFI Computer!

- UEFI will not prevent you from installing Linux or any other (supported) operating system.
- Secure Boot may make it difficult to install operating systems other than Windows 8.
- Some PC manufacturers also allow a "legacy mode" to be selected that will emulate a PC BIOS.

Myth #2

Linux will not work with Secure Boot!

Secure Boot can always be disabled, thus allowing you to install Linux or any other supported operating system.

Most newer versions of the popular Linux distributions now ship with support for Secure Boot, allowing you to not only install with Secure Boot enabled in UEFI, but also helping to secure the chain of trust.

Myth #3

I need TPM to get Secure Boot to work!

TPM (Trusted Platform Module)…

- Is just another layer that you can use to store your credentials.
- Was also implemented prior to UEFI on PC platforms (non-enterprise).
- Is already implemented by the major PC vendors, and offers an additional layer of security, but is not a prerequisite for Secure Boot.
- Is a hardware-based (chip) with an API layer that encompasses middleware and applications.

What Can I Do With UEFI?

- The UEFI Shell
  - A shell/terminal for the firmware
  - Allows launching of UEFI applications (e.g., UEFI bootloaders)
  - Can be used to obtain other system or firmware information (e.g., memory map (memmap), modifying boot manager variables (bcfg), running partitioning programs (diskpart), loading UEFI drivers, editing text files (edit), hexedit, etc.)
- uefitoolkit
  - Provides examples of what you can do with EFI
- Driver development – Intel
- Programmable boot options
  - Implementation depends on Hardware vendor

What Can I Do With UEFI?

rEFInd – Boot Manager for EFI/UEFI

- NOT a boot loader
- All EFI-capable Operating Systems include boot loaders.
- Most EFI implementations should provide a boot manager.
- Sadly, most are very poor or useless.

How Do I Get Linux To Work With Secure Boot?

Why not just disable Secure Boot?

This is BAD Mojo!

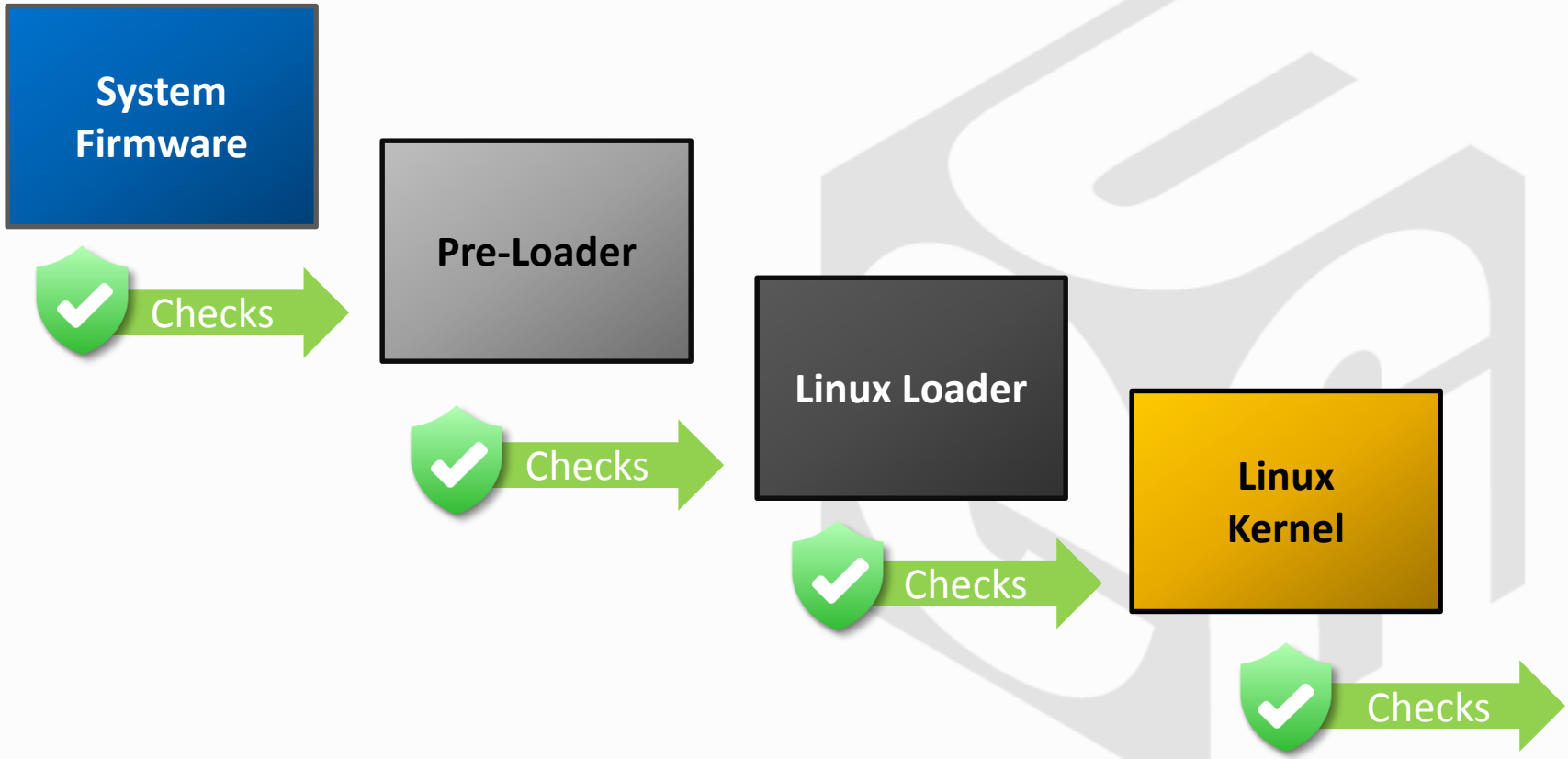
The main reason for Secure Boot is to keep the boot process and boot code secure.

- Disabling Secure Boot leaves you susceptible to rootkits that hide in the boot code and go undetected.
- Most major Linux distributions support Secure Boot!
- Add a signing key to the UEFI Firmware!

Chain-of-Trust:

Use Linux Distributions or Create Your Own

Implementing A Trust Chain



Who Do We Trust?

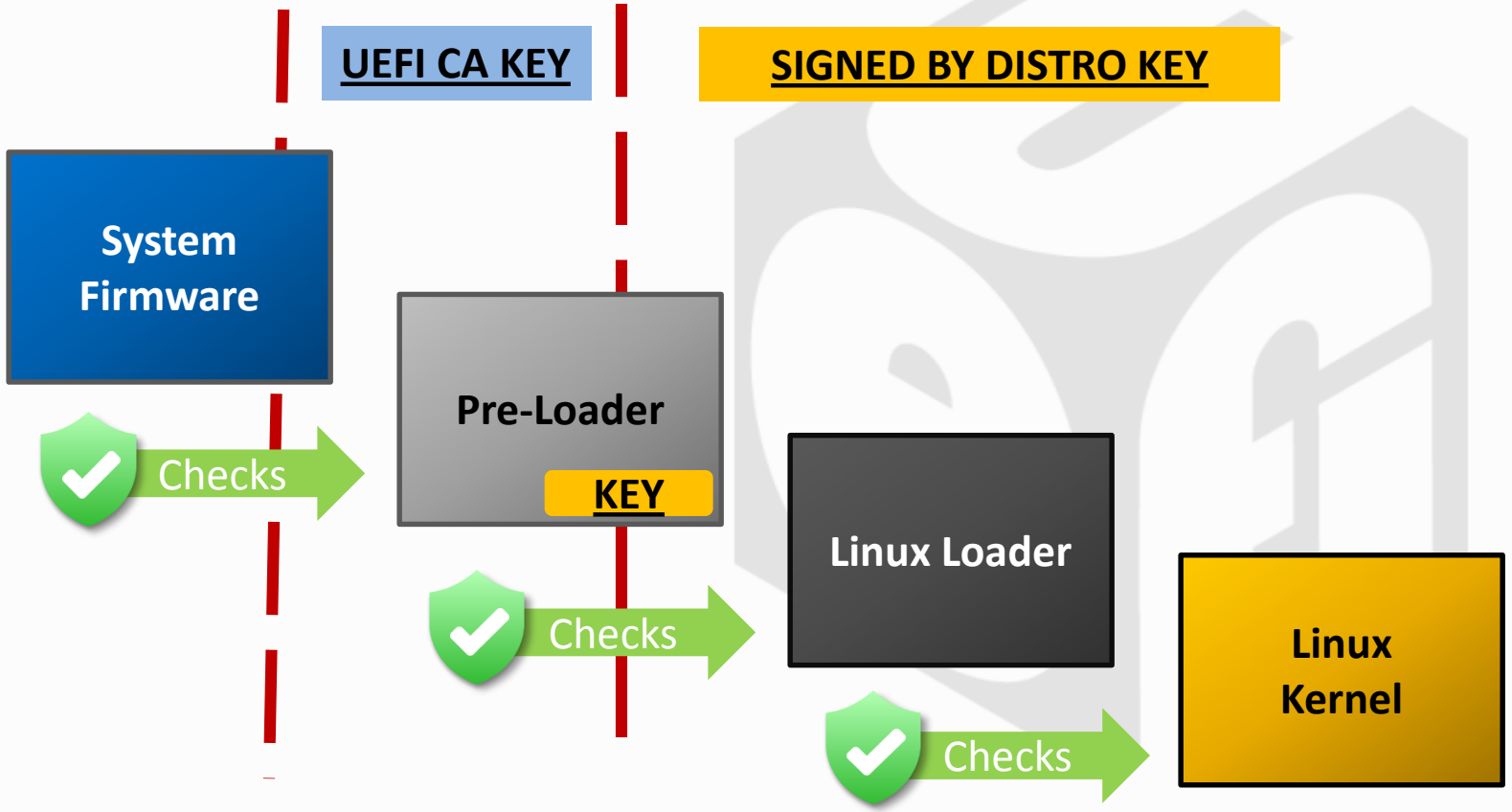
- **Firmware as the “Root of Trust” verifies First Stage Bootloader**
- shim.efi binary is signed by either:
  1) Microsoft CA – installed into system at factory
  2) OEM or Owner Created Key – Certificate must be provisioned into system (db)

Booting Continues

- **shim.efi in turn verifies Grub2, and Grub2 uses shim service to verify Kernel**
- grubx64.efi and kernel are signed by one of these:
  1) Distro-created key inserted into Distro-created build of SHIM.EFI, (which must then be submitted to MS for signing)
  2) Signed by Key enrolled by User during Linux Install (the MOK)

Using The 'Distro' Signing

Transitions from UEFI to Distro Signing

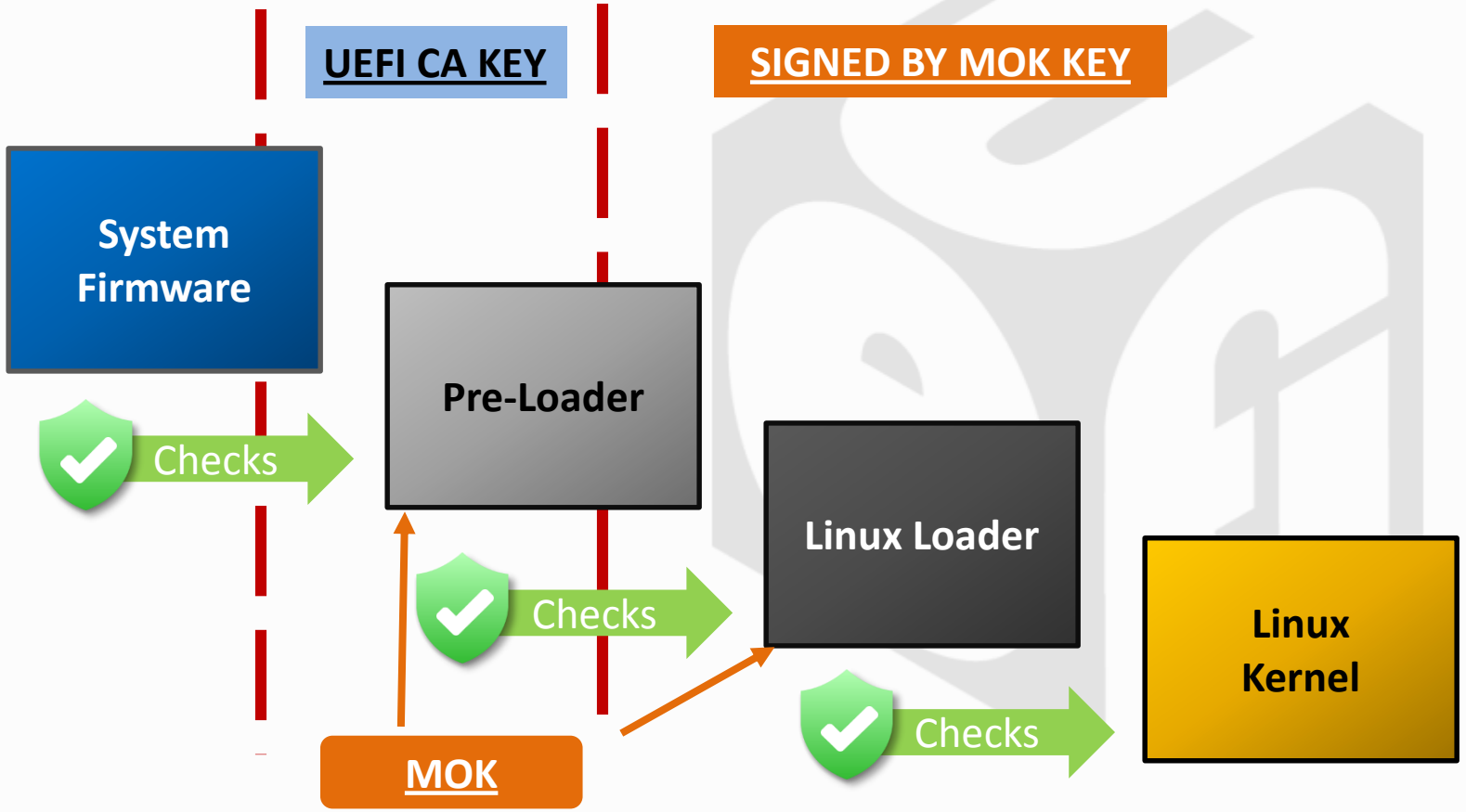


Using The 'Disto' Signed Boot

- Advantages:
  1) Simple – no per-machine configuration
  2) Do not need to keep local-generated private keys secure
- Disadvantages:
  1) Cannot use local-built grub, Linux, or drivers

Machine Owner Key (MOK)

Switch from UEFI to Local Security Space



Steps For Setting Up MOK

- Set up corporate infrastructure for creating public/private key pairs and keeping secure
- Use mokutil to request install
- On reboot, shim.efi loads MokManager.efi
- Load the MOK public key (kept safe by f/w)
- Sign all rebuilt components before install

Note: requires hands-on config for every system

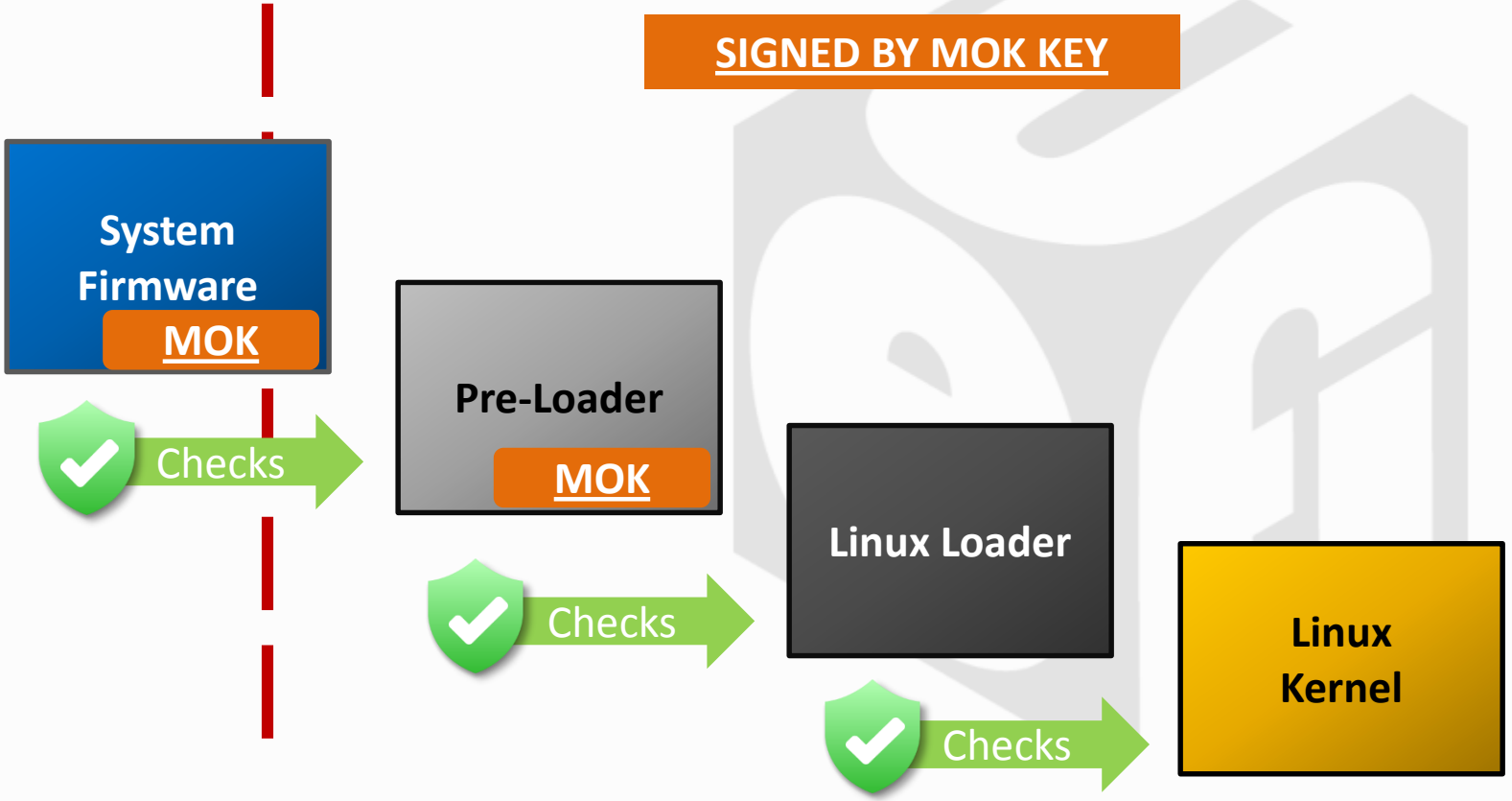
Possible Site Policy

	UEFI CA	DISTRO	Machine Owner
shim.efi			
grubx64.efi			
Linux Kernel			
Driver #1,3,4…			
Driver #2			

Note: Approved Key Accumulate as Boot Progresses

Site With Total Local Control

Complete Local Security Space



Replacing The Factory UEFI DB

1) Use “Efitools” at http://git.kernel.org/cgit/linux/kernel/git/jejb/efitools.git
2) Generate custom database in form of lockdown.efi
3) Clear firmware db using BIOS menus
4) Use BIOS menus to run lockdown.efi

High-touch and For Experts Only!

Help From Redhat

Chapter 24.7 in *Red Hat Enterprise Linux 7 System Administrators Guide*

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/

Fedora UEFI Secure Boot Guide

http://docs.fedoraproject.org/en-US/Fedora/18/html/UEFI_Secure_Boot_Guide/chap-UEFI_Secure_Boot_Guide-What_is_Secure_Boot.html

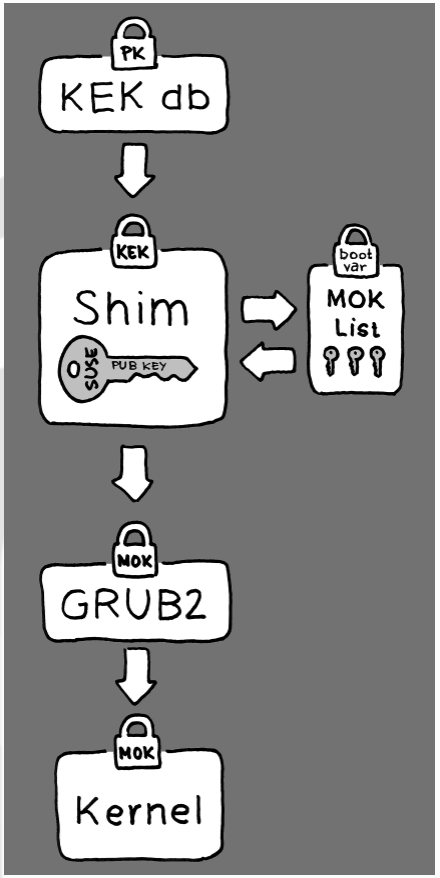
Help From SUSE/ openSUSE

Suse and Secure Boot: The Details

https://www.suse.com/communities/conversations/uefi-secure-boot-details/

http://en.opensuse.org/openSUSE:UEFI#Booting_a_custom_kernel

Chapter 11 in *SUSE Linux Enterprise Server – Administrators Guide*
https://www.suse.com/documentation/sles11/book_sle_admin/data/book_sle_admin.html



Helpful Blogs And Web Books

- http://www.rodsbooks.com/efi-bootloaders/secureboot.html
- http://blog.hansenpartnership.com/uefi-secure-boot/
- http://mjg59.dreamwidth.org/
- http://uefidk.intel.com/blog

For more information on the Unified EFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by