UEFI, Secure Boot, Hilfe!

Christian Seiler

2026-07-04

Hinweise

Umfang des Vortrags

- UEFI und Secure Boot auf x86 Systemen

Umfang des Vortrags

- UEFI und Secure Boot auf x86 Systemen
  - Manche ARM-Systeme (Windows-Laptops/Tablets, div. Server)
  - Manche RISC-V-Systeme
  - Weitere inoffizielle Varianten

Umfang des Vortrags

- UEFI und Secure Boot auf x86 Systemen
  - Manche ARM-Systeme (Windows-Laptops/Tablets, div. Server)
  - Manche RISC-V-Systeme
  - Weitere inoffizielle Varianten
- **Keine** Embedded-Systeme
- **Kein** TPM

Umfang des Vortrags

- UEFI und Secure Boot auf x86 Systemen
  - Manche ARM-Systeme (Windows-Laptops/Tablets, div. Server)
  - Manche RISC-V-Systeme
  - Weitere inoffizielle Varianten
- **Keine** Embedded-Systeme
- **Kein** TPM
- Alle Aussagen haben Ausnahmen
- Maßstäbe in Diagrammen sind nicht korrekt

Umfang des Vortrags

- UEFI und Secure Boot auf x86 Systemen
  - Manche ARM-Systeme (Windows-Laptops/Tablets, div. Server)
  - Manche RISC-V-Systeme
  - Weitere inoffizielle Varianten
- **Keine** Embedded-Systeme
- **Kein** TPM
- Alle Aussagen haben Ausnahmen
- Maßstäbe in Diagrammen sind nicht korrekt
- Bugs, Bugs, und noch mehr Bugs!

Rückblick: BIOS

Es war einmal, im Jahr 1975...

- Ende vom Vietnam-Krieg

Es war einmal, im Jahr 1975...

- Ende vom Vietnam-Krieg
- Ende der Franco-Diktatur in Spanien

Es war einmal, im Jahr 1975...

- Ende vom Vietnam-Krieg
- Ende der Franco-Diktatur in Spanien
- Das Apollo-Sojus-Test-Projekt

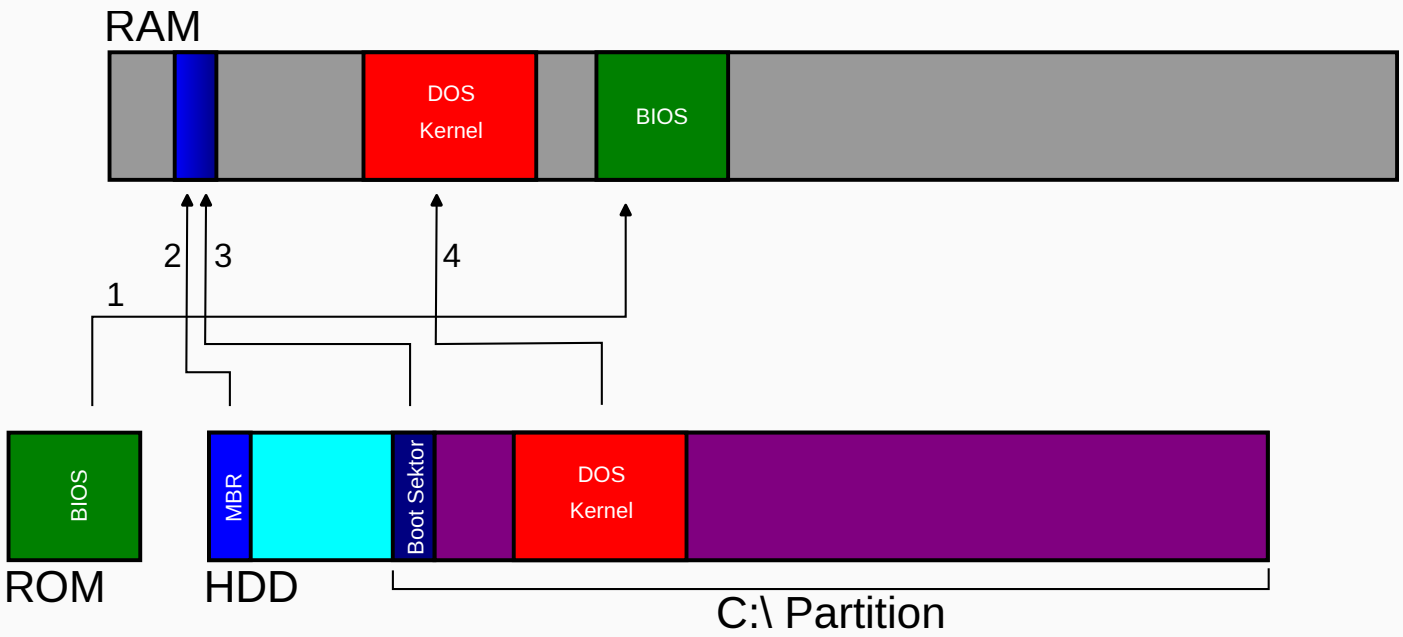
Es war einmal, im Jahr 1975...

- Ende vom Vietnam-Krieg
- Ende der Franco-Diktatur in Spanien
- Das Apollo-Sojus-Test-Projekt
- Der Film “Der weiße Hai” (“Jaws”)

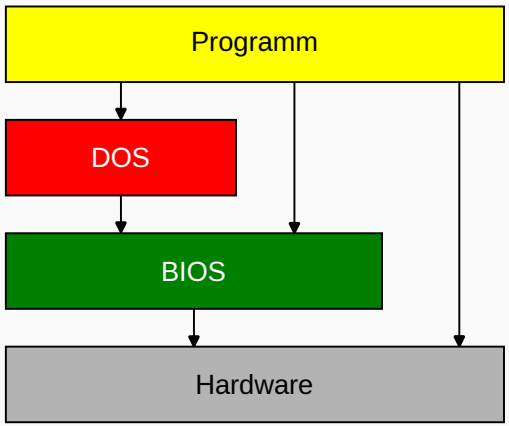
Es war einmal, im Jahr 1975...

- Ende vom Vietnam-Krieg
- Ende der Franco-Diktatur in Spanien
- Das Apollo-Sojus-Test-Projekt
- Der Film “Der weiße Hai” (“Jaws”)
- IBM führt das BIOS für den PC ein (CP/M als Betriebsystem)

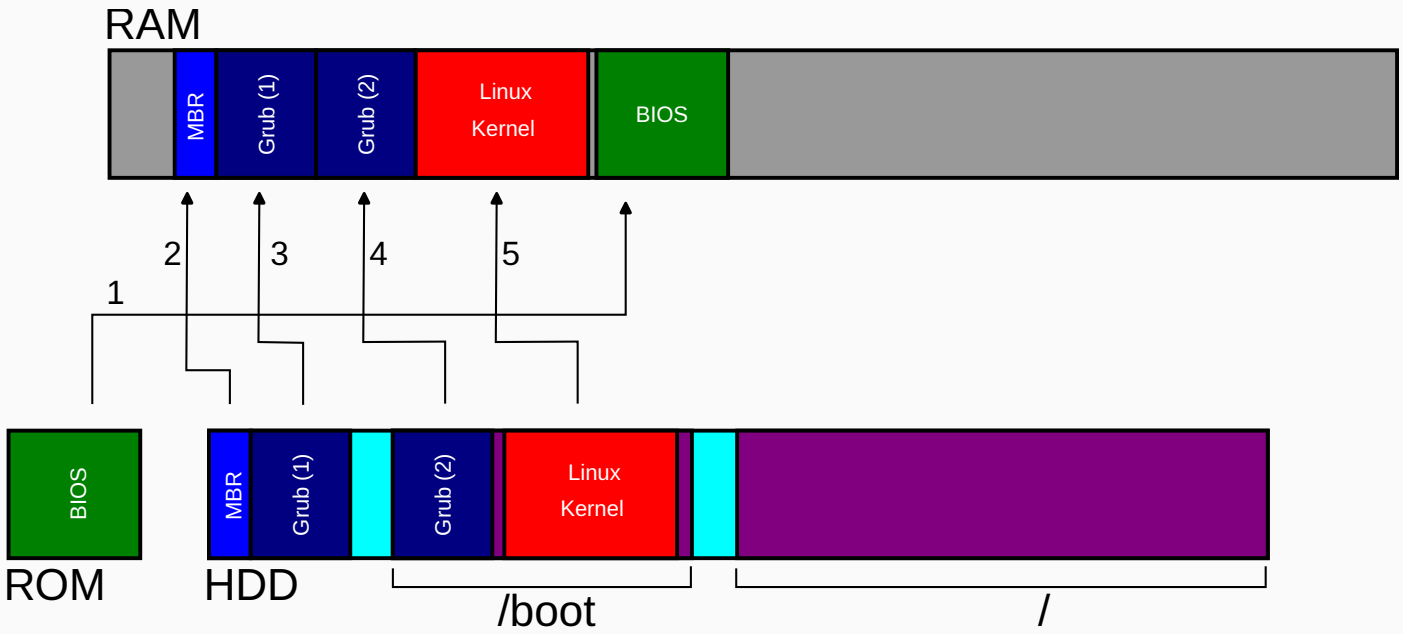
Bootvorgang, ca. 1990



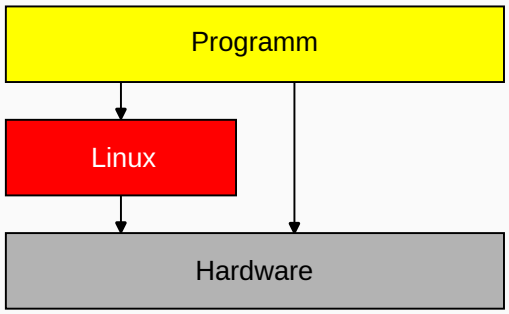
- BIOS laden, HW init
- MBR, Boot-Sektor laden
- Kernel laden
- BIOS läuft weiter



Bootvorgang, Linux, 1990er



- BIOS laden, HW init
- MBR, GRUB laden
- Kernel laden
- Linux übernimmt Kontrolle



UEFI - Unified Extensible Firmware Interface

Was ist UEFI?

- Ersatz für BIOS
- Ursprünglich von Intel, (“EFI”, 2005)
- UEFI Forum (Spezifikation)
- Referenzimplementierung: TianoCore EDKII
  (“EFI Development Kit 2”)
- Produkte: AMI Aptio, Phoenix SecureCore, InsydeH2O, . . .

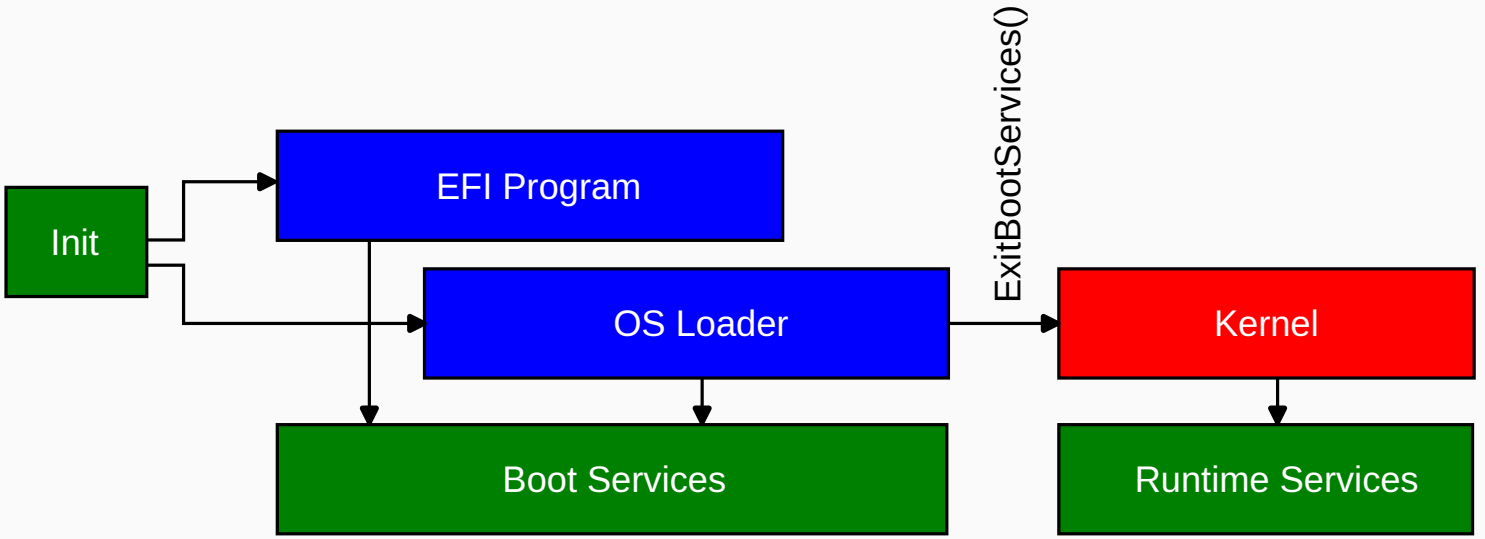
Unterschiede UEFI und BIOS

- Größer (mehrere MiB!)
- Kann Partitionstabellen lesen
- Kann FAT-Dateisysteme direkt lesen
- Kann HDDs/SSDs mit mehr als 2 TiB korrekt behandlen
- Definierte Schnittstelle für Boot-Programme **und** Betriebsystem

Compatibility Support Module (CSM)

- Emuliert ein BIOS auf Systemen mit UEFI
- Kann Betriebsysteme im BIOS-Modus starten
- In den “BIOS-Einstellungen” einstellbar, ob aktiviert
- In neueren Mainboards oft nicht mehr verfügbar
- In einigen nachinstallierbar (CSMWrap)

Bootvorgang mit UEFI



Betriebsystemstart:

- **OS-Loader**: spezielles EFI-Programm
- Betriebsystem (Kernel) hat volle Kontrolle über das System

EFI Programm:

- "Portable Executable" Format (DOS/Windows)
- Dateiendung: `.efi`

EFI-Variablen

- Zwei wichtigsten EFI-Services:
  `GetVariable()`, `SetVariable()`
- Gepseichert in nicht-volatilem RAM (NVRAM)
- Können (je nach Berechtigung) vom Betriebsystem geändert werden
- Manipulation unter Linux:
  - `efivarfs`-Dateisystem, `/sys/firmware/efi/efivars` (**Vorsicht!**)
  - `efi-readvar`, `efi-updatevar` (Paket: `efitools`)
  - `mokutil` (siehe später, Paket: `mokutil`)
  - `efibootmgr` (siehe später, Paket: `efibootmgr`)
  - Boot-Loader Installationsroutinen

EFI-Variablen vs. Boot-Vorgang

- `Boot0000`, `Boot0001`, `Boot0002`, ...
  - Im UEFI registrierte Programme (`*.efi`), die vom UEFI gestartet werden können
  - Nur die registrierten werden vom UEFI gebootet*
  - **Nachteil:** Festplatte kann nicht so einfach in neues System eingebaut werden
- `BootOrder`
  - Reihenfolge, in denen EFI-Programme ausprobiert werden vom UEFI
  - **Fallstrick:** in machen UEFI gibt es eine Einstellung `Boot Order Lock` - diese erlaubt keine Änderungen von `Boot*`-Variablen!
    - Vor installation von Linux deaktivieren!

EFI-Boot-Variablen

Beispiel (Festplatte, 1)

```
HD(1,GPT,e2dc8d76-8321-4264-b667-fe3fed6134f8,
0x800,0xee000)/File(\EFI\DEBIAN\SHIMX64.EFI)
```

Beispiel (Festplatte, 2)

```
HD(1,GPT,e2dc8d76-8321-4264-b667-fe3fed6134f8)
/File(\EFI\DEBIAN\SHIMX64.EFI)
```

- In der Variable selbst in Binärform abgelegt
- Auch komplexere Konfigurationen möglich (iSCSI über TCP über IPv6 via DHCPv6)
- GUID bei `HD` ist die `PARTUUID` der GPT-Partition
- Optionalen Hex-Zahlen in `HD`: Start und Größe der Partition
- Automatische Erkennung der Festplatte (Alternativ: PCIe-Pfad angeben vor `HD`)

EFI System Partition (ESP)

- Partitionstabelle: GPT (*theoretisch* auch MBR)
- Typ: EFI System Partition (`C12A7328-F81F-11D2-BA4B-00A0C93EC93B`)
- Dateisystem: FAT32 (*theoretisch* auch FAT12 oder FAT16)
  kein ExFAT!
- Sollte den Bootloader als `*.efi`-Datei enthalten
- Pfad beliebig, wenn explizit eingetragen in der Firmware
- Removable Media Path: `\EFI\BOOT\BOOTX64.EFI`
  - ARM64: `\EFI\BOOT\BOOTAA64.EFI`
  - Liste: `https://wiki.debian.org/UEFI#Booting_from_removable_media`
- Windows Bootloader: `\EFI\Microsoft\Boot\bootmgfw.efi`
- Mountpoint: `/boot/efi`

Exkurs: Verwirrung unter Windows

- Windows-Datenbank: “Boot Configuration Data” (BCD)
- Datei: `\EFI\Microsoft\Boot\BCD` auf der ESP
- Konfiguriert den Windows Boot Manager (`bootmgfw.efi`)
- `BCDEdit.exe` editiert diese Datei
- `BCDEdit.exe` editiert aber **auch** EFI-Variablen
- Viel Spaß, das zu verstehen. . .

EFI-Booteinträge bearbeiten (1)

- **Einträge anzeigen**:

```
efibootmgr -v
BootCurrent: 0000
Timeout: 1 seconds
BootOrder: 0000,0001
Boot0000* debian
  HD(1,GPT,...)/File(\EFI\DEBIAN\SHIMX64.EFI)
Boot0001* debian
  HD(1,GPT,...)/File(\EFI\DEBIAN\GRUBX64.EFI)..BO
```

EFI-Booteinträge bearbeiten (2)

- **Reihenfolge ändern**:
  `efibootmgr -o 1,0`
- **Eintrag löschen**:
  `efibootmgr -b 0 -B`
  Achtung: die Numerieung kann sich danach ändern, muss aber nicht
- **Eintrag anlegen**:
  `efibootmgr -c -L "Anzeigename" -l '\efi\something\myloader.efi'`
  Nimmt an, dass die Datei in der aktuellen ESP ist (`/boot/efi`)

EFI-Booteinträge bearbeiten (3)

- **GRUB neu installieren**:
  `grub-install --target=x86_64-efi`
  Optional: `--force-extra-removable`
- **systemd-boot neu installieren**:
  `bootctl install`

Hinweis zum Reparieren via Live-Stick:

Immer in einem chroot mit (bind-)gemountetem `/boot` (falls relevant), `/boot/efi`, `/proc`, `/sys`, `/dev`, `/dev/pts`, und `/sys/firmware/efi/efivars` ausführen.

Wer braucht schon einen Bootloader?

Unified Kernel Images

- EFI-Stub + Kernel + Initramfs
- Abgelegt in EFI System Partition
- Kann direkt von UEFI oder von einem generischen EFI-Bootloader geladen werden
- `https://uapi-group.org/specifications/specs/unified_kernel_image/`

Secure Boot

Secure Boot - Zielsetzung

Durch Kryptographie sicherstellen, dass auf dem System nur Code gestartet wird, der explizit bewilligt wurde

Gretchenfrage: Von **wem**?

Secure Boot - Threat Model

Was bringt Secure Boot?

- Malware ab Boot ("Bootkits") / "Boot-Sektor-Viren"
- Automatische Festplatten-Entschlüsselung
- Für Experten: explizite Kontrolle über gestartete Software

Secure Boot - Threat Model

Was bringt Secure Boot?

- Malware ab Boot ("Bootkits") / "Boot-Sektor-Viren"
- Automatische Festplatten-Entschlüsselung
- Für Experten: explizite Kontrolle über gestartete Software

Damokles-Schwert

- Hardware-Hersteller (und Microsoft) entscheiden, was ausgeführt wird und was nicht
- Vergleich: Mobiltelefone

Secure Boot - Sicherheitsstufen von EFI-Variablen

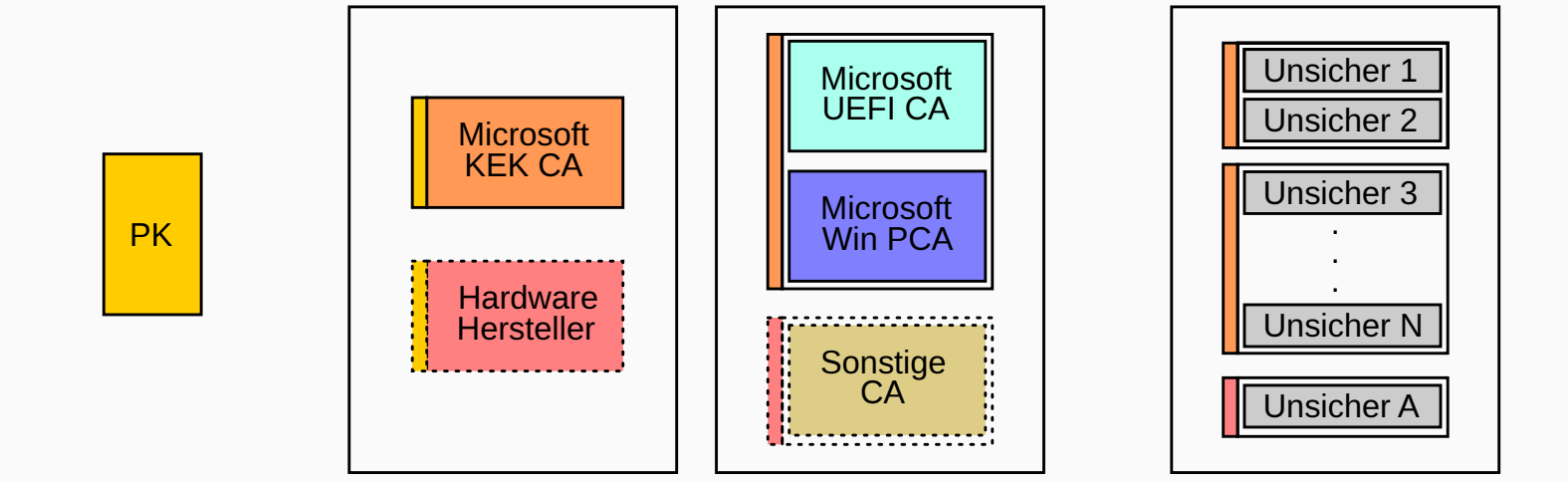
Sicherheitsstufen:

- Nur während UEFI-Initialisierung ("BIOS-Setup") änderbar
- Nur während Boot Services laufen änderbar
- Immer änderbar
- Speziell: nur anhängbar (Append Write)

Implementierung:

- Sicherheit durch Hardware

Secure Boot - Konzepte



Platform Key

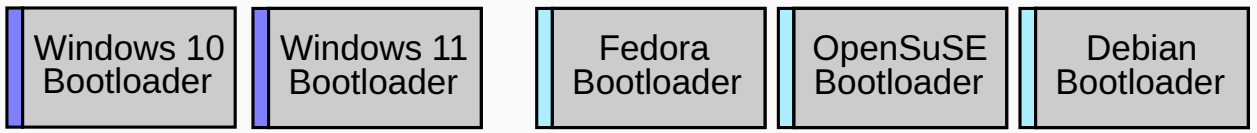
Hardware Hersteller

Nur 1x pro System

Key Exchange Key (KEK)

Database (DB)

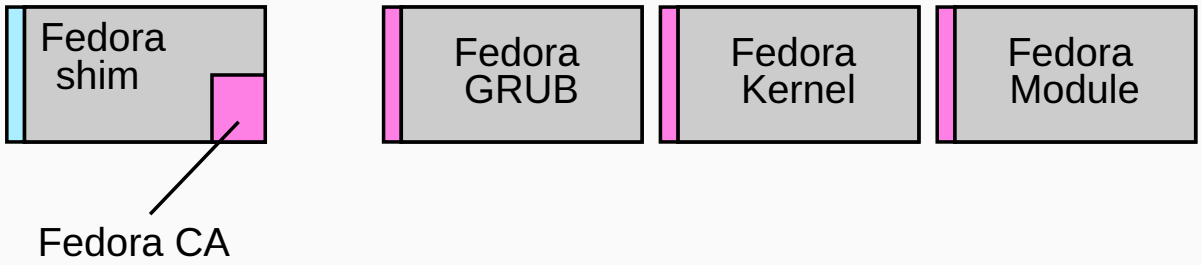
Forbidden Signature DB (DBX)



Secure Boot - Die Rolle von Microsoft

- *Theoretisch*: Secure Boot unabhängig von Microsoft
- *In der Praxis*: Geräte supporten Windows; Microsoft KEK und Microsoft CAs in der DB
- In der Praxis: Microsoft zentrale Anlaufstelle
  - Bestimmte Anforderungen an die Funktionalität
  - Bisher fair
  - Aber: langwieriger Prozess; nicht praktikabel, jeden Bootloader und Kernel durch Microsoft zu signieren

Secure Boot - shim



- Minimales EFI-Programm, startet nur den nächsten EFI-Bootloader
- Hat zusätzlich ein oder mehrere CAs der Distro integriert
- Verifiziert gegen DB + integrierte CA (+ MOK) und DBX
- Muss sich nicht häufig ändern
- Von Microsoft signiert
- Aber: eigentliche Bootloader, Kernel, etc. von Distro selbst signiert

Secure Boot - Die DBX-Katastrophe

- GRUB ursprünglich nicht für Secure Boot designed
- Threat-Modell für GRUB war ursprünglich ein anderes
- 30 Prozent DBX: unsichere GRUB-Versionen diverser Distros
- Nicht nachhaltig

Secure Boot - SBAT

- Secure Boot Advanced Targeting
- CSV Teil des EFI-Bootloaders:

```
sbat,1,SBAT Version,sbat,1,https://github.com/rhboot/shim/...
grub,5,Free Software Foundation,grub,2.06,https://www.gnu.org/...
grub.debian,5,Debian,grub2,2.06-13+deb12u2,https://...

grub.debian12,1,Debian,grub2,2.06-13+deb12u2,https://...
```

- Generations-Zahlen
- DBX kann Einträge enthalten: `grub,4`
- Erschlägt alle Distros mit der gleichen GRUB-Version

Secure Boot - SBAT-Einführung in 2024

- August 2024: Microsoft rollt per Windows Update aus, was alle `shim` per DBX sperrt, die kein SBAT können
- Allerdings: noch nicht alle Linux-Distros haben geupdatet
- **Versuch** von Microsoft, eine Dual-Boot-Erkennung einzubauen, und das Update zu unterdrücken
- Außerdem: Update nicht korrekt implementiert
- Fix für die Probleme: Mai 2025
- Inzwischen auch (fast) alle Linux-Distros mit SBAT

Secure Boot - Konsequenzen im Kernel

Threat-Modell von Secure Boot: es soll nichts im Kernel-Modus laufen, was nicht verifiziert wurde (`root` ist nicht mehr Gott)

- Nachladbare Module
- `/dev/mem`
- `kexec`
- `...`

Secure Boot - Konsequenzen im Kernel

Threat-Modell von Secure Boot: es soll nichts im Kernel-Modus laufen, was nicht verifiziert wurde (`root` ist nicht mehr Gott)

- Nachladbare Module
- `/dev/mem`
- `kexec`
- ...

Lösung (u.a. "Lockdown LSM"):

- Kernel prüft Signaturen von Modulen und von `kexec`-Zielen
- `/dev/mem` geht nicht mehr
- ...

Secure Boot - MOK

- Machine Owner Key
  - Eigener Schlüssel, mit dem der User auf einzelnen PCs selbst signieren kann

Secure Boot - MOK

- Machine Owner Key
  - Eigener Schlüssel, mit dem der User auf einzelnen PCs selbst signieren kann
- Konzept in `shim` - **kein** offizieller Teil von Secure Boot

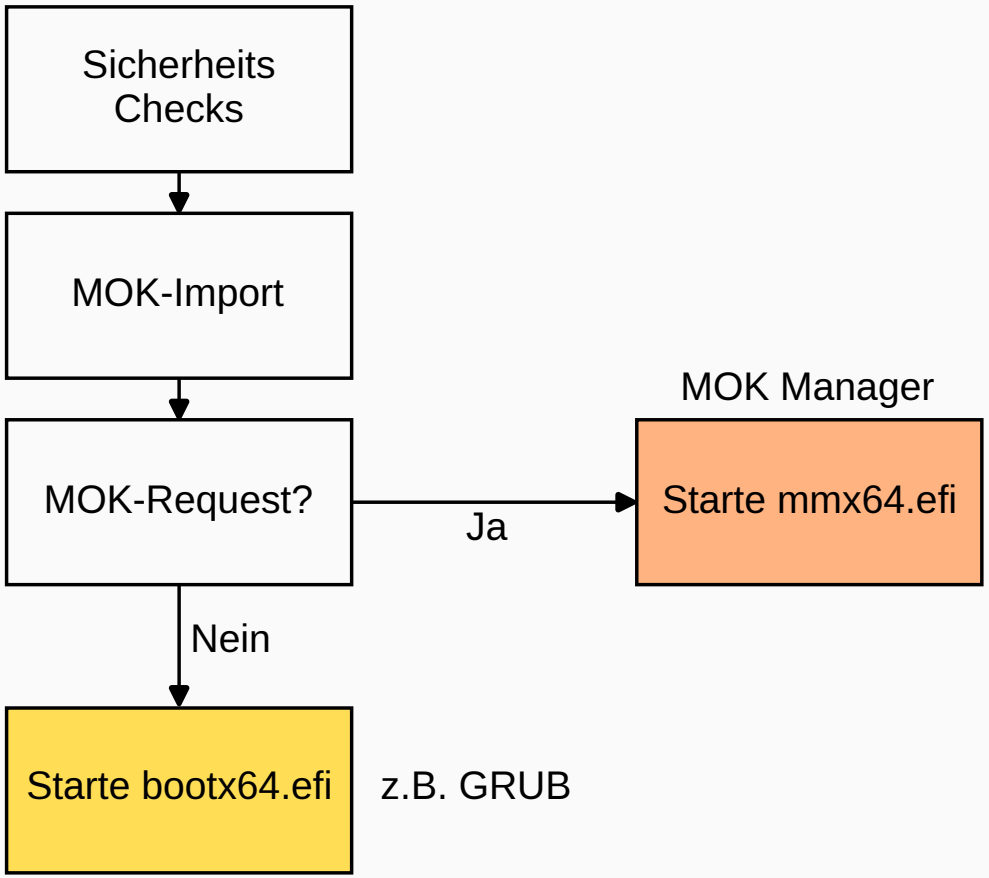
Secure Boot - MOK

- Machine Owner Key
  - Eigener Schlüssel, mit dem der User auf einzelnen PCs selbst signieren kann
- Konzept in `shim` - **kein** offizieller Teil von Secure Boot
- Macht damit `root` wieder zum Gott

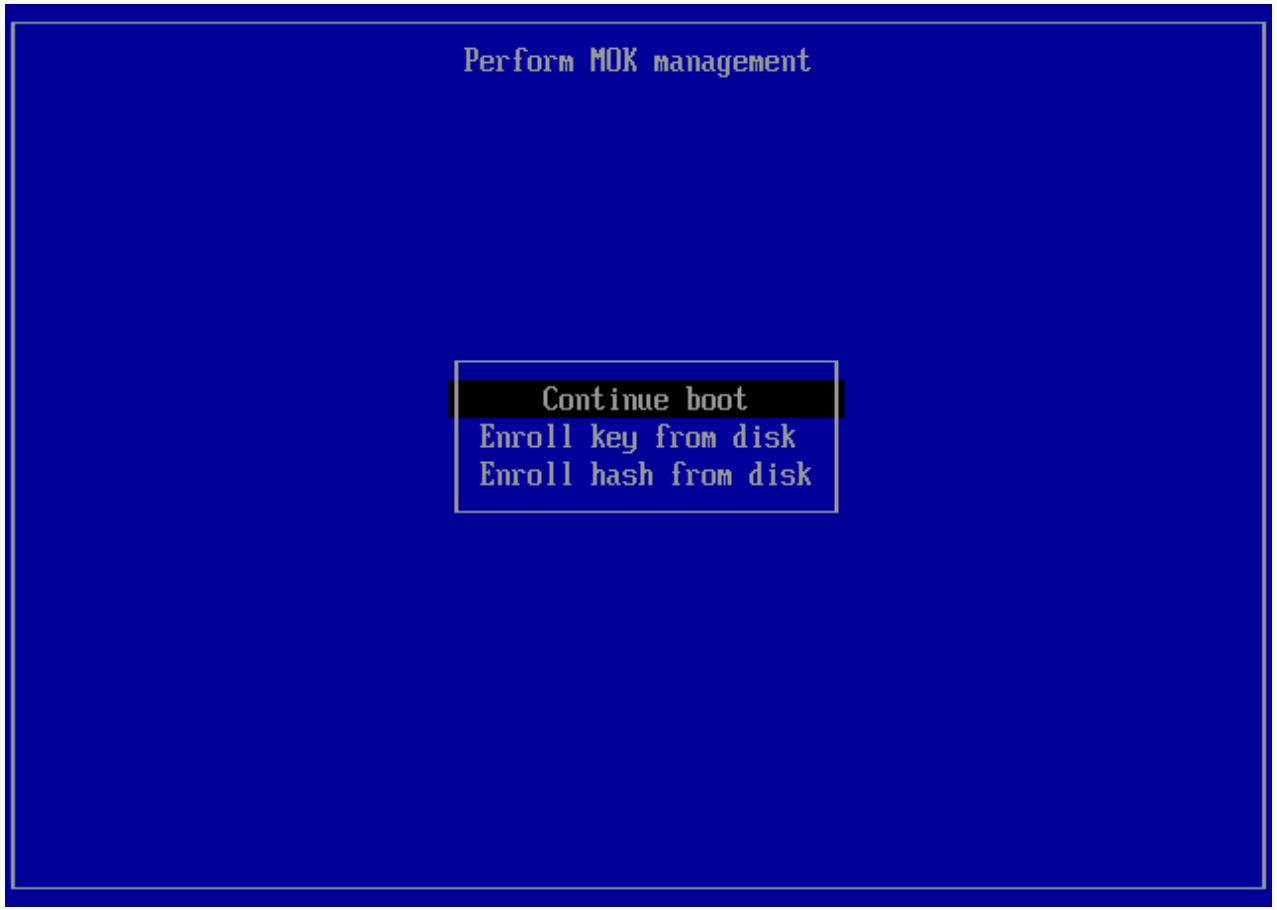
Secure Boot - MOK

- Machine Owner Key
  - Eigener Schlüssel, mit dem der User auf einzelnen PCs selbst signieren kann
- Konzept in `shim` - **kein** offizieller Teil von Secure Boot
- Macht damit `root` wieder zum Gott
- Status-Variablen (Boot Services Only): `MokList`, `MokListX`, ...
- Request-Variablen (Runtime Services): `MokNew`, `MokSB`, `MokPW`, `MokAuth`, `MokDel`, `MokDB`, `MokXNew`, `MokXDel`, `MokXAuth`, `MokListTrustedNew`

Secure Boot - MOK - Shim-Bootvorgang



Secure Boot - MOK Manager



Secure Boot - `mokutil`

- `mokutil --sb-state`
- `mokutil --password "ABC123" --import cert.der`
- `mokutil --list-new`
- `mokutil --list-enrolled`
- Anekdote: Mint-Installer vs. MOKManager

Secure Boot - Roll Your Own

- In vielen UEFI kann man PK, KEK, DB und DBX komplett selbst tauschen
- Volle Kontrolle über das, was im System starten soll
- Aufwand: Signaturen, Revocations, etc. alles selbst verwalten
- Vorteil für größere Organisationen
- Oder für paranoide User

Secure Boot - Microsoft Key Rollover

- Ursprüngliche Microsoft UEFI CA und Windows PCA von 2011 abgelaufen (Ende Juni 2026)
- Microsoft KEK CA wird Oktober 2026 auslaufen
- Die meisten Linux-Distros: aktuelle `shim` sind sowohl mit den alten und den neuen Microsoft CAs signiert
- Microsoft kontrolliert die jeweiligen PK nicht – KEK-Update im Herbst wird lustig
- Interessanter Datenpunkt: Viele UEFI-Firmwares prüfen das Signaturdatum nicht
- Dual-Boot: Windows Update aktualisiert zumindest DB und DBX
- Kein Dual-Boot: `fwupd` laufen lassen

Secure Boot - Ausblick: Post-Quantum

- Aktuelles Secure-Boot wäre nicht sicher gegen Quanten-Computer
- Bisher noch kein Teil der Spezifikation, der PQC-Algorithmen vorschreibt
- Es gibt Forschungsarbeiten dazu

Secure Boot - Ausblick: TPM

- Hauptanwendungszweck TPM (aktuell): automatische Festplatten-Entschlüsselung
  - Windows: Bitlocker
  - Linux: LUKS + `systemd-cryptenroll`
  - Nur in Verbindung mit Secure Boot sicher
- TPM selbst: orthogonal zu Secure Boot
- Viel Fehlinformationen zum TPM, leider auch von der FSF
- Vielleicht in einem Vortrag im nächsten Jahr, wenn Interesse besteht ;-)

Tools

- `efibootmgr`
- `efitools` (`efi-readvar`, `efi-updatevar`)
- `mokutil`
- `sbsigntool`

Lesestoff

- Generisches:
  - https://uefi.org/specs/UEFI/2.10/
  - https://arxiv.org/html/2311.03809v2
  - https://igor-blue.github.io/2021/02/04/secure-boot.html
  - https://github.com/rhboot/shim/blob/main/SBAT.md
- SBAT-Debakel im Detail:
  - https://windowsforum.com/threads/366278/
- Ablauf der UEFI-Keys:
  - https://lwn.net/Articles/1079808/
  - https://mjg59.dreamwidth.org/72892.html
  - https://blog.einval.com/2026/05/22#secure_boot_ca_rollover
  - https://techcommunity.microsoft.com/blog/linuxandopensourceblog/what-.../4530725

Ende

Vielen Dank für die Aufmerksamkeit!



Lizenz: CC-BY SA 4.0 (International)

Bilderquellen

- Diagramme: selbstgezeichnet (via Inkscape)
- MOKManager Screenshot `https://github.com/hakuna-m/wubiuefi/wiki/MOKManager_de` (GPL-2.0)
- CC-Logos `https://creativecommons.org/`