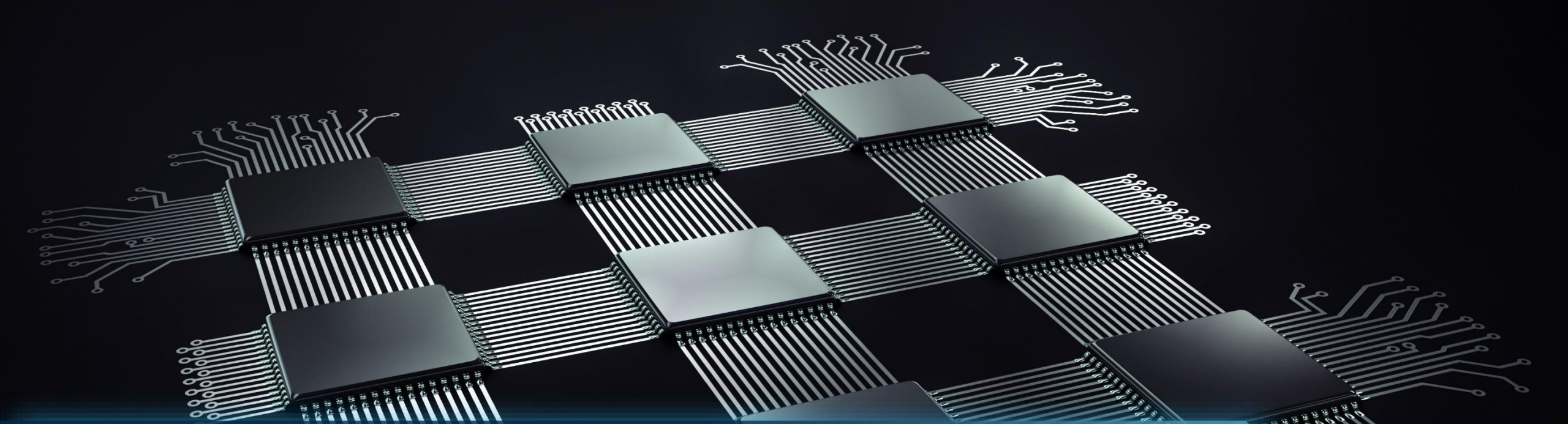


Operation System Security

02 – Secure Boot & Hauptspeicherverschlüsselung

SECURNITE

Heutige Agenda

Angriffsfläche: Bootprozess

Exkurs: Virtualisierung

Unified Extensible Firmware Interface (UEFI) – Secure Boot

Angriffsfläche: Hauptspeicher

Exkurs: Cloud Systeme

Motivation für Hauptspeicherverschlüsselung

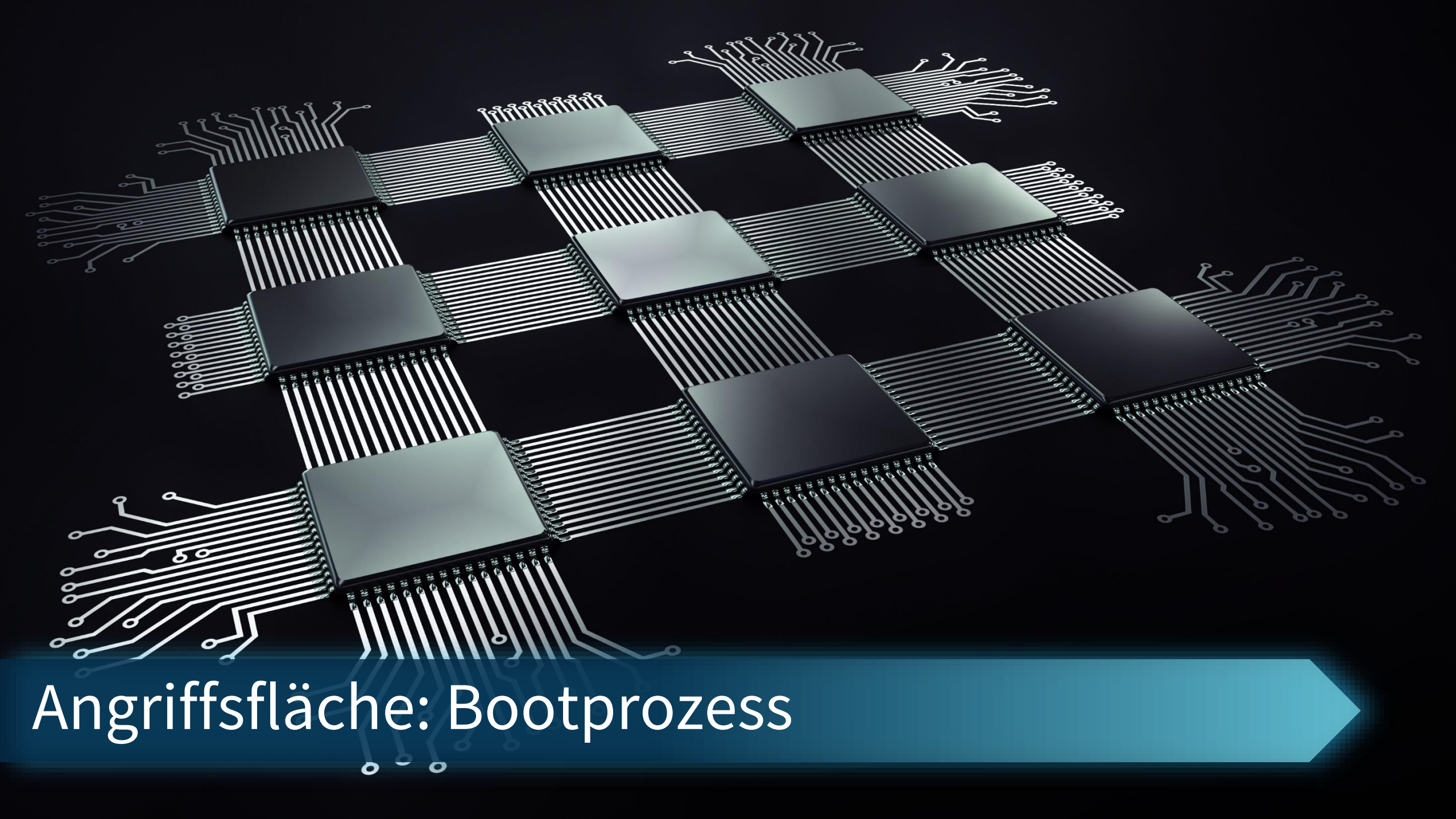
Trusted Platform Module (TPM)

AMD & Intel

Einsatzgebiete für Hauptspeicherverschlüsselung

Exkurs: Programmieren in Python

SECURNITE

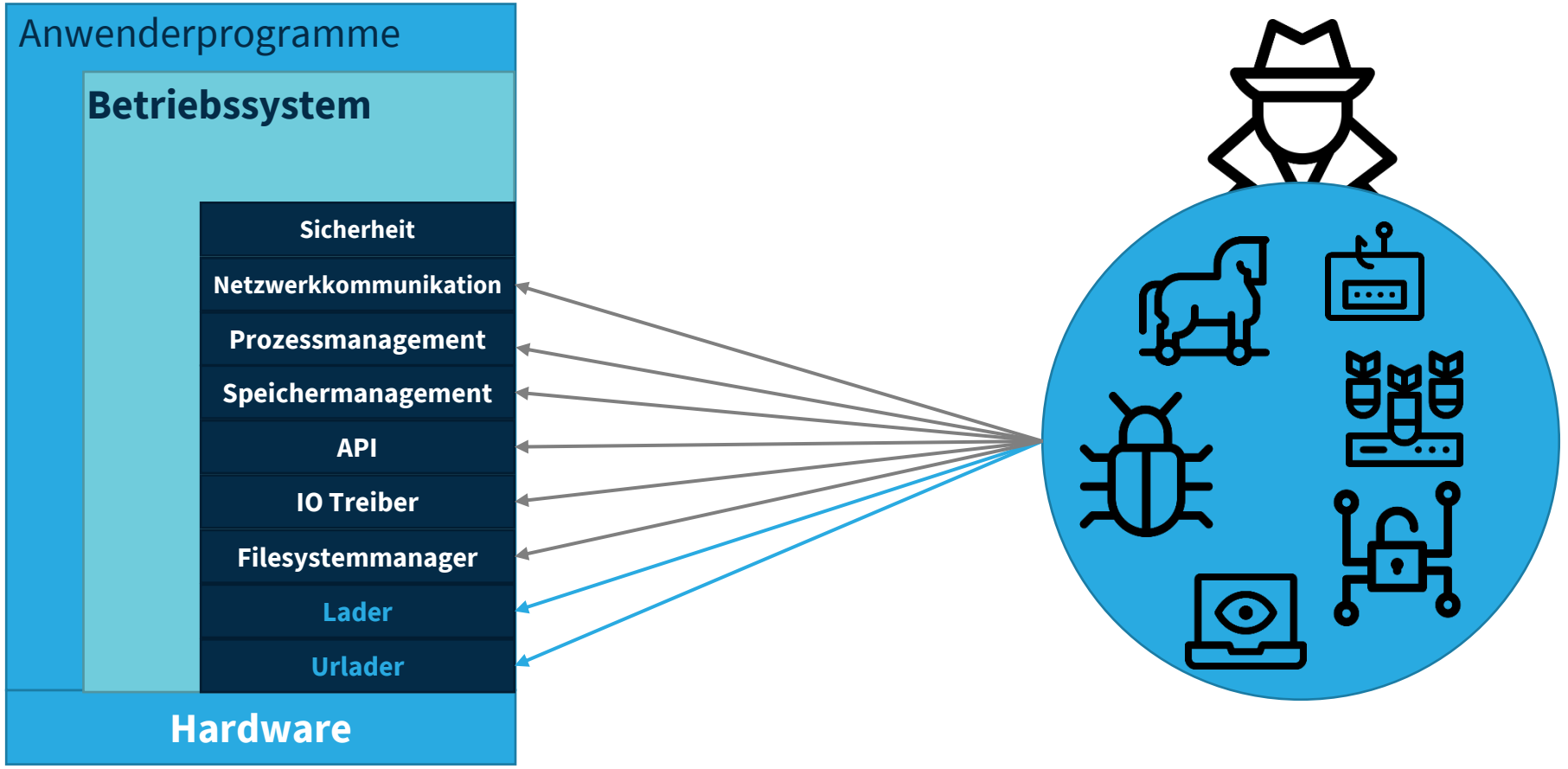


Angriffsfläche: Bootprozess

Boot-Ablauf: Vom «Power On» zum Betriebssystem

- **Urlader** lädt Programm von Bootsektor der CD oder USB Stick, welches in der Ausführung den **Lader** nachlädt und im Bootsektor der Festplatte (MBR oder Protective-MBR) speichert
- Bei jedem Start lädt der Urlader ab jetzt den Lader (Betriebssystem), d.h. der Urlader lädt vom Bootsektor eines bootfähigen Mediums den Betriebssystemlader
- Optional: der Urlader kann auch den Boot Manager laden, welcher dann wiederrum das Betriebssystem lädt

Angriffsvektoren auf den Boot-Ablauf



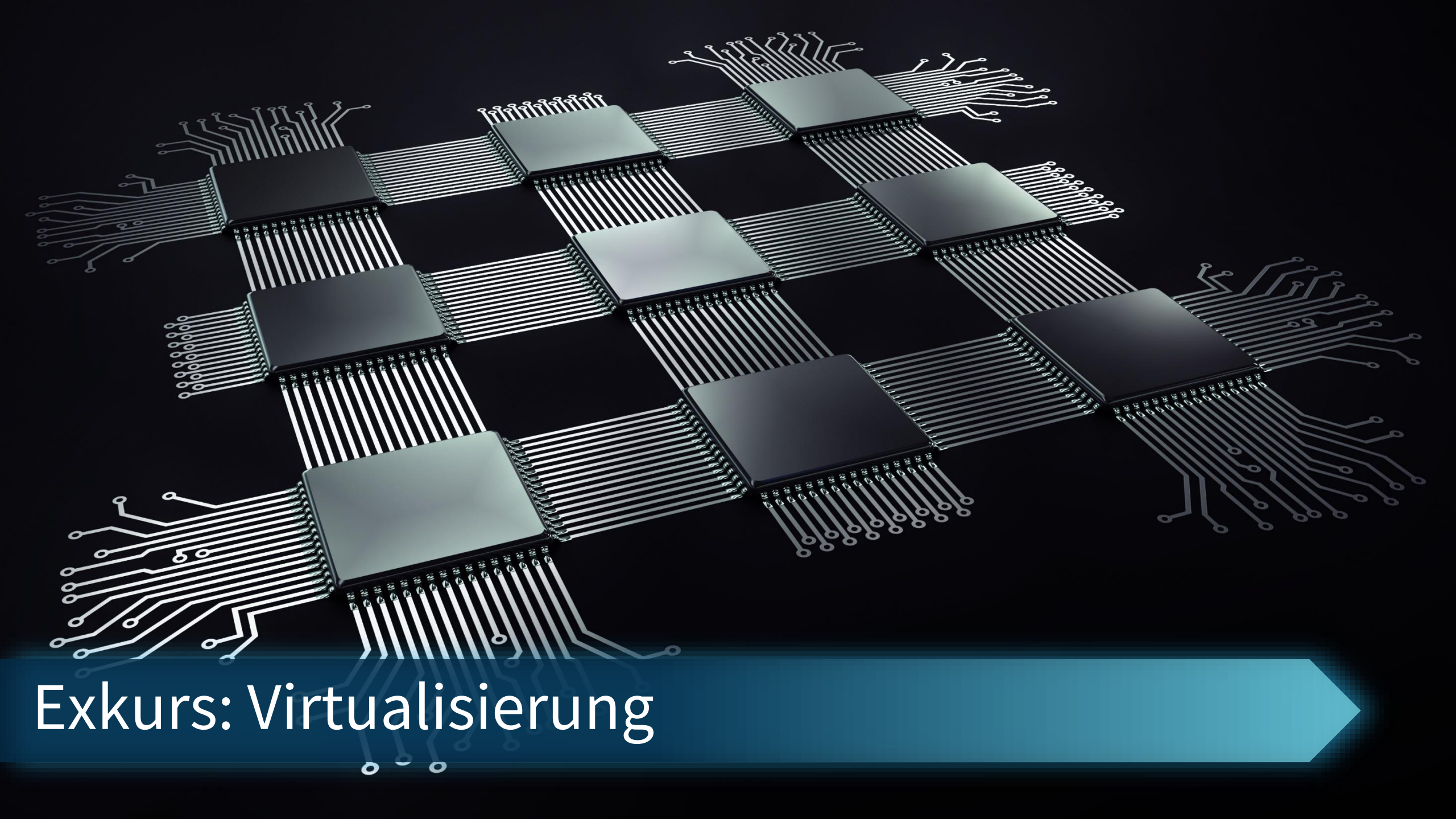
Boot Sektor Virus

Boot Sektor Virus

- Infiziert den Primary Boot Record von Festplatten, verschlüsselt ihn eventuell
- Wird geladen, bevor das Betriebssystem geladen wird
- Wird durch Dropper verbreitet
- Schutzmassnahmen bereits in modernen BIOS / UEFI enthalten (Secure Boot)

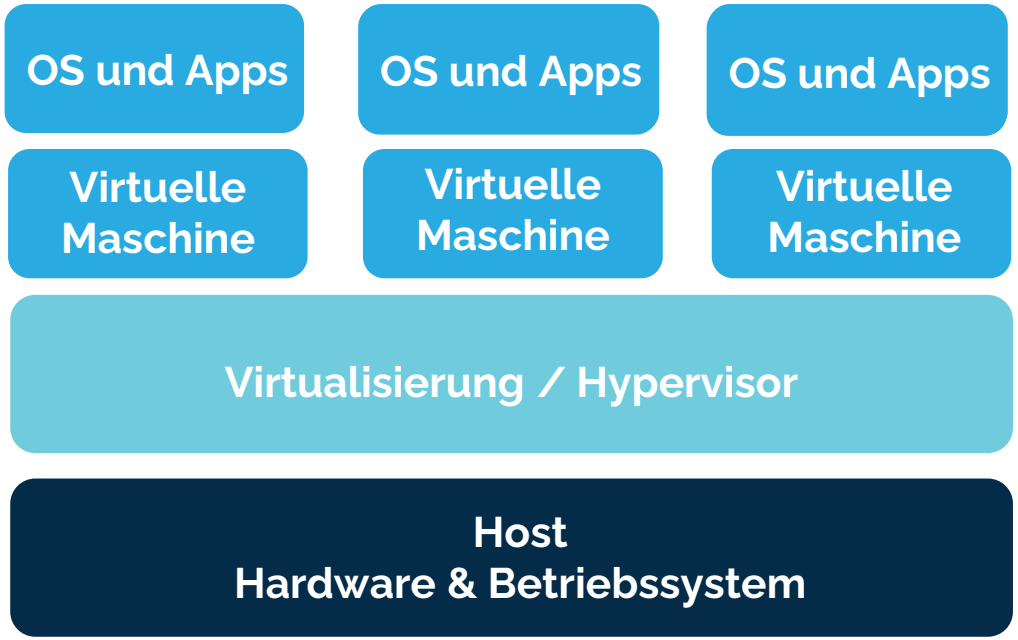
VMBR

- Startet eine VM im Bootvorgang und lädt das vorhandene Betriebssystem hinein
- Für das Betriebssystem nicht sichtbar
- Bootreihenfolge muss durch Ausnutzen von Schwachstellen verändert werden
- Existenz eines VMBR kann in der darunterliegenden Schicht (Hardware) erkannt werden, z.B. im Speicher oder durch Secure Boot



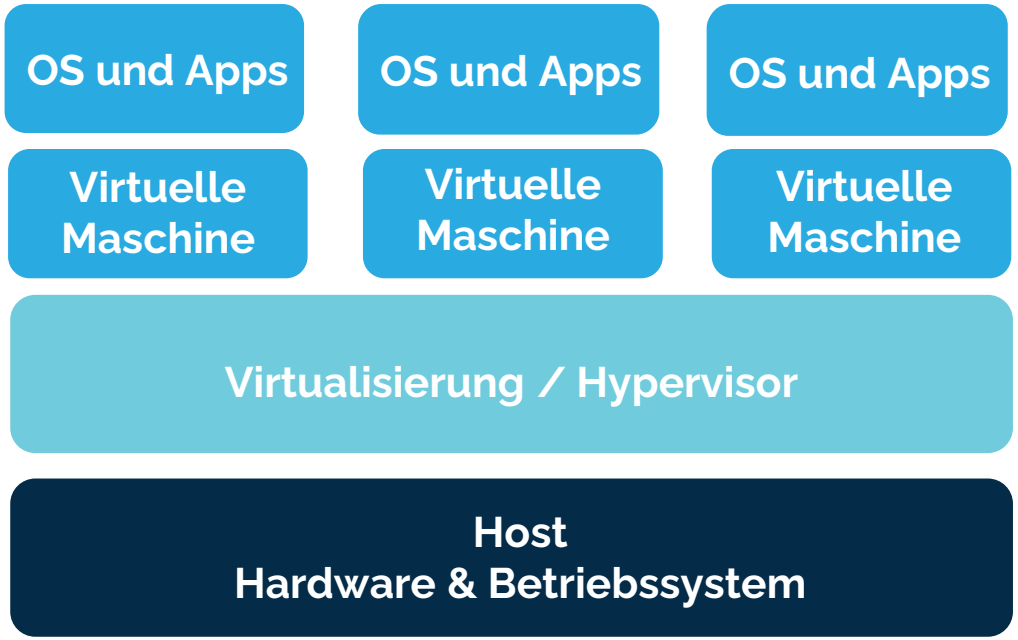
Exkurs: Virtualisierung

Was ist Virtualisierung?



- **Abstraktion** physischer IT Ressourcen (Hardware, Software Speicher etc.)
- Bereitstellung auf **virtueller** Ebene
- **Hardware Virtualisierung :** Bereitstellung von Hardware über virtuelle Maschinen

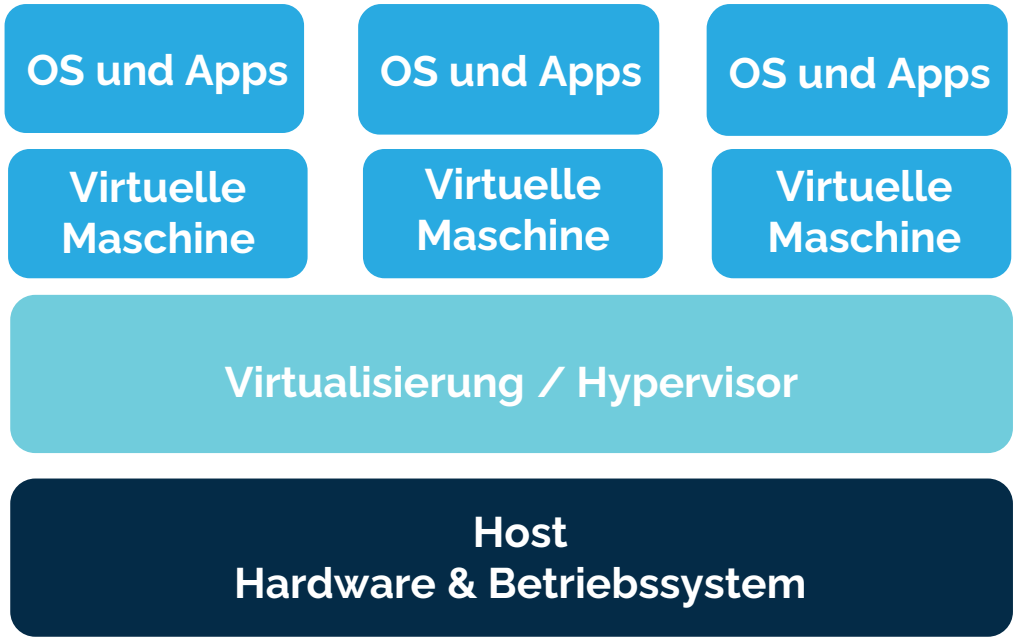
Was ist Virtualisierung?



Hardware Virtualisierung

- Bereitstellung von Hardware mittels **Software**
- **unabhängig** von der **physischen Grundlage**

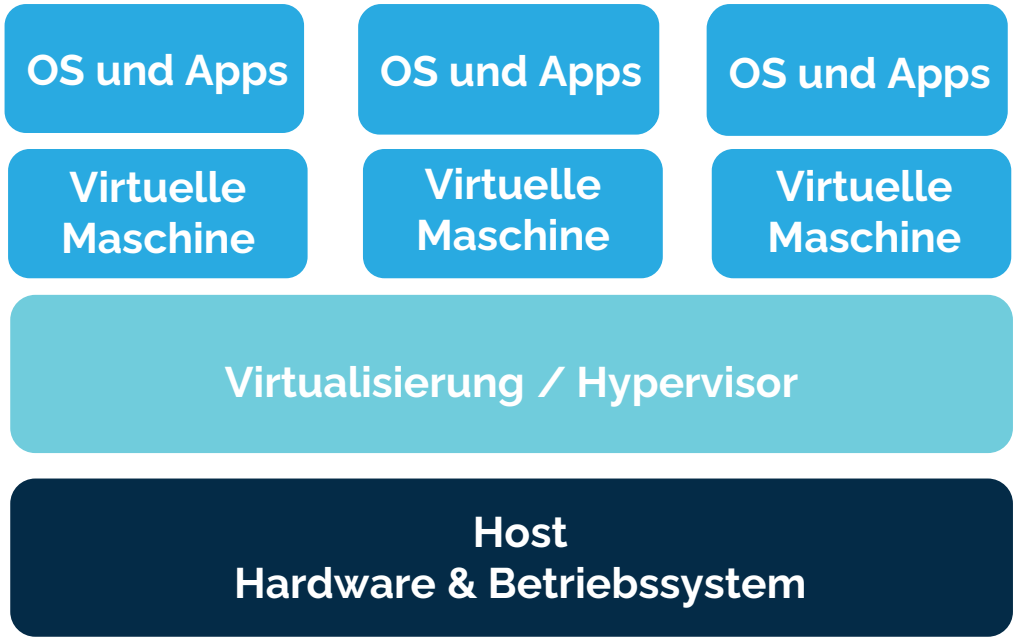
Was ist Virtualisierung?



Virtuelle Maschinen

- **Virtuelle** Computer
- Verhalten sich dem Nutzer gegenüber als **normale Computer** mit physischer Hardware und Betriebssystem
- **Nutzt** die mittels Software bereitgestellten physischen **Ressourcen** des **Host** Systems

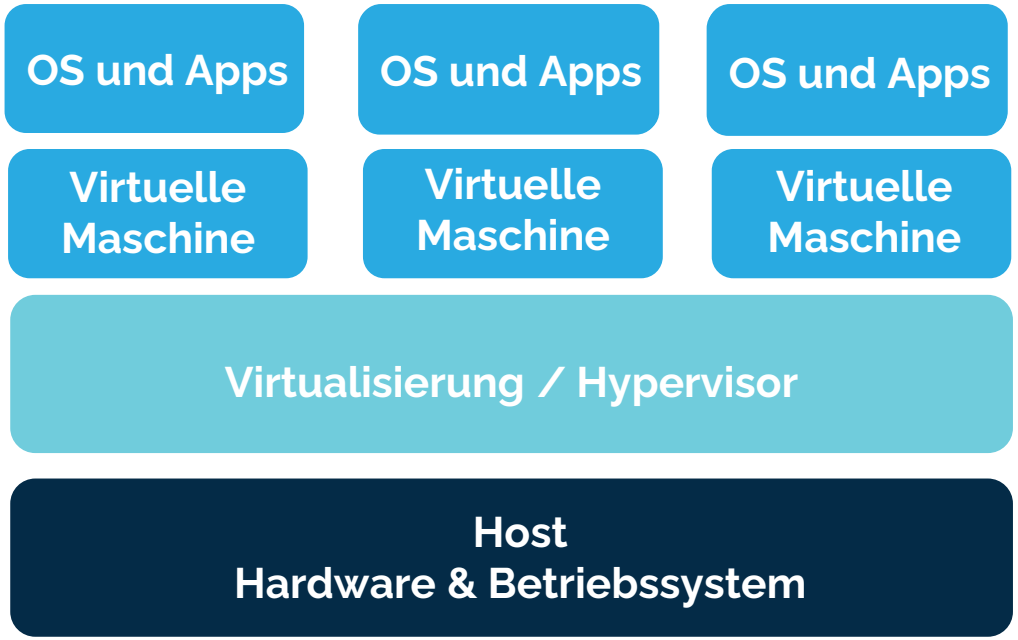
Was ist Virtualisierung?



Hypervisor

- **Verwaltet** physische Resscourcen des Host Systems
- CPU, RAM, Festplatte, Peripherie
- **Teilt** die physischen Resscourcen **auf** verschiedene Gast Systeme (virtuelle Maschinen) auf

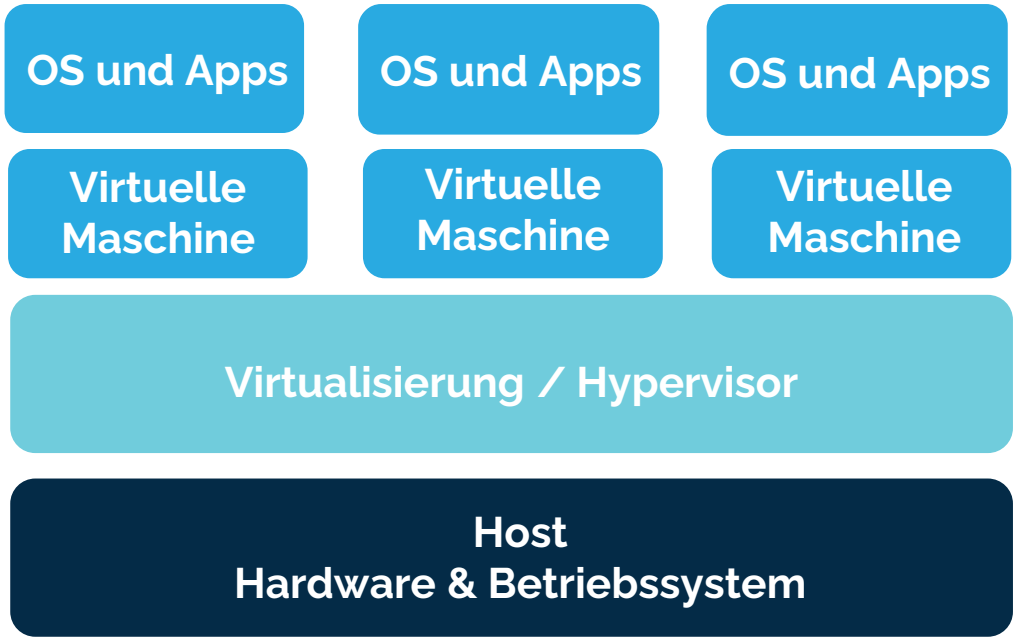
Was ist Virtualisierung?



Hypervisor

- Zwei **Arten** von Hypervisoren:
  - **Typ 1** (Bare Metal): Setzen direkt **auf** der **Hardware** auf
  - **Typ 2** (Hosted HV): Laufen **im Betriebssystem** des Hosts und nutzen dessen Gerätetreiber zum Zugriff auf die Hardware

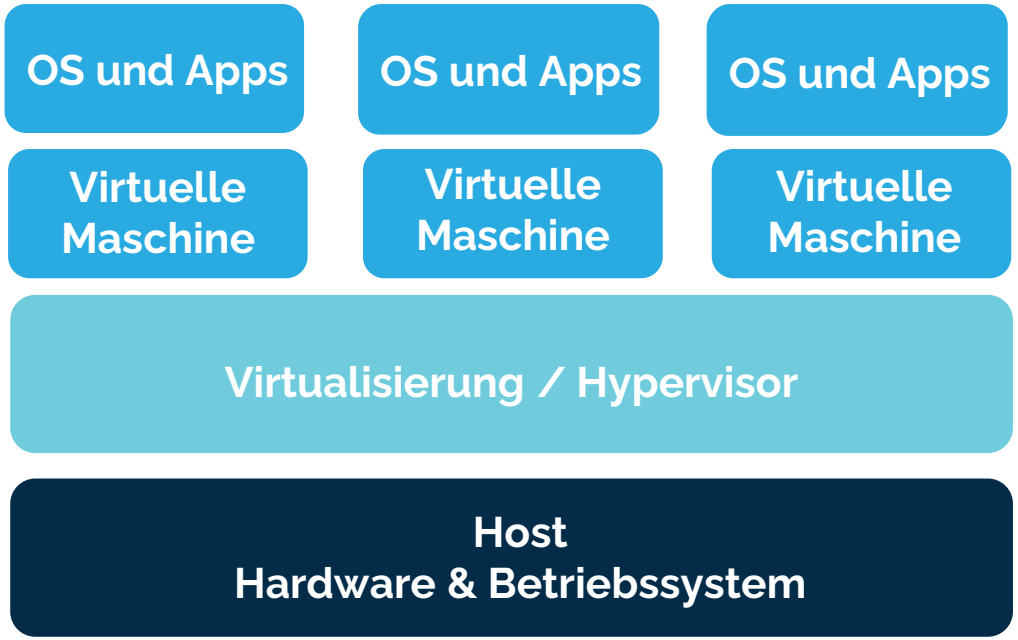
Was ist Virtualisierung?



Vollvirtualisierung

- Häufigste Form der Virtualisierung
- Hypervisor stellt **Gastsystemen** eine vollständige virtuelle Umgebung zur Verfügung

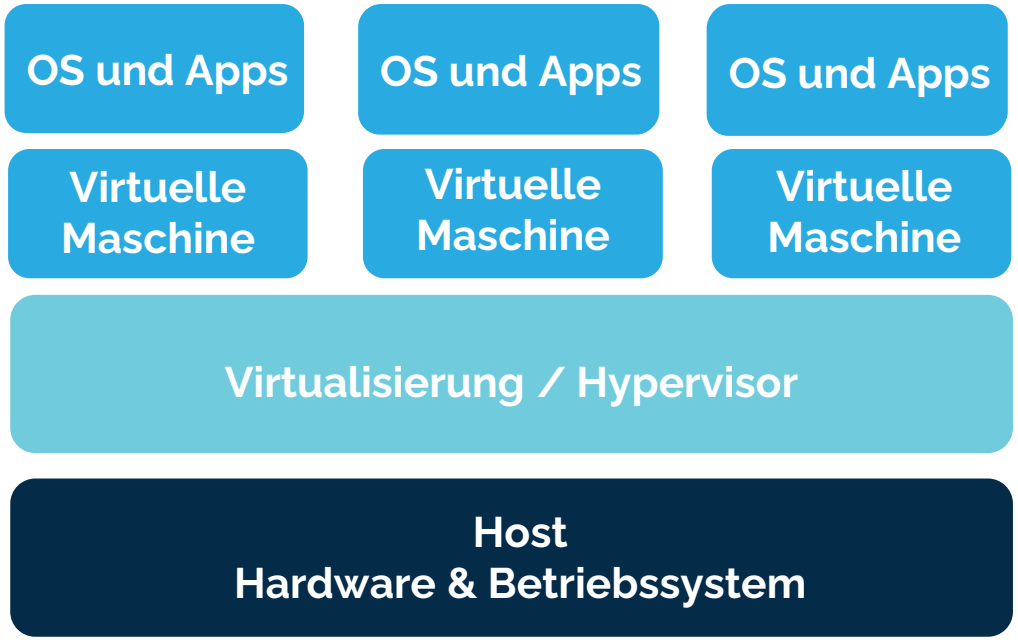
Was ist Virtualisierung?



Vollvirtualisierung

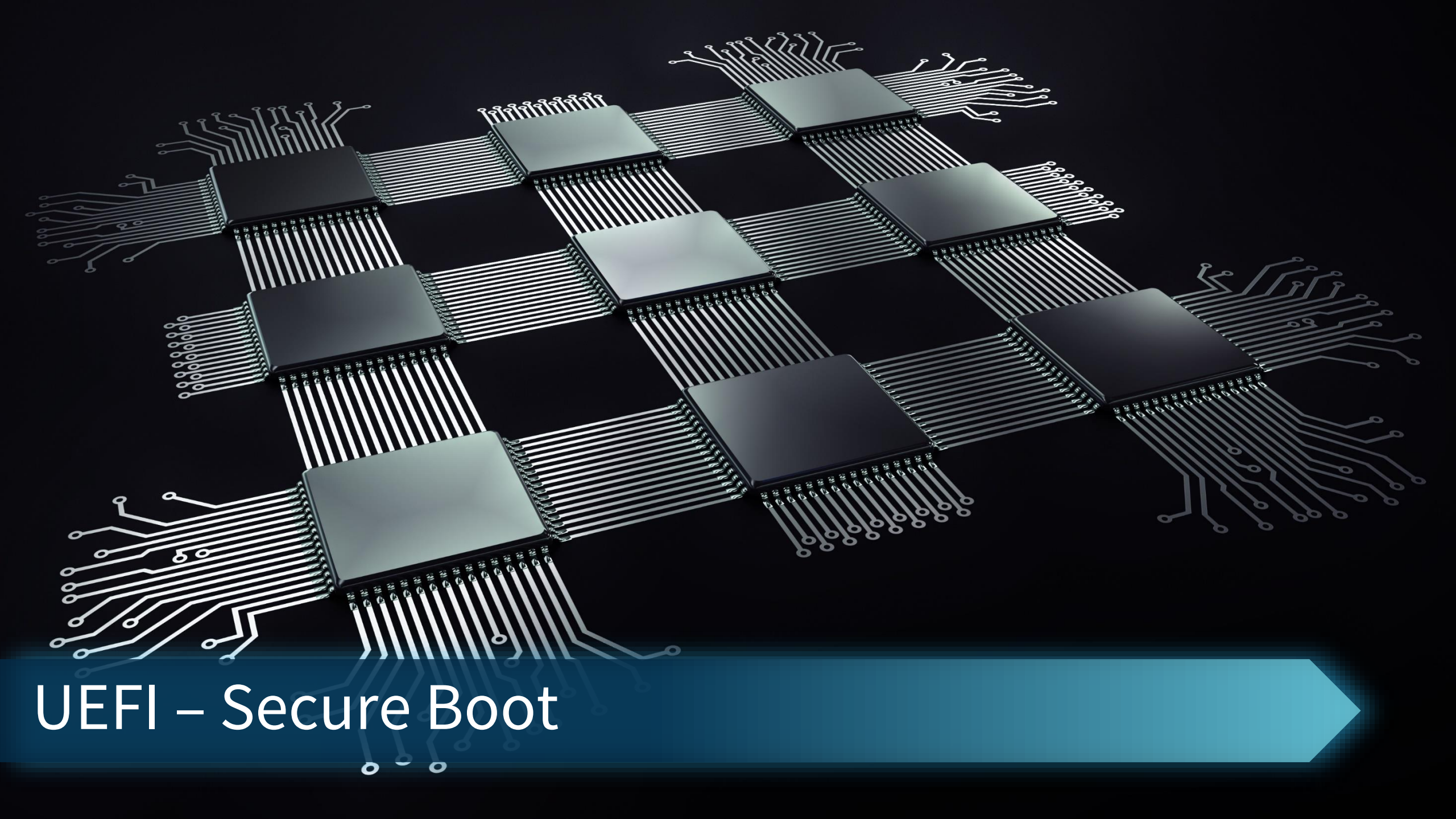
- Eigenes **Kontingent** an Hardware Ressourcen
- **Host** Betriebssystem und physische **Hardware** bleiben Gastsystem **verborgen**

Was ist Virtualisierung?



Virtualisierungssoftware

- VMware
- VirtualBox
- Microsoft Hyper-V
- Parallels Desktop



UEFI – Secure Boot

Unified Extensible Firmware Interface (UEFI)

- Konzeption als **industrieller Standard**, von **Intel** veröffentlicht, seit 2005 von vielen Firmen weiterentwickelt (AMD, Microsoft, HP, …)
- **Schnittstellendefinition**, **operative Ebene** zwischen Firmware und Betriebssystem
- Nachfolger des **BIOS** (seit 2020 teilweise nicht mehr abwärtskompatibel) mit grafische Bedienoberfläche
- Bestandteil der **Mainboard Firmware** mit eigenem Speicher
- Programmiersprache C, Modularer Aufbau

Unified Extensible Firmware Interface (UEFI)

- **kann** um spezielle Funktionen und Programme **erweitert werden** (z. B. Digital Rights Management, Spiele, Webbrowser, Hardware-Monitoring, Lüftersteuerung, …)
- Integrierter **Bootmanager**, der verschiedene **Bootloader** für unterschiedliche Betriebssysteme verwaltet
- Möglichkeit zur frühzeitigen **Integration** von **Treibern** (die dann nicht mehr das Betriebssystem laden muss)

Unified Extensible Firmware Interface (UEFI)

- UEFI-Shell für die **Diagnose** und **Fehlersuche**
- **Netzwerkfähigkeit** auch **ohne** aktives **Betriebssystem**, ermöglicht **Fernwartung** (Remote-Upgrade von Firmware-Komponenten oder der ganzen Firmware) und das **Booten** übers **Netzwerk**
- Erhöhte **Sicherheit** durch Secure-Boot-Feature
- **Voraussetzung**: GUID Festplattenpartitionierung

Durch die direkte **Netzanbindung** in der Boot-Phase können **Schädlinge** auf einen Rechner gelangen, bevor die **Schutzmechanismen** des **Betriebssystems** greifen.

2014 wurde eine erste Sicherheitslücke in der Schnittstelle entdeckt, 2018 identifizierten Experten mit **LoJax** den ersten UEFI-Virus in freier Wildbahn.

Unified Extensible Firmware Interface (UEFI)

Vorteile

- Durch **GPT-Partitionierung** beliebig viele primäre GPT-Partitionen möglich
- **Bootlaufwerke** können erstmalig die Festplattenkapazität von **2,2 TB** überschreiten
- **Pre-Boot-Anwendungen** sind möglich z. B.
  - Aufruf und Nutzung von Diagnose-Werkzeuge
  - Backup-Lösungen
- Bootet **schneller** als Legacy-BIOS-Systeme

Exkurs: UEFI Rootkit LoJax

- Hackergruppe **APT28** / Sofacy / Fancy Bear / Sednit / …
- beruht auf einer **Technik** und Software, die Lojack, früher bekannt als Computrace, als **Diebstahlsicherung** einsetzt
- ein UEFI-BIOS-Modul injiziert beim Booten des Systems **Überwachungssoftware** in Windows, bevor dieses überhaupt gestartet ist
- Malware kann selbst eine **Windows-Neuinstallation** oder einen Austausch der Festplatte überleben
- Installation erfordert **keinen physischen Zugang** zum System

Exkurs: UEFI Rootkit LoJax

- APT28 nutzt sehr gezielte **Spear-Phishing-Mails**, um Trojaner bei ihren Zielpersonen einzuschleusen
- Durch **Ausnutzen** von **Sicherheitslücken** kann der Trojaner **modifizierte Firmware** mit eigenem UEFI-Code in den Flash-Speicher des Systems schreiben
- Aktiviertes **Secure Boot** und ein **aktuelles UEFI-BIOS** sollten derartige Attacken verhindern

Exkurs: UEFI Rootkit APT28

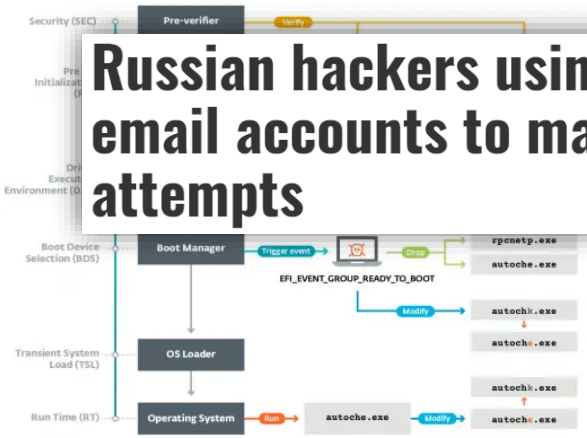
- **Gruppen:** SNAKEMACKEREL, Swallowtail, Group 74, Sednit, Sofacy, Pawn Storm, Fancy Bear, STRONTIUM, Tsar Team, Threat Group-4127, TG-4127
- Wird Russland zugeordnet
- seit mindestens 2004 aktiv

APT28 Mounts Rapid, Large-Scale Theft of Office 365 Logins

heise Security

Lojax: Der Spion, der aus dem BIOS kam

27.09.2018 13:00 Uhr
Jürgen Schmidt



Russian hackers using stolen corporate email accounts to mask their phishing attempts

Beim Booten via UEFI schleust Lojax die Datei autoche.exe in das Windows-System ein und sorgt dafür, dass sie gestartet wird. (Bild: Eset)

Die vermutlich auch für den Bundestag-Hack verantwortliche Hackergruppe APT28 infiziert Systeme auch per UEFI-BIOS.

Die Antiviren-Forscher von Eset haben auf mindestens einem System ein zu Spionagezwecken eingesetztes UEFI-Rootkit entdeckt. Man wusste zwar bereits, dass etwa die NSA und andere professionell arbeitende Cyber-Kriminelle solche Techniken einsetzen. Direkt in Aktion wurde allerdings bisher keines dieser ausgefeilten Spionage-Werkzeuge beobachtet.

Russia's Fancy Bear Hackers Likely Penetrated a US Federal Agency

New clues indicate that APT28 may be behind a mysterious intrusion that US officials disclosed last week.

Exkurs: UEFI Rootkit APT28

- **MITRE** führt eine Liste der von ATP28 eingesetzten Techniken und Software unter https://attack.mitre.org/groups/G0007/
- **NIST** stellt ein Advisory bereit, welches IOCs für Malware von APT28 sammelt, unter https://www.ncsc.gov.uk/news/indicators-of-compromise-for-malware-used-by-apt28

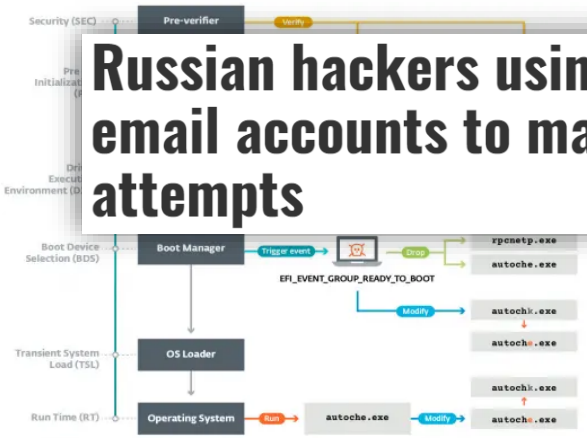
APT28 Mounts Rapid, Large-Scale Theft of Office 365 Logins

heise Security

Lojax: Der Spion, der aus dem BIOS kam

27.09.2018 13:00 Uhr
Jürgen Schmidt

Russian hackers using stolen corporate email accounts to mask their phishing attempts



Beim Booten via UEFI schleust Lojax die Datei autoche.exe in das Windows-System ein und sorgt dafür, dass sie gestartet wird. (Bild: Eset)

Die vermutlich auch für den Bundestag-Hack verantwortliche Hackergruppe APT28 infiziert Systeme auch per UEFI-BIOS.

Die Antiviren-Forscher von Eset haben auf mindestens einem System ein zu Spionagezwecken eingesetztes UEFI-Rootkit entdeckt. Man wusste zwar bereits, dass etwa die NSA und andere professionell arbeitende Cyber-Kriminelle solche Techniken einsetzen. Direkt in Aktion wurde allerdings bisher keines dieser ausgefeilten Spionage-Werkzeuge beobachtet.

Russia's Fancy Bear Hackers Likely Penetrated a US Federal Agency

New clues indicate that APT28 may be behind a mysterious intrusion that US officials disclosed last week.

Gruppenübung (20 Minuten)

- **Gruppe 1:** Recherchieren Sie die Hintergründe zum **Rootkit von Sony**, mit dem verhindert werden sollte, dass Käufer CDs kopieren. Gehen Sie besonders auf folgende Punkte ein:
  - Wie gelangte das Rootkit auf die PCs der Käufer?
  - Was waren die Auswirkungen?
  - Wie hätte der Fall verhindert werden können?

- **Gruppe 2:** Recherchieren Sie die Hintergründe zum **PoC Rootkit Lighteater** für UEFI. Gehen Sie besonders auf folgende Punkte ein:
  - Welche Schwachstelle wird ausgenutzt?
  - Welches Risiko geht von UEFI Rootkits aus?

- **Stellen Sie Ihre Ergebnisse vor (max. 5 Minuten)**

Secure Boot

- Software-Komponenten (Teile der UEFI-Firmware, Bootloader, Betriebssystemkernel usw.) werden **verifiziert**, bevor sie gestartet werden
- Für die Verifizierung werden **kryptografische Signaturen** herangezogen, die zuvor in der Signaturdatenbank der UEFI-Firmware hinterlegt wurden
- Ist das Gegenüber durch Viren manipuliert oder besitzt es keine Signatur bzw. keinen als gültig eingetragenen Schlüssel, dann **bricht** das System den **Systemstart ab**
- Kompatibel mit **Trusted Platform Module (TPM):** spezifizierter Chip, der Computer und andere Geräte mit weitreichenden Sicherheitsfunktionen ausstattet

Secure Boot

Nachteile

- **Plattform Key** befindet **nicht** unter der **Kontrolle** des Endkunden

- Für den nachträglichen **Einbau** von **Hardware** in Systemen muss sicher gestellt sein, dass sich der Key des Hardware-Herstellers des neuen Bauteils auf dem System befindet.

Secure Boot

Nachteile

- Alternative Betriebssysteme bzw. **Dual-Boot-Konfigurationen** werden erschwert:
  - Linux-Installationen nur nach einer **Deaktivierung** von Secure Boot möglich, oder
  - **Geeignete** Linux Distribution muss verwendet werden, oder
  - **Schlüssel** muss auf der Plattform **vorhanden** sein

- Beim **Dual-Boot** muss zwischen Secure und Nicht-Secure Boot **gewechselt** werden, um **signierte** und **nicht signierte** Betriebssysteme zu starten.

Secure Boot - Komponenten

- UEFI-Spezifikation (ab) 2.3 **Komponenten** und **Verfahren**:
  - Programmierschnittstelle für den **Zugriff** auf kryptographisch geschützte **UEFI-Variablen** im Flash-Speicher
  - Format zum **Speichern** von **X.509-Zertifikaten** in UEFI-Variablen
  - Verfahren zur **Validierung** des **Bootloaders** und der **Treiber** mithilfe von AuthentiCode-Signaturen
  - Mechanismus für den **Widerruf** (Revocation) kompromittierter Zertifikate und Signaturen.

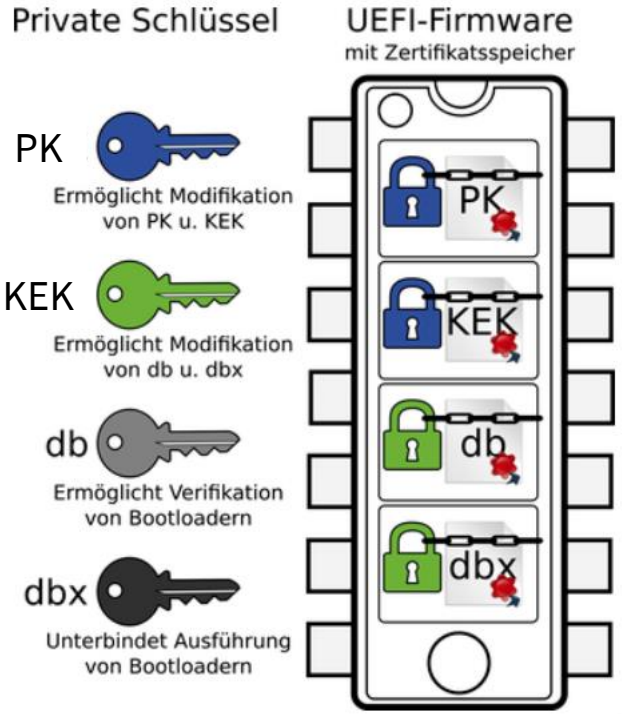


Abbildung 1: Diese Schlüssel können am Secure-Boot-Prozess beteiligt sein.

Secure Boot - Komponenten

- UEFI **Schlüssel:**
  - **Plattform Key** (**PK**): Meist ein Schlüssel des Hardware-Herstellers (OEM), PK erlaubt die **Manipulation** der **KEKs**, nur **ein** einziger **PK** möglich
  - **Key Exchange Key** (**KEK**): **Mehrere** Zertifikate **möglich**, unterschiedliche Schlüssel für unterschiedliche OS-Anbieter möglich, zum Beispiel: Microsoft KEK CA; KEK erlaubt die **Manipulation** der **db** und **dbx**

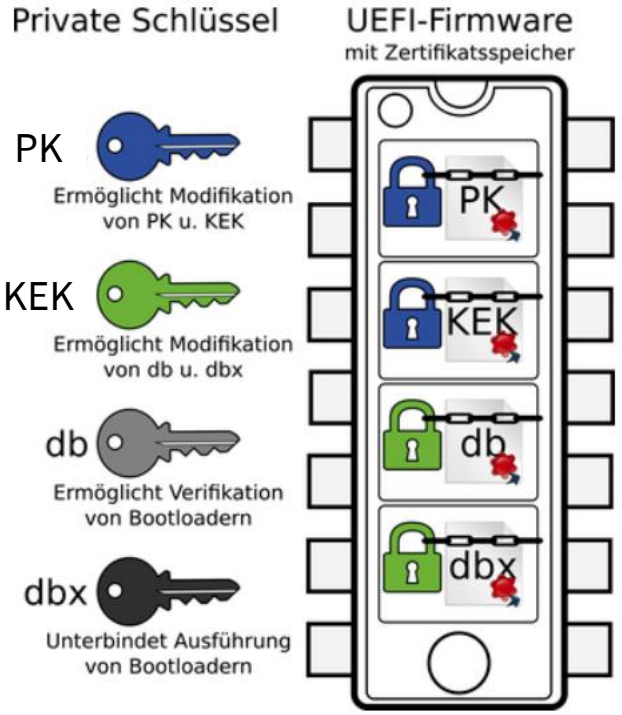


Abbildung 1: Diese Schlüssel können am Secure-Boot-Prozess beteiligt sein.

Secure Boot - Komponenten

- UEFI **Schlüssel:**
  - **Autorisierte DB** (**db**): **Mehrere** Zertifikate und Hashes **möglich**, zum Beispiel: Microsoft Windows Production CA; zur **Identifizierung** von vertrauenswürdigem **Code**.
  - **Nicht autorisierte DB** (**dbx**): **Mehrere** Zertifikate oder Hashes **möglich**; zur **Identifizierung** von **kompromittiertem Code** oder **Schadcode**.

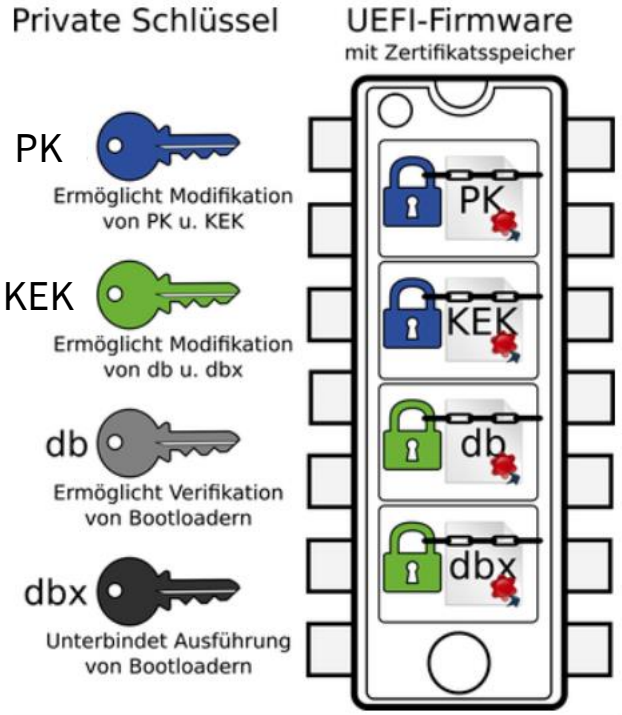
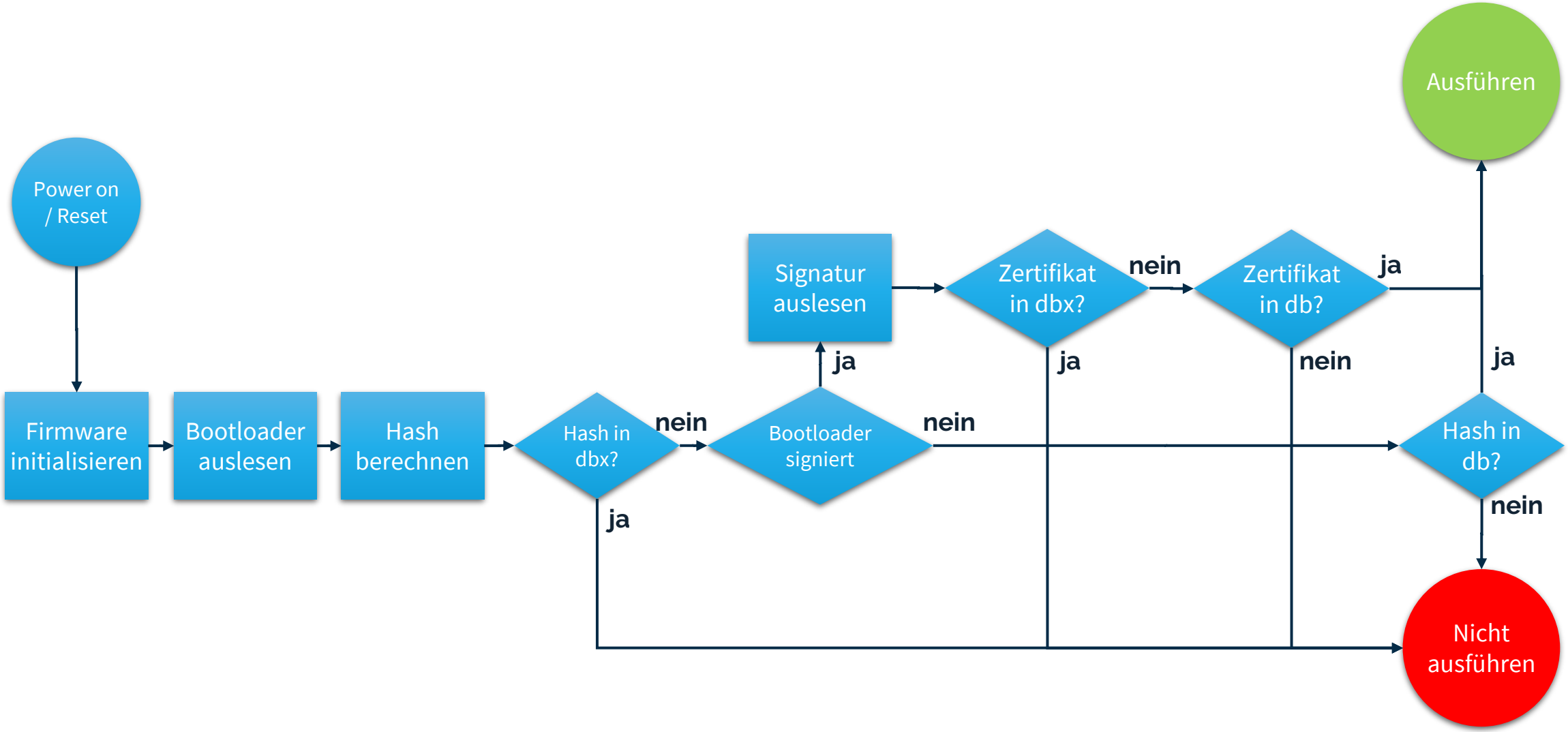


Abbildung 1: Diese Schlüssel können am Secure-Boot-Prozess beteiligt sein.

Secure Boot - Funktionsweise



Secure Boot - Modi

Setup Mode

- Zertifikatsspeicher ist **nicht** vor **Manipulation geschützt**
- Es ist insbesondere möglich, aus einem **laufenden Betriebssystem** heraus **Zertifikate** als auch Hashes in die UEFI-Zertifikatsspeicher **einzufügen** oder zu entfernen
- zur **Einrichtung** von Secure Boot

Secure Boot - Modi

User Mode

- **Manipulation** der Zertifikatsspeicher nur sehr **eingeschränkt** möglich
- Änderungen ohne Authentifizierung aus einem **laufenden Betriebssystem** nicht durchführbar
- Zur Veränderung des db- oder dbx-Zertifikatsspeichers ist eine **Autorisierung** mittels des **privaten Schlüssels** eines im **KEK-Speicher** hinterlegten Zertifikats nötig

Secure Boot - Modi

User Mode

- Zur **Änderung** des **KEK-** und **PK-Speichers** bedarf es hingegen des **privaten Schlüssels** des hinterlegten **Plattform-Key-Zertifikats.**
- **Wechsel** vom User Mode in den Setup Mode **durch** Entfernen des Plattform Keys oder mittels des **UEFI-Setups**

UEFI Secure Boot **verhindert nicht** die Installation von **Malware** oder die Modifikation des Bootloaders, sondern stellt dessen **Vertrauenswürdigkeit** während des Bootvorgangs sicher.

Kann die **Vertrauenswürdigkeit** nicht festgestellt werden, so wird das Booten unterbunden.

SECURNITE

Secure Boot

Mit verschiedenen Betriebssystemen

- **Signieren** der eigenen Bootloader durch **Microsoft**
- **Hinterlegen** eines eigenen **KEK-Schlüssels** durch den **Hardware-Hersteller**
- Erzeugen eines eigenen **Platform Key (PK)** und **Hinterlegen** in der **UEFI-Firmware**

Secure Boot

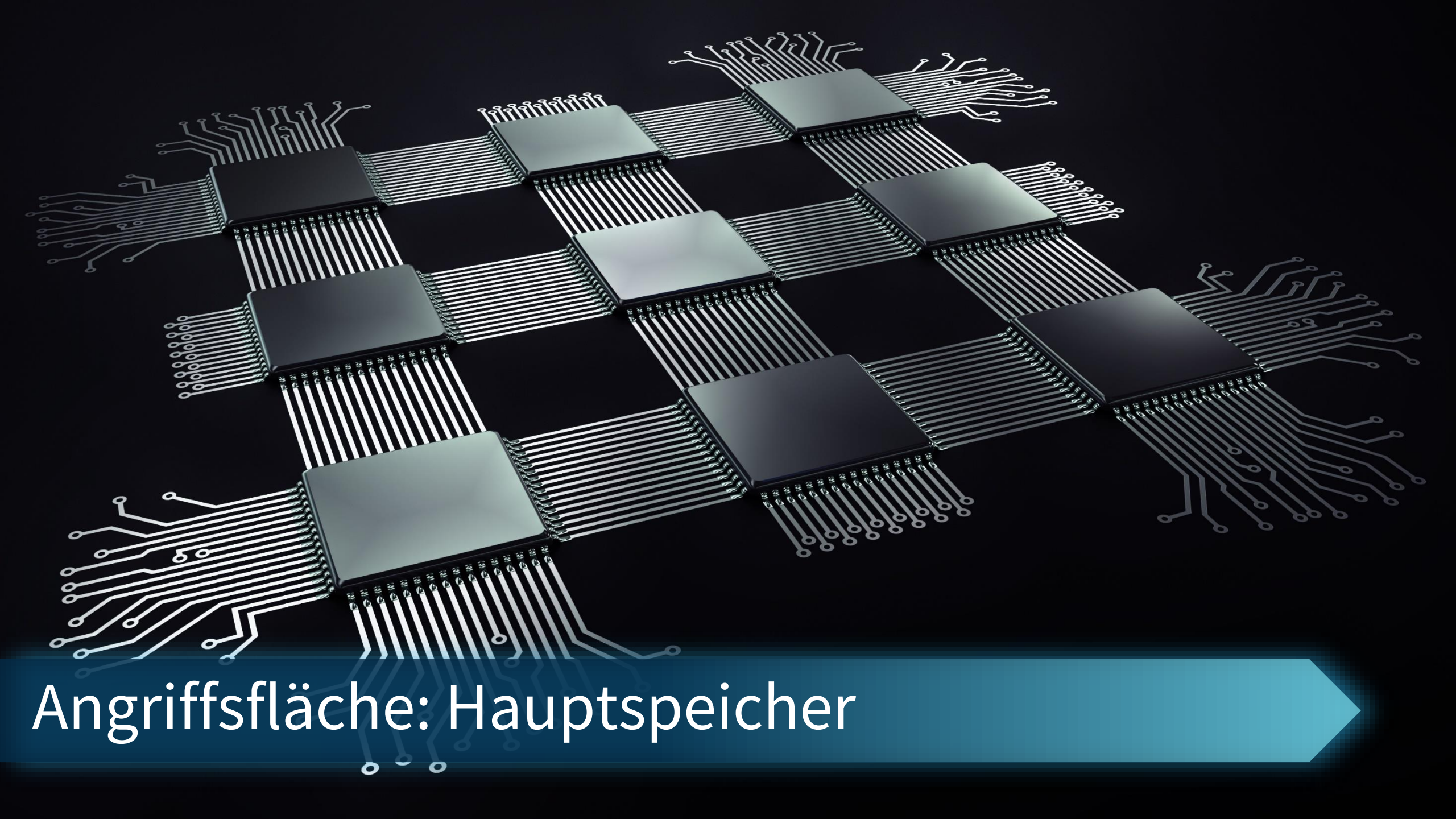
Signaturdienst von Microsoft für Bootloader von Drittanbietern

- Drittanbieter sendet Bootloader an Microsoft
- Microsoft sendet ihn mit einer AuthentiCode-Signatur zurück
- Signatur bestätigt **Unversehrtheit** des Bootloaders
- Signatur erfolgt mit dem Zertifikat Microsoft Corporation UEFI CA 2011
- **Risiken**:
    - Microsoft kann das Zertifikat **widerrufen**
    - Signatur ist nur für eine bestimmte **Zeit gültig**

Secure Boot

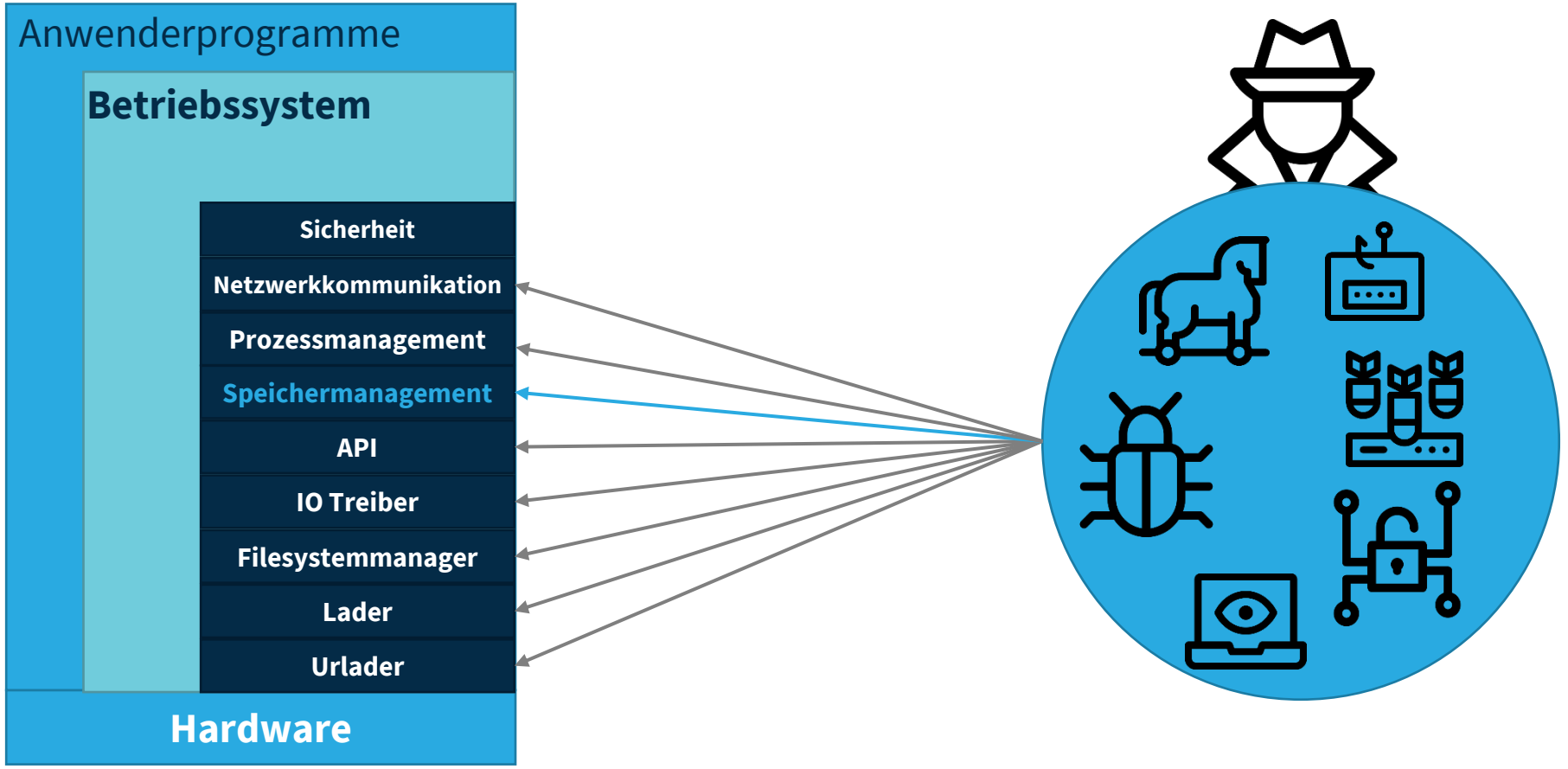
Shim

- einfach strukturierter **Open-Source-Bootloader**
- Auch mit Microsoft Schlüssel verfügbar
- startet indirekt Bootloader, die **nicht von Microsoft signiert** sind
- **überprüft** die **Signatur** des nächsten **Bootloaders**, der zu laden ist
- **Bindeglied** zwischen der von Microsoft geprägten Secure-Boot-Umgebung und Betriebssystemen Dritter

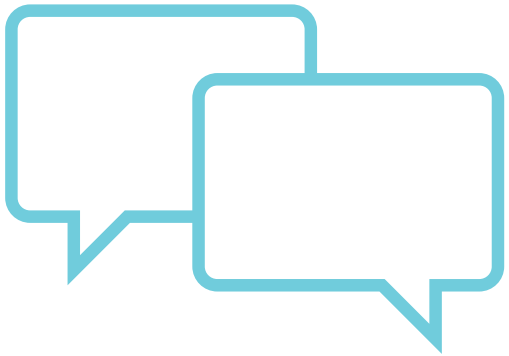


Angriffsfläche: Hauptspeicher

Angriffsvektoren auf den Hauptspeicher



Was ist Hauptspeicherverschlüsselung



- Direkte **Verschlüsselung** von **Daten** in **DIMM** (Dual Inline Memory Module) oder **NVDIMM** (Non Volatile Dual Inline Memory Module) **Speicherbausteinen**

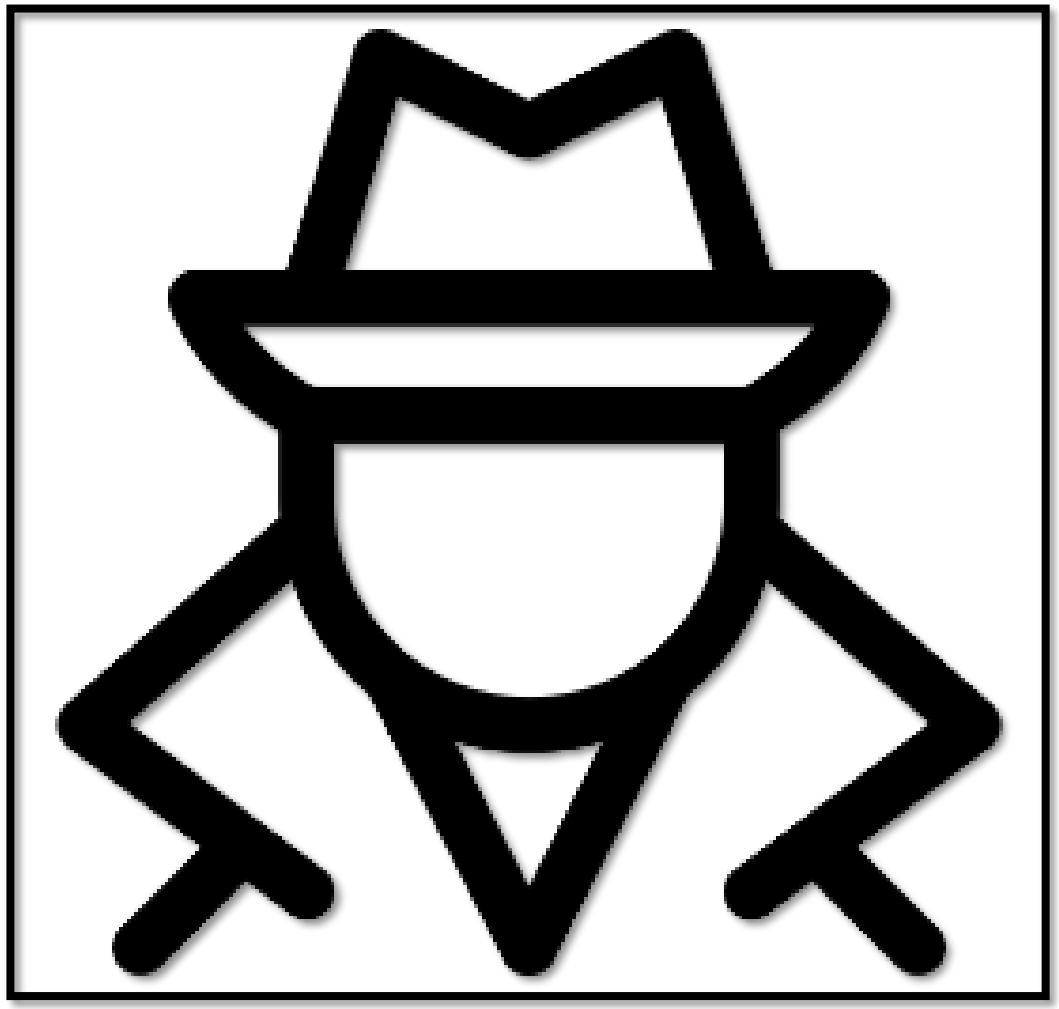
- Mehrere Ebenen
  - **Applikationsentwickler**
  - **Betriebssystem** (z.B. Windows Data Protection API, DAPI)

Diskussion: Welche Aspekte könnten hier ein Problem darstellen?

Gründe für Hauptspeicherverschlüsselung

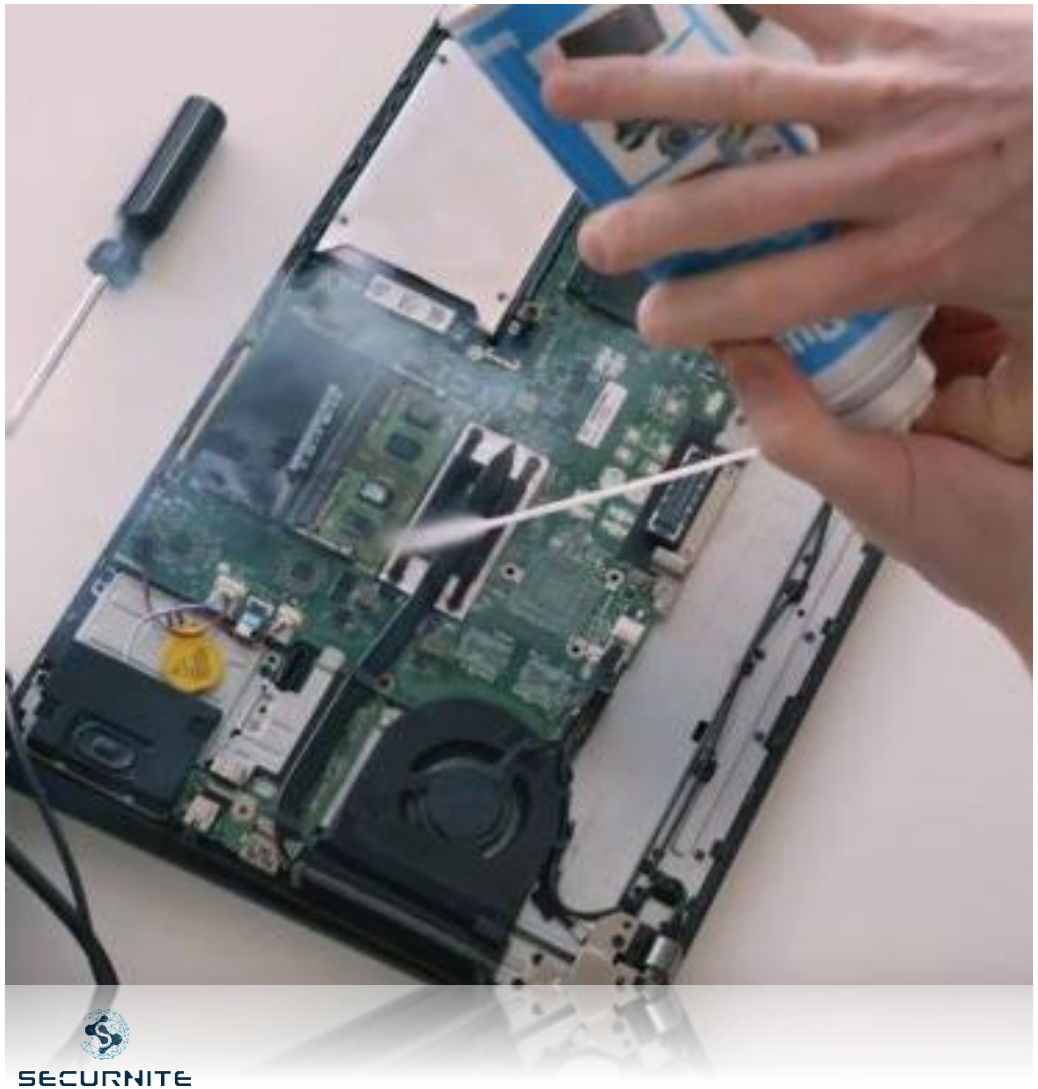
- Daten auf **Datenträgern** verschlüsselt & Daten im RAM im **Klartext**
- Folge: **Auslesen** von **Passwörter** und **vertraulichen** Informationen möglich

Mögliche Angriffe



- **Physischer Zugriff** auf Speicher
- **Unautorisierter** Zugriff durch Benutzer

Angriff: Physischer Zugriff



- NV (non volatile) DIMMs können **gestohlen** und **ausgelesen** werden
- Anzapfen und auslesen der **DRAM Schnittstelle**
- **Einfrieren** und **stehlen** der DIMMs (Cold Boot Attack)

Angriff: Physischer Zugriff

Cold Boot Angriff

- **Stehlen** eines Laptops, sicherstellen der **Stromversorgung** durch Akku/Netzteil
- Lokalisieren des **Speicherbausteins**
- **Kühlen** des Speichers zum Erhalt der Daten (daher Cold Boot)
- Verbinden eines In-System Programmers (**ISP**) mit dem Speicherbaustein
- Beinhaltet **manipuliertes UEFI Image** welches **Secure Boot** ausschaltet und das **Löschen** des Speichers beim Bootvorgang **verhindert**

Angriff: Physischer Zugriff

Cold Boot Angriff

- **Stromversorgung** wird **entfernt** und das **UEFI Image** im Speicher **überschrieben**
- **ISP** wird wieder **entfernt**
- **Strom wiederhergestellt**
- Laptop kann jetzt wieder neu **gebootet** werden, das **korrupte UEFI Image** im Speicher **verhindert** das **Löschen**

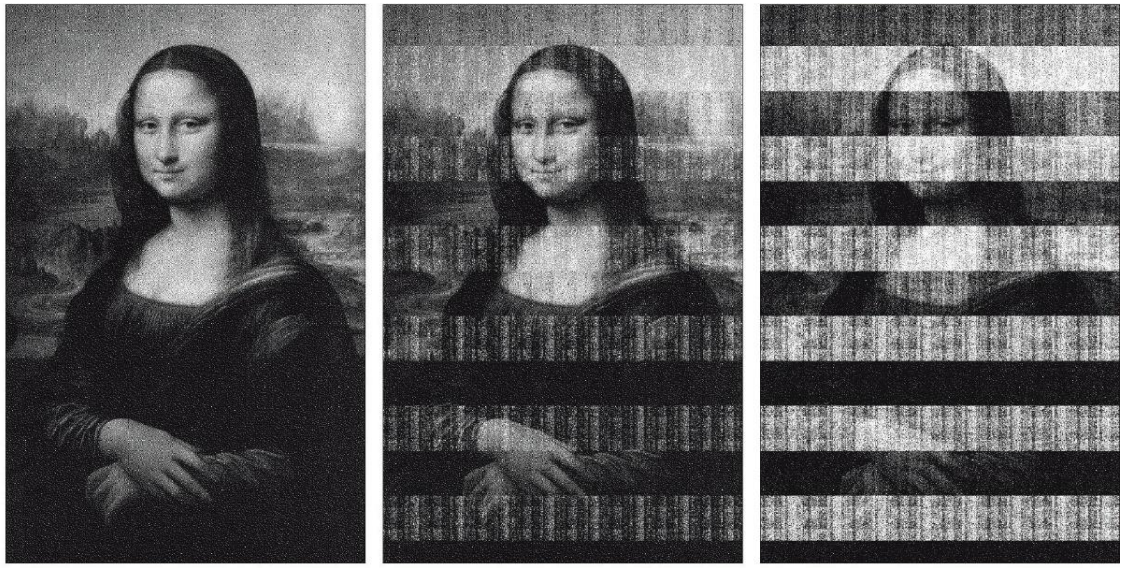
Angriff: Physischer Zugriff

Cold Boot Angriff

- Linux Image vom **USB Stick** booten
- **Auslesen** des **Speichers** mit normalen Linux Tools
- Auslesen von **vertraulichen Daten** die zum Zeitpunkt des Angriffs **im Speicher** waren
- Aber auch **Schlüssel** die dauerhaft **im Speicher** liegen (BitLocker Key)

Angriff: Physischer Zugriff

Cold Boot Angriff

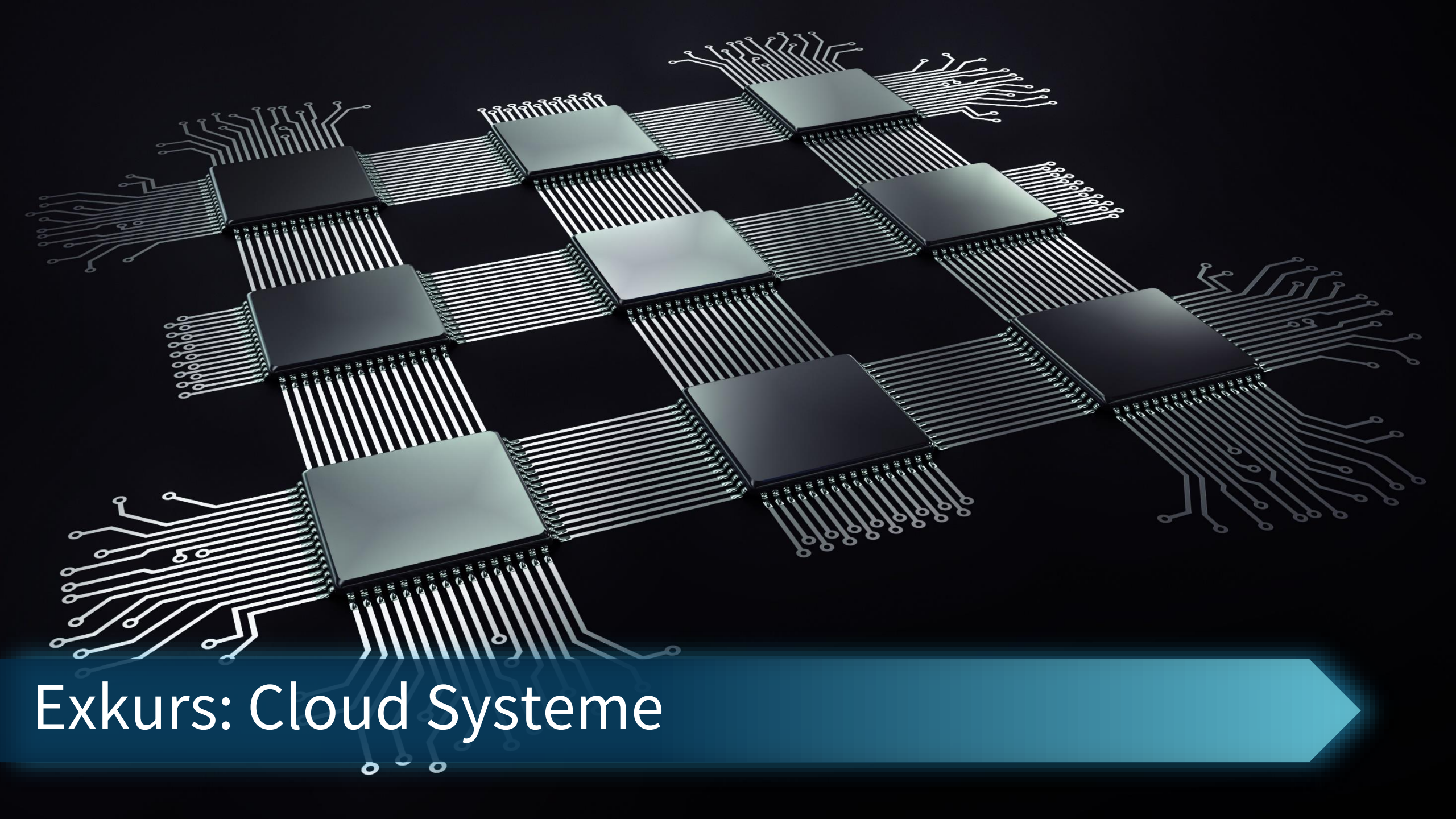


2008 demonstrierten Forscher den Cold-Boot-Angriff, indem sie eine Bilddatei aus dem PC-RAM extrahierten: 5 Sekunden nach dem Abschalten war sie komplett erhalten, nach 30 und 60 Sekunden verschwinden zunehmend Informationen aus den DRAM-Zellen.

https://doi.org/10.1016/j.diin.2016.01.009
(2016, Cold Boot Attacks on Scrambled Memory)

Angriff: Unautorisierter Zugriff

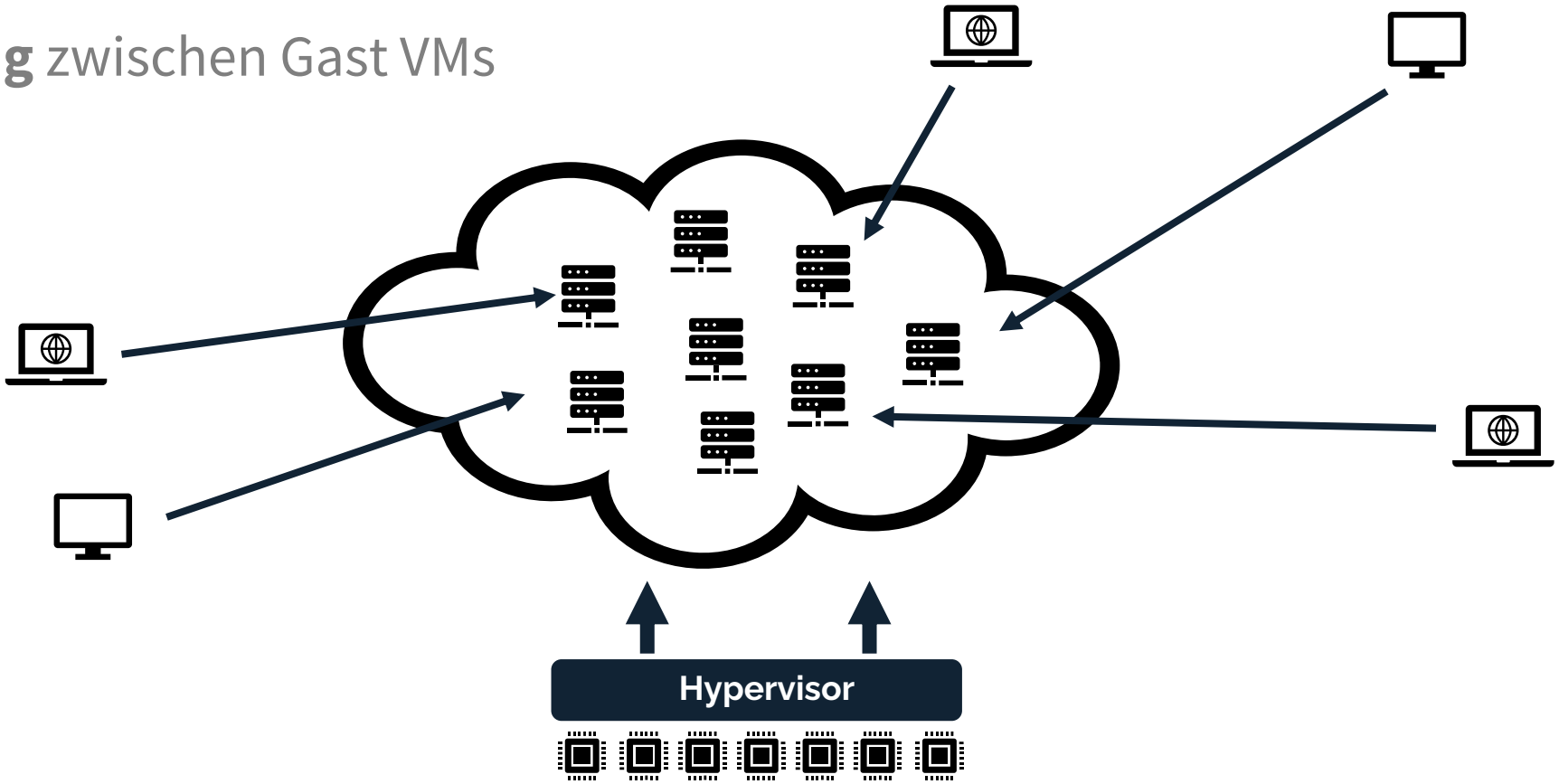
- **Speicher** kann auch **während** der **Laufzeit ausgelesen** werden, z.B. mit Mimikatz Modul sekurlsa
- Möglichkeit zur **Code Injection** in Virtuelle Maschinen
- Schwachstellen im **Hypervisor** erlauben **Zugriff** auf Daten **außerhalb** des eigenen **Gastsystem** (guest breakout)



Exkurs: Cloud Systeme

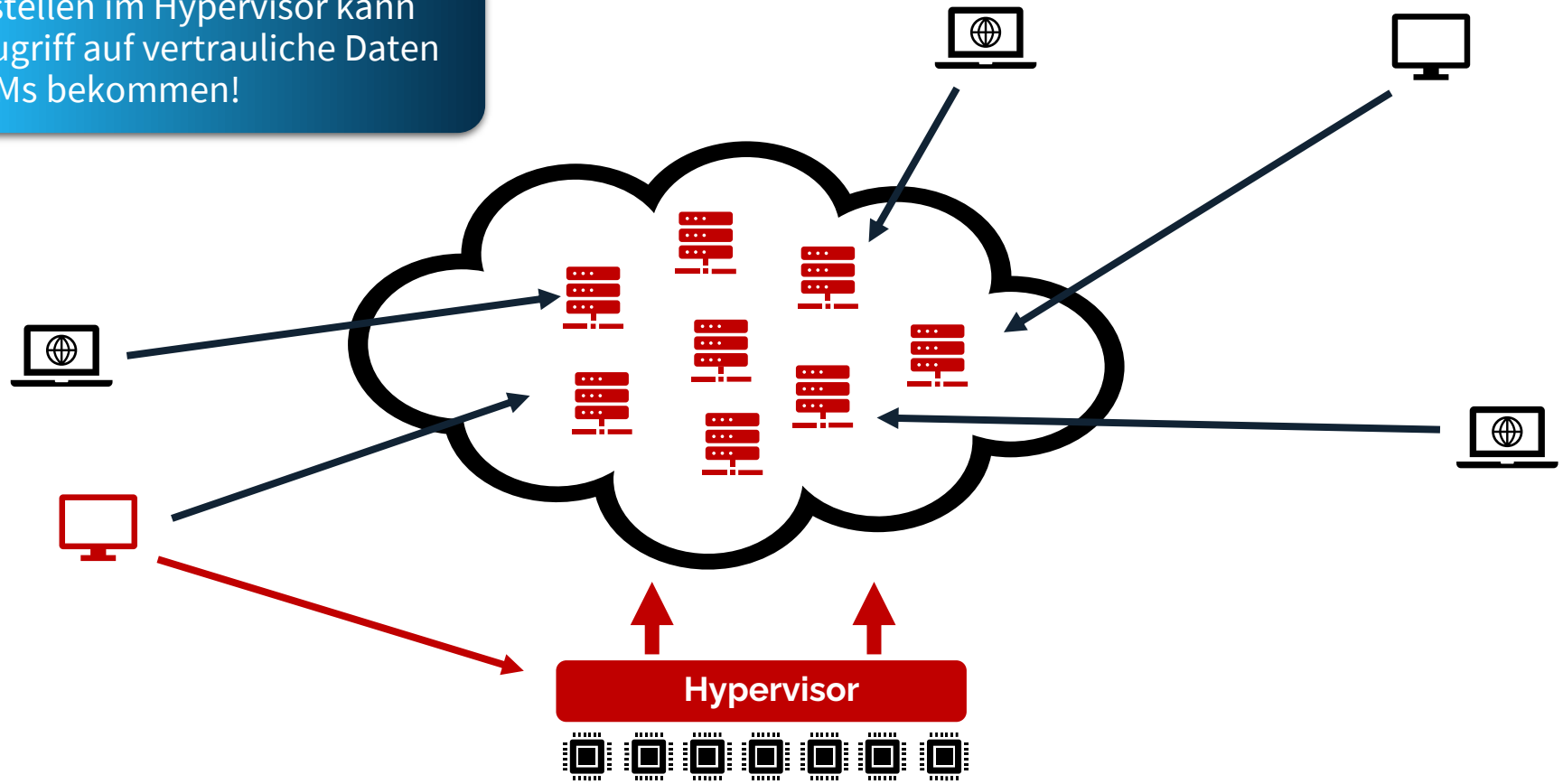
Hypervisor

- stellt **isolierte VM** zur Verfügung
- garantiert **Trennung** zwischen Gast VMs

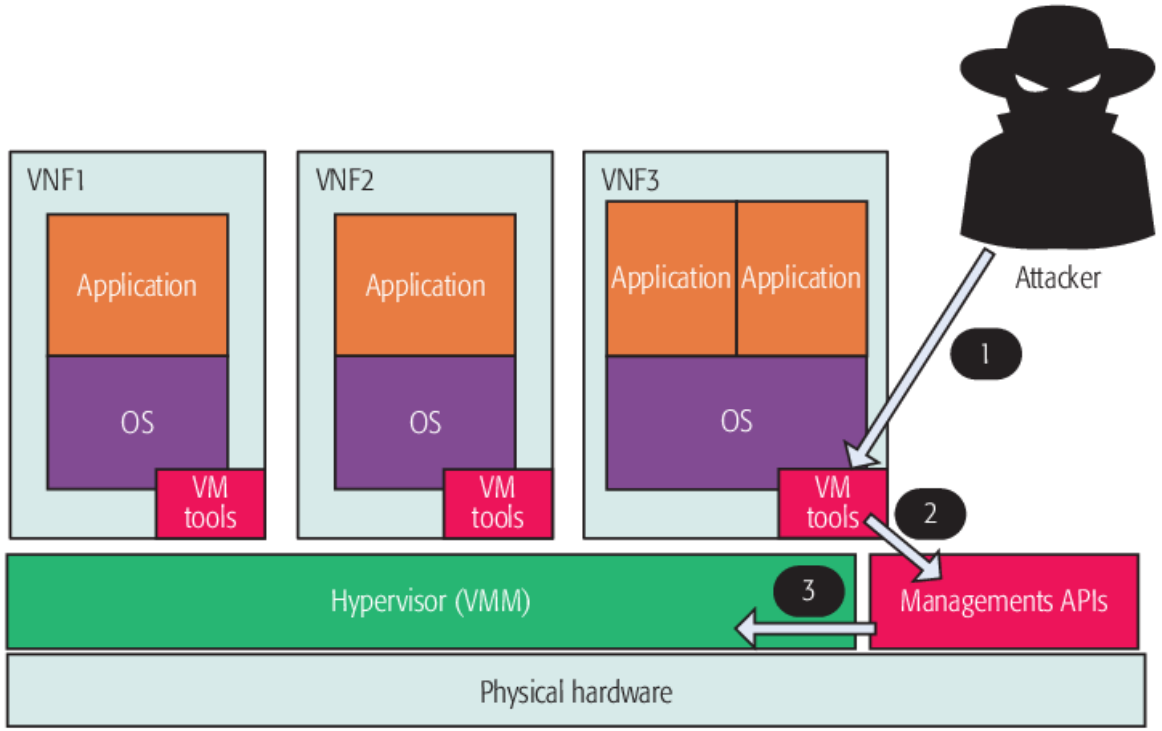


Hypervisor

Über Schwachstellen im Hypervisor kann ein Angreifer Zugriff auf vertrauliche Daten fremder Gast VMs bekommen!

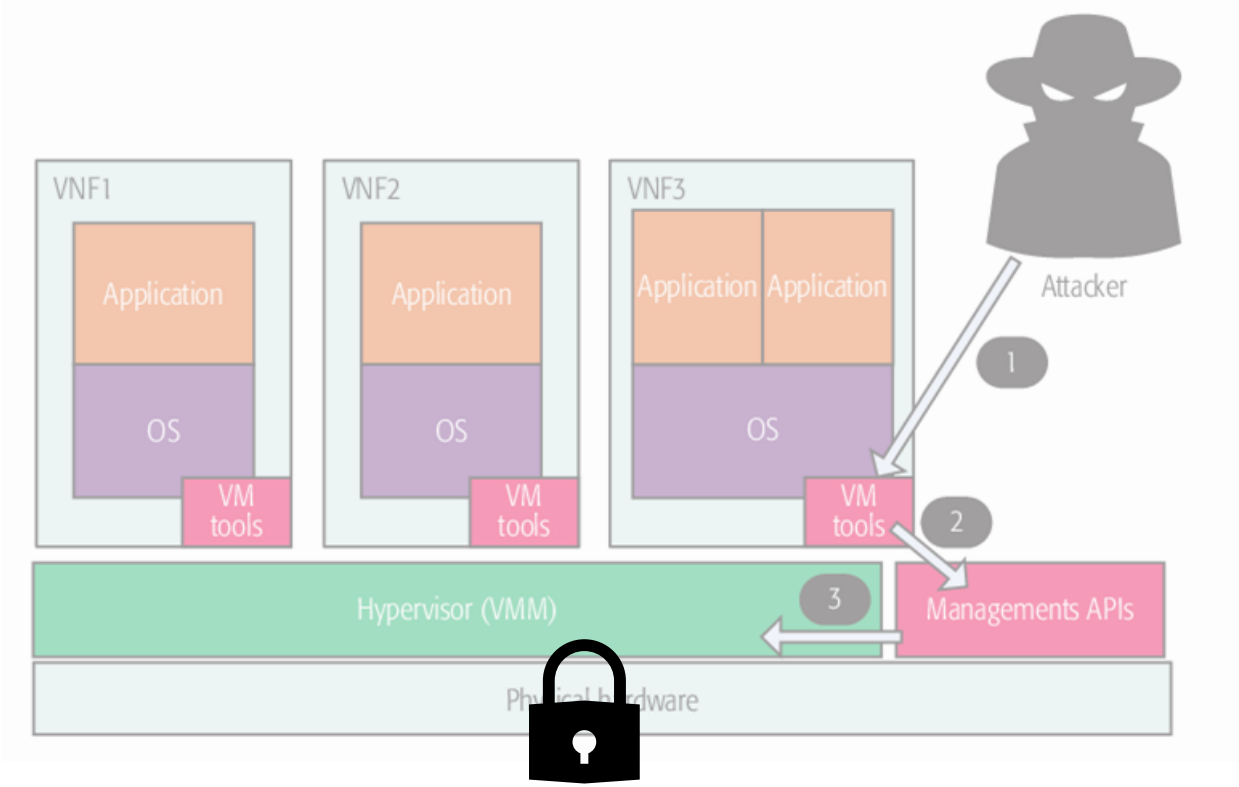


Hypervisor Exploit: CVE 2015 – 3456 Venom

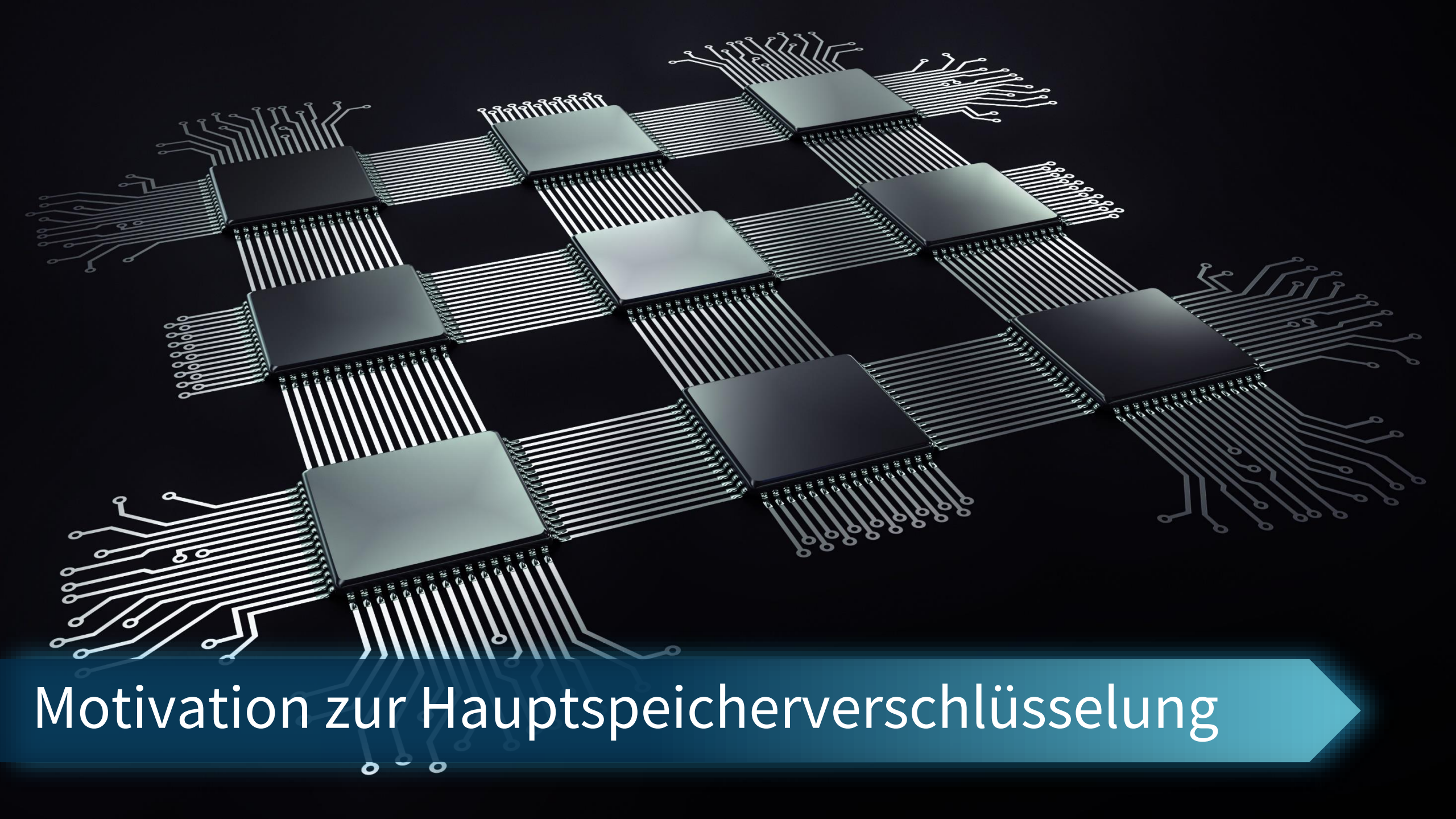


- **Buffer Overflow** Schwachstelle im QEMU Diskettentreiber
- **Mehrere Virtualiserungs-Plattformen** betroffen: KVM, Xen, VirtualBox
- **Guest Breakout → Zugriff** auf andere VMs und **Codeausführung** möglich
- Funktioniert mit **Standard Konfiguration**

Hypervisor Exploits



- Sind eine **Motivation** zur **Hauptspeicher Verschlüsselung:**
  - Schützt Daten vor **unerlaubten Zugriff** auch im Fall das der Hypervisor von einem Angreifer übernommen wird
  - Wichtig für Cloud Provider, die die **Integrität** & **Vertraulichkeit** von Kundendaten jederzeit **gewährleisten** müssen



Motivation zur Hauptspeicherverschlüsselung

Motivation zur Hauptspeicherverschlüsselung

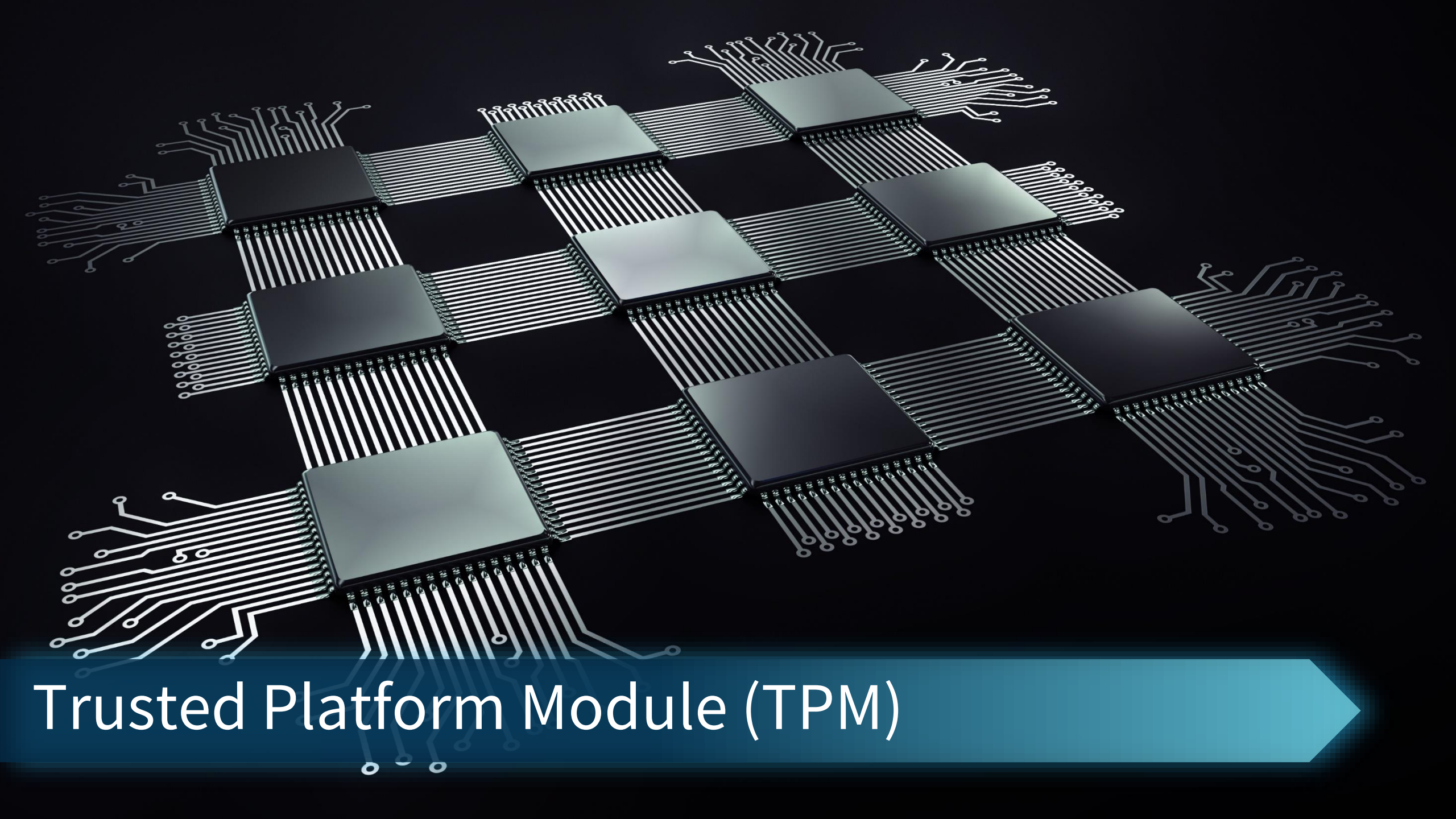
- Anwendungsentwicklern ermöglichen, vertrauliche Daten vor **unbefugtem Zugriff** oder Änderungen durch unerwünschte Software zu schützen
- Anwendungen erlauben, die **Vertraulichkeit** und **Integrität** vertraulicher Codes und Daten zu wahren
- Benutzern ermöglichen, die **Kontrolle** über die Plattformen zu behalten und Anwendungen und Dienste nach Belieben zu installieren und zu deinstallieren
- Der Plattform ermöglichen, die **Vertrauenswürdigkeit** einer Anwendung zu messen und Zertifikate zu erstellen, die im Prozessor verwurzelt sind

Motivation zur Hauptspeicherverschlüsselung

- **Skalierung** der Leistung vertrauenswürdiger Anwendungen mit den Funktionen des zugrunde liegenden Prozessors

- Anwendungen erlauben, sichere Code- und Datenbereiche zu definieren, die die **Vertraulichkeit** gewährleisten, selbst wenn ein Angreifer die physische Kontrolle über die Plattform hat und direkte Angriffe auf den Speicher ausführen kann

- **Cloud Computing:** Schutz von Daten in VMs



Trusted Platform Module (TPM)

Trusted Platform Module

- Das Trusted Platform Module (TPM) ist eine Art aufgelötete SmartCard als Vertrauensanker

- Bietet im Wesentlichen drei Grundoperationen:
  - **Binding** erlaubt es, Code oder Daten so zu verschlüsseln, dass sie sich nur auf demselben Gerät wieder entschlüsseln lassen
  - **Sealing** bezieht zusätzlich noch die aktuelle Plattform-Konfiguration mit ein (BIOS-Einstellungen, das laufende Betriebssystem, aktuell laufende Software) - Nur wenn diese Faktoren im selben Zustand sind wie bei der Versiegelung, lassen sich Daten wieder entsiegeln.

Trusted Platform Module

- Das Trusted Platform Module (TPM) ist eine Art aufgelötete SmartCard als Vertrauensanker

- Bietet im Wesentlichen drei Grundoperationen:
  - **Remote Attestation** erlaubt es einer entfernten Partei (Software-Hersteller) die aktuelle Plattformkonfiguration zu überprüfen
    - Der Hersteller liefert Code oder Daten nur dann aus oder gibt sie frei, wenn sich das System in einem von ihm gewollten Zustand befindet
    - Lässt sich auch verwenden, um nur solchen Notebooks den VPN-Zugriff auf das interne Netzwerk einer Firma zu gewähren, auf denen bestimmte (Virenschutz-)Software installiert ist

TPM - Einsatzbereiche

- Schutz des Bootvorgangs vor Manipulationen (Measured Launch)
  - TPM legt Hash-Werte der Firmware und des Bootloaders in seinen geschützten Speicherbereichen ab (Platform Configuration Registers)

- Windows-Festplattenverschlüsselung BitLocker
  - Sealing des Schlüssels
  - Daten auf der Platte oder SSD lassen sich nicht mehr entschlüsseln, wenn man den Datenträger vom Mainboard trennt

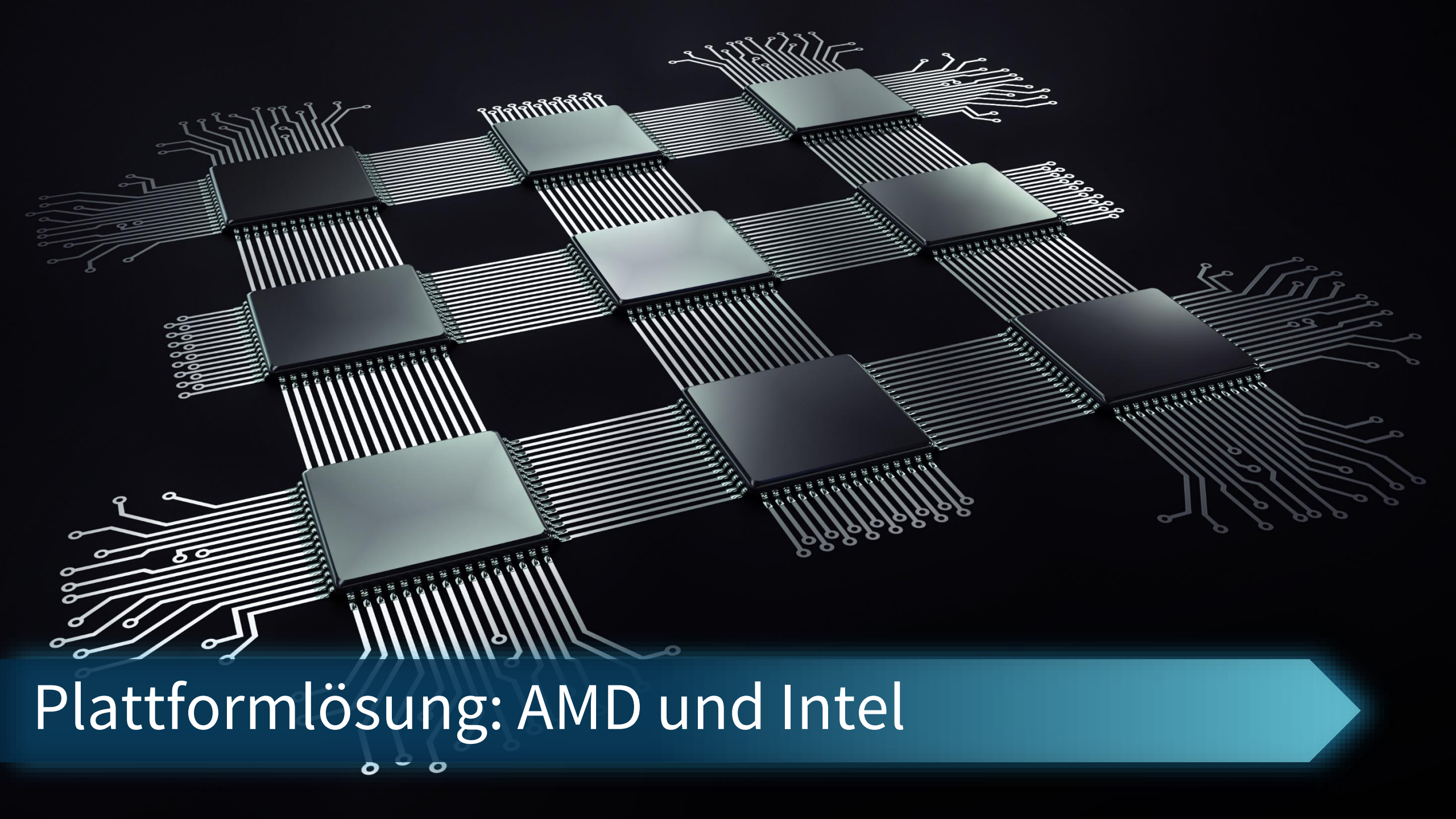
TPM - Nachteile

- Sämtliche Operationen bauen auf einem einzigen, unveränderlichen Vertrauensanker auf

- Für eine Messung der Vertrauenswürdigkeit per TPM muss die gesamte Konfiguration eines Systems betrachte werden:
  - über Measured Launch hinaus
  - Hashes für sämtliche Komponenten des Betriebssystems
  - Hashes für alle Treiber und jede geladene Software

TPM - Nachteile

- Nicht für DRM noch für das Szenario eines nicht vertrauenswürdigen Cloud-Providers geeignet
- Hersteller oder Kunde müsste den ganzen Softwarestack kontrollieren
- Kann nicht gegen physische Angriffe auf den Hauptspeicher schützen, da laufende Software oder gerade verwendete Daten im Klartext im Speicher liegen



Plattformlösung: AMD und Intel

Hauptspeicherverschlüsselung: AMD und Intel

- **RAM** und **NVRAM** wird verschlüsselt und dadurch dem **Zugriff** anderer laufender Prozesse **entzogen**

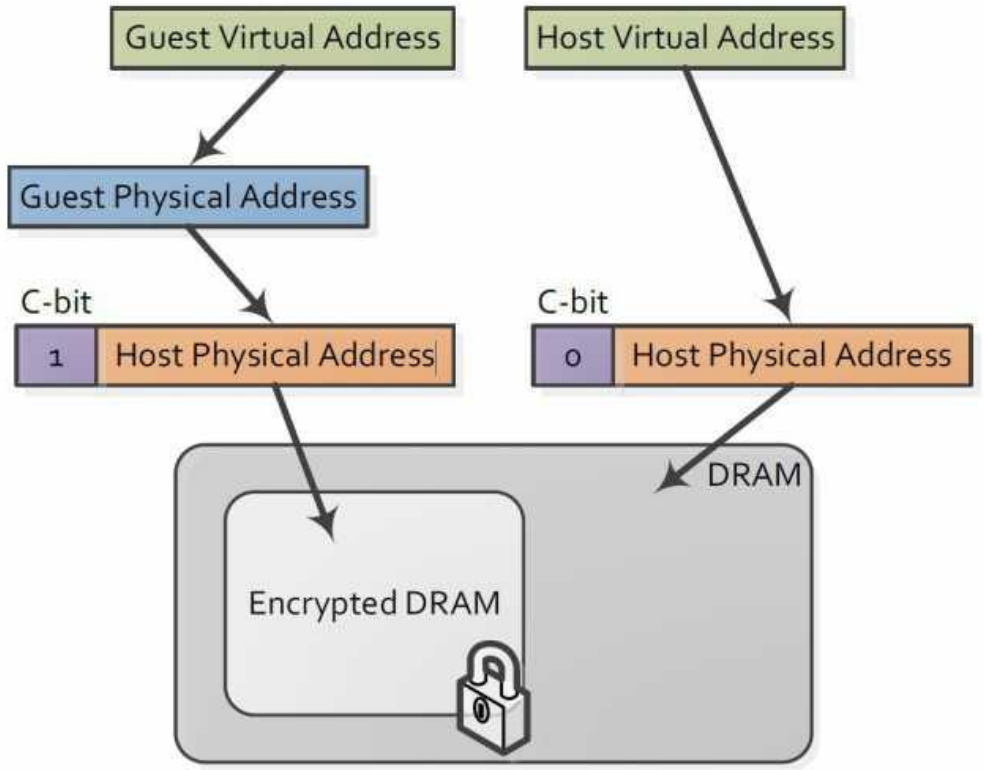
- **Intel:**
  - Software Guard Extensions (SGX):
    - richtet verschlüsselte Enklaven im RAM ein
    - Remote Attestation gibt einer entfernten Partei (Cloud-Kunde) Garantien über laufende Software
  - Total Memory Encryption (TME): verschlüsselt das gesamte RAM transparent
  - Multi-Key Total Memory Encryption (MKTME): verschlüsselt RAM-Bereiche, die ein Hypervisor festlegt

Hauptspeicherverschlüsselung: AMD und Intel

- **RAM** und **NVRAM** wird verschlüsselt und dadurch dem **Zugriff** anderer laufender Prozesse **entzogen**

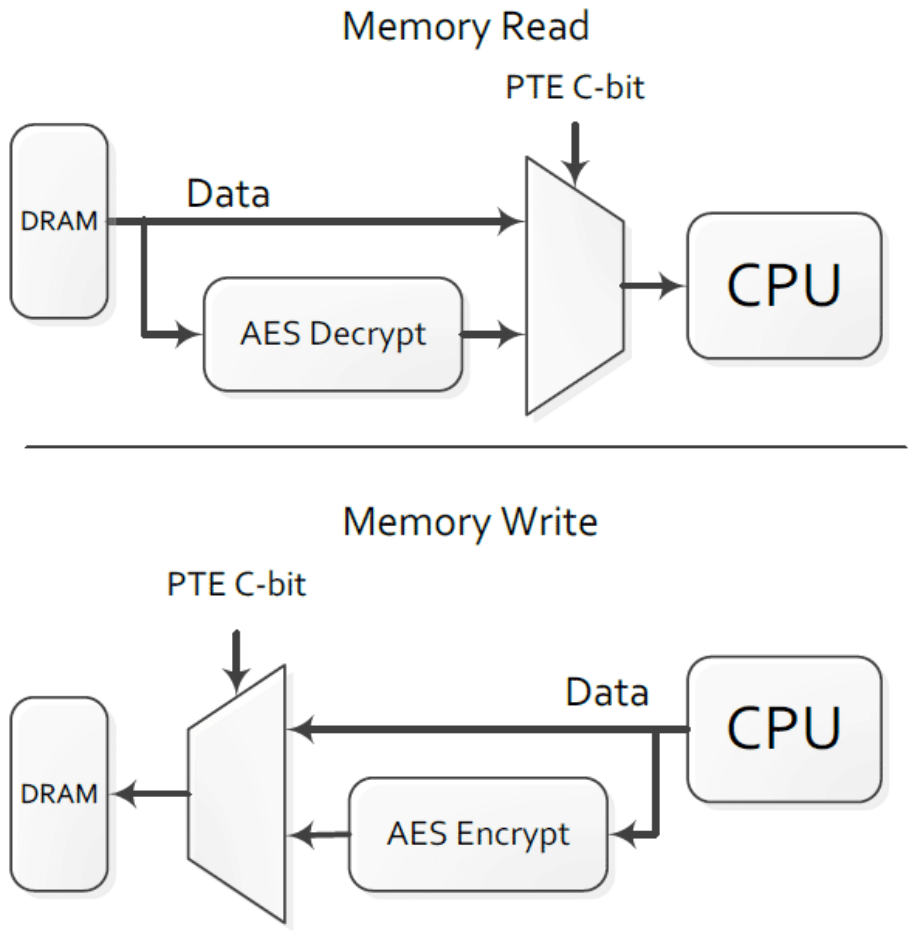
- **AMD:**
  - ARM TrustZone: trennt ein System in eine sichere und eine unsichere "Welt"
  - Secure Memory Encryption (SME): verschlüsselt das gesamte RAM transparent
  - Secure Encrypted Virtualization (SEV): verschlüsselt RAM-Bereiche, die ein Hypervisor festlegt

AMD: Secure Memory Encryption



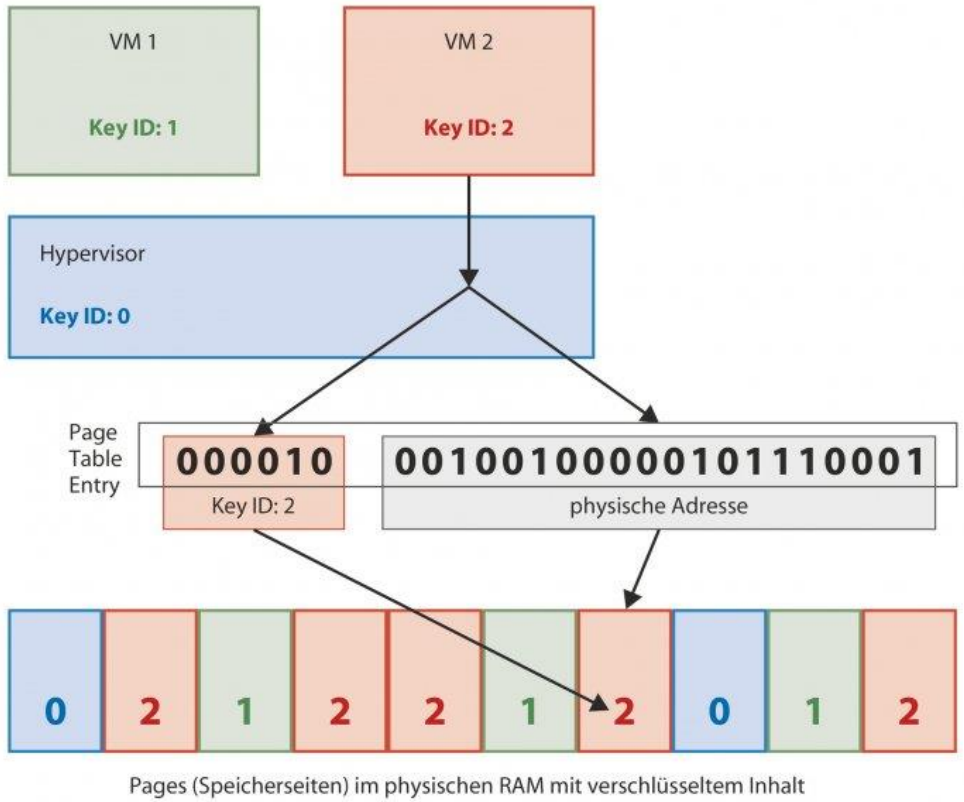
- (T)SME schützt vor Angriffen, die **physischen Zugriff** auf die Hardware des Systems benötigen:
  - Cold-Boot-Attacken
  - Sniffing-Hardware am Speicherbus
  - Diebstahl nichtflüchtiger Speichermodule

AMD: Secure Memory Encryption



- nur **bestimmte RAM-Adressbereiche** verschlüsselt
- **C-Bit** (C für enCryption) im zugehörigen Page-Table-Eintrag (PTE).
- Speicher der VMs kann auch mit **Sniffing-Software** nicht mehr im Klartext ausgelesen werden
- RAM jeder VM wird im **anderem Schlüssel** verschlüsselt

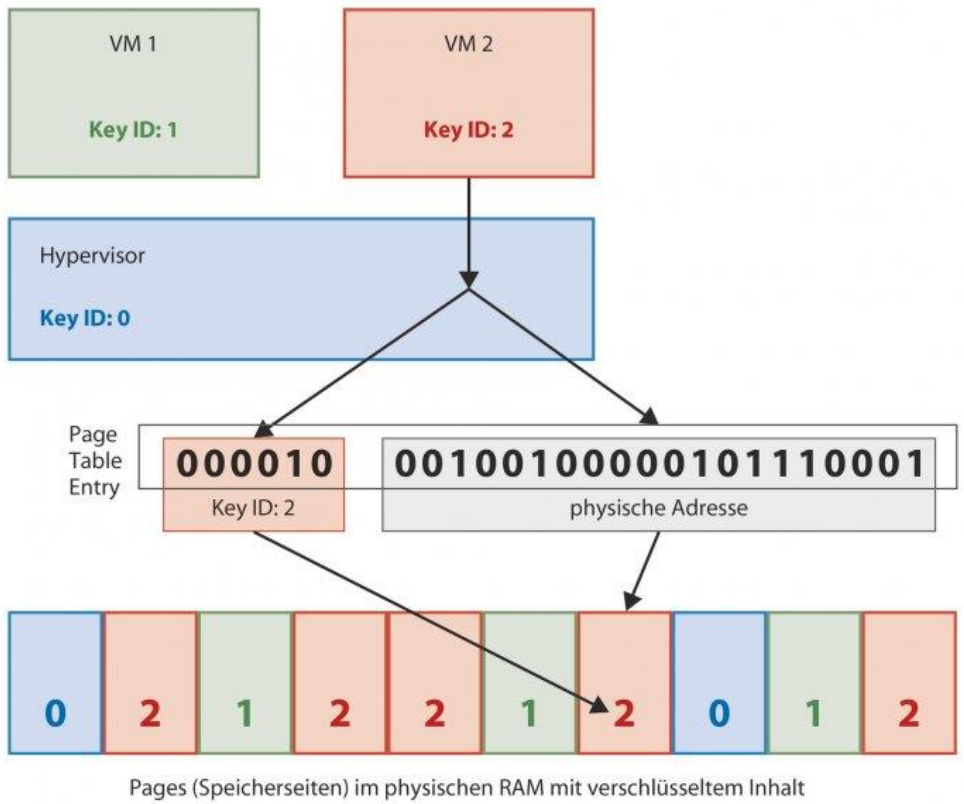
AMD: SEV



Pages (Speicherseiten) im physischen RAM mit verschlüsseltem Inhalt

- Für Serverprozessor
- Zielt auf nicht vertrauenswürdige Cloud-Provider, die sensible Kundendaten aus dem Server-RAM auslesen könnten
- Kann beliebig grosse Speicherbereiche einrichten
- Der Hypervisor setzt dazu pro VM einen Schlüssel (ASID), mit dem die CPU alle RAM-Pages verschlüsselt, die zur selben VM gehören.
- Die Schlüssel verwaltet der AMD Secure Processor (PSP) auf Basis von ARM TrustZone

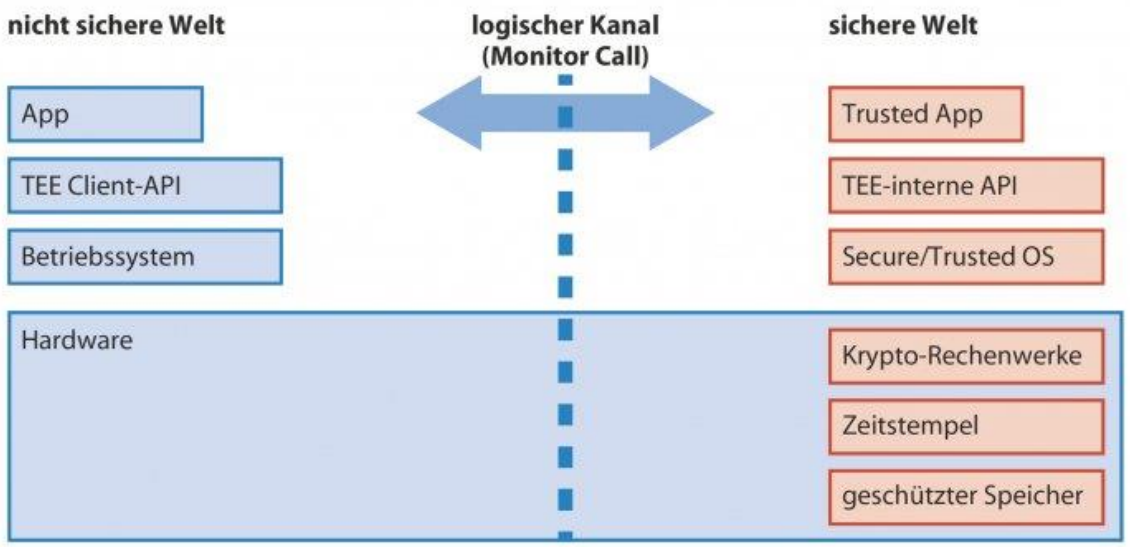
AMD: SEV



Pages (Speicherseiten) im physischen RAM mit verschlüsseltem Inhalt

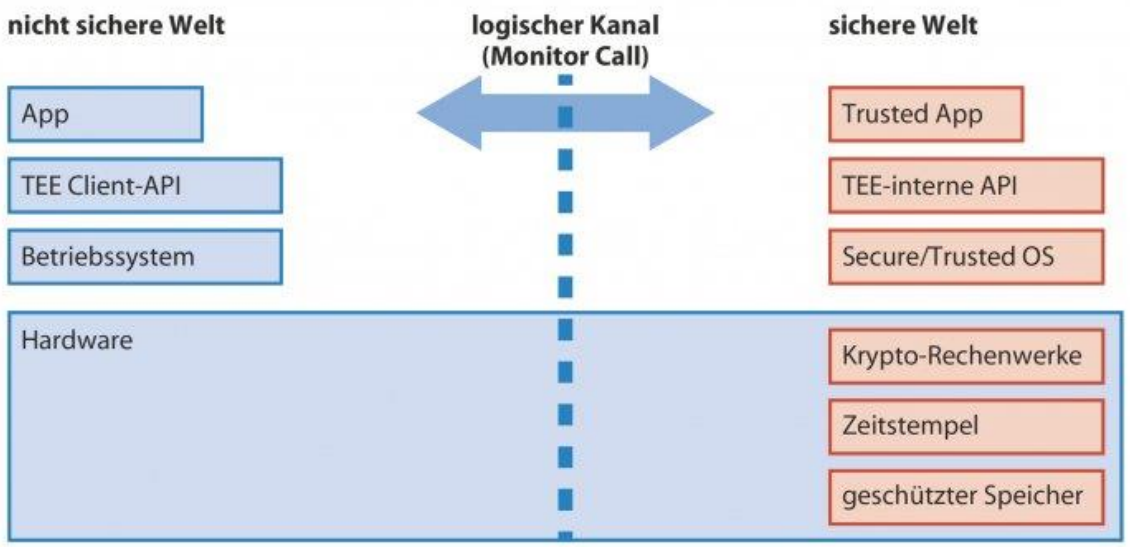
- Obwohl der Hypervisor viele VM-Funktionen steuert, darunter Geräteemulation und Scheduling, muss man ihm nicht mehr vollständig vertrauen
- Würde ein manipulierter Hypervisor falsche Schlüssel setzen, obwohl die VM vorher mit einem anderen Schlüssel gestartet wurde, würden bei der Entschlüsselung fehlerhafte Daten gelesen – und die Daten in der VM blieben geschützt
- Um einer entfernten Partei zu garantieren, dass die VM beim ersten Start nicht fehlerhaft zusammengesetzt wurde, verlässt sich AMD auf einen Attestierungsmechanismus (hier muss man AMD vertrauen)

AMD: TrustZone



- ARM TrustZone teilt das System in zwei Welten, die "Secure World" und die "Non-Secure World„
- Strikt voneinander getrennt, selbst höher privilegierte Software innerhalb der Non-Secure World einschließlich des Betriebssystems kann nicht auf Code und Daten in der Secure World zugreifen
- Verbindung beider Welten findet über einen Monitor-Call statt, der ähnlich wie ein Systemaufruf in das Betriebssystem zu verstehen ist.

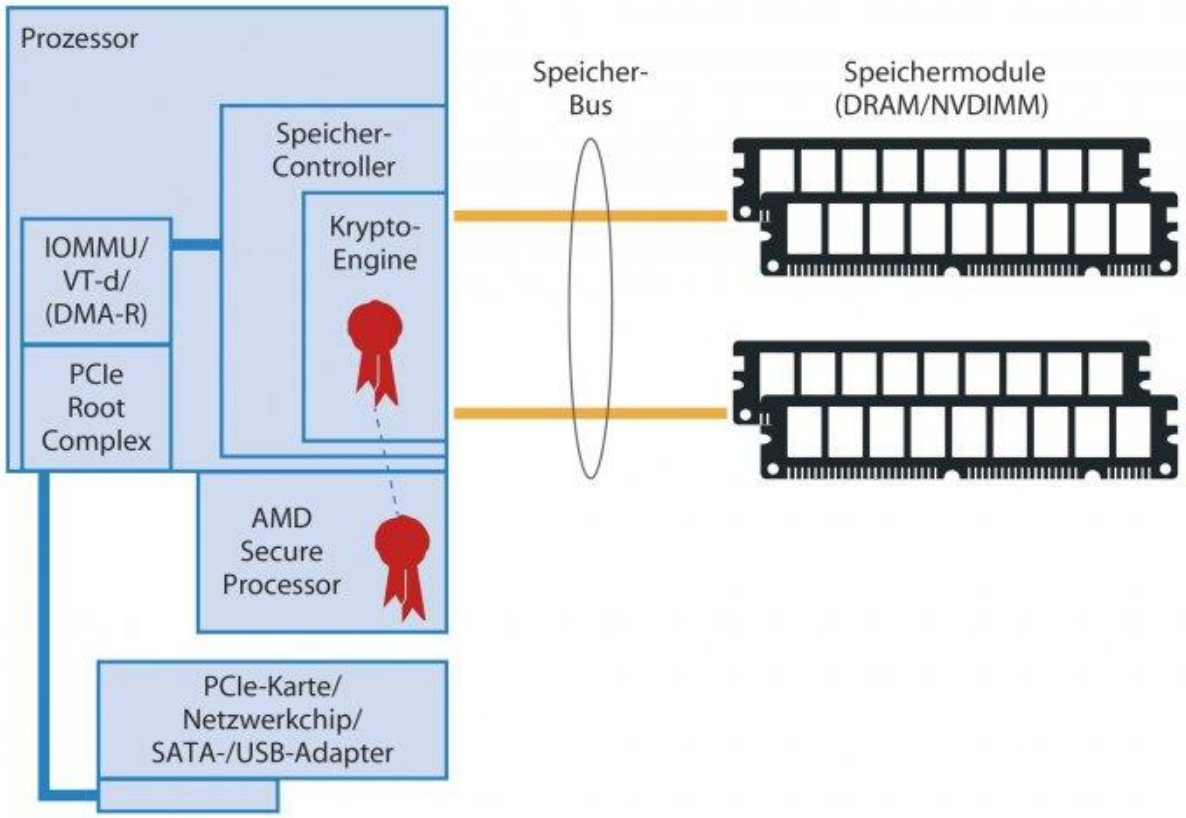
AMD: TrustZone



- Ein "Secure Bit" auf dem internen Bus des Chips sorgt dafür, dass bestimmte Baugruppen und Zusatzchips nur aus der Secure World heraus erreichbar sind

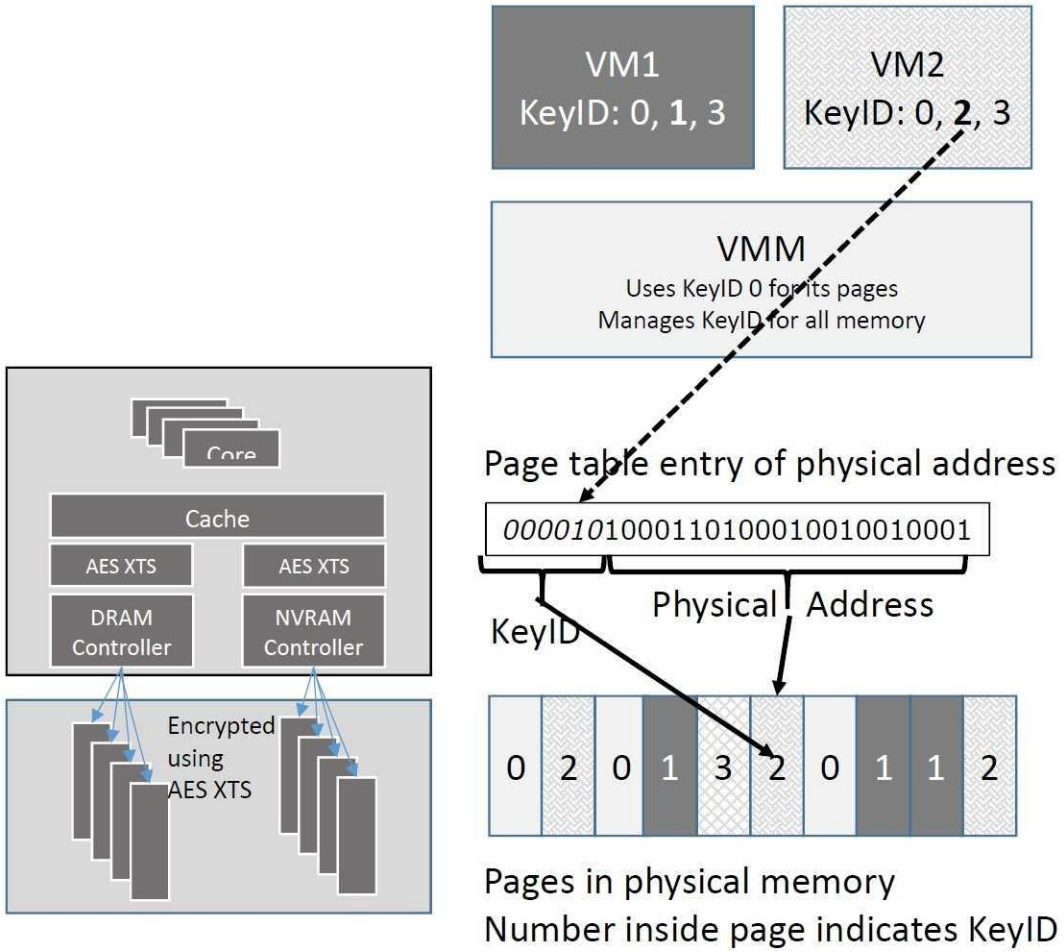
- Damit lassen sich z.B. Fingerabdruckleser sicher anbinden, um den gesamten Prozess der Authentifizierung im sicheren Bereich abzuwickeln

Intel: TME



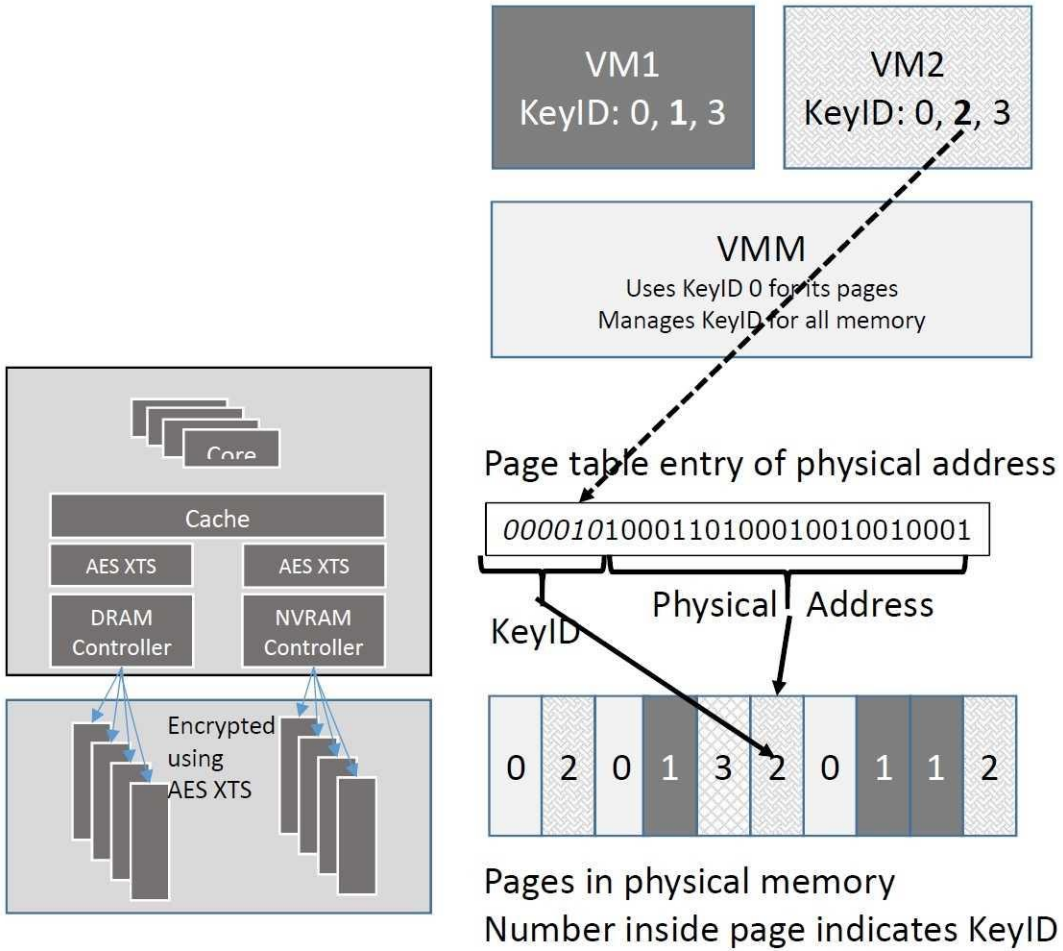
- Nutzt einen in der Regel beim Systemstart generierten 128-Bit-Schlüssel, um den gesamten Speicher zu verschlüsseln (per AES-XTS)
- Hilft nur gegen physische Angriffe, nicht gegen Software-Angriffe
- Alle Daten bleiben in Caches sowie auch alle Daten innerhalb des Prozessors im Klartext

Intel: Multi-Key Total Memory Encryption



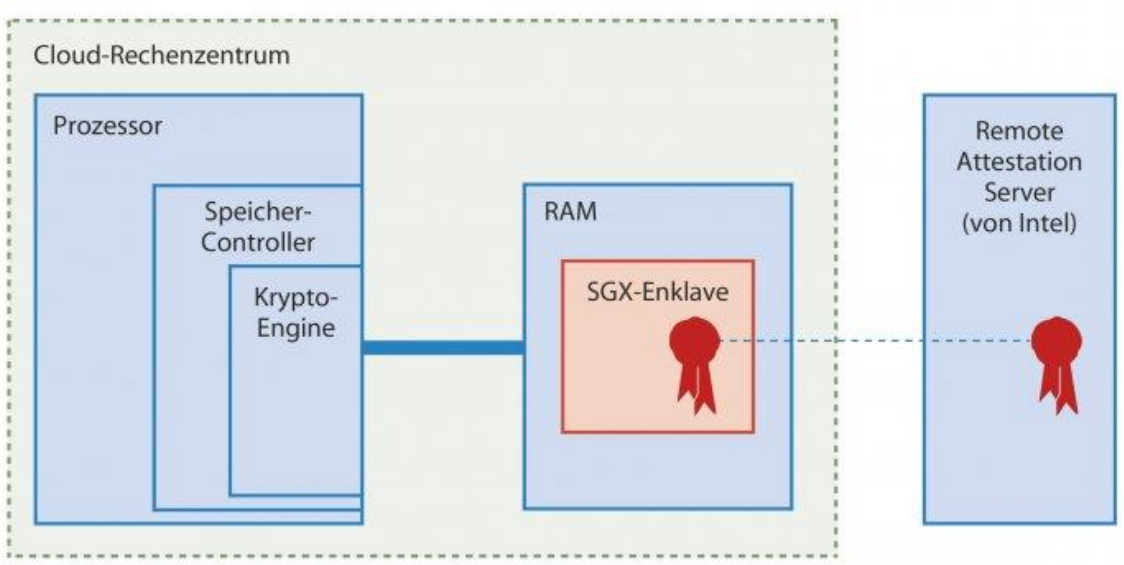
- Der im Prozessor integrierte Speicher-Controller verwaltet **128-Bit-Schlüssel**
- sämtliche Daten werden vor dem Schreiben per **AES-XTS** verschlüsselt
- Bei MKTME kann das System jeder VM einen **eigenen Schlüssel** zuweisen

Intel: Multi-Key Total Memory Encryption



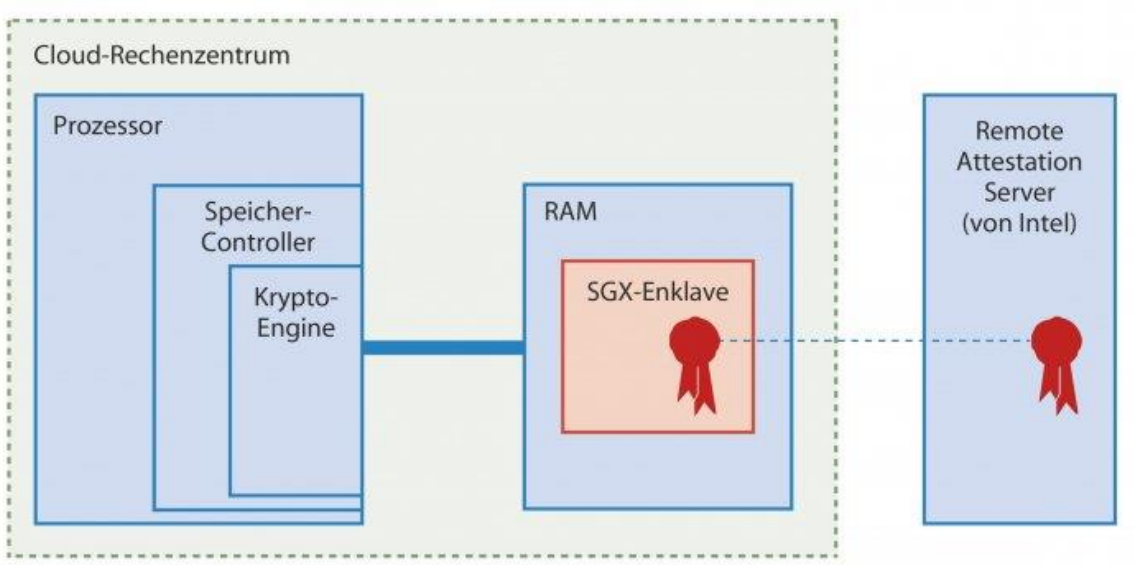
- Schlüssel werden automatisch generiert
- **UEFI** muss die Funktionen aktivieren
- **Steuerung** durch Software erfolgt über Model-Specific Registers (MSRs) des Prozessors

Intel: Software Guard Extensions



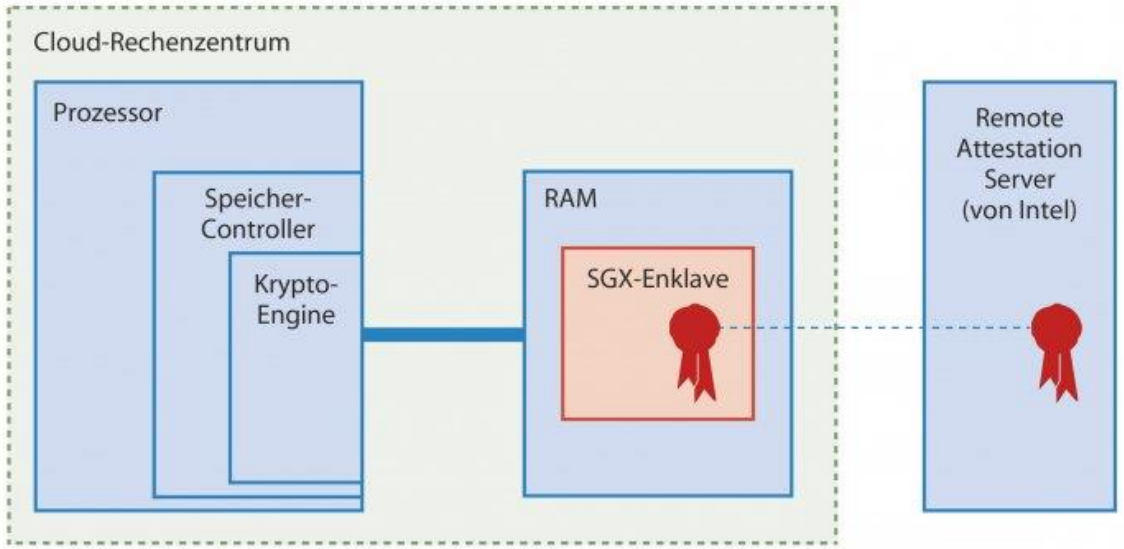
- Erlaubt es, Vertrauensanker dynamisch einzurichten.
- Es gibt nicht nur eine sichere Welt wie beim TPM und bei TrustZone, sondern viele sog. sichere Enklaven
- Diese lassen sich im normalen Adressraum eines Prozesses einrichten und sind trotzdem durch Hardware-Mechanismen vor höher privilegierter Software geschützt

Intel: Software Guard Extensions



- Code in der SGX-Enklave läuft mit hoher Performance
- Das zur Verschlüsselung verwendete Geheimnis erzeugt der Prozessor bei jedem Systemstart automatisch neu
- Es verlässt den Prozessor nie, ist also auch nicht auslesbar

Intel: Software Guard Extensions



- Jeder SGX-taugliche Prozessor enthält zwei "eingebrannte" (fused), individuelle, zufällige 128-Bit-Schlüssel
- Anhand des Root Provisioning Key, den Intel in einer Datenbank aufbewahrt, lässt sich nachweisen, dass der Prozessor tatsächlich existiert
- Vom Root Seal Key, den Intel nicht speichert, kann eine SGX-Enklave Sealing-Schlüssel ableiten, um verschlüsselte Daten ausserhalb der Enklave zu schützen

Einsatzgebiete für Hauptspeicherverschlüsselung

Digital Rights Management

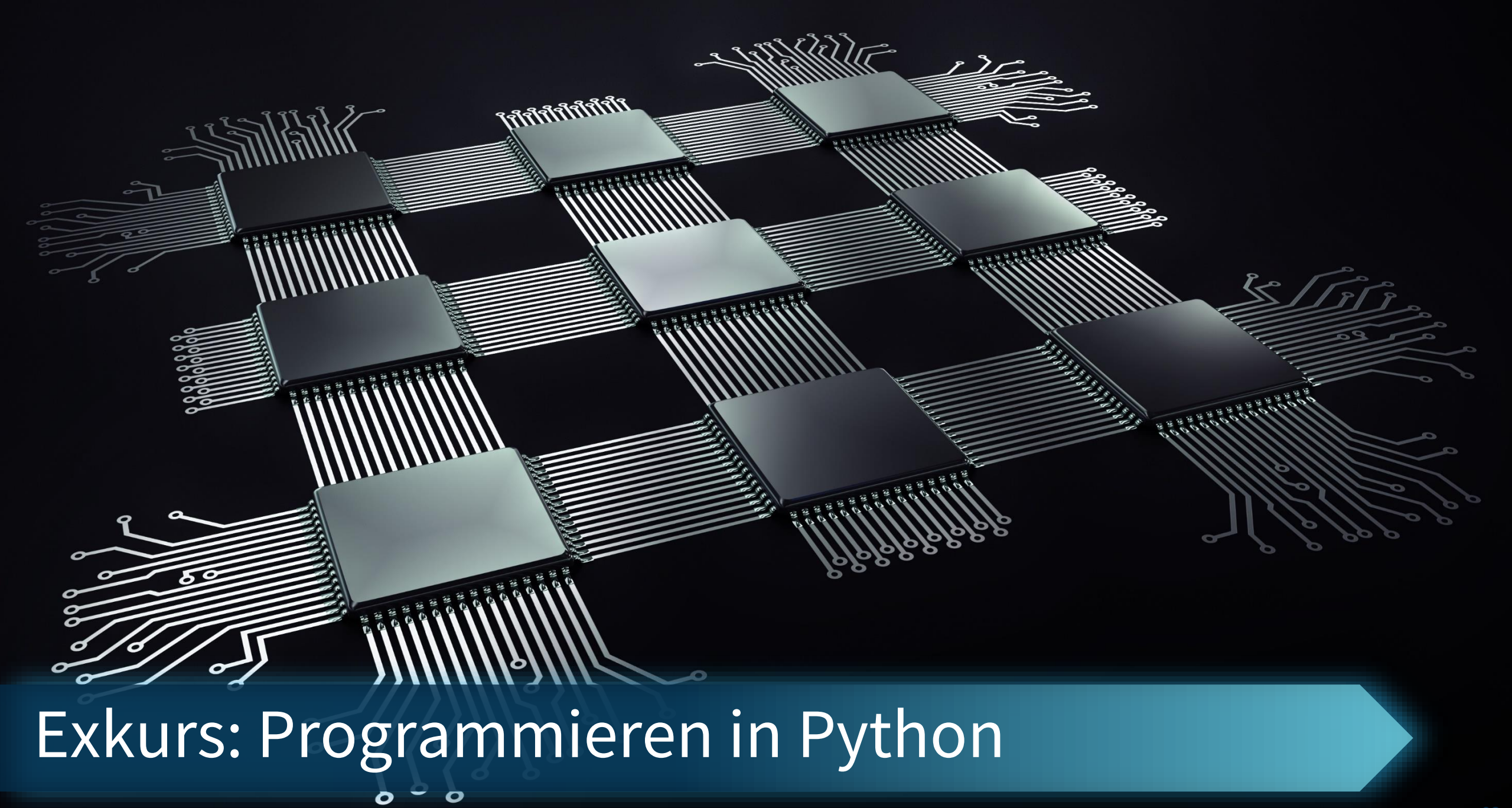
- Verbergen bestimmter Software oder Daten vor dem Benutzer

Netflix

- Nutzung von SGX beim Streaming von Ultra-HD-Filmen (kopiergeschützte Inhalte)
- Der geheime Schlüssel für das Videomaterial wird dabei nur in einer verschlüsselten SGX-Enklave verwendet, die weder das Betriebssystem noch laufende Programme auslesen können
- Der dekodierte Videodatenstrom wird dann mit anderen Methoden wie Protected Audio-Video Path (PAVP) und High Definition Content Protection (HDCP) geschützt zum Display übertragen

Cloud Anbieter

- Ziel: Vertrauen verlagern
- Bei typischen Cloud-Dienstleistern (Amazon AWS, Google Cloud, Microsoft Azure) haben die Administratoren die Möglichkeit, sämtliche Daten aus dem Server-RAM zu lesen
- Ist der Arbeitsspeicher mit Hardware-Funktionen des Prozessors verschlüsselt, kann der Cloud-Anbieter glaubhaft versichern, Nutzerdaten und Programmcode weder manipulieren noch auslesen zu können
- Kunden müssen dann „nur noch“ den CPU-Funktionen und der Schlüsselgewalt von AMD oder Intel vertrauen



Exkurs: Programmieren in Python

Variablen und Arithmetik

```
# Deklaration durch Zuweisung
a = 1
b = 2

# Rechnen
print(a+b) # output: 3

a += 4
print(a) # output: 5
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Strings

```
# Strings mit einfachen oder doppelten Anführungszeichen
a = 'Hello'
b = "World"

# Zusammensetzen
print(a + ' ' + b) # output: Hello World

# Multiplikation
print('a' * 5) # output: aaaaa
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Verzweigungen und Blöcke

```
a,b,c = 1,2,True # Zuweisung von mehreren Variablen mit Komma

if a>b and c:    # Logik-Operationen mit and, or und not
    print('a is bigger and c is true')
elif a==b:       # Equality mit == und Inequality mit !=
    print('a and b are equal')
else:
    print('b is bigger')
    print(b)
print('this is always printed')

# Code-Blöcke durch Einrücken (Leerzeichen oder Tabs)
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Funktionen

```
def myfunc(a):
    return a*a

x = myfunc(5)
print(x) # output: 25
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Komplexe Variablen: Listen

```
# Arrays mit variabler Länge
mylist = [5,'x',"potato"]

# Index beginnt bei 0
print(mylist[2]) # output: potato

# Element ersetzen
mylist[1] = 'y'

# Element hinzufügen
mylist.append('xyz')
mylist += ['abc'] # Zusammensetzen von zwei Listen
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Komplexe Variablen: Dictionaries

```
# Assoziative Arrays (Key-Value-Paare)
mydict = {'first': 5, 17: 'x', 'c': "potato"}

# Zugriff auf Element über Key
print(mydict['c']) # output: potato

# Element ersetzen oder hinzufügen
mydict['first'] = 420
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Komplexe Variablen und Schleifen

```
# Iteration über Lists
mylist = [5,'x',"potato"]
for element in mylist:
    print(element)


# Iteration über Dictionaries
mydict = {'first': 5, 17: 'x', 'c': "potato"}
for key in mydict:
    print(key)
    print(mydict[key])
```

```
5
x
potato


first
5
17
x
c
potato
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Logik-Operationen auf Dictionaries

```
# Existenz eines Keys
mydict = {'first': 5, 17: 'x', 'c': "potato"}
if 'c' in mydict:
    print(mydict['c'])

# Existenz eines Values
if 'potato' in mydict.values():
    print("potato")

# Vergleich zweier Dictionaries: Gleiche Keys, gleiche Values
if mydict == {'first': 5, 17: 'x', 'c': "potato"}:
    print("everything is the same")
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Dictionaries: Löschen von Werten

```
mydict = {'first': 5, 17: 'x', 'c': "potato"}

# Löschen eines Eintrags, Key existiert garantiert
del mydict['c']

# Löschen eines Eintrags falls er existiert
mydict.pop('c', None)
```

Verschachtelte Dictionaries

```
mydict = {'a': {'type': 'f', 'uid': 1000} }

# Zugriff auf verschachtelte Elemente mit mehreren []
if 'a' in mydict and mydict['a']['type'] == 'f':
    print("type of mydict['a'] is 'f'")

# Hinzufügen eines Unter-Dictionaries
if not 'c' in mydict:
    mydict['c'] = {'type': 'f', 'uid': 1000}
else:
    print("mydict['c'] already exists")
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Format-Strings

```
size,type,uid = 9000,'f',1000

# Formatstrings mit %
str = "size of a is %d" % size
print(str) # output: size of a is 9000

# Mehrere Werte mit ()
print("type is %s and uid is %d" % (type, uid) )
# output: type is f and uid is 1000

# %s = String, %d = Integer (Ganzzahl)
```

Quelle: Hochschule Luzern – Computer Science and Information Technology

Call By Object Reference

```
def f1(d): # Change Variable in Function
    d = {'a': 1}

def f2(d): # Change referenced Object
    d['a'] = 1

mydict = {'a': 0, 'b': 1}
print(mydict)
f1(mydict)
print(mydict)
f2(mydict)
print(mydict)
```

```
{'a': 0, 'b': 1}
{'a': 0, 'b': 1}
{'a': 1, 'b': 1}
```

Quelle: Hochschule Luzern – Computer Science and Information Technology