

## D Autorisierung und Zugriffskontrolle

### D.1 Autorisierung von Benutzern

- Sonderfall allgemeiner Schutzmechanismen
- Beispiel für einfache Realisierung: UNIX-Dateizugriffe
  - ◆ Benutzer ist UID zugeordnet
    - ▶ Korrektheit wird durch Authentifizierung sichergestellt
  - ◆ System kennt einzelne Benutzer und Gruppen
  - ◆ Für jede Datei kann die Berechtigung für Eigentümer, Gruppe, Rest festgelegt werden
    - ▶ rwx
- Zugriffskontrolle muss sicherstellen, dass jeder Zugriffsversuch kontrolliert wird und nur autorisierte Zugriffe möglich sind

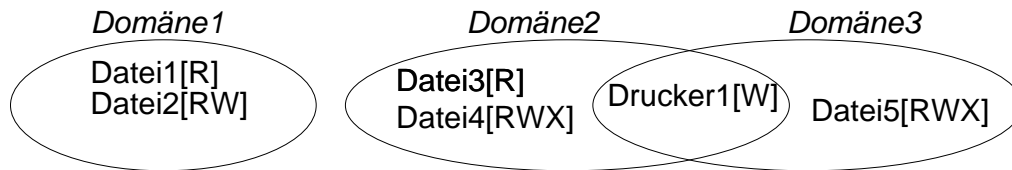
### D.2 Speicherschutz

- Schutzräume für Anwendungen (Prozesse) und Betriebssystem (Systemkern)
- Schutz zwischen Systemkern und Anwendungen ist Basis für alle weiteren Zugriffskontrollkonzepte
  - ▶ Verwaltung in Datenstrukturen des Systemkerns
  - ▶ logische/virtuelle Adressräume gewährleisten Abgrenzung Programmadressen → MMU → Maschinenadressen
  - ▶ MMU-Datenstrukturen durch Systemkern-Adressraum geschützt
  - ▶ Kontrollierte Übergänge zwischen Betriebsmodi (*user / kernel mode*) → Trap
- Alternative: Schutz durch Programmiersprache (Typ-Konzept) und Laufzeitsystem
  - ▶ Beispiel: Java + Verifier + Virtuelle Maschine

## D.3 Objektschutz

### 1 Schutzdomänen

- Domäne = Menge von (Objekt, Rechte)-Tupeln
  - ◆ Beispiele für Domänen
    - ein Benutzer
    - eine Gruppe
    - eine Rolle
- Recht = Menge von Operationen, die auf dem Objekt ausgeführt werden dürfen



### 1 Schutzdomänen (2)

- Ein Prozess ist zu jedem Zeitpunkt einer Schutzdomäne zugeordnet
- Wechsel der Schutzdomäne (am Beispiel UNIX)
  - exec einer Datei mit s-bit
  - setuid-Systemaufruf
- Verwaltung der Schutzdomänen im System
  - Schutzmatrix

Objekt:	Datei1	Datei2	Datei3	Datei4	Datei5	Drucker1
Domäne:1	R	RW				
2			R	RWX		W
3					RWX	W

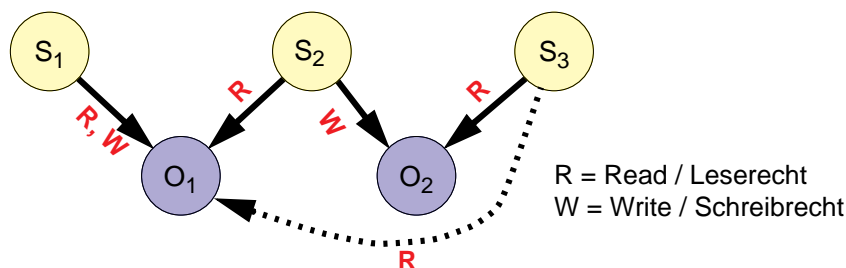
- Schutzmatrix in der Praxis nicht handhabbar
  - Information aus Spalten: Zugriffskrolllisten
  - Information aus Zeilen: Capabilities

## 1 Zugriffskontrolllisten (ACLs)

- Zugriffskontrolllisten an den Objekten (Dateien)
  - ◆ jedes Objekt hält eine Liste der Berechtigungen
    - welche Domänen haben welche Rechte
    - z. B. Unix-Datei: Inode enthält UID, GID und Rechte-Bits (primitiv)
- Zugriffskontrolllisten an den Subjekten (Prozesse)
  - ◆ jedes Subjekt hält eine Liste von Objekten und den Berechtigungen, die das Subjekt für das Objekt hat
  - ◆ ähnlich → Capabilities
- sichere Verwaltung
  - ◆ zentral: in Datenstrukturen des Systemkerns
  - ◆ dezentral
    - Objektverwaltung durch Serverprozesse
    - Aufteilung von Berechtigungs- und Zugriffskontrolle
    - Verwendung von kryptographischen Verfahren

## 2 Zugriffsausweise (Capabilities)

- Ein Benutzer (Subjekt) erhält eine Referenz auf ein Objekt
  - ◆ die Referenz enthält alle Rechte, die das Subjekt an dem Objekt besitzt
  - ◆ bei der Nutzung der Capability (Zugriff auf das Objekt) werden die Rechte überprüft



Subjekte und Objekte; Weitergabe einer Capability (O<sub>1</sub> von S<sub>2</sub> nach S<sub>3</sub>)

## 2 Zugriffsausweise (Capabilities) (2)

### ★ Vorteile

- ◆ keine Speicherung von Rechten beim Objekt oder Subjekt nötig; Capability enthält Zugriffsrechte
- ◆ leichte Vergabe von individuellen Rechten
- ◆ einfache Weitergabe von Zugriffsrechten möglich

### ▲ Nachteile

- ◆ Weitergabe nicht kontrollierbar
- ◆ Rückruf von Zugriffsrechten nicht möglich
- ◆ Capability muss vor Fälschung und Verfälschung geschützt werden
  - durch kryptographische Mittel (Capability wird signiert und wird bei Manipulation automatisch zerstört)
  - durch Speicherschutz (Capability wird im Systemkern gehalten und Anwendung erhält nur einen Index darauf)

## 3 Capabilities und ACLs am Beispiel UNIX

### ■ Systemaufruf open

- ◆ überprüft Zugriffsrechte des Prozesses (ACL)
- ◆ erzeugt *file handle* Struktur im Systemkern
  - Zugriffsmodus (read/write) wie bei open angegeben
  - Verweis auf Inode-Struktur
- ◆ Verweis auf *file handle* wird in *open file table* im Prozesskontrollblock eingetragen
- ◆ open liefert Filedeskriptor zurück = Index in *open file table*

### ■ Filedeskriptor + file handle = Capability

- ◆ nicht manipulierbar
  - file handle und open file table liegen im Systemkern
  - nur Verweise auf erfolgreich geöffnete Dateien werden eingetragen
- ◆ Rechte (bei open angegebener mode) werden in file handle gehalten

### 3 Capabilities und ACLs am Beispiel UNIX (2)

- **Kontrollstrukturen**  
(früher in SystemV, heute komplexer aber im Prinzip analog aufgebaut)

user file descriptor table

0	
1	
2	
3	
4	
5	
6	

file table

⋮
count mode offset 1 read 0
⋮
count mode offset 1 write 0
⋮
count mode offset 1 rd-wrt 0
⋮

inode table

⋮
ref-count: 2 (/etc/passwd)
⋮
ref-count: 1 (datei1)
⋮

- **Aufruf:**

```
fd1 = open("/etc/passwd", O_RDONLY);
fd2 = open("datei1", O_WRONLY);
fd3 = open("/etc/passwd", O_RDWR);

-> fd1 == 3, fd2 == 4, fd3 == 5
```

SYSSEC

### 3 Capabilities und ACLs am Beispiel UNIX (3)

- Capabilities können weitergegeben werden
  - im Rahmen von fork — offene Datei wird vererbt

```
① fd1 = open("datei", O_RDONLY);
② fork( );
```

user file descriptor table parent proc.

0	
1	
2	
3	
4	

user file descriptor table child proc.

0	
1	
2	
3	

file table

⋮
count mode offset 2 read ?
⋮
count mode offset 2 read 0
⋮

inode table

⋮
ref-count: 1 (datei1)
⋮

- durch Versenden über socketpairs

- **Empfangender Prozess benötigt dafür keine Zugriffsrechte (ACL)!**

SYSSEC

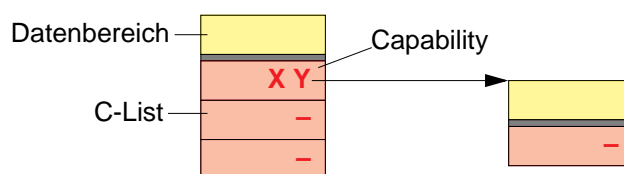
## D.4 Konzeptionelle Grundlagen von Capabilities

### 1 Hydra

- Hydra ist ein Capability-basiertes Betriebssystem
  - ◆ entwickelt Mitte der 1970er Jahre an der Carnegie-Mellon University
  - ◆ lief auf einem speziellen Multiprozessor namens **C.mmp**
  - ◆ Ziel: Experimentierplattform für verschiedene Betriebssystemmechanismen
    - verschiedene Dateisysteme, Scheduler, ...
    - Policy-Mechanism-Separation: Mechanismen im Systemkern, Policies können ausgelagert werden
    - frühes Mikrokern-Konzept
    - Problem: Schutz der "Objekte" vor unberechtigten Zugriffen
  - ◆ Capability-Mechanismen sind integraler Bestandteil des Betriebssystems
  - ◆ Relevanz aus heutiger Sicht:
    - Hydra-Capabilities sind gut mit geschützten Objekt-Referenzen in einer objekt-orientierten Sprachumgebung (wie Java) vergleichbar

### 1 Hydra (2)

- Objekte werden in Hydra durch Capabilities angesprochen und geschützt
  - ◆ Objekte haben einen Typ (z. B. Prozeduren, Prozesse/LNS, Semaphore, Dateien etc.)
  - ◆ Capabilities haben entsprechenden Typ
  - ◆ benutzerdefinierte Typen sind möglich
  - ◆ generische Operationen für alle Typen, implementiert durch das Betriebssystem
  - ◆ Objekte besitzen eine Liste von Capabilities auf andere Objekte (genannt *C-List*)
  - ◆ Capabilities enthalten Rechte
  - ◆ Objekte besitzen einen Datenbereich (impl. durch geschütztes Segment)



## 1 Hydra (3)

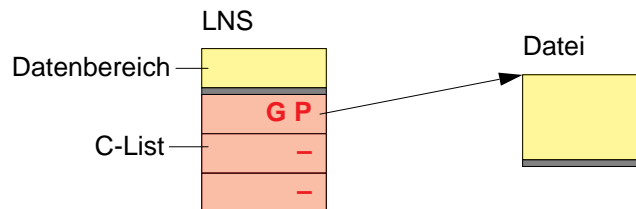
- Prozesse (Subjekte)
  - ◆ Prozesse besitzen einen aktuellen Kontext, den LNS (*Local name space*)
  - ◆ LNS ist ein Objekt
  - ◆ zum LNS gehört ein Aktivitätsträger (Thread)
  - ◆ Prozess kann nur auf Objekte zugreifen, die in der C-List seines LNS stehen (mehrstufige Zugriffe, z. B. auf die C-List eines Objekts, dessen Capabilities in der C-List des LNS steht, sind möglich → Pfad zur eigentlichen Capability)
- Capabilities
  - ◆ Prozesse können nur über Systemaufrufe ihre Capabilities bzw. ihre C-List bearbeiten
  - ◆ Capabilities können nicht gefälscht oder verfälscht werden
  - ◆ Betriebssystem kann sicheres Schutzkonzept basierend auf Capabilities implementieren

## 2 Datenzugriff in Hydra

- Operationen auf dem Datenbereich
  - ◆ *Getdata*: kopiere Abschnitt aus dem Datenbereich eines Objekts in den Datenbereich des LNS
  - ◆ *Putdata*: kopiere Abschnitt aus dem Datenbereich des LNS in den Datenbereich eines Objekts
  - ◆ *Adddata*: füge Daten zu dem Datenbereich eines Objekts hinzu
- Dazugehörige Rechte:
  - ◆ **G**ETRTS: erlaubt den Aufruf von *Getdata*
  - ◆ **P**UTRTS: erlaubt den Aufruf von *Putdata*
  - ◆ **A**DDRTS: erlaubt den Aufruf von *Adddata*
- Rechte müssen in der Capability zum Objekt gesetzt sein

## 2 Datenzugriff in Hydra (2)

- Beispiel: Implementierung von Dateien
  - ◆ *Getdata* erlaubt das Lesen von Daten
  - ◆ *Putdata* erlaubt das Schreiben von Daten
  - ◆ *Adddata* erlaubt das Anhängen von Daten
- Entsprechende Rechte können pro Capability gesetzt werden



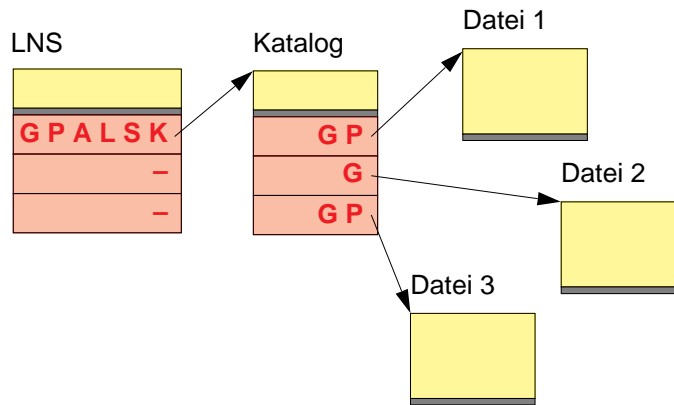
## 3 Zugriff auf Capabilities in Hydra

- Operationen auf der C-List eines Objekts:
  - ◆ *Load*: kopieren einer Capability aus der C-List eines Objekts in die C-List des LNS (  $\hat{=}$  Besorgen einer Objektreferenz von einem Nameserver)
  - ◆ *Store*: kopieren einer Capability aus der C-List des LNS in die C-List eines Objekts (dabei können Rechte maskiert werden)
  - ◆ *Append*: anfügen einer Capability in die C-List eines Objekts
  - ◆ *Delete*: löschen einer Capability aus der C-List eines Objekts
- Rechte:
  - ◆ **L**OADRTS: erlaubt Aufruf von *Load*
  - ◆ **S**TORTS: erlaubt Aufruf von *Store*
  - ◆ **A**PPRTS: erlaubt Aufruf von *Append*
  - ◆ **K**ILLRTS: erlaubt Aufruf von *Delete*



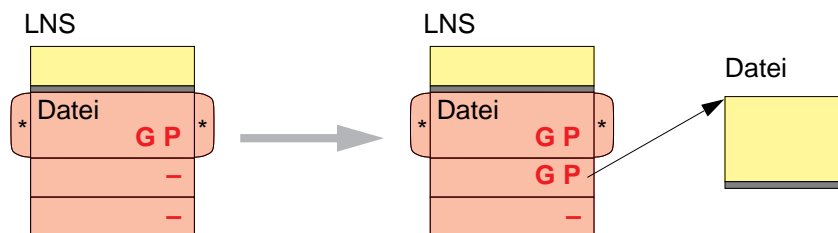
### 3 Zugriff auf Capabilities in Hydra (2)

- Beispiel: Implementierung von Katalogen  
Katalog = Objekt, das Referenzen (Capabilities) auf Dateien enthält
  - ◆ *Load* erlaubt das Auflösen von Namen / Öffnen einer Datei  
(Aufrufer bekommt die Capability)
  - ◆ *Store* und *Append* erlauben das Hinzufügen von Dateien zum Katalog
  - ◆ *Delete* erlaubt das Austragen von Dateien aus dem Katalog



### 4 Objekterzeugung in Hydra

- Objekterzeugung über Erzeugungsschablonen (*Creation Templates*)
  - ◆ mit der *create*-Operation können neue Objekte eines Typs erzeugt werden
  - ◆ LNS benötigt hierzu eine Erzeugungsschablone
  - ◆ Erzeugungsschablone enthält den Typ des neu zu erzeugenden Objektes und eine Rechtemaske
  - ◆ der Aufrufer der *create*-Operation erhält eine Capability auf das neu erzeugte Objekt, die nur die in der Maske angeschalteten Rechte enthält

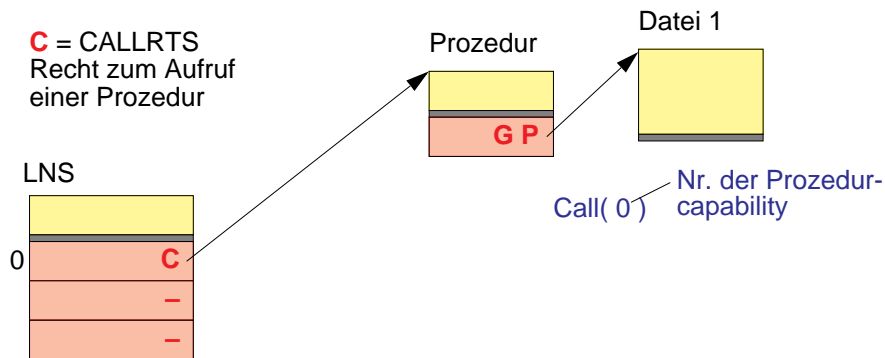


## 5 Prozeduraufruf in Hydra

- Prozedur ist ein Objekt, aus dem beim Aufruf ein neuer LNS im laufenden Prozess erzeugt wird

- ◆ neuer LNS wird aktueller Kontext (alte LNS stehen auf einem Stack; sie werden wieder aktiviert, wenn Prozedur zu Ende)

- Aufruf einer Prozedur



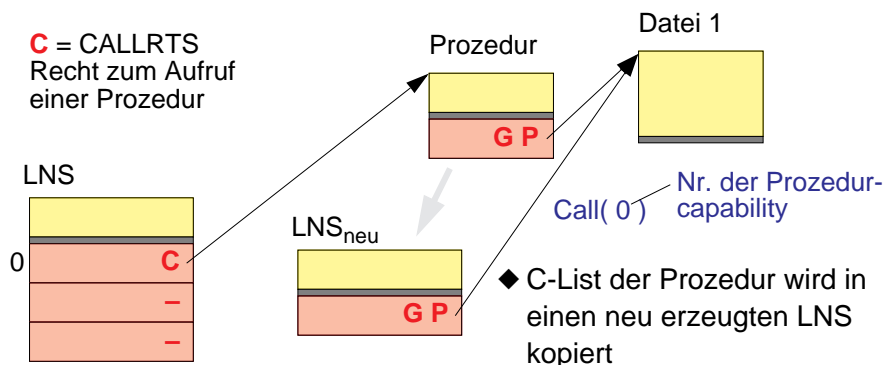
- Prozedur kann initiale Capabilities haben

## 5 Prozeduraufruf in Hydra

- Prozedur ist ein Objekt, aus dem beim Aufruf ein neuer LNS im laufenden Prozess erzeugt wird

- ◆ neuer LNS wird aktueller Kontext (alte LNS stehen auf einem Stack; sie werden wieder aktiviert, wenn Prozedur zu Ende)

- Aufruf einer Prozedur

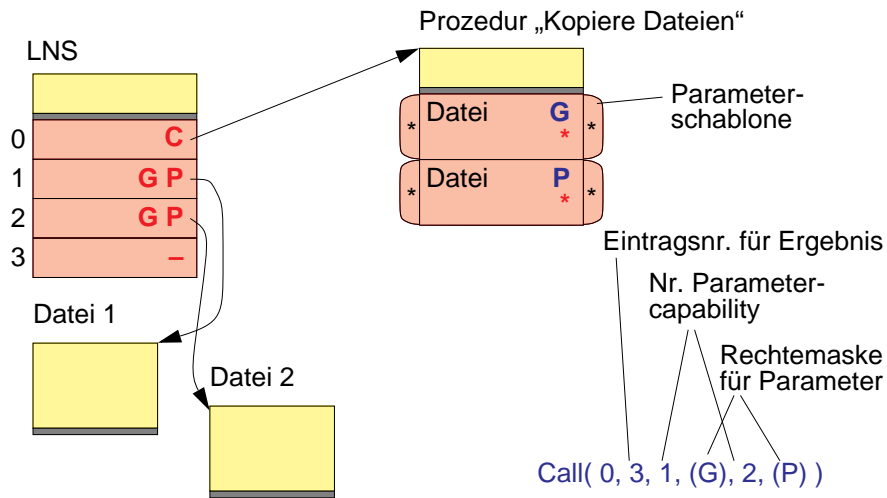


- Prozedur kann initiale Capabilities haben  
... und weitere durch Parameter übergeben bekommen

## 5 Prozeduraufruf in Hydra (2)

### ■ Übergabe von Parametern

◆ Beispiel: Prozedur zum Kopieren von Dateiinhalten

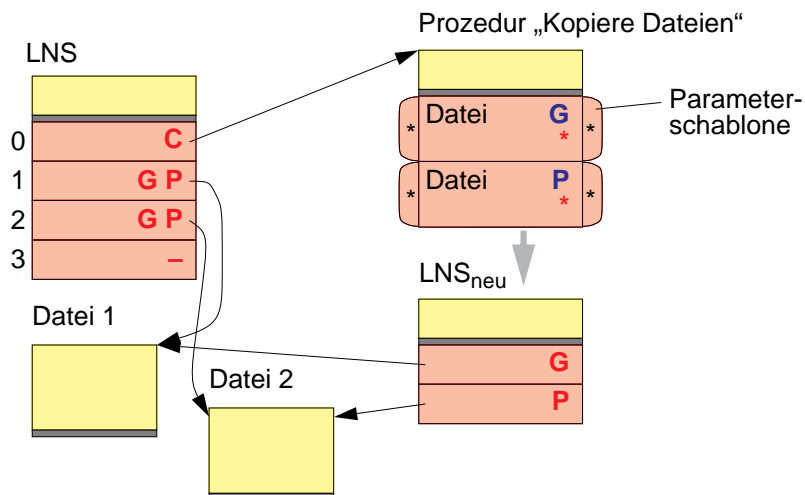


➤ Parameterschablone zum Überprüfen des Typs des Parameters und der mindestens erforderlichen Rechte

## 5 Prozeduraufruf in Hydra (3)

### ■ Übergabe von Parametern

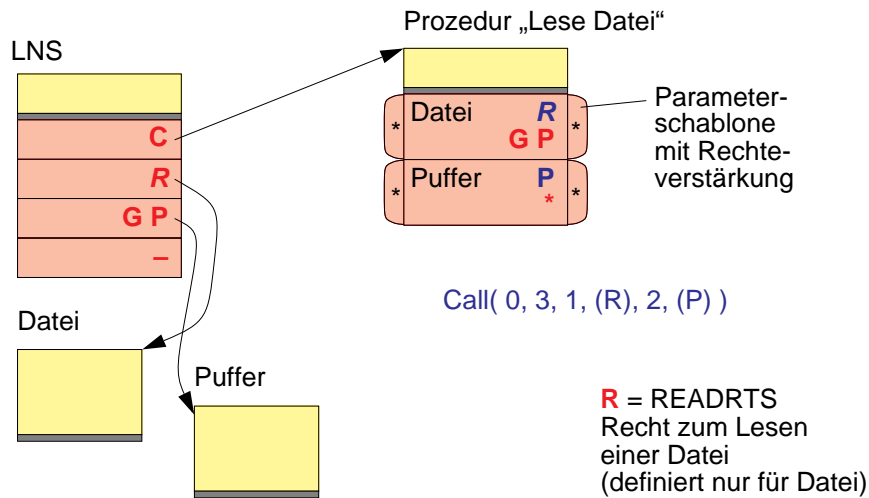
◆ Beispiel: Prozedur zum Kopieren von Dateiinhalten



## 5 Prozeduraufruf in Hydra (3)

### ■ Verstärken von Rechten

- ◆ Beispiel: Prozedur zum Lesen von Dateiinhalten in einen Puffer



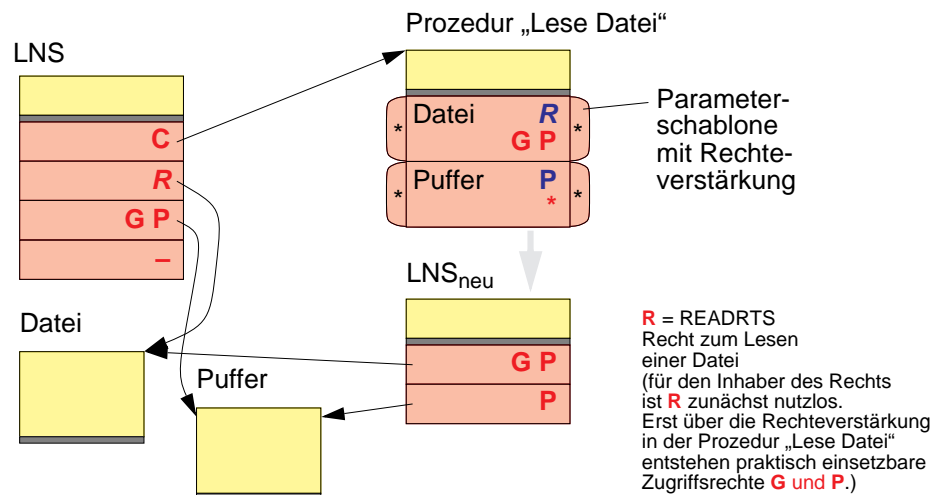
- Prozedur kann mehr Rechte besitzen, als der Aufrufer der Prozedur (analog zu s-bit bei UNIX-Programmen)

SYSSEC

## 5 Prozeduraufruf in Hydra (3)

### ■ Verstärken von Rechten

- ◆ Beispiel: Prozedur zum Lesen von Dateiinhalten in einen Puffer



SYSSEC

## 6 Problem: Gegenseitiges Misstrauen

- Aufrufer misstraut einer Prozedur
  - ◆ Aufrufer möchte der Prozedur nur soviel Rechte einräumen wie nötig
- Aufgerufene Prozedur misstraut dem Aufrufer
  - ◆ Aufrufer soll nur soviel Rechte und Zugang bekommen wie erforderlich
- ★ Hydra Prozeduraufruf unterstützt diese Forderungen direkt
  - ◆ Aufrufer übergibt Capabilities, die nötig sind
  - ◆ Aufrufer kann Rechte bei der Übergabe maskieren und damit ausschalten
  - ◆ Aufrufer erhält nur Zugang zu einem definierten Ergebnis
  - ◆ Prozedur kann eigene Capabilities besitzen, die einem LNS zur Verfügung stehen und die dem Aufrufer verborgen bleiben können

## 6 Problem: Gegenseitiges Misstrauen (2)

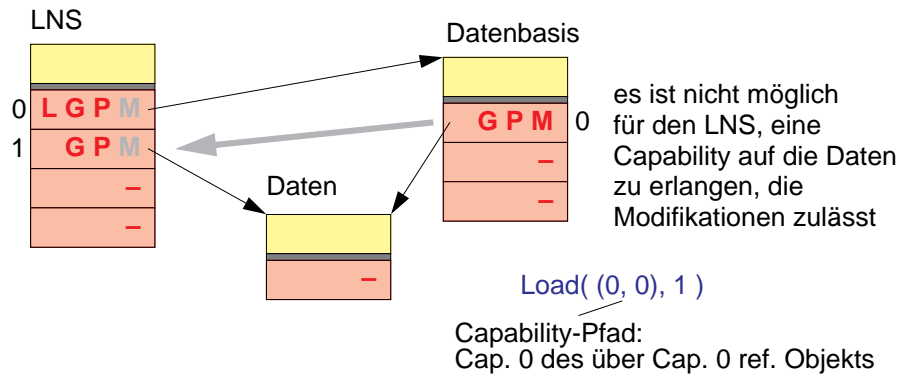
- ▲ Rechteverstärkung als Sicherheitslücke?
  - ◆ Verstärkungsschablone wird nur an vertrauenswürdige Prozeduren ausgegeben und kann nicht einfach erzeugt werden

## 7 Problem: Modifikationen

- Aufrufer möchte Modifikationen an und über Parameter ausschließen
  - ◆ eine Prozedur soll nichts verändern können
- Wegnehmen der entsprechenden Rechte reicht nicht
  - ◆ Prozedur kann lesend zu neuen Capabilities gelangen und über diese Änderungen vornehmen (Transitivität)
  - ◆ Rechteverstärkung könnte angewandt werden

## 7 Problem: Modifikation (2)

- ★ Einführung des Modifikationsrechts **MDFYRTS**
  - ◆ für alle modifizierenden Operationen an Datenbereichen und C-Lists muss zusätzlich das Modifikationsrecht für das Objekt vorhanden sein
  - ◆ Modifikationsrecht wird automatisch gelöscht, wenn eine Capability über einen Pfad geladen wird, auf dem eine der Capabilities kein Modifikationsrecht besitzt
  - ◆ Modifikationsrecht kann nicht über Rechteverstärkung erlangt werden



## 7 Problem: Modifikation (3)

- Parameterübergabe
  - ◆ Wegnahme des Modifikationsrecht bei Parametern stellt sicher, dass die aufgerufene Prozedur keinerlei Veränderungen beim Aufrufer durchführen kann

## 8 Problem: Ausbreitung von Capabilities

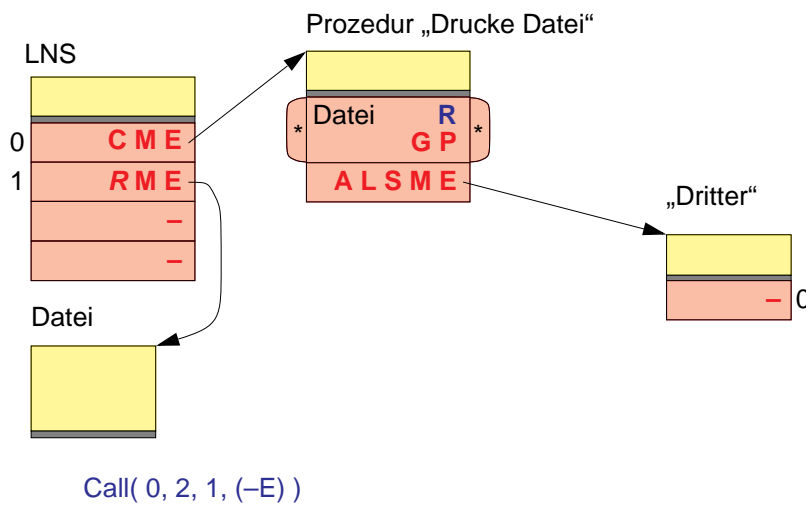
- Aufrufer will verhindern, dass eine übergebene Capability vom Aufgerufenen an einen Dritten weitergegeben wird (*Propagation Problem*)
  - ◆ Beispiel: Prozedur „Drucken“ soll niemandem eine Referenz auf die zu druckenden Daten weitergeben können

## 8 Problem: Ausbreitung von Capabilities (2)

- ★ Einführung des Environment-Rechts **ENVRTS**
  - ◆ für das Speichern oder Anfügen einer Capability an eine C-List muss die zu speichernde Capability selbst das Environment-Recht besitzen
  - ◆ Environment-Recht wird automatisch gelöscht, wenn eine Capability über einen Pfad geladen wird, auf dem eine der Capabilities kein Environment-Recht besitzt
  - ◆ Environment-Recht kann nicht über Rechteverstärkung erlangt werden

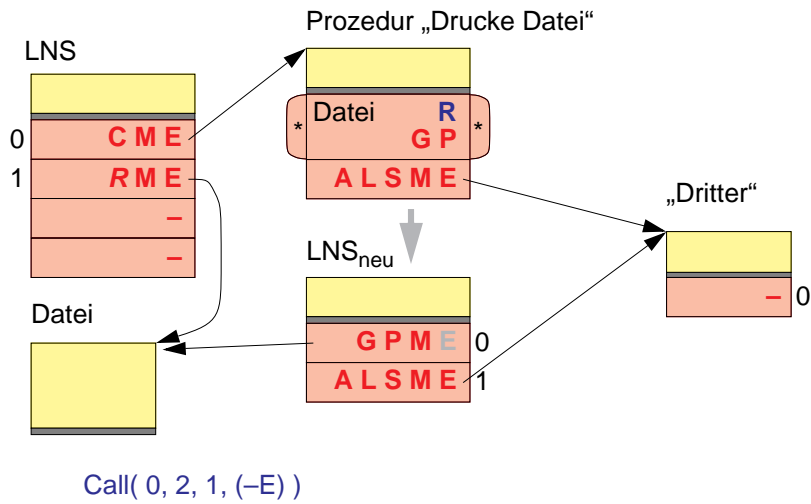
## 8 Problem: Ausbreitung von Capabilities (3)

- Versuchte Weitergabe einer Capability an einen Dritten



## 8 Problem: Ausbreitung von Capabilities (3)

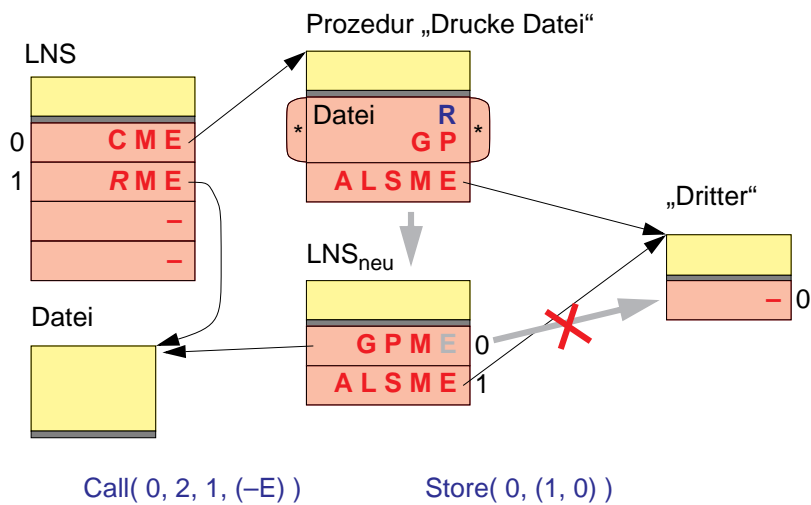
- Versuchte Weitergabe einer Capability an einen Dritten



SYSSEC

## 8 Problem: Ausbreitung von Capabilities (3)

- Versuchte Weitergabe einer Capability an einen Dritten



SYSSEC



## 9 Problem: Aufbewahrung von Capabilities

- Aufrufer möchte sicher sein, dass Aufgerufener keine Capabilities nach der Bearbeitung des Aufrufs zurückbehalten kann (*Conservation Problem*)
  - ★ Environment-Recht zusammen mit dem Aufrufmechanismus genügt
    - ◆ Aufgerufener kann Capability ohne ENVRTS nicht weitergeben und folglich nicht abspeichern
    - ◆ der LNS des Aufrufs wird mit Beendigung des Aufrufs vernichtet, so dass die übergebenen Capabilities nicht zurückbehalten werden können
    - ◆ ENVRTS wirkt transitiv, so dass auch die über eine Parameter-Capability gewonnenen Capabilities nicht weitergegeben werden können

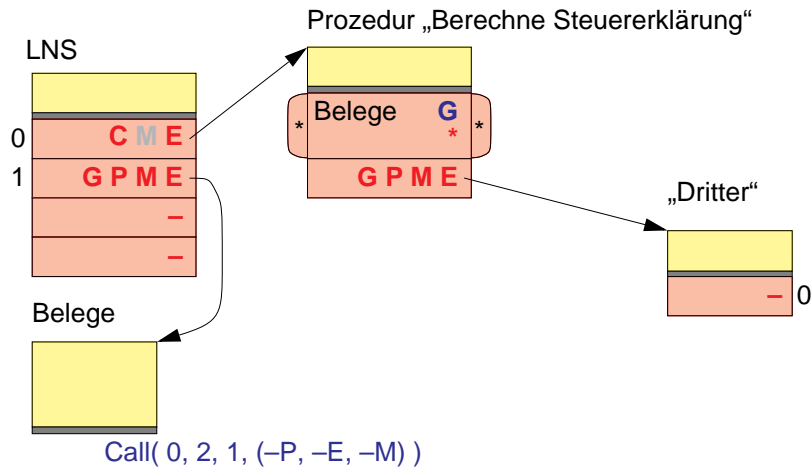
## 10 Problem: Informationsflussbegrenzung

- Aufrufer möchte die Verbreitung von Informationen aus übergebenen Parametern einschränken (*Confinement Problem*)
  - ◆ selektiv: bestimmte Informationen sollen nicht nach außen gelangen
  - ◆ global: gar keine Informationen sollen nach außen gelangen
  - ◆ ENVRTS ist nicht ausreichend, da Prozedur den Dateninhalt von Parameterobjekten kopieren könnte (ENVRTS wirkt nur auf die Weitergabe von Capabilities)
- Hydra realisiert nur globale Informationsflussbegrenzung
  - ★ Modifikationsrecht auf der Prozedur-Capability
    - ◆ wenn kein Modifikationsrecht vorhanden ist, werden bei allen in den LNS übernommenen Capabilities die Modifikationsrechte ausgeschaltet (gilt jedoch nicht für Parameter)

## 10 Problem: Informationsflussbegrenzung (2)

■ Beispiel: Prozedur zur Steuerberechnung

- ◆ die übergebenen Beleg- und Buchhaltungsdaten sollen nicht weitergegeben werden können

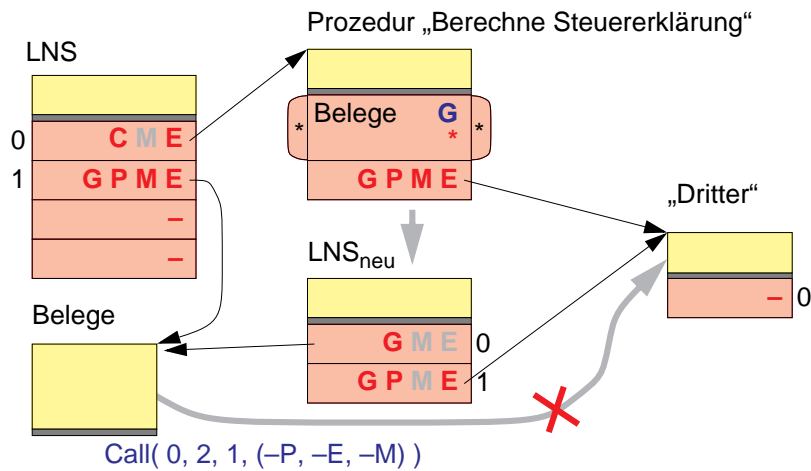


SYSSEC

## 10 Problem: Informationsflussbegrenzung (2)

■ Beispiel: Prozedur zur Steuerberechnung

- ◆ die übergebenen Beleg- und Buchhaltungsdaten sollen nicht weitergegeben werden können



SYSSEC

## 11 Problem: Initialisierung

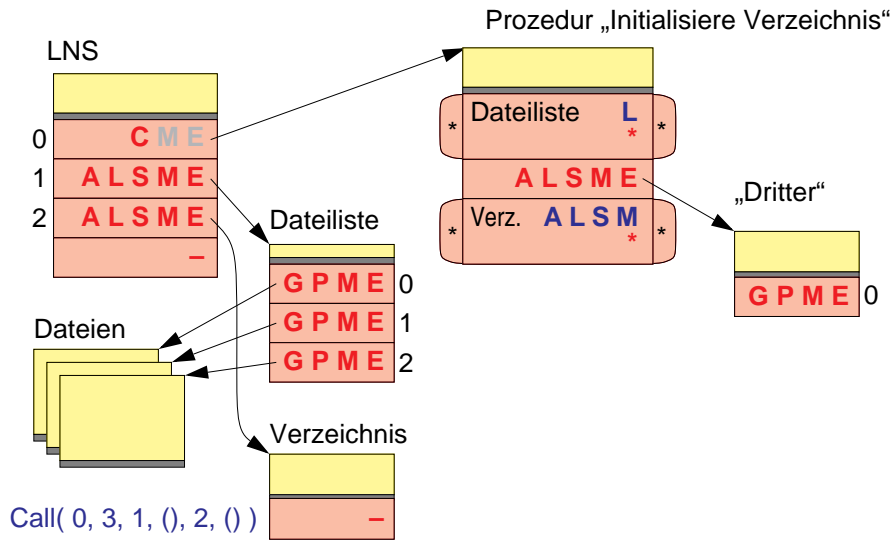
- Initialisierung von Objekten durch Prozeduren
  - ◆ Übergabe eines Objekts und verschiedener Capabilities, mit denen das Objekt initialisiert werden soll
  - ◆ Problem: Parameter-Capabilities müssen Environment-Recht besitzen (sonst ist das zu initialisierende Objekt nicht arbeitsfähig), gleichzeitig soll aber die Ausbreitung solcher Capabilities eingeschränkt werden
    - Lösung: Wegnahme des Modifikationsrechts auf der Prozedurcapability
  - ◆ Problem: Es muss verhindert werden, dass die Prozedur in das zu initialisierende Objekt eigene oder fremde Capabilities einsetzt, so dass es später Einfluss auf das zu initialisierende Objekt nehmen kann
  
- Beispiel: Prozedur zur Initialisierung eines Katalogs bekommt Capabilities auf die entsprechenden Dateien
  - ◆ es soll sichergestellt werden, dass Prozedur keine eigenen Dateicapabilities in den Katalog einfügt

## 11 Problem Initialisierung (2)

- ★ Environment-Recht auf der Prozedur-Capability
  - ◆ wenn kein Environment-Recht vorhanden ist, werden bei allen in den LNS übernommenen Capabilities die Environment-Rechte ausgeschaltet (gilt jedoch nicht für Parameter)
  - ◆ durch das fehlende Environment-Recht können alle bereits vorhandenen Capabilities nicht in das zu initialisierende Objekt gespeichert werden

# 11 Problem: Initialisierung (3)

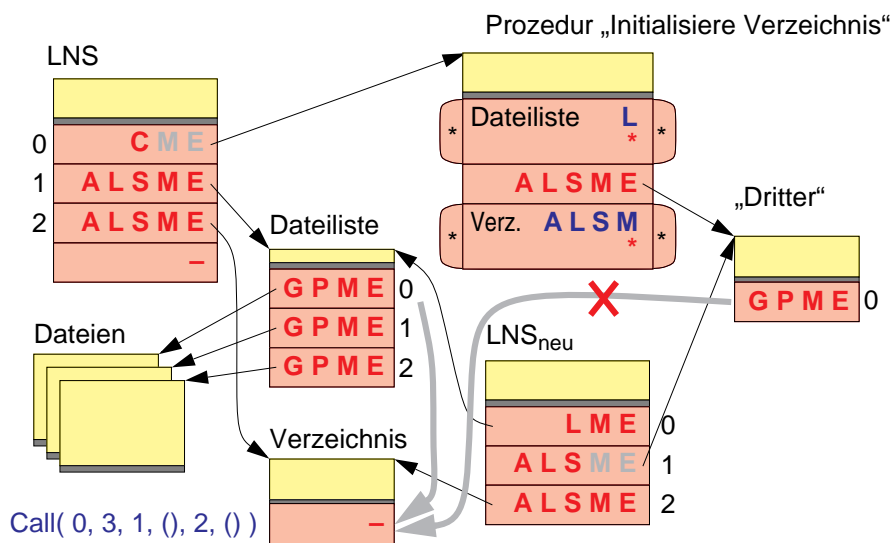
■ Beispiel: Initialisierung eines Verzeichnis



SYSSEC

# 11 Problem: Initialisierung (3)

■ Beispiel: Initialisierung eines Verzeichnis



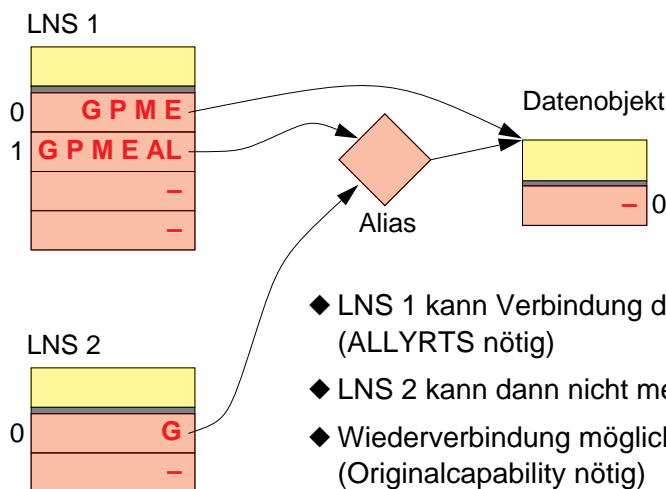
SYSSEC

## 12 Rückruf von Capabilities

- Anwender möchte ausgegebene Capabilities für ungültig erklären
  - ◆ sofortiger Rückruf — Rückruf nach einiger Zeit erst wirksam
  - ◆ dauerhafter Rückruf — Rückruf nur zeitlich begrenzt wirksam
  - ◆ selektiver Rückruf — Rückruf für alle Benutzer eines Objekts
  - ◆ partieller Rückruf — Rückruf aller Rechte an einem Objekt
  - ◆ Recht zum Rückruf; Rückruf des Rückrufrechts
- ★ Hydra setzt sogenannte Aliase ein
  - ◆ Alias ist eine Indirektionsstufe zu Capabilities
  - ◆ Statt auf ein Objekt können Capabilities auf Aliase verweisen und diese wiederum auf andere Aliase oder schließlich auf das eigentliche Objekt
  - ◆ Verbindung vom Alias zum Objekt kann gelöst werden: Fehler beim Zugriff
  - ◆ Recht zum Lösen der Verbindung **ALLYRTS** (engl. *ally* = verbünden)

## 12 Rückruf von Capabilities (2)

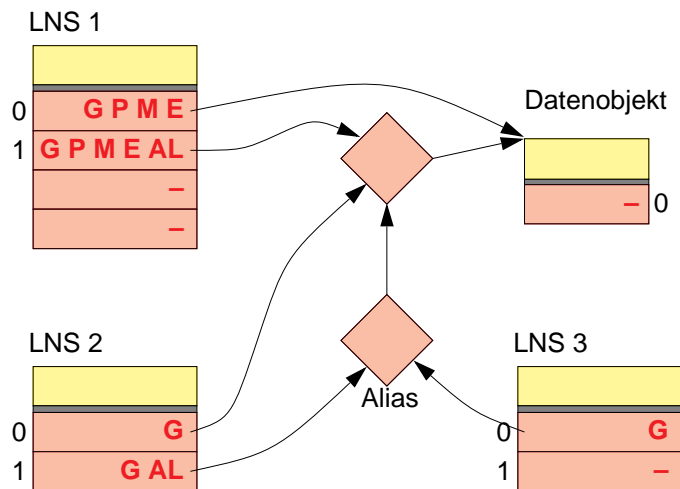
- Beispiel: Weitergabe einer rückrufbaren Capability



- ◆ LNS 1 kann Verbindung des Alias lösen (ALLYRTS nötig)
- ◆ LNS 2 kann dann nicht mehr zugreifen
- ◆ Wiederverbindung möglich (Originalcapability nötig)
- ◆ Aliase können hintereinander auftreten

## 12 Rückruf von Capabilities (3)

### ■ Beispiel: Aliasketten

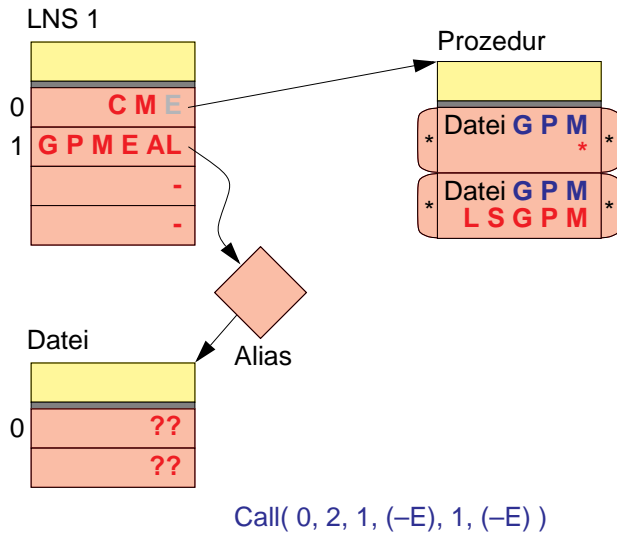


## 12 Rückruf von Capabilities (4)

- ▲ Problem: Rückruf während der Bearbeitung eines Objekts
  - ◆ inkonsistente Zustände möglich
- ★ Lösung in Hydra
  - ◆ Parameter-Capabilities, die durch eine rechtheverstärkende Parameterschablone angenommen werden, zeigen auf das Originalobjekt
- ▲ Nachteil
  - ◆ nicht vertrauenswürdige Prozeduren können rückruffreie Capability erlangen
  - ◆ Problem fällt in die selbe Kategorie wie rechtheverstärkende Parameterschablonen an sich

## 12 Rückruf von Capabilities (5)

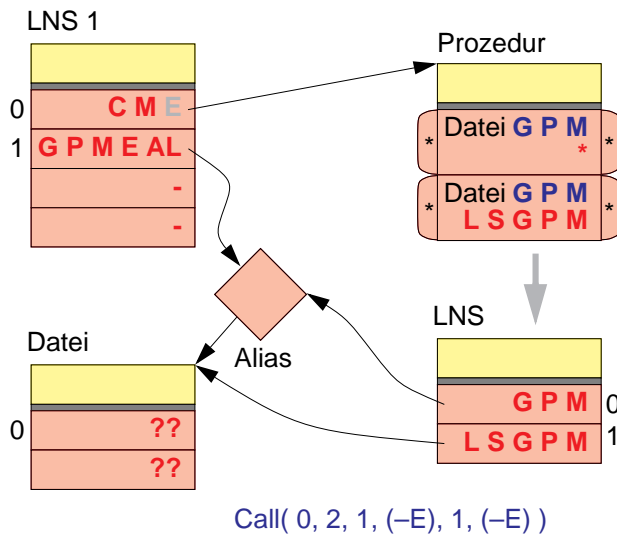
■ Beispiel:



SYSSEC

## 12 Rückruf von Capabilities (5)

■ Beispiel:



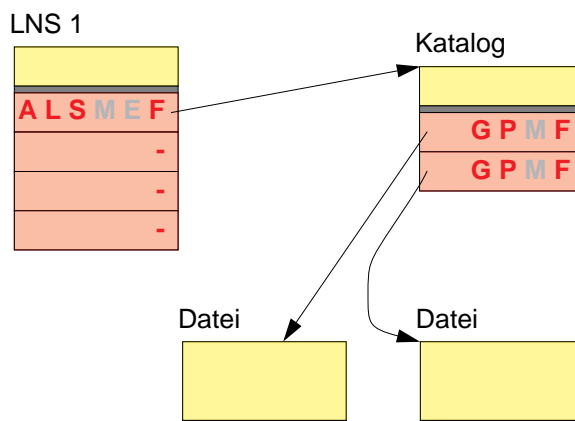
SYSSEC

## 13 Garantierter Zugriff

- Schutz vor Rückruf
  - ◆ Modifizierende Benutzer eines Objekts
    - kooperierende Benutzer: Rückruf keine Gefahr
    - nicht kooperierende Benutzer: Rückruf nötig
  - ◆ Benutzer eines Objekts ohne modifizierende Zugriffe
    - Aufruf von Prozeduren ist unkritisch, da Aufrufe nicht rückrufbar sind
    - lesende Zugriffe auf Objekte sind kritisch:
      - Verhindern des Rückrufs ist jedoch nicht ausreichend
      - Daten im Objekt könnten gelöscht oder verfälscht werden
      - Capabilities im Objekt oder deren Rechte könnten entfernt werden
- ★ Hydra führt das Einfrierrecht (*Freeze right FRZRTS*) ein
  - ◆ Einfrieren nimmt Modifikationsrecht weg
  - ◆ ein Objekt kann nur gefroren werden, wenn alle Capabilities in der C-List bereits das Einfrierrecht haben

## 13 Garantierter Zugriff (2)

- Beispiel:





## 14 Bewertung von Hydra

- Hydra demonstrierte die Beherrschbarkeit einer ganzen Reihen von Sicherheitsproblemen
  - ◆ Ergebnisse flossen in eine ganze Reihe von Systemen
  - ◆ reine Capability-basierte Systeme haben sich jedoch nie durchgesetzt
- Hydras Probleme
  - ◆ lagen im wesentlichen nicht am Capability-Mechanismus
  - ◆ es gabe keine vernünftigen Editoren und Compiler
  - ◆ Hardware besaß keine Spezialhardware zur Unterstützung von Paging

## D.5 Schutz von Dateien

### 1 Zugriffskontrolle in UNIX

- Beispiel für Zugriffskontrolle in einem klassischen Betriebssystem
  - zu schützende Objekte: Verzeichnisse und Dateien
  - Subjekte: Benutzer, Benutzergruppen und Prozesse
  - ➔ Zugriffskontrollkonzept zunächst sehr beschränkt
    - spezielle Objekte (Geräte, Prozesse, Arbeitsspeicher) werden als Dateien modelliert (*special files*)
    - Konzept wurde nachträglich auch auf einige weitere Systemobjekte (Semaphore, Shared Memory Segmente, Message Queues) erweitert
  - ➔ Zugriffskontrolle deckt die meisten relevanten Objekte im System ab
    - ABER: die Abbildung auf den Typ "Datei" mit den Operationen *read*, *write*, *ioctl* ist äußerst grobgranular

## 1 Zugriffskontrolle in UNIX (2)

- Zugriffsrechte pro Datei
  - lesen, schreiben, ausführen
  - ggf. ergänzt durch individuellere ACLs
- Zugriffsrechte im Dateibaum
  - lesen, schreiben (=Einträge erzeugen/löschen), durchgreifen
  - Rechte auf Directories limitieren ggf. den Zugriff auf darunterliegende Dateien
  - Durchgriffsrecht ohne Leserecht wird oft zum "Verstecken" von Dateien verwendet → fragwürdiger Schutz
  - Schreibrecht auf Directory kann mit Rechten darunterliegender Dateien kollidieren
    - Löschen und Neuanlegen einer Datei, auf die man kein Schreibrecht hätte ist möglich

## 2 Zugriffskontrolle unter Windows

- Authentisierung eines Benutzers führt zur Vergabe eines *Access Tokens* (enthält Security-Ids des Benutzers und seiner Gruppen, Default-ACL, ...)
  - entspricht einer Credential-Struktur
  - Access Token wird Prozess zugeordnet und an neu erzeugte Prozesse vererbt
- Objekte enthalten Security Descriptor
  - SID des Benutzers und der Gruppe
  - Discretionary ACL mit Liste von ACE (Access Control Elements)
    - Erlaubnis- und Verbots-ACEs
    - geben SID und erlaubte/verbotene Operationen an
    - pro Objekt 16 verschiedene Zugriffsrechttypen möglich (z. B. synchronize, write\_owner, write\_DAC, read, write)
- Zugriffsberechtigung bei open
  - anschließend Vergabe eines Object Handles (Capability) an den Prozess zur Vorlage bei den konkreten Zugriffe

## D.6 Verschlüsselung von Dateien

### 1 Motivation

- 2004 in Londoner Taxis liegen geblieben: 4.973 Laptops, 5.838 PDAs, 63.135 Mobiltelefone
- Reuters, 2005: Citigroup meldet den Verlust von Bändern mit den Daten - einschl. Sozialversicherungsnummern - von 3,9 Mio. Kunden
- Leyden, 2004: Kundendatenbank und die aktuellen Zugriffscodes für das Intranet eines der größten Finanzdienstleister in EU waren auf einer Platte, die bei eBay zum Verkauf angeboten worden war
- Noguchi, 2005: Laptop mit Namen und Sozialvers.nummern von 16.500 MCI Mitarbeitern wurde gestohlen.
- Reuters, 2005: Iron Mountain ... meldet Verlust von Bändern mit den Daten von 600.000 Time Warner Arbeitern

...

### 2 Anforderungen

- Schutz der Dateiinhalte
- Schutz der Metadaten
- Schutz muss auch gewährleistet sein
  - ▶ bei direktem Zugriff auf Datenträger
  - ▶ Transport der Daten über Netzwerke
  - ▶ für Datensicherung und Archivierung
    - langfristige Schlüsselaufbewahrung
    - automatische Schlüsselaktualisierung
- Mehraufwand für Benutzer begrenzen
  - ▶ Schutz möglichst transparent gewährleisten
- Einsatz in Mehrbenutzerumgebungen
  - ▶ Gruppenzugriffe
- Umgang mit Schlüsselverlust

### 3 Programme zur Dateiverschlüsselung

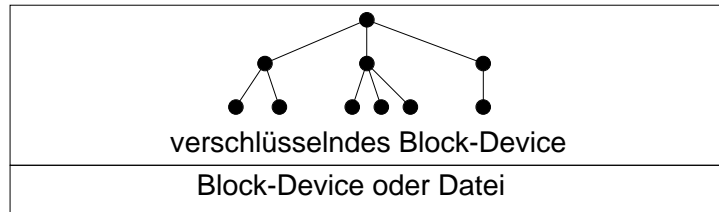
- Benutzer kann individuell für einzelne Dateien entscheiden, ob und wie sie verschlüsselt werden sollen
- Realisierung
  - ◆ Werkzeuge
    - ▶ PGP, GnuPG, Ncrypt
  - ◆ Betriebssystemmechanismen
- + Verschlüsselung nur dort, wo es tatsächlich erforderlich ist
- + unterschiedliche Schlüssel für unterschiedliche Dateien möglich
- Datei-Metadaten (Name, Attribute) sind nicht verschlüsselt
- großes Risiko undichter Stellen (*Leakage*)

### 4 Undichte Stellen bei Verschlüsselung einzelner Dateien

- Um mit den Daten zu arbeiten, müssen sie unverschlüsselt vorliegen
- Viele Programme erzeugen temporäre Kopien
  - ▶ bei Abstürzen des Programms oder des Rechners können die Kopien unbemerkt liegen bleiben
  - ▶ auch wenn die Dateien gelöscht werden, bleiben die Daten auf der Platte, bis der Platz wiederverwendet wird
- Absturz/Abschalten des Rechners während die Verschlüsselung noch läuft
- Swap-Partition enthält Teile des Arbeitsspeichers
- ➔ Verschlüsselte Daten sickern in unverschlüsselte Teile des Speichermediums durch
- ➔ Dateiverschlüsselung wird wirkungslos

## 5 Verschlüsselung von Dateisystem-Partitionen

- Schicht zwischen Dateisystem und Plattentreiber sorgt für Verschlüsselung
  - Pseudo-Plattentreiber - verhält sich "nach oben" wie eine Partition, ver-/entschlüsselt und greift auf Dateien oder "normale" Partition zu



- Aufbau auf herkömmlichem Block-Device: alle Daten werden einheitlich verschlüsselt
- Aufbau auf Datei: Benutzer kann sich mehrere kleine Dateisysteme schaffen, die unterschiedlich verschlüsselt sind
- ◆ Beispiel: *dm-crypt* für Linux

## 5 Verschlüsselung von Dateisystem-Partitionen (2)

- + Vorteile
  - Datei-Metadaten (Name, Attribute) werden mitverschlüsselt
  - alle Daten auf der Partition werden grundsätzlich verschlüsselt
  - Risiko undichter Stellen wird erheblich reduziert
- Nachteile
  - Dateibaum und Dateiattribute nicht mehr erkennbar
    - z. B. kein selektives Backup mehr möglich
  - ein gemeinsamer Schlüssel für alle Daten der Partition
  - Austausch von Schlüsseln schwierig
  - so lange nicht ALLE Partitionen verschlüsselt werden bleibt das Risiko undichter Stellen
    - problematisch vor allem bei kleinen, benutzerspezifischen Dateisystemen

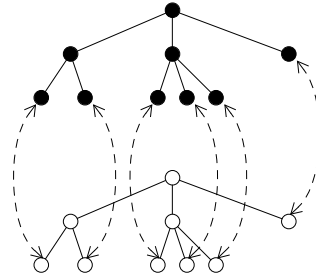
## 6 Verschlüsselnde Dateisysteme

- Aufsatz eines verschlüsselnden Dateisystems auf einem normalen unverschlüsselten Dateisystem

- *Stacked Filesystem*

- ◆ Grundprinzip:

Inhalt und Meta-Daten von Dateien des verschlüsselnden Dateisystems werden in Dateien des darunterliegenden Dateisystems abgelegt



- Struktur der Partition bleibt sichtbar (Directorystruktur und Dateiattribute)

## 6 Verschlüsselnde Dateisysteme (2)

- Design-Alternativen

- ◆ Verschlüsselung von Datei- und Directorynamen

- + zusätzliche Sicherheit, da keine Schlüsse aus den Datei-Metadaten gezogen werden können
  - Management wird komplexer (z. B. Backup-Restaurierung)

- ◆ Realisierung als Teil des Betriebssystemkerns

- z. B. EFS unter Windows, eCryptfs in Linux
  - + bessere Performance
  - weniger flexibel

- ◆ Realisierung als Anwendungsprozess (*Userland File System*)

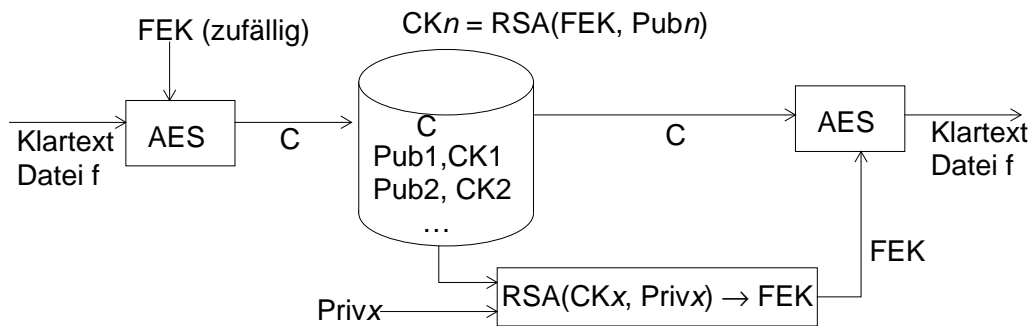
- z. B. CryptoFS oder EncFS unter Linux
  - mehr Flexibilität, schlechterer Performance

- ◆ Ein- oder Mehrbenutzerfähigkeit

- wird ein einheitlicher Schlüssel verwendet oder kann jeder Benutzer eigene Schlüssel einsetzen?

## 7 Verschlüsselnde Dateisysteme — EFS

- seit Windows 2000, Erweiterungen in XP und Vista
  - Verschlüsselung mit AES, früher DES3 oder DESX
  - setzt auf NTFS auf (nicht auf FAT-Dateisystemen)
- Mehrbenutzerfähig
  - Datei mit zufälligem Schlüssel (FEK - File Encryption Key) verschlüsselt
  - FEK wird mit Public Key aller berechtigten Benutzer verschlüsselt (CK), Ablage in Attributen der verschlüsselten Datei



## 7 Verschlüsselnde Dateisysteme — EFS (2)

- ▲ Kritische Punkte
- Speicherung der privaten Schlüssel
  - idealerweise auf Smartcard (ab Vista unterstützt)
  - sonst verschlüsselt (mit Passwort der Benutzers oder Smartcard-Schlüssel) auf der Festplatte
    - privater Schlüssel kann bei login in EFS-Prozess geladen werden und steht danach für den Benutzer zur Verfügung
    - EFS-Prozess sperrt Seiten mit Schlüssel-Information gegen Paging
    - im Hauptspeicher sind die privaten Schlüssel aber vorhanden
    - Angriffe auf den EFS-Prozess?
    - Benutzerpasswort meist deutlich schwächer als erzeugte Schlüssel
    - Änderung des Benutzerpassworts durch Admin führt dazu, die privaten Schlüssel für den Benutzer nicht mehr lesbar sind (→ private Schlüssel außerhalb sichern!)

## 7 Verschlüsselnde Dateisysteme — EFS (3)

- ▲ Kritische Punkte (2)
- Schlüssel-Recovery
  - ▶ Recovery-Agent (z. B. Administrator) kann verschlüsselte Dateien restaurieren
  - ▶ FEKs werden auch mit Public Key des Recovery Agenten verschlüsselt
    - aus X509-Zertifikat des Recovery Agenten
    - wenn Zertifikat abgelaufen ist, muss der Recovery-Schlüssel bei erneutem Speichern einer Datei erneuert werden
  - ▶ Private Recovery Key kann außerhalb des Systems verwahrt werden (eigentlich MUSS! - sonst reicht es, Administrator-Rechte zu erlangen, um alle verschlüsselten Dateien entschlüsseln zu können)
    - alte Schlüssel müssen aufbewahrt werden (für alte Dateien)

## 7 Verschlüsselnde Dateisysteme — EFS (4)

- ▲ Kritische Punkte (3)
- Verschlüsselung auf NTFS-Dateisysteme beschränkt
  - ▶ vom Benutzer für ein Verzeichnis und ggf. alle Unterverzeichnisse einstellbar
  - ▶ Dateien werden beim Kopieren auf andere Dateisysteme entschlüsselt
  - ▶ Transfer von Dateien über Netzverbindungen unverschlüsselt
- Verschlüsselung der privaten Schlüssel mit dem Benutzer-Passwort
  - ▶ schwaches Glied in der Kette
  - ▶ Problem wenn Administrator Benutzer-PW ändert
    - ↳ privaten Schlüssel extrahieren und separat speichern



## 8 Verschlüsselnde Dateisysteme — eCryptfs

- In Linux ab Version 2.6.19
- Ähnliche Struktur wie EFS
  - ▶ mehrbenutzerfähig
  - ▶ zufälliger FEK wird mit Public Key verschlüsselt und in Metadaten der Datei abgelegt
- Schlüsselverwaltung über *user session key ring* im Kern oder über user-level Prozess *ecryptfsd*

## 9 Generelles Problem partieller Verschlüsselung

- Partielle Verschlüsselung schützt bei Verlust des Mediums oder des Rechners
- sie schützt nur teilweise bei Angriffen auf das laufende System
  - ▶ während der Datenbearbeitung existieren - zumindest temporär - immer undichte Stellen
- ➔ nur in Kombination mit absolut dichtem Zugriffsrechtekonzept wirkungsvoll
- ▲ ABER: zwei große Schwachstellen:
  - ◆ Administrator (Super-User)
    - ▶ hat Zugriff auf Speicher und unverschlüsselte Partitionen
  - ◆ das Betriebssystem selbst
    - ▶ was tut eigentlich das Betriebssystem?
    - ▶ so lange das Betriebssystem selbst nicht verschlüsselt (oder signiert) wird, ist es immer angreifbar

## 9 Generelles Problem partieller Verschlüsselung (2)

- ? wie sicher sind Daten und Programme auf einem Notebook
  - in einem Hotel-Zimmer
  - im Büro
  - in aufgegebenem Gepäck
  - in einem Schließfach am Bahnhof
  - in einem geparkten Auto
- ? welches Betriebssystem läuft eigentlich auf dem Rechner (danach!)?
- ? welchen Wert hat jetzt die Verschlüsselung von Dateien oder Partitionen?
- ➔ so lange man das Gerät nicht immer im Blick hat, kann man nicht sicher sein, was mit ihm passiert ist!

## 9 Generelles Problem partieller Verschlüsselung (3)

- ➔ Lösung
  - Verschlüsselung aller Partitionen inklusive der Root-Partition und der Swap-Bereiche
    - normales Booten nicht mehr möglich
  - Booten von sicherem Medium
    - USB-Stick
    - CD
  - ◆ Eingabe des Root-Dateisystemschlüssels in der frühen Boot-Phase
  - ▲ ABER: auch solch ein System ist angreifbar, wenn es läuft!
    - Netzwerkzugriffe
    - unverschlüsselte NFS-Dateisysteme
    - Schwachstellen in Programmen

## D.7 Trusted Computing

### 1 Motivation

- Verschlüsselte Dateisysteme schützen nur statische Daten
  - Schwachstelle: Software zur Laufzeit
    - Betriebssystem, Anwendungen
- Sicherheitskonzepte zur Laufzeit ebenfalls unzureichend  
Basis: Speicherschutz, Adressräume
  - Schwachstelle: Betriebssystem - vor allem Treiber
    - woher kommt Software, ist sie vertrauenswürdig?
  - Auslagerung von Treibern in eigene Adressräume bringt wenig
    - DMA-Geräte können direkt auf Speicher zugreifen
  - Signieren von Software kann helfen
    - ABER: wer überprüft die Signatur?
    - ist diese Einheit angreifbar?

### 2 Fazit: Sicherheitsarchitektur

- Viele Schutzmechanismen sind zu grob-granular
- Es wird eine durchgängig Kette vertrauenswürdiger Einheiten vom Systemstart bis hin zu jeder Anwendung benötigt
- ➔ Vertrauenswürdige Sicherheitsarchitektur benötigt ineinander greifende, wirksame Hardware-, Firmware und Software-Sicherheitskonzepte
  - ◆ Hardware: stellt vertrauenswürdige, nicht umgehbare Basisfunktionen zur Verfügung
    - ➔ Beispiel: sichere Erzeugung und Speicherung von Schlüsseln
      - Nutzung über einheitliche Schnittstelle durch Betriebssystem und Anwendungen
  - ◆ BS und Anwendungen bieten darauf aufbauend komplexere Dienste an
    - ➔ Beispiel: Verschlüsselnder Objektspeicher
  - ◆ Strategien (Policies) werden von den Mechanismen strikt getrennt.
    - Policies können Betriebssystem- oder Anwendungs-abhängig sein
    - Mechanismen sind einheitliche Basis

### 3 TCPA und TCG

- **Trusted Computing Platform Alliance (TCPA)**
  - 1999: Microsoft, Intel, IBM, Compaq, HP
  - 2003: > 200 Mitglieder
  - ➔ handlungsunfähig (Beschlüsse nur einstimmig möglich)
- **TCPA-Ziele**
  - Hard- und Softwarestandards für vertrauenswürdigerer Rechner-Plattformen (→ E-Business-Transaktionen)
  - Plattform = Motherboard, CPU, E/A-Geräte, BIOS, ...
- **TCPA-Funktionen**
  - sicheres Booten
  - *Attestation* (Bestätigung der Integrität der Systemkonfiguration einer Plattform gegenüber Kommunikationspartnern)
  - sichere Generierung und Aufbewahrung von Schlüsseln
- **Trusted Computing Group (TCG)**
  - 2003: AMD, HP, Intel, Microsoft (Beschlüsse mit 2/3-Mehrheit)

### 4 TCG-Architektur

- **Trusted Platform Module (TPM)**  
= Hardware-Teil (TCG-Chip)
- **Root of Trust for Measuring Integrity Metrics (RTM)**  
= Hardware / Firmware-Teil  
(entweder in TPM integriert oder im BIOS implementiert)
- **Trusted Software Stack (TSS)**  
= Software-Teil
- **TPM + RTM + TSS = TCG-Subsystem**
  - vertrauenswürdige Basisdienste als Bausteine für Betriebssysteme
  - OpenTC-Projekt ([www.opentc.net](http://www.opentc.net)) entwickelt Software-Framework für Trusted Computing – u. a. Betriebssystem auf Basis von Xen und L4

## 5 TCG-Subsystem

- TPM: Fest eingelöteter Smartcard-Prozessor
  - ◆ bietet kryptografische Operationen in Hardware
    - Zufallszahlen
    - Schlüsselgenerierung mit privatem Schlüssel der Chip nicht verlässt
    - Signieren und Signaturprüfung
    - Ver- und Entschlüsseln
  
- RTM "misst" in speziellen Registern den Systemzustand
  - bildet Hash über ausgeführte Befehlsfolge
  - damit können Modifikationen an Code festgestellt werden
  
- Sicheres Booten über eine BIOS Erweiterung (CRTM = Core RTM)
  - Spezielle Register "messen" den Bootvorgang vom BIOS über Bootloader zum Systemkern.
  - Softwareschicht kann aufgrund von Messwerten Rückschlüsse auf Systemintegrität ziehen

## 5 TCG-Subsystem (2)

- Integritäts-Messwerte werden in sicherem Speicher (auf TPM) abgelegt
  - *Attestierung* über eine Systemkonfiguration (=Messwert-Report)
  - Änderungen können durch Vergleich mit den ursprünglichen Messwerten entdeckt werden
  
- Anwendungen und Kommunikationspartner können Attestierungen abfragen
  - Gewissheit über Authentizität einer Plattform
  - Nachweis enthält Identität der Plattform + Angaben über konkrete Konfiguration
  
- TSS definiert Standard-API für Zugriff auf TPM
  - Ex- und Import von Kommunikationsschlüsseln
  - Erstellung Hashwerten und Integritätsbescheinigungen
  - Basis für Schnittstellen zu existierenden Krypto-APIs

## 6 Eindeutige Identität und Besitzerkonzept

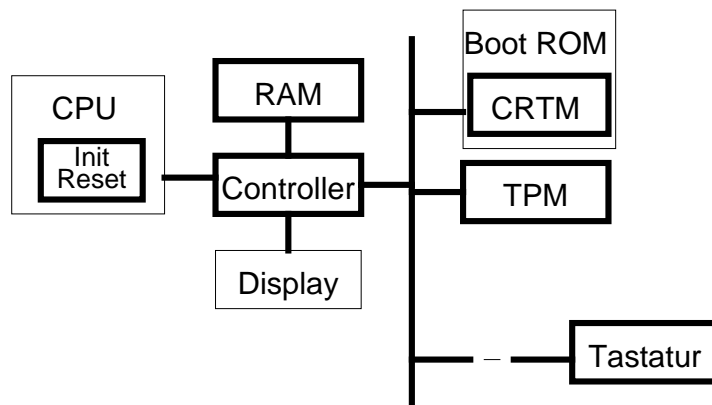
- TPM eindeutig identifizierbar
  - *Endorsement Key* und Zertifikat
  - Integration bei der Chip-Herstellung
  - Privacy-Problem → *Endorsement Key* gegenüber Dritten verschleiern  
→ *Attestation Identity Keys (AIK)*
- Besitzer muss TCG-Plattform explizit übernehmen
  - Festlegung eines Passworts
  - Basis für Aktivierung und Deaktivierung
  - Basis für Zugriff auf geschützte Objekte im TPM-Speicher
- Rücksetzen des TPM durch Jumper auf dem Board möglich
  - Löscht das Besitzer-Passwort
  - Verlust aller Schlüssel auf dem TPM

## 7 Roots of Trust

- "Vertrauenswurzeln" = Ausgangspunkte für den Aufbau einer vertrauenswürdigen Systemkonfiguration
  - den Roots of Trust muss vertraut werden - Fehlverhalten ist nicht aufdeckbar
- *Root of Trust for Measurement (RTM)*
  - Funktionen, die Integrität einer Konfiguration beim Booten, bei Reset und nach Suspend prüfen können
  - teilweise auf TPM, teilweise BIOS-Komponente (nicht ersetzbar!)
- *Root of Trust for Storage (RTS)*
  - schützt Schlüssel und vertrauenswürdige Daten
  - hierarchische Schlüsselverwaltung - Elternknoten verschlüsselt darunter liegende Schlüssel, Wurzel = Storage Root Key (SRK)
  - SRK wird bei Plattform-Inbesitznahme generiert und verlässt TPM nie
- *Root of Trust for Reporting (RTR)*

## 8 Trusted Building Blocks

- Komponenten und Verbindungswege, die sicherheitskritisch sind



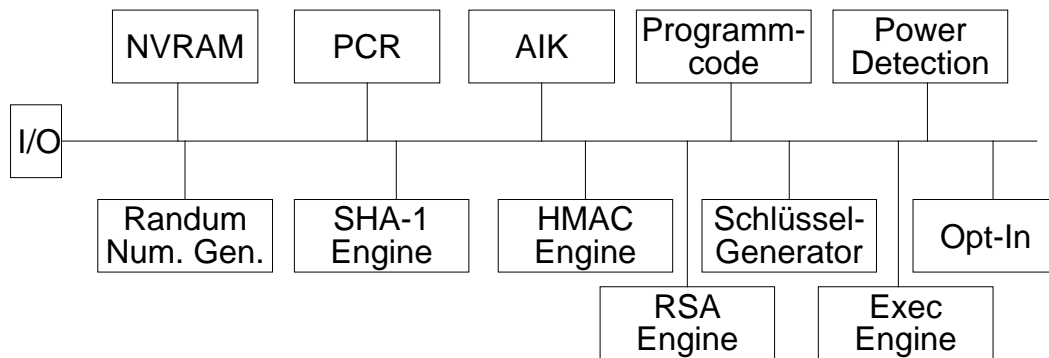
- ★ häufige Schwachstelle: Verbindung zwischen Tastatur und System

## 9 TPM im Detail

- Aufgaben
  - Generierung von symmetrischen und asymmetrischen Schlüsseln (Hardware-basierter Zufallszahlengenerator), Signaturerstellung, Hashwertberechnungen, Chip-interne Verschlüsselung
  - Verschlüsseln von kryptographischen Schlüsseln (Wrapping) (zur Verwaltung der Schlüssel außerhalb des TPM)
  - sichere Speicherung von (kleinen) Objekten und Hashwerten  
→ shielded Register, Zugriff nur über bestimmte Sicherheitsdienste (Signieren, Verschlüsseln)
  - Erstellen von signierten Reports über gespeicherte Werte (nur nach Autorisierung durch den Plattformbesitzer)
  - Endorsement-Key und Generierung/Bestätigung von AIKs
  - Vertrauenswürdiger Timer für Zertifikate mit Gültigkeitsdauer

## 9 TPM im Detail (2)

### ■ Architektur



## 10 Sicheres Booten

- normaler Bootvorgang:  
Power-on → ROM → BIOS → MBR → Boot-Block → BS-Kern → ...
- Sicherheitsprobleme
  - ◆ MBR (Master Boot Record) modifiziert  
(Boot-Sektor-Virus)
    - wird ungeschützt in den nackten Speicher geladen
  - ◆ Boot-Block manipuliert
    - kann modifiziertes Betriebssystem laden
    - Modifizierter Betriebssystemkern
      - kann überall zugreifen
      - kann Anwendungen falsche Tatsachen vorspiegeln
      - ...



## 10 Sicheres Booten

- sicheres Booten und TCG
  - Aufbau einer Vertrauenskette vom Einschalten bis zu den Anwendungen
- Power-on → CRTM
  - ◆ CRTM berechnet Hash-Werte und speichert sie in Register  $PCR_0$  auf TPM
    1.  $HASH(CRTM) \rightarrow PCR_0$  – bevor CRTM in Speicher geladen wird
    2.  $HASH(BIOS) \rightarrow PCR_1$
- BIOS laden und starten
  3.  $HASH(\text{Motherboard-Firmware} \mid PCR_1) \rightarrow PCR_1$
  4.  $HASH(\text{Hardware-Konfiguration} \mid PCR_1) \rightarrow PCR_1$
  5.  $HASH(\text{Option ROMS}) \rightarrow PCR_{2/3}$
  6.  $HASH(MBR) \rightarrow PCR_4$
- MBR laden
  7.  $HASH(\text{Boot-Block} \mid PCR_4) \rightarrow PCR_4$

## 10 Sicheres Booten (2)

- Boot-Loader ausführen
  8.  $HASH(\text{Betriebssystem-Kern} \mid PCR_4) \rightarrow PCR_4$
- Betriebssystem laden und ausführen
  - ◆ BS kann verschlüsselt auf Platte abgelegt sein
    - TPM übernimmt sichere Speicherung des Schlüssels und Entschlüsselung
    - BS kann Hashwerte über Anwendungen erstellen
- Referenzwerte für gemessene Hash-Werte müssen über vertrauenswürdigen Kanal zugeführt werden
  - Vergleich mit aktuellen Werten führt zur Erkennung von Manipulationen

## 11 Zertifikate (Credentials)

- Ziel: Attestieren der Identität einer Plattform oder eines AIK einer Plattform
- Fünf Zertifikattypen für unterschiedliche Anwendungsbereiche
  - es werden nur die jeweils benötigten Informationen bereitgestellt
- ◆ Endorsement (EK)
  - von TPM-Hersteller ausgestellt und signiert
  - erlaubt eindeutige Identifikation einer Plattform
- ◆ Conformance
  - bestätigt, dass TBB des TPM die Spezifikation erfüllen
- ◆ Platform
  - identifiziert Hersteller der Plattform und einige Eigenschaften der Plattform
- ◆ Validation
  - attestiert Vertrauenswürdigkeit von einzelnen Komponenten

## 11 Zertifikate (Credentials) (2)

- ◆ Attestation Identity
  - beglaubigt private AIK-Schlüssel, die zum Signieren von TPM-Werten verwendet werden können
- ➔ Entkopplung der Informationen über die TPM-Plattform von den Informationen über die Eigenschaften der Plattform
  - Nutzung der mit dem Endorsement-Zertifikat verbundenen Schlüssel gibt automatisch Informationen über die konkrete Plattform und damit über den Benutzer preis
  - AI-Zertifikate nutzen Trusted Third Party (Privacy CA) zur Anonymisierung

## 11 Zertifikate (Credentials) (3)

### ■ Erzeugung von AI-Credentials und AIK-Schlüsseln

- (1) Kommando TPM-MakeIdentity an TPM
  - erzeugt Schlüsselpaar (-AIK, +AIK)
  - erzeugt Identity-Binding und signiert dieses mit dem AIK → AIK-Request
 
$$AR = (+AIK, EK-Cred, Platform-Cred, Conformance-Cred)$$

$$C = \{AR\}^{-AIK}$$
- (2) Request an CA
 
$$TPM \rightarrow CA: \{(+AIK, C, EK-Cred, Platform-Cred, Conformance-Cred)\}^{+CA}$$
- (3) CA prüft Credentials, extrahiert AR, erzeugt Session-Schlüssel K und erstellt AIK-Credential
 
$$KC = \{K\}^{+EK} \quad C-Cred = \{AIK-Cred\}^K$$

$$CA \rightarrow TPM: (C-Cred, KC)$$
- (4) TPM kennt -EK und kann damit K und damit AIK-Cred entschlüsseln

## 11 Zertifikate (Credentials) (4)

### ■ Nutzung von AIK-Credentials bei Diensten

- ◆ AIK-Schlüssel können zum Signieren von PCR-Registerinhalten genutzt werden
- ◆ Ermittlung der PCR-Werte und Signieren laufen vollständig im TPM ab
  - ▶ wenn sichergestellt ist, dass der TPM vertrauenswürdig ist, ist der signierte PCR-Wert authentisch
  - ▶ CA kennt Zertifikate des TPM und kann bestätigen, dass ein AIK von einer Standard-konformen TPM-Plattform stammt

## 12 TCG für Digital Rights Management

- Idee: Software oder Daten (z. B. Musikstücke) sind mit Regeln verknüpft, die die Eigenschaften der Umgebung spezifizieren
  - Software läuft nur in bestimmter Umgebung
  - Daten können nur in bestimmter Umgebung genutzt werden
- ➔ Durchsetzung von Lizenzbestimmungen
- ➔ Schutz vor Raubkopien
- ▲ Konsequenz: private Rechner arbeiten unter der Policy eines Software- oder Content-Anbieters
- ▲ Verallgemeinerung:
  - ◆ Policy wird an Inhalte geknüpft um geistiges Eigentum zu schützen
    - Anwendungsszenarien in betrieblichen Geschäftsprozessen
    - Geschützter Austausch von Dokumenten (Schutz aber auch nur beschränkt möglich)