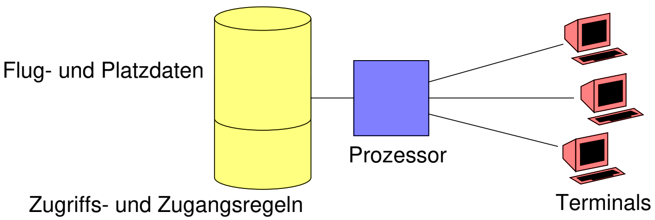
I Datensicherheit und Zugriffschutz

I Datensicherheit und Zugriffsschutz

1 Problemstellung

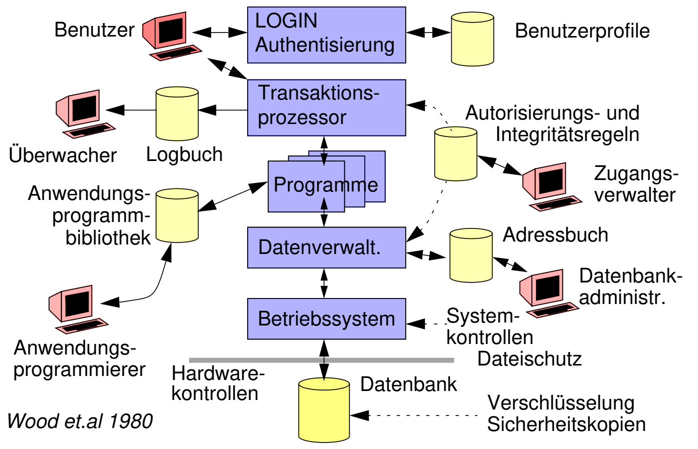
- Beispiel: Zugang zu einer Datenbank zur Flugreservierung und -buchung



  - Was sind mögliche Beeinträchtigungen der Datensicherheit?

1 Problemstellung (2)

- Überprüfungen beim Transaktionsbetrieb (Datenbankanwendung)



Nach Wood et.al 1980

1 Problemstellung (3)

- Illegaler Datenzugriff
  - Daten sind zugreifbar, die vertraulich behandelt werden sollen
- Illegales Löschen von Daten
  - Kein Zugriff, aber Daten werden gelöscht
- Illegales Manipulieren von Daten
  - Daten werden in böswilliger Absicht verändert
- Zerstörung von Rechensystemen
  - physisches Zerstören von Teilen der Rechenanlage

1.1 Umgebung der Rechenanlage

- Naturkatastrophen
  - Erdbeben, Vulkanausbrüche etc. können Rechenanlage und Datenbestand zerstören
- Unfälle
  - Gasexplosion, Kühlwasserlecks in der Klimaanlage oder Ähnliches zerstören Rechner und Daten
- Böswillige Angriffe
  - Zerstörung der Rechenanlage und des Datenbestands durch Sabotage (Bombenanschlag, Brandanschlag etc.)
- Unbefugter Zutritt zu den Räumen des Rechenzentrums
  - Diebstahl von Datenträgern
  - Zerstörung von Daten
  - Zugang zu vertraulichen Daten

1.2 Systemsoftware

- Versagen der Schutzmechanismen
  - System lässt Unbefugte auf Daten zugreifen oder Operationen ausführen
- Durchsickern von Informationen
  - Anwender können anhand scheinbar unauffälligen Systemverhaltens Rückschlüsse auf vertrauliche Daten ziehen *(Covert channels)*
- Beispiel: verschlüsselt abgespeicherte Passwörter sind zugänglich
  - Entschlüsselungsversuch der Passwörter außerhalb der Rechenanlage
  - "Wörterbuchattacke": Raten von Passwörtern möglich

1.3 Systemprogrammierer

- Umgehen oder Abschalten der Schutzmechanismen
- Installation eines unsicheren Systems
  - erlaubt dem Systemprogrammierer die Schutzmechanismen von außen zu umgehen
- Fehler beim Nutzen von Bibliotheksfunktionen innerhalb sicherheitskritischer Programme
  - S-Bit Programme unter UNIX laufen mit der Benutzerkennung des Dateibesitzers, nicht unter der des Aufrufers
    - Fehlerhafte S-Bit Programme können zur Ausführung von Code unter einer fremden Benutzerkennung gebracht werden
    - S-Bit Programme mit "root-Rechten" besonders gefährlich

1.3 Systemprogrammierer (2)

- Buffer-Overflow Fehler bei Funktionen
  `gets(), strcpy(), strcat(), sprintf( "%s",..)`
  - Zu lange Eingaben überschreiben den Stackspeicher des Prozessors
  - Mit genauer Kenntnis ist das Ausführen beliebigen Codes erzwingbar
- Fehlerhafte Parameterprüfung beim Aufruf von Funktionen
  `system()`
- Beispiel: Löschen einer Datei
  - Name der Datei wurde in Variable **`file`** eingelesen
  - Aufruf von **`system`** mit Parameter **`strcat( "rm ", file)`**
  - Gibt man für den Dateinamen den String
    `"fn ; xterm -display myhost:0 &"`
    ein, bekommt man ein Fenster auf der aufgebrochenen Maschine

1.4 Rechnerhardware

- Versagen der Schutzmechanismen
  - erlauben nicht-autorisierten Zugriff
- Fehlerhaft Befehlsausführung
  - Zerstörung von wichtigen Daten
- Abstrahlungen
  - erlaubt Ausspähen von Daten

1.5 Datenbasis

- Falsche Zugriffsregeln
  - erlauben nicht-autorisierten Zugriff

1.6 Operateur

- Kopieren vertraulicher Datenträger
- Diebstahl von Datenträgern
- Initialisierung mit unsicherem Zustand
  - Operateur schaltet beispielsweise Zugriffskontrolle ab
- Nachlässige Rechnerwartung
  - Nachbesserungen der Systemsoftware *(Patches)* werden nicht eingespielt
    - Sicherheitslücken werden nicht gestopft

1.7 Sicherheitsbeauftragter

- Fehlerhafte Spezifikation der Sicherheitspolitik
  - dadurch Zugang für Unbefugte zu vertraulichen Daten oder
  - Änderungen von Daten durch Unbefugte möglich
- Unterlassene Auswertung von Protokolldateien
  - Einbrüche und mögliche Sicherheitslücken werden nicht rechtzeitig entdeckt

1.8 Kommunikationssystem

- Abhören der Kommunikationsleitungen *(Sniffing)*
  - z.B. Telefonverbindung bei Modemnutzung oder serielle Schnittstellen
  - z.B. Netzwerkverkehr auf einem Netzwerkstrang

  - Ermitteln von Passwörtern und Benutzerkennungen
    - manche Dienste übertragen Passwörter im Klartext (z.B. ftp, telnet, rlogin)

  - Zugriff auf vertrauliche Daten

  - unbefugte Datenveränderungen
    - Verfälschen von Daten
    - Übernehmen von bestehenden Verbindungen *(Hijacking)*
    - Vorspiegeln falscher Netzwerkadressen *(Spoofing)*

1.8 Kommunikationssystem (2)

- Illegale Nutzung von Diensten über das Netzwerk
  - Standardsysteme bieten eine Menge von Diensten an (z.B. ftp, telnet, rwho u.a.)
  - Sicherheitslücken von Diensten werden publik gemacht und sind auch von "dummen" Hackern nutzbar *(Exploit scripts)* http://www.rootshell.org
  - Auch bei temporär am Netzwerk angeschlossenen Computern eine Gefahr
    - z.B. Linux-Maschine mit PPP-Verbindung an das Uni-Netz
    - Voreinstellungen der Standardinstallation meist unsicher

1.9 Terminal

- Ungeschützter Zugang zum Terminal
  - Nutzen einer fremden Benutzerkennung
  - Zugriff auf vertrauliche Daten
  - unbefugte Datenveränderungen

1.10 Benutzer

- Nutzen anderer Kennungen
  - erlauben nicht-autorisierten Zugriff
  - unbefugte Datenveränderungen
  - unbefugte Weitergabe von Informationen
- Einbruch von Innen
  - leichterer Zugang zu möglichen Sicherheitslöchern (z.B. bei Diensten)

1.11 Anwendungsprogrammierung

- Nichteinhalten der Spezifikation
  - Umgehen der Zugriffskontrollen
- Einfügen von „bösartigen“ Befehlsfolgen
  - *Back door:* Hintertür gibt dem Programmierer im Betrieb Zugang zu vertraulichen Daten oder illegalen Operationen
  - *Trojan horse:* Unter bestimmten Bedingungen werden illegale Operationen ohne Trigger von außen angestoßen

1.12 "Tracker Queries"

- Beispiel: Datenbanksysteme
  - Zugriff auf Einzelinformationen ist verboten (Vertraulichkeit)
  - statistische Informationen sind erlaubt
- Grenzen möglicher Sicherheitsmaßnahmen:
  Zugriff auf Einzelinformationen dennoch möglich
  - geeignete Anfragen kombinieren *(Tracker queries)*
- Beispiel: Gehaltsdatenbank

1.12 "Tracker Queries" (2)

- Tabelle der Datenbankeinträge:

Nr.	Name	Geschl.	Fach	Stellung	Gehalt	Spenden
1	Albrecht	m	Inf.	Prof.	60.000	150
2	Bergner	m	Math.	Prof.	45.000	300
3	Cäsar	w	Math.	Prof.	75.000	600
4	David	w	Inf.	Prof.	45.000	150
4	Engel	m	Stat.	Prof.	54.000	0
5	Frech	w	Stat.	Prof.	66.000	450
6	Groß	m	Inf.	Angest.	30.000	60
8	Hausner	m	Math.	Prof.	54.000	1500
9	Ibel	w	Inf.	Stud.	9.000	30
10	Jost	m	Stat.	Angest.	60.000	45
11	Knapp	w	Math.	Prof.	75.000	300
12	Ludwig	m	Inf.	Stud.	9.000	0

1.12 "Tracker Queries" (3)

- Anfragen und Antworten:
  - Anzahl('w'): 5
  - Anzahl('w' und (nicht 'Inf' oder nicht 'Prof.')): 4
  - mittlere Spende('w'): 306
  - mittlere Spende('w' und (nicht 'Inf.' oder nicht 'Prof.')): 345
- Berechnung:
  - Spende('David'): 306 * 5 - 345 * 4 =
    1530 - 1380 =
    150

2 Zugriffslisten

- Identifikation von Subjekten, Objekten und Berechtigungen
  - Subjekt: Person oder Benutzerkennung im System
    (repräsentiert jemanden, der Aktionen ausführen kann)
  - Objekt: Komponente des Systems
    (repräsentiert Ziel einer Aktion)
  - Berechtigung: z.B. Leseberechtigung auf einer Datei
    (repräsentiert die Erlaubnis für die Ausführung einer Aktion)
- Erfassung der Berechtigungen in einer Subjekt-Objekt-Matrix:
  Zugriffsliste *(Access control list, ACL)*

2.1 Beispiel für Zugriffslisten

- Personaldatensatz
  - besteht aus: Name, Abteilung, Personalnummer, Lohn- oder Gehaltsgruppe
- Personaldateien (Objekte)
  - D_{LA}: Personaldaten der leitenden Angestellten
  - D_{AN}: Personaldaten der sonstigen Angestellten
  - D_{AR}: Personaldaten der Arbeiter
- Prozeduren (gehören zu den Aktionen)
  - R_{LA}: Lesen von Pers.-Nr. und Lohn-/Gehaltsgr. aus D_{LA}
  - $R_{AN/AR}$: Lesen von Pers.-Nr. und Lohn-/Gehaltsgr. aus D_{AN} oder D_{AR}
  - R_{post}: Lesen von Name, Abteilung und Pers.-Nr.

2.1 Beispiel für Zugriffslisten (2)

- Benutzer (Subjekte)
  - S_{pers}: Leiter des Personalbüros
    - Besitzer aller Dateien und Prozeduren
    - Lese- und Schreibrecht für alleDateien
    - Aufrufrecht für alle Prozeduren
  - S_{stellv}: Sachbearb. leitende Angestellte, stellvertr. Leiter Personalbüro
    - Lese- und Schreibrecht für D_{AN} und D_{AR}
    - Aufrufrecht für R_{LA}
  - S_{sach}: Sachbearbeiter Angestellte u. Arbeiter
    - Aufrufrecht für $R_{AN/AR}$
  - S_{post}: Poststelle
    - Aufrufrecht für R_{post} auf alle Dateien

2.1 Beispiel für Zugriffslisten (3)

- Berechtigungen werden in Matrix ausgedrückt:

	D_{LA}	D_{AN}	D_{AR}	R_{LA}	$R_{AN/AR}$	R_{post}
S_{pers}	O, R, W	O, R, W	O, R, W	O, I	O, I	O, I
S_{stellv}		R, W	R, W	I		
S_{sach}					I	
S_{post}						I
R_{LA}	R					
$R_{AN/AR}$		R	R			
R_{post}	R	R	R			

  - O = *Owner;* Besitzer der Datei oder Prozedur
  - R = *Read;* volle Leseberechtigung
  - W = *Write;* volle Schreibberechtigung
  - I = *Invoke;* Aufrufberechtigung

2.2 Beispiel: UNIX

- Zugriffslisten für
  - Dateien und Geräte
  - Shared-Memory-Segmente
  - Message-Queues
  - Semaphore
  - etc.
- Berechtigungen:
  - Lesen *(read)*, Schreiben *(write)*, Ausführen *(execute)*
  - für Besitzer, Gruppe und alle anderen unterscheidbar
- Subjekte:
  - Prozesse
  - Besitzer (Benutzer) und Zugehörigkeit zu einer oder mehreren Gruppen

2.2 Beispiel: UNIX

- Superuser
  - Benutzer *root* hat automatisch alle Zugriffsrechte
- S-Bit–Programme
  - S-Bit ist ein besonderes Recht auf der Binärdatei des Programms
  - Besitzer der Datei wird bei der Ausführung auch Besitzer des Prozess (sonst wird Aufrufer Besitzer des Prozess)
- ★ Vorteil
  - Bereitstellen von Prozessen, die kontrolliert Aufrufern höhere Zugriffsberechtigungen erlauben
- ▲ Nachteil
  - Fehler im Prozess gibt Aufrufer volle Rechte des Programmbesitzers
  - fatal, falls das Programm *root* gehört

2.3 Implementierung

- Globale Tabelle/Matrix
  - System hält eine Datenstruktur und prüft im betreffenden Eintrag die Berechtigungen
  - Tabelle üblicherweise recht groß: passt evtl. nicht in den Speicher
- Zugriffslisten an den Objekten
  - jedes Objekt hält eine Liste der Berechtigungen (z.B. Unix Datei: Inode)
  - verringert üblicherweise den Platzbedarf für die Einträge (unnötige Felder der Matrix werden nicht repräsentiert)
- Zugriffslisten an den Subjekte
  - jedes Subjekt hält eine Liste von Objekten und den Berechtigungen, die das Subjekt für das Objekt hat
  - ähnlich Capabilities

3 Schutz durch Speicherverwaltung

- Schutz vor gegenseitigem Speicherzugriff
  - Segmentierung und Seitenadressierung erlauben es, jedem Prozess nur den benötigten Speicher einzublenden
  - Segmentverletzung löst Unterbrechung aus
- Systemaufrufe
  - definierter Weg von einer Schutzumgebung (der des Prozesses) in eine andere (der des Betriebssystems)
- Erweiterung dieses Konzepts:
  - allgemeine Prozeduraufrufe zwischen verschiedenen Schutzumgebungen, realisiert mit der Speicherverwaltung und deren Hardware (MMU)

3.1 Modulkonzept von Habermann

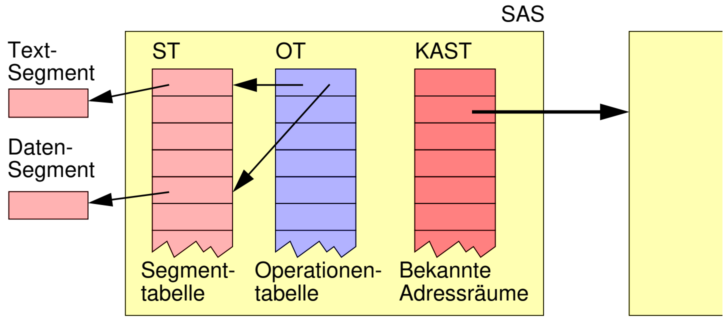
- Idee (von 1976)
  - Adressräume (Module) bilden Schutzumgebungen
  - Adressräume bieten definierte Operationen an (ähnlich wie das Betriebssystem Systemaufrufe anbietet)
  - Parameter werden in speziellen Segmenten übergeben

★ Bietet allgemeinen Schutz der Module und erlaubt kontrollierte Interaktionen

- Module besitzen einen statischen Adressraum *(SAS, Static address space)*
  - enthält Liste von Segmenten, die zu dem Modul gehören bzw. von dem Modul zugegriffen werden dürfen
  - enthält Liste von angebotenen Operationen mit den Angaben, welche Segmente jede Operation benötigt (u.a. Segment für die auszuführenden Instruktionen)

3.1 Modulkonzept von Habermann (2)

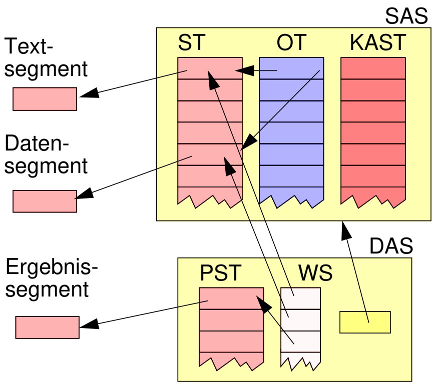
- enthält Liste von bekannten Adressräumen anderer Module (dort können dann Operationen aufgerufen werden)



KAST = *Known address space table*

3.1 Modulkonzept nach Habermann (3)

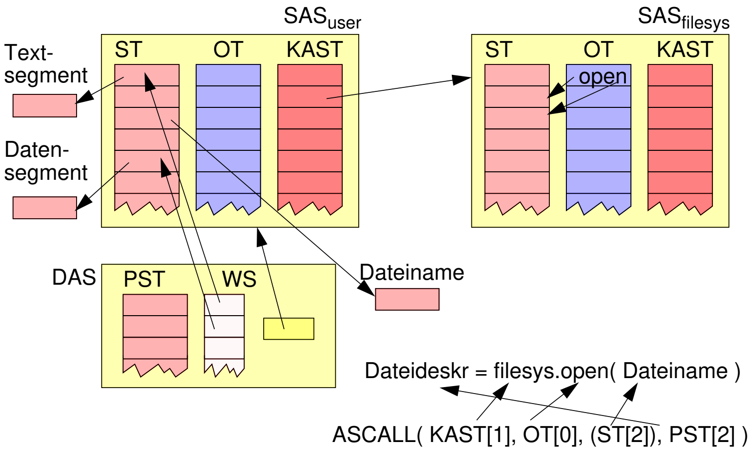
- Aktivitätsträger sind einen dynamische Adressraum zugeordnet *(DAS, Dynamic address space)*



- ◆ enthält Segmenttabelle für Parameter und Ergebnisse *(PST)*
- ◆ enthält Verweise auf Segmenttabellen, die den Adressraum zusammenstellen *(WS= Working space)*

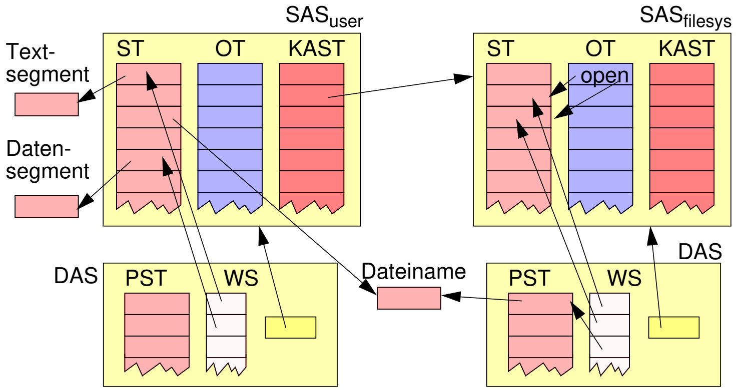
3.2 Beispielaufruf

- SAS des Benutzers ruft Operation „open“ des Dateisystem-SAS auf



3.2 Beispielaufruf (2)

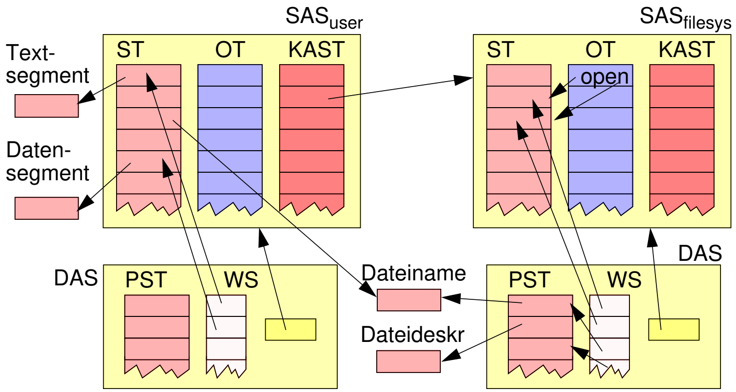
- SAS des Benutzers ruft Operation „open“ des Dateisystem-SAS auf



- ◆ für den Aufruf von „open“ wird ein neuer DAS erzeugt

3.2 Beispielaufruf (3)

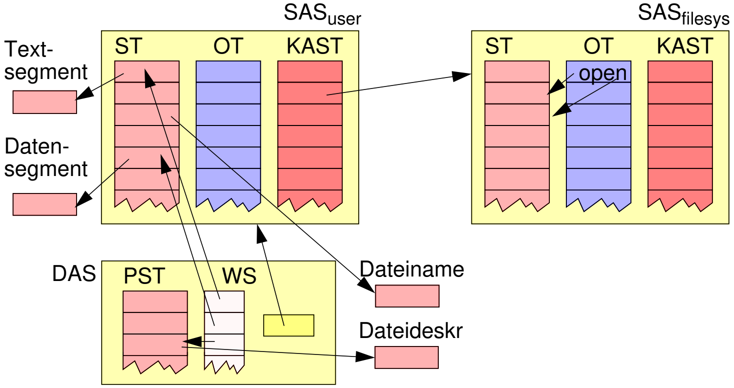
- SAS des Benutzers ruft Operation „open“ des Dateisystem-SAS auf



- ◆ „open“ erzeugt neues Segment für den Dateideskriptor

3.2 Beispielaufruf (4)

- SAS des Benutzers ruft Operation „open“ des Dateisystem-SAS auf



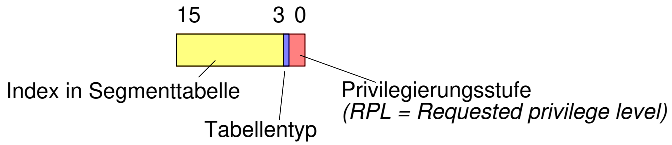
◆ Ergebnissegment wird an den Aufrufer zurückgegeben

3.3 Beispiel: Pentium

- Privilegierungsstufen
  - ◆ Stufe 0: höchste Privilegien (privilegierte Befehle, etc.): BS Kern
  - ◆ Stufe 1: BS Treiber
  - ◆ Stufe 2: BS Erweiterungen
  - ◆ Stufe 3: Benutzerprogramme
- Merke:
  - ◆ kleine Stufennummer: hohe Privilegien
  - ◆ große Stufennummer: kleine Privilegien

3.3 Beispiel: Pentium (2)

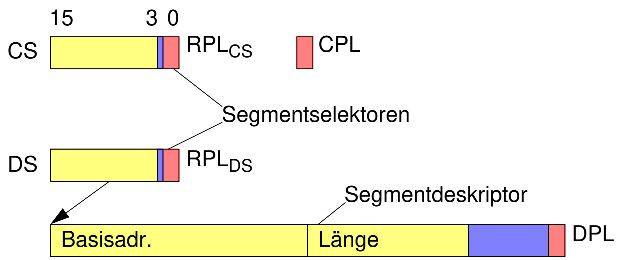
- Segmentselektoren enthalten Privilegierungsstufe



- Codesegmentselektor (CS) bestimmt aktuelle Privilegierungsstufe
  - ◆ CPL = *Current privilege level*

3.3 Beispiel: Pentium (3)

- Datenzugriff (z.B. auf Datensegment DS)



- ◆ DPL = *Descriptor privilege level*
- ◆ CPL ist normalerweise gleich RPL_{CS}
- ◆ Zugriff wird erlaubt, wenn: $DPL \geq max(CPL, RPL_{DS})$
- ◆ ansonsten wird Unterbrechung ausgelöst (Schutzverletzung)

3.3 Beispiel: Pentium (4)

- Erläuterung:
  - DPL < CPL: *Schutzverletzung*
    augenblickliche Privilegierungsstufe hat weniger Privilegien als im Deskriptor verlangt
    (CPL hat höhere Stufennr. als der Deskriptor)
  - DPL ≥ CPL: OK
    augenblickliche Privilegierungsstufe ist mindestens so hoch wie die im Deskriptor

★ Segment kann nur angesprochen werden, wenn augenblickliche Stufe die gleichen oder mehr Privilegien beinhaltet.

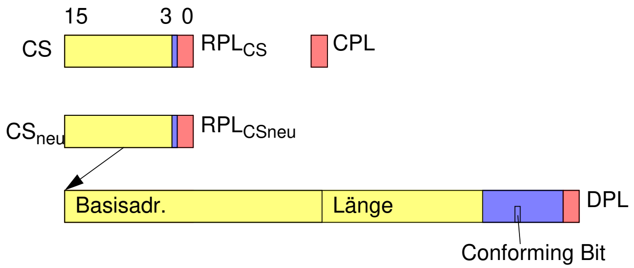
3.3 Beispiel: Pentium (5)

- Erläuterung:
  - DPL < RPL(ds): *Schutzverletzung*
    Selektor hat weniger Privilegien als der Deskriptor
    (Selektor hat größere Stufe als der Deskriptor)
  - DPL ≥ RPL(ds): OK
    Selektor hat mindestens die gleiche Privilegierungsstufe wie der Deskriptor

★ Selektor darf keine geringeren Privilegien versprechen als wirklich verlangt sind.

3.3 Beispiel: Pentium (5)

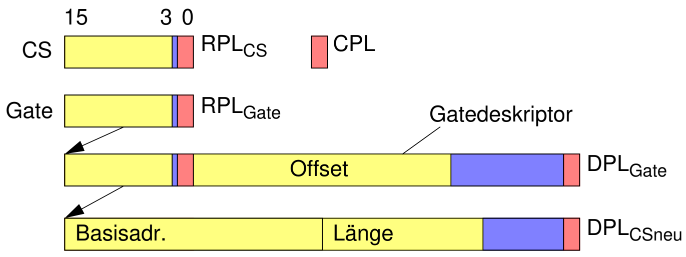
- Sprünge in andere Codesegmente *(Far Call)*



- Sprung wird erlaubt, falls: DPL = CPL oder
  Conforming Bit gesetzt und DPL ≤ CPL
- Im Falle von DPL ≤ CPL wird jedoch CPL nicht geändert
  (Codesegment hat höheres Privileg, CPL bleibt aber unverändert)

3.3 Beispiel: Pentium (6)

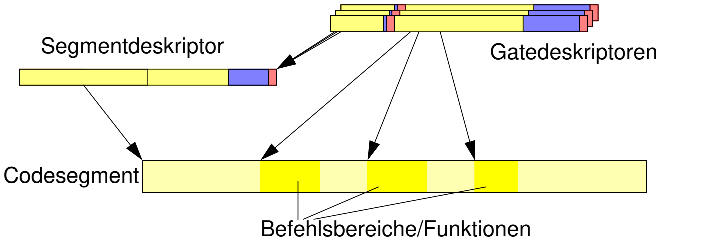
- Kontrolltransfer mit einem *Gate*



- Gatedeskriptoren stehen wie Segmentdeskriptoren in der Segmenttabelle
- Gatedeskriptor enthält Segmentselektor für das Codesegment und einen Offset zu diesem Segment, an dem der Einsprungpunkt liegt
- Kontrolltransfer (*CALL* Aufruf) wird erlaubt, falls:
  $DPL_{Gate} \geq \max(CPL, RPL_{Gate})$ und $DPL_{CSneu} \leq CPL$

3.3 Beispiel: Pentium (7)

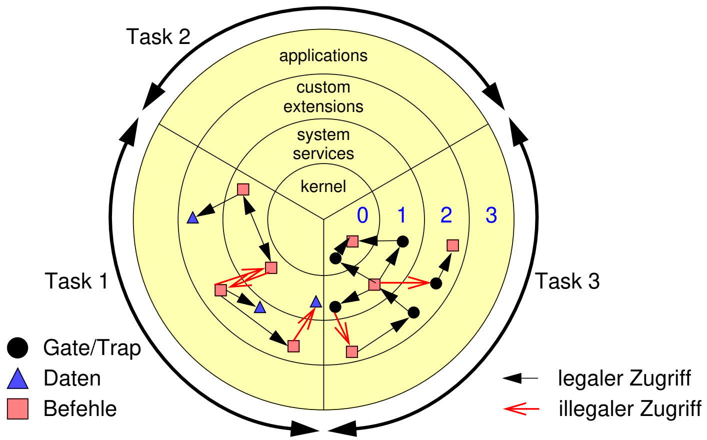
- Gates erlauben den kontrollierten Sprung in ein privilegierten Befehlsbereich



  - es existiert auch der entsprechende Rücksprung
  - für jede Privilegierungsstufe gibt es einen eigenen Stack; dieser wird mit umgeschaltet
  - Parameter werden automatisch auf den neuen Stack kopiert (Anzahl wird im Gatedeskriptor vermerkt)

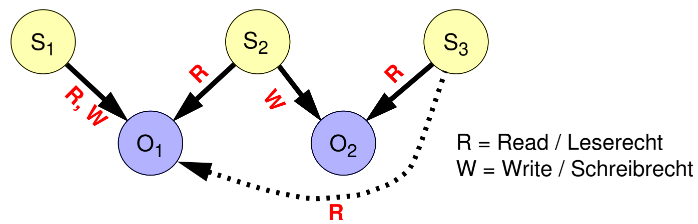
3.3 Beispiel: Pentium (8)

- Beispiel eines realisierten Schutzsystems



4 Capability-basierte Systeme

- 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_1 von S_2 nach S_3)

4 Capability-basierte Systeme (2)

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

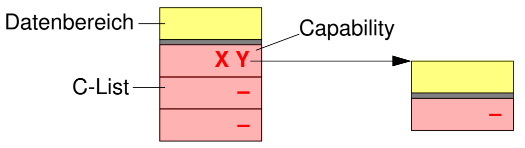
▲ Nachteile
  - Weitergabe nicht kontrollierbar
  - Rückruf von Zugriffsrechten nicht möglich
  - Capability muss vor Fälschung und Verfälschung geschützt werden (z.B. durch kryptographische Mittel oder durch Speicherverwaltung)

4.1 Beispiel: Hydra

- Hydra ist ein Capability-basiertes Betriebssystem
  - entwickelt Mitte der Siebziger Jahre an der Carnegie-Mellon University
  - lief auf einem speziellen Multiprozessor namens **C.mmp**
  - Capability-Mechanismen sind integraler Bestandteil des Betriebssystems
- Objekte in Hydra werden durch Capabilities angesprochen und geschützt
  - Objekte haben einen Typ
    (z.B. Prozeduren, Prozesse/LNS, Semaphoren, Datei etc.)
  - Capabilities haben entsprechenden Typ
  - benutzerdefinierte Typen sind möglich

4.1 Beispiel: Hydra (2)

- 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 (implementiert durch geschütztes Segment)



4.1 Beispiel: Hydra (3)

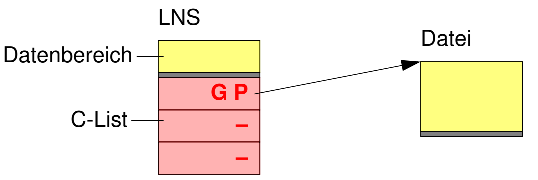
- Prozesse (Subjekte)
  - Prozesse besitzen einen aktuellen Kontext, den LNS *(Local name space)*
  - LNS ist ein Objekt
  - zum LNS gehört einen Aktivitätsträger (Thread)
  - LNS kann nur auf Objekte zugreifen, die in seiner C-List stehen (mehrstufige Zugriffe, z.B. auf die C-List eines Objekts, dessen 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

4.2 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:
  - *GETRTS:* erlaubt den Aufruf von *Getdata*
  - *PUTRTS:* erlaubt den Aufruf von *Putdata*
  - *ADDRTS:* erlaubt den Aufruf von *Adddata*
- Rechte müssen in der Capability zum Objekt gesetzt sein

4.2 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

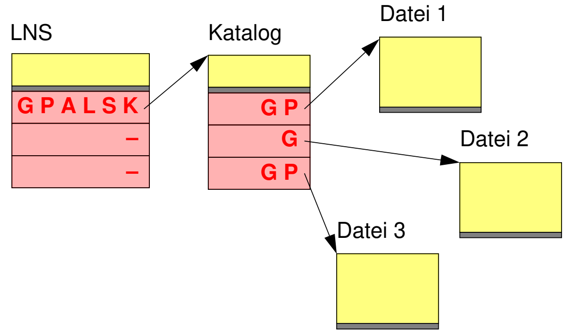


4.3 Zugriff auf Capabilities in Hydra

- Operationen auf der C-List:
  - *Load:* kopieren einer Capability aus der C-List eines Objekts in die C-List des LNS
  - *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*

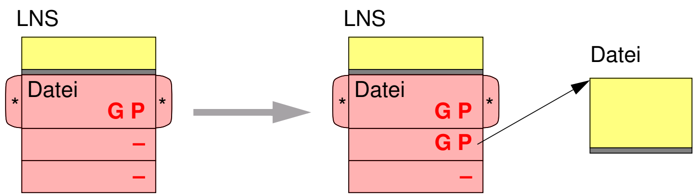
4.3 Zugriff auf Capabilities in Hydra (2)

- Beispiel: Implementierung von Katalogen
  - *Load* erlaubt das Auflösen von Namen (Aufrufer bekommt die Capability)
  - *Store* und *Append* erlauben das Hinzufügen von Dateien zum Katalog
  - *Delete* erlaubt das Austragen von Dateien aus dem Katalog



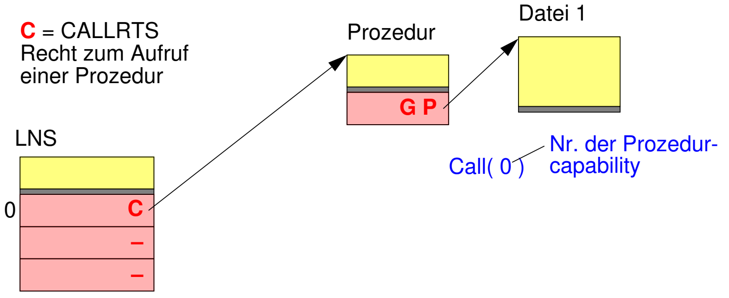
4.4 Objekterzeugung in Hydra

- Objekterzeugung über Erzeugungsschablonen *(Creation Templates)*
  - Erzeugungsschablone enthält den Typ des neu zu erzeugenden Objektes und eine Rechtemaske
  - nur die in der Maske angeschalteten Rechte werden dem Aufrufer in einer neuen Capability gegeben



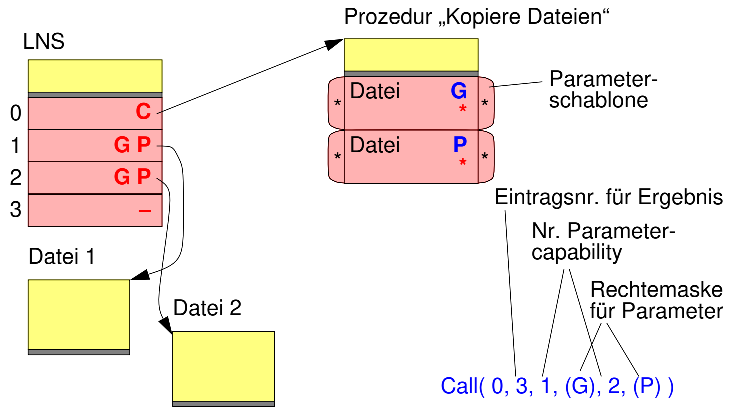
4.5 Prozeduraufruf in Hydra

- Prozedur ist ein Objekt, aus dem beim Aufruf ein LNS des laufenden Prozesses erzeugt wird
  - ◆ neuer LNS wird aktueller Kontext (alte LNS stehen auf einem Stack; sie werden wieder aktiviert, wenn Prozedur zu Ende)
- Aufruf einer Prozedur



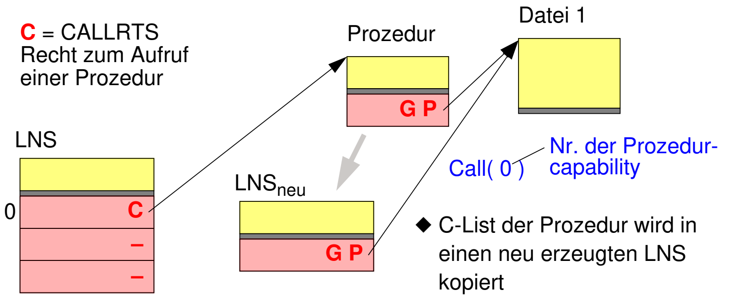
4.5 Prozeduraufruf in Hydra (2)

- Übergabe von Parametern
  - ◆ Beispiel: Prozedur zum Kopieren von Dateiinhalten



4.5 Prozeduraufruf in Hydra

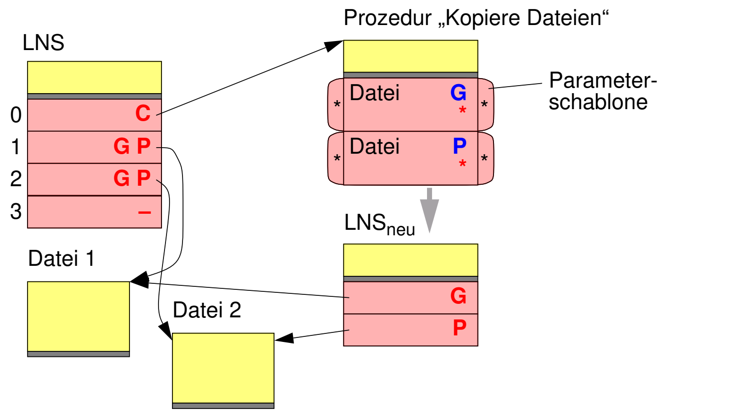
- Prozedur ist ein Objekt, aus dem beim Aufruf ein LNS des laufenden Prozesses erzeugt wird
  - ◆ neuer LNS wird aktueller Kontext (alte LNS stehen auf einem Stack; sie werden wieder aktiviert, wenn Prozedur zu Ende)
- Aufruf einer Prozedur



- ◆ C-List der Prozedur wird in einen neu erzeugten LNS kopiert

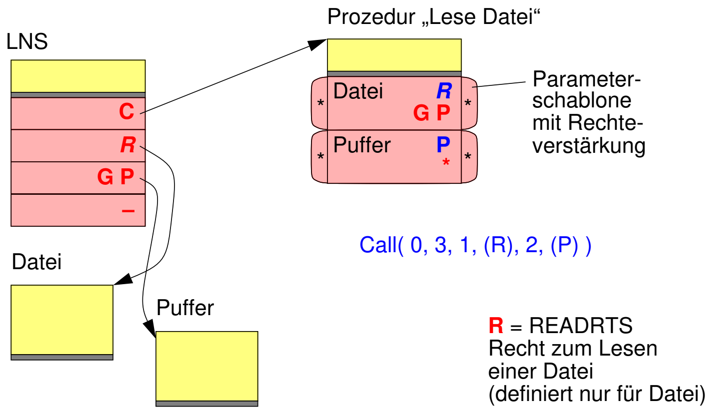
4.5 Prozeduraufruf in Hydra (3)

- Übergabe von Parametern
  - ◆ Beispiel: Prozedur zum Kopieren von Dateiinhalten



4.5 Prozeduraufruf in Hydra (3)

- Verstärken von Rechten
  - Beispiel: Prozedur zum Lesen von Dateiinhalten in einen Puffer

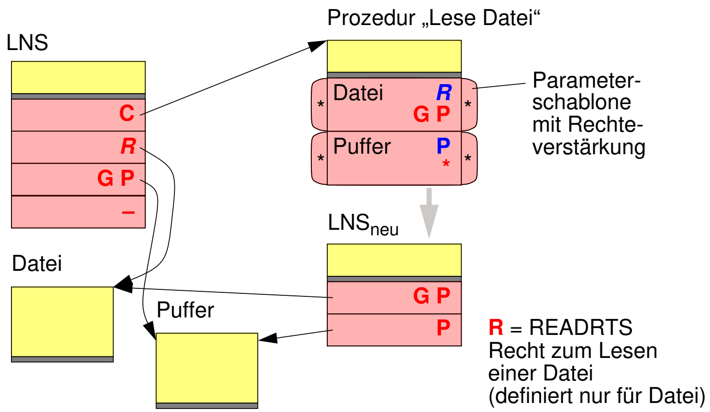


4.6 Problem: Gegenseitiges Misstrauen

- 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

4.5 Prozeduraufruf in Hydra (3)

- Verstärken von Rechten
  - Beispiel: Prozedur zum Lesen von Dateiinhalten in einen Puffer



4.6 Problem: Gegenseitiges Misstrauen (2)

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

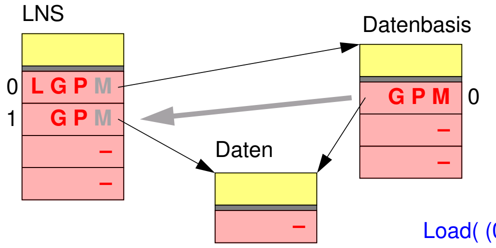
4.7 Problem: Modifikationen

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

4.7 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



es ist nicht möglich für den LNS, eine Capability auf die Daten zu erlangen, die Modifikationen zulässt

Load((0, 0), 1)

4.7 Problem: Modifikation (3)

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

4.8 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

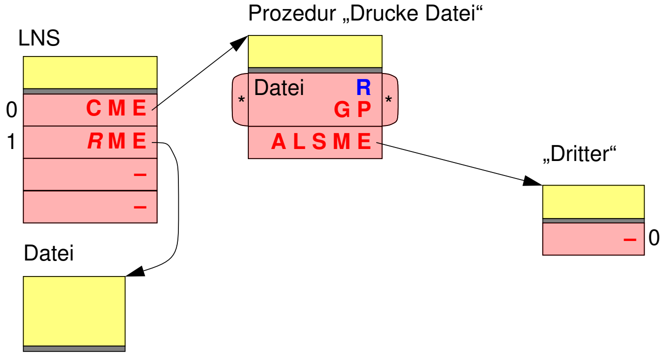
4.8 Problem: Ausbreitung von Capabilities (2)

★ 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

4.8 Problem: Ausbreitung von Capabilities (3)

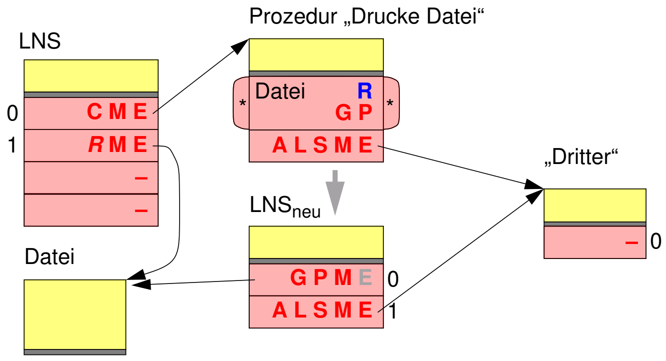
- Versuchte Weitergabe einer Capability an einen Dritten



Call(0, 2, 1, (–E))

4.8 Problem: Ausbreitung von Capabilities (3)

- Versuchte Weitergabe einer Capability an einen Dritten



Call(0, 2, 1, (–E))

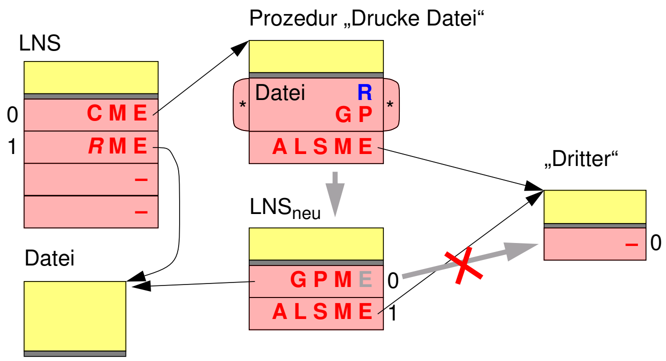
4.9 Problem: Aufbewahrung von Capabilities

- 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

4.8 Problem: Ausbreitung von Capabilities (3)

- Versuchte Weitergabe einer Capability an einen Dritten



Call(0, 2, 1, (–E)) Store(0, (1, 0))

4.10 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

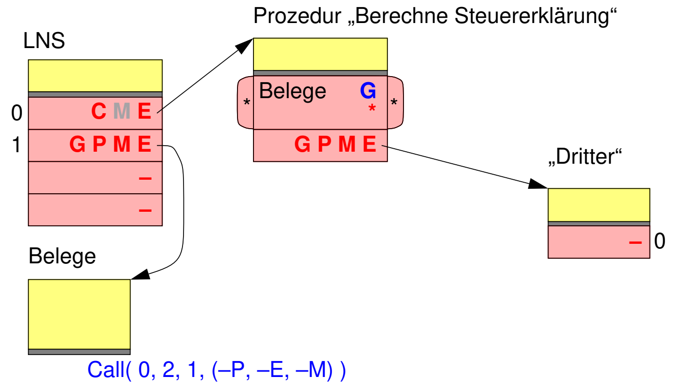
  - ◆ ENVRTS ist nicht ausreichend, da Prozedur den Dateninhalt von Parameterobjekten kopieren könnte
(ENVRTS wirkt nur auf die Weitergabe von Capabilities)

- 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)

4.10 Problem: Informationsflussbegrenzung (2)

- Beispiel: Prozedur zur Steuerberechnung
  - die übergebenen Beleg- und Buchhaltungsdaten sollen nicht weitergegeben werden können

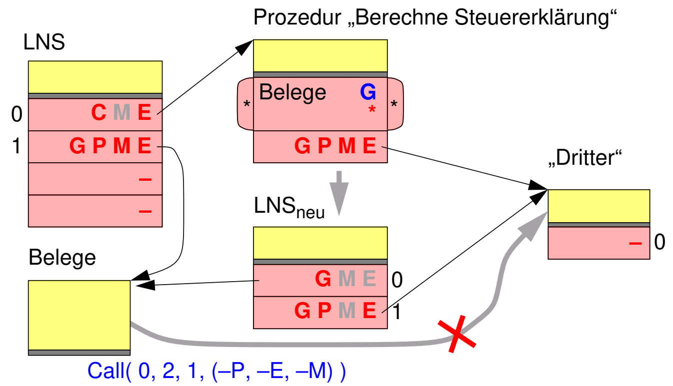


4.11 Problem: Initialisierung

- 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

4.10 Problem: Informationsflussbegrenzung (2)

- Beispiel: Prozedur zur Steuerberechnung
  - die übergebenen Beleg- und Buchhaltungsdaten sollen nicht weitergegeben werden können

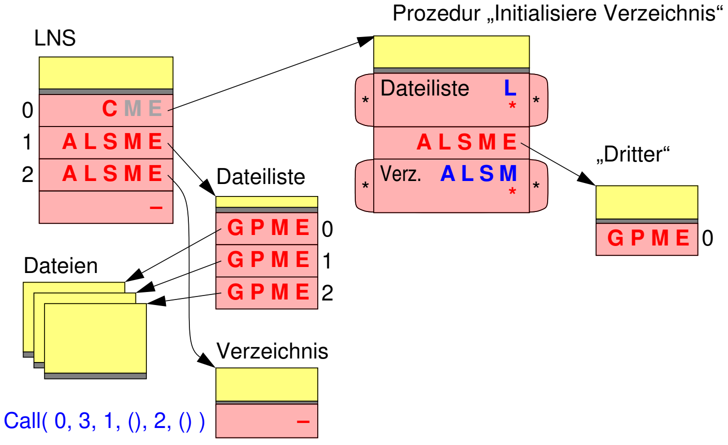


4.11 Problem Initialisierung (2)

★ 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

4.11 Problem: Initialisierung (3)

- Beispiel: Initialisierung eines Verzeichnis



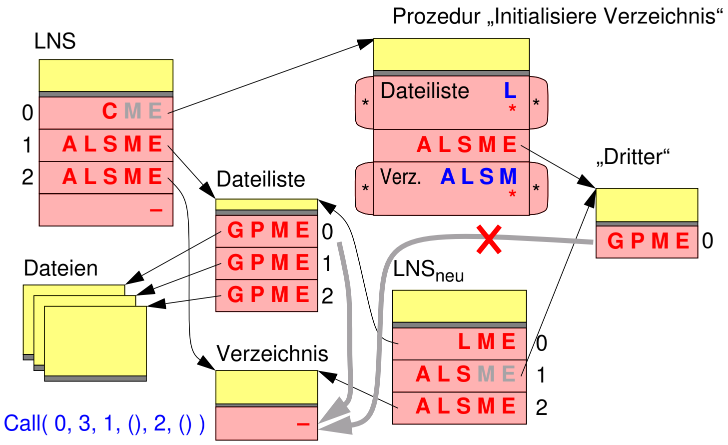
4.12 Rückruf von Capabilities

- 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 ***AL**LYRTS*
    (engl. *ally* = verbünden)

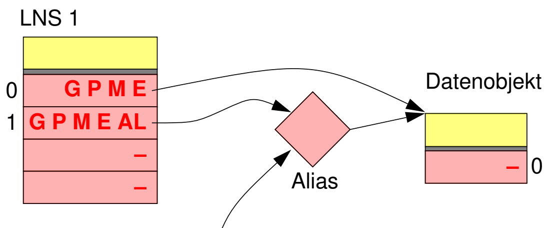
4.11 Problem: Initialisierung (3)

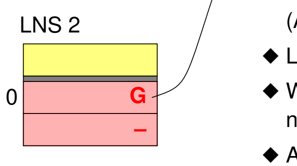
- Beispiel: Initialisierung eines Verzeichnis



4.12 Rückruf von Capabilities (2)

- Beispiel: Weitergabe einer rückrufbaren Capability

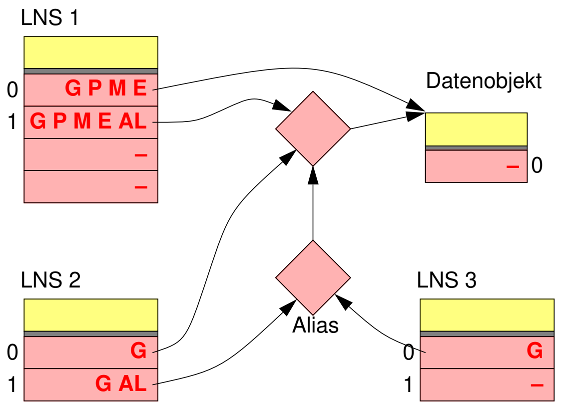




  - LNS 1 kann Verbindung des Alias lösen (ALLYRTS nötig)
  - LNS 2 kann dann nicht mehr zugreifen
  - Wiederverbindung möglich (Originalcapability nötig)
  - Aliase können hintereinander auftreten

4.12 Rückruf von Capabilities (3)

- Beispiel: Aliasketten



4.12 Rückruf von Capabilities (4)

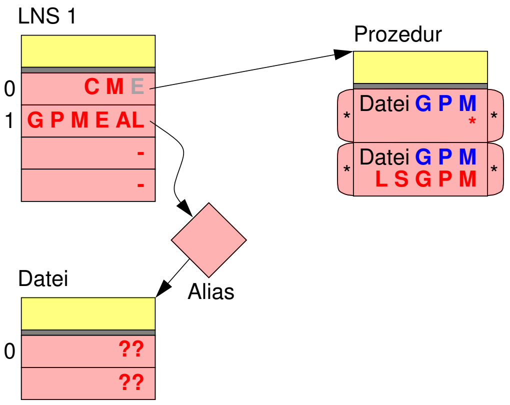
▲ Problem: Rückruf während der Bearbeitung eines Objekts
- ◆ inkonsistente Zustände möglich

★ Lösung in Hydra
- ◆ Parameter-Capabilities, die durch eine rechteverstärkende Parameterschablone angenommen werden, zeigen auf das Originalobjekt

▲ Nachteil
- ◆ nicht vertrauenswürdige Prozeduren können rückruffreie Capability erlangen
- ◆ Problem fällt in die selbe Kategorie wie rechteverstärkende Parameterschablonen an sich

4.12 Rückruf von Capabilities (5)

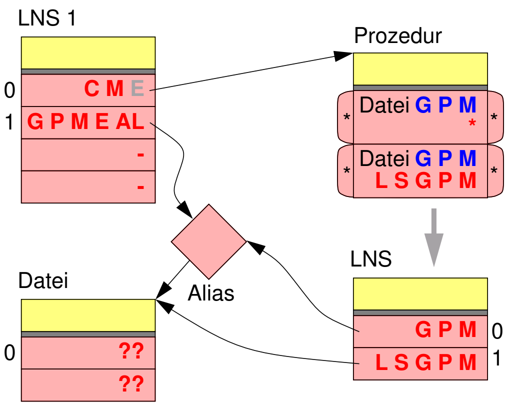
- Beispiel:



Call(0, 2, 1, (–E), 1, (–E))

4.12 Rückruf von Capabilities (5)

- Beispiel:



Call(0, 2, 1, (–E), 1, (–E))

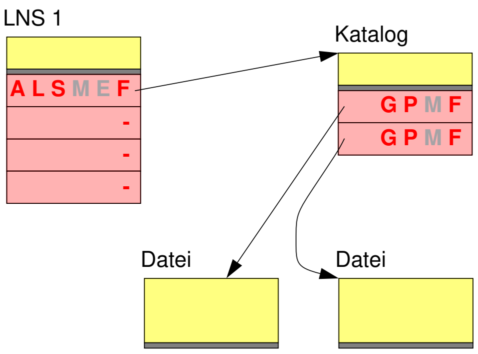
4.13 Garantierter Zugriff

- 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 **F**RZRTS)* ein
  - Einfrieren nimmt Modifikationsrecht weg
  - ein Objekt kann nur gefroren werden, wenn alle Capabilities in der C-List bereits das Einfrierrecht haben

4.13 Garantierter Zugriff (2)

- Beispiel:



4.14 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

5 Kryptographische Maßnahmen

- Verschlüsseln und Entschlüsseln vertraulicher Daten
  - aus den verschlüsselten Daten soll die Originalinformation nur mit Hilfe eines Schlüssels restauriert werden können
  - Schlüssel bleibt geheim
- Authentisierung
  - Empfänger kann verifizieren, wer der Absender ist
- Sicherer Kanal
  - gesendete Informationen können nicht gefälscht und verfälscht werden
  - nur der adressierte Empfänger kann die Informationen lesen
  - Empfänger kann den Absender authentisieren

5 Kryptographische Maßnahmen (2)

- Funktionen
  - Verschlüsselungsfunktion *E (encrypt)*: $E(K_1, T) \rightarrow C$
  - Entschlüsselungsfunktion *D (decrypt)*: $D(K_2, C) \rightarrow T$
  - K = Schlüssel, T= zu verschlüsselnder Text/Daten
- Verwandte Schlüssel
  - es gilt: K_1 und K_2 sind verwandt, wenn gilt: $\forall T: D(K_2, E(K_1, T)) = T$
- Symmetrisches Verschlüsselungsverfahren
  - es gilt: $K_1 = K_2$

5 Kryptographische Maßnahmen (3)

- Forderungen an ein Verschlüsselungsverfahren
  - Wenn K_2 unbekannt ist, soll es sehr aufwendig sein aus $E(K_1, T)$ das T zu ermitteln (Entschlüsselungsangriff)
  - Es soll sehr aufwendig sein aus T und $E(K_1, T)$ den Schlüssel K_1 zu ermitteln (Klartextangriff)
  - Bei asymmetrischen Verfahren soll es sehr aufwendig sein, aus K_1 den Schlüssel K_2 zu ermitteln und umgekehrt.

5.1 Monoalphabetische Verfahren

- Verfahren nach Caesar

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E

  - Verschlüsselungsfunktion: $E: M \rightarrow (M + k) \bmod 26$
  - k ist variierbar (26 Möglichkeiten)
- Zufällige Substitution

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
F	Q	H	A	J	U	L	G	N	S	P	W	R	O	T	C	V	Y	X	M	Z	K	B	I	D	E

  - 26! Möglichkeiten

5.1 Monoalphabetische Verfahren (2)

★ Nachteil
- vollständiges Ausprobieren möglich bei Caesar
- Häufigkeitsanalyse der Buchstaben
  - für eine Sprache gibt es häufigere Buchstaben, z.B. e im Deutschen
  - durch die Häufigkeitsanalyse können die Möglichkeiten stark eingeschränkt werden; vollständiges Probieren wird ermöglicht

5.2 Polyalphabetische Verschlüsselung

- Einsatz von vielen Abbildungen, die durch einen Schlüssel ausgewählt werden
  - Beispiel: Vigenére (Caesar-Verschlüsselung mit zyklisch wiederholten Folgen von Verschiebungswerten)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
													...													
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
X	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

5.2 Polyalphabetische Verschlüsselung (2)

- Auswahl der Zeile durch den entsprechenden Buchstaben des Schlüsselwortes

W	I	C	H	T	I	G	E	N	A	C	H	R	I	C	H	T	Originaltext
G	E	H	E	I	M	G	E	H	E	I	M	G	E	H	E	I	Schlüsselwort (wiederholt)
C	M	J	L	B	U	M	I	U	E	K	T	X	M	J	L	B	verschlüsselter Text

- Gilt als nicht sicher
  - Koinzidenzanalyse
  - Häufigkeitsanalysen und Brute force Attacke

5.2 Polyalphabetische Verfahren (3)

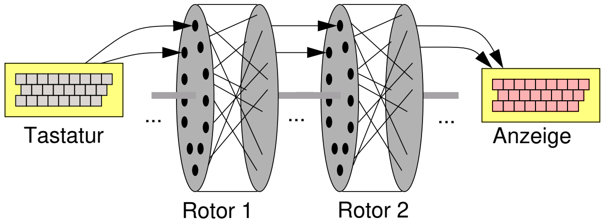
- Koinzidenz
  - Wahrscheinlichkeit für zwei gleiche Buchstaben untereinander bei umbrechendem Text
    - zufällige Buchstabenwahl: 3,8%
    - englischer Text: 6,6%
  - Brechen polyalphabetischer Verfahren
    - Bestimmen der Koinzidenz für verschiedene Textlängen
    - Textlänge mit höchster Koinzidenz ist wahrscheinlich Schlüsseltextlänge
    - danach Häufigkeitsanalyse pro Buchstabe des Schlüsseltexts

5.3 One-Time Pad Verfahren

- Theoretisch sicheres Verfahren
  - Liste von Zufallszahlen (soviele wie Zeichen in der Nachricht): $r[i]$
  - Zeichen $z[i]$ der Nachricht wird verschlüsselt mit $c[i] = (z[i] + r[i]) \bmod 26$
  - Empfänger braucht die gleiche Liste
  - theoretisch sicher, da aus dem $c[]$ nicht auf $z[]$ geschlossen werden kann
- Praktisch unbrauchbar
  - echte Zufallszahlen nötig
  - lange Liste nötig
    - jede Liste kann nur einmal verwendet werden
    - Liste muss so lang wie die Nachricht sein

5.4 Rotormaschinen

- Drehende Scheiben verändern ständig die Permutation

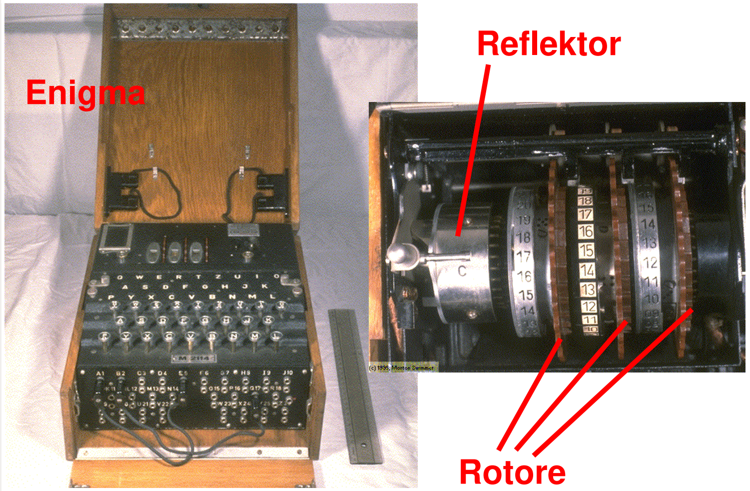


- ◆ Einstellen einer Anfangsposition für die Rotoren
- ◆ bei jedem Zeichen wird erster Rotor um eine Position weitergedreht
- ◆ zweiter Rotor rotiert mit niedrigerer Geschwindigkeit
- ◆ zum Entschlüsseln sind entsprechende Gegenstücke nötig

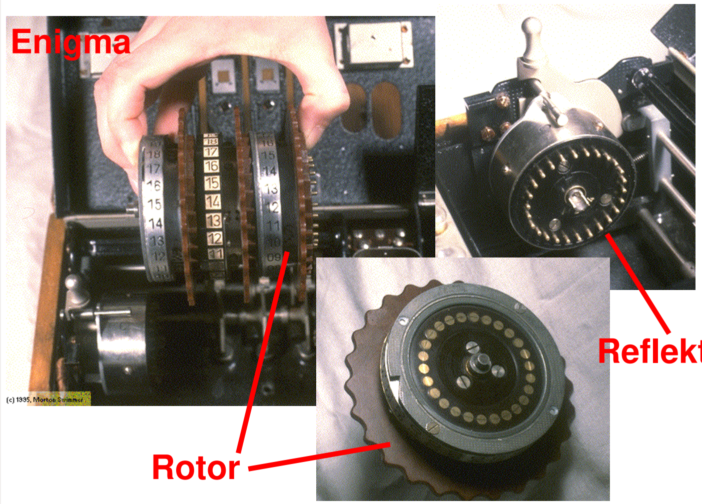
5.4 Rotormaschinen (2)

- Enigma
  - ◆ deutsche Chiffriermaschine aus dem zweiten Weltkrieg
  - ◆ drei Rotore und Reflektor
    - Reflektor leitet Strom wieder bei einer anderen Position durch die Rotoren zurück: Verfahren wird symmetrisch
    - Entschlüsseln mit den gleichen Rotoren möglich
- Verfahren gilt als nicht sicher
  - ◆ Brute force Attacke: *Collosus* Computer
- Schlüsseldemo
  - ◆ http://www.ugrad.cs.jhu.edu/~russell/classes/enigma/

5.4 Rotormaschinen



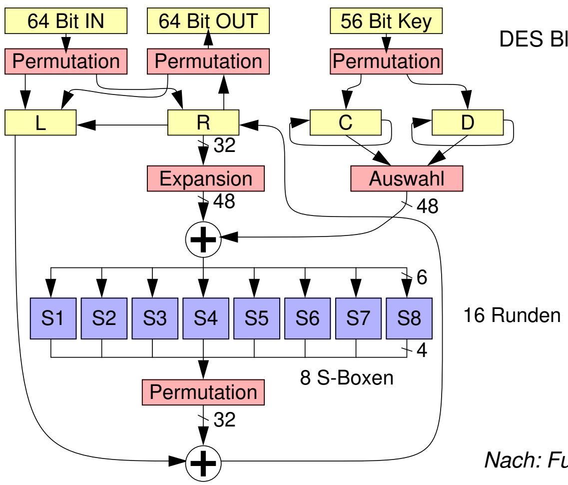
5.4 Rotormaschinen (2)



5.5 Heutige symmetrische Verfahren

- DES *(Data Encryption Standard, 1977)*
  - ◆ entwickelt von IBM
  - ◆ amerikanischer Standard (Kriegswaffe)
  - ◆ blockorientiertes Verfahren (64 Bit Block, 56 Bit Schlüssel)
  - ◆ 16 Runden
  - ◆ gilt heute als nicht mehr ganz sicher, da Rechenleistung von Großrechnern oder Rechenverbünden zum Brechen manchmal ausreicht

5.5 Heutige symmetrische Verfahren (2)



DES Blockdiagramm

Nach: Fumy/Rieß 1994

5.5 Heutige symmetrische Verfahren (3)

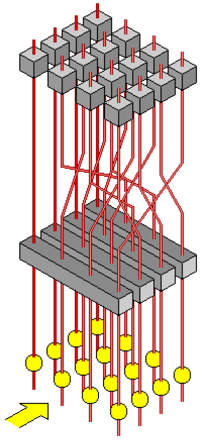
- Tripple DES
  - ◆ dreifache Verschlüsselung mit DES
  - ◆ Nutzung von drei oder mindestens zwei verschiedenen Schlüsseln
- IDEA *(International Data Encryption Algorithm)*
  - ◆ Alternative zu DES
  - ◆ 64 Bit Blockgröße
  - ◆ 128 Bit Schlüssel
  - ◆ keine Permutationen und S-Boxen
  - ◆ stattdessen: Addition, Multiplikation und XOR
  - ◆ 8 Runden und Output-Transformation
  - ◆ Einsatz: z.B. PGP *(Pretty Good Privacy)*

5.5 Heutige symmetrische Verfahren (4)

- AES — Advanced Encryption Standard (Rijndael)
  - ◆ entwickelt von Joan Daemen und Vincent Rijmen
  - ◆ blockorientiertes Verfahren
    - Blockgröße 128, 192 oder 256 Bits
    - Schlüsselgröße 128, 192 oder 256 Bits
  - ◆ 9, 11 oder 13 Runden je nach Schlüssellänge
  - ◆ wurde aus mehreren Vorschlägen als Nachfolger für DES ausgewählt

5.5 Heutige symmetrische Verfahren (5)

- Blockdiagramm einer Runde (stark vereinfacht)



Nach John Savard 2000

5.6 Beispiel: UNIX Passwörter

- Passwörter wurden zunächst im Klartext gespeichert
  - Passwortdatei muss streng geschützt werden
  - strenger Schutz oft nicht möglich (z.B. Backup der Platte)
  - Superuser kann die Passwörter von Benutzern einsehen
- Verschlüsseln der Passwörter
  - nur die verschlüsselte Version wird gespeichert
  - verschlüsselte Passwörter dürfen nicht leicht entschlüsselt werden können
- ▲ Ausprobieren von Passwörtern
  - Benutzer wählen Namen und Gegenstände als Passwort
  - Verschlüsseln von gängigen Begriffen und Vergleich mit verschlüsselt gespeicherten Passwörtern
  - Verschlüsselungszeit fließt mit ein in die Sicherheitsbetrachtung

5.6 Beispiel: UNIX Passwörter (2)

- Heutiges Verfahren
  - zufällige Auswahl eines von 4096 Werten *(Salt)*
  - der Salt fließt mit in die Verschlüsselung ein, so dass ein und dasselbe Passwort in 4096 Varianten vorkommen kann
  - Verschlüsselung mit DES
  - Zugriff auf verschlüsselte Passwörter wird weitestgehend verhindert (Shadow-Passwortdatei)
- ★ Vorteil
  - Ausprobieren von Passwörtern benötigt mehr Zeit
  - Vergleich zweier Passwörter weitgehend unmöglich

5.6 Beispiel: UNIX Passwörter (3)

- Politik am Institut für Informatik
  - Mindestlänge 8 Zeichen
  - mindestens 5 verschiedene Zeichen
  - mindestens 3 Zeichenklassen (Groß-, Kleinbuchstaben, Ziffern, Sonderzeichen)
  - keine Wiederholungen von Zeichenfolgen erlaubt
  - keine aufeinanderfolgenden Zeichen erlaubt, z.B. "123"
  - ...
  - Begriffe, Namen, etc. werden ausgeschlossen und müssen hinreichend verfremdet sein
- ★ Angriff durch Ausprobieren wird weitestmöglich erschwert

5.7 Heutige asymmetrische Verfahren

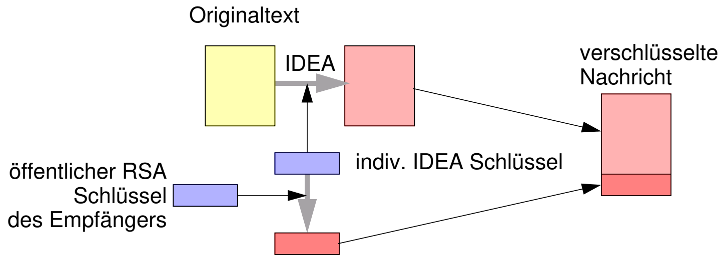
- RSA (Rivest, Shamir und Adleman)
  - Öffentlicher Schlüssel (zum Verschlüsseln) besteht aus *(e, N)*
  - Ein Block *M* wird verschlüsselt durch: $C = E((e, N), M) = M^e \bmod N$
  - *C* wird entschlüsselt durch: $M = D((d, N), C) = C^d \bmod N$
  - Wahl der Schlüssel:
    - Es muss gelten $\forall M : (M^e)^d = M \bmod N$
    - Aus Kenntnis von *e* und *N* darf *d* nur mit hohem Aufwand ermittelbar sein
  - Lösung:
    - $N = pq$ mit *p* und *q* zwei hinreichend große Primzahlen
    - zufällige Wahl von *d*, teilerfremd zu $(p-1)(q-1)$
    - Berechnung von *e* aus der Bedingung: $ed = 1 \bmod ((p-1)(q-1))$
  - Es ist aufwendig, die Primfaktoren von *N* zu berechnen (mit diesen wäre es möglich *d* zu ermitteln)

5.7 Heutige asymmetrische Verfahren (2)

- Vorteil asymmetrischer Verfahren (*Public key*-Verfahren)
  - nur ein Schlüsselpaar pro Teilnehmer nötig
    (sonst ein Schlüsselpaar pro Kommunikationskanal!)
  - Schlüsselverwaltung erheblich vereinfacht
    - jeder Teilnehmer erzeugt sein Schlüsselpaar und
    - veröffentlicht seinen öffentlichen Schlüssel
  - Authentisierung durch digitale Unterschriften möglich
  - gilt als sicher bei hinreichend großer Schlüssellänge (1024 Bit)
- Nachteil
  - relativ langsam berechenbar
  - gemischter Betrieb von asymmetrischen und symmetrischen Verfahren zur Geschwindigkeitssteigerung

5.7 Heutige asymmetrische Verfahren (3)

- Beispiel: PGP Verschlüsselung



  - Daten werden mit einem individuellen Schlüssel IDEA-verschlüsselt
  - IDEA-Schlüssel wird RSA-verschlüsselt der Nachricht angehängt

5.7 Heutige asymmetrische Verfahren (4)

★ Nachricht an mehrere Adressaten verschickbar
  - lediglich der IDEA-Schlüssel muss in mehreren Varianten verschickt werden
    (je eine Version verschlüsselt mit dem öffentl. Schlüssel des jeweiligen Empfängers)

5.8 Digitale Unterschriften

- Authentisierung des Absenders
  - Bilden eines Hash-Wertes über die zu übermittelnde Nachricht
    - Hash-Wert ist ein Codewort fester Länge
    - es ist unmöglich oder nur mit hohem Aufwand möglich, für einen gegebenen Hash-Wert eine zugehörige Nachricht zu finden
  - Verschlüsseln des Hash-Wertes mit dem geheimen Schlüssel des Absenders (digitale Unterschrift, digitale Signatur)
  - Anhängen des verschlüsselten Hash-Wertes an die Nachricht
  - Empfänger kann den Hash-Wert mit dem öffentlichen Schlüssel des Absenders dechiffrieren und mit einem selbst berechneten Hash-Wert der Nachricht vergleichen
  - stimmen beide Werte überein muss die Nachricht vom Absender stammen, denn nur der besitzt den geheimen Schlüssel

5.8 Digitale Unterschriften (2)

- Kombination mit Verschlüsselung
  - erst signieren
  - dann mit dem öffentlichen Schlüssel des Adressaten verschlüsseln
  - sonst Signatur verfälschbar
- ▲ Reihenfolge wichtig
  - Man signiere nichts, was man nicht entschlüsseln kann / versteht.
- Heute gängiges Hash-Verfahren
  - MD5
  - 128 Bit langer Hash-Wert

5.8 Digitale Unterschriften (3)

- Woher weiß ich, dass ein öffentlicher Schlüssel authentisch ist?
  - Ich bekomme den Schlüssel vom Eigentümer (persönlich, telefonisch).
    - Hash-Wert auf öffentlichen Schlüsseln, die leichter zu überprüfen sind (Finger-Prints)
  - Ich vertraue jemandem (Bürge), der zusichert, dass der Schlüssel authentisch ist.
    - Schlüssel werden von dem Bürgen signiert.
    - Bürge kann auch eine ausgezeichnete Zertifizierungsstelle sein.
    - Netzwerk von Zusicherungen auf öffentliche Schlüssel *(Web of Trust)*
  - Möglichst weite Verbreitung von öffentlichen Schlüsseln erreichen (z.B. PGP: Webserver als Schlüsselserver)

5.8 Digitale Unterschriften (4)

- ▲ Mögliche Probleme von Public key-Verfahren
  - Geheimhaltung des geheimen Schlüssels (Time sharing-System, Backup; Schlüsselpasswort / Pass phrase)
  - Vertrauen in die Programme (z.B. PGP)
  - Ausspähung während des Ver- und Entschlüsselungsvorgangs

6 Authentisierung im Netzwerk

- Viele Klienten, die viele Dienste in Anspruch nehmen wollen
  - Dienste *(Server)* wollen wissen welcher Benutzer *(Principal)*, den Dienst in Anspruch nehmen will (z.B. zum Accounting, Zugriffsschutz, etc.)
  - Im lokalen System reicht die (durch das Betriebssystem) geschützte Benutzerkennung (z.B. UNIX UID) als Ausweis
  - Im Netzwerk können Pakete abgefangen, verfälscht und gefälscht werden (einfache Übertragung einer Benutzerkennung nicht ausreichend sicher)
- Public key-Verfahren
  - Authentisierung durch digitale Unterschrift (mit geheimen Schlüssel des Senders) und Verschlüsseln (mit öffentlichem Schlüssel des Empfängers)
  - Nachteile
    - jeder Dienst benötigt sicheren Zugang zu allen öffentlichen Schlüsseln
    - Verschlüsseln und Signieren mit RSA ist sehr teuer

6 Authentisierung im Netzwerk (2)

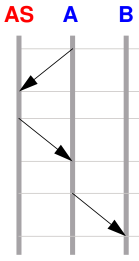
- ★ Einsatz von Authentisierungsdiensten
  - zentraler Server, der alle Benutzer kennt
  - Authentisierungsdienst garantiert einem Netzwerkdienst, dass ein Benutzer auch der ist, der er vorgibt zu sein
- Benutzerausweis
  - Authentisierungsdienst erkennt den Benutzer anhand eines geheimen Schlüssels oder Passworts
  - Schlüssel ist <u>nur</u> dem Authentisierungsdienst <u>und</u> dem Benutzer bekannt

6 Authentisierung im Netzwerk (3)

- Vorgang
  - Benutzer *(Principal)* will mit einem Programm *(Client)* einen Dienst *(Server)* in Anspruch nehmen
  - durch geeignetes Protokoll erhalten Client und Server jeweils einen nur ihnen bekannten Schlüssel, mit dem sie ihre Kommunikation verschlüsseln können *(Session key)*

6.1 Einfacher Authentisierungsdienst

- A will den Dienst B in Anspruch nehmen (Nach Needham-Schröder):



A schickt an den Authentisierungsdienst: a, b

AS erzeugt einen Session key $S_{A,B}$

AS antwortet: $E(K_A, \langle b, S_{A,B}, E(K_B, \langle S_{A,B}, a \rangle) \rangle)$

A entschlüsselt mit K_A die Nachricht

A sendet an B: $E(K_B, \langle S_{A,B}, a \rangle)$

B entschlüsselt mit K_B die Nachricht

  - K_x ist der geheime Schlüssel, den nur Authentisierungsdienst und X kennen
  - nach dem Protokollablauf kennen sowohl A und B den Session key
  - A weiß, dass nur B den Session key kennt
  - B weiß, dass nur A den Session key kennt

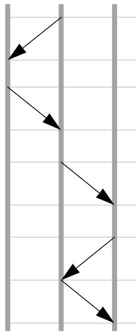
6.1 Einfacher Authentisierungsdienst (2)

- Problem
  - letzte Nachricht von A an B könnte aufgefangen und später erneut ins Netz gegeben werden *(Replay attack)*
  - Folge: Kommunikation zwischen A und B kann gestört werden
- Korrektur durch zusätzliches Versenden einer Verbindungsbestätigung durch B und A

6.2 Authentisierungsdienst mit Bestätigung

- Bestätigung enthält Einmalinformation *(Nonce)*

AS A B



A schickt an den Authentisierungsdienst: a, b, i_a

AS erzeugt einen Session key $S_{A,B}$

AS antwortet: E(K_A, <i_a, b, $S_{A,B}$, E(K_B, <$S_{A,B}$, a>)>)

A entschlüsselt mit K_A die Nachricht

A sendet an B: E(K_B, <$S_{A,B}$, a>)

B entschlüsselt mit K_B die Nachricht

B schickt an A: E($S_{A,B}$, i_b)

A antwortet an B: E($S_{A,B}$, i_b+1)

- ein Wiedereinspielen der Nachricht E(K_B, <$S_{A,B}$, a>) oder E($S_{A,B}$, i_b+1) wird erkannt und kann ignoriert werden

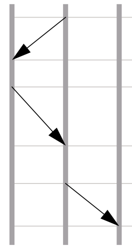
6.2 Authentisierungsdienst mit Bestätigung (2)

- Problem
  - Aufzeichnen von E(K_B, <$S_{A,B}$, a>) und
  - Brechen von $S_{A,B}$ erlaubt das Aufbauen einer Verbindung.
  - ein Dritter kann dann die erste Bestätigung abfangen und die zweite Bestätigung verschicken
- Lösung
  - Einführung von Zeitstempeln *(Time stamp)* und Angaben zur Lebensdauer *(Expiration time)*

6.3 Authentisierungsdienst mit Zeitstempeln

- Authentisierungsdienst versieht seine Nachrichten mit Zeitstempeln

AS A B



A schickt an den Authentisierungsdienst: a, b

AS erzeugt einen Session key $S_{A,B}$

AS antwortet:
E(K_A, <b, $S_{A,B}$, T_S, T_{Exp}, E(K_B, <$S_{A,B}$, a, T_S, T_{Exp}>)>)

A entschlüsselt mit KA die Nachricht

A sendet an B: E(K_B, <$S_{A,B}$, a, T_S, T_{Exp}>)

B entschlüsselt mit KB die Nachricht

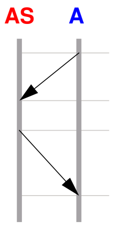
- T_S = Zeitstempel der Nachrichtenerzeugung
- T_{Exp} = maximale Lebensdauer der Nachricht
- aufgezeichnete Nachricht kann nach kurzer Zeit (z.B. 5min) nicht nocheinmal zum Aufbau einer Verbindung verwendet werden

6.4 Beispiel: Kerberos

- Kerberos V5
  - Softwaresystem implementiert Weiterentwicklung des Needham-Schröder-Protokolls
  - entwickelt am MIT seit 1986
- Ziel
  - Authentisierung und Erzeugung eines gemeinsamen Schlüssels durch den vertrauenswürdigen Kerberos-Server
- Idee
  - Trennung von Authentisierungsdienst und Schlüsselerzeugung
  - reduziert die nötige Übertragung einer Identifikation oder eines Passworts zum Kerberos-Server

6.4 Beispiel: Kerberos (2)

- Benutzer holt sich zunächst ein Ticket vom Authentisierungsdienst



A schickt an den Authentisierungsdienst: a, tgs, T_{Exp}, i_a

AS erzeugt ein Ticket für den Ticket Server tgs

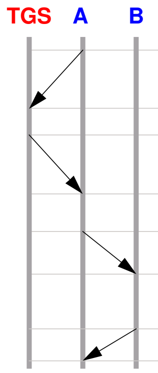
AS antwortet:
$E(K_A, \langle S_{A,TGS}, tgs, T_{Exp}, i_a, E(K_{TGS}, \langle S_{A,TGS}, a, T_{Exp}\rangle)\rangle)$

A entschlüsselt mit KA die Nachricht

- das Ticket besteht aus $\langle S_{A,TGS}, a, T_{Exp}\rangle$
- es enthält einen Session key für die Kommunikation mit einem Ticket granting server, der dann die Verbindung zu einem Netzwerkdienst bereitstellen kann

6.4 Beispiel: Kerberos (3)

- Authentisierungsdienst versieht seine Nachrichten mit Zeitstempeln



A schickt an den Ticket granting server:
$E(S_{A,TGS}, T_s), E(K_{TGS}, \langle S_{A,TGS}, a, T_{Exp}\rangle), b, T_{Exp}, i_a'$

TGS erzeugt einen Session key $S_{A,B}$

TGS antwortet:
$E(S_{A,TGS}, \langle S_{A,B}, b, T_{Exp}, i_a'\rangle), E(K_B, \langle S_{A,B}, a, T_{Exp}\rangle)$

A entschlüsselt mit $S_{A,TGS}$ die Nachricht

A sendet an B: $E(S_{A,B}, \langle T_s, C, S_{sub}\rangle), E(K_B, \langle S_{A,B}, a, T_{Exp}\rangle)$

B entschlüsselt mit K_B den zweiten Teil der Nachricht (Ticket) und mit $S_{A,B}$ den ersten Teil

B antwortet (optional): $E(S_{A,B}, T_s)$

- C = Checksumme zur Überprüfung der richtigen Entschlüsselung
- A kann mehrere Verbindungen mit seinem Ticket öffnen

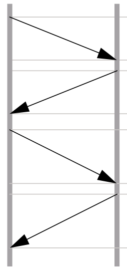
6.4 Beispiel: Kerberos (4)

- Unterscheidung zwischen Benutzer *(User)* und Benutzerprogramm *(Client)*
  - Wie kann ein Benutzerprogramm seinen Benutzer identifizieren?
  - Geheimer Schlüssel vom Benutzer (K_X) hängt von einem Passwort ab
  - mittels einer Einwegfunktion wird aus dem Passwort der Schlüssel K_X erzeugt
  - Benutzerprogramm braucht also das Passwort zur Verbindungsaufnahme
- Beispiel: *kinit, klogin*
  - Anmeldung beim Authentisierungsdienst mit *kinit* und Passworteingabe
  - Ticket wird im Benutzerverzeichnis gespeichert
  - *klogin* erlaubt das Einloggen auf einem entfernten Rechner mit Datenverschlüsselung und ohne Passwort

6.5 Austausch öffentlicher Schlüssel

- A und B tauschen ihre öffentlichen Schlüssel aus

A **B**



A schickt an B seinen öffentlichen Schlüssel: K_A

B schickt an A seinen öffentlichen Schlüssel: K_B

A verschickt signierte und verschlüsselte Nachricht:
$E(K_B, E(K_A^{-1}, T_1))$

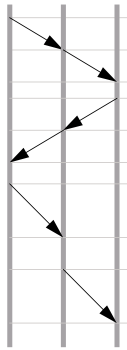
B verschickt signierte und verschlüsselte Nachricht:
$E(K_A, E(K_B^{-1}, T_2))$

- Problem
  - A und B können nicht sicher sein, dass der öffentliche Schlüssel wirklich vom jeweils anderen stammt

6.5 Austausch öffentlicher Schlüssel (2)

- Aktiver Mithörer C fängt Datenverbindungen ab *(Man in the middle attack)*

A **C** **B**



A versucht an B seinen öffentlichen Schlüssel zu schicken: K_A

C fängt Nachricht ab und schickt an B seinen eigenen öffentlichen Schlüssel: K_C

B versucht an A seinen öffentlichen Schlüssel zu schicken: K_B

C fängt Nachricht ab und schickt an A seinen eigenen öffentlichen Schlüssel: K_C

A versucht an B eine Nachricht zu schicken:
$C_1 = E(K_C, E(K_A^{-1}, T_1))$

C dekodiert die Nachricht: $T_1 = D(K_C^{-1}, D(K_A, C_1))$

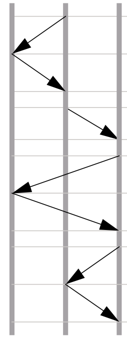
C schickt die Nachricht weiter: $E(K_B, E(K_C^{-1}, T_1))$

B glaubt eine Nachricht von A zu empfangen

6.5 Austausch öffentlicher Schlüssel (3)

- Einsatz eines Authentisierungsdienstes

AS **A** **B**



A kontaktet AS und möchte Verbindung mit B: a, b

AS verschickt den Schlüssel von B: $E(K_{AS}^{-1}, \langle K_B, b\rangle)$

A schickt initiale Nachricht an B: $E(K_B, \langle a, i_a\rangle)$

B kontaktet AS und möchte den Schlüssel von A: b, a

AS verschickt den Schlüssel von A: $E(K_{AS}^{-1}, \langle K_A, a\rangle)$

B antwortet an A: $E(K_A, \langle i_a, i_b\rangle)$

A bestätigt: $E(K_B, \langle i_b\rangle)$

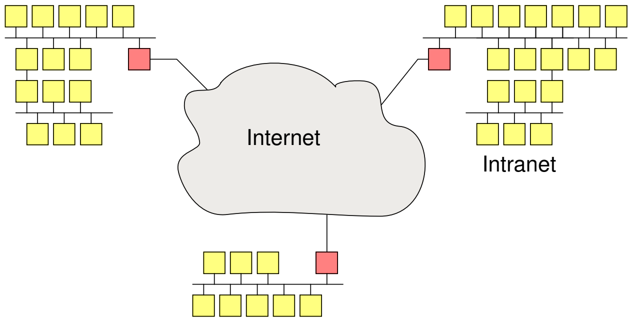
- Replay-Probleme
  - Hinzunahme von Zeitstempel und Lebendauer

7 Firewall

- Trennung von vertrauenswürdigen und nicht vertrauenswürdigen Netzwerksegmenten durch spezielle Hardware *(Firewall)*
  - Beispiel: Trennen des firmeninternen Netzwerks (Intranet) vom allgemeinen Internet
- Funktionalität
  - Einschränkung von Diensten
    - von innen nach außen, z.B. nur Standarddienste
    - von außen nach innen, z.B. kein Telnet, nur WWW
  - Paketfilter
    - Filtern „defekter“ Pakete, z.B. SYN-Pakete
  - Inhaltsfilter
    - Filtern von Pornomaterial aus dem WWW oder News
  - Authentisieren von Benutzern vor der Nutzung von Diensten

7 Firewall (2)

- Virtual private network
  - Verbinden von Intranet-Inseln durch spezielle Tunnels zwischen Firewalls
  - getunnelter Datenverkehr wird verschlüsselt
  - Benutzer sieht ein „großes Intranet“ (nur virtuell vorhanden)



8 Richtlinien für den Benutzer

8.1 Passwörter

- Wahl eines Passworts
  - hinreichend komplexe Passwörter wählen
  - Schutz vor Wörterbuchangriffen
  - verschiedene Passwörter für verschiedene Aufgaben (z.B. PPP-Passwort ungleich Benutzerpasswort)
- Aufbewahrung
  - möglichst nirgends aufschreiben
  - nicht weitergeben
  - kein Abspeichern auf einem Windows-Rechner (Option immer wegklicken)

8.1 Passwörter

- Eingabe
  - niemals über eine unsichere Rechnerverbindung eingeben
    - `ftp, telnet, rlogin` Dienste vermeiden
    - nur sichere Dienste verwenden: `ssh, slogin`
    - Datenweg beachten, über den das Passwort läuft: ein unsicheres Netzwerk ist bereits genug
- Änderung
  - Passwörter regelmäßig wechseln
  - alte Passwörter nicht wiederverwenden

8.2 Schlüsselhandhabung

- Einsatz von PGP oder S/MIME
  - Zugang zu den privaten Schlüsseln für andere verhindern
  - Dateirechte auf der Schlüsseldatei prüfen
  - privater Schlüssel nur auf Diskette
  - Passphrase wie ein Passwort behandeln
  - privaten Schlüssel nie über unsichere Netze transportieren

8.3 E-Mail

- Authentisierung
  - Bei elektronischer Post ist der Absender nicht authentisierbar
  - Digitale Unterschriften einsetzen (z.B. mit PGP oder S/MIME)
- Abhören
  - Elektronische Post durchläuft viele Zwischenstationen und kann dort jeweils gelesen und verfälscht werden
  - Verschlüsselung einsetzen (z.B. mit PGP oder S/MIME)

8.4 Programmierung

- S-Bit Programme vermeiden
  - Oft kann das S-Bit durch geschickte Vergabe von Benutzergruppen an Dateien vermieden werden
- Verwendung zusätzlicher Rechte (z.B. durch S-Bit) nur in Abschnitten
  - Trusted Computing Base (TCB) [hier kein vollständiger Schutz]

```
seteuid(getuid());/* am Programmanfang Rechte wegnehmen */
...
seteuid(0);        /* setzt root Rechte */
fd = open("/etc/passwd", O_RDWR);
seteuid(getuid());/* nimmt root Rechte wieder weg */
...
```

- Sorgfältige Programmierung
  - Funktionen wie **strcpy, strcat, gets, sprintf, scanf, sscanf, system, popen** vermeiden oder durch **strncpy, fgets, snprintf** ersetzen

8.5 World Wide Web

- Cookies
  - Akzeptieren von Cookies erlaubt einer Website die angesprochenen Seiten genau einem Benutzer zuzuordnen
  - funktioniert über Sessions hinweg
- JavaScript
  - schwere Sicherheitslücken erlauben es, alle für den Benutzer lesbare Dateien an einen Dritten weiterzugeben
  - Ausschalten!
- Abhören
  - WWW-Verbindungen können abgehört werden
  - keine privaten Daten, wie z.B. Kreditkartennummern übertragen
  - Secure-HTTP mit SSL-Verschlüsselung benutzen; https-URLs