4 Elementare Zugriffskontrolle

Zugriffskontrolle soll sicherstellen, dass jeder Zugriff auf Schutzobjekte berechtigt ist.

Jegliche Zugriffskontrolle basiert auf zwei Voraussetzungen:

1. Die (persistenten) Daten, die die Rechte des Benutzers festlegen, können nicht unberechtigt geändert werden. Dazu müssen nicht nur Benutzer-Schutzobjekte, sondern auch System-Schutzobjekte (z.B. Rechtetabellen) geschützt werden.

2. Verlässliche **Identifikation**, um Maskeraden, also Vortäuschung falscher Identitäten, zu verhindern. Dies umfasst:

- Benutzeridentifikation: Kein Benutzer soll in der Lage sein, die Zugriffsrechte eines anderen zu erlangen. Für diesen Zweck führt man Authentifikationsverfahren (z.B. mittels Vorweisen von **Wissen** wie Passworte, von **Besitz** wie Magnet- oder Chipkarte oder von physischen **Eigenschaften** wie Fingerabdruck) während des Logins (besser: vor jeder wichtigen Handlung) ein.

- Identifikation von Operationen als Schutz gegen **trojanische Pferde**. Ein trojanisches Pferd ist ein verborgenes Stück Software, das ohne Kenntnis des Subjektes, aber mit der Identität des Subjektes handelt und daher alle Rechte des Subjektes hat. (Insbesondere kann man ein Programm mit einem unentdeckten **Virus** als trojanisches Pferd auffassen.)

- Identifikation von Granulen, um unverfälschte Originaldaten zu bearbeiten;
  insbesondere auch wegen möglicher Rollenveränderungen: Granule können Prozeduren oder Programme sein und Prozeduren können Operationen werden, Programme können Prozesse (Subjekte) werden.

Viele Zugriffskontrollen basieren auf dem Konzept des (Daten-)**Eigentum**s (*ownership*), d.h. die Benutzer verwalten die Rechte für die Schutzobjekte, die sie besitzen.

4.1 Elementare Theorie der Erlaubnisse

Zunächst sind die Zugriffsrechte zu spezifizieren. Dazu benötigen wir folgende Definitionen:

Sei S eine Menge von **Subjekten** (aktive Einheit / Handelnde wie z.B. Benutzer, Rolle eines Benutzers, Prozess),

O eine Menge von **Operationen** (Aktion / Zugriffstyp wie z.B. read, insert, modify, append, create index, execute, create, drop, own, grant, revoke),

G eine Menge von **Granulen** (Schutzobjekt / passive Einheit / Gegenstand wie z.B. Datenbankobjekt, Relation, Sicht, Attribute, Tupel, Element, Datei, Programm, Speicherseite, Gerät).
(manchmal betrachtet man die Subjekte als spezielle Schutzobjekte: S G)

Genauer müsste man eigentlich unterscheiden zwischen

- den endlichen Mengen von existierenden Subjekten, Operationen, Granulen eines konkreten Systems zu einem konkreten Zeitpunkt und
- abzählbar unendlichen Mengen von möglichen Subjekten, Operationen und Granulen (um stets neue erzeugen zu können).

Eine **Erlaubnis** legt dann fest, welche **Handlung** erlaubt ist, genauer:
welches **Subjekt** welche **Operation** bzgl. welchen **Granul**s ausführen darf.

Abstrakt beschreibt man einen Rechtezustand oft mit Hilfe einer **Rechtematrix** (**Erlaubnismatrix**) M.

Die klassische Rechtematrix ist zweidimensional mit je einer Achse für Subjekte s und Granule g.
Jeder Matrixeintrag enthält dann die Menge der Operationen, die für ein (Subjekt, Granul)-Paar erlaubt sind:
M : S×G → (O)

$$M = \begin{array}{c} \begin{array}{cccc} [\,g1 & g2 & g3 & g4\,] \end{array} \\ \begin{array}{cccc} \{execute\} & \{read\} & & \\ & \{read, write\} & \{read\} & \\ & & & \\ & & & \end{array} \end{array} \begin{array}{l} \\ s1 \\ s2 \\ s3 \\ s4 \end{array}$$

Diese Sichtweise ist adäquat, falls die Menge der anwendbaren Operationen klein und konstant ist.

In erweiterbaren (also auch objektorientierten) Systemen ist dies nicht unbedingt der Fall.
Daher sollte die Menge der Operationen gleichberechtigt eine eigene Achse bekommen.

Die Rechtematrix ist dann dreidimensional mit Einträgen wie true oder {erlaubt} für "Recht vorhanden" bzw. false oder Ø für "Recht nicht vorhanden". M : S×O×G → ({erlaubt})

(Leider ist die Rechtematrix jetzt schwieriger zu malen, z.B. durch entsprechende Projektionen.)

$$
\begin{array}{l} \\ \text{M(execute)} = \\ \\ \\ \\ \end{array}
\begin{array}{c}
\begin{bmatrix} g_1 & g_2 & g_3 & g_4 \end{bmatrix} \\
\begin{bmatrix}
\varnothing & \{\text{erlaubt}\} & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing
\end{bmatrix}
\begin{array}{c} \\ \\ \\ \\ \end{array}
\begin{array}{c}
\\ s_1 \\ s_2 \\ s_3 \\ s_4
\end{array}
\end{array}
$$

$$
\begin{array}{l} \\ \text{M(read)} = \\ \\ \\ \\ \end{array}
\begin{array}{c}
\\
\begin{bmatrix}
\varnothing & \varnothing & \{\text{erlaubt}\} & \varnothing \\
\varnothing & \varnothing & \{\text{erlaubt}\} & \{\text{erlaubt}\} \\
\varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing
\end{bmatrix}
\begin{array}{c} \\ \\ \\ \\ \end{array}
\begin{array}{c}
\\ s_1 \\ s_2 \\ s_3 \\ s_4
\end{array}
\end{array}
$$

$$
\begin{array}{l} \\ \text{M(write)} = \\ \\ \\ \\ \end{array}
\begin{array}{c}
\\
\begin{bmatrix}
\varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \{\text{erlaubt}\} & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing \\
\varnothing & \varnothing & \varnothing & \varnothing
\end{bmatrix}
\begin{array}{c} \\ \\ \\ \\ \end{array}
\begin{array}{c}
\\ s_1 \\ s_2 \\ s_3 \\ s_4
\end{array}
\end{array}
$$

Diese Rechtematrix kann sich ändern

- im Wert (durch Änderung von Rechten), sowie
- in der Struktur (durch Änderung der Anwendungswelt, nämlich der Subjekte, Operationen oder Granule).

Diese Matrix kann man jetzt aus der Sicht eines Subjektes, einer Operation oder eines Granuls betrachten. Üblicherweise wird damit auch der Ort der physischen Speicherung der Rechte festgelegt.

Die Erlaubnisse eines Subjekts (Zeile der zwei-dimensionalen Matrix) werden durch eine **capability list** C beschrieben. Die Rechte werden hier beim Subjekt abgespeichert.

(Alltagsbeispiel: Schlüsselbund)

Beispiel: C(s1)={(execute, g2), (read, g3)}.

Die Erlaubnisse auf ein Granul (Spalte der zwei-dimensionalen Matrix) werden durch eine **Zugriffskontrollliste** A (***a****ccess* ***c****ontrol* ***l****ist, authorization list*) beschrieben.
Die Rechte werden hier beim Granul abgespeichert.

(Alltagsbeispiel: Zahlenschloss)

Beispiel: A(g3)={(s1, read), (s2, read), (s2, write)}.

Die Erlaubnisse für eine Operation werden in einer **method control list** B beschrieben (Ebene der dritten Dimension in der drei-dimensionalen Matrix).
Die Rechte werden hier bei der Operation abgespeichert.

Dies ist eine angemessene Sichtweise, wenn in objekt-orientierten Systemen die Zugriffskontrolle innerhalb der Methoden stattfinden soll.

Beispiel: B(read)={(s1,g3), (s2,g3), (s2,g4)}.

Da sich diese verschiedenen Interpretationen abstrakt zunächst nur durch unterschiedliche Gruppierung unterscheiden, ist klar, dass es acht verschiedene Möglichkeiten der Interpretation gibt:

1. Rechtematrix:	$S \times O \times G \rightarrow (\{erlaubt\})$
2. :	$O \times G \rightarrow (S)$
3. klassische Rechtematrix:	$S \times G \rightarrow (O)$
4. :	$S \times O \rightarrow (G)$
5. Capabilities:	$S \rightarrow (O \times G)$
6. method control list:	$O \rightarrow (S \times G)$
7. Zugriffskontrolllisten:	$G \rightarrow (S \times O)$
8. Rechtemenge:	$\rightarrow (S \times O \times G)$

••• analog **Verbotsmatrix** $M : S \times O \times G \rightarrow (\{verboten\})$
••• analog **Erlaubnis- und Verbotsmatrix** $M : S \times O \times G \rightarrow (\{erlaubt, verboten\})$

Der **Rechtezustand** eines Systems wird also beschrieben durch:

- die endlichen Mengen der vorhandenen Subjekte, Operationen, Granule,
- die Zugriffskontrollmatrix,
- globale Vorschriften (insbesondere zum Erzeugen und Entfernen von Subjekten, Operationen und Granulen).

Da die globalen Vorschriften im wesentlichen unveränderbar sind, und die Menge der aktuell vorhandenen Subjekte, Granule und Operationen auch implizit aus der Zugriffskontrollmatrix ersehen werden kann, beschränken wir uns im folgenden auf die Zugriffskontrollmatrix.

In (relationalen) Datenbanksystemen sind z.B.

- die Subjekte Datenbankbenutzer und
- die Operationen Lesen, Einfügen, Ändern, Löschen, Index_anlegen, Schema_erweitern,
- die Granule Datenbankbereiche, Relationen, Sichten, Tupel, oder Tupelkomponenten (Werte).

Jeder Eintrag in der (dreidimensionalen) Zugriffskontrollmatrix ist eine Regel, die sagt, unter welchen

Bedingungen Benutzer s mit Operation o auf Granul g zugreifen darf.

Erlaubnisse können **unbedingt** (wie bei Betriebssystemen) oder **bedingt** sein.

Bei unbedingten Erlaubnissen steht {erlaubt} bzw. true oder Ø bzw. false in der (dreidimensionalen) Zugriffskontrollmatrix, bei bedingten Erlaubnissen spezifiziert oft ein Prädikat (das dann zu true oder false ausgewertet werden kann) die Bedingung.

Diese Bedingungen können z.B.

- **wertabhängig** sein, d.h. eine Funktion der Werte der Daten, auf die zugegriffen werden soll (z.B. Gehalt<5000).
- **zeitabhängig** sein (z.B. Zugriff nur zwischen 9 und 17 Uhr),
- **terminalabhängig** sein (z.B. sensible Operationen nur von der Systemkonsole),
- **kontext-abhängig** sein (z.B. ein Benutzer darf Noten oder Namen von Studierenden sehen, aber nicht beides zusammen), oder (als Kombination)
- **von der Vorgeschichte abhängen** (z.B. darf jemand, der ein wichtiges Staatsgeheimnis erfahren hat, vielleicht nicht gleich anschließend Interviewpartner einer Zeitung sein; abstrakter: ein Prozess darf nicht in eine unklassifizierte Datei schreiben, wenn er vorher klassifizierte Daten bearbeitet hat).

4.2 Weitergabe von Rechten

Ein Subjekt s_1 hat ein Recht bezüglich einem Granul g und Operation o und gibt dieses Recht an Subjekt s_2 weiter.

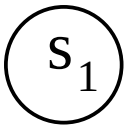
Dabei legt s_1 durch eine **Weitergabeberechtigung** (*grant flag*) f fest, ob s_2 dieses Recht an weitere Subjekte weitergeben darf.

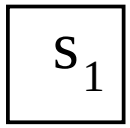
Formal ist eine **Rechteweitergabe** also ein Tupel $rwg=(s_1, o, g, s_2, f)$.

(In den mittleren drei Komponenten können ggf. auch Mengen stehen.)

Zur Veranschaulichung gibt es für jedes Granul g einen **Rechteweitergabegraphen** RWG(g)=(N,E,m). Dies ist ein gerichteter Graph mit Kantenmarkierungen:

Die Knotenmenge bestimmt sich als $N := S \times \{aktiv, passiv\}$,
d.h. jedes Subjekt kommt in diesem Graphen zweimal vor, einmal in der Rolle, dass das Subjekt Rechte besitzt, die es weitergeben kann (aktiv), zum zweiten, dass das Subjekt -nicht weitergebbare- Rechte besitzt (passiv).

Graphisch veranschaulichen wir die Situation, indem wir (s_1, aktiv) als Kreis zeichnen, 

(s_1, passiv) als Quadrat zeichnen 

und isolierte Knoten aus Gründen der Übersichtlichkeit weglassen.

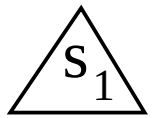
Die Kantenmenge $E \quad N \times N$ von RWG(g) ist bestimmt durch

$e=((s_1,r_1),(s_2,r_2)) \quad E :\Leftrightarrow$
r_1=aktiv und Rechteweitergabe (s_1, o, g, s_2, f): r_2=(if f then aktiv else passiv)

Die weitergebene Operation wird durch die Kantenmarkierung repräsentiert:

m: $E \rightarrow$ (O), wobei O die Menge der Operationen ist und
o_1 $m(e)=m((s_1,r_1),(s_2,r_2))$: Rechteweitergabe (s_1, o, g, s_2, f): (o_1=o und f=(r_2=aktiv))

Die Wurzel des RWG ist der Erzeuger s_1 des Granules, der alle möglichen Rechte aktiv hat.

Wir zeichnen für einen Erzeuger (als Spezialfall eines aktiven Subjekts) kurz: 

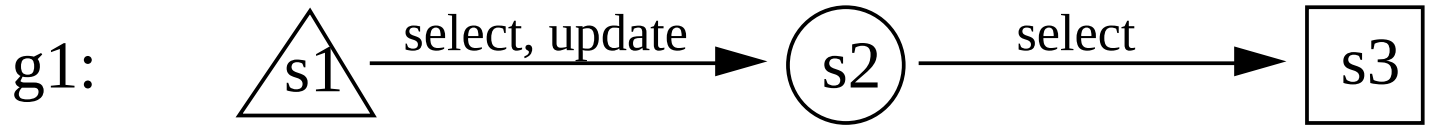
Beispiel: s_1 sei Eigentümer (repräsentiert durch Dreieck) eines Granuls g_1.

s_1 gibt das Select-Recht und das Update-Recht auf g_1 an s_2 einschließlich der Weitergabeoption (repräsentiert durch Kreis):

s_1: GRANT select, update ON g_1 TO s_2 WITH GRANT OPTION.

s_2 gibt das Select-Recht an s_3 ohne Weitergabeoption (repräsentiert durch Quadrat):

s_2: GRANT select ON g_1 TO s_3.



Achtung:
Das gleiche Subjekt kann zweimal (als Kreis bzw. Dreieck oder als Quadrat) im Graphen auftreten.

Beim **Rechterückruf** kann man einen rekursiven und nicht-rekursiven Rechterückruf unterscheiden.

Ruft s_1 im obigen Beispiel das Select-Recht von s_2 zurück, so verliert s_2 dieses Recht in jedem Fall.

Ein rekursiver Rückruf entfernt außerdem das Selekt-Recht von s_3, da s_3 dieses Recht nur von s_2 erhalten hat.

(Kanten um Zeitpunkte ergänzbar für rekursiven Rückruf)

4.3 Klassifikation der Sicherheitsstrategien für Zugriffskontrolle

Sicherheitsstrategien für Zugriffskontrolle können grob unterschieden werden bezüglich:

A. Strategien bzgl. der Rechteverwaltung

1. Zentralisierungsgrad:
Werden alle Sicherheitsaspekte eines Systems **zentral** (von einer Person oder Gruppe) (wie z.B. in INGRES) oder **dezentral** (verschiedene Administratoren für verschiedene Teile der Datenbank) (und daher effizienter oder bequemer) gesteuert?

2. Rechteverwalter:
Wer verwaltet die Rechte?
Ein spezieller (Rechte-)**Administrator** (*administration*) für alle Rechte oder jeder **Benutzer** (*ownership*) für die von ihm erzeugten Granule?

3. Rechteänderungen:
Sind Änderungen der Rechte (z.B. Erzeugung neuer Rechte, Löschung vorhandener Rechte, Weitergabe von Rechten, Rückruf von weitergegebenen Rechten) durch den Verwalter der Rechte vorgesehen? Ist es ein **statisches** oder **dynamisches** (Rechte-)System?

B. Strategien bzgl. der Spezifikation der Zugriffsrechte

1. Enge oder weite Informationsverbreitung:
Wird der Zugang zu Informationen auf die Personen beschränkt, die diese Information für ihre Arbeit brauchen:

kleinstmögliche Berechtigungen (*need-to-know policy, policy of least privilege*)?
Dieses Prinzip minimiert die Anzahl der möglichen Informationslecks und der möglichen Integritäts-verletzungen. (z.B. bei personenbezogenen Daten, Geschäftsgeheimnissen etc.)

Eine umgekehrte Strategie versucht, größtmöglichen Zugriff auf Information zu gewähren:

maximale Informationsverbreitung (*maximized sharing*).
(z.B. bei Bibliotheken, Informationsvermittlungsstellen, Werbung etc.)

2. Fehlende Spezifikation:
Ist eine Situation, für die keine Rechte (also weder Erlaubnisse noch Verbote) vorhanden sind, erlaubt oder verboten?
Je nachdem spricht man von einem **positiven** (*open system*) oder **negativen System** (*closed system*) ("Alles, was nicht spezifiziert ist, ist verboten.").

In einer Welt nur mit Erlaubnissen formuliert man gern ein negatives System:
"Alles, was nicht spezifiziert ist, ist verboten!"
(und analog ein positives System nur mit Verboten: "Alles, was nicht spezifiziert ist, ist erlaubt!")

Bei einem negativen System werden Spezifikationsfehler (z.B. vergessene Rechte) zwangsläufig schneller erkannt. Für Sicherheitsbetrachtungen ist üblich, negative Systeme zugrunde zu legen.

3. Detaillierungsgrad:
Welche Unterteilungen (Granularität) der Granule sind rechtemäßig möglich, d.h. für wie kleine Einheiten kann man (unterschiedliche) Rechte vergeben?
z.B. für Relationen, für Tupel, für Attribute oder für Werte?

4. Adressierung:
Wie werden die Granule (ggf. auch die Subjekte, Operationen), für die ein Recht gelten soll, bestimmt?

Über **Namen** (von Relationen, Attribute etc.) (*name dependent access control*, *content independent access control*) oder

den **Wert** (*content dependent access control*) eines Granuls?

C. Strategien bzgl. (der Kontrolle) des Informationsflusses

1. Kann ein Recht (nach beliebigen lokalen Kriterien des Eigentümers) an andere Benutzer weitergegeben werden: **eigentümergesteuerte Zugriffskontrolle** (*owner driven access control, discretionary access control, dac*)?

Oder gibt es neben einem Mechanismus, der überprüft, wer was wie (nicht) machen darf, außerdem noch einen Mechanismus, der stets prüft, ob bestimmte maximale (statische) Rechte eines Nutzers nicht doch überschritten werden (*nondiscretionary access control, mandatory access control, mac*)?

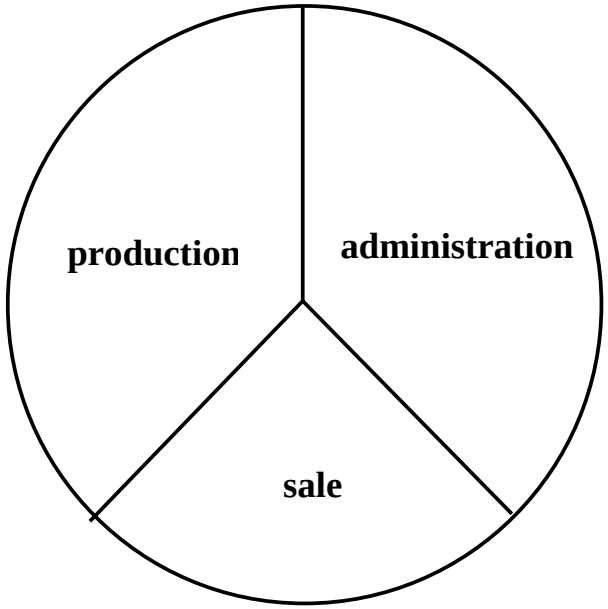
So kann eine Anwendung in verschiedene Bereiche (z.B. Produktion, Verkauf, Werbung) aufgeteilt werden und jeder Benutzer eines Bereiches kann nur auf die Daten seines Bereiches zugreifen:

Zugriffskontrolle mit vorgegebenen Schranken (*access control with borders, compartment access control*). Dieser Ansatz ist gut mit dem Prinzip der kleinstmöglichen Berechtigungen verträglich.

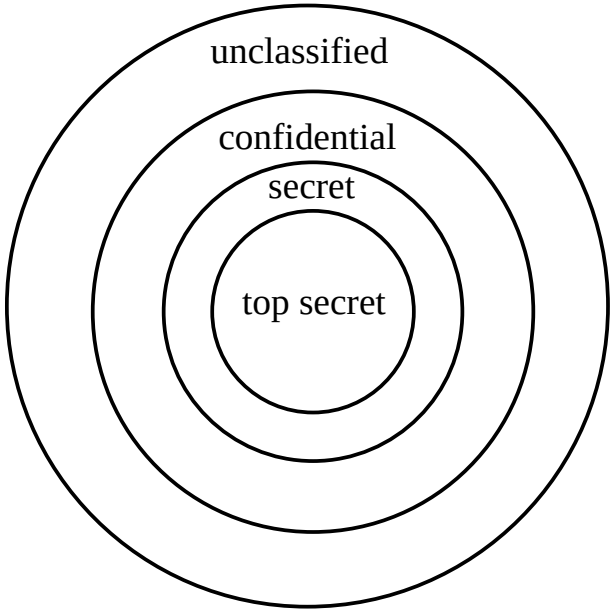
Verbreitet ist auch ein Ansatz mit halbdurchlässigen Schranken mittels Sicherheitsstufen (z.B. geheim, offen):

Mehrstufige Zugriffskontrolle (*multilevel access control*), wo gefordert wird, dass ein Benutzer keine Daten lesen kann, sofern nicht seine Sicherheitsstufe größer gleich der Sicherheitsstufe der Daten ist und dass Schreiben keinen Informationsfluss von einer größeren zu einer kleineren Sicherheitsstufe verursacht. Die beiden letzten Ansätze treten oft kombiniert auf.

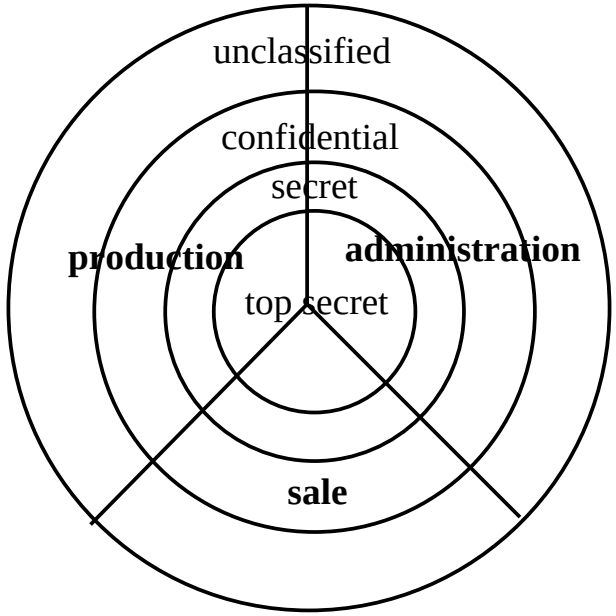
nondiscretionary access control



multilevel access control



nondiscretionary multilevel access control



Typische Umgebungen sind:
a) Discretionary Access Control: Negatives System, benutzer-verwaltet, dynamisch, Referenz-Monitor-Überprüfung
b) Mandatory Access Control: Negatives System, adminstrator-verwaltet, statisch, Überschreitung der maximalen Rechte wird überprüft, (evtl. kombiniert mit einem zusätzlichen DAC-Modell)

D. Strategien bzgl. der Kontroll-Durchsetzung

1. Werden Mechanismen verwendet, die eine Verletzung der Sicherheitsstrategie **verhindern** oder **erkennen** (z.B. durch Protokollierung) sie diese nur?

4.4 Eigentümergesteuerte Zugriffskontrolle (Eigentümergesteuerte Rechteweitergabe)

(*owner driven access control, discretionary access control*)

Politik:

1. Zu jedem Granul gibt es (mindestens) einen ausgezeichneten Benutzer, der **Eigentümer** (*owner*) eines Granuls ist.

2. Als Eigentümer hat er automatisch das **own-Recht** auf dieses Granul.
   Damit hat der Eigentümer eines Granuls das Recht, Benutzern (sich selbst und anderen!) beliebige andere Rechte bzgl. dieses Granuls **weiterzugeben** (*grant*) bzw. **zurückzufordern** (*revoke*).

   Der Bequemlichkeit halber kann man vereinbaren, dass eine voreingestellte Menge von Rechten auf das neue Granul automatisch weitergeben wird (z.B. Leserecht für den Eigentümer).
   Aber eigentlich bekommt der Benutzer bei der Erzeugung eines neuen Granuls zunächst nur das own-Recht. Dies ist besser als unmittelbar schon alle Rechte zu haben, die man dann versehentlich benutzen kann: "Sicherheit vor seinen eigenen Fehlern".
   (Daher sollte man auch stets explizit machen, welche seiner Rechte man nutzen will. Dies kann z.T. bei einer Operation explizit angegeben, z.T. implizit durch Voreinstellungen realisiert werden.)

3. Der **Erzeuger** (*creator*) eines Granuls wird standardmäßig Eigentümer dieses Granuls.
   Damit bestimmt das Recht, neue Granule zu erzeugen (*create*), darüber, wer Eigentümer eines (neuen) Granuls werden kann.

Dies ist offensichtlich ein dezentraler Ansatz.

Sowohl die Eigentumsrechte an Information (*ownership of information*) als auch Regeln zur Rechteweitergabe (*delegation of rights*) sind dezentral geregelt.

Darf das Weitergaberecht weitergegeben werden?

Damit gibt der Eigentümer sein alleiniges Bestimmungsrecht über Rechte an diesem Granul auf und kann nicht mehr kontrollieren, wohin die Rechte gewandert sind.

(Aber in realen Situationen ist Weitergabe nicht zu verhindern.) (Transitivitätsproblem)

Zur Implementation eigentümergesteuerter Zugriffskontrolle werden in Datenbanksystemen vor allem zwei Verfahren verwendet:

a. inhaltsbezogene Zugriffskontrolle mit Hilfe des **Sicht**enkonzepts (*view*) (System R u.a.).

Eine Sicht ist eine durch eine Anfrage beschriebene virtuelle Relation, die erst zur Laufzeit einer Anfrage bzgl. dieser Sicht ermittelt wird. Granule sind dann diese virtuellen Sicht-Relationen.

Der Benutzer erhält dann i.a. Erlaubnisse nur für Sichten und nicht für abgespeicherte (Basis-)Relationen.

Beispiel (SQL-like):
CREATE VIEW sales AS
SELECT empname, empno, mgr FROM employee WHERE job='Salesman';
GRANT select, update ON sales TO salesmgr;

Durch Sichten kann der Datenausschnitt, auf den zugegriffen werden darf, sowohl

• horizontal (durch geeignete Bedingungen in der WHERE-Klausel) wie auch

• vertikal (durch Auswahl der Attribute in der SELECT-Klausel) oder gar nur

• auf Aggregationen der Relation (durch Verwendung von Aggrationsfunktionen in der SELECT-Klausel) begrenzt werden.

Ein Benutzer mit Berechtigung für diese Sicht erhält dann die korrekte Antwort, ein Benutzer ohne Berechtigung erhält nur eine Fehlermeldung.

b. Prinzip der **Anfragemodifikation** (*query modification*) (Ingres u.a.).

Die Rechte jedes Benutzers werden als Beschränkungen in Form eines Prädikates in einer Systemtabelle gespeichert und die Bedingungs-(=Selektions-)klausel jeder Benutzeranfrage wird mit der Autorisierungsbedingung des Benutzers konjunktiv verknüpft.

Beispiel (INGRES-like):
RANGE OF e IS employee
PERMIT salesmgr TO employee FOR retrieve(e.empname, e.empno, e.mgr)
WHERE e.job='Salesman'

Dies hat allerdings zur Konsequenz, dass zwei Benutzer auf die gleiche Anfrage unterschiedliche Anfrageergebnisse erhalten können!

Beide Verfahren sind sehr flexibel und für die meisten Verfahren ausreichend. In der Praxis hat sich das erste Verfahren durchgesetzt.

Unter dem Aspekt der Sicherheit gibt es folgende Kritik an diesem Ansatz der Steuerung durch Eigentümer:

1. Es gibt keine Mechanismen, das Verhalten der Eigentümer an eine globalere Unternehmensstrategie zu binden.

   Das Unternehmen muss den **(allen!) Eigentümern vertrauen**, dass sie ggf. vorgegebene informelle Rechteweitergaberegeln des Unternehmens einhalten.

   (Die Kette reißt am schwächsten Glied!)

   Die Verantwortung für die Einhaltung einer Sicherheitspolitik liegt daher zwangsläufig bei den Datenbankbenutzern.

   Noch schwieriger stellt sich die Situation dar, wenn diese informellen Regeln geändert werden müssen.

2. Änderungen von Daten via Sichten können nicht in jedem Fall integritätserhaltend durchgeführt werden.

3. Eigentümergesteuerte Zugriffskontrolle schützt nicht vor unberechtigter Weitergabe von Autorisierungen mittels **Trojanischer Pferde**, solange sich nur die Subjekte authentifizieren.

4. Anfragen an Sichten bzw. Anfragemodifikation dauern u.U. länger.