

Sicherheitsaspekte

Sicherheit im DBMS

- Identifikation und Authentisierung
- Autorisierung und Zugriffskontrolle
- Auditing

Angriffsarten

- Missbrauch von **Autorität**
- **Inferenz** und **Aggregation**
- **Maskierung**
- Umgehung der **Zugriffskontrolle**
- **Browsing**
- **Trojanische Pferde**
- **Versteckte Kanäle**

Discretionary Access Control

Zugriffsregeln (o, s, t, p, f) mit

- $o \in O$, der Menge der **Objekte** (z.B. Relationen, Tupel, Attribute),
- $s \in S$, der Menge der **Subjekte** (z.B. Benutzer, Prozesse),
- $t \in T$, der Menge der **Zugriffsrechte** (z.B. $T = \{\text{lesen, schreiben, löschen}\}$),
- p ein **Prädikat** (z.B. $Rang = ,C4'$ für die Relation *Professoren*), und
- f ein **Boolescher Wert**, der angibt, ob s das **Recht** (o, t, p) an ein anderes Subjekt s' weitergeben darf.

Discretionary Access Control

Realisierung:

- Zugriffsmatrix
- Sichten
- „Query Modification“

Nachteile:

- Erzeuger der Daten = Verantwortlicher für deren Sicherheit

Zugriffskontrolle in SQL

Beispiel:

grant select

on Professoren

to eickler;

grant update (MatrNr, VorlNr, PersNr)

on prüfen

to eickler;

Zugriffskontrolle in SQL

Weitere Rechte:

- delete
- insert
- references

Weitergabe von Rechten:

- with **grant** option

Entzug von Rechten:

revoke update (MatrNr, VorlNr, PersNr)

on prüfen

from eickler **cascade**;

Sichten

Realisierung des Zugriffsprädikats:

```
create view ErstSemestler as  
  select *  
  from Studenten  
  where Semester = 1;  
grant select  
  on ErstSemestler  
  to tutor;
```

Schutz von Individualdaten durch Aggregation:

```
create view VorlesungsHärte (VorlNr, Härte) as  
  select VorlNr, avg(Note)  
  from prüfen  
  group by VorlNr;
```

Sichten: k-Anonymität

```
create view VorlesungsHärte (VorlNr, Härte) as  
  select VorlNr, avg(Note)  
  from prüfen  
  group by VorlNr  
  having count(*) > 11;
```

Individuelle Privilegien für eine Gruppe

```
CREATE VIEW StudentenNotenView AS  
SELECT * FROM pruefen p  
WHERE EXISTS (SELECT * FROM Studenten  
WHERE MatrNr = p.MatrNr AND Name = USER)
```

```
GRANT SELECT ON StudentenNotenView  
to <StudentenGruppe>
```

Auditing

Beispiele:

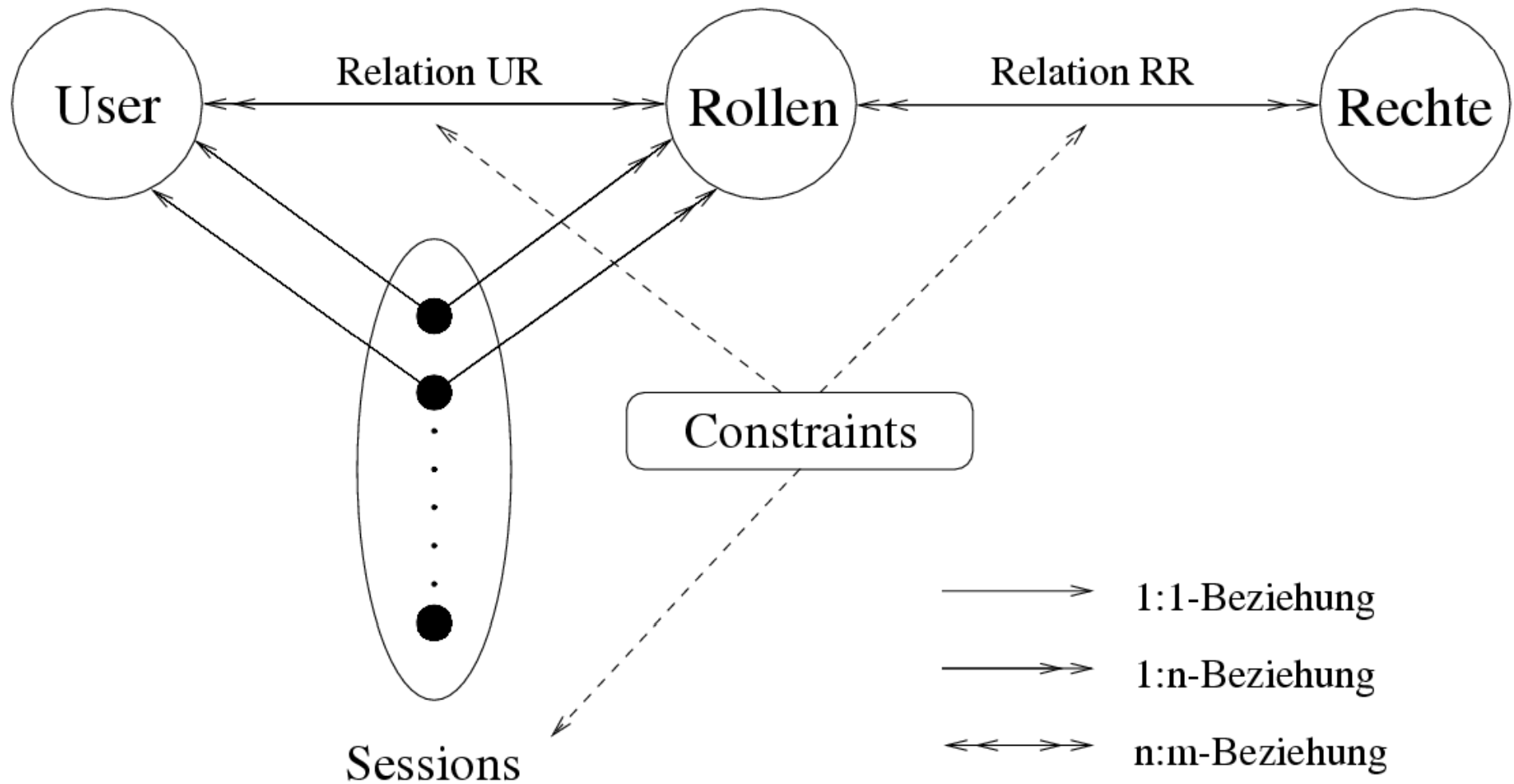
audit session by system

whenever not successful;

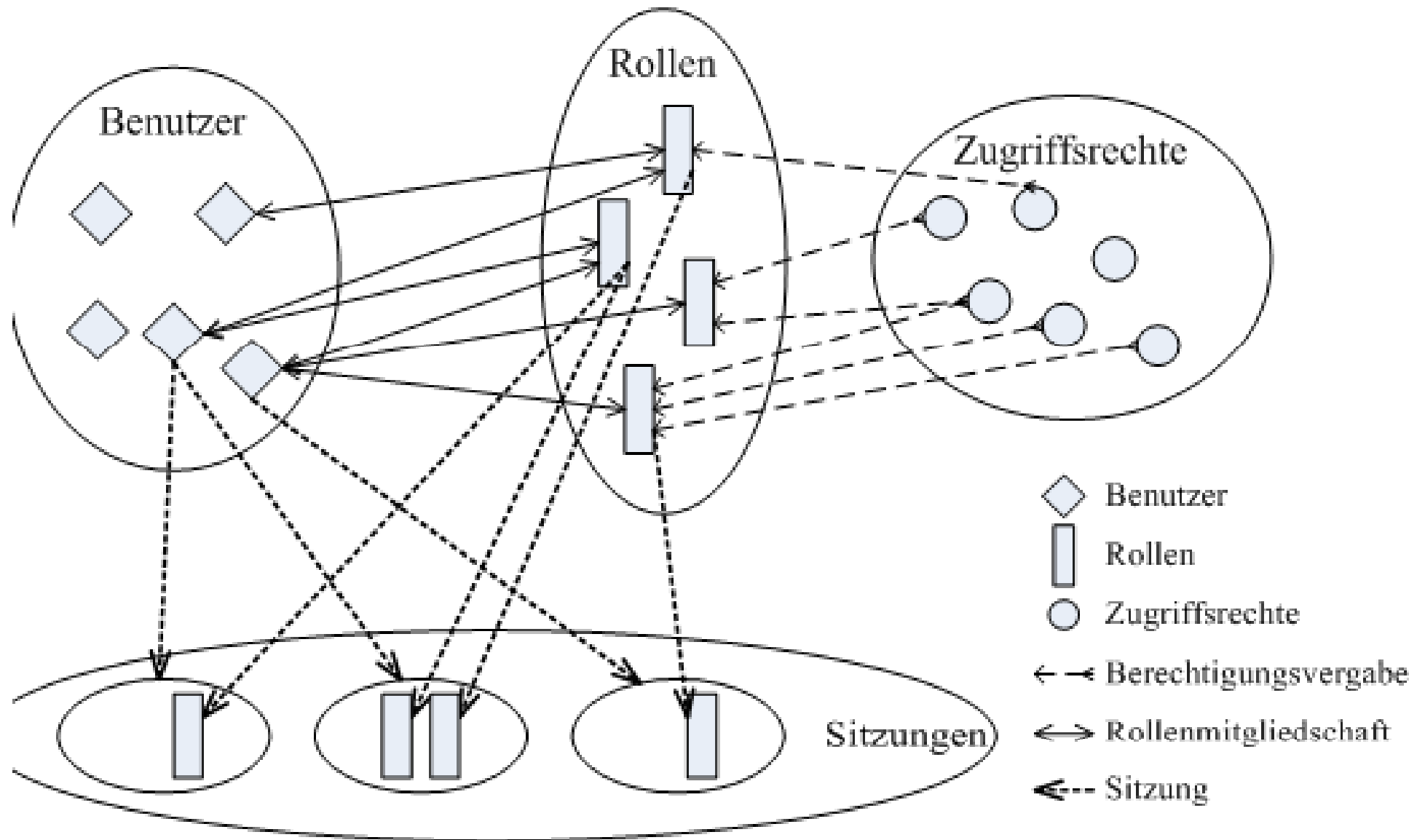
audit insert, delete, update on Professoren;

Rollenbasierte Autorisierung

RBAC: Role Based Access Control



RbAC: Role based Access Support



Verfeinerungen des Autorisierungsmodells

- explizite / implizite Autorisierung
- positive / negative Autorisierung
- starke / schwache Autorisierung

Autorisierungsalgorithmus:

wenn es eine explizite oder implizite starke Autorisierung (o, s, t) gibt,

dann erlaube die Operation

wenn es eine explizite oder implizite starke negative Autorisierung $(o, s, \neg t)$ gibt,

dann verbiete die Operation

ansonsten

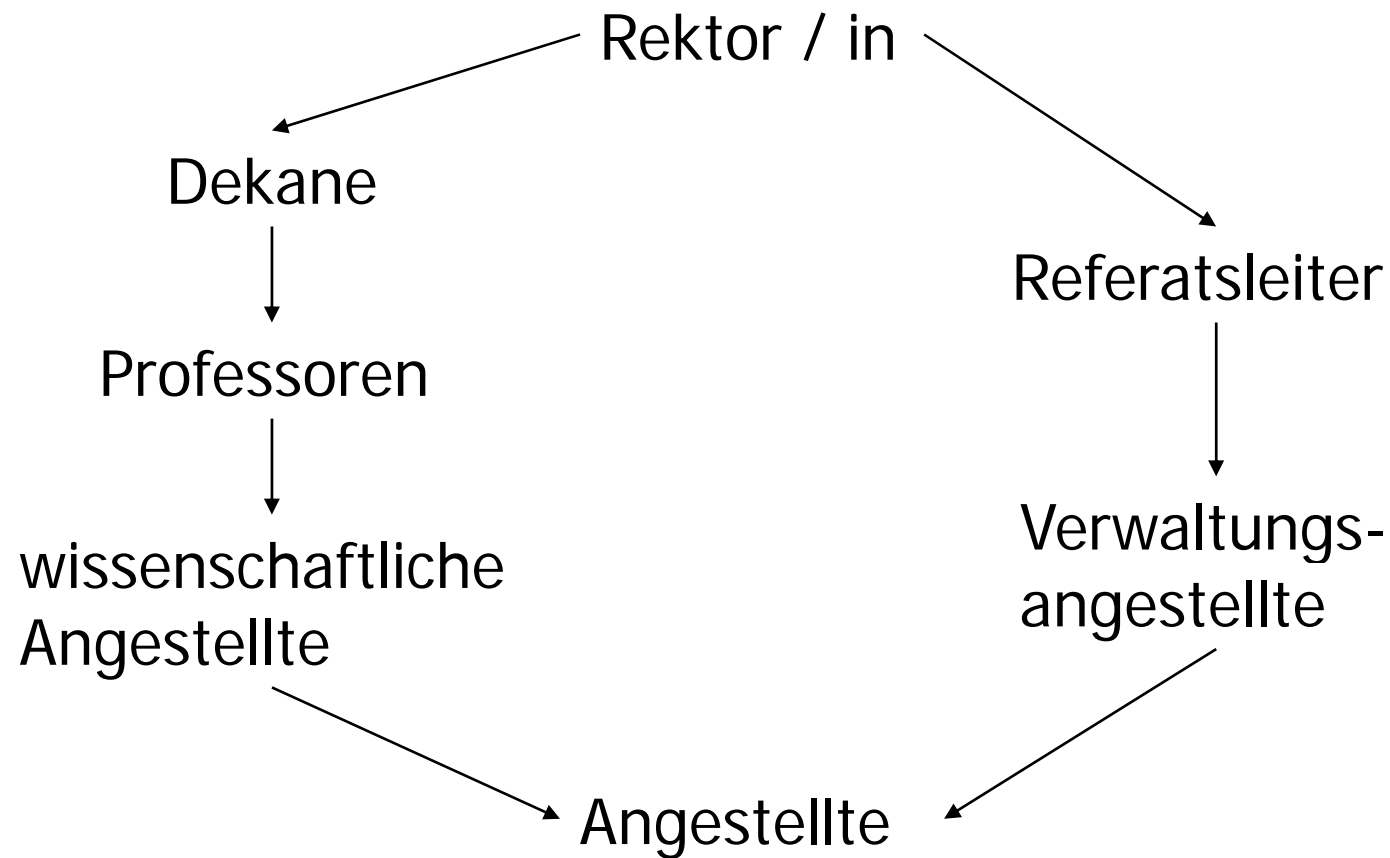
wenn es eine explizite oder implizite schwache Autorisierung $[o, s, t]$ gibt,

dann erlaube die Operation

wenn es eine explizite oder implizite schwache Autorisierung $[o, s, \neg t]$ gibt,

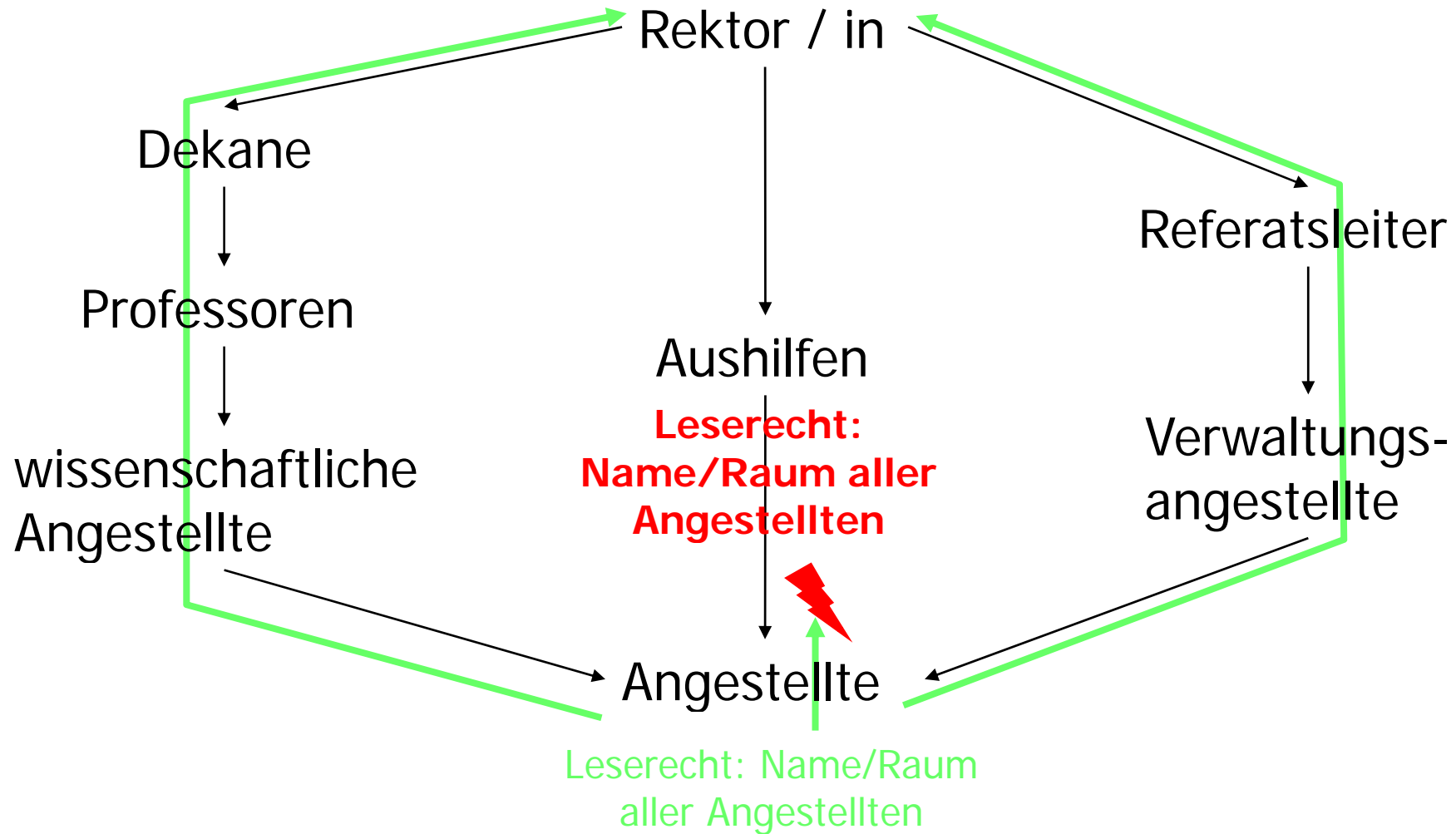
dann verbiete die Operation

Implizite Autorisierung von Subjekten



- explizite positive Autorisierung
 - ⇒ implizite positive Autorisierung auf allen *höheren* Stufen
- explizite negative Autorisierung
 - ⇒ implizite negative Autorisierung auf allen *niedrigeren* Stufen

Starke und schwache Autorisierung am Beispiel der Autorisierung von Subjekten



starke pos. Autorisierung

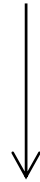
schwache pos. Autorisierung

starke neg. Autorisierung

schwache neg. Autorisierung

Implizite Autorisierung von Operationen

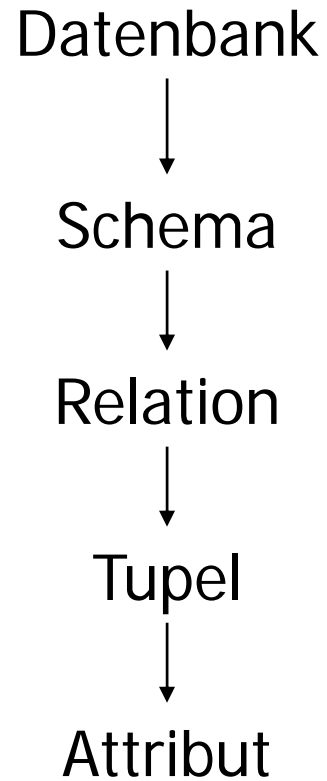
schreiben



lesen

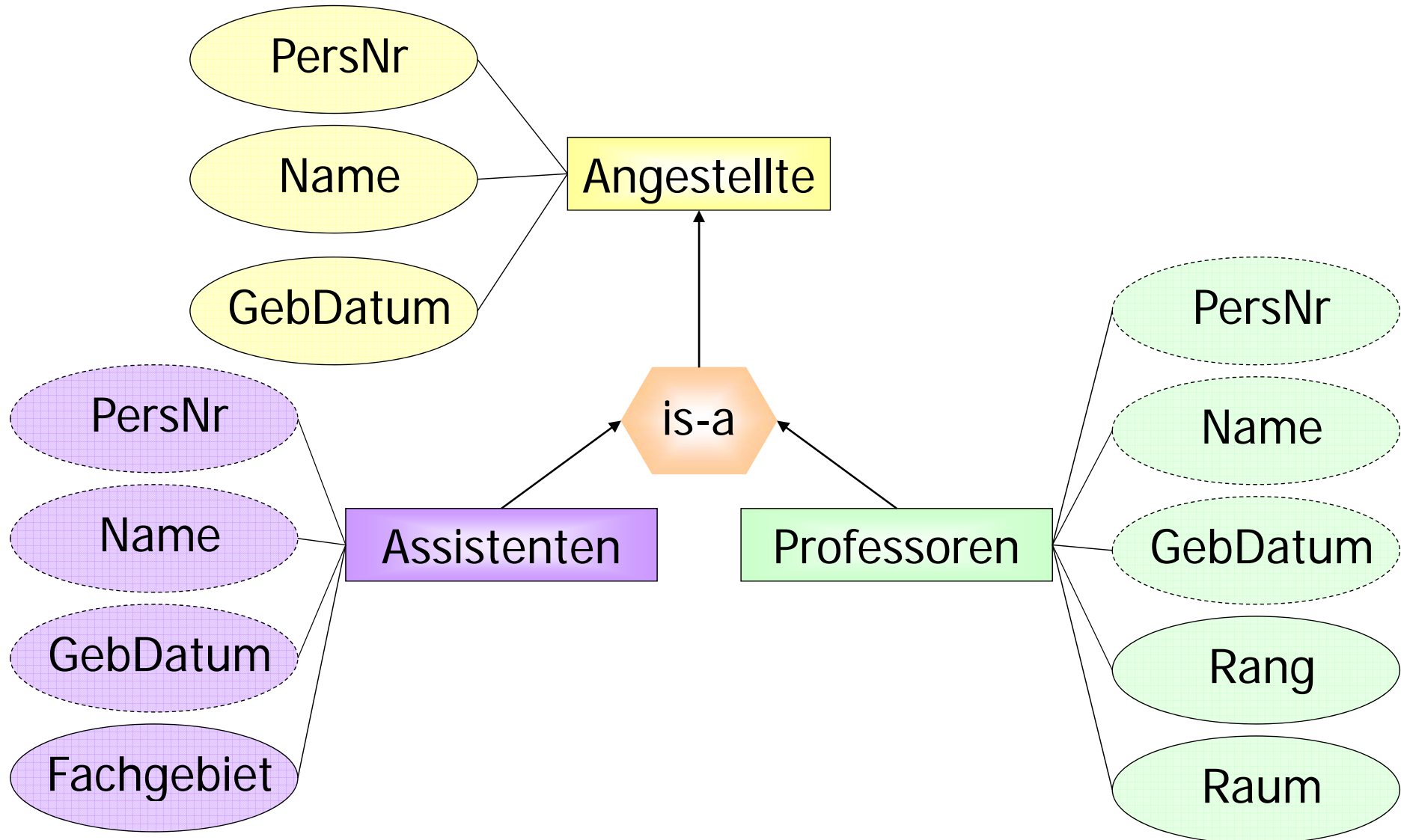
- explizite positive Autorisierung
 - ⇒ implizite positive Autorisierung auf allen *niedrigeren* Stufen
- explizite negative Autorisierung
 - ⇒ implizite negative Autorisierung auf allen *höheren* Stufen

Implizite Autorisierung von Objekten



- Implikationen abhängig von Operation

Implizite Autorisierung entlang einer Typhierarchie



Implizite Autorisierung entlang einer Typhierarchie

Benutzergruppen:

- Verwaltungsangestellte dürfen die Namen aller Angestellten lesen
- wissenschaftliche Angestellte dürfen Namen und Rang aller Professoren lesen

Anfragen:

- lese die Namen aller Angestellten
- lese Namen und Rang aller Professoren

Implizite Autorisierung entlang einer Typhierarchie

Regeln:

- Benutzer mit einem Zugriffsrecht auf einem Objekttypen haben auf die geerbten Attribute in den Untertypen ein gleichartiges Zugriffsrecht
- Ein Zugriffsrecht auf einen Objekttyp impliziert auch ein Zugriffsrecht auf alle von Obertypen geerbte Attribute in diesem Typ.
- Ein Attribut, das in einem Untertyp definiert wurde, ist nicht von einem Obertyp aus erreichbar.

Mandatory Access Control

- hierarchische Klassifikation von Vertrauenswürdigkeit und Sensitivität
- *clear(s)*, mit *s* Subjekt (*clearance*)
- *class(o)*, mit *o* Objekt (*classification*)
- Ein Subjekt *s* darf ein Objekt *o* nur lesen, wenn das Objekt eine geringere Sicherheitseinstufung besitzt ($class(o) \leq clear(s)$).
- Ein Objekt *o* muss mit mindestens der Einstufung des Subjektes *s* geschrieben werden ($clear(s) \leq class(o)$).

Multilevel-Datenbanken

- Benutzer soll sich der Existenz unzugänglicher Daten nicht bewusst sein

Beispiel (TC = Klassifizierung des gesamten Tupels = Maximum der Attributklassifizierungen):

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
sg	007	g	Blond, James	g	meucheln	sg
sg	008	sg	Mata, Harry	sg	spitzeln	sg

Sichtweise eines „geheim“ eingestuftem Benutzers:

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
g	007	g	Blond, James	g	-	g

Probleme:

- „geheimer“ Benutzer fügt Tupel mit Schlüssel „008“ ein
- „geheimer“ Benutzer modifiziert Spezialität von „007“

Multilevel-Relationen

Multilevel-Relation \mathcal{R} mit Schema

$$\mathcal{R} = \{A_1, C_1, A_2, C_2, \dots, A_n, C_n, TC\}$$

Relationeninstanzen \mathcal{R}_c mit Tupeln

$$[a_1, c_1, a_2, c_2, \dots, a_n, c_n, tc]$$

- $c \geq c_i$
- a_i ist sichtbar, wenn $class(s) \geq c_i$

Integritätsbedingungen

Sei κ *sichtbarer Schlüssel* der Multilevel-Relation R

Entity-Integrität. R erfüllt die Entity-Integrität genau dann, wenn für alle Instanzen R_c und $r \in R_c$ die folgende Bedingungen gelten:

1. $A_i \in \kappa \Rightarrow r.A_i \neq \text{Null}$
2. $A_i, A_j \in \kappa \Rightarrow r.C_i = r.C_j$
3. $A_i \notin \kappa \Rightarrow r.C_i \geq r.C_\kappa$ (wobei C_κ die Zugriffsklasse des Schlüssels ist)

Integritätsbedingungen

Sei κ *sichtbarer Schlüssel* der Multilevel-Relation R

Null-Integrität. R erfüllt die Null-Integrität genau dann, wenn für jede Instanz R_c von R gilt:

1. $\forall r \in R_c, r.A_j = \text{Null} \Rightarrow r.C_j = r.C_\kappa$
2. R_c ist subsumierungsfrei, d.h. es existieren keine zwei Tupel r und s , bei denen **für alle** Attribute A_j entweder
 - $r.A_j = s.A_j$ und $r.C_j = s.C_j$ oder
 - $r.A_j \neq \text{Null}$ und $s.A_j = \text{Null}$ gilt.

Subsumtionsfreiheit von Relationen

a) R_{sg}

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
g	007	g	Blond, James	g	-	g

b) Änderung von R_{sg}

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
sg	007	g	Blond, James	g	meucheln	sg

c) Fehlende Subsumtionsfreiheit

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
g	007	g	Blond, James	g	-	g
sg	007	g	Blond, James	g	meucheln	sg

Integritätsbedingungen

Interinstanz-Integrität. R erfüllt die Interinstanz-Integrität genau dann, wenn für alle Instanzen R_c und $R_{c'}$ von R mit $c' < c$

$$R_{c'} = f(R_c, c')$$

gilt. Die Filterfunktion f arbeitet wie folgt:

1. Für jedes $r \in R_c$ mit $r.C_k \leq c'$ muss ein Tupel $s \in R_{c'}$ existieren, mit

$$s.A_i = \begin{cases} r.A_i & \text{wenn } r.C_i \leq c' \\ \text{Null} & \text{sonst} \end{cases}$$

$$s.C_i = \begin{cases} r.C_i & \text{wenn } r.C_i \leq c' \\ r.C_k & \text{sonst} \end{cases}$$

2. $R_{c'}$ enthält außer diesen keine weiteren Tupel.
3. Subsumierte Tupel werden eliminiert.

Integritätsbedingungen

Polyinstanziierungsintegrität. R erfüllt die Polyinstanziierungsintegrität genau dann, wenn für jede Instanz R_c für alle a_i die folgende funktionale Abhängigkeit gilt:

$$\{\kappa, C_{\kappa}, C_j\} \rightarrow A_j.$$

SQL-Injection Attacken

- Hinter den meisten Web-Applikationen verbergen sich Datenbanksysteme
- Aus den Eingabe-Parametern werden SQL-Anfragen generiert
- Man darf diesen Eingabe-Parametern NIEMALS trauen, da sie „ausführbaren“ SQL-Code enthalten könnten

Naive Authentifizierung

Studenten			
MatrNr	Name	Semester	Passwort
24002	Xenokrates	18	AlterGrieche
25403	Jonas	12	Bruno
26120	Fichte	10	Idealismus
26830	Aristoxenos	8	Halbton
27550	Schopenhauer	6	WilleUndVorstellung
28106	Carnap	3	logischeAnalyse
29120	Theophrastos	2	Peripatos
29555	Feuerbach	2	Naturalismus

prüfen			
MatrNr	PersNr	VorlNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Mit jeder Anfrage wird das
Passwort übergeben

Select *

From Studenten s **join** prüfen p **on** s.MatrNr = p.MatrNr

Where s.Name = ... **and** s.Passwort = ...

Select *

From Studenten s **join** prüfen p **on** s.MatrNr = p.MatrNr

Where s.Name = 'Schopenhauer' **and**

s.Passwort = 'WilleUndVorstellung'

prüfen			
MatrNr	PersNr	VorlNr	Note
27550	4630	2137	2

Attacke ...

Name:

Passwort:

Select *

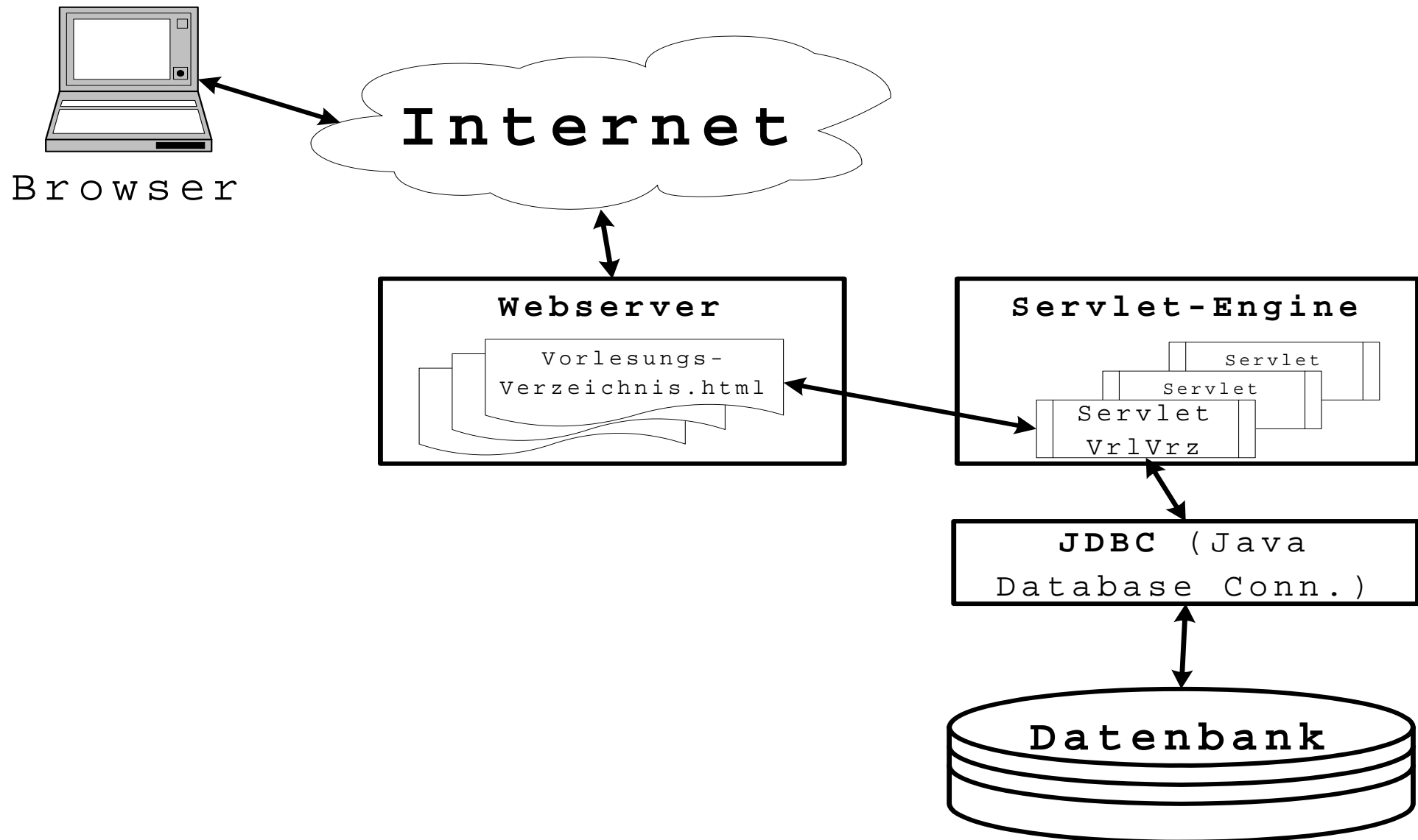
From Studenten s **join** prüfen p **on** s.MatrNr = p.MatrNr

Where s.Name = 'Schopenhauer' **and**

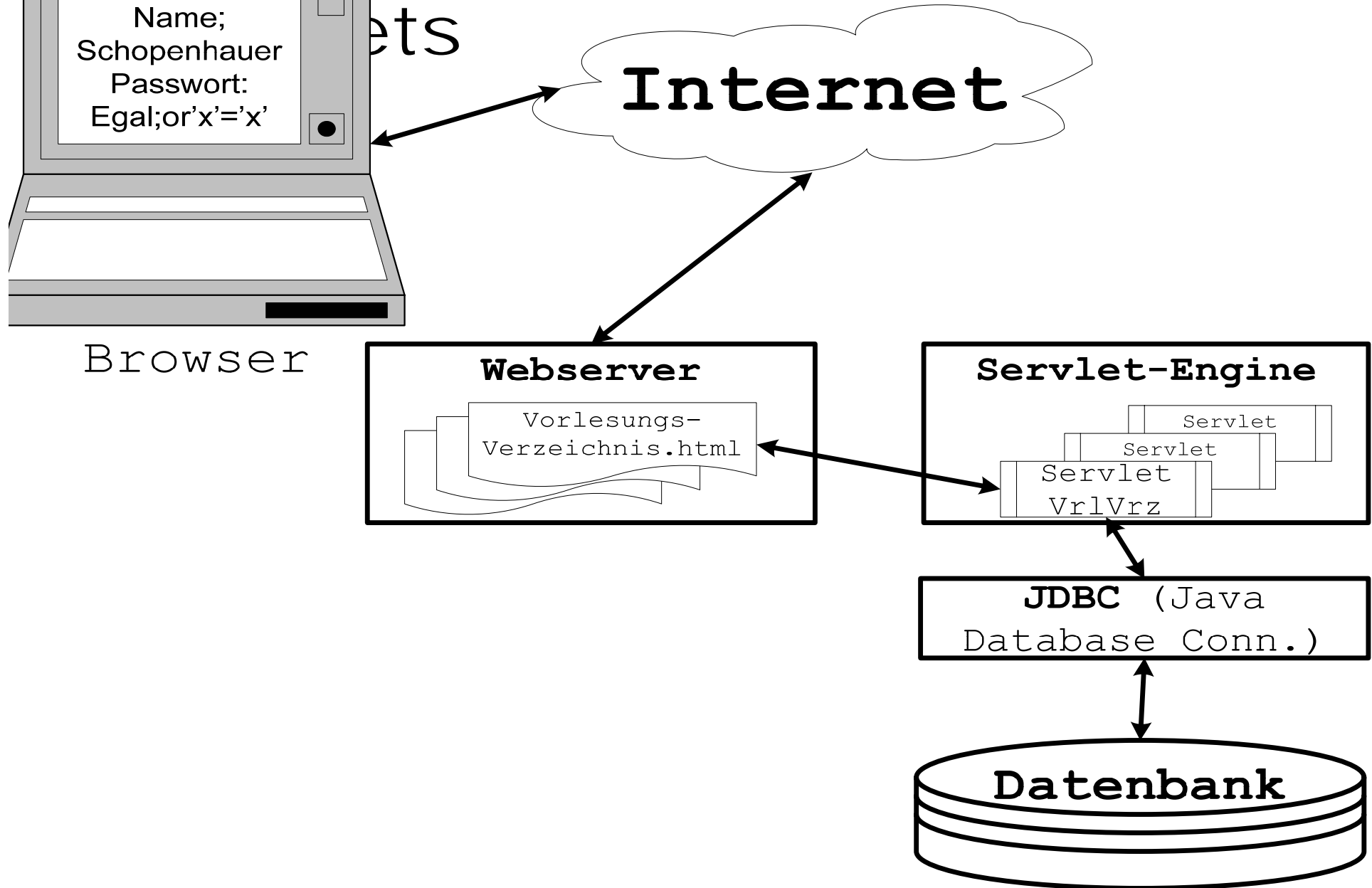
s.Passwort = 'WilleUndVorstellung' **or** 'x' = 'x'

prüfen			
MatrNr	PersNr	VorlNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Web-Anbindung von Datenbanken via Servlets



Web-Anbindung von Datenbanken



SQL-Injektion via Web-Schnittstelle

```
String _name = ... //Auslesen aus der Session etc = Benutzereingabe
```

```
String _pwd = ... // analog
```

```
String _query =
```

```
    "select * " +
```

```
    "from Studenten s join prüfen p on s.MatrNr = p.MatrNr" +
```

```
    "where s.Name = '" + _name +
```

```
        "' and s.Passwort = '" + _pwd + "';";
```

```
// initialisiere Connection c;
```

```
Statement stmt = c.createStatement;
```

```
ResultSet rs = stmt.execute(_query); // oder ähnlich;
```

Attacke ...

Name: Schopenhauer

Passwort: weissIchNichtAberEgal' or 'x' = 'x'

Select *

From Studenten s **join** prüfen p **on** s.MatrNr = p.MatrNr

Where s.Name = 'Schopenhauer' **and**

s.Passwort = 'weissIchNichtAberEgal' **or** 'x' = 'x'

prüfen			
MatrNr	PersNr	VorlNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Attacke ...

Name: Schopenhauer

Passwort: Egal'; delete from prüfen where 'x' = 'x'

Select *

From Studenten s **join** prüfen p **on** s.MatrNr = p.MatrNr

Where s.Name = 'Schopenhauer' **and**

s.Passwort = 'Egal'; **delete from** prüfen **where** 'x' = 'x'

prüfen			
MatrNr	PersNr	VorlNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Attacke ...

Name: Schopenhauer

Passwort: Egal'; update prüfen set Note = 1 where MatrNr = 25403;

Select *

From Studenten s **join** prüfen p **on** s.MatrNr = p.MatrNr

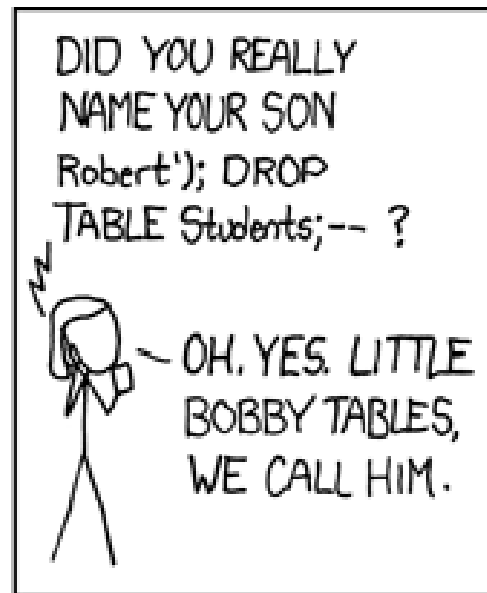
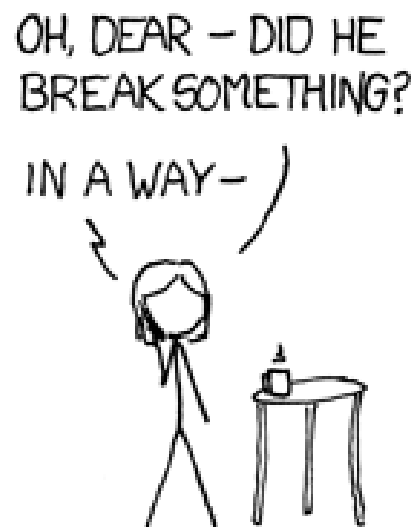
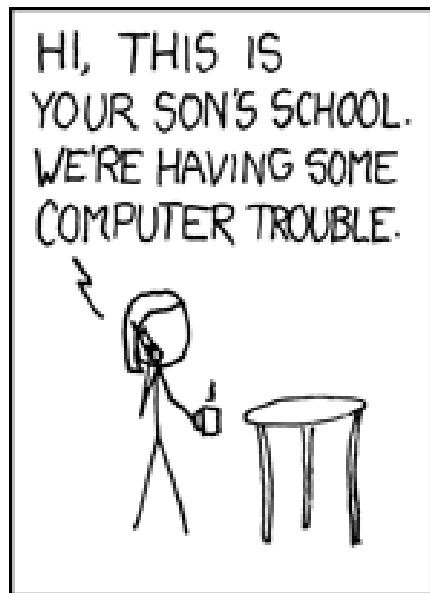
Where s.Name = 'Schopenhauer' **and**

s.Passwort = 'Egal'; **update** prüfen **set** Note = 1
where MatrNr = 25403;

prüfen			
MatrNr	PersNr	VorlNr	Note
28106	5001	2126	1
25403	5041	2125	1
27550	4630	2137	2

Karikatur

Quelle: xkcd



Schutz vor SQL-Injection-Attacken

- **Prepared Statements**

```
PreparedStatement stmt = conn.prepareStatement(  
    "select * from Vorlesungen v join Professoren p  
        on v.gelesenVon = p.PersNr  
    where v.Titel = ? and p.Name = ? ");
```

```
String einzulesenderTitel = "Logik";
```

```
String einzulesenderName = "Sokrates";
```

```
stmt.setString(1, einzulesenderTitel);
```

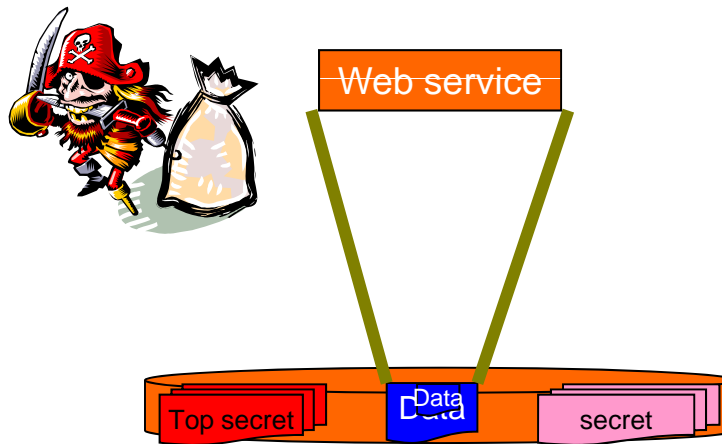
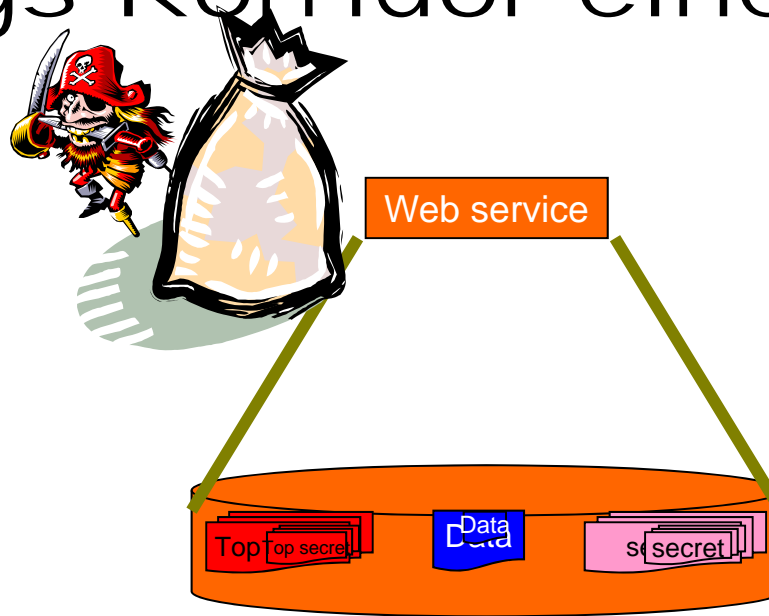
```
stmt.setString(2, einzulesenderName);
```

```
ResultSet rs = stmt.executeQuery();
```

Schutz vor SQL-Injection-Attacken

- Filterung der Eingabe-Parameter
- Restriktive Autorisierungskorridore für die Anwendungen

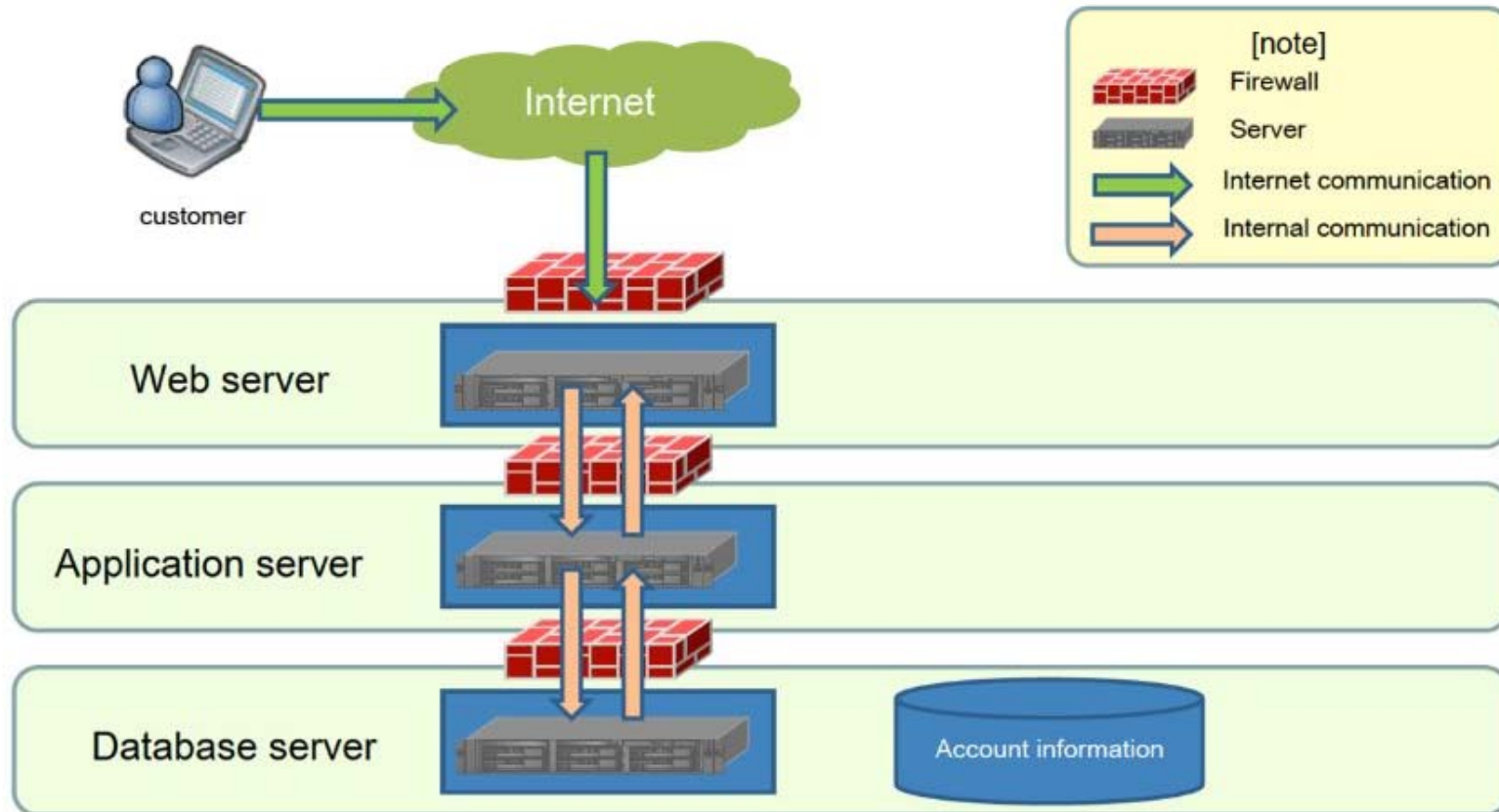
Autorisierungs-Korridor einer Web-Anwendung



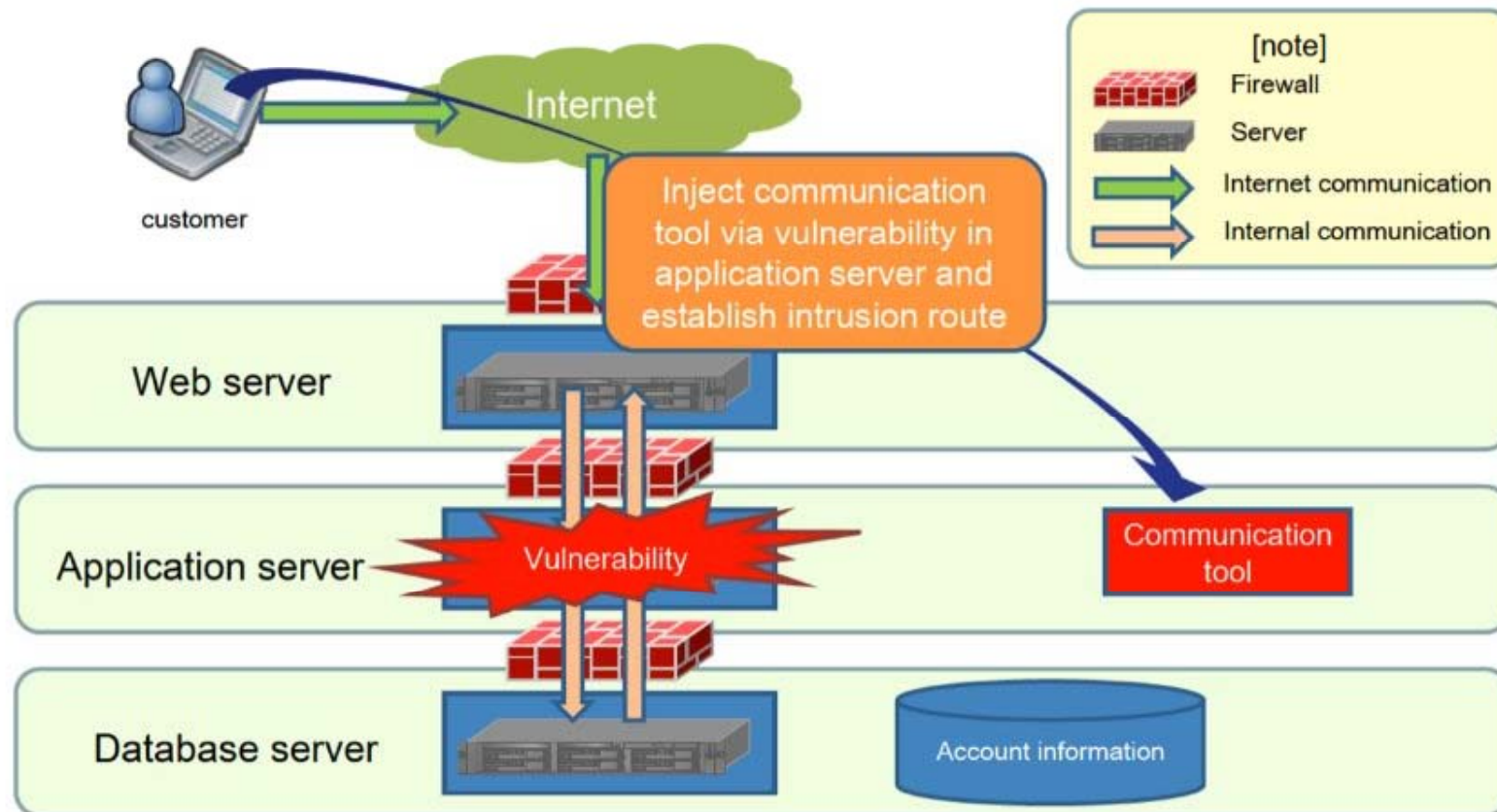
Sony Datendiebstahl

Quelle: Spiegel online

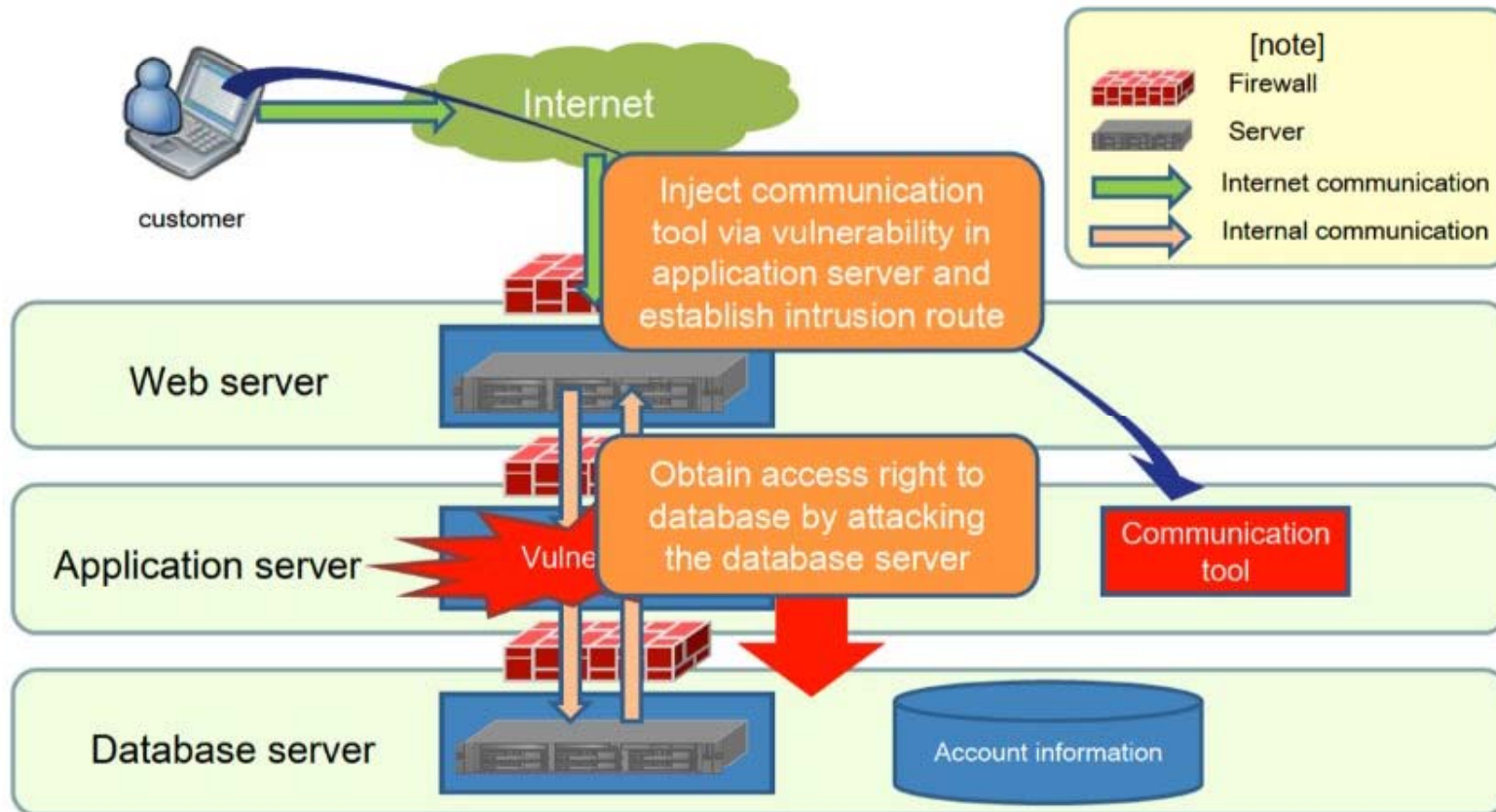
System Configuration



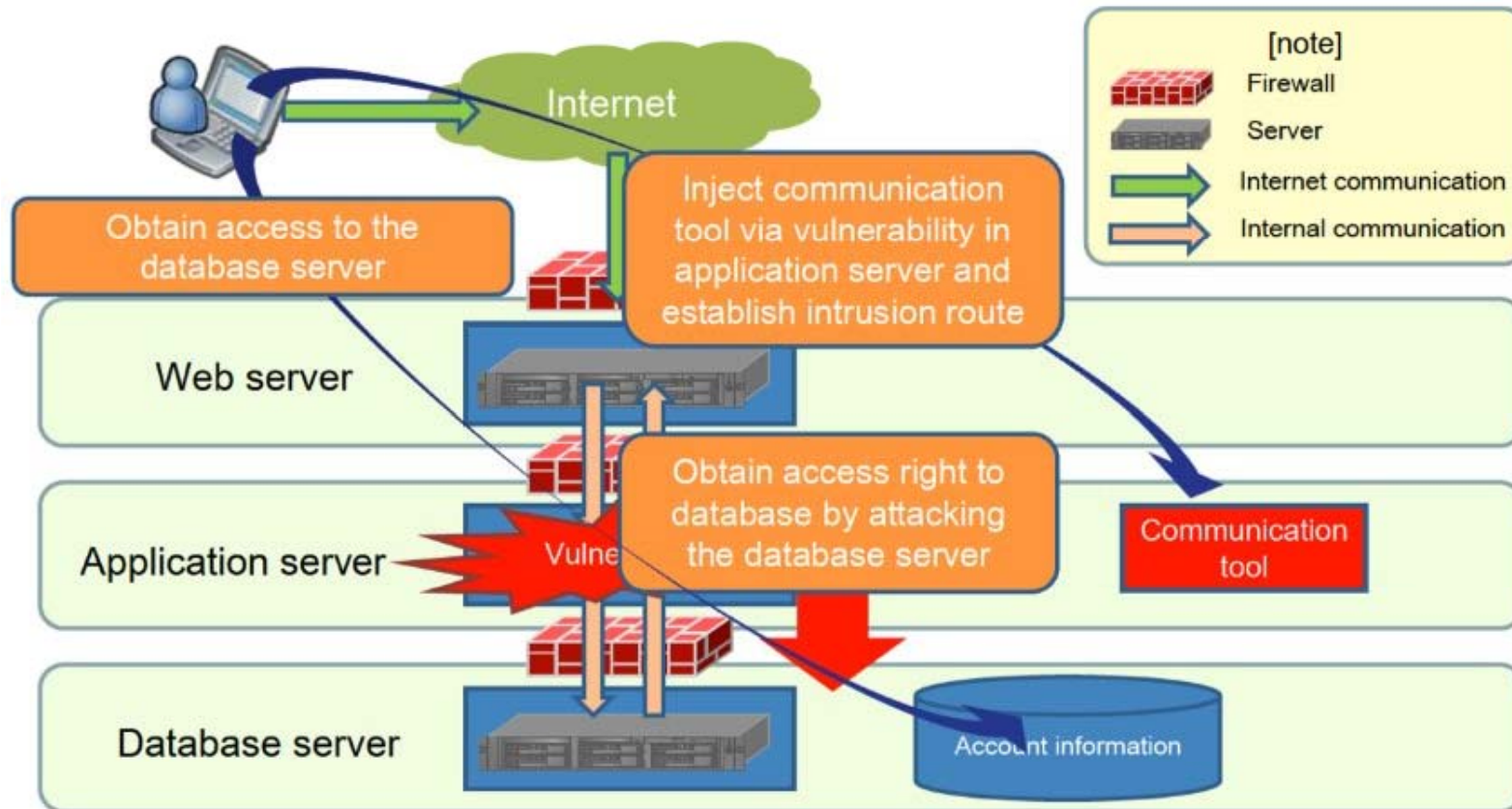
Intrusion route to the system



Intrusion route to the system



Intrusion route to the system

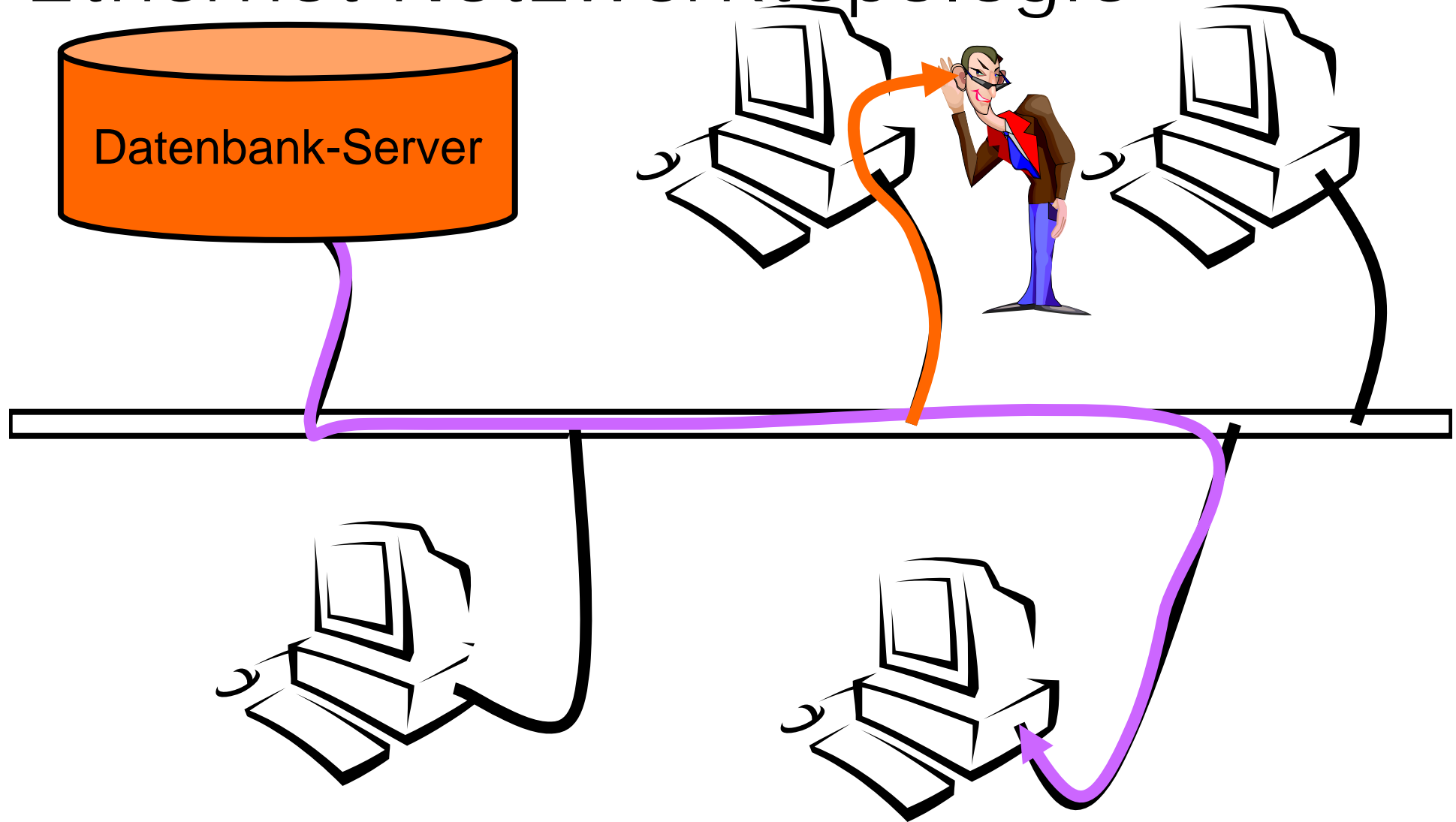


Kryptographie

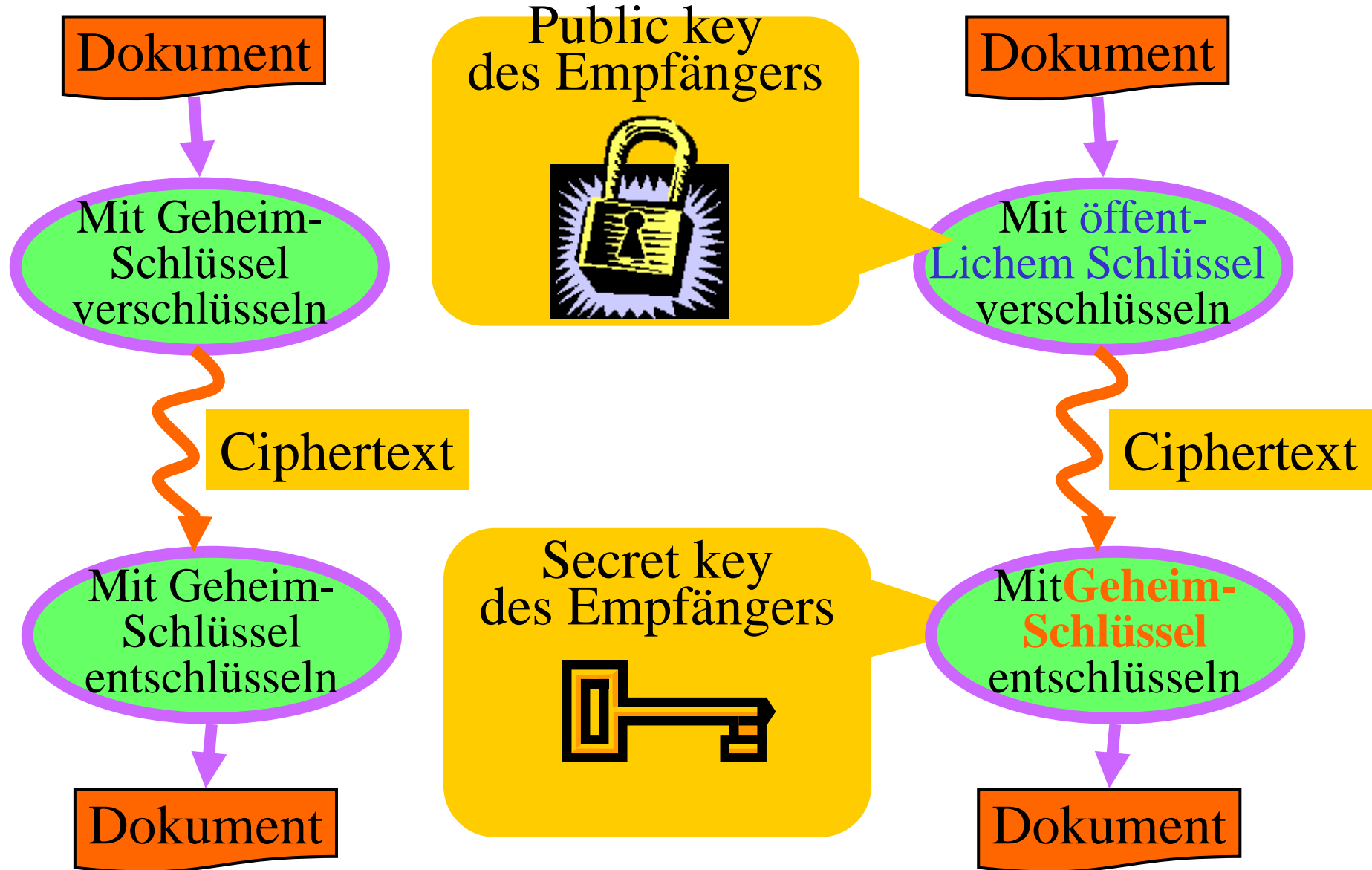
- Gerade die Gefahr des Abhörens von Kommunikationskanälen ist in heutigen Datenbankarchitekturen und Anwendungen sehr groß.
- Die meisten Datenbankanwendungen werden in einer verteilten Umgebung betrieben – sei es als Client / Server-System oder als „echte“ verteilte Datenbank.
- In beiden Fällen ist die Gefahr des unlegitimierten Abhörens sowohl innerhalb eines LAN (local area network, z.B. Ethernet) als auch im WAN (wide area network, z.B. Internet) gegeben und kann technisch fast nicht ausgeschlossen werden.
- Deshalb kann nur die Verschlüsselung der gesendeten Information einen effektiven Datenschutz gewährleisten.



Ethernet-Netzwerktopologie

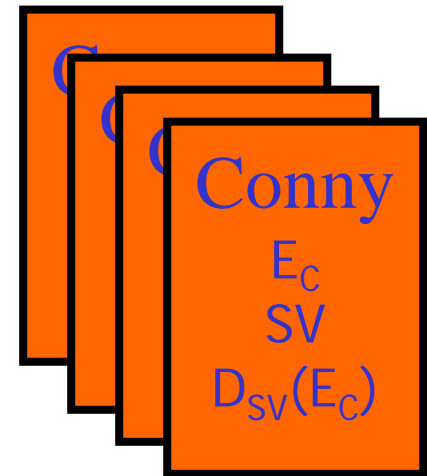


Geheimschlüssel vs Public Key

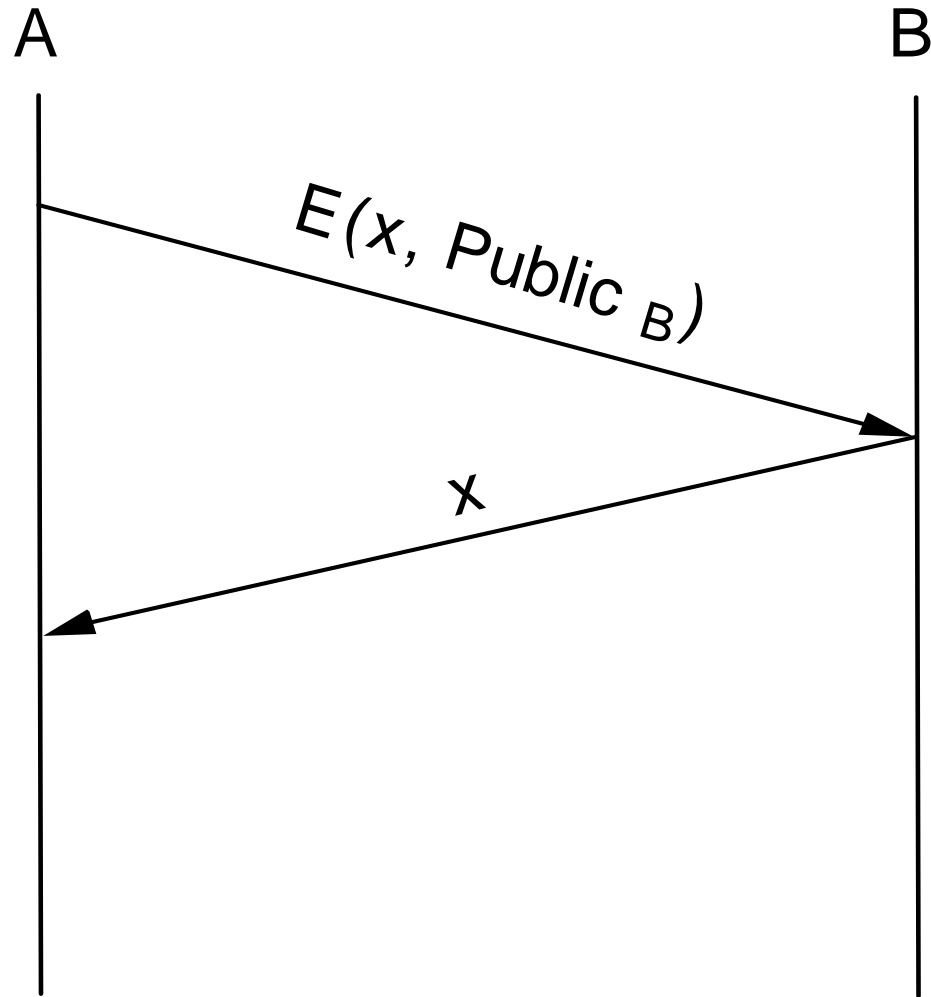


Verwaltung und Verteilung der öffentlichen Schlüssel

- X.509 – Standard
- Digitale Zertifikate
- Certification Authorities (CA)
 - Banken, Telekom, Firmen (Verisign, ...)
 - Ein Zertifikat von CA X ist nur für den sinnvoll, der den öffentlichen Schlüssel von X kennt
- Ein X.509 – Zertifikat enthält
 - Name der Organisation/Person: Conny
 - Öffentlichen Schlüssel: E_C
 - Name der Zertifizierungsautorität: SV
 - Digitale Signatur der CA: $D_{SV}(E_C)$
- Besitz eines Zertifikats sagt gar nichts aus
 - Zertifikate werden kopiert, gepuffert, etc.
 - Nur der Besitz des zugehörigen geheimen Schlüssels authentifiziert den rechtmäßigen Besitzer
- Hierarchie von CAs: X zertifiziert Y zertifiziert Z
 - Wenn ich X kenne kann ich dem Zertifikat für Z trauen, ohne Y zu kennen



- Public Key Authentifizierung



Das RSA-Verfahren

- Rivest, Shamir und Adleman (1978)
 - das älteste “*public key*”-Kryptographieverfahren
 - beruht auf der “Erfahrung”, daß Faktorisierung ein “hartes” Problem ist
-
- **Verschlüsselung einer Nachricht M**

öffentlicher Schlüssel: $(e, n) \approx E$

1. repräsentiere die Nachricht M als natürliche Zahl, so daß gilt: $0 \leq M \leq n - 1$
 - längere Nachrichten sind entsprechend aufzuspalten
2. berechne $C = E(M) := M^e \bmod n$

• Verschlüsselung einer Nachricht M

öffentlicher Schlüssel: $(e, n) \approx E$

1. repräsentiere die Nachricht M als natürliche Zahl, so daß gilt: $0 \leq M \leq n - 1$
 - längere Nachrichten sind entsprechend aufzuspalten

2. berechne $C = E(M) := M^e \bmod n$

$$M^{14}$$

$$14 = \overbrace{(1110)}^E$$

$$(M^2 * M) \bmod n$$

$$(M^3)^2 * M \bmod n$$

$$(M^7)^2 = M^{14} \bmod n$$

• Entschlüsselung von $C = E(M)$

geheimer Schlüssel: $(d, n) \approx D$

$$D(C) := C^d \bmod n$$

Auswahl der Schlüssel

- öffentlicher Schlüssel: (e, n)
- geheimer Schlüssel: (d, n)

1. Wähle zwei (sehr große) Primzahlen p und q
 - mindestens 100-stellig
 - zufällig ausgewählt
(etwa jede 115-te ungerade 100-stellige Zahl ist prim)
2. Berechne $n := p * q$
3. Wähle eine “große” Zahl d , für die gilt:

$$\text{ggT}(d, \underbrace{(p-1) * (q-1)}_{\phi(n)}) = 1$$

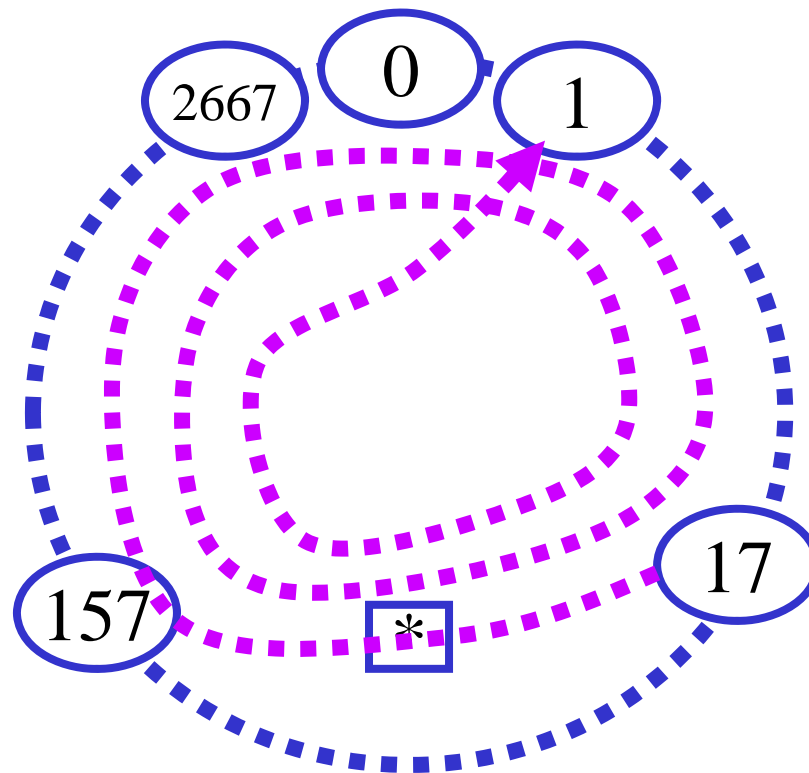
- man wähle z.B. eine Primzahl $d > \max(p, q)$
4. Berechne e , so daß gilt:
$$e * d \equiv 1 \pmod{(p-1) * (q-1)}$$
 - e ist das “multiplikative Inverse” von d
(im Ring $\mathbf{Z}_{\phi(n)}, +_{\phi(n)}, *_{\phi(n)}$)
 - e existiert und ist eindeutig
(weil $\text{ggT}(d, \phi(n)) = 1$)

Illustration von $e=157$ und $d=17$ im Zahlenring \mathbb{Z}_{2668}^*

$$2668=46 \cdot 58$$

$$P=47$$

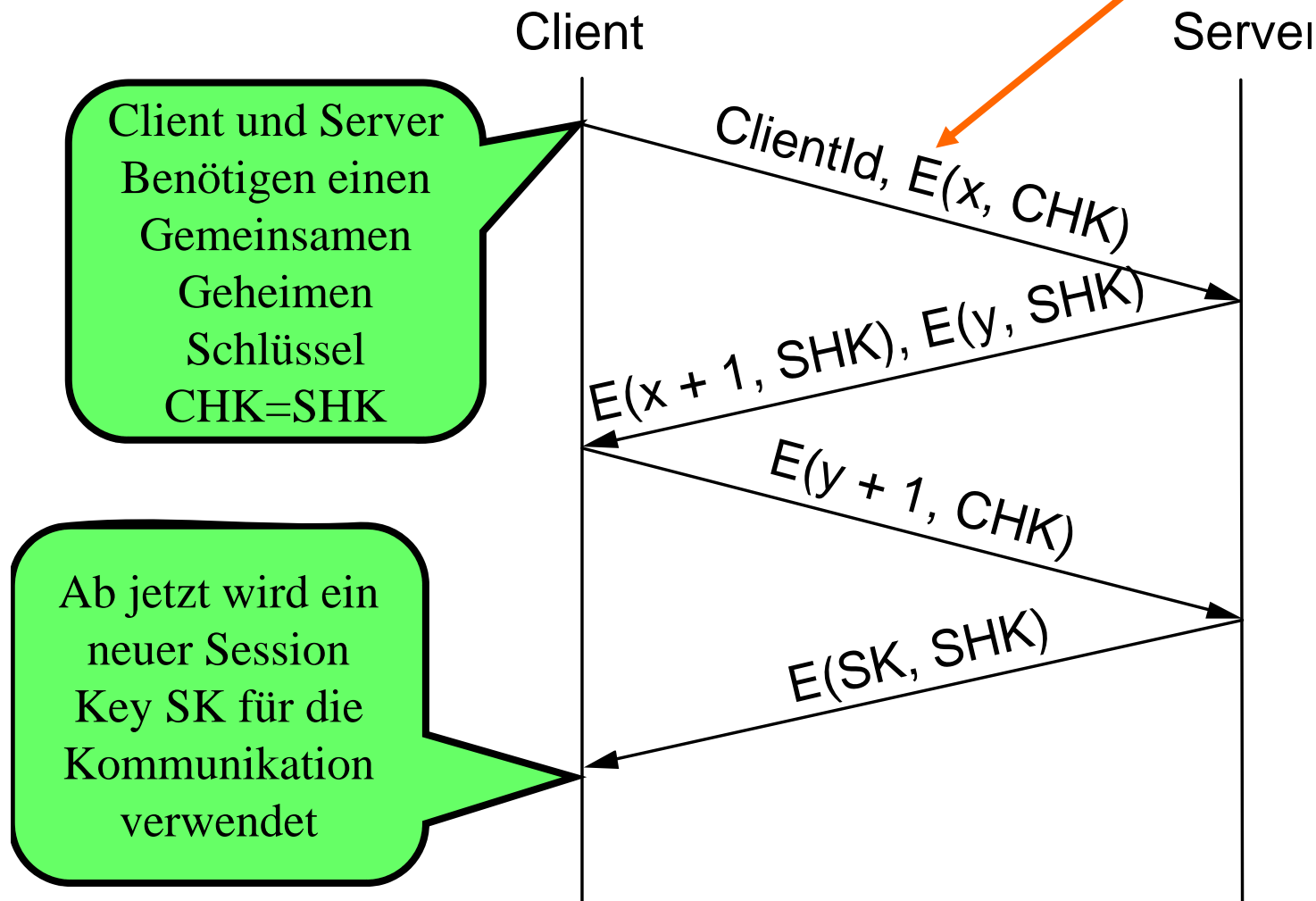
$$q=59$$



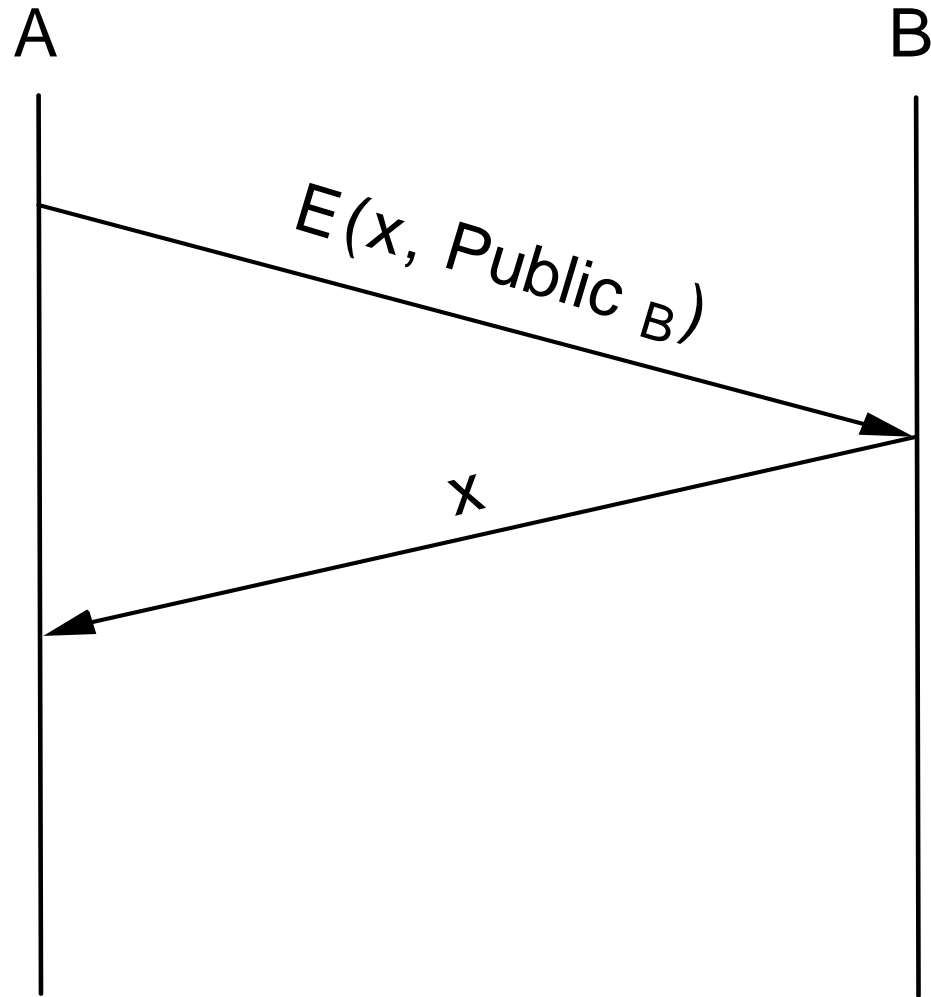
Authentifizierungs-Protokolle

- Three-way handshake

Encode mit
Schlüssel CHK



- Public Key Authentifizierung



Ebenen des Datenschutzes

legislative Maßnahmen

organisatorische Maßnahmen

Authentisierung

Zugriffskontrolle

Kryptographie

Datenbank

