Kapitel 15

Sicherheit

In diesem Kapitel geht es um den Schutz gegen absichtliche Beschädigung oder Enthüllung von sensiblen Daten. Abbildung 15.1 zeigt die hierarchische Kapselung verschiedenster Maßnahmen.

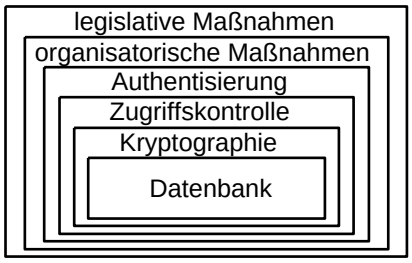

Abbildung 15.1: Ebenen des Datenschutzes

15.1 Legislative Maßnahmen

Im *Gesetz zum Schutz vor Mißbrauch personenbezogener Daten bei der Datenverarbeitung* ist festgelegt, welche Daten in welchem Umfang schutzbedürftig sind.

15.2 Organisatorische Maßnahmen

Darunter fallen Maßnahmen, um den persönlichen Zugang zum Computer zu regeln:

- bauliche Maßnahmen
- Pförtner
- Ausweiskontrolle
- Diebstahlsicherung
- Alarmanlage

15.3 Authentisierung

Darunter fallen Maßnahmen zur Überprüfung der Identität eines Benutzers:

- Magnetkarte
- Stimmanalyse/Fingerabdruck
- Paßwort: w ohne Echo eintippen,
  System überprüft, ob $f(w)$ eingetragen ist, f^{-1} aus f nicht rekonstruierbar
- dynamisches Paßwort: vereinbare Algorithmus, der aus Zufallsstring gewisse Buchstaben heraussucht

Paßwortverfahren sollten mit Überwachungsmaßnahmen kombiniert werden (Ort, Zeit, Fehleingabe notieren)

15.4 Zugriffskontrolle

Verschiedene Benutzer haben verschiedene Rechte bzgl. derselben Datenbank. Tabelle 14.1 zeigt eine Berechtigungsmatrix (wertunabhängig):

Benutzer	Ang-Nr	Gehalt	Leistung
A (Manager)	R	R	RW
B (Personalchef)	RW	RW	R
C (Lohnbüro)	R	R	—

Tabelle 14.1: Berechtigungsmatrix

Bei einer wertabhängigen Einschränkung wird der Zugriff von der aktuellen Ausprägung abhängig gemacht:

$$\begin{array}{lrlll} \text{Zugriff } (A, \text{Gehalt}): & R\text{:} & \text{Gehalt} & < & 10.000 \\ & W\text{:} & \text{Gehalt} & < & 5.000 \end{array}$$

Dies ist natürlich kostspieliger, da erst nach Lesen der Daten entschieden werden kann, ob der Benutzer die Daten lesen darf. Ggf. werden dazu Tabellen benötigt, die für die eigentliche Anfrage nicht verlangt waren. Beispiel: Zugriff verboten auf Gehälter der Mitarbeiter an Projekt 007.

Eine Möglichkeit zur Realisierung von Zugriffskontrollen besteht durch die Verwendung von Sichten:

```
define view v(angnr, gehalt) as
select angnr, gehalt from angest
where gehalt < 3000
```

Eine andere Realisierung von Zugriffskontrollen besteht durch eine Abfragemodifikation.

- **Beispiel:**
  Die Abfrageeinschränkung

  ```
  deny (name, gehalt) where gehalt > 3000
  ```

  liefert zusammen mit der Benutzer-Query

  ```
  select gehalt from angest where name = 'Schmidt'
  ```

  die generierte Query

  ```
  select gehalt from angest
  where name = 'Schmidt' and not gehalt > 3000
  ```

In statistischen Datenbanken dürfen Durchschnittswerte und Summen geliefert werden, aber keine Aussagen zu einzelnen Tupeln. Dies ist sehr schwer einzuhalten, selbst wenn die Anzahl der referierten Datensätze groß ist.

- **Beispiel:**
  Es habe Manager X als einziger eine bestimmte Eigenschaft, z. B. habe er das höchste Gehalt. Dann läßt sich mit folgenden beiden Queries das Gehalt von Manager X errechnen, obwohl beide Queries alle bzw. fast alle Tupel umfassen:

  ```
  select sum (gehalt) from angest;
  select sum (gehalt) from angest
  where gehalt < (select max(gehalt) from angest);
  ```

In SQL-92 können Zugriffsrechte dynamisch verteilt werden, d. h. der Eigentümer einer Relation kann anderen Benutzern Rechte erteilen und entziehen.

Die vereinfachte Syntax lautet:

```
grant { select | insert | delete | update | references | all }
on <relation> to <user> [with grant option]
```

Hierbei bedeuten

`select`:	darf Tupel lesen
`insert`:	darf Tupel einfügen
`delete`:	darf Tupel löschen
`update`:	darf Tupel ändern
`references`:	darf Fremdschlüssel anlegen
`all` :	`select + insert + delete + update + references`
`with grant option`:	`<user>` darf die ihm erteilten Rechte weitergeben

- **Beispiel:**
  A: `grant read, insert on angest to` B `with grant option`
  B: `grant read on angest to` C `with grant option`
  B: `grant insert on angest to` C

Das Recht, einen Fremdschlüssel anlegen zu dürfen, hat weitreichende Folgen: Zum einen kann das Entfernen von Tupeln in der referenzierten Tabelle verhindert werden. Zum anderen kann durch das probeweise Einfügen von Fremdschlüsseln getestet werden, ob die (ansonsten lesegeschützte) referenzierte Tabelle gewisse Schlüsselwerte aufweist:

```
create table Agententest(Kennung character(4) references Agenten);
```

Jeder Benutzer, der ein Recht vergeben hat, kann dieses mit einer *Revoke*-Anweisung wieder zurücknehmen:

```
revoke { select | insert | delete | update | references | all }
on <relation> from <user>
```

- **Beispiel:**
  B: `revoke all on angest from` C
  Es sollen dadurch dem Benutzer C alle Rechte entzogen werden, die er von B erhalten hat, aber nicht solche, die er von anderen Benutzern erhalten hat. Außerdem erlöschen die von C weitergegebenen Rechte.

Der Entzug eines Grant `G` soll sich so auswirken, als ob `G` niemals gegeben worden wäre!

- **Beispiel:**
  A: `grant read, insert, update on angest to` D
  B: `grant read, update on angest to` D `with grant option`
  D: `grant read, update on angest to` E
  A: `revoke insert, update on angest from` D

  Hierdurch verliert D sein `insert`-Recht, E verliert keine Rechte. Falls aber vorher A Rechte an B gab, z.B. durch

  A: `grant all on angest to` B `with grant option`

  dann müssten D und E ihr *update*-Recht verlieren.

15.5 Auditing

Auditing bezeichnet die Möglichkeit, über Operationen von Benutzern Buch zu führen. Einige (selbsterklärende) Kommandos in SQL-92:

```
audit delete any table;
noaudit delete any table;
audit update on erika.professoren whenever not successful;
```

Der resultierende *Audit-Trail* wird in diversen Systemtabellen gehalten und kann von dort durch spezielle Views gesichtet werden.

15.6 Kryptographie

Da die meisten Datenbanken in einer verteilten Umgebung (Client/Server) betrieben werden, ist die Gefahr des Abhörens von Kommunikationskanälen sehr hoch. Zur Authentisierung von Benutzern und zur Sicherung gegen den Zugriff auf sensible Daten werden daher *kryptographische Methoden* eingesetzt.

Der prinzipielle Ablauf ist in Abbildung 15.2 skizziert: Der Klartext x dient als Eingabe für ein Verschlüsselungsverfahren *encode*, welches über einen Schlüssel e parametrisiert ist. Das heißt, das grundsätzliche Verfahren der Verschlüsselung ist allen Beteiligten bekannt, mit Hilfe des Schlüssels e kann der Vorgang jeweils individuell beeinflußt werden. Auf der Gegenseite wird mit dem Verfahren *decode* und seinem Schlüssel d der Vorgang umgekehrt und somit der Klartext rekonstruiert.

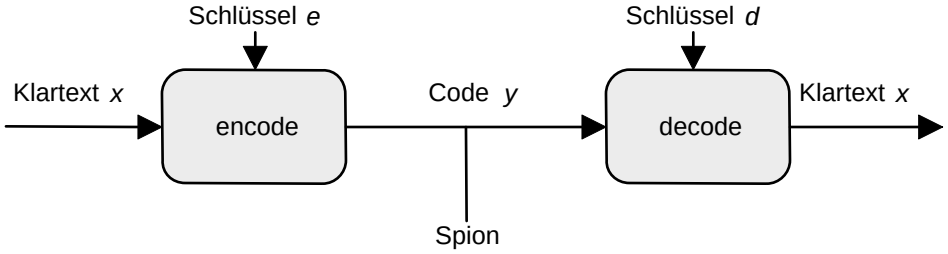

Abbildung 15.2: Ablauf beim Übertragen einer Nachricht

Zum Beispiel kann eine Exclusive-OR-Verknüpfung des Klartextes mit dem Schlüssel verwendet werden, um die Chiffre zu berechnen. Derselbe Schlüssel erlaubt dann die Rekonstruktion des Klartextes.

Klartext	`010111001`	
Schlüssel	`111010011`	
Chiffre	`101101010`	= Klartext $\oplus$ Schlüssel
Schlüssel	`111010011`	
Klartext	`010111001`	= Chiffre $\oplus$ Schlüssel

Diese Technik funktioniert so lange gut, wie es gelingt, die zum Bearbeiten einer Nachricht verwendeten Schlüssel e und d auf einem sicheren Kanal zu übertragen, z. B. durch einen Kurier. Ein Spion, der ohne Kenntnis der Schlüssel die Leitung anzapft, ist dann nicht in der Lage, den beobachteten Code zu entschlüsseln (immer vorausgesetzt, der Raum der möglichen Schlüssel wurde zur Abwehr eines vollstängigen Durchsuchens groß genug gewählt). Im Zeitalter der globalen Vernetzung besteht natürlich der Wunsch, auch die beiden Schlüsselpaare e und d per Leitung auszutauschen. Nun aber laufen wir Gefahr, daß der Spion von ihnen Kenntnis erhält und damit den Code knackt.

Dieses (auf den ersten Blick) unlösbare Problem wurde durch die Einführung von *Public Key Systems* behoben.

15.6.1 Public Key Systems

Gesucht sind zwei Funktionen $enc,\ dec : \mathbb{N} \to \mathbb{N}$ mit folgender Eigenschaft:

1. $dec((enc(x)) = x$
2. effizient zu berechnen
3. aus der Kenntnis von enc läßt sich dec nicht effizient bestimmen

Unter Verwendung dieser Funktionen könnte die Kommunikation zwischen den Partner Alice und Bob wie folgt verlaufen:

1. Alice möchte Bob eine Nachricht schicken.
2. Bob veröffentlicht sein enc_B.
3. Alice bildet $y := enc_B(x)$ und schickt es an Bob.
4. Bob bildet $x := dec_B(y)$.

15.6.2 Das RSA-Verfahren

Im Jahre 1978 schlugen Rivest, Shamir, Adleman folgendes Verfahren vor:

geheim:	Wähle zwei große Primzahlen $p,\ q$ (je 500 Bits)
öffentlich:	Berechne $n := p \cdot q$
geheim:	Wähle d teilerfremd zu $\varphi(n) = (p-1) \cdot (q-1)$
öffentlich:	Bestimme d^{-1}, d.h. e mit $e \cdot d \equiv 1 \bmod \varphi(n)$
öffentlich:	enc(x):= $x^e \bmod n$
geheim:	dec(y):= $y^d \bmod n$

- **Beispiel:**
  $p = 11,\ q = 13,\ d = 23\ \Rightarrow$
  $n = 143,\ e = 47$
  $enc(x) := x^{47} \bmod 143$
  $dec(y) := y^{23} \bmod 143$

15.6.3 Korrektheit des RSA-Verfahrens

Die Korrektheit stützt sich auf den **Satz von Fermat/Euler:**

$$x \text{ rel. prim zu } n \Rightarrow x^{\varphi(n)} \equiv 1 \bmod n$$

15.6.4 Effizienz des RSA-Verfahrens

Die Effizienz stützt sich auf folgende Überlegungen:

a) **Potenziere mod** n

Nicht e-mal mit x malnehmen, denn Aufwand wäre $O(2^{500})$, sondern:

$$x^e := \begin{cases} (x^{e/2})^2 & \text{falls } e \text{ gerade} \\ (x^{\lfloor e/2 \rfloor})^2 \cdot x & \text{falls } e \text{ ungerade} \end{cases}$$

Aufwand: $O\ (\log e)$, d.h. proportional zur Anzahl der Dezimalstellen.

b) **Bestimme** $e := d^{-1}$

Algorithmus von Euklid zur Bestimmung des ggt:

$$ggt(a,b) := \begin{cases} a & \text{falls } b = 0 \\ ggt(b,\ a \bmod b) & \text{sonst} \end{cases}$$

Bestimme $ggt(\varphi(n),\ d)$ und stelle den auftretenden Rest als Linearkombination von $\varphi(n)$ und d dar.

Beispiel:

$$\begin{array}{rcrcl} & & 120 & = & \varphi(n) \\ & & 19 & = & d \\ 120 \bmod 19 & = & 6 & = & \varphi(n) - 6 \cdot d \\ 19 \bmod 6 & = & 1 & = & d - 3 \cdot (\varphi(n) - 6d) = 19d - 3 \cdot \varphi(n) \\ & & & \Rightarrow & e = 19 \end{array}$$

c) **Konstruktion einer großen Primzahl**

Wähle 500 Bit lange ungerade Zahl x.

Teste, ob $x,\ x+2,\ x+4,\ x+6, \ldots$ Primzahl ist.

Sei $\Pi(x)$ die Anzahl der Primzahlen unterhalb von x. Es gilt:
$\Pi(x) \approx \frac{x}{\ln x} \Rightarrow$ Dichte $\approx \frac{1}{\ln x}\ \Rightarrow$ mittlerer Abstand $\approx \ln x$

Also zeigt sich Erfolg beim Testen ungerader Zahlen der Größe $n = 2^{500}$ nach etwa $\frac{\ln 2^{500}}{4} = 86$ Versuchen.

Komplexitätsklassen für die Erkennung von Primzahlen:

$$\begin{array}{l} \text{Prim} \overset{?}{\in} \mathbb{P} \\ \text{Prim} \in \mathbb{NP} \\ \overline{\text{Prim}} \in \mathbb{NP} \\ \overline{\text{Prim}} \in \mathbb{RP} \end{array}$$

$L \in \mathbb{RP} : \Longleftrightarrow$ es gibt Algorithmus A, der angesetzt auf die Frage, ob $x \in L$, nach polynomialer Zeit mit ja oder nein anhält und folgende Gewähr für die Antwort gilt:

$$x \notin L \Rightarrow \text{ Antwort: nein}$$
$$x \in L \Rightarrow \text{ Antwort: } \underbrace{\text{ja}}_{>1-\varepsilon} \text{ oder } \underbrace{\text{nein}}_{<=\varepsilon}$$

Antwort: ja $\Rightarrow x$ ist zusammengesetzt.

Antwort: nein $\Rightarrow x$ ist höchstwahrscheinlich prim.

Bei 50 Versuchen $\Rightarrow$ Fehler $\leq \varepsilon^{50}$.

Satz von Rabin:

Sei $n = 2^k \cdot q + 1$ eine Primzahl, $x < n$

1) $x^q \equiv 1 \bmod n$ oder
2) $x^{q \cdot 2^i} \equiv -1 \bmod n$ für ein $i \in \{0, \ \ldots, \ k-1\}$

Beispiel:

Sei $n = 97 = 2^5 \cdot 3 + 1$, sei x = 2.

Folge der Potenzen	x	x^3	x^6	x^{12}	x^{24}	x^{48}	x^{96}
Folge der Reste	2	8	64	22	-1	1	1

Definition eines Zeugen:

Sei $n = 2^k \cdot q + 1$.

Eine Zahl $x < n$ heißt Zeuge für die Zusammengesetztheit von n

1) $ggt(x, n) \ \neq 1$ oder
2) $x^q \not\equiv 1 \bmod n$ und $x^{q \cdot 2^i} \not\equiv -1$ für alle $i \in \{0, \ldots, k-1\}$

Satz von Rabin:

Ist n zusammengesetzt, so gibt es mindestens $\frac{3}{4}n$ Zeugen.

```
function prob-prim (n: integer): boolean
z:=0;
repeat
    z=z+1;
    wuerfel x;
until (x ist Zeuge fuer n) OR (z=50);
return (z=50)
```

Fehler: $(\frac{1}{4})^{50} \sim 10^{-30}$

15.6.5 Sicherheit des RSA-Verfahrens

Der Code kann nur durch das Faktorisieren von n geknackt werden.
Schnellstes Verfahren zum Faktorisieren von n benötigt

$$n^{\sqrt{\frac{\ln\ln(n)}{\ln(n)}}} \text{ Schritte.}$$

Für $n = 2^{1000} \Rightarrow \ln(n) = 690, \ln\ln(n) = 6.5$

Es ergeben sich $\approx \sqrt[10]{n}$ Schritte $\approx 10^{30}$ Schritte $\approx 10^{21}$ sec (bei 10^9 Schritte pro sec) $\approx 10^{13}$ Jahre.

15.6.6 Implementation des RSA-Verfahrens

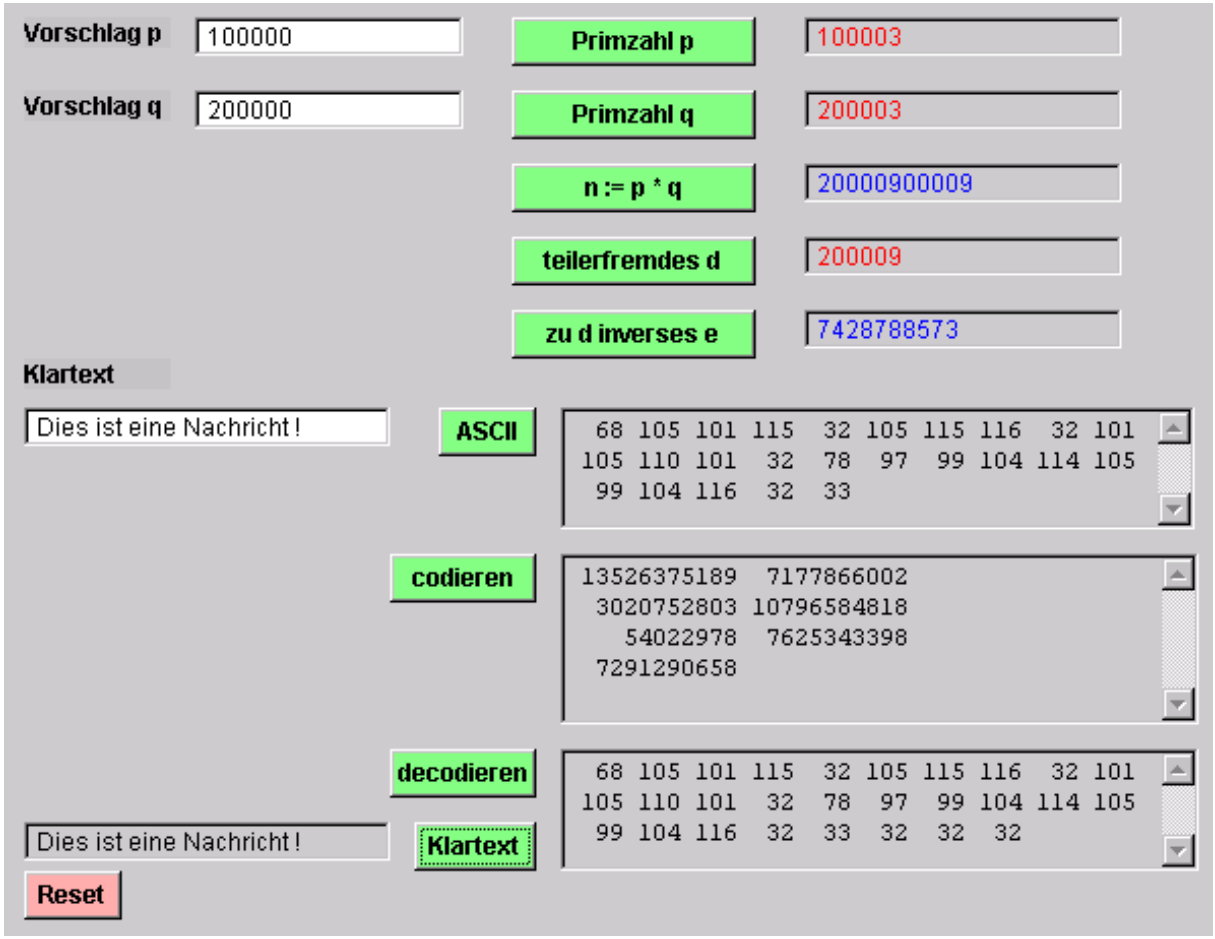

Abbildung 15.3: Java-Applet mit RSA-Algorithmus