UNIVERSITÄT DES SAARLANDES

Saarbrücken, 16.07.2015
Information Systems Group

# Vorlesung „Informationssysteme“

## Vertiefung Kapitel 11: Verschlüsseltes Datenmanagement

Erik Buchmann (buchmann@cs.uni-saarland.de)

Foto: M. Strauch

# Bevor es losgeht: Die Hülle

- Gegeben: ein Schema
  [R] = {[A, B, C, D, E, F, G]}
  und eine Menge von Funktionalen Abhängigkeiten FD auf [R]
  $A \rightarrow B, B \rightarrow C, A \rightarrow D, BD \rightarrow E, EC \rightarrow A, F \rightarrow G$
- Die „Hülle“ von Attributen
  - Intuitiv: welche Attribute sind funktional von einem anderen abhängig?
  - Achtung, auch offensichtliche, transitive und abgeleitete FDs berücksichtigen, die nicht explizit genannt wurden!
    - Attribut bestimmt sich immer selbst $A \rightarrow A$
    - Wenn $A \rightarrow B$ und $B \rightarrow C$, dann $A \rightarrow C$
    - Wenn $A \rightarrow B$, dann auch $AF \rightarrow BF$
- Hülle von A: $A \rightarrow B, B \rightarrow C, A \rightarrow D, BD \rightarrow E$, also F(A) = ABCDE
- Hülle von B: $B \rightarrow C$, also F(B) = BC
- Hülle von C: F(C) = C
- Hülle von AF: alle FDs, also F(AF) = ABCDEFG

# Aus den Videos wissen Sie...

- …dass DBMS Mechanismen zum Rechtemanagement besitzen
  - Schutz vor Einsicht und Änderungsoperationen
- ...dass der DMBS-Administrator aber alles darf
  - CREATE ROLE name SUPERUSER
- ***Wie Datenbanken sicher an externe Firmen auslagern?***

- Vertiefung heute:
  - (Sehr) kurze Einführung kryptographischer Ansätze
  - Verschlüsseltes Datenmanagement

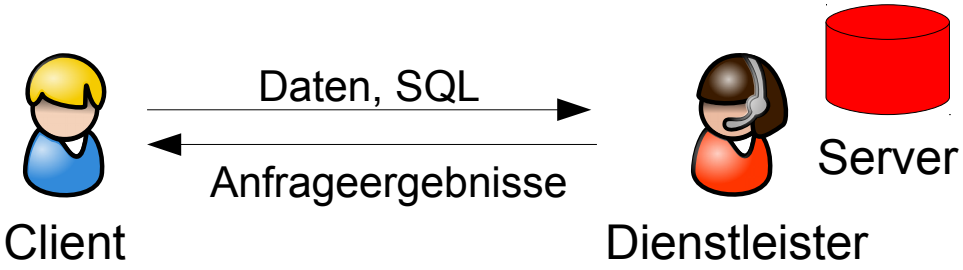
# Der Betrieb großer Datenbanken...

- ...ist nicht trivial
  - Regelmäßige Sicherheitsupdates und Backups
  - Diagnose, Wartung, Pflege, Tuning der Datenbank
  - Integration von neuen Anwendungen im laufenden Betrieb
  - Migration auf neue Hardware, Virtualisierungslösungen etc.
- ...ist nicht billig
  - Datenbank-Spezialisten zur Administration
  - Hardware mit genügend Reserveleistung für Lastspitzen
  - Personelle und technische Redundanzen gegen Systemausfall

*Rechenzentrum von Facebook in Pineville*

# Outsourcing an Datenbank-Dienstleister

- Dienstleister hat viele Kunden
  - Kann sich Spezialisten leisten
  - Reserve an Rechenleistung, Speicherkapazität, USV, Hardware etc. für mehrere Kunden ökonomischer
- Dienstleister kann in Haftung genommen werden
  - Verträge für garantierte Leistungen



# Outsourcing in die Cloud

- Anwendungen, Daten, Infrastruktur in verteiltes System ausgelagert
- Endgeräte nur noch Anzeigefunktion
- Ziele
  - Verfügbarkeit
  - zentrales Management
  - dynamische Skalierbarkeit

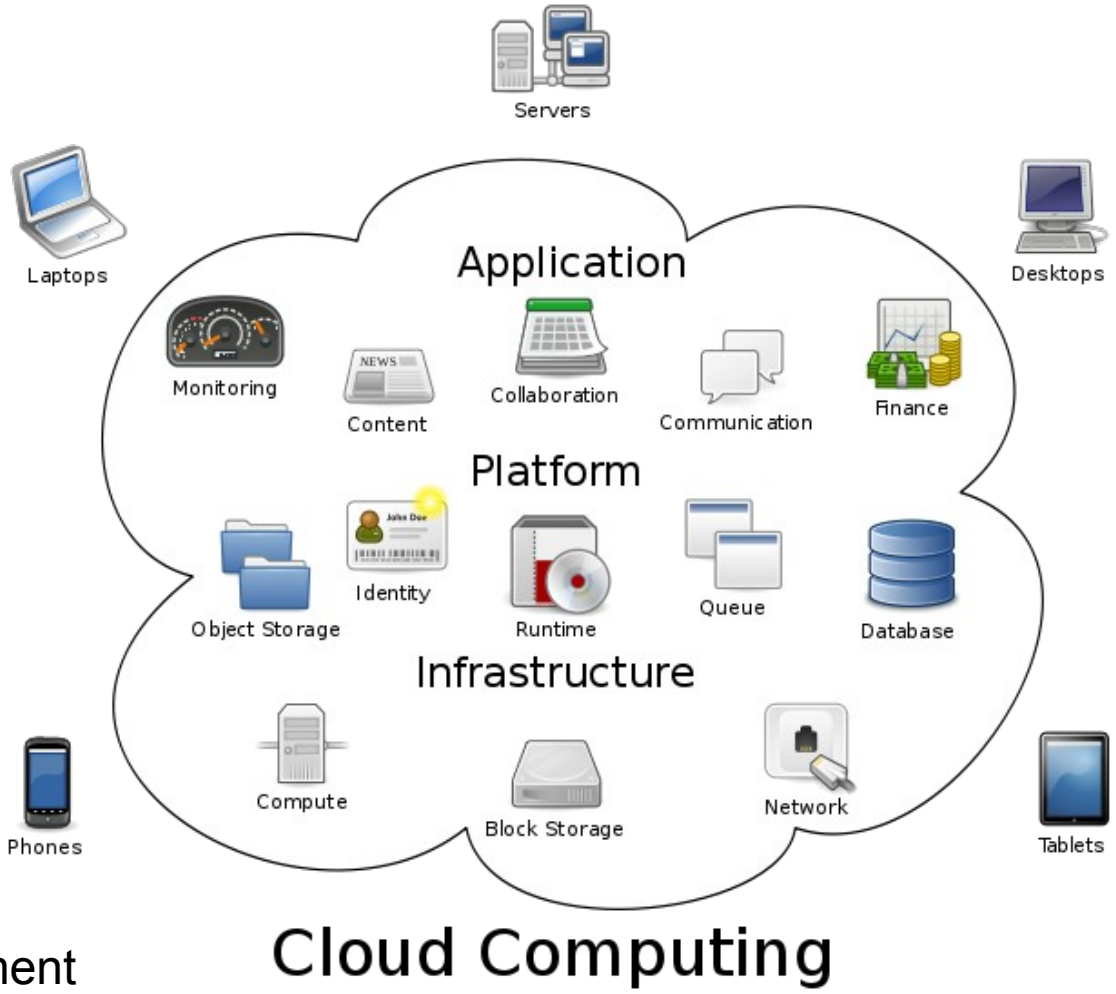
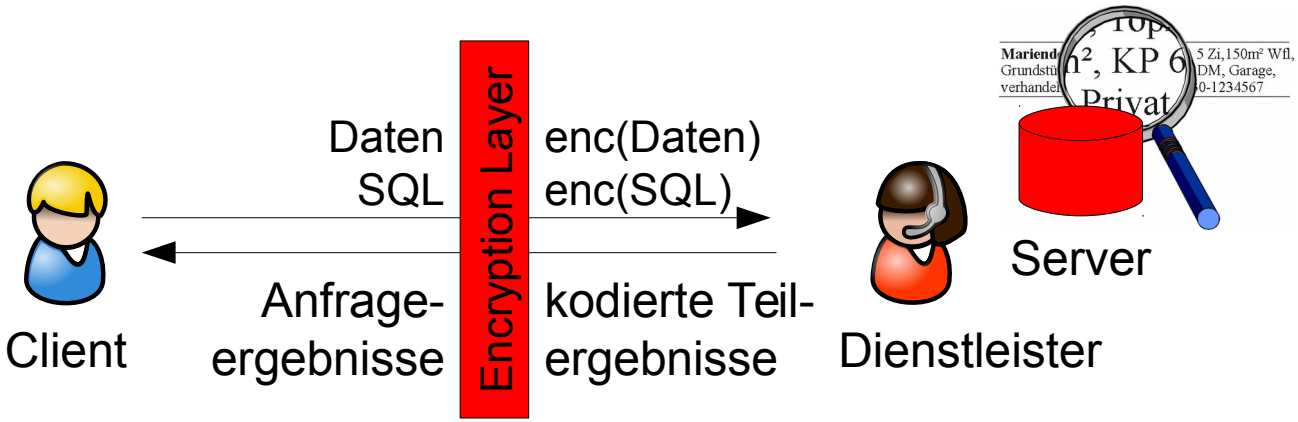



*Bild: Wikimedia Commons*

# Datenschutz, Vertraulichkeit?

- Daten auf fremden Servern, ggf. im Ausland
  - Datenschutzpflichten
  - Schutz von Geschäftsgeheimnissen
- **Zentrales Ziel: Dienstleister soll mit Daten arbeiten, sie aber nicht einsehen können**
  - Zugriffsschutz des DBMS nicht ausreichend

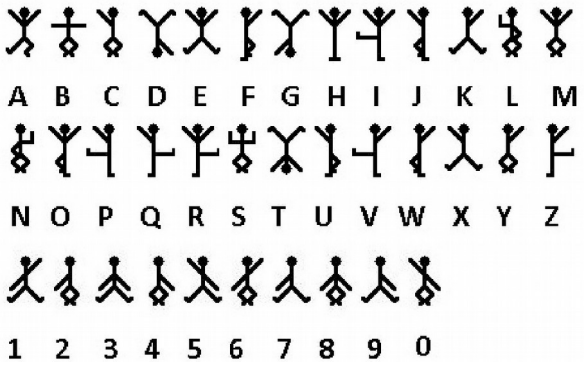


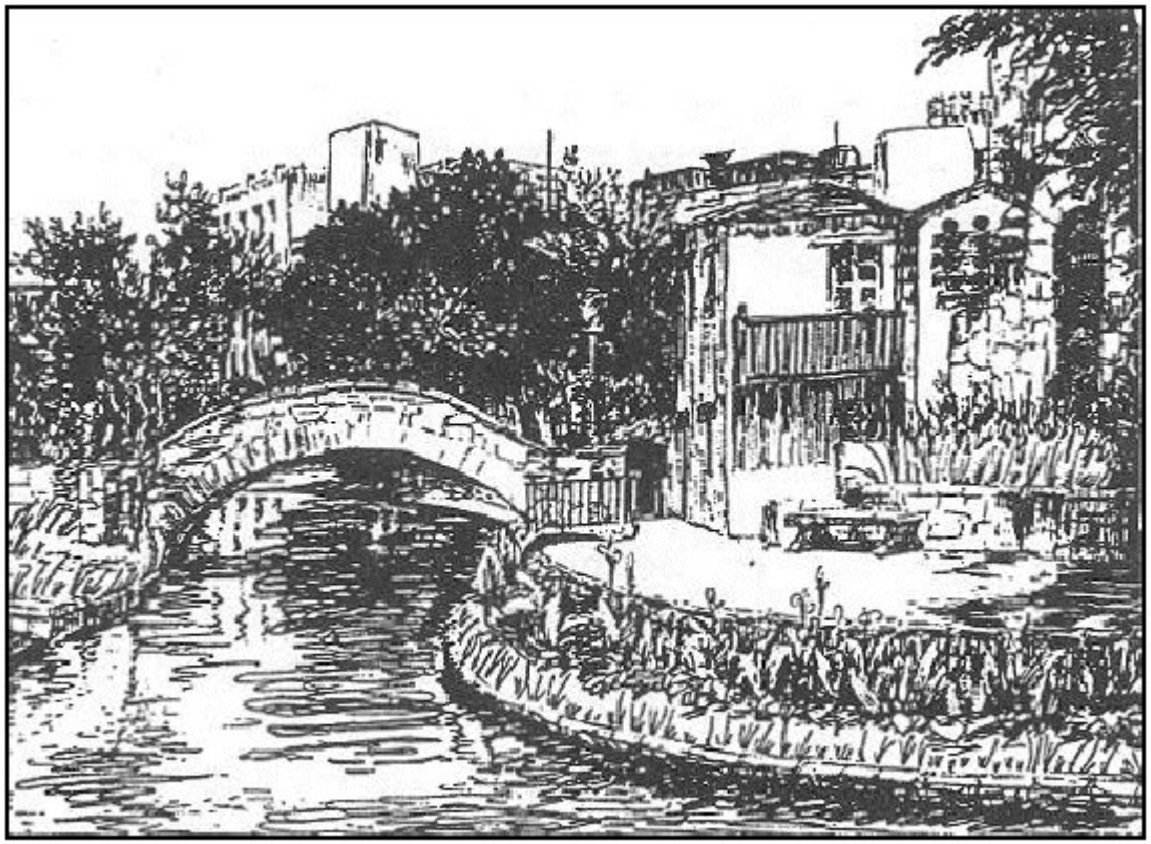
# Kryptographische Ansätze

Foto: M. Strauch

# Wer etwas zu verbergen hat...

- ...versucht es zu verstecken.
  - Unsichtbare Tinte (Zitronensaft trocknet unsichtbar, bei erhitzen braun)
  - Codebücher
  - In Bildern (rechts: Morsecode in Grashalm-länge codiert)
  - Geheinschrift (unten: nach Arthur C. Doyle)



# Zielsetzungen

- Klassische Ziele der Kryptographie
  - **Vertraulichkeit**
    - Nur Berechtigte sollen Nachricht lesen, Absender/Empfänger in Erfahrung bringen, von Existenz einer Nachricht erfahren
  - **Integrität**
    - Daten nachweislich unverfälscht vom Sender zum Empfänger
  - **Authentizität**
    - Urheber einer Nachricht eindeutig nachprüfbar
  - **Verbindlichkeit**
    - Absender einer Nachricht soll Urheberschaft nicht abstreiten können
- Neue Ziele in unserem Kontext
  - **Plausible Deniability** (Glaubhafte Abstreitbarkeit)
    - Gegenteil von Verbindlichkeit, Urheberschaft überzeugend leugnen können
  - **Separation of Duties**
    - Funktionstrennung unter Nichtverknüpfbarkeit

# Klassische Vorgehensweise

- Security through obscurity
  - Mache das Verfahren so kompliziert wie möglich und halte es geheim
- break-it fix-it break-it fix-it...
  - Alice entwickelt Verschlüsselung
  - Carol versucht Verfahren zu knacken
  - Nach einiger Zeit kein Angriff: → Verfahren gilt als „sicher“
    Bei erfolgreichem Angriff: → Alice entwickelt nächstes Verfahren
- Beispiel von so entstandenen Verfahren
  - Enigma, 2. WK (drehbare Rotoren mit unregelmäßiger Verzahnung)

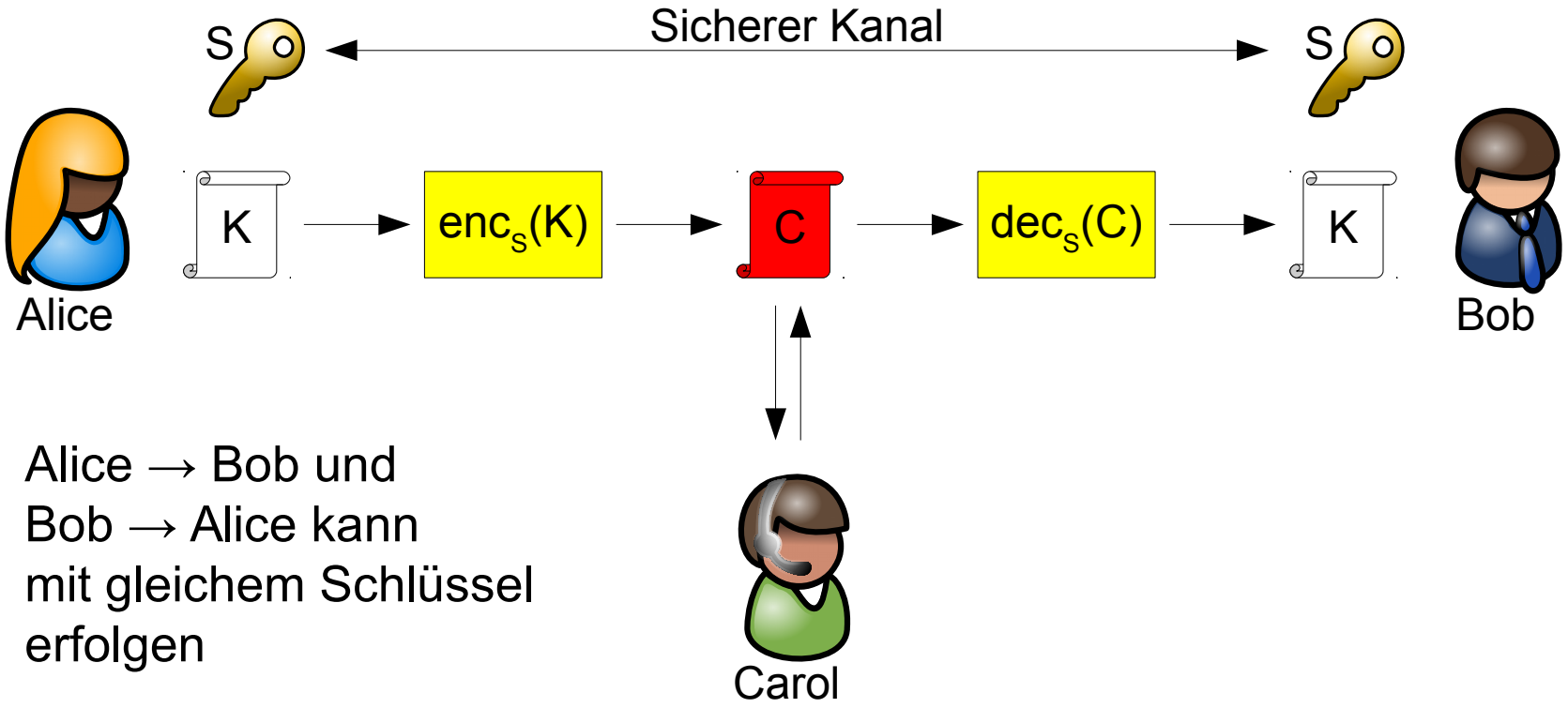
# Aktuelle Verschlüsselungsverfahren

- Verfahren werden veröffentlicht
  - Claude Shannon: „The enemy knows the system“
  - Sicherheit der Verschlüsselung nur durch Geheimhaltung des Schlüssels nicht Geheimhaltung des Verfahrens!

- Sicherheit wird unterschieden nach
  - Ressourcen des Angreifers
    - **polynomial beschränkter Angreifer**
  - Ziele des Angreifers, z.B.
    - Berechnung des Schlüssels S
    - Mindestens ein Bit Klartext $k \in K$ aus Ciphertext $C \rightarrow enc_S(K)$ extrahieren
    - Bestimmung von Meta-Informationen aus gegebenem Ciphertext C (Länge des Schlüssels, Länge des Textes, Sprache, Urheber etc.)
  - Angreifermodell
    - Ciphertext-Only, Known-Plaintext, Chosen-Plaintext, Chosen-Ciphertext

# Symmetrische Verschlüsselung

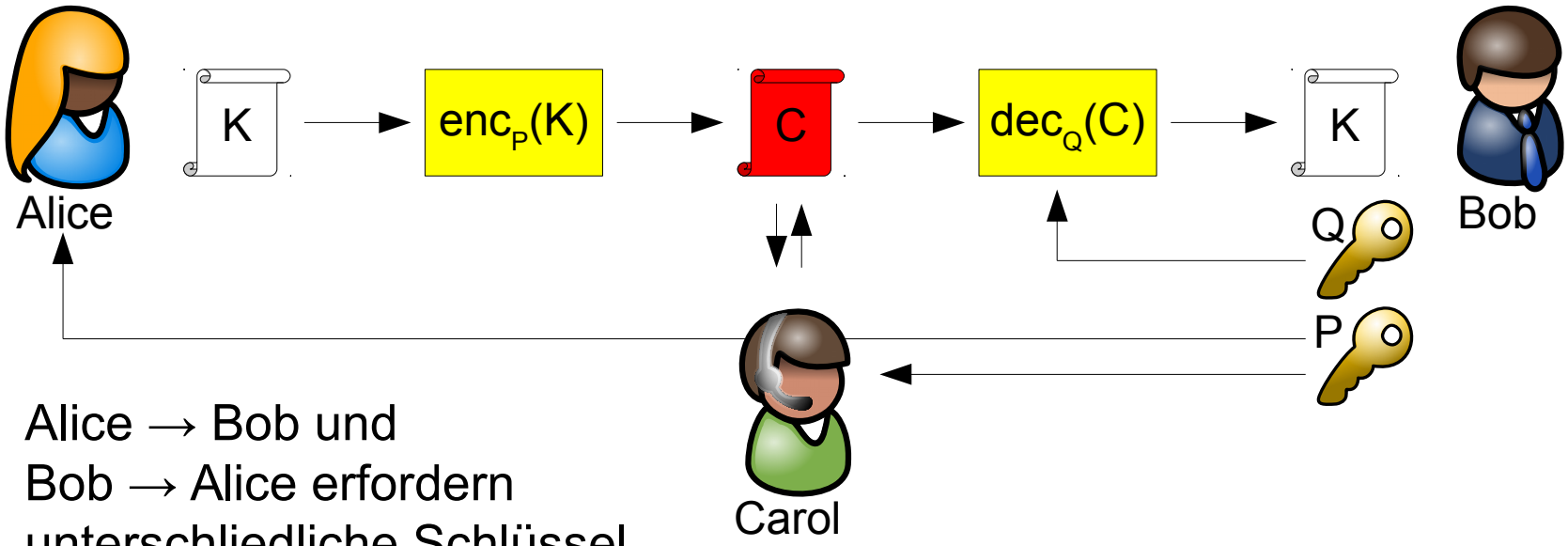
- Alice und Bob haben gleichen Schlüssel
- Ein Schlüssel zur Ver- und Entschlüsselung
- Schlüssel ist geheim, muss über gesicherten Kanal übertragen werden




- Alice → Bob und Bob → Alice kann mit gleichem Schlüssel erfolgen

# Asymmetrische Verschlüsselung

- auch: Public-Key-Verschlüsselung
- Öffentlicher Schlüssel P zur Verschlüsselung, Übertragung ungesichert
- Privater Schlüssel Q zur Entschlüsselung,
  wird vom Empfänger geheimgehalten




- Alice → Bob und
  Bob → Alice erfordern
  unterschliedliche Schlüssel

# Beispiel für asymm. Verfahren: RSA

- 1977 von Ron Rivest, Adi Shamir, Leonard Adleman am MIT entwickelt
- Erstes veröffentlichtes Public-Key-Verschlüsselungsverfahren

- Idee: Multiplikation von zwei Primzahlen ist „billiger“ als Primfaktorzerlegung

- Public Key
  - Für die Verschlüsselung, kann veröffentlicht werden
  - Schlüssel: {RSA-Modul, Verschlüsselungskomponente}
    (Produkt zweier Primzahlen, Hilfswert)
- Private Key
  - Für die Entschlüsselung, wird geheim gehalten
  - Schlüssel: {RSA-Modul, Entschlüsselungsexponent}
    (Produkt zweier Primzahlen, anderer Hilfswert)

# Schlüsselerzeugung für RSA

- Wähle zwei große Primzahlen p,q
- Berechne den RSA-Modul
  $N = p*q$
- Berechne die Eulersche φ-Funktion des RSA-Moduls
  $\varphi(p,q) = ( p-1 ) * ( q-1 )$
- Wähle zufällig eine zu φ(p,q) teilerfremde Zahl e mit
  $1 < e < \varphi(p,q)$
- Berechne den Entschlüsselungsexponenten d, so dass gilt
  $d * e \equiv 1 \text{ MOD } \varphi(p,q)$
- Die Parameter p,q und φ(p,q) können nun gelöscht werden

- Public Key: {N,e}
- Private Key: {N,d}

# Ver- und Entschlüsselung mit RSA

- Public Key: {N,e}
- Private Key: {N,d}

- Verschlüsseln: $C = K^e \text{ MOD } N$
- Entschlüsseln: $K = C^d \text{ MOD } N$

- Rechenbeispiel
  - Verschlüssele 7
    gewählt: p = 11, q = 13
    berechnet: e = 23, d = 47
    enc(7) = (7^23) MOD (11*13) = 2
  - Entschlüssele 2
    dec(2) = (2^47) MOD (11*13) = 7

*Hilfsvariablen:*

$$N = p*q$$
$$\varphi(p,q) = (p-1)*(q-1)$$
$$e = \text{zu } \varphi(p,q) \text{ teilerfremde Zahl}$$
$$d * e \equiv 1 \text{ MOD } \varphi(p,q)$$


Foto: M. Strauch

# Architekturmodell im Folgenden

- Dienstleister/Cloud verwaltet verschlüsselte Datenbank
  - Client sendet umkodierte SQL-Anfragen
  - Povider führt Anfragen aus, liefert kodierte Zwischenergebnisse
  - Client ermittelt endgültiges Anfrageergebnis aus Zwischenergebnis

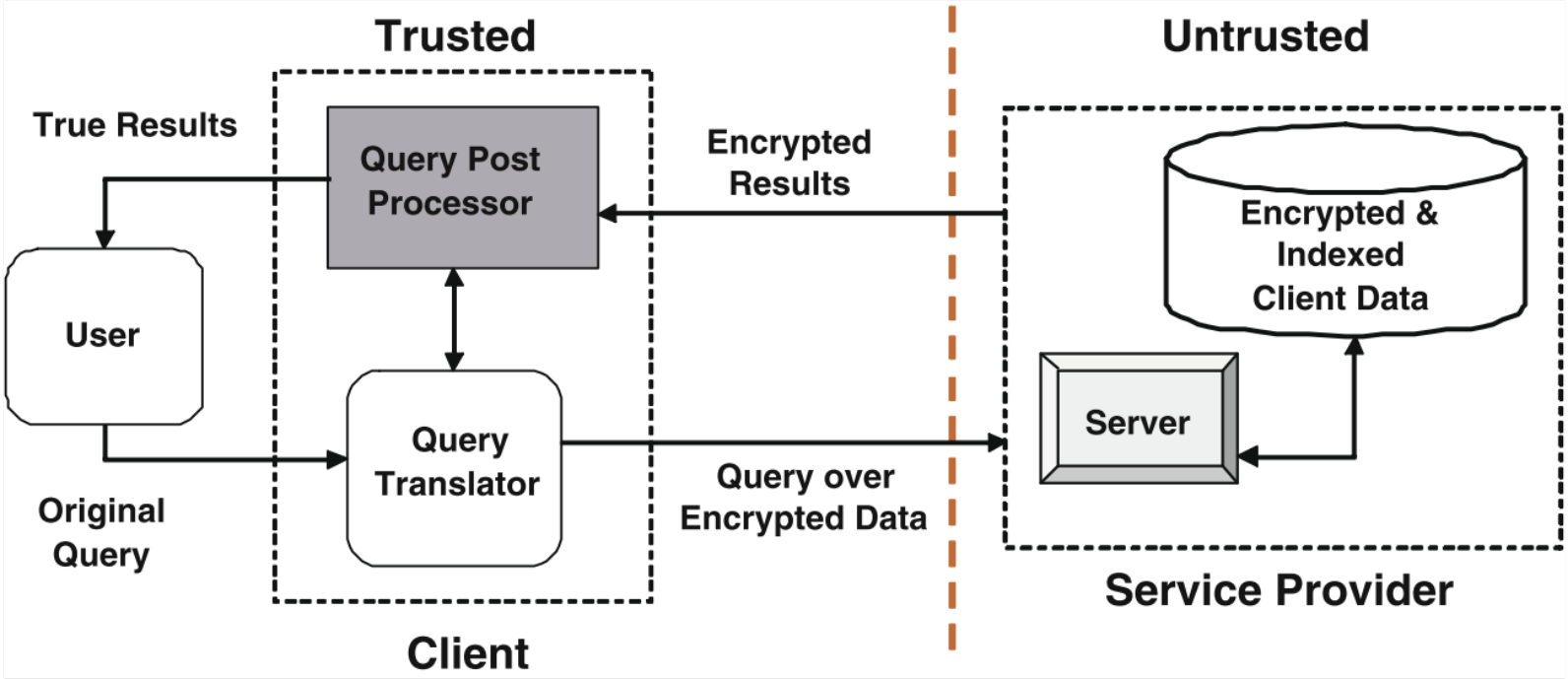



*Bild:* [3]

# Anfragetypen

- Exact-Match-Anfragen

```
SELECT * FROM db WHERE Diagnose='Schnupfen';
```

- (Mehrdimensionale) Bereichsanfragen

```
SELECT * FROM db WHERE PLZ>76221 AND PLZ<76225;
```

- Verbundanfragen

```
SELECT * FROM db AS a, db AS b WHERE a.Name=b.Name AND
a.Diagnose='Schnupfen' and b.Diagnose='Heiser';
```

- Mengenoperationen

```
SELECT * FROM db WHERE Diagnose='Schnupfen'
MINUS SELECT * FROM db WHERE Diagnose='Heiser';
```

- Aggregate (Min, Max, Avg, Count)

```
SELECT AVG(Gbt) FROM db
WHERE Diagnose='Schnupfen';
```

- Gruppierung

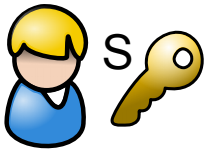
```
SELECT AVG(Gbt) FROM db
GROUP BY Diagnose
HAVING AVG(Gbt) > 1985;
```

**db**

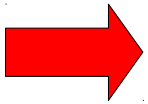
| Name | Gbt | PLZ | Diagnose |
|---|---|---|---|
| Alice | 1981 | 76227 | Schnupfen |
| Bob | 1974 | 76221 | Husten |
| Carol | 1995 | 76221 | Heiser |
| Dave | 2001 | 76229 | Schnupfen |
| Alice | 1981 | 76227 | Husten |

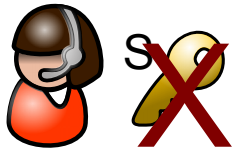
# Daten „einfach so“ verschlüsseln?

- Client verschlüsselt die Daten, die der Dienstleister auf Server speichert




| Name | PLZ | Diagnose |
|---|---|---|
| Alice | 76227 | Schnupfen |
| Bob | 76221 | Husten |
| Carol | 76221 | Heiser |
| Dave | 76229 | Schnupfen |
| Alice | 76227 | Husten |



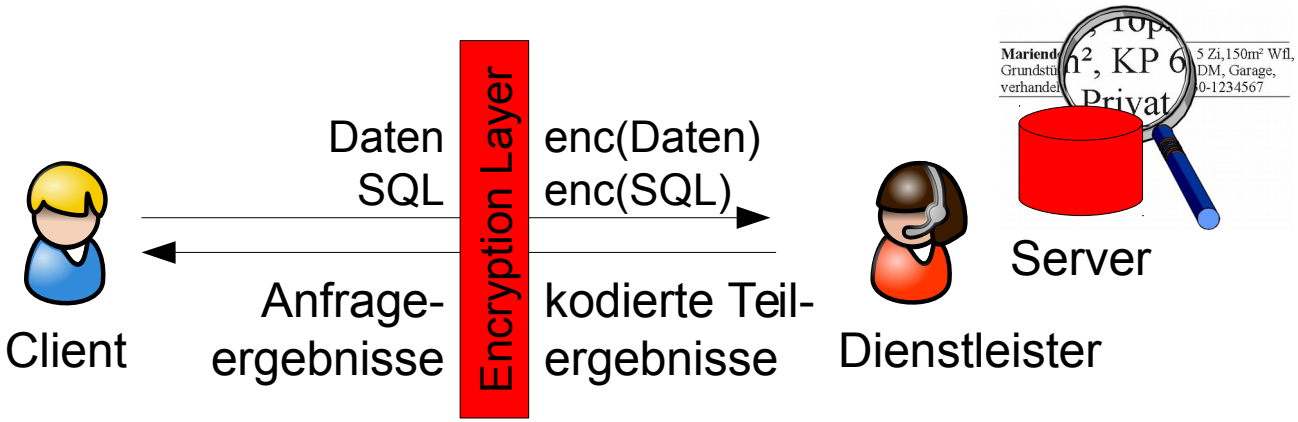



| Anzr | CYM | Qvntabfr |
|---|---|---|
| Nyvpr | 21772 | Fpuahcsra |
| Obo | 21776 | Uhfgra |
| Pneby | 21776 | Urvfre |
| Qnir | 21774 | Fpuahcsra |
| Nyvpr | 21772 | Uhfgra |

- Welche Arten von Anfragen effizient auf Server ausführbar?
  - select Diagnose from db where PLZ = 76229
  - select count(*) from db where PLZ >= 76225 and Krankheit = 'Husten'
  - select A.Diagnose, B.Diagnose from db as A, db as B where A.Name = B.Name

# Angriffe auf verschlüsseltes Datenmanagement

- Dienstleister ist **Honest-but-Curious**
  - arbeitet **korrekt**
    - kein Löschen, Manipulieren von Daten
  - Ausspähen von **Daten und Anfragen** falls möglich
    - Zugriff auf DB
    - Zugriff auf Logs
    - **Frequenz- und Zuordnungsangriffe** durchführen



# Zuordnungsangriffe

- Bisher vorgestellte Verschlüsselungsverfahren sind deterministisch
  - Bei jeder Verschlüsselung:
    Identischer Klartext → identischer Ciphertext
  - Public-Key-Verfahren erlauben Chosen-Plaintext-Angriffe,
    d.h., man kann ausprobieren, ob man den Klartext erraten kann

- Beispiel: Befindet sich „Alice“ in einer verschlüsselten DB?
  - Angreifer kennt Public Key:
    „Alice“ → „H71A00EF91“

| Person | Krankheit | Alter |
|---|---|---|
| AA73F9E001 | Magersucht | 24 |
| **H71A00EF91** | Diabetes | 31 |
| 3429A9F64E | Schnupfen | 26 |

# Frequenzangriffe

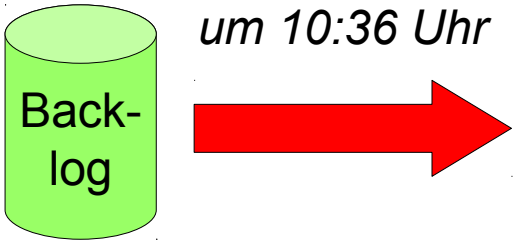
- Bisher vorgestellte Verschlüsselungsverfahren sind deterministisch
  - Bei jeder Verschlüsselung:
    Identischer Klartext → identischer Ciphertext
  - Aus der Verteilung von identischen Ciphertexten kann auf Inhalte geschlossen werden

- Beispiel: Hat Alice Schnupfen?
  - Angreifer weiß, dass Schnupfen die häufigste Diagnose ist

| Person | ~ | Qvntabfr |
|---|---|---|
| Alice | | **Fpuahcsra** |
| Bob | | Uhfgra |
| Carol | | Urvfre |
| Dave | | **Fpuahcsra** |
| Eddie | | **Fpuahcsra** |

# Seitenkanalangriffe

- Angriffe gegen die physische Implementierung des Datenbanksystems
  - Zugriff auf Query-Logs, Backlog
  - Beobachtung von Input und Output des Systems
  - Beobachgung, welche Teilnehmer (IP-Adressen) wann zugreifen
- Beispiel: Dienstleister, der die DB verwaltet, verursacht Änderungen
  - Angreifer wird im Krankenhaus mit Schnupfen diagnostiziert, schaut danach ins Backlog




*Neuer Eintrag um 10:36 Uhr*

| Anzr | CYM | Qvntabfr |
|---|---|---|
| Nyvpr | 21772 | Fpuahcsra |
| Obo | 21776 | Uhfgra |
| Pneby | 21776 | Urvfre |
| **Qnir** | **21774** | **Fpuahcsra** |
| Nyvpr | 21772 | Uhfgra |

# Sicherheitsbeweis: IND-ICP

- Indistinguishability under Independent Column Permutation (IND-ICP)
  - Nachweis, dass Frequenz- und Zuordnungsangriffe unmöglich sind
- Input
  - Datenbank DB
  - Datenbankfunktion f: DB → DB verschlüsselt Datenbank
  - Datenbankfunktion p: DB → DB permutiert alle Spalten in der Datenbank unabhängig voneinander
- Angriff
  - Für eine große Zahl von Runden
  - Zwei Datenbanken $DB_1$ = f(DB) und $DB_2$ = f(p(DB))
  - Angreifer erhält zufällig mit Gleichverteilung ($DB_1$,$DB_2$) oder ($DB_2$,$DB_1$)
  - Angriff ist erfolgreich, wenn Angreifer mit Wahrscheinlichkeit > 50% + ε sagen kann, ob die erste DB die originale oder permutierte DB ist
    *(ε ist „Sicherheitsmarge“ für polynomiale Beschränktheit des Angreifers)*

# Bucketization

Foto: M. Strauch

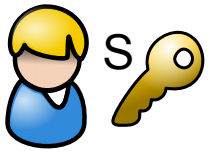
# Schutzziel

- Angreifermodell
  - Angreifer ist **Honest-but-Curious**
  - Angreifer hat Zugang zur Datenbank (Outsourcing-Szenario)
- Angreifer will
  - Vollständigen Inhalt der Datenbank offenlegen
  - Zuordnung von Individuen zu sensitiven Attributen erfahren (**Frequenz- und Zuordnungsangriffe**)

| VersNr | Name | Gbt | PLZ | Arzt | Diagnose |
|---|---|---|---|---|---|
| 123 | Alice | 1981 | 76227 | Dr. Brown | Schnupfen |
| 456 | Bob | 1974 | 76221 | Dr. Red | Husten |
| 789 | Carol | 1995 | 76221 | Dr. Olive | Heiser |
| 101 | Dave | 2001 | 76229 | Dr. Blue | Schnupfen |
| 112 | Alice | 1981 | 76227 | Dr. Red | Husten |

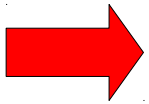
# Systemmodell

- Server speichert
  - Datenbank db mit Relationenschema R,
    **tupelweise verschlüsselt:** enc = encode({$A_0$, …, $A_n$}), $A_0$, …, $A_n \in R$
  - kodierten Index




db

| Name | PLZ | Diagnose |
|---|---|---|
| Alice | 76227 | Schnupfen |
| Bob | 76221 | Husten |
| Carol | 76221 | Heiser |
| Dave | 76229 | Schnupfen |
| Alice | 76227 | Husten |



encdb

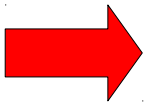
| enc | $I_N$ | $I_P$ | $I_D$ |
|---|---|---|---|
| 770E5B0F8 | α | I | 1 |
| 86D3EFBC2 | β | II | 2 |
| EB51DA3C6 | γ | II | 3 |
| 15487E7B5 | δ | III | 1 |
| 8ABE19436 | α | I | 2 |

- Client
  - entschlüsselt Zwischenergebnisse vom Server
  - berechnet Endergebnisse
- **Wie den Index kodieren?** [1]

# Anfrageverarbeitung auf Buckets

db

| Name | PLZ | Diagnose |
|---|---|---|
| Alice | 76227 | Schnupfen |
| Bob | 76221 | Husten |
| Carol | 76221 | Heiser |
| Dave | 76229 | Schnupfen |

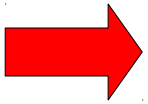


encdb

| enc | $I_N$ | $I_P$ | $I_D$ |
|---|---|---|---|
| 770E5B0F8 | α | I | 1 |
| 86D3EFBC2 | β | II | 2 |
| EB51DA3C6 | γ | II | 3 |
| 15487E7B5 | δ | III | 1 |

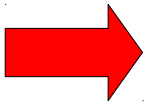
- Query Rewriting auf verschlüsselten Index

```
SELECT Diagnose
FROM db
WHERE PLZ = 76221
```



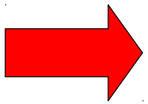
```
SELECT enc
FROM encdb
WHERE I_P = 'II'
```

- Indexierte Tupel vom Server holen, auf Client entschlüsseln
  - 86D3EFBC2, EB51DA3C6



{Bob, 76221, Husten}
{Carol, 76221, Heiser}

- Client berechnet Ergebnis
  - SELECT Diagnose FROM db
    WHERE PLZ = 76221



| Husten |
|---|
| Heiser |

# Indexerstellung

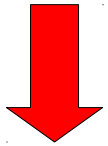
- Kodierte Attribute als Indexeinträge
- Kodierung bzw. Verschlüsselung hat Einfluß auf Query-Performanz Sicherheit

- <u>Kardinalität des Mappings</u>
  - 1:1-Mapping von Attributwerten auf Index
  - N:M-Mapping von Attributwerten auf Index (mit N ≥ M, „Bucketization“)

- <u>Art der Kodierung</u>
  - Zufällige Kodierung
  - Ordnungserhaltende Kodierung

db

| Name | PLZ | Diagnose |
|---|---|---|
| Alice | 76227 | Schnupfen |
| Bob | 76221 | Husten |
| Carol | 76221 | Heiser |
| Dave | 76229 | Schnupfen |
| Alice | 76227 | Husten |

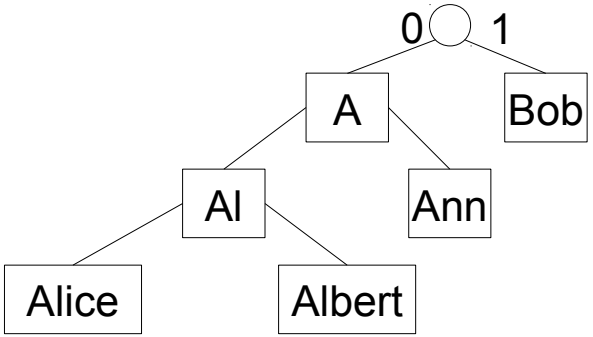


encdb

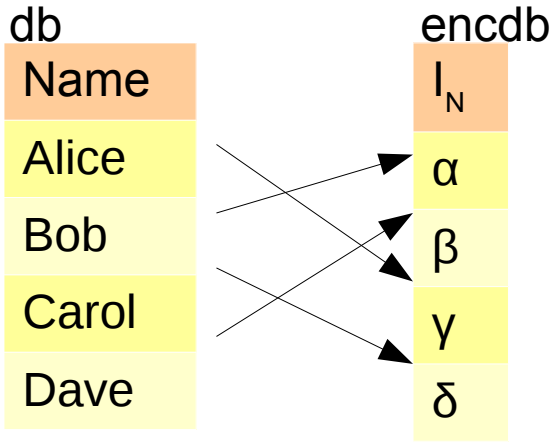
| enc | $I_N$ | $I_P$ | $I_D$ |
|---|---|---|---|
| 770E5B0F8 | α | I | 1 |
| 86D3EFBC2 | β | II | 2 |
| EB51DA3C6 | γ | II | 3 |
| 15487E7B5 | δ | III | 1 |
| 8ABE19436 | α | I | 2 |

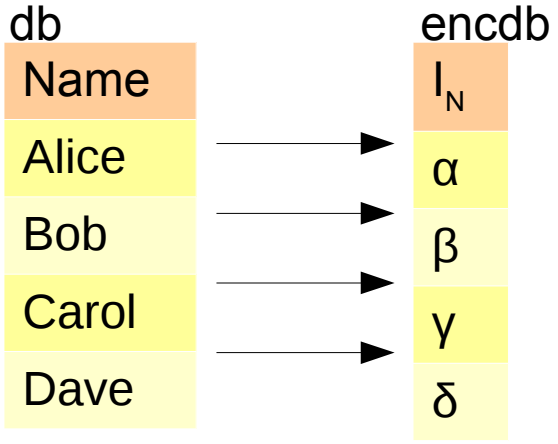
# Art der Kodierung

- Zufällig
  - Keine Ordnungsrelation zwischen Attributwerten und Indexwerten
  - z.B. kryptographische Hashfunktion oder symmetrische Verschlüsselung
- Ordnungserhaltend
  - Bestehende Ordnungsrelation
    - z.B. Prefix eines Tries bestimmt Hash-Bucket
  - Performante Bereichsanfragen
  - evtl. Zuordnungsangriffe möglich



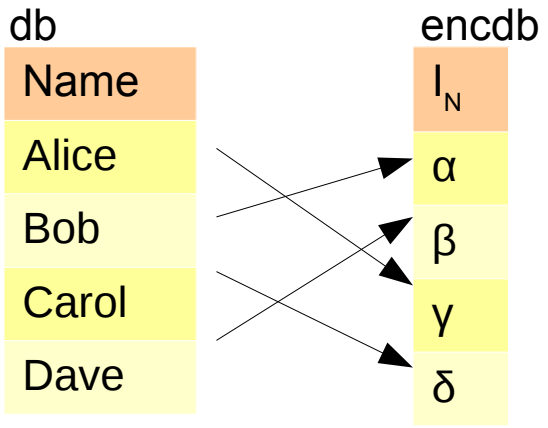

*Anm.: Kodierung ist unabhängig davon, ob 1:1 oder M:N-Mapping*



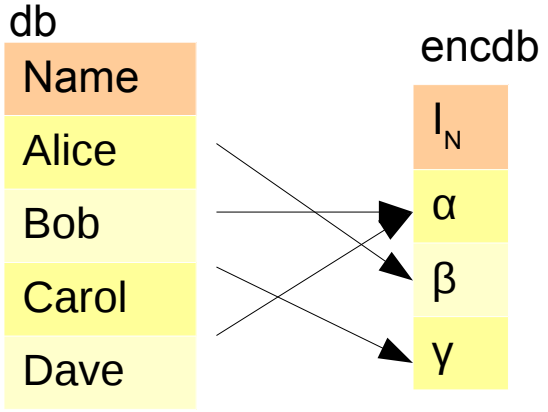



# Mapping der Attributwerte

- 1:1-Mapping von Attributwerten auf Index
  (ein Attributwert entspricht einem verschlüsselten Indexeintrag)
  - Symmetrische oder asymm. Verschlüsselung
  - Performante Exact-Match-Anfragen möglich
  - Frequenz- und Zuordnungsangriffe möglich

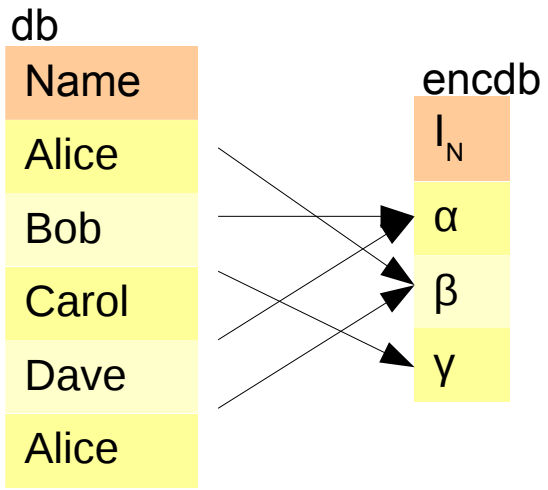



- Hash-Funktion über Attributwerte
  (mehrere Attributwerte entsprechen dem selben Indexeintrag)
  - Konzept äquivalent zu Hash-Index
  - N:M-Mapping von Attributwerten auf Index
    (N ≥ M, „**Bucketization**“)
  - Performanz, Frequenz- und Zuordnungsangriffe hängen stark von dem Mapping ab
    → **False Positives** bei der Anfrage



# Die Hashfunktion

- Eigenschaften einer „sicheren“ Hashfunktion
  - Umkehrung ist nicht zu berechnen
    - nicht von Index auf Attributwert schließen
  - Attribute mit Gleichverteilung abbilden
    - Verteilung der Attributwerte auf Server entspricht nicht mehr Verteilung der Werte in den Hash-Buckets
  - Benachbarte Attributwerte landen nicht in benachbarten Buckets
    - Keine Ordnungsrelation zwischen Attributwerten und Indexwerten (kann für ordnungserhaltende Kodierung anders gelöst sein)



# Angreifbarkeit

1.) Wissen: encdb + Erwartungswerte der Attributverteilung in db
z.B. Angreifer kennt Häufigkeit der Diagnosen
- Angriff auf Regel *Attributwert* → *Indexwert*
- Angriff zur Aufdeckung des Klartexts von db

2.) Wissen: encdb + db
z.B. Dienstleister wechselt von unverschlüsselter auf verschlüsselte DB
- Angriff auf Regel *Attributwert* → *Indexwert*

| $Freq_{Name}$ |
|---|
| 2 |
| 1 |
| 1 |
| 1 |

?

| $I_N$ |
|---|
| 2α |
| 2β |
| 1γ |

- **Messung der Exposure (Preisgabe):**
  - Zahl der möglichen Lösungen für die Abbildung von Attributwerten auf Indexwerte
    - Wenn nur eine Lösung, ist Verfahren untauglich
  - Lösung für 2.): Alice muss α oder β sein; die 3 anderen permutieren: 6 Lösungsvarianten

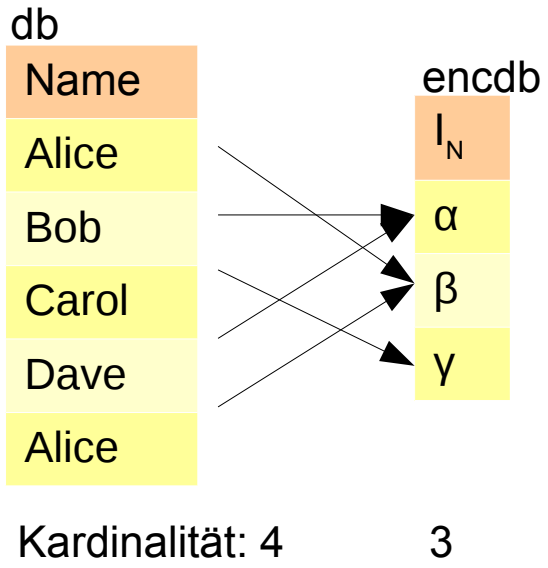
| Name |
|---|
| Alice |
| Bob |
| Carol |
| Dave |
| Alice |

?

| $I_N$ |
|---|
| 2α |
| 2β |
| 1γ |

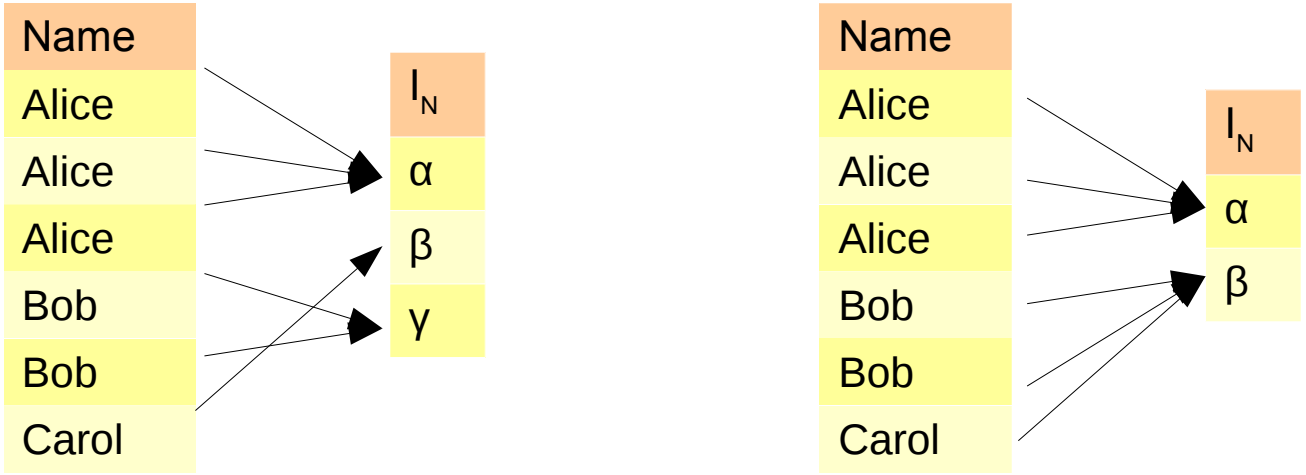
# Query Performance

- Kardinalität der Attributwerte zur Kardinalität der Co-Domain der Hash-Funktion bestimmt Performanz von Exact-Match-Anfragen
- Beispiel
  - SELECT * FROM db WHERE Name = 'Bob'
    Rewriting:
    SELECT * FROM encdb WHERE $I_N$ = 'α'
  - Ergebnismenge: {Bob, Dave}
    - Client muss false positives entschlüsseln und filtern
    - Große Buckets: hoher Aufwand
- Was ist mit
  - Bereichsanfragen?
  - Verbundanfragen wie Equi-Join, Theta-Join?
  - AVG, SUM, GROUP-BY, ORDER-BY?



# Diskussion Bucketization

- Vorteile
  - Kompromiss zwischen Performanz und Privatheit steuerbar
  - Einfach einzusetzen, keine großen Änderungen an existierenden DBMS
- Nachteile
  - Je nach Verteilung der Attributwerte (z.B. Power-Law)
    - Angreifbar durch einzigartige Verteilung der Indexwerte (1:1-Mapping) oder ordnungserhaltende Kodierung
    - Schlechte Performanz durch viele „false positives“ in den Buckets oder zufällige Kodierung



# Mehrdimensionale Bucketization

Foto: M. Strauch

# Multidimensionale Bereichsanfragen

- Bereichsanfragen über mehrere (viele) Attribute
  - SELECT Name FROM db
    WHERE PLZ > 76225 AND PLZ < 76228
    AND Gbt > 1982 AND Gbt < 1986

- Bucketization
  - viele false Positives
    → *schlaueres Verfahren?* [3]

**db**

| Name | Gbt | PLZ | Diagnose |
|---|---|---|---|
| Alice | 1981 | 76227 | Schnupfen |
| Bob | 1974 | 76221 | Husten |
| Carol | 1995 | 76224 | Heiser |
| Dave | 2001 | 76229 | Schnupfen |
| **Eve** | **1985** | **76227** | **Husten** |
| Frank | 1978 | 76229 | Heiser |
| **George** | **1983** | **76226** | **Heiser** |
| Harold | 1999 | 76223 | Husten |

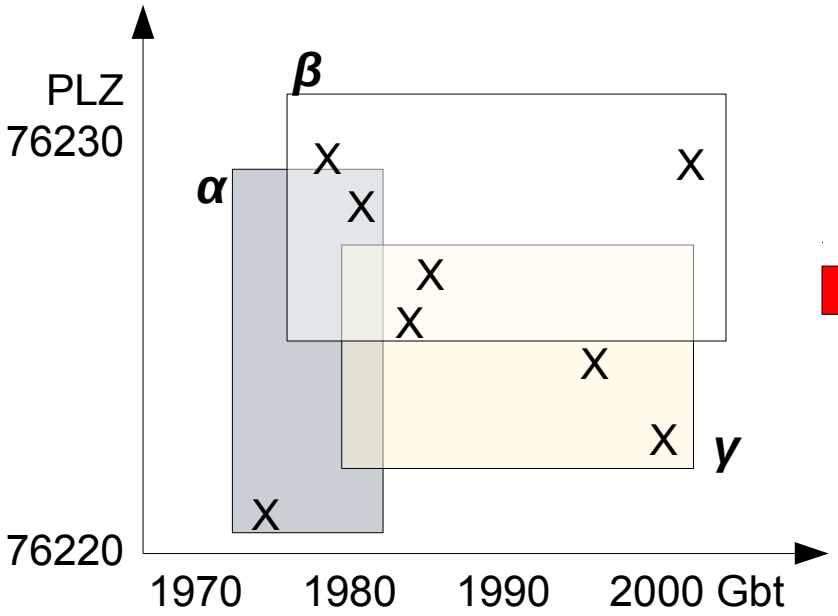
# Multidimensionale Bucketization

- Client speichert
  - Schlüssel
  - Zuordnungsregel: Bucket-Grenzen

| | |
|---|---|
| α | <(1970,76220),(1980,76229)> |
| β | <(1975,76226),(2010,76233)> |
| γ | <(1980,76223),(2000,76227)> |

- Server speichert
  - Datenbank, **tupelweise verschlüsselt** enc = encode({$A_0$, …, $A_n$})
  - Bucket-ID (B)

| Gbt | PLZ |
|---|---|
| 1981 | 76227 |
| 1974 | 76221 |
| 1995 | 76224 |
| 2001 | 76229 |
| 1985 | 76227 |
| 1978 | 76229 |
| 1983 | 76226 |
| 1999 | 76223 |




| enc | B |
|---|---|
| 770E5B0F8 | β |
| 86D3EFBC2 | α |
| EB51DA3C6 | γ |
| 15487E7B5 | β |
| 8ABE19436 | γ |
| C2AB7DA00 | α |
| 2814F82C4 | β |
| 9B7DFAFA0 | γ |

# Anfrageverarbeitung

- Query Rewriting
  - SELECT * FROM db WHERE
    PLZ > 76225 AND PLZ < 76228 AND
    Gbt > 1982 AND Gbt < 1986
  - SELECT enc FROM encdb WHERE B IN (β,γ)

| | |
|---|---|
| α | <(1970,76220), (1980,76229)> |
| β | <(1975,76226), (2010,76233)> |
| γ | <(1980,76223), (2000,76227)> |

- Zwischenergebnis vom Server entschlüsseln
  - {770E5B0F8, EB51DA3C6, 15487E7B5, 8ABE19436, 2814F82C4, 9B7DFAFA0}
  - {Alice, …}, {Carol, …}, {Dave, …}
    {Eve, …}, {George, …}, {Harold, ...}

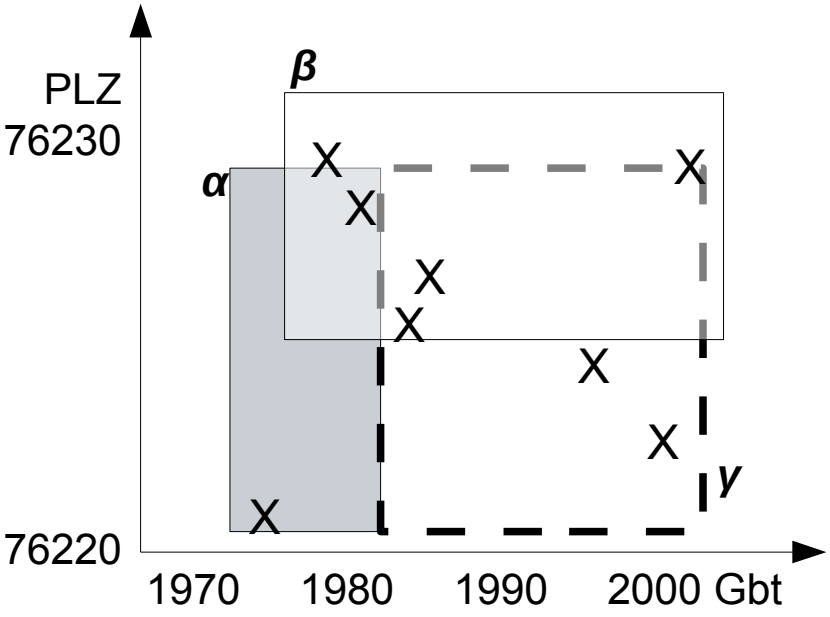
| enc | B |
|---|---|
| 770E5B0F8 | β |
| 86D3EFBC2 | α |
| EB51DA3C6 | γ |
| 15487E7B5 | β |
| 8ABE19436 | γ |
| C2AB7DA00 | α |
| 2814F82C4 | β |
| 9B7DFAFA0 | γ |

- Anfrageergebnis berechnen
  - False Positives herausfiltern

| | | | |
|---|---|---|---|
| Eve | 1985 | 76227 | Husten |
| George | 1983 | 76226 | Heiser |

# Angreifbarkeit

- Angreifer kennt DB, darf Daten einfügen
  - Welches Bucket Label bekommt neuer Datensatz?
- Angreifer will Wertverteilung und/oder Näherungswerte auslesen
- Ausprobieren von Bucket Labels
  - MIN/MAX der Werte im Bucket werden offenbar
  - Jedes neue Label verrät detailliertere Informationen




| enc | B |
|---|---|
| 770E5B0F8 | β |
| 86D3EFBC2 | α |
| EB51DA3C6 | α |
| 15487E7B5 | β |
| 8ABE19436 | α |
| C2AB7DA00 | α |
| 2814F82C4 | β |

+γ

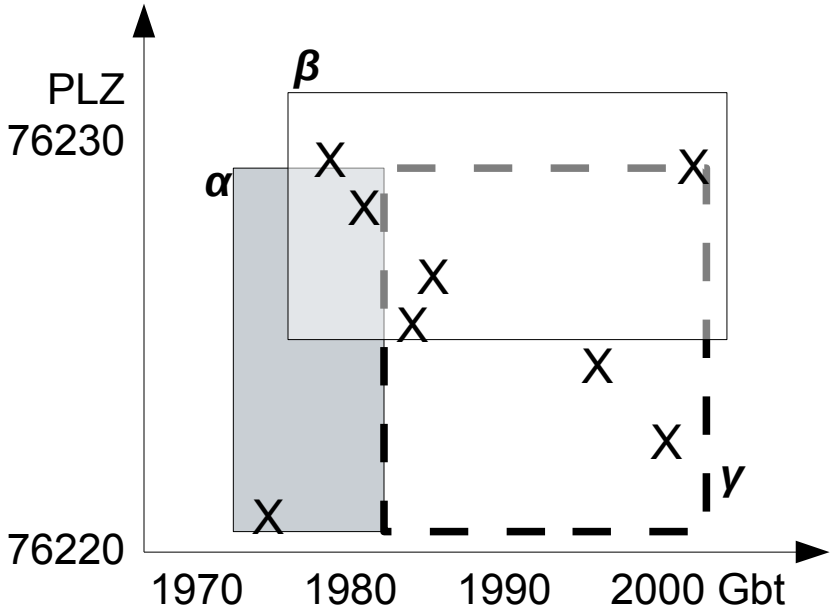
| enc | B |
|---|---|
| 770E5B0F8 | β |
| 86D3EFBC2 | α |
| EB51DA3C6 | γ |
| 15487E7B5 | β |
| 8ABE19436 | γ |
| C2AB7DA00 | α |
| 2814F82C4 | β |
| 9B7DFAFA0 | γ |

# Angreifbarkeit, formal

- *„Wie gut kann ein Angreifer einen Attributwert aus dem Bucket Label vorhersagen?“*

- Maß basiert auf der Entropie eines Buckets
  - Information einer Variable X über eine Variable Y:
    $H(X|Y) = -\sum_{x \in X,\, y \in Y} p(x,y) \log p(x|y)$

  - *Anmerkung: Ziel ist das Maximieren der Entropie eines Buckets!*



# Optimale Verteilung der Buckets

- Extrem 1: Alle Werte in dem selben Bucket
  - Hohes Maß an Privatheit, keinerlei Rückschlüsse auf Attributverteilung
  - Maximale Anzahl von False Positives, schlechtestmögliche Performanz
- Extrem 2: Alle Werte in jeweils eigenem Bucket
  - Niedrigstes Maß an Privatheit, Buckets geben Attributverteilung wieder
  - Keine False Positives, aber riesengroße Anfragen beim Query Rewriting
- Was ist das richtige Maß?
  1) Zielfunktion für ein Optimierungsverfahren
     - wenn alle Bereichsanfragen gleich wahrscheinlich, M Buckets, jedes Bucket t mit $F_t$ Datensätzen und Kantenlänge $b_{t,d}$ in Dimension $d \in D$:
       $cost = \sum_{t = 1...M} (F_t * \sum_{d = 1...D} b_{t,d})$
  2) Enthüllungsrisiko von vorangegangenen Folien
     - Maß für Risiko der Aufdeckung von sensiblen Daten

# Algorithmus zur Bucket-Einteilung

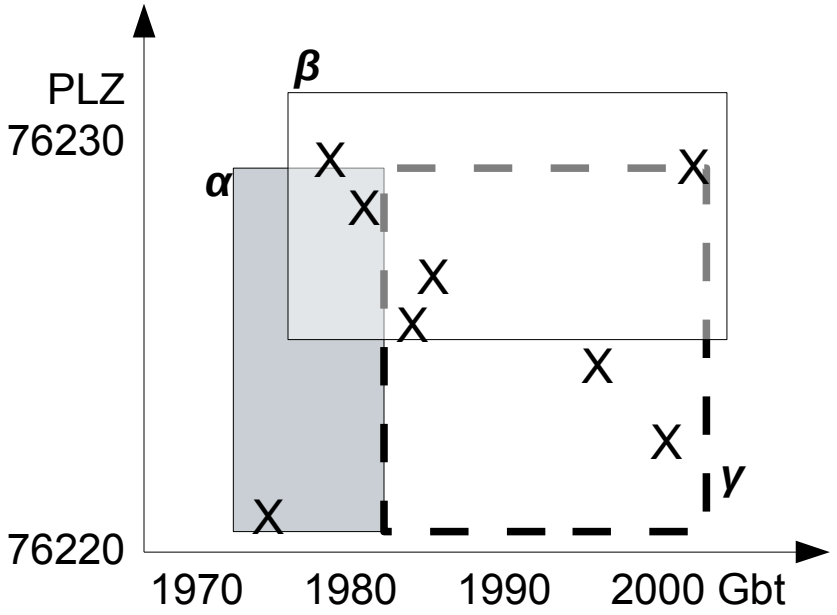
- geg.: Datenbank db
  Anzahl Buckets M
  Performance degradation factor K
  → max. K mal mehr false Positives als mit optimaler Bucketaufteilung
- Algorithmus in 2 Schritten:

*1. Query-Optimal Bucketization*
starte mit einem einzelnen Bucket, das alle Datensätze in db enthält
**for** j = 1 **to** M-1
    suche 2 Datensätze als neue Bucket-Grenzen, die die Kosten minimieren
    sortiere alle passenden Datensätze in neues Bucket ein
    berechne neue Bucket-Grenzen als Minimum Bounding Rectangles
**end for**
*2. Controlled Diffusion*
**for each** Bucket B
    verteile K * |B| * M / |db| Datensätze zufällig auf andere Buckets
    berechne neue Bucket-Grenzen als Minimum Bounding Rectangles
**end for**
**return** M buckets

# Diskussion multidim. Bucketization

- Vorteile
  - Verfahren zum Einstellen eines Kompromisses aus
    - Performanz (false positives) und
    - Datenschutz (Entropie der Buckets)

- Nachteile
  - Nur Greedy-Lösung für Bucket-Einteilung
    - Ggf. suboptimale Lösung
    - Abhängig von den Start-parametern
  - Nur performant bei mehrdimensionalen Bereichsanfragen
    - Bei anderen Anfragen viele false positives



# Fragmentation

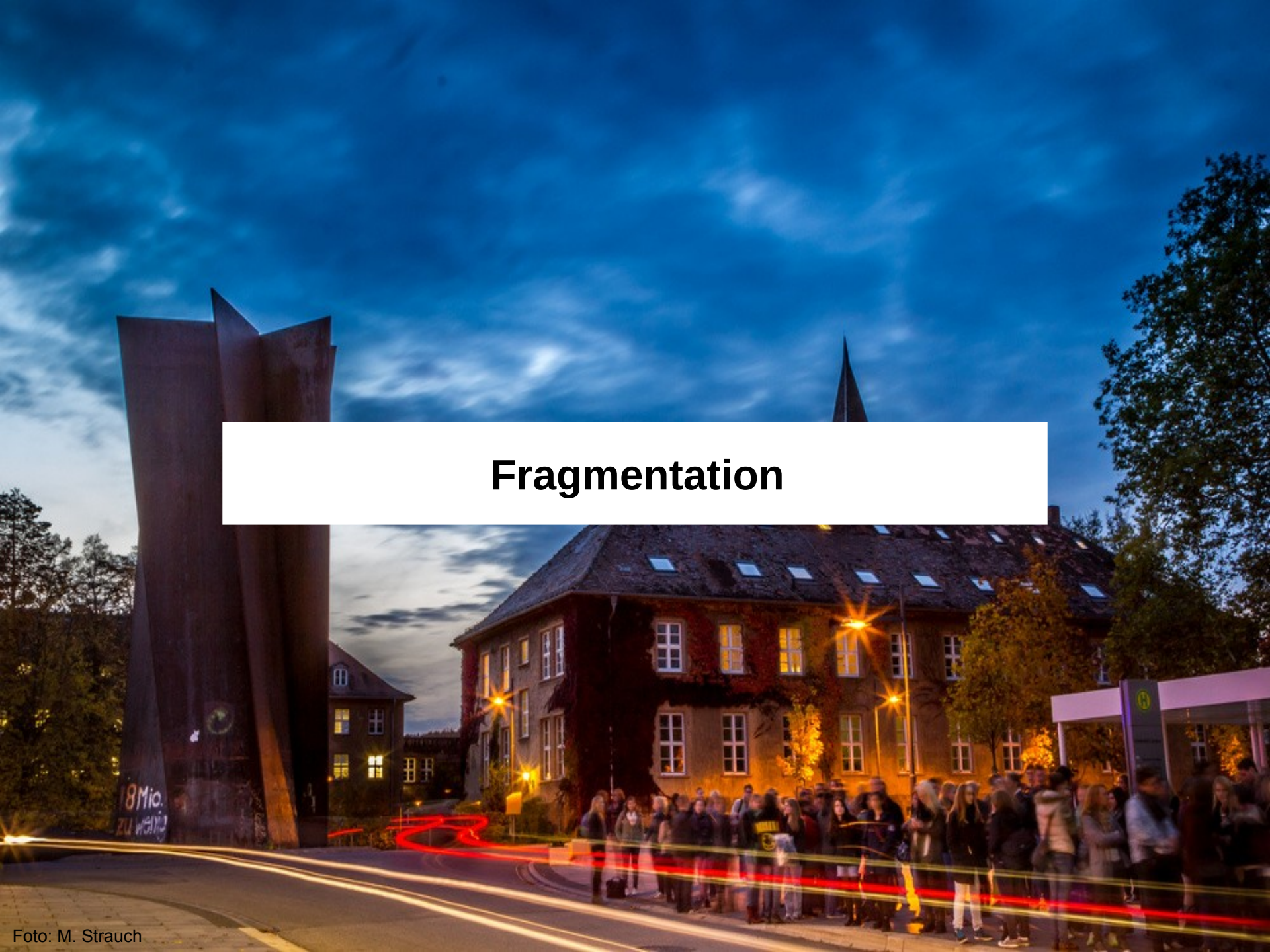


Foto: M. Strauch

# Schutzziel

- Angreifermodell
  - Angreifer ist **Honest-but-Curious**
  - Angreifer hat Zugang zur Datenbank (Outsourcing-Szenario)
- Angreifer will
  - Zuordnung von sensitiven Attributen erfahren
    - z.B. VersNr-Name, Name-Diagnose etc.
  - Attributverteilung ist egal, z.B. die in der DB enthaltenen Diagnosen
    - kein Schutz gegen Frequenzangriffe, **einzelne Attribute im Klartext**

| VersNr | Name | Gbt | PLZ | Arzt | Diagnose |
|---|---|---|---|---|---|
| 123 | Alice | 1981 | 76227 | Dr. Brown | Schnupfen |
| 456 | Bob | 1974 | 76221 | Dr. Red | Husten |
| 789 | Carol | 1995 | 76221 | Dr. Olive | Heiser |
| 101 | Dave | 2001 | 76229 | Dr. Blue | Schnupfen |
| 112 | Alice | 1981 | 76227 | Dr. Red | Husten |

# Systemmodell

- Datenbesitzer legt Vertraulichkeitsregeln (Constraints) fest
  - Welche Attribute und Zuordnungen **nicht** offenlegen?


| VersNr | Name | Gbt | PLZ | Arzt | Diagnose |
|---|---|---|---|---|---|
| 123 | Alice | 1981 | 76227 | Dr. Brown | Schnupfen |
| 456 | Bob | 1974 | 76221 | Dr. Red | Husten |
| 789 | Carol | 1995 | 76221 | Dr. Olive | Heiser |

$C_0$: {VersNr}
$C_1$: {Name,Gbt}
$C_2$: {Name,PLZ}
$C_3$: {Name,Diagnose}
$C_4$: {Name,Arzt}
$C_5$: {Gbt,PLZ,Diagnose}
$C_6$: {Gbt,PLZ,Arzt}

- Dienstleister erhält
  - **Teilweise** verschlüsselte Fragmente der Datenbank, die nur vom Schlüsselbesitzer zusammenzuführen sind [2]

$F_1$

| salt | enc | Name |
|---|---|---|
| $S_1$ | #### | Alice |
| $S_2$ | #### | Bob |
| $S_3$ | #### | Carol |

$F_2$

| salt | enc | Gbt | PLZ |
|---|---|---|---|
| $S_4$ | #### | 1981 | 76227 |
| $S_5$ | #### | 1974 | 76221 |
| $S_6$ | #### | 1995 | 76221 |

$F_3$

| salt | enc | Arzt | Diagnose |
|---|---|---|---|
| $S_7$ | #### | Dr. Brown | Schnupfen |
| $S_8$ | #### | Dr. Red | Husten |
| $S_9$ | #### | Dr. Olive | Heiser |

# Fragmente

- Relationenschema R = $\{A_0, A_1, \ldots, A_n\}$, Constraints C = $\{c_0, c_1, \ldots, c_k\}$
- Jedes Fragment speichert
  - **salt**
    - beliebige Zufallszahl im Klartext, Primärschlüssel in R
  - **Sichtbare Attribute**
    - Nicht-sensible Attribute
      $\{A_i, \ldots, A_j\} \subseteq R$
  - **enc**
    - Verschlüsselung aller restlichen Attribute eines Tupels XOR salt
      enc = encode( $[R - \{A_i, \ldots, A_j\}] \otimes$ salt )

$F_1$

| salt | enc | Name |
|---|---|---|
| 124 | encode({123, 1981, 76227, Dr. Brown, Schnupfen} ⊗ 124) | Alice |
| 893 | encode({456, 1974, 76221, Dr. Red, Husten} ⊗ 893) | Bob |
| 410 | encode({789, 1995, 76221, Dr. Olive, Heiser} ⊗410) | Carol |

# Anfrageverarbeitung auf Fragmenten

- Query Rewriting auf passendes Fragment
  - SELECT Diagnose FROM db WHERE Name = 'Alice' AND PLZ = 76227
  - SELECT salt, enc FROM $F_1$ WHERE Name = 'Alice'

    oder SELECT salt, enc FROM $F_2$ WHERE PLZ = 76227

    oder SELECT salt, enc FROM $F_3$ (*je nach Selektivität des Fragments*)
- Client dekodiert Ergebnis vom Server
  - dec = decode(enc) ⊗ salt
- Client berechnet Anfrageergebnis
  - SELECT Diagnose FROM dec WHERE Name = 'Alice' AND PLZ = 76227

$F_1$

| salt | enc | Name |
|---|---|---|
| 124 | encode({123, 1981, 76227, Dr. Brown, Schnupfen} ⊗ 124) | Alice |
| 893 | encode({456, 1974, 76221, Dr. Red, Husten} ⊗ 893) | Bob |
| 410 | encode({789, 1995, 76221, Dr. Olive, Heiser} ⊗410) | Carol |

# Constraints

- Relationenschema R = $\{A_0, A_1, \ldots, A_n\}$,
  Constraint $c \subseteq R$
  - $|c| = 1$ (nur 1 Attribut): **Attribut** nicht veröffentlichen ($c_0$)
  - $|c| > 1$: **Assoziation** zwischen den Attributen nicht veröffentlichen ($c_1 \ldots c_6$)
  - alle anderen Attribute und Attributkombinationen veröffentlichbar
    (z.B. {Arzt, Gbt} oder {Diagnose, PLZ} sind erlaubt)

- W**ohldefiniertheit** einer Menge von Constraints $C = \{c_0, c_1, \ldots, c_k\}$
  - Constraints dürfen sich nicht überschneiden
    $\forall\, c_i, c_j \in C: i \neq j \wedge c_i \not\subset c_j$

db

| VersNr | Name | Gbt | PLZ | Arzt | Diagnose |
|---|---|---|---|---|---|
| 123 | Alice | 1981 | 76227 | Dr. Brown | Schnupfen |
| 456 | Bob | 1974 | 76221 | Dr. Red | Husten |
| 789 | Carol | 1995 | 76221 | Dr. Olive | Heiser |

$C_0$: {VersNr}
$C_1$: {Name,Gbt}
$C_2$: {Name,PLZ}
$C_3$: {Name,Diagnose}
$C_4$: {Name,Arzt}
$C_5$: {Gbt,PLZ,Diagnose}
$C_6$: {Gbt,PLZ,Arzt}

# Optimale Fragmentierung

- Korrektheit
  - Keine Informationspreisgabe bezüglich Constraints
- Maximale Sichtbarkeit
  - Möglichst viele Attribute im Klartext
  - Ausfiltern von unnötigen Zwischenergebnissen ohne Entschlüsselung
- Minimale Fragmentierung
  - Möglichst wenig Fragmente mit nur wenigen Attributen
  - Anfragen mit mehreren Attributen effizient ausführen

# Korrektheit

R: { VersNr,
Name,
Gbt,
PLZ,
Arzt,
Diagnose}

$C_0$: {VersNr}
$C_1$: {Name,Gbt}
$C_2$: {Name,PLZ}
$C_3$: {Name,Diagnose}
$C_4$: {Name,Arzt}
$C_5$: {Gbt,PLZ,Diagnose}
$C_6$: {Gbt,PLZ,Arzt}

- geg.: Relation $R = \{A_0, A_1, \ldots, A_n\}$,
  Constraints $C = \{c_0, c_1, \ldots, c_k\}$
- Korrektheit der Fragmentierung
  $F = \{F_0, F_1, \ldots, F_k\}$
  - Kein Fragment mit verbotenen Attributkombinationen
    $\forall f \in F, \forall c \in C: c \not\subseteq f$
  - Fragmente haben keine gemeinsamen Attribute (Verknüpfbarkeit)
    $\forall p,q \in F: p \cap q = \emptyset$
- Zahlreiche korrekte Fragmentierungen, welche ist die beste?
  - { }
  - {Name}
  - {Gbt, PLZ}
  - {PLZ, Arzt, Diagnose}
  - {Name}, {Gbt, Arzt, Diagnose}, {PLZ}
  - {Name}, {Gbt, PLZ}, {Arzt, Diagnose}
  - {Name}, {Gbt}, {PLZ, Arzt, Diagnose}
  - {Name}, {Gbt}, {PLZ}, {Arzt}, {Diagnose}

# Maximale Sichtbarkeit

R: { VersNr,
Name,
Gbt,
PLZ,
Arzt,
Diagnose}

$C_0$: {VersNr}
$C_1$: {Name,Gbt}
$C_2$: {Name,PLZ}
$C_3$: {Name,Diagnose}
$C_4$: {Name,Arzt}
$C_5$: {Gbt,PLZ,Diagnose}
$C_6$: {Gbt,PLZ,Arzt}

- geg.: Relation $R = \{A_0, A_1, \ldots, A_n\}$,
  Fragmente $F = \{F_0, F_1, \ldots, F_k\}$
- viele Attribute im Klartext
  - Client muss kleinere Zwischenergebnisse entschlüsseln, wenn Server Tupel ausfiltern kann
- Alle Attribute in R, die nicht in einem einelementigen Constraint enthalten sind, sind Bestandteil eines Fragments
  - $\forall a \in R, \{a\} \notin C: \exists f \in F$ so dass $a \in f$
- Welche Fragmentierung ist optimal?
  - { }
  - {Name}
  - {Gbt, PLZ}
  - {PLZ, Arzt, Diagnose}
  - {Name}, {Gbt, Arzt, Diagnose}, {PLZ}
  - {Name}, {Gbt, PLZ}, {Arzt, Diagnose}
  - {Name}, {Gbt}, {PLZ, Arzt, Diagnose}
  - {Name}, {Gbt}, {PLZ}, {Arzt}, {Diagnose}

# Minimale Fragmentierung

- Je weniger Fragmente, desto besser
  - Server kann eventuell Anfragen mit mehr als einem Attributen bearbeiten
    SELECT * FROM db WHERE Arzt = 'Dr. Red' AND Diagnose='Husten'
- geg.: zwei korrekte Fragmentierungen F, F' mit max. Sichtbarkeit
- F ist minimal wenn kein **korrektes** F' mit **max. Sichtbarkeit** existiert, das weniger Fragmente enthält
  - es ist unmöglich, aus zwei Fragmenten in F eines in F' zu bauen
  - ∄F' so dass |F'| < |F|'
- *Es kann mehrere minimale Fragmentierungen geben*
  - { }
  - {Name}
  - {Gbt, PLZ}
  - {PLZ, Arzt, Diagnose}
  - {Name}, {Gbt, Arzt, Diagnose}, {PLZ}
  - {Name}, {Gbt, PLZ}, {Arzt, Diagnose}
  - {Name}, {Gbt}, {PLZ, Arzt, Diagnose}
  - {Name}, {Gbt}, {PLZ}, {Arzt}, {Diagnose}

# Query Performance

- Anfragen an einzelne sichtbare Attribute
  - Performanz besser als Bucketization
    - Kein Entschlüsseln von „false Positives“, d.h. nicht gesuchten Attributen im gleichen Bucket
    - Bereichsanfragen, Gruppierung, Sortierung etc. möglich
- Selektionsanfragen auf unsichtbare Attribute, Selektionen über mehrere Attribute, Verbundanfragen
  - Client muss vollständige Fragmente herunterladen und lokal entschlüsseln

$F_1$

| salt | enc | Name |
|---|---|---|
| $S_1$ | #### | Alice |
| $S_2$ | #### | Bob |
| $S_3$ | #### | Carol |

$F_2$

| salt | enc | Gbt | PLZ |
|---|---|---|---|
| $S_4$ | #### | 1981 | 76227 |
| $S_5$ | #### | 1974 | 76221 |
| $S_6$ | #### | 1995 | 76221 |

$F_3$

| salt | enc | Arzt | Diagnose |
|---|---|---|---|
| $S_7$ | #### | Dr. Brown | Schnupfen |
| $S_8$ | #### | Dr. Red | Husten |
| $S_9$ | #### | Dr. Olive | Heiser |

# Diskussion Fragmentation

- Vorteile
  - Detaillierte Schutzziele definierbar
  - Einfach einzusetzen, keine großen Änderungen an existierenden DBMS
  - Sehr gute Performance bei Anfragen auf einzelne sichtbare Attribute
- Nachteile
  - Relativ niederige Schutzziele, Attributverteilung etc. offen
  - Verbundanfragen oder Anfragen über mehrere Fragmente erfordern herunterladen und entschlüsseln des kompletten Fragments

$F_1$

| salt | enc | Name |
|---|---|---|
| $S_1$ | #### | Alice |
| $S_2$ | #### | Bob |
| $S_3$ | #### | Carol |

$F_2$

| salt | enc | Gbt | PLZ |
|---|---|---|---|
| $S_4$ | #### | 1981 | 76227 |
| $S_5$ | #### | 1974 | 76221 |
| $S_6$ | #### | 1995 | 76221 |

$F_3$

| salt | enc | Arzt | Diagnose |
|---|---|---|---|
| $S_7$ | #### | Dr. Brown | Schnupfen |
| $S_8$ | #### | Dr. Red | Husten |
| $S_9$ | #### | Dr. Olive | Heiser |


# Zum Abschluss

Foto: M. Strauch

# Literatur


[1] Alberto Ceselli et al.: *Modeling and Assessing Inference Exposure in Encrypted Databases.* In: ACM Transactions on Information and System Security, 8(1), 2005

[2] Valentina Ciriani et al.: *Combining Fragmentation and Encryption to Protect Privacy in Data Storage.* In: ACM Transactions on Information and System Security, 13(3), 2010

[3] Bijit Hore et al.: *Secure Multidimensional Range Queries over Outsourced Data*. In: VLDB Journal 21/2012

[4] Stefan Hildenbrand et al.: *Query Processing on Encrypted Data in the Cloud,* Technical Report 735, ETH Zürich, 2011

# Wie geht es weiter?

- bis Montag, 20.07., 12 Uhr
  - allerletztes Quiz: Indexierung
- Dienstag, 21.07., GHH 12-14 Uhr: Tutoriumstermin
  - kurze Besprechung von Aufgabenblatt 11
  - letztes Aufgabenblatt des Semesters
- Donnerstag, 22.07.: Präsenztermin
  - Index-Verwaltung in SQL
- *Donnerstag, 30.07.: Große Fragerunde zur Endklausur*
- *Donnerstag, 06.08.: Endklausur, Beginn 13:00 Uhr*