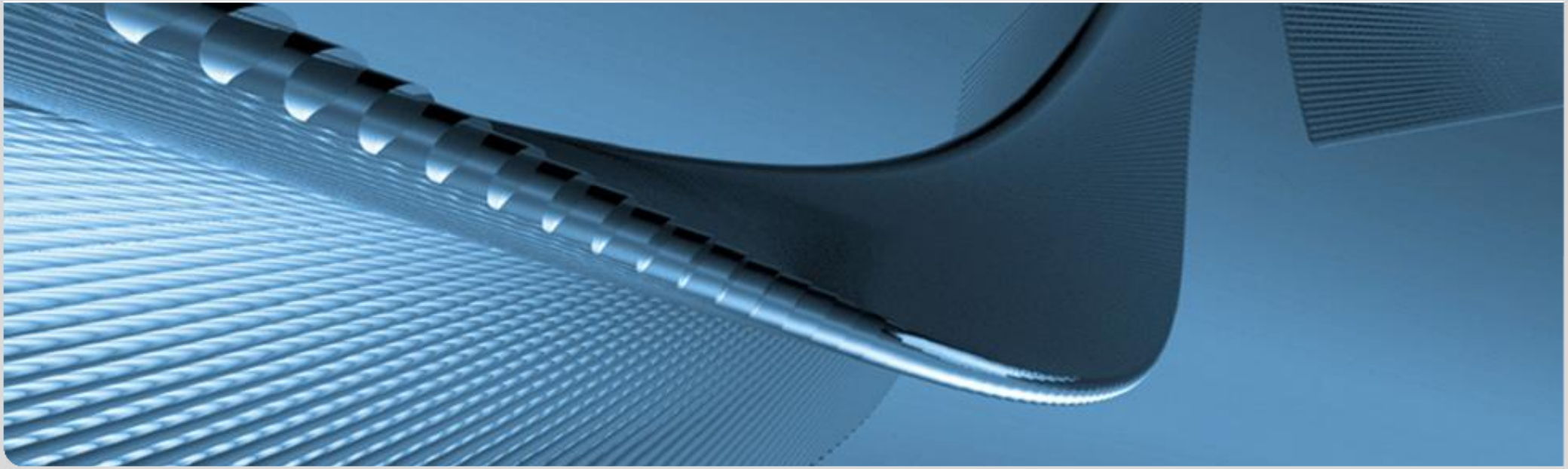
Karlsruher Institut für Technologie

Datenschutz und Privatheit in vernetzten Informationssystemen

Kapitel 8: Verschlüsseltes Datenmanagement

Erik Buchmann (buchmann@kit.edu)

IPD, Systeme der Informationsverwaltung, Nachwuchsgruppe „Privacy Awareness in Information Systems“



Inhalte und Lernziele dieses Kapitels

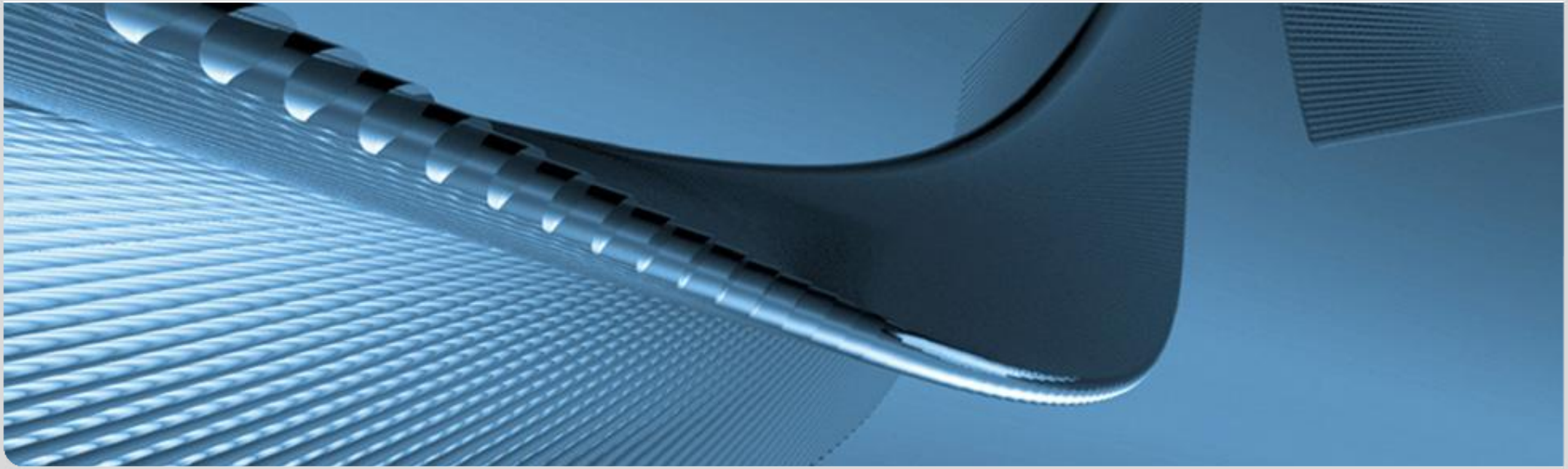
- Einführung: Verschlüsselung und Datenhaltung
- Bucket-Verfahren
  - Partially Order Preserving Encryption
  - Bucketization
  - Mehrdimensionale Bucketization
- Fragmentation
- Homomorphe Verschlüsselung
- Abschluss

- Lernziele
  - Sie können die Datenschutzprobleme erläutern, die sich durch das Auslagern von Datenbank-Funktionalität ergeben.
  - Sie können die vorgestellten Verfahren detailliert erläutern.
  - Anhand einer Vorgabe aus Datenschutzzielen, Anfragetypen und Performanzkriterien können Sie ein geeignetes Verfahren auswählen.

Karlsruher Institut für Technologie

Verschlüsselung und Datenhaltung

IPD, Systeme der Informationsverwaltung, Nachwuchsgruppe „Privacy Awareness in Information Systems“



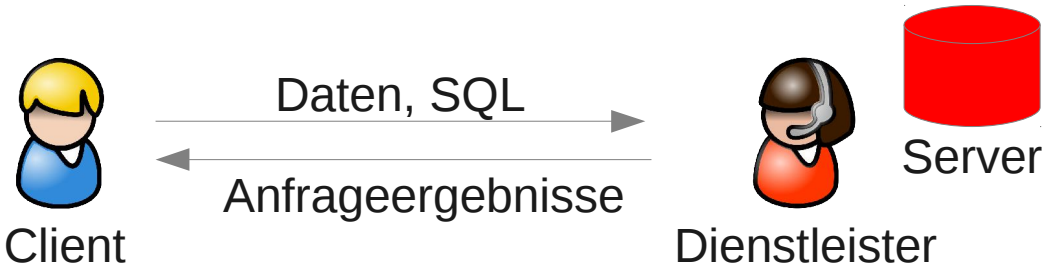
Der Betrieb großer Datenbanken...

- ...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

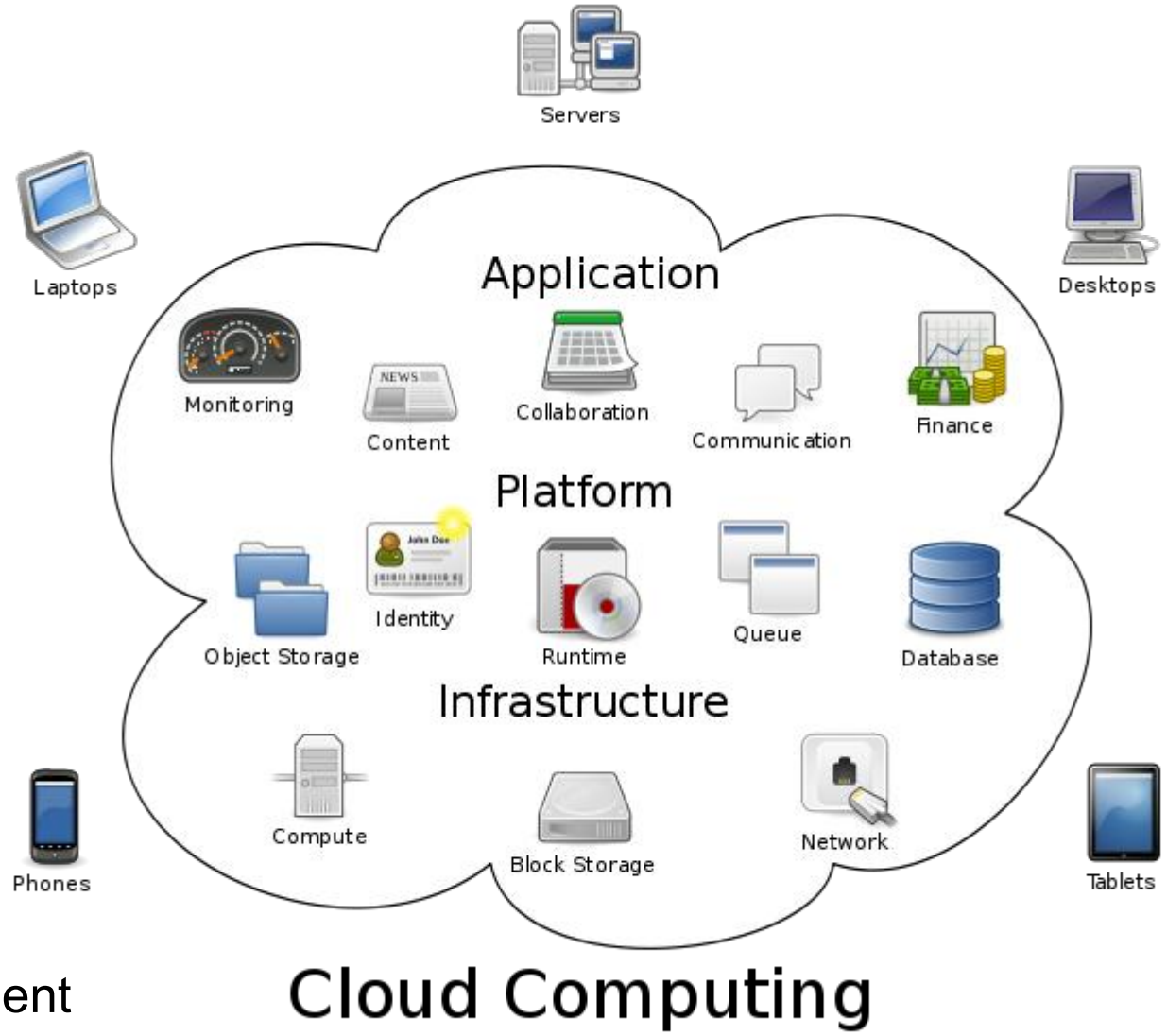
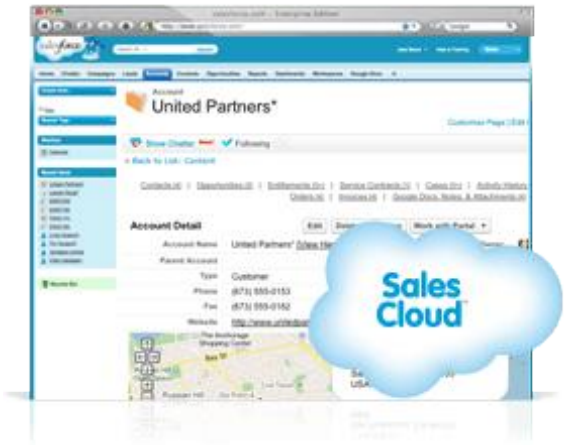


Bild: Wikimedia Commons

Arten von Cloud-Dienstleistungen

- Software as a Service
  - Software installiert vom Service-Provider
  - Zugriff über Webbrowser vom Endgerät
  - z.B. CRM wie Salesforce

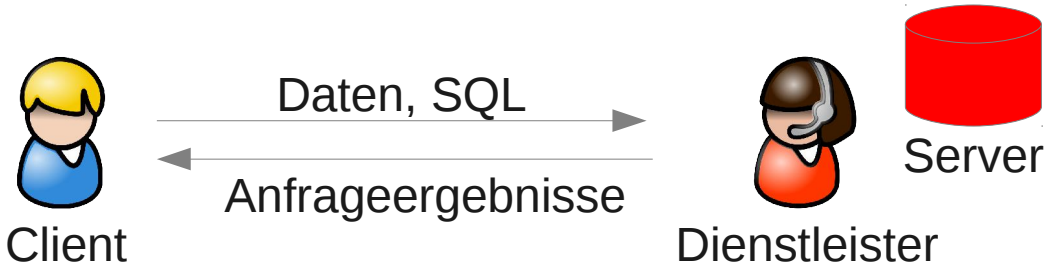


- Platform as a Service
  - Speicher, Backup, Zugriffsschutz bereitgestellt vom Provider
  - Zugriff über Anwendungen auf Endgerät
  - z.B. Amazon S3

- Infrastructure as a Service
  - Rechenleistung bereitgestellt vom Provider
  - Zugriff über Middleware-Frameworks wie Hadoop, MapReduce
  - z.B. Google App Engine

Datenschutz, Vertraulichkeit?

- Daten auf fremden Servern, ggf. im Ausland
  - Datenschutzpflichten
  - Schutz von Geschäftsgeheimnissen
- Fremdanbieter sollen mit Daten arbeiten, sie aber nicht einsehen können
  - Zugriffsschutz nicht ausreichend
- Im Folgenden
  - Systemarchitektur
  - Angreifermodell
  - Anforderungen



Architekturmodell in dieser Vorlesung

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

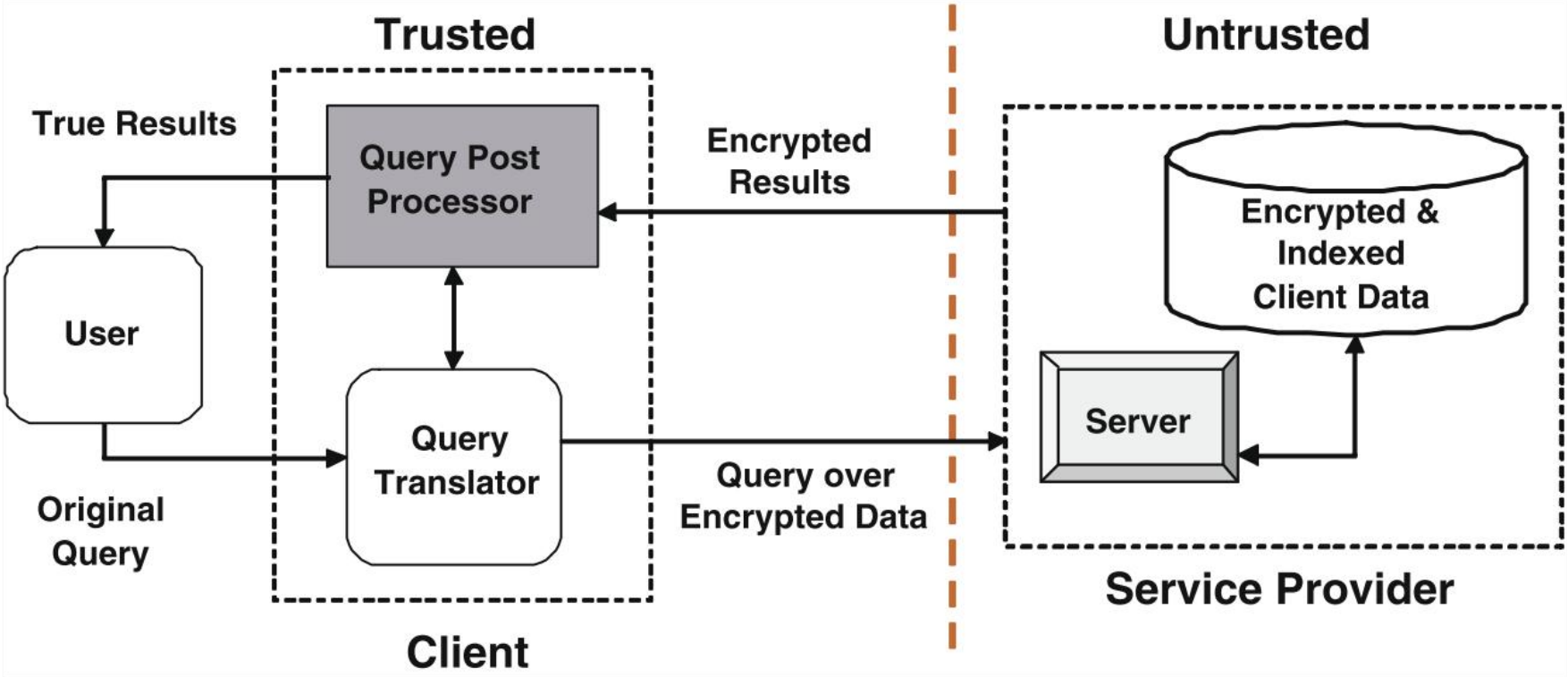
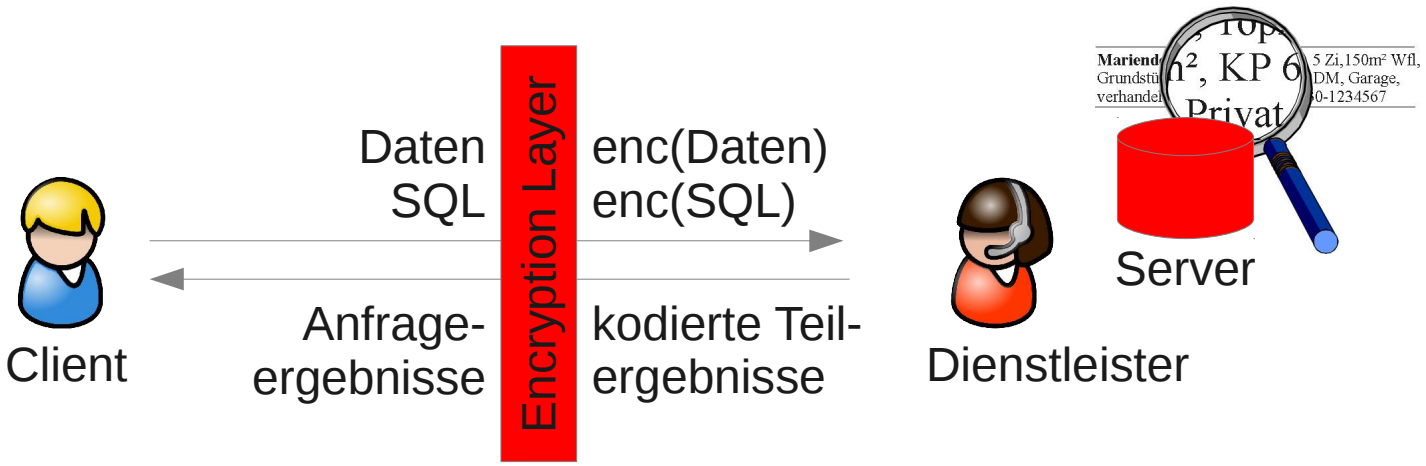


Bild: [3]

Angreifermodell

- 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



Angriffe *(Wiederholung)*

- Zuordnungsangriffe
  - Ausprobieren, ob man den Klartext erraten kann
  - Beispiel: Befindet sich „Alice“ in einer verschlüsselten DB?
    - Angreifer kennt Public Key: „Alice“ → „H71A00EF91“

Person	Diagnose	Alter
AA73F9E001	Magersucht	24
H71A00EF91	Diabetes	31
3429A9F64E	Schnupfen	26

- Frequenzangriffe
  - Aus der Verteilung der Ciphertexte kann auf Inhalte geschlossen werden
  - Beispiel: Hat „Alice“ in verschlüsselter Datenbank Urlaub genommen?
    - Angreifer weiß, dass am Jahresanfang „30 Tage“ häufigster Eintrag

Person	Urlaubstage
Alice	**A7**
Bob	EF
Carol	**A7**
Dave	**A7**
Eve	21
Frank	FF

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

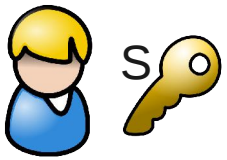
```
SELECT AVG(Gbt) FROM db
GROUP BY Diagnose
HAVING AVG(Gbt) > 1985
```

db

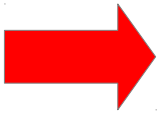
Name	Gbt	PLZ	Diagnose
Alice	1981	76227	Schnupfen
Bob	1974	76221	Husten
Carol	1995	76221	Heiser
Dave	2001	76229	Schnupfen
Alice	1981	76227	Husten

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

- Frequenz- und Zuordnungsangriffe möglich?
- 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 and B.Diagnose from db as A, db as B where A.Name = B.Name

Performance vs. begrenzte Offenlegung

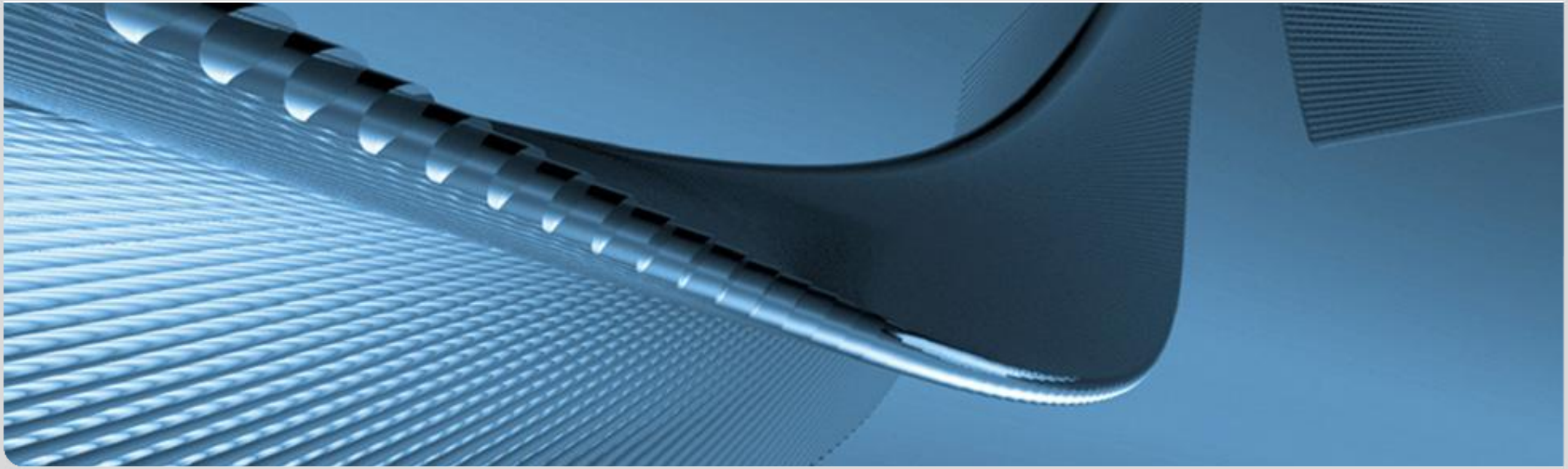
- Guter Datenschutz
  - Starke Verschlüsselung ist langsam
    - Insbes. probabilistische Verschlüsselung
  - Hoher Aufwand gegen Frequenz- und Zuordnungsangriffe
    - Dienstleister kann Anfragen und Zugriffe auf Server mitloggen
  - Client muss festlegen können, welche Informationen offenbar werden dürfen

- Gute Performanz
  - Erzwingt einfache Verfahren, Risiko für Offenlegung steigt
  - Anfrageausführung auf dem Server
    - Wenn Server den Schlüssel benötigt, ist nichts gewonnen
  - Möglichst kleine Zwischenergebnisse zum Client schicken
    - Wenn Client die ganze Datenbank lokal herunterladen und dekodieren muss, ist nichts gewonnen

Karlsruher Institut für Technologie

Partially Order Preserving Encryption

IPD, Systeme der Informationsverwaltung, Nachwuchsgruppe „Privacy Awareness in Information Systems“



Schutzziel

- Angreifermodell
  - Angreifer ist **Honest-but-Curious**
  - Angreifer hat Zugang zur Datenbank
- Angreifer will
  - Datenverteilung erfahren
  - Zuordnung von sensitiven Attributen erfahren
    → begrenzter Schutz gegen Frequenzangriffe, 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

Partially Order Preserving Encryption [4]

- Daten so auf kryptographische Codes abbilden, dass
  - **Sortierreihenfolge** der Codes gleich (ähnlich) der Reihenfolge der Originaldaten ist
  - SQL-Anfragen Operationen vom Dienstleister **ohne Kenntnis des Schlüssels** auf den verschlüsselten Daten berechnet werden können

- Laufendes Beispiel:

```
select count(*) from table
where PLZ >= 76130
      and PLZ < 76140
      and Diagnose = 'Magersucht'
```

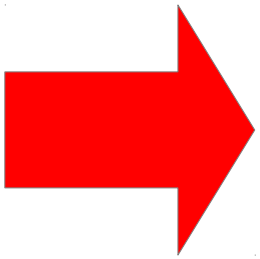
Ergebnis: 2

PLZ	Diagnose
76127	Impotenz
76128	Hodenkrebs
76129	Sterilität
76131	Schizophrenie
76133	Diabetes
76133	Magersucht
76136	Magersucht

Grundlage: Order Preserving Encryption

- Abbildung der Daten auf ordnungserhaltende Codes

PLZ	Diagnose
76127	Impotenz
76128	Hodenkrebs
76129	Sterilität
76131	Schizophrenie
76133	Diabetes
76133	Magersucht
76136	Magersucht



K1S	AS9
15	100
16	80
29	154
56	153
70	53
70	104
71	104

- Umschreiben der Anfragen

```
select count(*) from table
where PLZ >= 76130
and PLZ < 76140
and Diagnose = 'Magersucht'
```

```
select count(*) from table
where K1S >= 30
and K1S < 100
and AS9 = 104
```

OP-Encryption ist angreifbar

- Frequenzangriffe
  - *Hintergrundwissen*: Magersucht ist die häufigste Diagnose in Table
    → Magersucht hat Code 104
  - *Hintergrundwissen*: PLZ-Bezirk 76133 hat die meisten Einwohner
    → 76133 hat Code 70
- Domänenangriffe
  - *Hintergrundwissen*: Im Krankenhaus wird Sterilität und Schizophrenie behandelt; Magersucht hat Code 104
    → lexikalisch „S...“ > „M...“, also hat Schizophrenie Code 153 und Sterilität Code 154

K1S	AS9
15	100
16	80
29	154
56	153
70	53
70	104
71	104

Partially Order Preserving Encryption

- Unterteile das Data Set zufällig in mehrere Fragmente
  - Zahl der Fragmente bestimmt Dekodierungsaufwand
  - Zahl der Fragmente bestimmt Sicherheit gegen Angriffe
- Führe Order Preserving Encryption auf jedes Fragment aus
  - Ordnungen zwischen den Fragmenten unabhängig
- Beispiel für Spalte PLZ:

Original	1 Fragment	2 Fragmente	3 Fragmente	7 Fragmente
76127	1,15	1,15	3,1	1,0
76128	1,16	2,1	2,1	3,12
76129	1,29	1,29	1,2	7,12
76131	1,56	1,50	1,5	4,15
76133	1,70	2,70	2,7	2,12
76133	1,70	2,70	3,2	6,0
76136	1,71	1,70	1,7	5,7

Anfrageausführung

- Query Rewriting
  - Original
    select count(*) from table
    where PLZ >= 76130 and PLZ < 76140
  - Für 2 Fragmente
    select count(*) from table
    where (K1S >= 1,30 and K1S < 1,100)
    or (K1S >= 2,2 and K1S < 2,80)

- Aufwand der Anfrageausführung abhängig von der Zahl der Fragmente
  - Je mehr Fragmente, desto
    - mehr Metadaten muss der Verschlüsselungslayer speichern
    - aufwändiger die Anfrage
    - weniger funktionieren Indexe und Optimierungen der Datenbank

PLZ	K1S, 2 Fragmente
76127	1,15
76128	2,1
76129	1,29
76131	1,50
76133	2,70
76133	2,70
76136	1,70

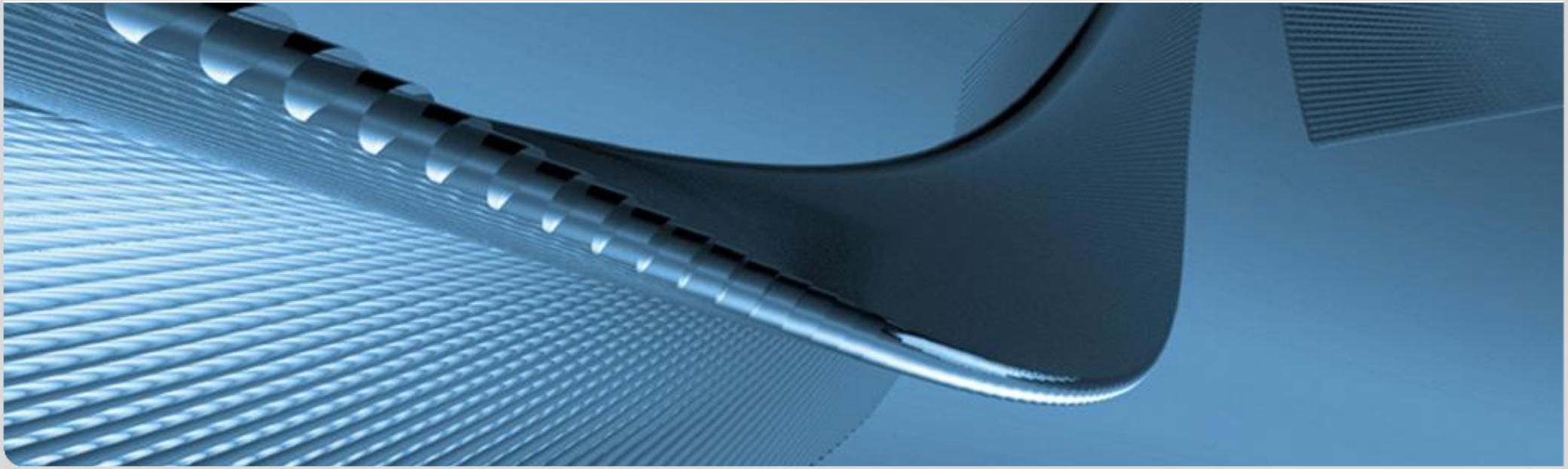
Diskussion

- POP-Encryption geeignet für viele Datenbankoperationen
  - Selektion, Verbund, Gruppierung, Mengenoperationen, Aggregate...
  - Nicht: arithmetische Operationen
- Einstellbarer Kompromiß zwischen Sicherheit und Performanz
  - nur ein Fragment: Order Preserving Encryption
  - soviele Fragmente wie Datensätze: herkömmliche Verschlüsselung
- Leicht realisierbar
  - keine Änderungen beim Cloud-Anbieter,
  - Encryption-Layer in der Middleware beim Dienstnutzer
- Sicherheit und Performanz ist abhängig von der Datenverteilung
  - „schiefe“ Verteilung erfordert zum Verbergen viele Fragmente
  - Anwendungen, die Datenschutz UND Performanz brauchen?

Karlsruher Institut für Technologie

Bucketization

IPD, Systeme der Informationsverwaltung, Nachwuchsgruppe „Privacy Awareness in Information Systems“



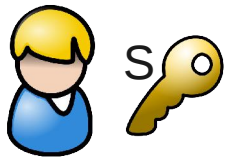
Schutzziel

- 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**)

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

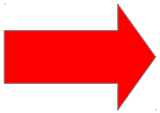
Systemmodell

- 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

- Query Rewriting
  - SELECT Diagnose FROM db
    WHERE PLZ = 76221
  - SELECT enc FROM encdb
    WHERE I_P = 'II'
- Zwischenergebnis vom Server entschlüsseln
  - 86D3EFBC2 → {Bob, 76221, Husten}
  - EB51DA3C6 → {Carol, 76221, Heiser}
- Anfrageergebnis berechnen
  - SELECT Diagnose FROM db
    WHERE PLZ = 76221

Husten
Heiser

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

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

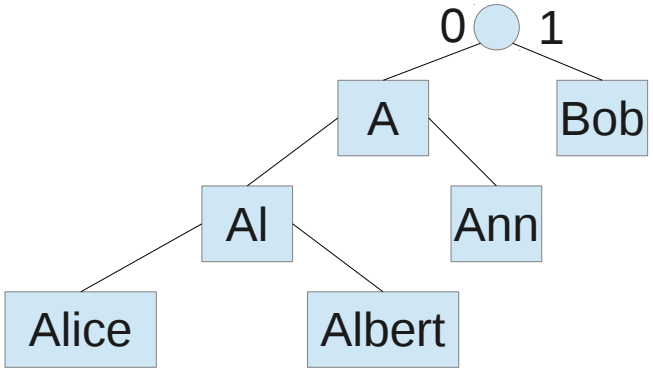
Name	PLZ	Diagnose
Alice	76227	Schnupfen
Bob	76221	Husten
Carol	76221	Heiser
Dave	76229	Schnupfen
Alice	76227	Husten

encdb

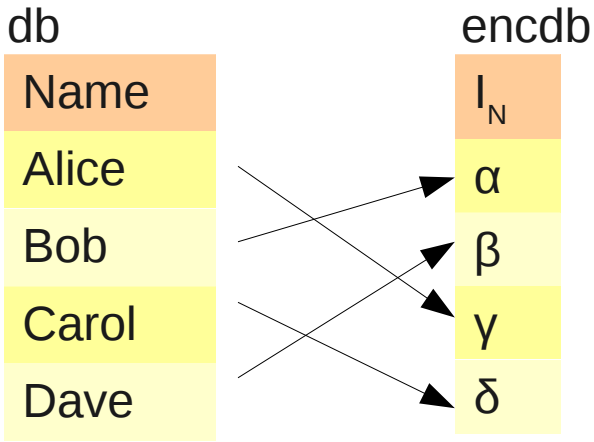
enc	I_N	I_P	I_D
770E5B0F8	α	I	1
86D3EFBC2	β	II	2
EB51DA3C6	γ	II	3
15487E7B5	δ	III	1
8ABE19436	α	I	2

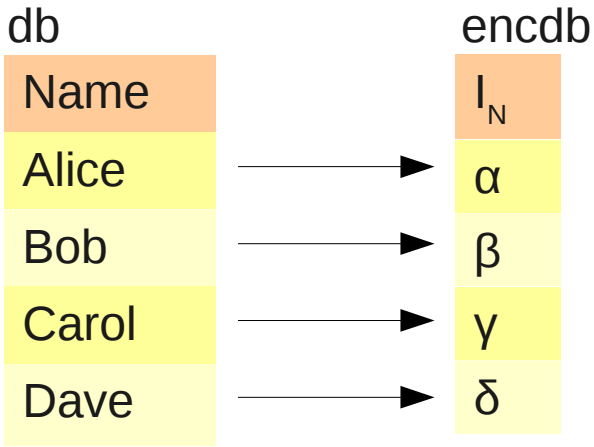
Art der Kodierung

- 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



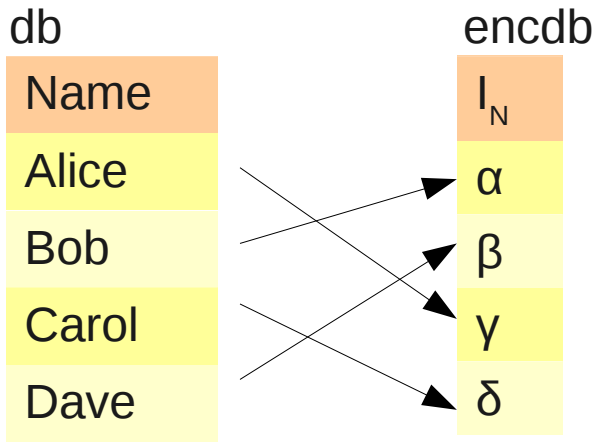
Anm.: Kodierung ist unabhängig davon, ob 1:1 oder M:N-Mapping



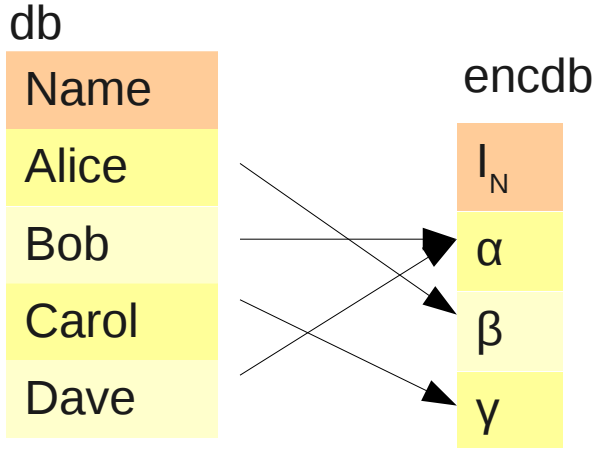


Mapping der Attributwerte

- 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

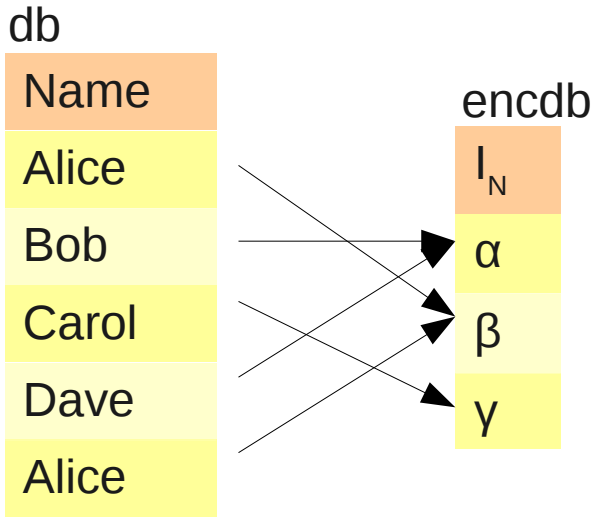


- Hash-Funktion über Attributwerte (mehrere Attributwerte entsprechen dem selben Indexeintrag)
  - Konzept äquivalent zu Hash-Index
  - N:M-Mapping von Attributwerten auf Index ($N \geq M$, „**Bucketization**“)
  - Performanz, Frequenz- und Zuordnungsangriffe hängen stark von dem Mapping ab
    → **False 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 mit Hintergrundwissen?
  - nächste Folie

Angreifbarkeit Bucketization

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 es nur eine Lösung gibt, ist Schema untauglich
  - Lösung für 2.): Alice muss α oder β sein; die 3 anderen permutieren: 6 Lösungsvarianten

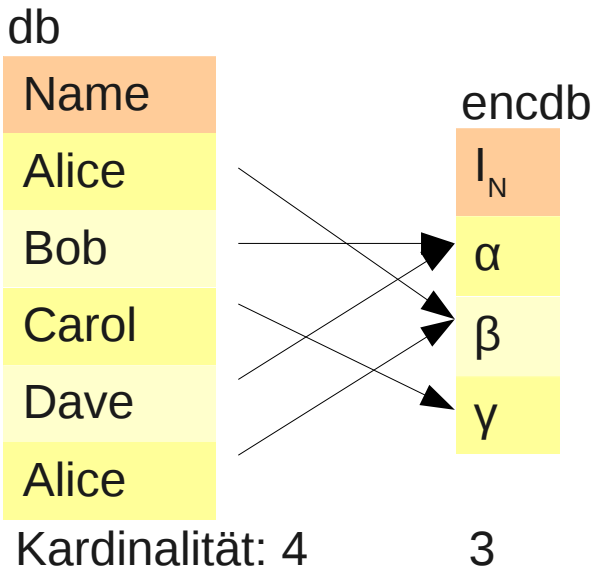
Name
Alice
Bob
Carol
Dave
Alice

?

I_N
2α
2β
1γ

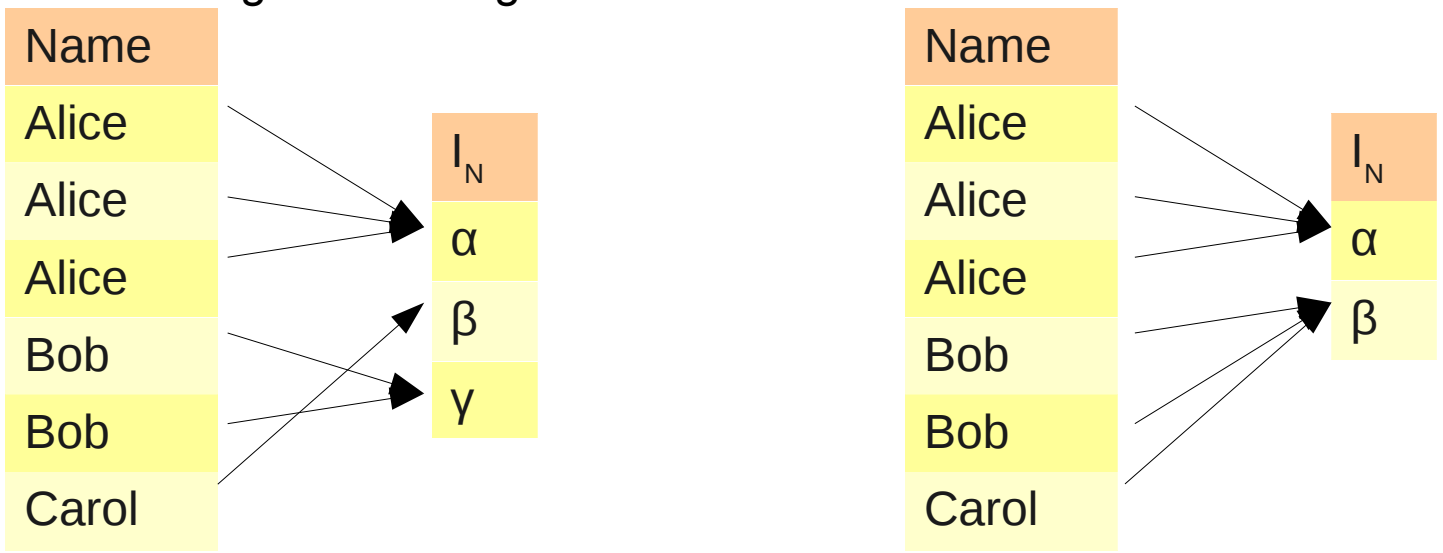
Query Performance

- 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

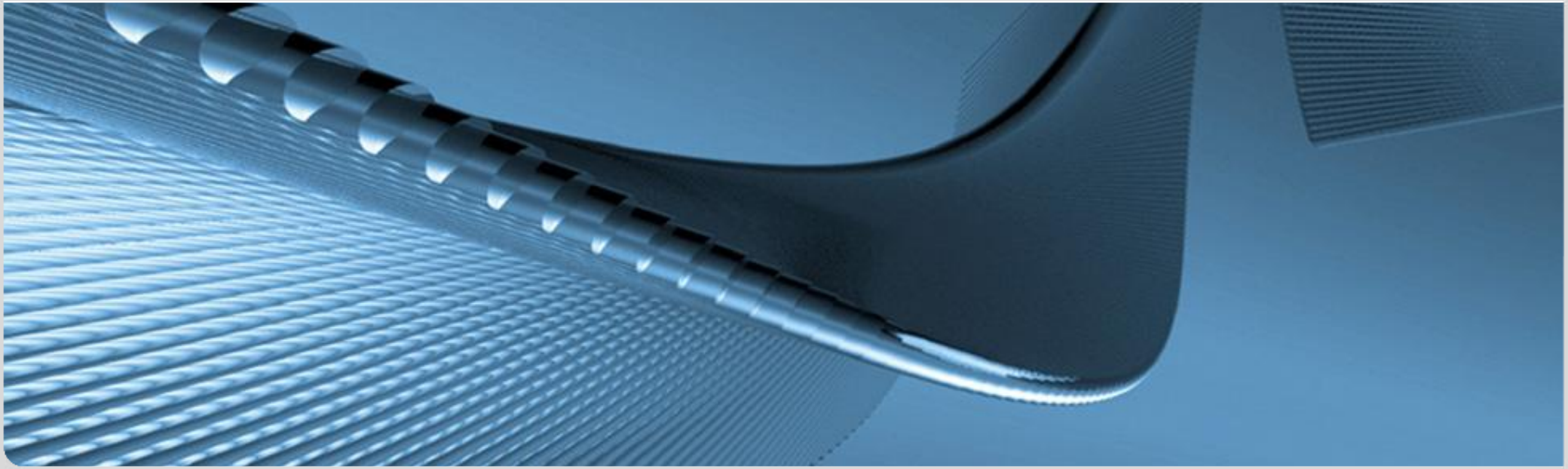
- Vorteile
  - Kompromiss zwischen Performanz und Privatheit steuerbar
  - Einfach einzusetzen, keine großen Änderungen an existierenden DBMS
- Nachteile
  - Je nach Verteilung der Attributwerte (z.B. Power-Law)
    - Angreifbar durch einzigartige Verteilung der Indexwerte (1:1-Mapping) oder ordnungserhaltende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



Karlsruher Institut für Technologie

Mehrdimensionale Bucketization

IPD, Systeme der Informationsverwaltung, Nachwuchsgruppe „Privacy Awareness in Information Systems“



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

Schutzziel

- Angreifermodell
  - Angreifer ist **Honest-but-Curious**
  - Angreifer hat Zugang zur Datenbank (Outsourcing-Szenario)

- Angreifer
  - Darf einzelne Werte in DB einfügen und Verschlüsselung beobachten (Chosen-Plaintext-Angriff)
  - Will Informationen über die gespeicherten Daten bekommen
    - Ungefähre Verteilung der Werte
    - Näherungswerte für bekannte Individuen

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

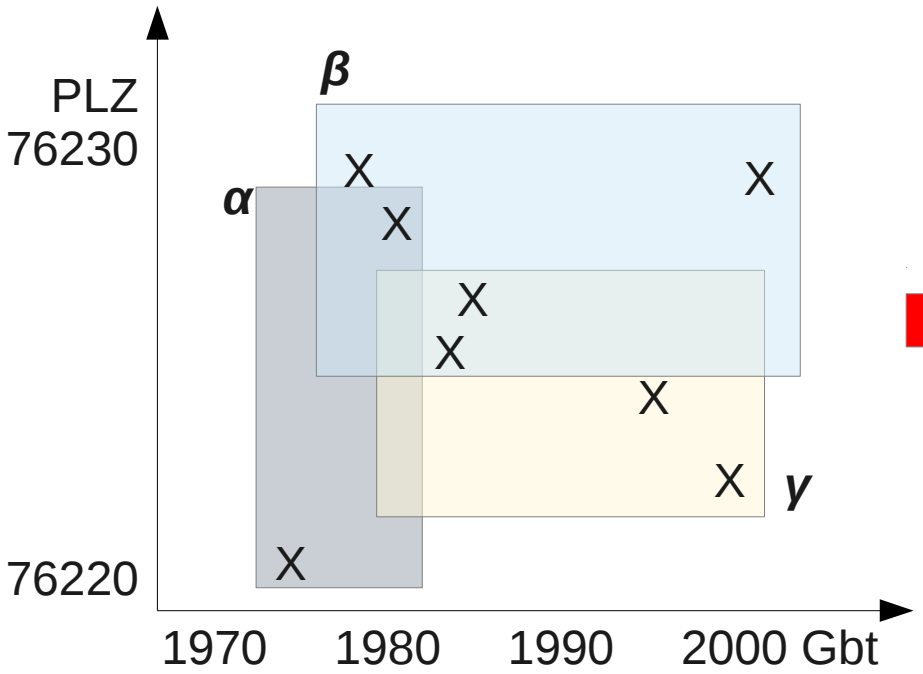
Multidimensionale Bucketization

- Client speichert
  - Schlüssel
  - Zuordnungsregel

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, ...}

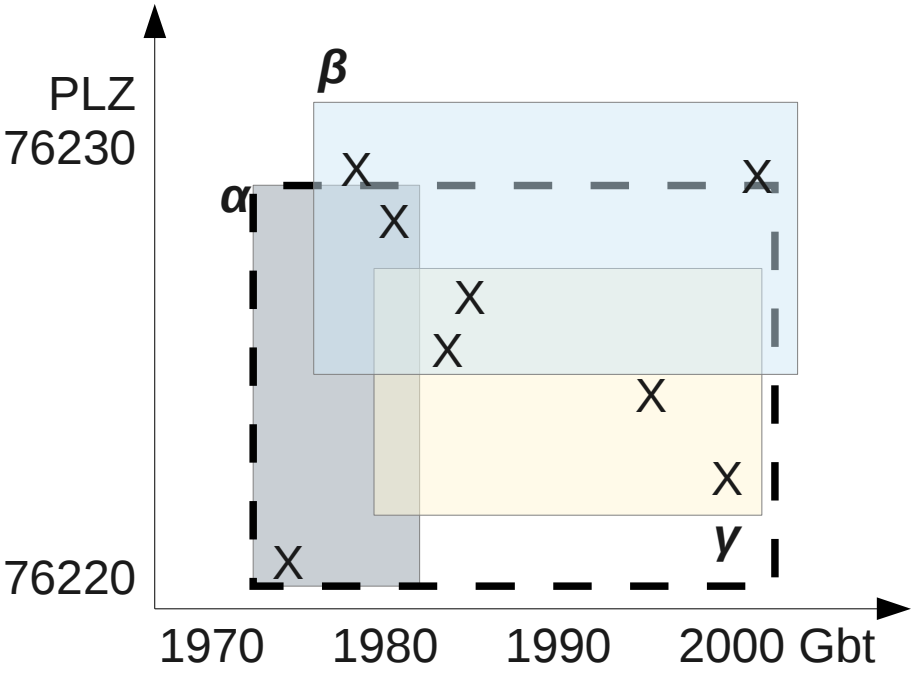
enc	B
770E5B0F8	β
86D3EFBC2	α
EB51DA3C6	γ
15487E7B5	β
8ABE19436	γ
C2AB7DA00	α
2814F82C4	β
9B7DFAFA0	γ

- Anfrageergebnis berechnen
  - false Positives herausfiltern

Eve	1985	76227	Husten
George	1983	76226	Heiser

Enthüllungsrisiko

- 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
9B7DFAFA0	**α**

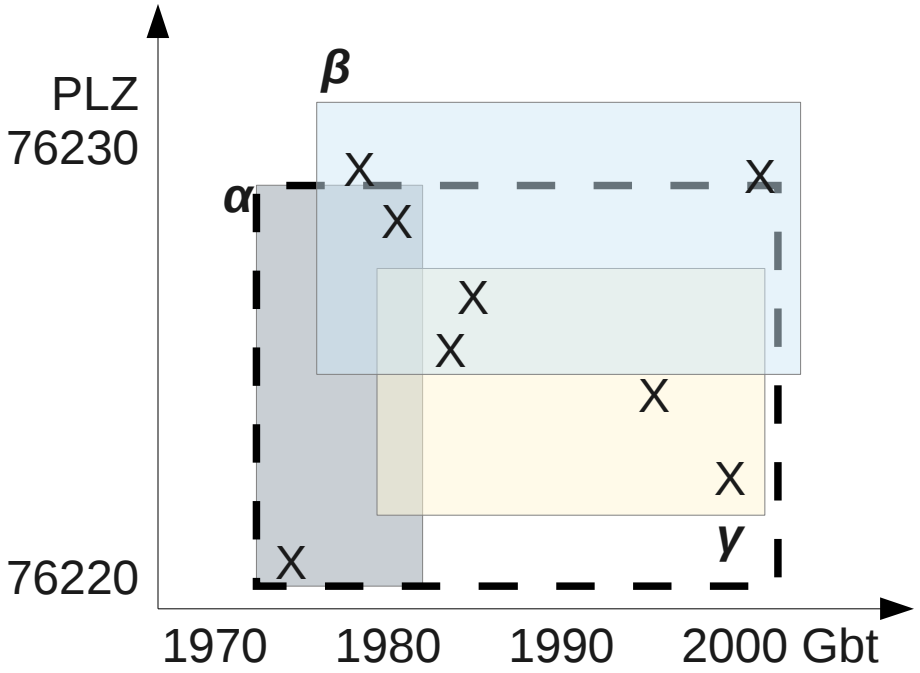
+γ

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

Enthüllungsrisiko, 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 1: Ziel ist das Maximieren der Entropie eines Buckets!*
  - *Anmerkung 2: Maß äquivalent zur l-Diversity!*



Optimale Verteilung der Buckets

- Extrem 1: Alle Werte in einem Bucket
  - Hohes Maß an Privatheit, keinerlei Rückschlüsse auf Attributverteilung
  - Maximale Anzahl von False Positives, schlechtestmögliche Performanz
- Extrem 2: Alle Werte in einem eigenen 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

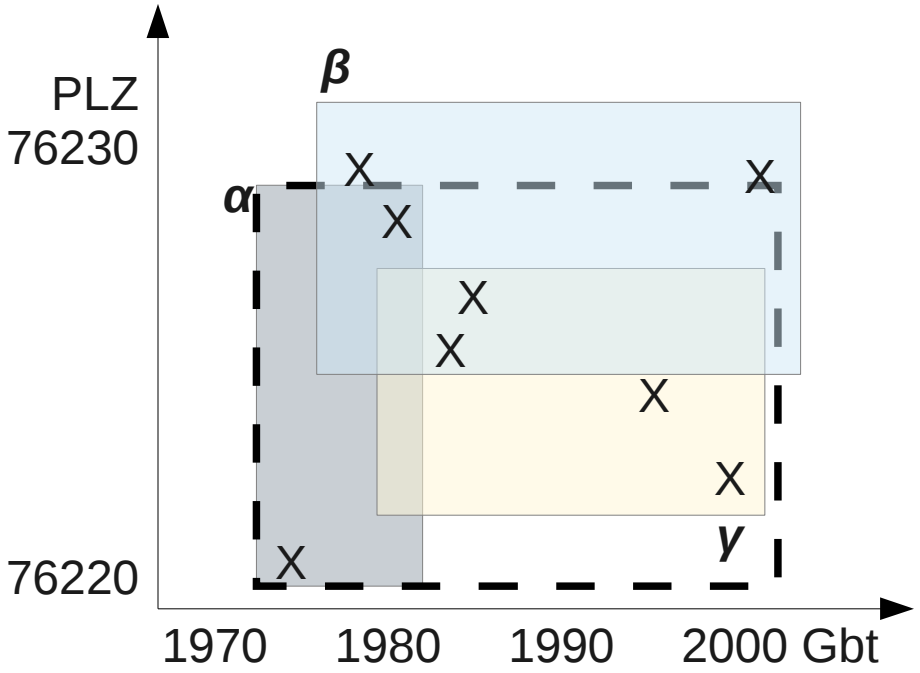
- Algorithmus in 2 Schritten:

1. Query-Optimal Bucketization
starte mit einem Bucket mit allen Datensätzen in db
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)

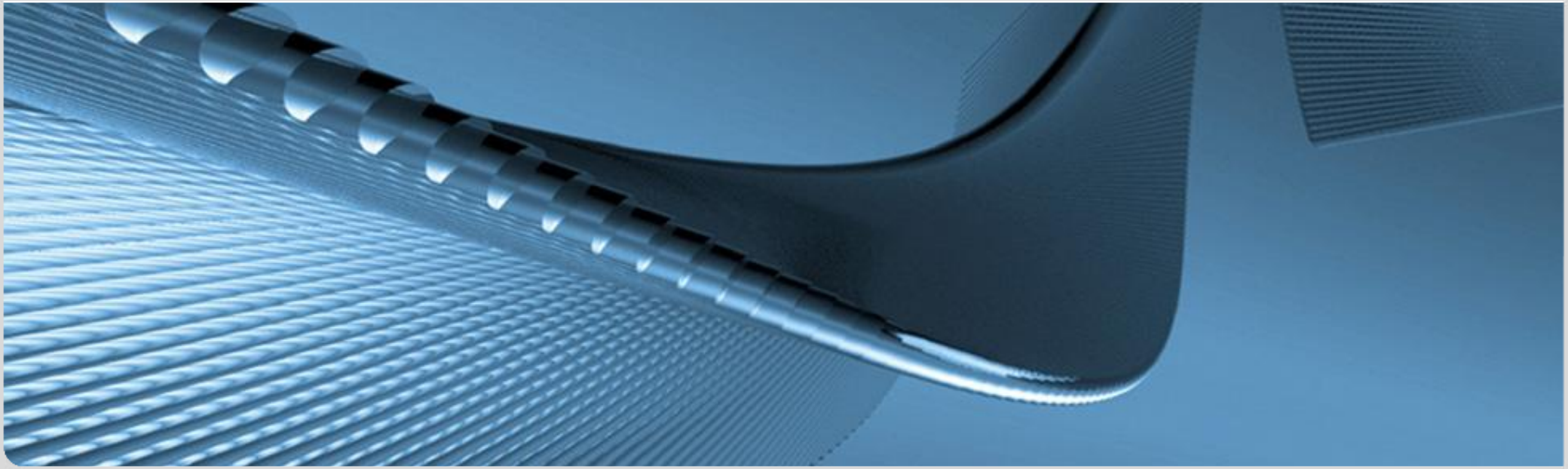
- Nachteile
  - Nur Greedy-Lösung für Bucket-Einteilung
    - Ggf. suboptimale Lösung
    - Abhängig von den Start-parametern
  - Nur performant bei mehrdimensionalen Bereichsanfragen
    - Bei anderen Anfragen viele false positives



Karlsruher Institut für Technologie

Fragmentation

IPD, Systeme der Informationsverwaltung, Nachwuchsgruppe „Privacy Awareness in Information Systems“



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, …, A_n}, Constraints C = {c_0, c_1, …, c_k}
- Jedes Fragment speichert
  - **salt**
    - beliebige Zufallszahl im Klartext, Primärschlüssel in R
  - **Sichtbare Attribute**
    - Nicht-sensible Attribute
      {A_i, …, A_j} ⊆ R
  - **enc**
    - Verschlüsselung aller restlichen Attribute eines Tupels XOR salt
      enc = encode([R – {A_i, …, A_j}] ⊗ 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, …, A_n},
  Constraint c ⊆ R
  - |c| = 1 (nur 1 Attribut): **Attribut** nicht veröffentlichen (c_0)
  - |c| > 1: **Assoziation** zwischen den Attributen nicht veröffentlichen (c_1...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, …, 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

- geg.: Relation $R = \{A_0, A_1, \ldots, A_n\}$,
  Constraints $C = \{c_0, c_1, \ldots, c_k\}$

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}

- 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

- geg.: Relation R = {A_0, A_1, …, A_n},
  Fragmente F = {F_0, F_1, …, F_k}

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}

- 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 ganze 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

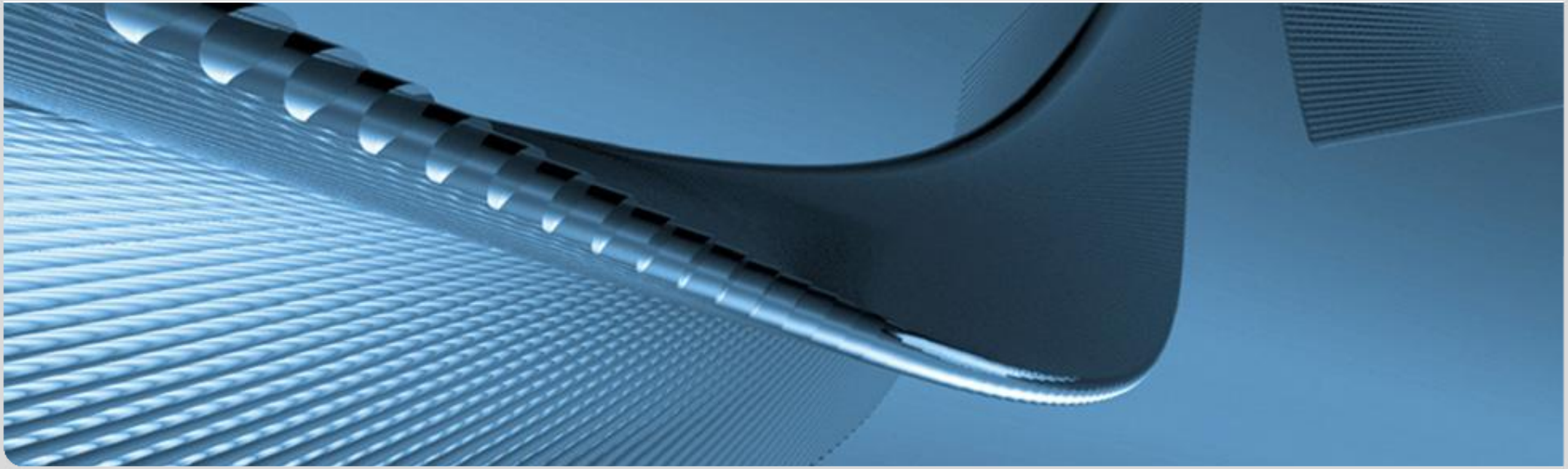
F_3

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

Karlsruher Institut für Technologie

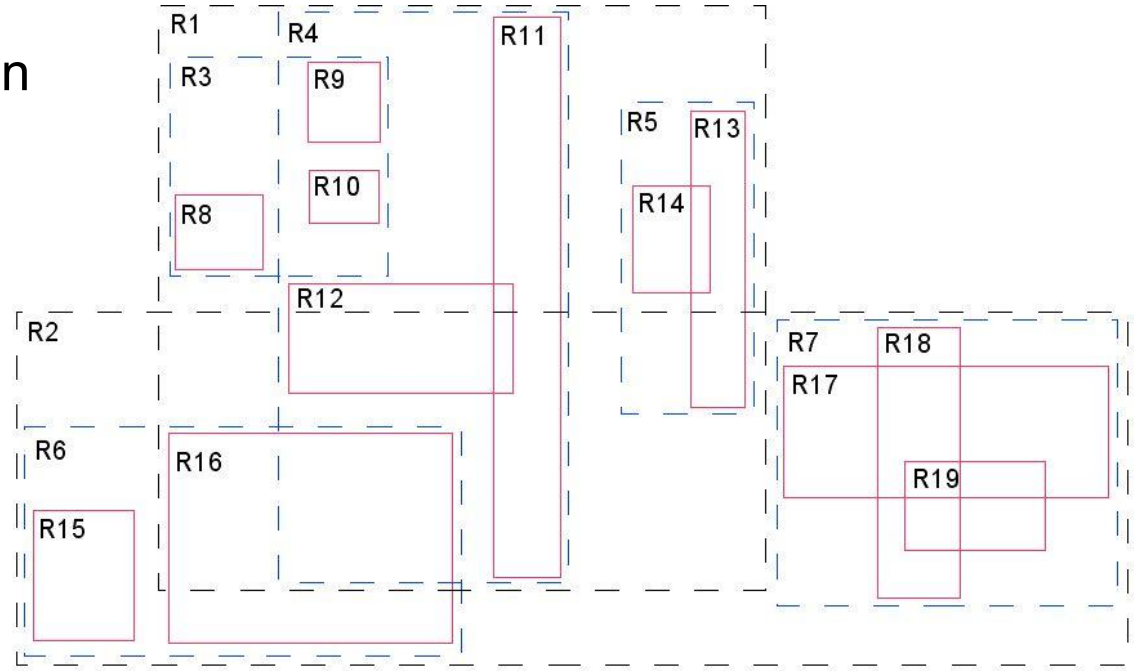
Homomorpe Verschlüsselung

IPD, Systeme der Informationsverwaltung, Nachwuchsgruppe „Privacy Awareness in Information Systems“



Szenario

- komplexe Rechenoperationen beim Dienstleister/in der Cloud
  - Indexieren von räumlichen Daten (Vergleichs-operationen im R-Baum)
  - Clusteranalyse von Kundendaten (Abstandsmaße auf Vektoren)
  - etc.



$$dist^2(x, y) = \sum_{i=1}^{d} (x_i - y_i)^2$$

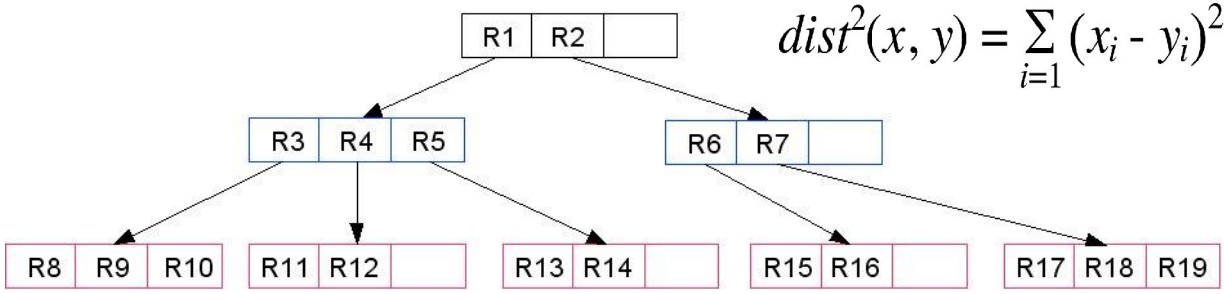


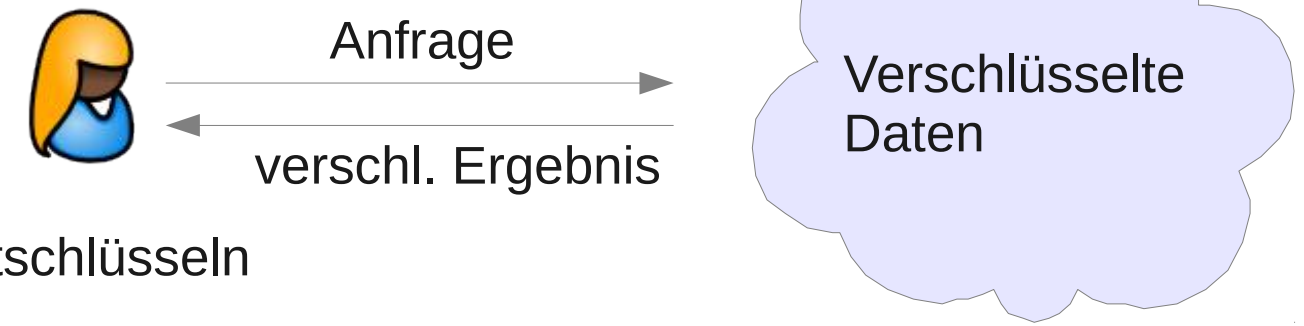
Bild: Wikimedia

Privacy Homomorphismus

- Daten so verschlüsseln, dass
  - Rechenoperationen **ohne Kenntnis des Schlüssels** auf verschlüsselten Daten ausgeführt werden
  - die Cloud verschlüsselte Ergebnisse an den Client übermittelt
  - der Client nach Entschlüsselung das Rechenergebnis erhält

- verschlüsselt mögliche Operationen (alle anderen Operationen muss Client berechnen)
  - Summe, Subtraktion
  - Multiplikation, Division

$$dist^2(x, y) = \sum_{i=1}^{d} (x_i - y_i)^2$$



Homomorphismus

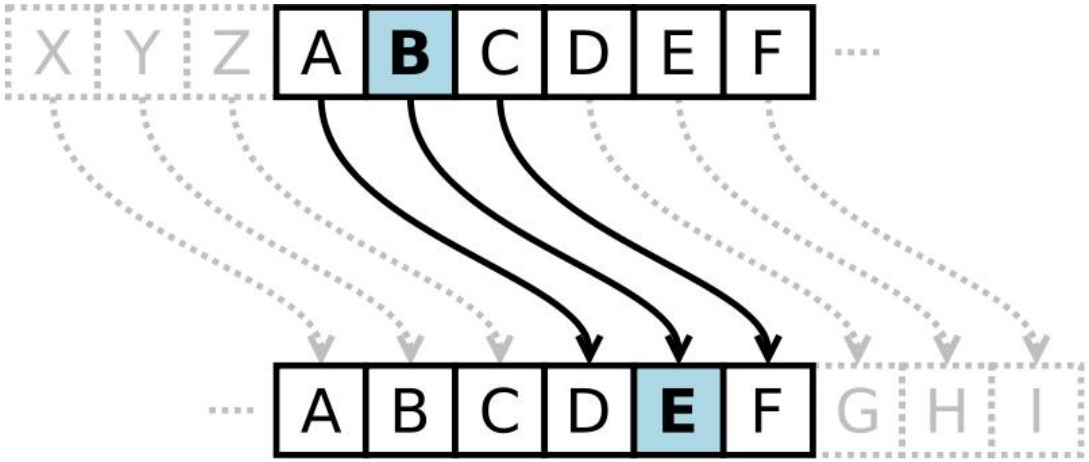
- Definition
  - zwei Verknüpfungen *p*, *q*,
    Abbildungsvorschrift *f,*
    Attribute $a_1 ... a_i$
  - *f* ist homomorph, wenn $f(p(a_1, a_2, \ldots a_i)) = q(f(a_1), f(a_2), \ldots f(a_i))$

- Beispiel: Addition
  - *f, q* seien Additionsoperationen,
    p ist eine Multiplikation mit 1,
    zwei Attribute a_1, a_2
  - *f* ist homomorph:
    $f(a_1 + a_2) = f(a_1) + f(a_2)$

→ Zentrale Frage: Welche Verschlüsselung ist für p, q geeignet und welche Abbildungsvorschrift unterstützt sie?

Beispiel: Caesar Chiffre

- Einfacher Ersetzungschiffre



- Caesar Chiffre ist **partiell homomorph**: unterstützt Konkatenation
  - Beispiel:

rot13(„Hello“)	= „Uryyb“
rot13(„World“)	= „Jbeyq“
Konkatenation(„Uryyb“, „Jbeyq“)	= „UryybJbeyq“
rot13(„UryybJbeyq“)	= „HelloWorld“

Bild: Wikimedia

Beispiel: RSA-Chiffre

- asymmetrische Public-Key-Verschlüsselung
  - Prinzip: Multiplikation von Primzahlen 'billiger' als Primzahlzerlegung

Klartext	k	öffentlicher Schlüssel	p*q, e
Primzahlen	p, q	privater Schlüssel	p*q, d
Ciphertext	$c := k^e$ MOD p*q	Entschlüsselung	$k := c^d$ MOD p*q

- RSA ist **partiell homomorph**: unterstützt Multiplikation
  Beispiel: verschlüsselte Berechnung von 7 * 2
  - Verschlüssele 7 und 2
    gewählt: p = 11, q = 13, berechnet: e = 23, d = 47
    enc(7) = (7^23) MOD (11*13) = 2
    enc(2) = (2^23) MOD (11*13) = 85
  - Multipliziere verschlüsselte Werte
    2 * 85 = 170
  - Entschlüssele Ergebnis
    dec(170) = (170^47) MOD (11*13) = 14

Hilfsvariablen:
φ(p,q) = (p-1)*(q-1)
e = zu φ(p,q) teilerfremde Zahl
d * e ≡ 1 MOD φ(p,q)

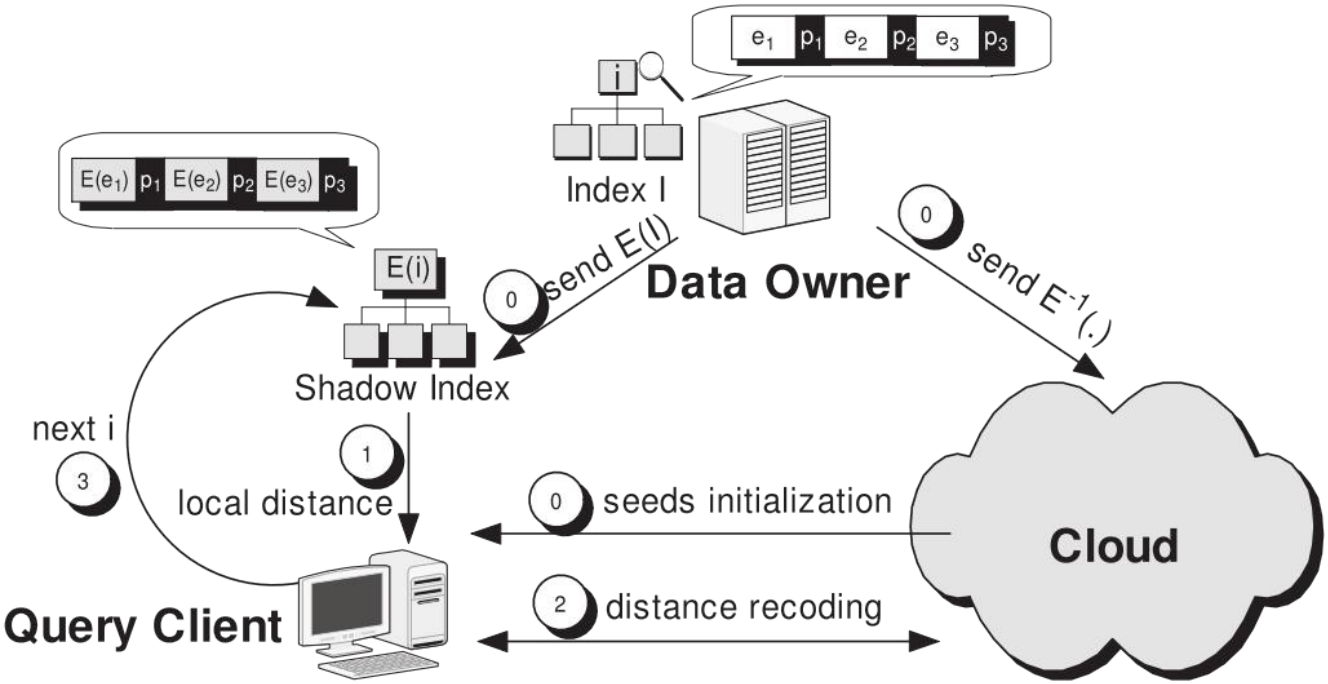
Volle Homomorphie nach [5]

- Verschlüsselung
  - Klartext k, geheime Primzahlen p, q
    Ciphertext (c,d) = enc(k) = [k MOD p, k MOD q]
- Operationen
  - Addition MOD p*q, Subtraktion MOD p*q, Multiplikation MOD p*q
  - alle anderen Operationen daraus herleitbar
  - auf dem Ciphertext (c,d) komponentenweise möglich
- Entschlüsselung
  - k = dec(c,d) durch in polynomialer Zeit berechenbares Gleichungssystem

- *praktisches Problem*
  - Entschlüsselung sehr teure Operation,
    polynomial bezüglich Schlüssellänge

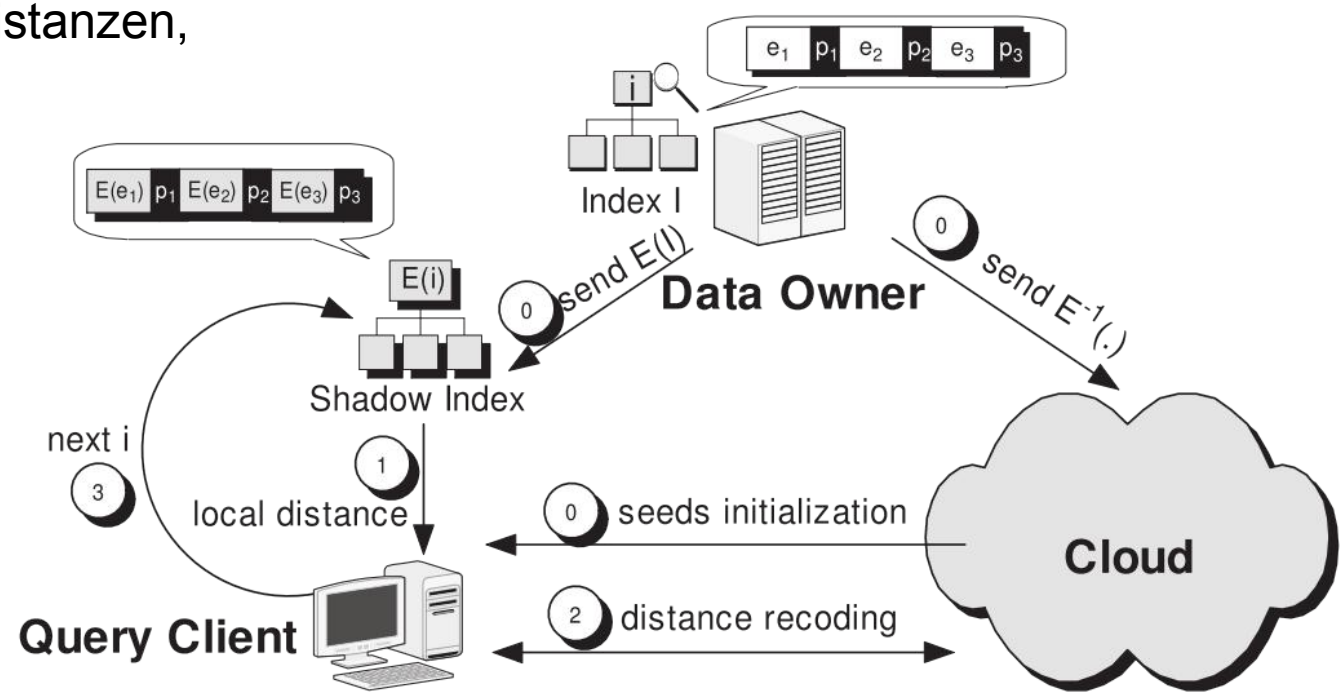
Beispiel: Private Distanz-Anfragen (1/3)

- Data Owner
  - speichert alle Schlüssel
  - speichert unverschlüsselten Index (R-Baum)
- Client
  - erhält vom Data Owner verschlüsselte Teile vom R-Baum (Shadow Index): Zeiger unverschlüsselt, Werte verschlüsselt
- Cloud
  - übernimmt Ver- und Entschlüsseln als teuerste Operation
  - *(könnte auch den Shadow Index senden)*



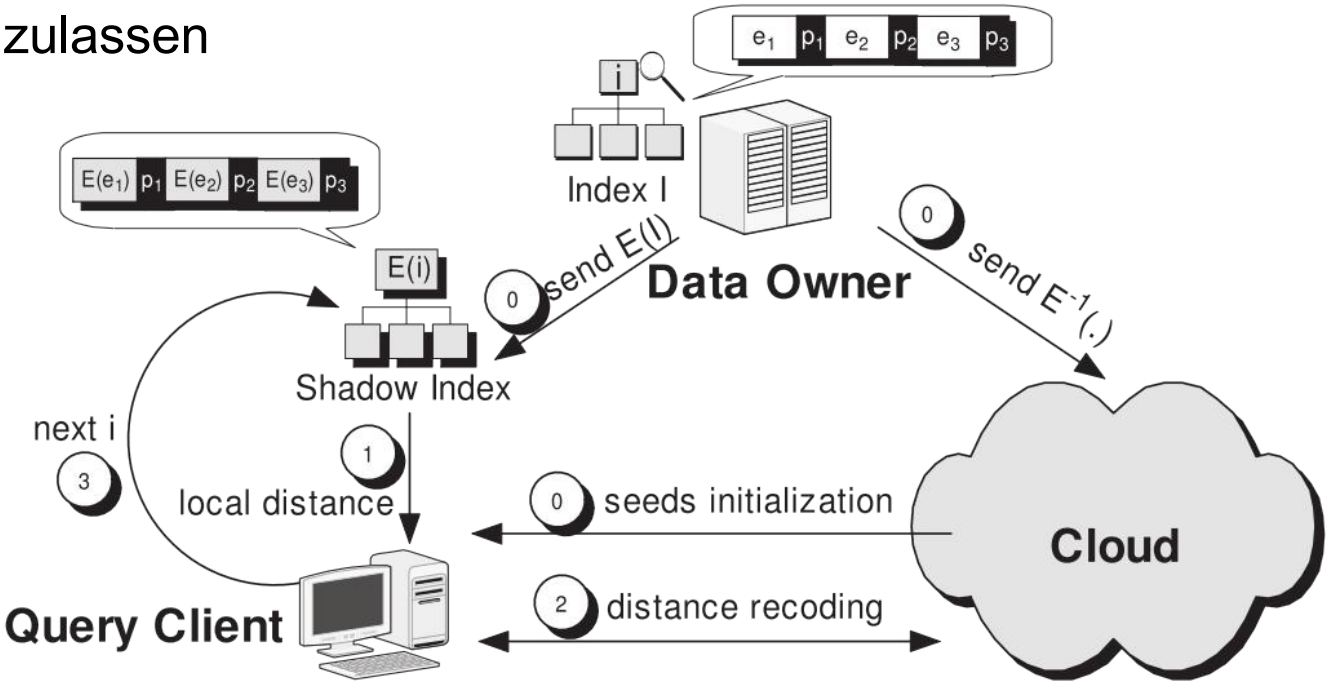
Beispiel: Private Distanz-Anfragen (2/3)

- Client erhält verschlüsselten Ausschnitt aus R-Baum (0)
- Traversierung des R-Baumes:
  - Client berechnet Distanzen zwischen eigenem Suchobjekt und den allen Index-Werten im aktuellen Knoten im Baum (1)
    - Berechnung auf verschlüsselten Werten
  - Client sendet verschlüsselte, permutierte Distanzen an Cloud
  - Cloud entschlüsselt Distanzen, rekodiert sie und sendet sie an Client (2)
  - Client wählt nächsten Knoten im Baum (3)
  - weiter mit (1)



Beispiel: Private Distanz-Anfragen (3/3)

- Data Owner
  - erfährt, wer Distanzanfragen im Baum durchführt
- Client
  - erfährt das Anfrageergebnis
  - Berechnung von Distanzen erfolgt verschlüsselt, Cloud teilt Client verfälschte Werte mit, die nur Auswahl von Knoten zulassen
- Cloud
  - erfährt Distanzen, aber nicht zwischen welchen Objekten diese berechnet werden



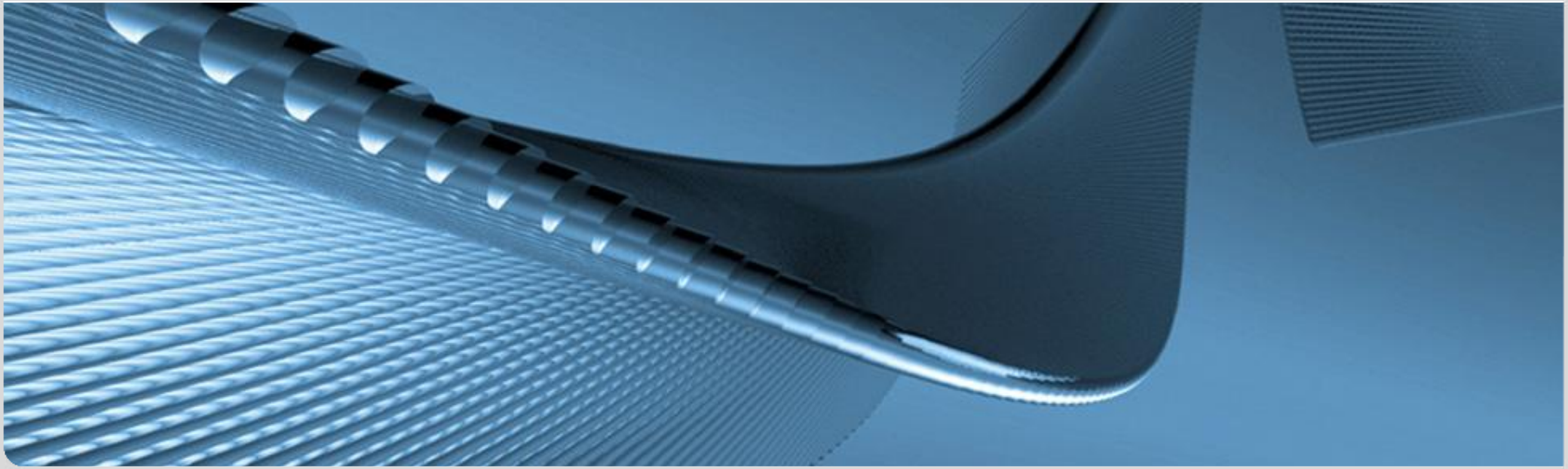
Diskussion

- Homomorphe Verschlüsselung ist berechnungsvollständig
  - im Prinzip alle arithmetischen Operationen durchführbar ohne Kenntnis der Originalwerte
  - komplexe Anwendungen sicher realisierbar
    - Beispiele: räumliche Indexierung, kNN-Suche, Clustering etc
- Aber: sehr teure Operation
  - einfache Additionen oder Multiplikationen bekommen polynomialen Aufwand
  - Ver- und Entschlüsselung in der Cloud (s. Beispiel) nur für wenige Szenarien geeignet

Karlsruher Institut für Technologie

Abschluss

IPD, Systeme der Informationsverwaltung, Nachwuchsgruppe „Privacy Awareness in Information Systems“



Zusammenfassung

- Outsourcen von Datenbanken ist ein aktuelles Thema

- Datenschutzverfahren unterscheiden sich deutlich hinsichtlich
  - Performanz bei der Verarbeitung von unterschiedlichen Anfragen
    - manche Anfragen: komplette Entschlüsselung der Datenbank beim Client
  - Unterstützte Datenschutzziele

- In dieser Vorlesung
  - Partitionierungsverfahren
    - flexibler Kompromiss aus Performanz (false positives) und Datenschutz
  - Bucket-Verfahren
    - schützt nicht gegen Ausspähen der Datenverteilung
    - sehr performant bei Anfragen auf sichtbare Attribute
  - Homomorphe Verschlüsselung
    - sehr sicher, aber warten auf performante Lösungen aus der Kryptographie

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

[5] Haibo Hu, Jianliang Xu, Chushi Ren, Byron Choi: *Processing Private Queries Over Untrusted Data Cloud Through Privacy Homomorphism.* ICDE 2011