Building Privacy-Aware Database Systems

CS848 Winter 2021
Module 3
This course will explore …

- How to define a good privacy promise?
- How to design a privacy-preserving algorithms?
- How to build a privacy-aware database systems?

Greatly depend on the architecture setup and trust assumptions

Client-server with trusted data curator

Data federation

Cloud service provider
Cloud Setting

• But data leaks can happen
  – Hackers with illegal access
  – Snooping administrators

• Encrypted databases and evaluation metrics

![Diagram showing cost, functionality, and security axes with data types and leakage metrics.]
Lecture Focus

• Functionality: SQL-aware
• Passive adversary
  – Honest but curious (HBC)
  – Does not alter database or results
• Desired functionality under encryption
  – Encrypted data stored, encrypted queries, and encrypted results
  – But that does not guarantee security

Lecture slides are mainly adapted from: “Querying Encrypted Data”, Arasu et al. ICDE 2016
Outline

• Part I: Design Choices

• Part II: Security Guarantees

• Upcoming Papers and Announcements
Outline

• Part I: Design Choices
  – Encryption basics
  – Trusted client-based systems
  – Secure in-cloud computing

• Part II: Security Guarantees

• Upcoming Papers and Announcements
Encryption Scheme

Key: 000102030405060708090a0b0c0d0e0f

Plaintext

The quick brown fox jumps over the lazy dog

Encr

Ciphertext

a7be1a6997ad739bd8c9ca451f618b61
b6ff744ed2c2c9bf6c590cbf0469bf41
47f7f7bc95353e03f96c32bcfd8058df

Decr

Plaintext

The quick brown fox jumps over the lazy dog

Key: 000102030405060708090a0b0c0d0e0f

Crypto Textbook: [KL 07]
Encryption Scheme

Public Key: 000102030405060708090a0b0c0d0e0f

Private Key: 47b6ffedc2be19bd5359c32bcfd8dff5

Plaintext: The quick brown fox jumps over the lazy dog

Ciphertext: a7be1a6997ad739bd8c9ca451f618b61b6ff744ed2c2c9bf6c590cbf0469bf4147f7f7bc95353e03f96c32bcfd8058df

 decrypted to: The quick brown fox jumps over the lazy dog

Crypto Textbook: [KL 07]
AES + CBC Mode

The quick brown fox jumps over the lazy dog........

Variable IV => Non-deterministic

[AES, KL 07]
AES + CBC Mode

The quick brown fox jumps over the lazy dog.

Variable IV => Non-deterministic

[AES, KL 07]
Nondeterministic Encryption Scheme

The quick brown fox jumps over the lazy dog

Key: 000102030405060708090a0b0c0d0e0f

Example: AES + CBC + variable IV
AES + ECB Mode

The quick brown

fox jumps over t

lazy dog

AES

Key

a7be1a6997a7...

b6ff744ed2c2...

47ff7ffbc9535...

[AES, KL 07]
Deterministic Encryption Scheme

Key: 000102030405060708090a0b0c0d0e0f

The quick brown fox jumps over the lazy dog

Encr

The quick brown fox jumps over the lazy dog

Encr

Example: AES + ECB

More secure deterministic encryption: [PRZ+11]
Strong Security => Non-Deterministic

Deterministic Encryption

\[ \sigma_{\text{StudentId}=1} \]

```
select * 
from assignment 
where studentid = 1
```
## Deterministic Encryption

The diagram illustrates a query that selects specific columns from the `assignment` table where the `studentid_det` matches a given value. The query is:

```
select * 
from assignment 
where studentid_det = 'bd6e7c3df2b5779e0b61216e8b10b689'
```

Here is a table showing the data extracted from the `assignment` table:

<table>
<thead>
<tr>
<th>StudentId_DET</th>
<th>AssignId</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>bd6e7c3df2b5779e0b61216e8b10b689</td>
<td>1</td>
<td>68</td>
</tr>
<tr>
<td>bd6e7c3df2b5779e0b61216e8b10b689</td>
<td>2</td>
<td>71</td>
</tr>
<tr>
<td>7ad5fda789ef4e272bca100b3d9ff59f</td>
<td>4</td>
<td>99</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Homomorphic Encryption

Encryption key is not an input
Order Preserving Encryption

<table>
<thead>
<tr>
<th>Value</th>
<th>Enc (Value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0001102789d5f50b2befffd9f3dca4ea7</td>
</tr>
<tr>
<td>2</td>
<td>0x0065fda789ef4e272bcf102787a93903</td>
</tr>
<tr>
<td>3</td>
<td>0x009b5708e13665a7de14d3d824ca9f15</td>
</tr>
<tr>
<td>4</td>
<td>0x04e062ff507458f9be50497656ed654c</td>
</tr>
<tr>
<td>5</td>
<td>0x08db34fb1f807678d3f833c2194a759e</td>
</tr>
</tbody>
</table>

\[ x < y \rightarrow \text{Enc}(x) < \text{Enc}(y) \]

[BCN11, PLZ13]
**Order-Preserving Encryption**

\[ \sigma_{Score \geq 90} \]

```sql
select *
from assignment
where score >= 90
```

<table>
<thead>
<tr>
<th>StudentId</th>
<th>AssignId</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>68</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>71</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>99</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Order-Preserving Encryption

$$\sigma_{Score\_o.pe \geq 04e0...}$$

<table>
<thead>
<tr>
<th>StudentId</th>
<th>AssignId</th>
<th>Score_OPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0x0065fda789ef4e272bcf102787a93903</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0x009b5708e13665a7de14d3d824ca9f15</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0x08db34fb1f807678d3f833c2194a759e</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Homomorphic Encryption Schemes

- Fully Homomorphic Encryption
  - Paillier Cryptosystem \([P99]\)
  - ElGamal Cryptosystem \([E84]\)

- Order-Preserving Encryption
  - \([BCN11, PLZ13]\)

- Deterministic Encryption

- Non-Deterministic Encryption

\((\phi)\)
Homomorphic Encryption Schemes

- Fully Homomorphic Encryption (Any function) [G09, G10]
- Order-Preserving Encryption (≤) [BCN11, PLZ13]
- Deterministic Encryption (==)
- Non-Deterministic Encryption (∅)

Partial Homomorphic Encryption

- Paillier Cryptosystem (+) [P99]
- ElGamal Cryptosystem (×) [E84]
Homomorphic Encryption Schemes

- **Impractical**
  - Fully Homomorphic Encryption
    - (Any function)
    - [G09, G10]
  - Partial Homomorphic Encryption
    - Paillier Cryptosystem [P99]
    - ElGamal Cryptosystem [E84]

- **Expensive**
  - Order-Preserving Encryption
    - \((\leq)\)
    - [BCN11, PLZ13]
  - Deterministic Encryption
    - \((==)\)

- **Practical**
  - Non-Deterministic Encryption
    - \((\emptyset)\)
# Homomorphic Encryption Schemes: Performance

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Space for 1 integer (bits)</th>
<th>Time for 1 operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Homomorphic Encryption</td>
<td>$2^{14}$</td>
<td>Cosmic time scales</td>
</tr>
<tr>
<td>Paillier ElGamal</td>
<td>2048</td>
<td>$\approx$ ms</td>
</tr>
<tr>
<td>Deterministic Order-preserving</td>
<td>128</td>
<td>$\approx$ $\mu$s</td>
</tr>
</tbody>
</table>
Design Choices

- **F.H.E**
- **P.H.E**

- **USE SECURE LOCATION**
  - Client
  - Server

- **COMPUTE ON ENCRYPTED DATA**
Trusted Client based Systems

- F.H.E
- P.H.E
- Client
- Server

COMPUTE ON ENCRYPTED DATA

USE SECURE LOCATION
Outline

• Part I: Design Choices
  – Encryption basics
  – Trusted client-based systems
  – Secure in-cloud computing

• Part II: Security Guarantees
  – Security for data at rest
  – Dynamic security

• Upcoming Papers and Announcements
Trusted Client Architecture

- Data not decrypted in DBMS
  - Only ciphertext seen in the DBMS
- No changes to DBMS/Client App
Trusted Client-based Systems

• Minimal Client Computation
  – Use P.H.E (CryptDB)

• Residual Query Processing in Client
  – Blob Store
  – Use in conjunction with P.H.E (Monomi)
CryptDB Architecture

- Web proxy rewrites queries, decrypts result
- Leverage P.H.E techniques

[PRZ+11]
Database Design

• Plaintext Schema
  – students(ID, grade)
    • Point Lookups on ID column
    • SELECT and AGGREGATION queries on grade

• Ciphertext Schema
  – students(ID_DET, grade_OPE, grade_PAILLIER)
    • Need to store columns encrypted in multiple ways
    • Static/Dynamic design based on workload

[PRZ+11]
Query Processing

Client App

select COUNT(*) from students
where grade > 3.5

Web Proxy

SUM(grade), COUNT(*)
decrypt(PAILLIER_SUM(grade_PAILLIER))

Key

DBMS + UDFs

students(ID_DET, grade_OPE, grade_PAILLIER)

select PAILLIER_SUM(grade_PAILLIER), COUNT(*)
from students

[PRZ+11]
Onions of Encryptions

• Same key for all items in a column for same onion layer
• Start out DB with the most secure encryption scheme
Dynamic Database Design

Client App

Web Proxy

Key

\[ \text{select COUNT(*) from students } \]
\[ \text{where grade > 3.5} \]

students(ID_DET, grade**)

DBMS + UDFs

DET

OPE

ID

grade OPE

grade

[PRZ+11]
Limitations

• P.H.E is not “free” – space overheads
  – For Paillier, to store one integer (32 bits), the ciphertext need to use 2048 bits!
  – Compact representation for paillier that is updatable is an open problem.

• P.H.E is inherently limited – cannot address all of SQL

```sql
select STDEV(grade) from students
```
Pre-computation for complex queries

• Find student submissions that have been handled in a day late

```sql
select id, count(*) from students
where submissiondate = deadline + 1
group by id
```

• Students(ID_DET, submissiondate_DET, deadline_DET, deadline_PAILLIER_DET)
  – Cannot “Mix and Match” different encryptions

```sql
select id, count(*) from students
where submissiondate = deadline_plusone
group by id
```

[TFM13]
Trusted Client: Summary

- No server changes required to DBMS
- Works well for workloads where amount of data shipped is small
  - Physical design is important for distributed queries
  - Pre-computation is not free
- Generality of approach is unproven
  - Integrity constraints, Triggers etc.
  - Automated tools to migrate database applications
Outline

• Part I: Design Choices
  – Encryption basics
  – Trusted client-based systems
  – Secure in-cloud computing

• Part II: Security Guarantees
  – Security for data at rest
  – Dynamic security

• Upcoming Papers and Announcements
Secure In-Cloud Processing

- COMPUTE ON ENCRYPTED DATA
  - F.H.E
  - P.H.E
- USE TRUSTED MODULE
  - Client-End Solution
  - In-Cloud Solution

- Traditional Servers: Easy to setup but hard to secure
- Secure Hardware
Secure In-Cloud Processing

- Compute on Encrypted Data
  - F.H.E
  - P.H.E

- Use Trusted Module
  - Client-End Solution
  - In-Cloud Solution
    - IBM 4764 PCI-X Cryptographic Coprocessor
    - Secure FPGA

- Traditional Servers
- Secure Hardware

Limited Recourses!
Trusted Client Architecture

- Distributed query processing between untrusted DBMS and client-end DBMS shell
Secure In-Cloud Compute Architecture

- Distributed query processing between untrusted DBMS and trusted cloud compute
- Solutions differ in granularity of integration
Secure Processors

- TrustedDB
  - Trusted compute is a full DBMS
    - loosely coupled system
      - Simple division of labor
      - Simple to build
    - Inefficient use of resources

[BS11]
Dedicated Expression Evaluation

- Cipherbase
  - Trusted compute is only expression evaluation

[ABE+12, ABE+13]
Dedicated Expression Evaluation

• Advantages
  – Efficiency of trusted compute resources
  – Dedicated circuits ➔ virus-proof
  – Small footprint ➔ formal verification

• Drawbacks
  – Fundamentally changes expression evaluation ➔ non-trivial changes to host DBMS

[ABE+12, ABE+13]
Secure In-cloud: Summary

• Secure in-cloud trusted compute resources

• Open issues
  – Query optimization
    • e.g. Statistics on encrypted data, security-aware type matching
  – Execution engine
    • e.g. Data/computation reuse, masking latency to trusted computation
  – Physical Design
    • e.g. Leveraging stronger encryption
Outline

• Part I: Design Choices
  – Encryption Basics
  – Trusted Client based Systems
  – Secure In-Cloud Computing

• Part II: Encryption & Security
  – Security for data at rest
  – Dynamic security

• Upcoming Papers and Announcements
Encryption and Security

Key:

• Semantic security:
  – No information leakage except input length
• Winner Turing Award

[KL07]
Encryption and Security

Key:

Encryption schemes such as AES in CBC mode (non-deterministic) are believed to be semantically secure.
Security of Database Encryption

- Apply AES-CBC to every cell
-Leaks cell lengths
Deterministic Encryption

- Leaks cell lengths
- Also, no. of distinct values + frequency distribution [BFO+08]

<table>
<thead>
<tr>
<th>Disease</th>
<th>Disease_DET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flu</td>
<td>!x8J</td>
</tr>
<tr>
<td>Diabetes</td>
<td>)zFr#x</td>
</tr>
<tr>
<td>Flu</td>
<td>!x8J</td>
</tr>
<tr>
<td>Cold</td>
<td>BxU3</td>
</tr>
</tbody>
</table>
Order-Preserving Encryption

- Leaks cell lengths
- Also, order of cell values and more [AKS+04, BCN11, NKW15, Grubbs et al. S&P17]
Impact of Querying & Updating

Client

Update Employee
Set Salary = *\&@#
Where Name = 'Alice'

Server

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary_NDET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>X%*!</td>
</tr>
<tr>
<td>Bob</td>
<td>~4Yz</td>
</tr>
<tr>
<td>Chen</td>
<td>T$H2</td>
</tr>
<tr>
<td>Dan</td>
<td>&lt;$&gt;fB</td>
</tr>
</tbody>
</table>
Impact of Querying & Updating

**Trusted Client**

- **Client**
- **Key**

**Untrusted Server**

Update Employee
Set Salary = *!-#
Where Name = ‘Alice’

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary_NDET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>X%*!</td>
</tr>
<tr>
<td>Bob</td>
<td>~4Yz</td>
</tr>
<tr>
<td>Chen</td>
<td>T$H2</td>
</tr>
<tr>
<td>Dan</td>
<td>&lt;*fB</td>
</tr>
</tbody>
</table>
Impact of Querying & Updating

Update Employee
Set Salary = 23=${<
Where Name = ‘Bob’
Impact of Querying & Updating

Client

Server

Update Employee
Set Salary = +=$<
Where Name = ‘Bob’

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary_NDET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>X%*!</td>
</tr>
<tr>
<td>Bob</td>
<td>~4Yz</td>
</tr>
<tr>
<td>Chen</td>
<td>T$H2</td>
</tr>
<tr>
<td>Dan</td>
<td>&lt;*&gt;fB</td>
</tr>
</tbody>
</table>
Impact of Querying & Updating

**Trusted Client**

**Untrusted Server**

Update Employee
Set Salary = #2$^\wedge$
Where Name = ‘Bob’

- **Background knowledge**
  - Full-time employees earn more
  - Salaries of hourly-wage employees updated more
- **Learn partial ordering of employee salary**
  - Alice’s salary > Bob’s

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary_NDET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>X%!*</td>
</tr>
<tr>
<td>Bob</td>
<td>~4Yz</td>
</tr>
<tr>
<td>Chen</td>
<td>T$H2</td>
</tr>
<tr>
<td>Dan</td>
<td>&lt;*fB</td>
</tr>
</tbody>
</table>

Query access patterns reveal information! [OS07]
Impact of Querying & Updating

• Sort leaks ordering
• Encryption across the stack (disk + in-memory) does NOT imply no information leakage

The overall query workflow reveals information

*Dynamic security* (different from security of data at rest)
Design Space

<table>
<thead>
<tr>
<th>Operations on column</th>
<th>Leakage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality (including joins)</td>
<td>Frequency distribution</td>
</tr>
<tr>
<td>Indexing/Sorting/range predicates</td>
<td>Order</td>
</tr>
</tbody>
</table>

Can we bridge this gap?

Stop with encryption

Full Leakage

No Leakage Impractical

Cipherbase, TrustedDB, CryptDB, Monomi, BlobStore
Design Space

- Cipherbase, TrustedDB, CryptDB, Monomi, BlobStore

Stop with encryption

Full Leakage

Output Size, Running Time

No Leakage

Impractical
Oblivious Simulation

- **Simulation**: $P'$ equivalent to $P$
- **Theoretically Efficient**: Running time of $P'$ within polylog factor of running time of $P$
- **Oblivious**: Access patterns of $P'$ look random
- **Information leakage**: input size, output size, running time

[GO96, W12, SS13]
Application to DBMS

• Destroys spatial and temporal locality of reference

• Range scan of 100M records on hard disk → 100M seeks
  • $10^5$ seconds (~1 day)
Design Space

Full Leakage

Cipherbase, TrustedDB, CryptDB, Monomi, BlobStore

Stop with encryption

Output Size, Running Time

No Leakage

Is there a stronger and practically achievable security model?

e.g. Opaque Zheng et al. NSDI 2017
Opaque: Secure Distributed Analytics

• Obliviousness: hiding access control

• Two-part solution:
  – Distributed oblivious SQL operators:
    • Oblivious filter
    • Oblivious sort
    • Oblivious aggregation
    • Oblivious join
  – Query optimization
    • Rule-based optimization
    • Cost model
    • Cost-based optimization

[Zheng et al. NSDI 2017]
Oblivious Aggregation Example

- SELECT count(*) FROM medical GROUP BY disease

The “diabetes” group is split

How to aggregate obliviously and in parallel?
Oblivious Aggregation Example

- SELECT count(*) FROM medical GROUP BY disease
Oblivious Aggregation Example

- SELECT count(*) FROM medical GROUP BY disease

Aggregation has at least 2 oblivious sorts, Can Opaque do better?
Rule-based Optimization Example

- SELECT count(*) FROM medical WHERE age > 30 GROUP BY disease;

Opaque op.

Sort on “disease”

Sort on “age>30” column

A multi-column sort on “disease” and “age>30”
Summary

• Part I: Design Choices
  – Encryption basics
  – Trusted client-based systems
  – Secure in-cloud computing

• Part II: Security Guarantees
  – Security for data at rest
  – Dynamic security
Other Challenges

• Application Security
  – DBMS is only a part of the overall system stack

• Usability
  – Clients need tools and interpretable security models to navigate security-performance tradeoff

• Connections to other areas of security
  – Data privacy, access control, auditing
Discussion Time
Paper Readings

• Week 11
  – 3a. Seabed
  – 3b. OPE Leakage
  – 3c. Partitioned Data Security

• Week 12
  – 3d. StealthDB
  – 3e. EncDBDB
Announcement

• Assignment 3
  – Release after Wed session
  – Latex file will be available
  – Submit pdf on Learn, by Mar 29, 11pm

• Project midterm report
  – Submit pdf on Learn, by March 22, 11pm
Slides Credits

- Slides for “Querying Encrypted Data”, Arasu et al. ICDE 2016
- Slides for “Opaque”, Zheng et al. NSDI 2017