Query-private DB Query Processing Technique

김명선[†]

정보보호학과@수원대학교

2차워크샵@암호연구회, September 9, 2016

[†] Joint work with Hyung Tae Lee and other NTU & A^*STAR Guys

Contents

1. System model
2. Private query
3. Problem statement
4. Possible approaches
5. Idea sketch
6. The construction
7. Wrap-up

Contents

1. System model
2. Private query
3. Problem statement
4. Possible approaches
5. Idea sketch
6. The construction
7. Wrap-up

Contents

1. System model
2. Private query
3. Problem statement
4. Possible approaches
5. Idea sketch
6. The construction
7. Wrap-up

Contents

1. System model
2. Private query
3. Problem statement
4. Possible approaches
5. Idea sketch
6. The construction
7. Wrap-up

Contents

1. System model
2. Private query
3. Problem statement
4. Possible approaches
5. Idea sketch
6. The construction
7. Wrap-up

Contents

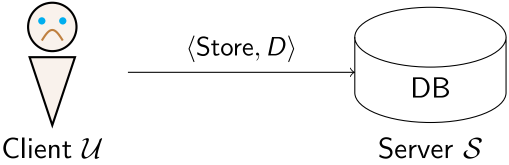
1. System model
2. Private query
3. Problem statement
4. Possible approaches
5. Idea sketch
6. The construction
7. Wrap-up

Contents

1. System model
2. Private query
3. Problem statement
4. Possible approaches
5. Idea sketch
6. The construction
7. Wrap-up

System model

- Symmetric **Outsourced** Database: Naïve model
  - ∗ Client $\mathcal{U}$: DB D를 Server $\mathcal{S}$에 저장
  - ∗ Privacy issue: S learns all tuples in D

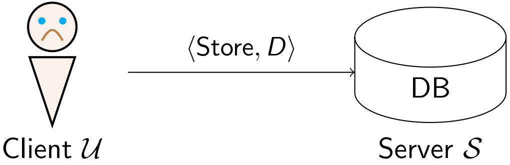


Database D

A_1	A_2	$\cdots$	A_n
a_{11}	a_{12}	$\cdots$	a_{1n}
$\vdots$	$\vdots$	$\cdots$	$\vdots$
a_{m1}	a_{m2}	$\cdots$	a_{mn}

System model

- Symmetric **Outsourced** Database: Naïve model
  - ∗ Client $\mathcal{U}$: DB D를 Server $\mathcal{S}$에 저장
  - ∗ Privacy issue: S learns all tuples in D

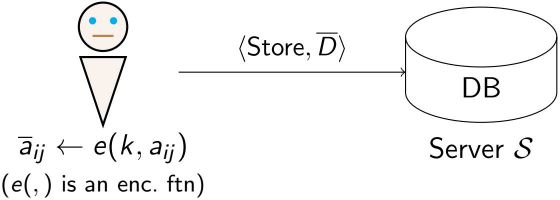


Database D

A_1	A_2	$\cdots$	A_n
a_{11}	a_{12}	$\cdots$	a_{1n}
$\vdots$	$\vdots$	$\cdots$	$\vdots$
a_{m1}	a_{m2}	$\cdots$	a_{mn}

System model

- Symmetric **Outsourced** Database: Somewhat private···
  - Client $\mathcal{U}$: 암호화한 DB $\overline{D}$를 Server $\mathcal{S}$에 저장
  - Technical issue: How to get $\alpha = \sum_{i=1}^{m} a_{ij}$?

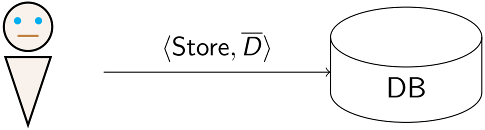


Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

System model

- Symmetric **Outsourced** Database: Somewhat private···
  - Client $\mathcal{U}$: 암호화한 DB $\overline{D}$를 Server $\mathcal{S}$에 저장
  - Technical issue: How to get $\alpha = \sum_{i=1}^{m} a_{ij}$?



$\overline{a}_{ij} \leftarrow e(k, a_{ij})$
($e(,)$ is an enc. ftn)

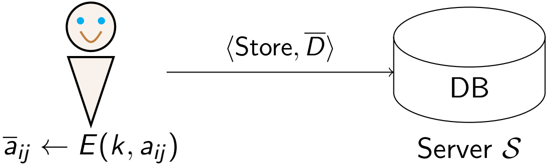
Server $\mathcal{S}$

Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

System model

- Symmetric **Outsourced** Database: Better private!
  - $*$ Client $\mathcal{U}$: Homo Enc $E()$로 암호화한 DB $\overline{D}$를 Server $\mathcal{S}$에 저장
  - $*$ $\mathcal{S}$ computes $\overline{\alpha} = \sum_{i=1}^{m} \overline{a}_{ij}$
  - $*$ $\mathcal{U}$ learns α by decrypting $\overline{\alpha}$

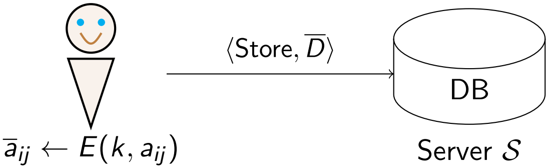


Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

System model

- Symmetric **Outsourced** Database: Better private!
  - ∗ Client $\mathcal{U}$: Homo Enc $E()$로 암호화한 DB $\overline{D}$를 Server $\mathcal{S}$에 저장
  - ∗ $\mathcal{S}$ computes $\overline{\alpha} = \sum_{i=1}^{m} \overline{a}_{ij}$
  - ∗ $\mathcal{U}$ learns α by decrypting $\overline{\alpha}$

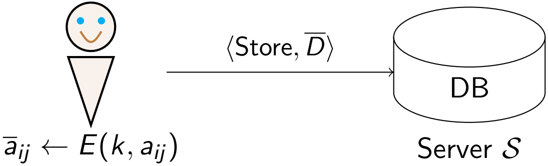


Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

System model

- Symmetric **Outsourced** Database: Better private!
  - $*$ Client $\mathcal{U}$: Homo Enc $E()$로 암호화한 DB $\overline{D}$를 Server $\mathcal{S}$에 저장
  - $*$ $\mathcal{S}$ computes $\overline{\alpha} = \sum_{i=1}^{m} \overline{a}_{ij}$
  - $*$ $\mathcal{U}$ learns α by decrypting $\overline{\alpha}$

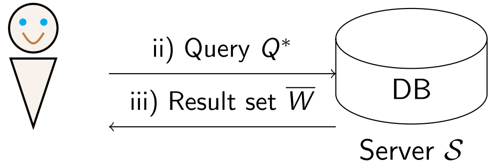


Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

Private Query

- How to query over $\overline{D}$?



i) $Q^* \leftarrow \text{Compiler}(k, D, \ldots)$

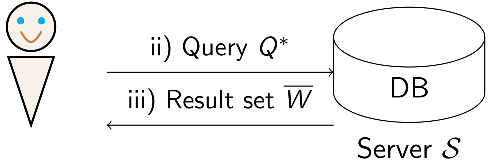
Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

- How to work the **Compiler**?
  1. Q ::= SELECT attribute_list FROM Relation WHERE select_condition;
  2. Encrypt all constants in select_condition
  3. Q^* ::= SELECT attribute_list FROM Relation WHERE $\overline{\text{select\_condition}}$;
  4. Run Q^* over $\overline{D}$

Private Query

- How to query over $\overline{D}$?



i) $Q^* \leftarrow \text{Compiler}(k, D, \ldots)$

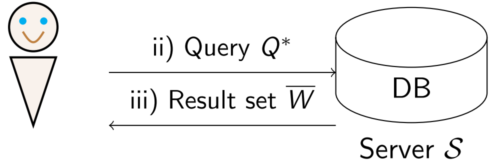
Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

- How to work the **Compiler**?
  1. Q ::= SELECT attribute_list FROM Relation WHERE select_condition;
  2. Encrypt all constants in select_condition
  3. Q^* ::= SELECT attribute_list FROM Relation WHERE $\overline{\text{select\_condition}}$;
  4. Run Q^* over $\overline{D}$

Private Query

- How to query over $\overline{D}$?



Database $\overline{D}$

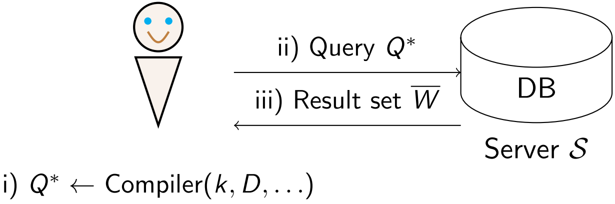
A_1	A_2	$\cdots$	A_n
$\bar{a}_{11}$	$\bar{a}_{12}$	$\cdots$	$\bar{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\bar{a}_{m1}$	$\bar{a}_{m2}$	$\cdots$	$\bar{a}_{mn}$

i) $Q^* \leftarrow \text{Compiler}(k, D, \ldots)$

- How to work the **Compiler**?
  1. Q ::= `SELECT` attribute_list `FROM` Relation `WHERE` select_condition;
  2. Encrypt all constants in select_condition
  3. Q^* ::= `SELECT` attribute_list `FROM` Relation `WHERE` $\overline{\text{select\_condition}}$;
  4. Run Q^* over $\overline{D}$

Private Query

- How to query over $\overline{D}$?



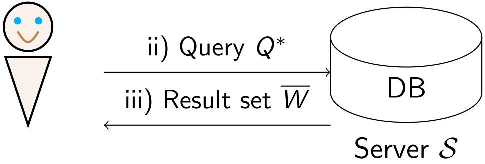
Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

- How to work the **Compiler**?
  1. Q ::= SELECT attribute_list FROM Relation WHERE select_condition;
  2. Encrypt all constants in select_condition
  3. Q^* ::= SELECT attribute_list FROM Relation WHERE $\overline{\text{select\_condition}}$;
  4. Run Q^* over $\overline{D}$

Private Query

- How to query over $\overline{D}$?



i) $Q^* \leftarrow \text{Compiler}(k, D, \ldots)$

Database $\overline{D}$

A_1	A_2	$\cdots$	A_n
$\overline{a}_{11}$	$\overline{a}_{12}$	$\cdots$	$\overline{a}_{1n}$
$\vdots$	$\vdots$	$\cdots$	$\vdots$
$\overline{a}_{m1}$	$\overline{a}_{m2}$	$\cdots$	$\overline{a}_{mn}$

- How to work the **Compiler**?
  1. Q ::= `SELECT` attribute_list `FROM` Relation `WHERE` select_condition;
  2. Encrypt all constants in select_condition
  3. Q^* ::= `SELECT` attribute_list `FROM` Relation `WHERE` $\overline{\text{select\_condition}}$;
  4. Run Q^* over $\overline{D}$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - ∗ An example
    - Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - $\overline{a}_1 \leftarrow E(k, \text{'A'})$ and $\overline{a}_2 \leftarrow E(k, \text{'M'})$
    - Q^* = SELECT Name FROM STUDENT WHERE Grd=$\overline{a}_1$ AND Sex=$\overline{a}_2$;
- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\texttt{EQ}(\text{Grd}[i], \overline{a}_1) \cdot \texttt{EQ}(\text{Sex}[i], \overline{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$
- Is everybody happy?

  ∗ ∗ ∗ No!! ∗ ∗ ∗

$$\texttt{EQ}(\overline{a}, \overline{b}) := (a = b)\ ?\ \overline{1} : \overline{0}$$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - ∗ An example
    - – Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - – $\bar{a}_1 \leftarrow E(k, \text{'A'})$ and $\bar{a}_2 \leftarrow E(k, \text{'M'})$
    - – Q^* = SELECT Name FROM STUDENT WHERE Grd=$\bar{a}_1$ AND Sex=$\bar{a}_2$;

- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\text{EQ}(\text{Grd}[i], \bar{a}_1) \cdot \text{EQ}(\text{Sex}[i], \bar{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$

- Is everybody happy?

  ∗ ∗ ∗ No!! ∗ ∗ ∗

$\text{EQ}(\bar{a}, \bar{b}) := (a = b)\ ?\ \bar{1} :\ \bar{0}$

Private Query

- Details of select_condition
  - ∗ An example
    - – Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - – $\overline{a}_1 \leftarrow E(k, \text{'A'})$ and $\overline{a}_2 \leftarrow E(k, \text{'M'})$
    - – Q^* = SELECT Name FROM STUDENT WHERE Grd=$\overline{a}_1$ AND Sex=$\overline{a}_2$;
- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\texttt{EQ}(\text{Grd}[i], \overline{a}_1) \cdot \texttt{EQ}(\text{Sex}[i], \overline{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$
- Is everybody happy?

∗ ∗ ∗ No!! ∗ ∗ ∗

$$\texttt{EQ}(\overline{a}, \overline{b}) := (a = b) \; ? \; \overline{1} : \overline{0}$$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - ∗ An example
    - Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - $\bar{a}_1 \leftarrow E(k, \text{'A'})$ and $\bar{a}_2 \leftarrow E(k, \text{'M'})$
    - Q^* = SELECT Name FROM STUDENT WHERE Grd=$\bar{a}_1$ AND Sex=$\bar{a}_2$;

- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\text{EQ}(\text{Grd}[i], \bar{a}_1) \cdot \text{EQ}(\text{Sex}[i], \bar{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$

- Is everybody happy?

  ∗ ∗ ∗ No!! ∗ ∗ ∗

$\text{EQ}(\bar{a}, \bar{b}) := (a = b) \; ? \; \bar{1} : \; \bar{0}$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - ∗ An example
    - – Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - – $\overline{a}_1 \leftarrow E(k, \text{'A'})$ and $\overline{a}_2 \leftarrow E(k, \text{'M'})$
    - – Q^* = SELECT Name FROM STUDENT WHERE Grd=$\overline{a}_1$ AND Sex=$\overline{a}_2$;

- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\text{EQ}(\text{Grd}[i], \overline{a}_1) \cdot \text{EQ}(\text{Sex}[i], \overline{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$

- Is everybody happy?

  ∗ ∗ ∗ No!! ∗ ∗ ∗

$\text{EQ}(\overline{a}, \overline{b}) := (a = b)\ ?\ \overline{1} : \overline{0}$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - ∗ An example
    - – Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - – $\overline{a}_1 \leftarrow E(k, \text{'A'})$ and $\overline{a}_2 \leftarrow E(k, \text{'M'})$
    - – Q^* = SELECT Name FROM STUDENT WHERE Grd=$\overline{a}_1$ AND Sex=$\overline{a}_2$;

- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\texttt{EQ}(\text{Grd}[i], \overline{a}_1) \cdot \texttt{EQ}(\text{Sex}[i], \overline{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$

- Is everybody happy?

  ∗ ∗ ∗ No!! ∗ ∗ ∗

$\texttt{EQ}(\overline{a}, \overline{b}) := (a = b) \ ? \ \overline{1} : \ \overline{0}$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - * An example
    - – Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - – $\overline{a}_1 \leftarrow E(k, \text{'A'})$ and $\overline{a}_2 \leftarrow E(k, \text{'M'})$
    - – Q^* = SELECT Name FROM STUDENT WHERE Grd=$\overline{a}_1$ AND Sex=$\overline{a}_2$;

- Computation on $\mathcal{S}$'s side
  - * Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\texttt{EQ}(\text{Grd}[i], \overline{a}_1) \cdot \texttt{EQ}(\text{Sex}[i], \overline{a}_2))$
  - * Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$

- Is everybody happy?

  * * * No!! * * *

$$\texttt{EQ}(\overline{a}, \overline{b}) := (a = b) \; ? \; \overline{1} : \; \overline{0}$$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - ∗ An example
    - – Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - – $\overline{a}_1 \leftarrow E(k, \text{'A'})$ and $\overline{a}_2 \leftarrow E(k, \text{'M'})$
    - – Q^* = SELECT Name FROM STUDENT WHERE Grd=$\overline{a}_1$ AND Sex=$\overline{a}_2$;
- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\texttt{EQ}(\text{Grd}[i], \overline{a}_1) \cdot \texttt{EQ}(\text{Sex}[i], \overline{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$
- Is everybody happy?
  - ∗ ∗ ∗ No!! ∗ ∗ ∗

$\texttt{EQ}(\overline{a}, \overline{b}) := (a = b)\ ?\ \overline{1} :\ \overline{0}$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - ∗ An example
    - – Q = SELECT Name FROM STUDENT WHERE Grd='A' AND Sex='M';
    - – $\overline{a}_1 \leftarrow E(k, \text{'A'})$ and $\overline{a}_2 \leftarrow E(k, \text{'M'})$
    - – Q^* = SELECT Name FROM STUDENT WHERE Grd=$\overline{a}_1$ AND Sex=$\overline{a}_2$;

- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\text{EQ}(\text{Grd}[i], \overline{a}_1) \cdot \text{EQ}(\text{Sex}[i], \overline{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$

- Is everybody happy?

  ∗ ∗ ∗ No!! ∗ ∗ ∗

$$\text{EQ}(\overline{a}, \overline{b}) := (a = b)\ ?\ \overline{1} : \overline{0}$$

Private Query

- Details of $\overline{\text{select\_condition}}$
  - ∗ An example
    - – $Q = \texttt{SELECT}$ Name $\texttt{FROM}$ STUDENT $\texttt{WHERE}$ Grd='A' $\texttt{AND}$ Sex='M';
    - – $\bar{a}_1 \leftarrow E(k, \text{'A'})$ and $\bar{a}_2 \leftarrow E(k, \text{'M'})$
    - – $Q^* = \texttt{SELECT}$ Name $\texttt{FROM}$ STUDENT $\texttt{WHERE}$ Grd=$\bar{a}_1$ $\texttt{AND}$ Sex=$\bar{a}_2$;

- Computation on $\mathcal{S}$'s side
  - ∗ Construct a circuit C_{Q^*} : $\forall i, \text{Name}[i] \cdot (\texttt{EQ}(\text{Grd}[i], \bar{a}_1) \cdot \texttt{EQ}(\text{Sex}[i], \bar{a}_2))$
  - ∗ Run the circuit, $\overline{W} = \{\overline{w}_1, \ldots, \overline{w}_m\} \leftarrow C_{Q^*}(\overline{D})$

- Is everybody happy?

  ∗ ∗ ∗ No!! ∗ ∗ ∗

$$\texttt{EQ}(\bar{a}, \bar{b}) := (a = b)\ ?\ \bar{1} :\ \bar{0}$$

Problem statement

Why "No"?

- $\mathcal{S}$ can learn the logical operators (e.g., and/or) from Q^*
- Our problem

> *How to hide the operators from the suspicious server?*

An Example

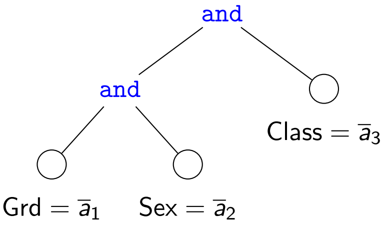


Figure 1: A Query Tree

SELECT Name, Depart, Address
FROM STUDENT
WHERE Grd = $\bar{a}_1$
AND Sex = $\bar{a}_2$
AND Class = $\bar{a}_3$;

Figure 2: An SQL Statement

Possible approaches

1. Searchable symmetric encryption (SSE)
   * Single/multi-keyword search (e.g., [SWP00, CGKO06, BW07, HK14])
     ⇒ Reveal query statements
   * SSE+2-party Protocol (e.g., [CJJ+13])
     ⇒ Reveal query conditions
   * Dynamic SSE (e.g., [KPR12, KP13, NPG14])
     ⇒ Allow update operations but reveal query statements
   * Multi-user SSE (e.g., [JJK+13]
     ⇒ Insecure against collusion among users
   * Zhang et al.'s powerful attack over SSE [ZKP16]
2. ORAM
   * Great privacy but very costly (e.g., [SDS+13, NPG14])
3. Hybrid techniques
   * CryptDB and its variants (e.g., [SNCB15])
     ⇒ OPE-level security but practical

Possible approaches

1. Searchable symmetric encryption (SSE)
   * Single/multi-keyword search (e.g., [SWP00, CGKO06, BW07, HK14])
     $\Rightarrow$ Reveal query statements
   * SSE+2-party Protocol (e.g., [CJJ+13])
     $\Rightarrow$ Reveal query conditions
   * Dynamic SSE (e.g., [KPR12, KP13, NPG14])
     $\Rightarrow$ Allow update operations but reveal query statements
   * Multi-user SSE (e.g., [JJK+13]
     $\Rightarrow$ Insecure against collusion among users
   * Zhang et al.'s powerful attack over SSE [ZKP16]
2. ORAM
   * Great privacy but very costly (e.g., [SDS+13, NPG14])
3. Hybrid techniques
   * CryptDB and its variants (e.g., [SNCB15])
     $\Rightarrow$ OPE-level security but practical

Possible approaches

1. Searchable symmetric encryption (SSE)
   * Single/multi-keyword search (e.g., [SWP00, CGKO06, BW07, HK14])
     ⇒ Reveal query statements
   * SSE+2-party Protocol (e.g., [CJJ+13])
     ⇒ Reveal query conditions
   * Dynamic SSE (e.g., [KPR12, KP13, NPG14])
     ⇒ Allow update operations but reveal query statements
   * Multi-user SSE (e.g., [JJK+13]
     ⇒ Insecure against collusion among users
   * Zhang et al.'s powerful attack over SSE [ZKP16]
2. ORAM
   * Great privacy but very costly (e.g., [SDS+13, NPG14])
3. Hybrid techniques
   * CryptDB and its variants (e.g., [SNCB15])
     ⇒ OPE-level security but practical

Possible approaches

1. Searchable symmetric encryption (SSE)
    * Single/multi-keyword search (e.g., [SWP00, CGKO06, BW07, HK14])
      ⇒ Reveal query statements
    * SSE+2-party Protocol (e.g., [CJJ+13])
      ⇒ Reveal query conditions
    * Dynamic SSE (e.g., [KPR12, KP13, NPG14])
      ⇒ Allow update operations but reveal query statements
    * Multi-user SSE (e.g., [JJK+13]
      ⇒ Insecure against collusion among users
    * Zhang et al.'s powerful attack over SSE [ZKP16]
2. ORAM
    * Great privacy but very costly (e.g., [SDS+13, NPG14])
3. Hybrid techniques
    * CryptDB and its variants (e.g., [SNCB15])
      ⇒ OPE-level security but practical

Idea sketch

Our goals

1. Hide the select_condition clause
   ⇒ Protect both private constants & logical operators
2. Harmonize security and performance
   ⇒ Allow to reveal the `select`-statement & the `from`-statement
   ⇒ Apply SIMD, Automorphism, Dynamic programming, Heuristics

Our settings

1. Underlying encryption: (leveled) Fully homomorphic encryption
2. Main tool: `EQ` circuit
   * $\mathrm{depth}(\mathtt{EQ}) = \lceil \log n \rceil$ for two n-bit inputs
3. Support: Conj., Disj. and Threshold Conj. queries

Our idea

- Express all of target queries as the **same structure** of circuits

Idea sketch

Our goals

1. Hide the select_condition clause
   ⇒ Protect both private constants & logical operators
2. Harmonize security and performance
   ⇒ Allow to reveal the `select`-statement & the `from`-statement
   ⇒ Apply SIMD, Automorphism, Dynamic programming, Heuristics

Our settings

1. Underlying encryption: (leveled) Fully homomorphic encryption
2. Main tool: EQ circuit
   * $\mathrm{depth}(\mathtt{EQ}) = \lceil \log n \rceil$ for two n-bit inputs
3. Support: Conj., Disj. and Threshold Conj. queries

Our idea

- Express all of target queries as the **same structure** of circuits

Idea sketch

Our goals

1. Hide the select_condition clause
   ⇒ Protect both private constants & logical operators
2. Harmonize security and performance
   ⇒ Allow to reveal the `select`-statement & the `from`-statement
   ⇒ Apply SIMD, Automorphism, Dynamic programming, Heuristics

Our settings

1. Underlying encryption: (leveled) Fully homomorphic encryption
2. Main tool: EQ circuit
   * $\mathrm{depth}(\mathtt{EQ}) = \lceil \log n \rceil$ for two n-bit inputs
3. Support: Conj., Disj. and Threshold Conj. queries

Our idea

- Express all of target queries as the **same structure** of circuits

Idea sketch

Our goals

1. Hide the select_condition clause
   ⇒ Protect both private constants & logical operators
2. Harmonize security and performance
   ⇒ Allow to reveal the `select`-statement & the `from`-statement
   ⇒ Apply SIMD, Automorphism, Dynamic programming, Heuristics

Our settings

1. Underlying encryption: (leveled) Fully homomorphic encryption
2. Main tool: EQ circuit
   * $\mathrm{depth}(\mathtt{EQ}) = \lceil \log n \rceil$ for two n-bit inputs
3. Support: Conj., Disj. and Threshold Conj. queries

Our idea

- Express all of target queries as the **same structure** of circuits

Idea sketch

Our goals

1. Hide the select_condition clause
   ⇒ Protect both private constants & logical operators
2. Harmonize security and performance
   ⇒ Allow to reveal the `select`-statement & the `from`-statement
   ⇒ Apply SIMD, Automorphism, Dynamic programming, Heuristics

Our settings

1. Underlying encryption: (leveled) Fully homomorphic encryption
2. Main tool: `EQ` circuit
   * $\mathrm{depth}(\mathtt{EQ}) = \lceil \log n \rceil$ for two n-bit inputs
3. Support: Conj., Disj. and Threshold Conj. queries

Our idea

- Express all of target queries as the **same structure** of circuits

Idea sketch

Our goals

1. Hide the select_condition clause
   ⇒ Protect both private constants & logical operators
2. Harmonize security and performance
   ⇒ Allow to reveal the `select`-statement & the `from`-statement
   ⇒ Apply SIMD, Automorphism, Dynamic programming, Heuristics

Our settings

1. Underlying encryption: (leveled) Fully homomorphic encryption
2. Main tool: `EQ` circuit
   * $\mathrm{depth}(\texttt{EQ}) = \lceil \log n \rceil$ for two n-bit inputs
3. Support: Conj., Disj. and Threshold Conj. queries

Our idea

- Express all of target queries as the **same structure** of circuits

Idea sketch

Our goals

1. Hide the select_condition clause
   ⇒ Protect both private constants & logical operators
2. Harmonize security and performance
   ⇒ Allow to reveal the `select`-statement & the `from`-statement
   ⇒ Apply SIMD, Automorphism, Dynamic programming, Heuristics

Our settings

1. Underlying encryption: (leveled) Fully homomorphic encryption
2. Main tool: EQ circuit
   * $\mathrm{depth}(\mathtt{EQ}) = \lceil \log n \rceil$ for two n-bit inputs
3. Support: Conj., Disj. and Threshold Conj. queries

Our idea

- Express all of target queries as the **same structure** of circuits

Idea sketch

Main observations

1. Using EQ, target queries can be expressed into the same circuit
   * $\mathbb{F}_{2^\ell}$: the plaintext domain
   * $a_i, b, c, d \in \mathbb{F}_{2^\ell}$ and $A_i \in \mathbb{F}_{2^\ell}$
   * Circuit $C^\star$ defined by

$$C^\star = \overline{d} + \prod_{i=1}^{n} \left(\overline{b} + \mathtt{EQ}\left(\overline{A}_i, \overline{a}_i\right) \cdot \overline{c} \right)$$

2. Evaluation table

Query type	b	c	d	Result of $C^\star$
Conjunction	0	1	0	0/1
Disjunction	1	1	1	0/1
Threshold Conjunction	1	$1+t$	0	t^κ

$t \in \mathbb{F}_{2^\ell}^*$ and κ: # of threshold conditions

Idea sketch

Main observations

1. Using EQ, target queries can be expressed into the same circuit
   * $\mathbb{F}_{2^\ell}$: the plaintext domain
   * $a_i, b, c, d \in \mathbb{F}_{2^\ell}$ and $A_i \in \mathbb{F}_{2^\ell}$
   * Circuit $C^\star$ defined by

$$C^\star = \overline{d} + \prod_{i=1}^{n} \left(\overline{b} + \mathtt{EQ}\left(\overline{A}_i, \overline{a}_i\right) \cdot \overline{c} \right)$$

2. Evaluation table

Query type	b	c	d	Result of $C^\star$
Conjunction	0	1	0	0/1
Disjunction	1	1	1	0/1
Threshold Conjunction	1	$1+t$	0	t^κ

$t \in \mathbb{F}^*_{2^\ell}$ and κ: # of threshold conditions

Idea sketch

Main observations–continued

- Still, a bit problem: the results for threshold queries are still $\overline{t^i}$
  $\Rightarrow$ Modify the results into either $\overline{0}$ or $\overline{1}$
- Our technique
  - ∗ $\mathcal{S}$ is required to evaluate an encrypted polynomial $\overline{g}(X)$ regardless of query types

$$g(X) = \begin{cases} 1 & \text{if a condition holds} \\ 0 & \text{otherwise} \end{cases}$$

where for a polynomial $g = \sum g_i X^i \in \mathbb{F}_{2^\ell}[X]$, its encryption $\bar{g} := (\bar{g}_i)_i$

Idea sketch

Main observations–continued

- Still, a bit problem: the results for threshold queries are still $\overline{t^i}$
  $\Rightarrow$ Modify the results into either $\overline{0}$ or $\overline{1}$
- Our technique
  - $\mathcal{S}$ is required to evaluate an encrypted polynomial $\overline{g}(X)$ regardless of query types

$$g(X) = \begin{cases} 1 & \text{if a condition holds} \\ 0 & \text{otherwise} \end{cases}$$

where for a polynomial $g = \sum g_i X^i \in \mathbb{F}_{2^\ell}[X]$, its encryption $\bar{g} := (\bar{g}_i)_i$

The construction

- Client's activities
  $R(A_1, A_2, \ldots, A_m)$: a table schema.
  $J \subseteq [m] := \{1, 2, \ldots, m\}$: indices of attributes at query-condition.
  $a_{j \in J}$: a constant for comparison to the attribute value in A_j.

Conjunctive: $\bigwedge_{j \in J}(A_j = a_j)$

1. $j \in J$: $b_j = 0, c_j = 1$
2. $j \notin J$: $b_j = 1, c_j = 0, a_j \xleftarrow{\$} \mathbb{F}_{2^\ell}$
3. $\forall j \in [m], d_j = 0$
4. $g \in \mathbb{F}_{2^\ell}[X]$ such that
   $g(1) = 0 \wedge g(0) = 0$

Disjunctive: $\bigvee_{j \in J}(A_j = a_j)$

1. $j \in J$: $b_j = 1, c_j = 1$
2. $j \notin J$: $b_j = 1, c_j = 0, a_j \xleftarrow{\$} \mathbb{F}_{2^\ell}$
3. $\forall j \in [m], d_j = 1$
4. $g \in \mathbb{F}_{2^\ell}[X]$ such that
   $g(1) = 0 \wedge g(0) = 0$

The construction

- Client's activities
  $R(A_1, A_2, \ldots, A_m)$: a table schema.
  $J \subseteq [m] := \{1, 2, \ldots, m\}$: indices of attributes at query-condition.
  $a_{j \in J}$: a constant for comparison to the attribute value in A_j.

Conjunctive: $\bigwedge_{j \in J}(A_j = a_j)$

1. $j \in J$: $b_j = 0, c_j = 1$
2. $j \notin J$: $b_j = 1, c_j = 0, a_j \xleftarrow{\$} \mathbb{F}_{2^\ell}$
3. $\forall j \in [m], d_j = 0$
4. $g \in \mathbb{F}_{2^\ell}[X]$ such that $g(1) = 0 \wedge g(0) = 0$

Disjunctive: $\bigvee_{j \in J}(A_j = a_j)$

1. $j \in J$: $b_j = 1, c_j = 1$
2. $j \notin J$: $b_j = 1, c_j = 0, a_j \xleftarrow{\$} \mathbb{F}_{2^\ell}$
3. $\forall j \in [m], d_j = 1$
4. $g \in \mathbb{F}_{2^\ell}[X]$ such that $g(1) = 0 \wedge g(0) = 0$

The construction

- Client's activities–continued
  - $R(A_1, A_2, \ldots, A_m)$: a table schema
  - $J \subseteq [m] := \{1, 2, \ldots, m\}$: indices of attributes at query-condition
  - $a_{j \in J}$: a constant for comparison to the attribute value in A_j

Conjunctive for a threshold T

1. $j \in J$: $b_j = 1, c_j = (1 + t)$
2. $j \notin J$: $b_j = 1, c_j = 0, a_j \stackrel{\$}{\leftarrow} \mathbb{F}_{2^\ell}$
3. $\forall j \in [m], d_j = 0$
4. $g \in \mathbb{F}_{2^\ell}[X]$ such that $g(t^\kappa) = \begin{cases} 1 & \text{if } T < \kappa \le m \\ 0 & \text{if } 0 \le \kappa \le T \end{cases}$

- The client encrypts all a_j, b_j, c_j, d_i and all coefficients of g and sends them to the server.

The construction

- Server's activities
  - $R(A_1, A_2, \ldots, A_m)$: a table schema
  - n: # of tuples
  - A_i: an attribute in the `select` statement

1. Receive $(\bar{a}_j, \bar{b}_j, \bar{c}_j, \bar{d}_j)_{j\in[m]}$ and $\bar{g}$
2. Compute $\bar{\beta}_{ij} = \bar{b}_j + \mathtt{EQ}(\bar{A}_j[i], \bar{a}_j) \cdot \bar{c}_j$
   where i: tuple index and j: attribute index
3. Compute $\bar{\zeta}_i = \bar{d}_j + \prod_{j\in[m]} \bar{\beta}_{ij}$
4. Compute $\bar{\gamma}_i = \bar{g}(\bar{\zeta}_i) \cdot \bar{A}_i$
5. Return $\{\bar{\gamma}_i\}_{i\in[n]}$

Evaluation

Cost estimation

- Computation costs
  1. For $\bar{\beta}_{ij}$: $\log \ell + 1$ mul. depth
  2. For $\prod \bar{\beta}_{ij}$: $\log m$ mul. depth
  3. For $\bar{\gamma}_i$: $\log(m+1) + 1$ mul. depth
- Communication costs
  1. Client: $n + 4m + 1$ BGV ciphertexts
  2. Server: n BGV ciphertexts

Evaluation

PoC implementation

- NTL+GMP+HElib-based implmentation
- Dataset: a relation schema with degree 12 & each attribute of about 40 bits
- Parameter selection for the BGV scheme
  1. Security parameter:80 $\sim$ 125
  2. Multiplcative depth: 17, # of slots: 336 $\sim$ 396
  3. # of tuples: 336 $\sim$ 16384

λ	m	$\phi(m)$	$\mathcal{P}$	# of Slots in a Ciphertext	Query Encrypt (Client)	EQTest (Server)	Total Time (Server)	Amortised Time (Server)	Result Decrypt (Client)
80	14491	14112	$\mathbb{F}_{2^{42}}$	336	2.00 sec	33.66 sec	39.61 sec	0.12 sec	0.02 sec
99	30705	15488	$\mathbb{F}_{2^{44}}$	352	4.00 sec	64.29 sec	75.92 sec	0.22 sec	0.04 sec
104	17173	15840	$\mathbb{F}_{2^{60}}$	264	3.00 sec	64.17 sec	73.85 sec	0.28 sec	0.04 sec
118	31695	16896	$\mathbb{F}_{2^{44}}$	384	4.00 sec	67.96 sec	80.90 sec	0.21 sec	0.05 sec
125	27393	17424	$\mathbb{F}_{2^{44}}$	396	4.00 sec	67.90 sec	80.69 sec	0.20 sec	0.05 sec

Each element consists of 11 attributes of 40-bit entries and we used a leveled BGV scheme of 17 levels.
λ: security parameter (bit), m: parameter for FFT, $\phi(m)$: the dimension of the polynomial ring of the utilized BGV scheme
$\mathcal{P}$: the plaintext space of the utilized BGV scheme

Wrap-up

Summary

- Project review
- Description of our techniques
- Review of experimental implementation

Wrap-up

Summary

- Project review
- Description of our techniques
- Review of experimental implementation

********** Thanks & Question? **********

Wrap-up

Summary

- Project review
- Description of our techniques
- Review of experimental implementation

********** Thanks & Question? **********

References

[ARCI13] M. Asghar, G. Russello, B. Crispo & M. Ion: *Supporting complex queries and access policies for multi-user encrypted databases*. CCSW 2013.

[BW07] D. Boneh & B. Waters: *Conjunctive, subset, and range queries on encrypted data.* TCC 2007.

[BGV12] Z. Brakerski, C. Gentry & V. Vaikuntanathan: *(Leveled) fully homomorphic encryption without bootstrapping*. ITCS 2012.

[CWL+14] N. Cao, C. Wang, M. Li, K. Ren & W. Lou: *Privacy-preserving multi-keyword ranked search over encrypted cloud data.* IEEE Tran. Parallel and Dist. Sys. 2014.

[CJJ+13] D. Cash, S. jarecki, C. Jutla, H. Krwaczyk, M. Roșu & M. Steiner: *Highly scalable searchable symmetric encryption with support for Boolean queries.* Crypto 2013.

References

[CGKO06] R. Curtmola, J. Garay, S. Kamara & R. Ostrovsky: *Searchable symmetric encryption: improved definitions and efficient constructions*. CCS 2006.

[FPCM14] L. Ferretti, F. Pierazzi, M. Clajanni & M. Marchetti: *Scalable architecture for multi-user encrypted SQL operations on cloud database service.* Trans. Parallel Distrib. Syst. 2014.

[HK14] F. Hahn & F. Kerschbaum: *Searchable encryption with secure and efficient updates.* CCS 2014.

[HElib] S. Halevi & V. Shoup: *Algorithms in HElib*, Eurocrypt 2014.

[HKD15] I. Hang, F. Kerschbaum & E. Damiani: *ENKI: Access control for encrypted query processing.* SIGMOD 2015.

[IKLO16] Y. Ishai, E. Kushilevitz, S. Lu & R. Ostrovsky: *Private large-scale databases with distributed searchable symmetric encryption.* CT-RSA 2016.

References

[JJK+13] S. Jarecki, C. Jutla, H. Krawczyk, M. Roșu & M. Steiner: *Outsourced symmetric private information retrieval.* CCS 2013.

[KP13] S. Kamara & C. Papamanthou: *Parallel and dynamic searchable encryption.* FC 2013.

[KPR12] S. Kamara, C. Papamanthou & T. Roeder: *Dynamic searchable symmetric encryption.* CCS 2012.

[NPG14] M. Naveed, M. Prabhakaran & C. Gunter: *Dynamic searchable encrypted via blind storage.* S&P 2014.

[NTL] NTL: A Library for doing Number Theory, `http://http://www.shoup.net/ntl/`.

[RG15] P. Rizomiliotis & S. Gritzalis: *ORAM-based forward privacy preserving dynamic searchable symmetric encryption schemes.* CCSW 2015.

References

[SNCB15] M. Sarfraz, M. Nabeel, J. Cao & E. Bertano: *DBMask: Fine-grained access control on encrypted databases.* CODASPY 2015.

[SJB14] B. Samanthula, W. Jiang & E. Bertino: *Privacy-preserving complex query evaluation over semantically secure encrypted data.* ESORICS 2014.

[SWP00] D. Song, D. Wagner & A. Perrig: *Practical techniques for searches on encrypted data.* S&P 2000.

[SPS14] E. Stefanov, C. Papamanthou & E. Shi: *Practical dynamic searchable encryption with small leakage*. NDSS 2014.

[SDS+13] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu & S. Devadas: *Path ORAM: An extremely simple Oblivious RAM protocol.* CCS 2013.

References

[WSLH15] B. Wang, W. Song, W. Lou & Y. Hou: *Inverted index-based multi-keyword public-key searchable encryption with strong privacy guarantee.* INFOCOM 2015.

[YLCZ15] Y. Yang, J. Liu, K. Choo & J. Zhou: *Extended proxy-assisted approach: Achieving revocable fine-grained encryption of cloud data.* ESORICS 2015.

[ZKP16] Y. Zhang, J. Katz & C. Papamanthou: *All your queries are belong to us: The power of file-injection attacks on searchable encryption.* Usenix Security 2016.