Security in Outsourced Databases
(Query Answer Assurance)

Traditional Client-Server Arch.

- Client queries are satisfied by a trusted server
- Secure the server
- Secure the communication channel, e.g. use SSL

DB Client — Query — Results — Owner

1

2
Data Publishing
(Database-as-a-Service)

DB Client

Owner

Database

Third Party Server

Query

Results

DB Client

Owner

Third Party Server
Data Publishing

- Pushes business logic and data processing from corporate data centers to third party servers at the “edge” of the network
  - Distribution of (part of) the database to edge servers
  - Edge servers perform query processing
- Why?
  - Most organizations need DBMSs
  - DBMSs extremely complex to deploy, setup, maintain
  - Require skilled DBAs (at very high cost!)
- Advantages
  - Cuts down network latency and produces faster responses
  - Cheaper way to achieve scalability
  - Lowers dependency on corporate data center (removes single point of failure)
  - Reduced cost to client
    - Get what you need, pay for what you use and not for: hardware, software infrastructure or personnel to deploy, maintain, upgrade…
  - Reduced overall cost
    - cost amortization across users
  - Better service
    - leveraging experts

The Challenge

DB Client

Owner

Third Party Server

Untrusted!

Query

Results

The Truth?
The Whole Truth?
Nothing But The Truth?
The Challenge

Sel * FROM Emp
WHERE Sal < 5000

ID   Name  Sal    Dept
5    A      2000    1
2    C      3500    2
1    D      8010    1
4    B      2200    3
3    E      7000    2

Result =
2    C      3500    2
4    B      2200    3

The Challenge

Sel * FROM Emp
WHERE Sal < 5000

ID   Name  Sal    Dept
5    A      2000    1
2    C      3500    2
1    D      8010    1
4    B      2200    3
3    E      7000    2

Result =
2    C      3500    2
4    B      2200    3
Security Concerns

Server ID   Name  Sal    Dept
5   A    2000   1
2   C    3500   2
1   D    8010   1
4   B    2200   3
3   E    7000   2

5 A  2000  1
Result =  2 C  3500  2
        4 B  2200  3

Query

Result'

DB Client

Server

Server is trustworthy!
Security Concerns

Server is malicious!
Records are tampered

Server is malicious!
Answers are dropped
(Incompleteness)
Security Concerns

Server is malicious!
Spurious answers are added

Data Security Challenge:

Design Objectives:
• **Authenticity**: Every entry originated from the owner
• **Completeness**: No result entry is omitted from the answer
• **Precision**: Minimum information leakage
• **Security**: Computationally infeasible to cheat
• **Efficiency**: Polynomial proof
Collision-resistant (one-way) hash functions

- Given x, easy to compute h(x); given h(x), difficult to determine x
- i.e., it is computationally hard to find x₁ and x₂ s.t. h(x₁) = h(x₂)
- Computational hard? Based on well established assumptions such as discrete logarithms
- E.g., SHA, MD5

Public key digital signature schemes

Cryptographic tool for authenticating the signed message as well as its origin, e.g., RSA, DSA

- KeyGen → (SK, PK)
- SK
- Sign(h(m), SK) → \( \sigma \)
- Insecure Channel
- m
- Ver(m, PK, \( \sigma \)) → valid?
- By checking:
  - h(m) =? Sign⁻¹(PK, \( \sigma \))
Authentic Publication Scheme

Naïve Scheme

Each attribute has a signed digest
Each tuple has a signed digest

Relation R

<table>
<thead>
<tr>
<th>$D_T$</th>
<th>$(A_i, D_i)$</th>
<th>...</th>
<th>$(A_i, D_i)$</th>
<th>...</th>
</tr>
</thead>
</table>

$D_T$ – Signed tuple digest
$D_{Ai}$ – attribute digest
Naïve Scheme

Query: SELECT A3, A4, … FROM R

<table>
<thead>
<tr>
<th>Result tuples</th>
<th>Filtered attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1 A1 A2 …</td>
<td>D1 D2 D3 …</td>
</tr>
</tbody>
</table>

D1 – Signed tuple digest
D1 – attribute digest of A1

Naïve Scheme (Example)

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
<th>C1</th>
<th>a1</th>
<th>b1</th>
<th>c1</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td>T2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>a3</td>
<td>b3</td>
<td>c3</td>
<td>T3</td>
</tr>
</tbody>
</table>

T = sign(g(h(A)|h(B)|h(C))
g and h are collision-resistant hash functions
ai = h(Ai)

Retrieve whole of first tuple:
Server returns A1, B1, C1, T1; Client can compute h(A1), h(B1) and h(C1), and verify T1 from A1, B1 and C1

Retrieve only attributes A1 and B1 of first tuple:
Server returns A1, B1, c1 and T1; Client has no access to C1, so c1 has to be provided

Issues??
Using Merke Hash Tree (MHT)

- For each tuple \( t \), a tuple hash \( h(t) \) is computed
  \[
  h(t) = h(h(t.A1) | h(t.A2) | \ldots | h(t.An))
  \]
- Assume a total order on attribute \( A \) of a relation \( R \) with \( |R| \) tuples (e.g., based on the primary key)
  - \( \text{MHT}(R,A) \) is a binary tree with \( |R| \) leaf nodes and hash values \( h(i) \) associated with node \( i \)
  - If \( i \) is a leaf node, then \( h(i) = h(t_i) \), \( t_i \) is the \( i \)th tuple in the order
  - If \( i \) is an internal node, then \( h(i) = h(h(l), h(r)) \) where \( l \) and \( r \) are the left and right children of node \( i \).
  - The root hash is the digest of all values in the Merkle-hash tree \( \text{MHT}(R,A) \).

Merkle Hash Tree

Ordering attribute: \( k_1 < k_2 < k_3 < k_4 \); \( d_i \) are tuples
Owner needs to sign root node \( (N_{1234}) \)
MHT: Point Search

\[ N_{1234} = h(N_{12} \mid N_{34}) \]

\[ \text{Sign}(h_{1234}, SK) \]

Query: Retrieve tuple \( d_2 \)

Edge server returns \( d_2, N_1, N_{34} \) and signed \( N_{1234} \)
Client computes \( N_{1234} = h(h(h(d_2)\mid N_1), N_{34}) \) and verify that the signed value is correct.
MHT: Point Search

Edge server returns $d_2$, $N_1$, $N_3$, and signed $N_{1234}$ (and the structure)
Client computes $N_{1234} = h(h(h(d_2)|N_1), N_3)$ and verify that the signed value is correct.

Range Queries
Example: Range queries

What are returned?

Example: Range queries

What are returned?

digest
Example: Range queries

What are returned?

Proving Authenticity is Easy

Certified Hash Tree

h_a = h(h(2)|h(4))

h(2)  h(4)  h(6)  h(8)  h(10)  h(12)

Data:  2   4   6   8   10  12

Query: 5 ≤ r ≤ 7
Proving Authenticity is Easy

Certified Hash Tree

\[
\begin{align*}
& s(h_d) \\
& h_c \\
& h_a \\
& h(2) \quad h(4) \quad h(6) \\
& h_b \\
& h(8) \quad h(10) \quad h(12) \\
& h_c \\
\end{align*}
\]

Data: 2 4 6 8 10 12

Query: \(5 \leq r \leq 7\)

Proving Completeness is Easy But …

Certified Hash Tree

\[
\begin{align*}
& s(h_d) \\
& h_c \\
& h_a \\
& h(2) \quad h(4) \quad h(6) \\
& h_b \\
& h(8) \quad h(10) \quad h(12) \\
& h_c \\
\end{align*}
\]

Data: 2 4 6 8 10 12

Query: \(5 \leq r \leq 7\)
Precision may be compromised!

Certified Hash Tree

\[
\begin{align*}
&h_c \
\rightarrow & h_a & h_b & h_c \\
&h(2) & h(4) & h(6) & h(8) & h(10) & h(12) \\
\end{align*}
\]

Data:  2  4  6  8  10  12

Query: 5 \leq r \leq 7

- Compromise precision: Disclose left and right neighbors
- May violate access control policy

Example

- Access control: U can only see records with salary < 8000
- Results are records 2, 3, and 5.
- If system does not return record 1, U will not know that the answer is complete since it is possible that there is a record with Sal > 7000 but < 8000 that is not returned.
- If system returns record 1, then it violates the access control policy!
- Need an authentication mechanism that verifies completeness without compromising access control rules
What’s the problem?

• A Merkle hash tree is needed for every sort-order on a table
• VO (Verification Object – the data used for verification) needs to contain links all the way to the root,
  – VO grows linearly to query result and logarithmic to base table size
• Projections may have to be performed by clients
• No provision for dynamic updates on the database
• Weak in terms of access control
  – Attributes that are supposed to be filtered out must also be returned for verification

A signature-chain-based scheme:
Let’s start simple …

• Consider a sorted list of distinct integers, \( R = \{r_1, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots r_n\} \)
• Retrieve record whose value is greater than or equal to \( \alpha \)
  – \( \alpha \leq r \) (i.e., \( \sigma_{\alpha \leq r} (R) \))
• Result \( Q = \{r_a, r_{a+1}, \ldots r_b\} \), i.e., \( r_{a-1} < \alpha \leq r_a \leq r_{a+1} < \ldots r_b = r_n \)
• Result is complete if:
  – Contiguity: Each pair of successive entries \( r_i, r_{i+1} \) in \( Q \) also appears in \( R \) (based on Signature Chain)
  – Terminal: Last element of \( Q \) is also last element of \( R \), i.e., \( r_b = r_n \) (based on Signature Chain)
  – Origin: \( r_a \) is the first element in \( R \) that satisfies the query condition, i.e., \( r_{a-1} < \alpha \leq r_a \) (based on Private Boundary Proof)
Signature Chain

- For each data value, there is an associated signature
  - Computed from its own value, and that of its left and right neighbors
  - \( \text{sig}(r_i) = s(h(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1}))) \)

- Owner stores the \((r_i, \text{sig}(r_i))\) pair in the server
- During querying, server returns (answer, signature) pairs and more ...(verification objects) ...

\[ h_i(r) = h_{i-1}(h(r)) \quad h^0(r) = h(r) \quad g(r) = h^{U-r-1}(r) \]

\( U = \text{max value outside of domain} (\text{known to all users}) \)

\( s \) is a signature function using owner’s private key

---

Signature Chain Ensures Contiguity

Server returns \((r_i, \text{sig}(r_i))\)-pairs

Result Q

<table>
<thead>
<tr>
<th>Server: ( \ldots )</th>
<th>( r_{i-1} )</th>
<th>( r_i )</th>
<th>( r_{i+1} )</th>
<th>( r_{i+2} )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>User: ( \ldots )</td>
<td>( g(r_{i-1}) )</td>
<td>( g(r_i) )</td>
<td>( g(r_{i+1}) )</td>
<td>( g(r_{i+2}) )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

Client: \( \text{ver}(H_i, \text{sig}(r_i), \text{PK})? \)

\( H_i = h(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1})) \)

Signature chain: \( \text{sig}(r_i) = s(h(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1}))) \)
Signature Chain Ensures Contiguity

Query: $55 \leq r$

Result $Q$

Server:

| 60 | 70 | 80 | 90 |

User:

| 60 | 70 | 80 | 90 |

ver($H_{70}$, sig(70), PK)?

ver($H_{80}$, sig(80), PK)?

INCORRECT! Data has been tampered!
Signature Chain Ensures Contiguity

Query: $55 \leq r$

Server: $\cdots \ 60 \ \times \ 80 \ 90 \ \cdots$

User: $\cdots \ g(60) \ g(80) \ g(90) \ \cdots$

Result $Q$

$ver(H_{80}, \ sig(80), \ PK)$?

$H_{60-80}$ will be computed (without 70) - will not match sig(80). INCORRECT!!!

How To Ensure $r_n$ Is The Last Record?

Result $Q$

Server: $r_a \ r_{a+1} \ \cdots \ r_n$

User: $g(r_a) \ g(r_{a+1}) \ \cdots \ g(r_n)$

Create a fictitious record $r_{n+1}$ that is larger than the largest value but smaller than $U$

$\bullet \ sig(r_{n+1}) = s(h(g(r_n)\|g(r_{n+1})\|h(U)))$
**Signature Chain Ensures Terminal**

Result $Q$

Server: $r_a$, $r_{a+1}$, ..., $r_n$, $g(r_{n+1})$

User: $g(r_a)$, $g(r_{a+1})$, ..., $g(r_n)$, $U$

Create a fictitious record $r_{n+1}$ that is larger than the largest value but smaller than $U$

- $\text{sig}(r_{n+1}) = s(h(g(r_n)|g(r_{n+1})|h(U)))$
- server returns $g(r_{n+1})$ instead of $r_{n+1}$

$\text{ver}(H_{n+1}, \text{sig}(r_{n+1}), PK)$?

**How to prove Origin (without revealing the boundary point)?**

40 50 60 70 80 ....
How to prove Origin??

40  50  60  70  80  ....

Query: 55 ≤ r

A naïve solution is to return 50. By proving that 50 is chained to 60, we know that no answer has been dropped. But, this reveals the value of 50.

How about this …

Server:  
\[
g(r_{a-1}) \quad r_a \quad r_{a+1}
\]

User:  
\[
g(r_a) \quad g(r_{a+1})
\]

ver(H_a, sig(r_a), PK)?

Query: \( \alpha \leq r \)
The basic idea fails …

Server: 50 60 g(70) 80 90

User:

- g(ra)
- g(ra+1)

ver(Ha, sig(ra), PK)?

User can’t detect!

Query: 55 \leq r

Private Boundary Proof Ensures Origin

Server:

- \( h^{a_{r_{a-1}}^{-1}}(r_{a-1}) \)
- r_a
- r_{a+1}

User:

- ??
- g(r_{a-1})
- g(r_a)
- g(r_{a+1})

ver(Ha, sig(r_a), PK)?

- \( h^i(r) = h^{i-1}(h(r)) \)
- \( g(r) = h^{U_{r-1}}(r) \)

Query: \( \alpha \leq r \)
Private Boundary Proof Ensures Origin

Server: \[ h^{a-r_{a-1}}(r_{a-1}) \]
User: \[ g(r_{a-1}) \]
User: \[ g(r_a) \]
User: \[ g(r_{a+1}) \]

A collaborative scheme to compute the hash value

Query: \( \alpha \leq r \)

Back to our example

Server: \[ h^{a-70}(70) \]
User: \[ 80 \]
User: \[ 90 \]

Cheat!

Require that the inverse of \( h^i \) for \( i < 0 \) be undefined

User: \[ g(r_a) \]
User: \[ g(r_{a+1}) \]

Wrong! Undefined!

User detects cheating

Query: \( 55 \leq r \)
Back to our example

Server: \( h^{55-50} (50) \), 60, 70, ...

User: hash U - 55 times

\( g(50), g(60), g(70) \)

ver(H_{60}, sig(60), PK)?

Query: \( 55 \leq r \)

Putting the Pieces Together

Distributor: \( h^{a-r_{a+1}} (r_{a-1}) \), \( r_a, r_{a+1}, \ldots, r_n \), \( g(r_{n+1}) \)

User: hash U - \( a \) times

\( g(r_{a-1}), g(r_a), g(r_{a+1}), \ldots, g(r_n) \)

ver(H_a, sig(r_a), PK)?, ver(H_{a+1}, sig(r_{a+1}), PK)?

Query: \( a \leq r \)
Other cases

- $\alpha \leq r$
- $\beta \geq r$ (Result = \{r\alpha, r\alpha+1, ..., r\beta\}, i.e., $r\alpha, ..., r\beta \leq \beta < r_{b+1}$
  - Need to verify that $r_{b+1} > \beta$
  - Define $g(r) = h^{r_{b+1} - 1}(r) = h^{\beta - 1}(h^{r_{b+1} - 1}(r))$ where $L$ is a value outside of the minimum value of the domain
- So, we have $\alpha \leq r \leq \beta$
- $r = \alpha \equiv \alpha \leq r \leq \alpha$
- $\alpha < r < \beta \equiv \alpha + 1 \leq r \leq \beta - 1$
- $\alpha \neq r \equiv (L < r < \alpha) \cup (\alpha < r < R)$

NULL Answers??

- Consider Q: $\alpha \leq r$.
- $Q = \emptyset$ because $r_n < \alpha$.
  - Server returns $h^{\alpha - r_{n+1}}(r), g(r_{n+1}), \text{sig}(r_{n+1})$
  - User computes $h^{\alpha - r_{i+1}}(r)$ and verifies $\text{ver}(H_{n+1}, \text{sig}(r_{n+1}), PK)$?
- How about $r_i < \alpha \leq \beta < r_{i+1}$?
One More Vulnerability

- User can discover $r_{a-1}$ through brute force enumeration of numbers below $r_a$
- Solution:
  - Record $[K, A_1, .., A_m]$, $K =$ ordering attribute
  - $g(r_i.K | r_i.A_1 | ... | r_i.A_m)$
  - Brute-force attack is no longer feasible

Completeness Verification for Range Queries

Verify $\alpha < r_{a-1}.K$

Record $r_{a-1}: [K, A_1, A_2, ..., A_m]$  

Query: $\alpha \leq K \leq \beta$. Result: $\{r_\alpha, r_{\alpha+1}, ..., r_\beta\}$
Other queries

- SP Query
  - Based on MHT(r.A)
  - Ordering attribute has to be returned (even if it is not part of the target attributes). Why?
  - For attributes that are filtered out, digests may need to be returned

- SPJ Query
  - R.Ai = S.Aj (Ai is foreign-key in R, Aj is primary key in S)
    - Referential integrity constraint mandates that every instance of R.Ai must have a matching entry in S.Aj
    - So, only need to deal with selection conditions on R.Ai or S.Aj
    - Create a signature chain for R.Ai

What else?

- What about data freshness?
- More efficient scheme
- Ad-hoc joins
- Aggregates
- Multi-dimensional data
- Computation
- Complete (complex) queries
Summary

- Malicious service provider may cheat
- Users need assurance on their query answers
- Merkle hash tree offers a good solution but ...
- Signature chain guarantee completeness without violating access control policy