Privacy, Authenticity and Integrity in Outsourced Databases

Gene Tsudik
Computer Science Department
School of Information & Computer Science
University of California, Irvine

gts@ics.uci.edu
http://sconce.ics.uci.edu
Software-as-a-Service

- Popular trend
- Get
  - what you need
  - when you need it
- Pay for
  - what you use
- Don’t worry about:
  - Deployment, installation, maintenance, upgrades
  - Hire/train/retain people
Software-as-a-Service: Why?

• Advantages
  – reduced cost to client
    • 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 across organizations

• Driving Forces
  – Faster, cheaper, more accessible networks
  – Virtualization in server and storage technologies
  – Established e-business infrastructures

• Already in Market
  – Horizontal storage services, disaster recovery services, e-mail services, rent-a-spreadsheet services etc.
  – Sun ONE, Oracle Online Services, Microsoft .NET My Services, etc

Better Service → Cheaper
Emerging Trend: Database-as-a-Service

Most Significant DB Execution Problems

- Ease of Administration: 58%
- Qualified Administrators: 57%
- Compatibility: 51%
- Qualified Programmers: 51%
- Platform Independence: 40%

% of respondents (Source: InfoWeek Research)

• Why?
  – Most organizations need DBMSs
  – DBMSs extremely complex to deploy, setup, maintain
  – require skilled DBAs (at very high cost!)
The DAS Project

**Goal:** Security for Database-as-a-Service model

**People:** Sharad Mehrotra, Gene Tsudik, Ravi Jammala, Maithili Narasimha, Bijit Hore, Einar Mykletun, Yonghua Wu

**Supported in part by NSF ITR grant “Security & Privacy in Database as a Service”**
Rough Outline

- What we want to do
- Design space
- Challenges
- Architecture
- Bucketization = DB Partitioning
- Integrity & Authenticity
- Aggregated signatures
- Hash trees
- Related work
What do we want to do?

- **Database as a Service (DAS) Model**
  - DB management transferred to service provider for
    - backup, administration, restoration, space management, upgrades etc.
  - use the database “as a service” provided by an ASP
    - use SW, HW, human resources of ASP, instead of your own
DAS variables

- Database types
- Interaction dynamics
- Trust in Server
Database Types in the DAS Model:

- Warehousing (write once, read many)
- Archival (append only, read many)
- Dynamic (read/write)
1. Unified Owner Scenario

**BTW:**
- Querier may be weak (battery, CPU, storage)
- Querier might have a slow/unreliable link
- Data “deposit” is << frequent than querying

Data Deposit + Queries

Owner/Querier

Server Site

Server

Encrypted User Database
2. Multi-Querier Scenario

Owner/Querier

Server Site

Server

Encrypted User Database

Data Deposit & queries

Data Queries

Querier 1

Querier 2

Querier 3
3. Multi-Owner Scenario

Owner 1

Owner 2

Owner 3

Server Site

Server

Encrypted User Database

Data Queries

Data Deposit & queries

Querier 1

Querier 2
Challenges

• Economic/business model?
  – How to charge for service, what kind of service guarantees can be offered, costing of guarantees, liability of service provider.

• Powerful interfaces to support complete application development environment
  – User Interface for SQL, support for embedded SQL programming, support for user defined interfaces, etc.

• Scalability in the web environment
  – Overhead costs due to network latency (data proxies?)

• Privacy / Security
  - Protection of outsourced data from intruders and attacks
  - Protecting clients from misuse of data by service providers
  - Ensuring integrity+authenticity+completeness of query replies
  - Protecting service providers from “litigious” clients
We **do not fully trust** the service provider with sensitive information!
What do we mean by: “do not fully trust”? 

Degrees of mistrust in Server:

1. **More-or-less trusted**: outsider attacks only (e.g., on communication)
   - Encrypt data in transit, apply usual security measures

2. **Partially trusted**: break-ins, attacks on storage only

3. **Untrusted**: server can be subverted or be(come) malicious
Partially trusted server

Break-ins, attacks on storage

- Storage may be de-coupled from CPU
- Encrypt data “in situ”, keep keys elsewhere
- Where: in CPU, in secure HW (tamper-resistant, or token-style), at user side, etc.
Secure and Efficient RDBMS Storage Model

• Need to reduce overhead associated with encryption
  – Today’s storage models don’t lend themselves to efficient encryption solutions

• Server is partially trusted
  – Data encrypted on disk, unencrypted in memory

• We developed a new RDBMS storage model to:
  – Reduce number of encryption calls (start-up cost dominates)
  – Reduce padding overhead: database attributes can be especially sensitive
    • 16 byte blocks: 2 byte attribute requires 14 bytes padding (w/AES)
  – Avoid over-encrypting: queries on non-sensitive data should run with minimal overhead
Secure and Efficient RDBMS Storage Model

- **Start-up Cost**
  - Includes creating key schedule
  - Start-up cost incurred for each encryption operation
  - Fine encryption granularity results in many encryption operations

<table>
<thead>
<tr>
<th>Encryption Algorithm</th>
<th>100 Byte * 100,000</th>
<th>120 Byte * 83,333</th>
<th>16 Kbytes * 625</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>365</td>
<td>334</td>
<td>194</td>
</tr>
<tr>
<td>DES</td>
<td>372</td>
<td>354</td>
<td>229</td>
</tr>
<tr>
<td>Blowfish</td>
<td>5280</td>
<td>4409</td>
<td>170</td>
</tr>
</tbody>
</table>

Encryption of 10 Mbytes - all times in Msec

Fewer “large” encryptions better than many “small”
N-ary Storage Model (used today)

Records stored sequentially
- How do distinguish sensitive from non-sensitive attributes?
- Attribute level encryption (padding, cost)
PPC – Partition Plaintext Ciphertext Model (EDBT’04)

- Fewer “large” encryptions better than many “small”
- Create homogeneous mini-pages
- Distinguish sensitive from non-sensitive data
  - Maximum one encryption operation per page
  - Padding per mini-page (versus attribute / record)
  - No overhead when querying non-sensitive data

<table>
<thead>
<tr>
<th>Offset</th>
<th>Ciphertext</th>
<th>minipage (name,salary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike</td>
<td>8K</td>
<td></td>
</tr>
<tr>
<td>John</td>
<td>10K</td>
<td></td>
</tr>
<tr>
<td>Tom</td>
<td>6K</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Plaintext minipage (empNo,dept)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toys</td>
<td></td>
</tr>
<tr>
<td>Sales</td>
<td></td>
</tr>
</tbody>
</table>

Page Header

1  Toys                  2  Sales                  3  Toys

- PPC
Untrusted server

Cannot trust server with database contents
Rough Goals

- **Encrypt** client’s data and store at server
- Client:
  - runs queries over encrypted remote data
  - verifies integrity/authenticity of results
- **Most of the work** to be done by the server
System Architecture (ICDE'02)

Client Site

- Result Filter
- Temporary Results
- Client Side Query
- Query Translator
- Original Query
- Metadata
- Actual Results

Server Site

- Server Side Query
- Encrypted Results
- Service Provider
- Encrypted User
- Database

User
Query Processing 101...

- At its core, query processing consists of:
  - Logical comparisons (>, <, =, <=, >=)
  - Pattern based queries (e.g., *Arnold*egger*)
  - Simple arithmetic (+, *, /, ^, log)

- Higher level operators implemented using the above
  - Joins
  - Selections
  - Unions
  - Set difference
  - ...

- To support any of the above over encrypted data, need to have mechanisms to support basic operations over encrypted data
Fundamental Observation...

• Basic operations do not need to be fully implemented over encrypted data

• To test (AGE > 40), it might suffice to devise a strategy that allows the test to succeed in most cases (might not work in all cases)

• If test does not result in a clear positive or negative over encrypted representation, resolve later at client-side, after decryption.
Relational Encryption

Store an encrypted string – *etuple* – for each tuple in the original table

- This is called “row level encryption”
- Any kind of encryption technique can be used

Create an index for each (or selected) attribute(s) in the original table
Building the Index:

- **Partition function** divides domain values into partitions (buckets)

  \[ \text{Partition (} R.A \text{)} = \{ [0,200], (200,400], (400,600], (600,800], (800,1000] \} \]

  - partition function has impact on performance as well as privacy
  - very much domain/attribute dependent
  - equi-width vs. equi-depth partitioning?

- **Identification function** assigns a partition id to each partition of attribute \( A \)

  \[ \text{ident}_{R.A}( (200,400] ) = 7 \]

  - Any function can be use as identification function, e.g., hash functions
  - Client keeps partition and identification functions secret (as **metadata**)

---

Domain Values

Partition (Bucket) ids

\[ \begin{array}{c|c|c|c|c|c|c|c}
  0 & 200 & 400 & 600 & 800 & 1000 \\
\end{array} \]

\[ \begin{array}{c|c|c|c|c|c|c|c}
  0 & 2 & 7 & 5 & 1 & 4 \\
\end{array} \]
Bucketization / Partitioning / Indexing

- Primitive form of encryption, sort of a “substitution/permutation cipher”
- Can be viewed as partial encryption
- Leaks information about plaintext!!!
- Works fine with warehoused data but needs to be periodically re-done with highly dynamic data
- Attacks (assume domain known)
  - Ciphertext only
  - “Existential” plaintext
  - Known plaintext
  - Chosen plaintext
  - Adaptive chosen plaintext
Mapping Functions (SIGMOD’02)

- Mapping function maps a value $v$ in the domain of attribute $A$ to partition id

  - e.g., $Map_{R,A}(250) = 7$  $Map_{R,A}(620) = 1$
Storing Encrypted Data

\[ R = < A, B, C > \quad \Rightarrow \quad R^S = < \text{etuple}, A\_id, B\_id, C\_id > \]

\text{etuple} = encrypt ( A \mid B \mid C )

\[ A\_id = Map_{R,A}( A ), \quad B\_id = Map_{R,B}( B ), \quad C\_id = Map_{R,C}( C ) \]

### Table: EMPLOYEE

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>50000</td>
<td>2</td>
</tr>
<tr>
<td>Mary</td>
<td>110000</td>
<td>2</td>
</tr>
<tr>
<td>James</td>
<td>95000</td>
<td>3</td>
</tr>
</tbody>
</table>

### Table: EMPLOYEE$^S$

<table>
<thead>
<tr>
<th>Etuple</th>
<th>N_ID</th>
<th>S_ID</th>
<th>P_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>fErf!$Q!!vddf&gt;&gt;|</td>
<td>50</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>F%%3w&amp;%gfErf!$</td>
<td>65</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>&amp;%gfdfs%343v&lt;l</td>
<td>50</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td>2</td>
<td>20</td>
</tr>
</tbody>
</table>
**Mapping Conditions**

Q: SELECT name, pname FROM employee, project
WHERE employee.pin=project.pin AND salary>100k

- Server stores attribute indices determined by mapping functions
- Client stores metadata and uses it to translate the query

Conditions:
- Condition ← Attribute \( op \) Value
- Condition ← Attribute \( op \) Attribute
- Condition ← (Condition \( \lor \) Condition) | (Condition \( \land \) Condition) | (not Condition)
Example: Equality

- Attribute = Value
  - $Map_{\text{cond}}( A = v ) \Rightarrow A^s = Map_A(v)$
  - $Map_{\text{cond}}( A = 250 ) \Rightarrow A^s = 7$
Example: Inequality ($<$, $>$, etc.)

- Attribute $<$ Value
  - $Map_{\text{cond}}(A < v) \Rightarrow A^S \in \{\text{ident}_A(p_j) \mid p_j.\text{low} \leq v\}$
  - $Map_{\text{cond}}(A < 250) \Rightarrow A^S \in \{2,7\}$

At client site
Mapping Conditions (4)

- Attribute1 = Attribute2 (useful for JOIN-type queries)
  - $\text{Map}_{\text{cond}}( A = B ) \Rightarrow \bigvee_N (A^s = \text{ident}_A(p_k) \land B^s = \text{ident}_B(p_1))$
    where $N$ is $p_k \in \text{partition}(A)$, $p_1 \in \text{partition}(B)$, $p_k \cap p_1 \neq \emptyset$

<table>
<thead>
<tr>
<th>Partitions</th>
<th>A_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,100]</td>
<td>2</td>
</tr>
<tr>
<td>(100,200]</td>
<td>4</td>
</tr>
<tr>
<td>(200,300]</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partitions</th>
<th>B_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,200]</td>
<td>9</td>
</tr>
<tr>
<td>(200,400]</td>
<td>8</td>
</tr>
</tbody>
</table>

$C : A = B \Rightarrow C' : (A_{id} = 2 \land B_{id} = 9)$

$\checkmark$ $(A_{id} = 4 \land B_{id} = 9)$

$\checkmark$ $(A_{id} = 3 \land B_{id} = 8)$
Relational Operators over Encrypted Relations

- Partition the computation of the operators across client and server
- Compute (possibly) superset of answers at the server
- Filter the answers at the client
- **Objective**: minimize the work at the client and process the answers as soon as they arrive requiring minimal storage at the client

Operators studied:
- Selection
- Join
  - Grouping and Aggregation (in progress)
- Sorting
- Duplicate Elimination
- Set Difference
- Union
- Projection
Research Challenges

- Aggregation queries, e.g., how to do: $\sum (a \times b + c)$
  - RSA can do $\times$
  - Paillier can do $+$
  - How to do both?

- Complex queries
  - Nested, Embedded, Stored procedures, Updates

- Query optimization

- Privacy guarantees
  - Against different types of attacks -- ciphertext only attack, known plaintext attack, chosen plaintext attack (work-in-progress)

- Generalized DAS models
  - What if there are more than a single owner and server?
  - Can the model work for storage grid environments?

- Key management policies
• Bucketization / Partitioning is problematic

• Supports mainly range-style queries

• Other query types hard to accommodate

• What if server has a tamper-resistant secure co-processor?
SC Example (IBM 4758)

- Physical security sensing and response
- 486 processor and PC support
- DRAM
- Battery-backed RAM
- Flash, ROM
- Hardware locks
- Modular math
- Random-number generator
- Real-time clock
- Physical security boundary
- PCI bus Interface
- Host PCI bus
Actual IBM 4758 Device
Secure co-processor in applications

- Acts as a trusted device in an untrusted environment

Encryption key

Client → Encryption key → Untrusted Server → Encryption key → DB

SC
1) Client query
   - Select where Salary < 20K

2) Client splits query (based on meta data)
   - Server Query ($Q_S$): Select where Salary ID = 2 or 3
   - SC Query ($Q_{SC}$): Select where Salary ID < 20K

3) Client sends queries to server

4) Server executes $Q_S$, sends superset and $Q_{SC}$ to SC

4) SC executes $Q_{SC}$, sends encrypted results to server

5) Server sends encrypted results to client

No communication overhead (server to client)
No post-processing of query results at client
Integrity and Authenticity in DAS

- Not all outsourced data needs to be encrypted
- Some data might be only partially encrypted
- At times, authenticity is more important, especially, in multi-querier and multi-owner scenarios
- This is different from query completeness, i.e., making sure that server returned all records matching the query

- Need to minimize overhead:
  1. Bandwidth, storage, computation overhead at querier
  2. Bandwidth, storage, computation overhead at server
  3. Bandwidth, storage, computation overhead at owner
Challenge: how to provide efficient authentication + integrity for a potentially large and unpredictable set of records returned?
Integrity and Authenticity in DAS

- What granularity of integrity: page, relation, attribute, record?
- What mechanism: MACs, signatures?
- Not a problem in unified owner scenario (use MACs)
- For others: record-level signatures, but what kind?
  - Boneh, et al. → aggregated multi-signer signatures
  - Batch RSA
  - Batch DSA or other DL-based signature schemes
  - Hash Trees and/or other authenticated data structures
**Batch Verification of RSA Signatures**

- **Batching:** useful when many signature verifications need to be performed simultaneously

- **Reduces computational overhead**
  - By reducing the total number of modular exponentiations

- **Fast screening of RSA signatures (Bellare et al.):**
  - Given a batch instance of signatures \( \{\sigma_1, \sigma_2 \ldots \sigma_t\} \) on distinct messages \( \{m_1, m_2 \ldots m_t\} \)

\[
\left( \prod_{i=1}^{t} \sigma_i \right)^e \equiv \prod_{i=1}^{t} h(m_i) \pmod{n}
\]

where \( h() \) is a full domain hash function
**Fast Screening**

- Reduces (somewhat) querier computation but **not** bandwidth overhead
  - Individual signatures are sent to the querier for verification

- Bandwidth overhead can be overwhelming
  - Consider weak (anemic) queriers
  - Query reply can have thousands of records
  - Each RSA signature is at least 1024 bits!

*Can we do better?*
Condensed RSA (NDSS’04)

- **Server:**
  - Selects records matching posed query
  - Multiplies corresponding RSA signatures
  - Returns *single* signature to querier

---

**Server**

- Given $t$ record signatures:
  $$\{\sigma_1, \sigma_2 \ldots \sigma_t\},$$
- Compute combined signature
  $$\sigma_{1,t} = \prod \sigma_i \mod n$$
- Send $\sigma_{1,t}$ to the querier

**Querier**

- Given $t$ messages:
  $$\{m_1, m_2 \ldots m_t\}$$ and $\sigma_{1,t}$
- Verify combined signature:
  $$\left(\sigma_{1,t}\right)^e = ? = \prod h(m_i) \mod n$$
Condensed RSA

• Reduced querier computation costs
  – Querier performs \((t-1)\) mult-s and a \textbf{one} exponentiation
• Constant bandwidth overhead
  – Querier receives a single RSA signature
• As secure as batch RSA (with FDH)

However, still can’t aggregate signatures by different signers!

(NOTE: RSA modulus cannot be shared)

Condensed RSA \(\Rightarrow\) efficient for Unified-owner and Multi-querier but \textbf{NOT} great for Multi-owner
Batching DL-based signatures

• DL-based signatures (e.g., DSA) are efficient to generate
• Batch verification possible
• Unlike RSA, different signers can share the system parameters
  ➔ useful in the Multi-Owner Model?

Unfortunately, no secure way to aggregate DL-based signatures!
DL-based signatures...(cont’d)

• All current methods for batch verification of DL-based signatures require “small-exponent test”

• Involves verifier performing a mod exp (with a small exponent) on each signature before batching the verification.
  – Without this, adversary can create a batch instance which satisfies verification test without possessing valid individual signatures

• Thus, individual signatures are needed for verification
  ➔ aggregation seems impossible.
1. Condensed RSA
   - Cannot combine signatures by multiple signers
   - Querier computation, bandwidth overhead linear in # of signers

2. Batch DSA (and variants)
   - Can batch-verify signatures by distinct users and but cannot aggregate or condense
   - Querier computation as well as bandwidth overhead linear in # of signatures (records)!
Aggregated signatures (Boneh, et al.)

- Signatures on different messages by multiple signers can be combined into one small signature.
- Scheme requires bilinear map (in Gap DH groups)
- BGLS Details:

Key Generation:
- pick a random \( x \in \mathbb{Z}_p \) and compute \( v = g^x \)
  - \( v \) - public key, \( x \) - secret key.

Signing:
- let \( h = h(m) \) -- hash of message
  - \( \sigma = h^x \)

Aggregation:
- To aggregate \( t \) signatures, compute their product

Verification:
- Compute the product of the hashes and verify
  - where \( e() \) is a computable bilinear mapping

\[
e(\sigma_{1,t}, g) = \prod_{i=1}^{t} e(h_i, v_i)
\]

\[
e(\sigma_{1,t}, g) = e(\prod_{i=1}^{t} (h_i^{x_i}, g)) = \prod_{i=1}^{t} e(h_i, g^{x_i}) = \prod_{i=1}^{t} e(h_i, v_i)
\]
Aggregated signatures (Boneh, et al.)

- Applicable to all DAS flavors
- Constant bandwidth overhead
- For Unified-owner and Multi-querier, querier verification costs \((t-1)\) EC mults (where \(t\) is \# returned records) and two bilinear mappings
- For Multi-owner, verification of aggregated signature costs \((k+1)\) bilinear mappings (where \(k\) is \# signers) and \((t-k)\) multiplications
  - Bilinear mappings are expensive
  - Computing a single mapping in \(F_p\) (for \(|p|=512\)) on a 1GHz PIII takes 31 msecs!
  - One mapping equivalent to 8-10 exponentiations
## Cost Comparisons

### 1. Querier computation:

(P3-977MHz, Time in mSec)

<table>
<thead>
<tr>
<th></th>
<th>Condensed RSA</th>
<th>Batch DSA</th>
<th>BGLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign 1 signature</td>
<td>6.82</td>
<td>3.82</td>
<td><strong>3.54</strong></td>
</tr>
<tr>
<td>Verify 1 signature</td>
<td><strong>0.16</strong></td>
<td>8.52</td>
<td>62</td>
</tr>
<tr>
<td>t = 1000 sigs, k = 1 signer</td>
<td>44.12</td>
<td>1623.59</td>
<td>184.88</td>
</tr>
<tr>
<td>t = 100 sigs, k = 10 signers</td>
<td>45.16</td>
<td>1655.86</td>
<td>463.88</td>
</tr>
<tr>
<td>t = 1000 sigs, k = 10 signers</td>
<td>441.1</td>
<td>16203.5</td>
<td>1570.8</td>
</tr>
</tbody>
</table>

Parameters:
- For RSA: $|n| = 1024$
- For DSA: $|p| = 1024$ and $|q| = 160$
- For BGLS: Field $F_p$ with $|p| = 512$
2. Bandwidth overhead:

<table>
<thead>
<tr>
<th></th>
<th>Condensed RSA</th>
<th>Batch DSA</th>
<th>BGLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 signature</td>
<td>1024</td>
<td>1184</td>
<td>512</td>
</tr>
<tr>
<td>t =1000 sigs, k=1 signer</td>
<td>1024</td>
<td>1184000</td>
<td>512</td>
</tr>
<tr>
<td>t =100 sigs, k=10 signers</td>
<td>10240</td>
<td>1184000</td>
<td>512</td>
</tr>
<tr>
<td>t =1000 sigs, k = 10 signers</td>
<td>10240</td>
<td>11840000</td>
<td>512</td>
</tr>
</tbody>
</table>

(单位：位)

3. Server overhead (less important):

- Batch DSA: none
- BGLS: t mult-s
- Condensed RSA: t mult-s
Merkle Hash Tree (MHT)

- Authenticate a sequence of data values $D_0, D_1, \ldots, D_N$
- Construct binary tree over data values

Diagram:

```
  T0
 /   \
T1   T2
|     |
T3   T4
|     |
D0   D1
|     |
D2   D3
|     |
D4   D5
|     |
D6   D7
```
MHT contd.

- Verifier knows $T_0$
- How can verifier authenticate leaf $D_i$?
- Solution: re-compute $T_0$ using $D_i$
- Example authenticate $D_2$, send: $D_3, T_3, T_2$
- Verify $T_0 = H( H( T_3 \| H( D_2 \| D_3 )) \| T_2 )$
MHT Example -- Certificate Revocation Tree

Leaf nodes sorted by certificate serial number

h7 = h(h5, h6)

h5 = h(h1, h2)

h2 = h(h(cert3), h(cert4))

h1 = h(h(cert1), h(cert2))

cert1  cert2  cert3  cert4

h6 = h(h3, h4)

h3 = h(h(cert5), h(cert6))

cert5  cert6  cert7  cert8

h4 = h(h(cert7), h(cert8))

Signed by CA
• Can use MHTs with leaf nodes representing records and the root node signed by the owner
  – Authentic 3rd party publishing
  – Prior work by Martel, Stubblebine, Devanbu, et al.

• For Multi-owner scenario:
  – Individual trees for each owner OR
  – A single tree with a shared signing key among all owners
  – Mixed tree
As a response to a posed query, server

1. Selects records that match query predicate
2. Sends records along with hashes on co-paths for each record.
3. Attach a single signature corresponding to root of hash tree

Upon receiving query reply, querier

1. Computes hashes of all records returned
2. Using hashes of nodes on co-paths, computes hashes for each path to the root
3. Verifies signature of root node
MHT Overhead

• For n leaf nodes and t records in the query reply
  
  – Lower server-storage overhead compared to per-record signatures
    • At most: \( (2n-1)\times |\text{hash}| + |\text{sig}| \) as opposed to \( n\times |\text{sig}| \)
  
  – Record insertion (owner computation overhead) requires 2 extra rounds of communication
    • to make structural changes to the tree
  
  – Querier computation cost lower since verification involves computing hashes
    • Compared with Combined RSA which involves mod mults...
  
  – However, bandwidth overhead increases!
    • Hashes for all nodes on co-paths must be supplied
Bandwidth overhead

• Expected overhead
  – For \( n \) leaf nodes and \( t \) records in query reply
  – Let \( n=2^h \) and wlog, let \( P(\text{a leaf node is selected}) = \frac{t}{n} \)
  – Expected # of additional hashes (non-leaf nodes) returned is given by:

\[
\sum_{k=0}^{h-1} 2^{h-k} \left( 1 - \left( 1 - \frac{t}{n} \right)^{2^k} \right) \left( 1 - \frac{t}{n} \right)^{2^k}
\]

... 

e.g., if \( h=30 \), \( t=1024 \), and \(|\text{hash}| = 160\) then,
Bandwidth overhead = 3,132,000 bits
(for condensed RSA \( \rightarrow 1,024 \) bits)
In conclusion...

- MHTs: good for computation, bad for bw and dynamic databases
  - Can be used to guarantee query completeness (for range queries)
  - Needs a sorted MHT for each attribute

- Aggregation/Condensation: good for bw; saves some computation;
  - How to filter “bad” signatures?

- Currently investigating hybrid model
  - Use MHTs along with record-level signatures.
  - Determine which is cheaper on a per-query basis

- Is it possible to aggregate/condense DSA-like signatures?

- Is it possible to aggregate multi-signer RSA? Perhaps…

- Any new efficient and practical signature scheme that allows multi-signer aggregation?

- How to prevent mutability in aggregated/condensed signatures?
What is “Query Completeness”

- Assurance that query reply contains ALL records matching query predicate(s)
- Example: MHT with leaves sorted along “Age” attribute
- Query: AGE>8 and AGE<26
- Minimal overhead: include sentinel leaves on both sides

- Same is possible but harder to achieve with record-level signatures…
Related Work

• Authentic 3\textsuperscript{rd} party publishing
• Private information retrieval (PIR)
• Searching encrypted data for keywords
  – Boneh, et al.
  – Song, et al.
• Encrypted aggregation
  – Privacy Homomorphisms (Rivest, et al.)
• Watermarking databases
  – Attallah, et al.
• Privacy-preserving data mining
  – Agrawal, et al.
• Batch signature verification (RSA, DSA, etc.)
Some project-related references

1. Hakan Hacigumus, Bala Iyer, Chen Li and Sharad Mehrotra
   Executing SQL over Encrypted Data in the Database-Service-Provider Model
   SIGMOD 2002

2. Hakan Hacigumus, Bala Iyer and Sharad Mehrotra
   Providing Database as a Service
   ICDE 2002

3. Maithili Narasimha, Einar Mykletun and Gene Tsudik
   Efficient Data Integrity in Outsourced Databases
   NDSS 2004

4. Bala Iyer, Sharad Mehrotra, Einar Mykletun, Gene Tsudik and Yonghua Wu
   A Framework for Efficient Storage Security in RDBMS
   EDBT 2004

5. Bijit Hore, Sharad Mehrotra and Gene Tsudik
   A Privacy-Preserving Index for Range Queries
   VLDB 2004

6. Maithili Narasimha, Einar Mykletun and Gene Tsudik
   Signature Bouquets: Immutability for Aggregated Signatures
   ESORICS 2004
Thank you!

Questions?
Selection Operator

\[ \sigma_c(R) = \sigma_c(D \left( \sigma_{Mapcond(c)}(R^s) \right) ) \]

Example:

\[ \sigma_{A=250}(R) \]

\[ \sigma_{A_id = 7}(E_TABLE) \]

Client Query

Server Query

<table>
<thead>
<tr>
<th>0</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
<th>1000</th>
</tr>
</thead>
</table>
Join Operator

\[ R \bowtie_c T = \sigma_c ( D ( R^S \bowtie_{\text{Mapcond}(c)} T^S ) ) \]

Example:

\[ C : A = B \Rightarrow C' : (A_{id} = 2 \land B_{id} = 9) \]
\[ \lor (A_{id} = 4 \land B_{id} = 9) \]
\[ \lor (A_{id} = 3 \land B_{id} = 8) \]

<table>
<thead>
<tr>
<th>Partitions</th>
<th>A_{id}</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,100]</td>
<td>2</td>
</tr>
<tr>
<td>(100,200]</td>
<td>4</td>
</tr>
<tr>
<td>(200,300]</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partitions</th>
<th>B_{id}</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,200]</td>
<td>9</td>
</tr>
<tr>
<td>(200,400]</td>
<td>8</td>
</tr>
</tbody>
</table>
Query Decomposition

Q: SELECT name, pname FROM EMPLOYEE, PROJECT WHERE EMPLOYEE.pid=PROJECT.pid AND salary > 100k

Client Query

\[ \pi_{name,pname} \]
\[ \sigma_{salary > 100k} \]
\[ EMPLOYEE \]
\[ e.pid = p.pid \]

Server Query

\[ \pi_{name,pname} \]
\[ \sigma_{salary > 100k} \]
\[ PROJECT \]
\[ e.pid = p.pid \]

Encrypted (EMP)

Encrypted (PROJ)

D
Query Decomposition (2)

Client Query

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{salary > 100k}} \)

\( \Join_{\text{e.pid = p.pid}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{salary > 100k}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{s_id = 1 v s_id = 2}} \)

Server Query

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{salary > 100k}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{s_id = 1 v s_id = 2}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{salary > 100k}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{s_id = 1 v s_id = 2}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{salary > 100k}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{s_id = 1 v s_id = 2}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{salary > 100k}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{s_id = 1 v s_id = 2}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{salary > 100k}} \)

\( \pi_{\text{name, pname}} \)

\( \sigma_{\text{s_id = 1 v s_id = 2}} \)
Query Decomposition (3)

Client Query

\[ \pi_{\text{name}, \text{pname}} (e.p_id = p.p_id) \]

\[ \sigma_{\text{salary} > 100k} \]

\[ \sigma_{s_id = 1 \lor s_id = 2} \]

Server Query

\[ \pi_{\text{name}, \text{pname}} (e.p_id = p.p_id) \]

\[ \sigma_{\text{salary} > 100k} \]

\[ \sigma_{s_id = 1 \lor s_id = 2} \]
Query Decomposition (4)

Client Query:
\[
\pi_{\text{name}, \text{pname}} \\
\sigma_{\text{salary} > 100k \land \text{e.pid} = \text{p.pid}} \\
\]

Server Query:
\[
\sigma_{\text{s_id} = 1 \lor \text{s_id} = 2} \\
\]

Q:
SELECT name, pname FROM EMPLOYEE, PROJECT WHERE EMPLOYEE.pid=PROJECT.pid AND salary > 100k

Q\text{\textsuperscript{S}}:
SELECT e_emp.etuple, e_proj.etuple FROM e_emp, e_proj WHERE e.p_id=p.p_id AND s_id = 1 OR s_id = 2

Q\text{\textsuperscript{C}}:
SELECT name, pname FROM temp WHERE emp.pid=proj.pid AND salary > 100k

π name, pname
σ salary > 100k ∧ e.pid = p.pid
D
σ s_id = 1 ∨ s_id = 2
E_PROJ
σ e.p_id = p.p_id
E_EMP
π name, pname
σ e.p_id = p.p_id
AND salary > 100k

π name, pname
σ s_id = 1 ∨ s_id = 2
D
σ e.p_id = p.p_id
E_PROJ
π name, pname
σ e.p_id = p.p_id
AND salary > 100k

π name, pname
σ e.p_id = p.p_id
AND salary > 100k