Secure Data Management at UW

Gerome Miklau and Dan Suciu

Trustworthy computing seminar
Autumn 2004

Ingredients for data sharing

- File formats and tools
  XML, XML query languages
- Distributed processing
  Networked data sources
  Mediator systems, distributed systems
- Access control
  Controlled distribution of data

Today’s talk

1. Controlled data publishing
   Secrecy: crypto + XML
   [VLDB 2003]

2. Analyzing information disclosure
   Secrecy: theory + relations
   [SIGMOD 2004]
   [ICDT 2005]

3. Tamper-resistant databases
   Integrity: crypto + relations
   [Current work]

XML data and access rights

Lab Technician
Physician
Administrator

<hospital>
<lab>
<immunology> <psychiatry> <cardiology>
<patients>
<patient>
<name> <room> <age> <pat_id> ...
<patients>
<billing> <insurance> <xray> <blood>
Joe Smith
1123 56 402
cancer
p65345
Publishing Many Views

XML data source

3 versions published

Admin  Phys  Lab

Publishing one view + keys

XML data source

Single, partially-encrypted version

3 sets of keys transmitted

\{k1,k2\} \{k2,k3\} \{k1\}

Admin  Phys  Lab

Publishing protected data

FOR $x$ in /doc/subj/subj
               $y$ in /doc/psy/phys
WHERE ...
KEY getKey($y$)
TARGET $x$

• XML data tree guarded with formulas over keys.
• Node is accessible if "satisfying" keys are supplied.

Tree Protection

- XML data tree guarded with formulas over keys.
- Node is accessible if "satisfying" keys are supplied.
Access Function

Given a set of keys $K$, \texttt{access}( \textit{K} ) computes the accessible nodes in a tree protection.

\texttt{access}( \{ k_1 \} ) is:

\begin{align*}
k_1 & \\
\text{True} & \\
k_3 & \quad k_4 \\
\text{(k_1 \triangleright k_3)} & \quad \triangleright k_4
\end{align*}

\textit{hosp}

\textit{nurse}

\textit{phys}

\textit{pat_id}

\textit{admin}

\textit{pat_id}

"k_3"

Access function can discover data values and use them as keys.

Example: \texttt{access}( \{ k_1, k_2 \} ) is ...

\begin{align*}
k_1 & \\
\text{True} & \\
k_3 & \quad k_4 \\
\text{(k_1 \triangleright k_3)} & \quad \triangleright k_4
\end{align*}

\textit{hosp}

\textit{nurse}

\textit{phys}

\textit{pat_id}

\textit{admin}

\textit{pat_id}

"k_3"

Normalization Rules

- a tree protection is normalized if every formula is atomic
- soundness: access function invariant under rewritings

\textit{Benaloh and Leichter. CRYPTO 1988}

Rewrite rules can be used to optimize formula expressions.
Implementing a Protection

...<EncryptedData>
    <EncryptionMethod Alg="AES" KeySize="128"/>
    <KeyInfo>
     ...    </CipherData>
</EncryptedData>
...
Encrypted leaf node, following XML Encryption Recommendation

Security

Security Property
given document D, set of keys K:
✓ if node x ∈ access( K ) then x is efficiently computable from D.
☐ if node x ∈ access( K ) then the best algorithm for finding x requires guessing keys.

Additional disclosure:
number of children of a node
size of ciphertext, duplicate subtrees,
policy information.

Today’s talk

1. Controlled data publishing
   secrecy: crypto + XML

2. Analyzing information disclosure
   secrecy: theory + relations

3. Tamper-resistant databases
   integrity: crypto + relations

Protecting data using views

confidential database D
View V1(D) Alice
View V2(D) Bob
View V3(D) Carol
Problem statement

Query - View security

- a published view V
- a sensitive query S
- Does V reveal anything about S?

Spectrum of disclosure

<table>
<thead>
<tr>
<th>Published View</th>
<th>Sensitive Query</th>
<th>Disclosure?</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1(name, phone)</td>
<td>S1(name)</td>
<td>total</td>
</tr>
<tr>
<td>V2(name, dept)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V3(name)</td>
<td>S3(phone)</td>
<td></td>
</tr>
<tr>
<td>V4(name, phone)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S4(name, phone)</td>
<td></td>
</tr>
</tbody>
</table>

Existing techniques

- Logical inference
- Answering queries using views
- Statistical databases
  - exact disclosure
  - aggregation of numerical attributes

Our basic strategy

Compare Mallory's knowledge about sensitive query S:

Knowledge about S, \( a \ priori \) = Knowledge about S, given V

When these are equal, V provides no information about S
The adversary’s knowledge

Knowledge about S, 

\[ ? = \text{Knowledge about S,} \]

\[ \text{a priori} \]

\[ \text{given V} \]

- Schema
- Domain
- Probability of each database

\[ \text{Probability of each answer to S} = \text{Probability of each answer to S} \]

Probabilities of databases

- Fix schema R, and domain.
- For each tuple t: \( 0 \leq \Pr[t] \leq 1 \)
- The probability of a database instance I is:

\[
\Pr[I] = \prod_{t \in I} \Pr[t] \times \prod_{t \notin I} (1 - \Pr[t])
\]

assumption: tuple-independence

Definition: query-view security

S and V are secure (denoted S|V ) if:

\[
\Pr[S(I)=x] = \Pr[S(I)=x | V(I)=y] \forall x \forall y
\]

- independence of probabilistic events
- inspired by Shannon’s perfect secrecy [1949]

Concrete example

- relation Edge(X,Y)
- nodes=\{a,b\}
- tuple probability = 1/2
- possible graphs:

  1 2 3
  (a,a) (a,a) (a,b)

Published view:

\[
V(y) :- \text{Edge(x,y)}
\]

for V(D)=\{ (a) \},

\[
\Pr[S(I)=\{ (a) \} | V(I)=\{ (a) \}] = 3/16
\]

Sensitive query:

\[
S(x) :- \text{Edge(x,y)}
\]

for S(I)=\{ (a) \},

\[
\Pr[S(I)=\{ (a) \}] = 1/3
\]
Goal: logical criterion

Brute force → Logical criterion

For each answer to S, answer to V

Compare probabilities

depends on domain & probability distribution

Deciding query-view security

Theorem

Given query S and view V, deciding whether $S | V$ is $\Pi_2^P$-complete

when S and V secure, then they are secure:
- for any (sufficiently large) domain
- for any probability distribution

Other consequences

Supplemental knowledge
- security standard easily extended
- compare

Encrypted view
- no sensitive query is secure

Supplemental info: cardinality
- no sensitive query is secure

Measuring disclosure

When query-view security fails:

for some x,y:
$\Pr[S(I)=x] \neq \Pr[S(I)=x | V(I)=y]$

how do we evaluate the difference?
- magnitude, or closeness to 1?
- aggregate over many answers
- see paper for limited case.
Partial disclosure

• Intuition
  • domains are large
  • databases are small (and of known size)

• New probabilistic model
  • each tuple $t$ equally-likely...
  • $\text{prob}[t]$ s.t. database size constant

• Practical security
  • $\lim \Pr(S \mid V) = 0$ as domain $\rightarrow \infty$

[ Dalvi, Miklau, Suciu. ICDT 2005 (to appear) ]

Today’s talk

1. Controlled data publishing
   secrecy: crypto + XML

2. Analyzing information disclosure
   secrecy: theory + relations

3. Tamper-resistant databases
   integrity: crypto + relations

Integrity

Data integrity
  an assurance that data has not been modified by an unauthorized party.

Consistency
  an assurance that the items in a collection are “fresh”.

Query integrity
  an assurance that a query answer is correct and complete.

Tampering threats to DB systems

• DB access control vulnerabilities
  • specification failure
  • enforcement failure
  • subversion (e.g. sql injection, weak authentication)

• DB extensions - user defined functions

• general OS and network threats

• privileged parties: OS admin, DB admin

• service provider model
Data integrity with hashing

Client

- File F

  compute y = hash(F)

  send F to server

  retrieve F' from server

Verifiable query (1)

CLIENT: Give me the record for ‘peter’

SERVER: answer is t₅

  hash path is h₁₀₁, h₁₁, h₀

CLIENT: verify by computing

  new root hash h'ₑ

  check h'ₑ = hₑ

This proves correctness and consistency

Verifiable query (2)

CLIENT: Return all employees with age between 35 and 40

SERVER: answer is t₅. Also return t₆, t₇.

  hash path is h₀₁₀, h₀₀, h₁₁

CLIENT: verify by computing

  new root hash h'ₑ

  check h'ₑ = hₑ

This proves correctness, completeness, and consistency
Updating the database

CLIENT: Move Peter from department HR to MGMT
-- verify Peter's record --
CLIENT: compute new $h_{100}, h_{10}, h_{1}, h_{\epsilon}$
send to server

Implementing a tamper-resistant database

Client

\[ \text{Alice} \]
- define table
- insert
- query

Server

\[ \text{DBMS} \]
- define tables
- verified insert
- verified query

- Smart client, oblivious server
- Relational representation of hash tree
- Query definition
- Index selection

Cost of integrity

- Reasonable communication overhead
- Reasonable client computation
- Modest storage overhead
- Good scalability
- Throughput:

<table>
<thead>
<tr>
<th></th>
<th>integrity</th>
<th>no integrity</th>
<th>multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>2.0 ms</td>
<td>.4 ms</td>
<td>5.0</td>
</tr>
<tr>
<td>Range query</td>
<td>6.1 ms</td>
<td>1.3 ms</td>
<td>4.7</td>
</tr>
<tr>
<td>Insert</td>
<td>8.3 ms</td>
<td>.8 ms</td>
<td>10</td>
</tr>
</tbody>
</table>

Multiple clients

How do we manage integrity with multiple users?
Questions ?