Building a Tamper-evident Database System

Gerome Miklau and Dan Suciu

Industrial Affiliates
Autumn 2004
Database tampering

Financial transactions

<table>
<thead>
<tr>
<th>Counterparty</th>
<th>Type</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acme Inc.</td>
<td>Debit</td>
<td>$19.99</td>
</tr>
<tr>
<td>UW</td>
<td>Credit</td>
<td>$100.00</td>
</tr>
</tbody>
</table>

Voter registration

<table>
<thead>
<tr>
<th>Name</th>
<th>Party</th>
<th>Felon?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally</td>
<td>Democrat</td>
<td>N</td>
</tr>
<tr>
<td>Joe</td>
<td>Republican</td>
<td>Y</td>
</tr>
</tbody>
</table>

Medical records

<table>
<thead>
<tr>
<th>Name</th>
<th>Disease</th>
<th>Blood Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Cancer</td>
<td>A</td>
</tr>
<tr>
<td>Fred</td>
<td>Schizophrenia</td>
<td>B</td>
</tr>
</tbody>
</table>

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

• Authentication
  • I’m ‘Alice’. My password is ‘secret’.

• Access control
  • Alice can add tuples to VOTER
  • Alice cannot remove tuples from VOTER
Tampering threats to DB systems

• DB access control vulnerabilities
• DB extensions - user defined functions
• general OS and network threats
• privileged parties: OS admin, DB admin
Tamper-evident DB system

- **Client**
  - Alice
  - define table
  - insert
  - query
  - verified answer
  - translation

- **Server**
  - DBMS
  - define tables
  - verified insert
  - verified query
  - answer

- **Client uses modest trusted resources**
  - preprocess data
  - verify query results

- **Server is oblivious, but tampering detected**
Data integrity with hashing

Cryptographic hash function
\[ f(\text{byte-sequence}) = 160 \text{ bits} \]

- Efficient to compute
- Computationally infeasible to
  - find a collision: \( x, x' \) s.t. \( f(x) = f(x') \)
  - invert: given \( f(x) \), find \( x \)
- Examples: MD5, SHA1
Data integrity with hashing

**Client**

- File F

  compute \( y = \text{hash}(F) \)

**Server** (untrusted)

- File F

  send F to server

  retrieve F’ from server

**Problems:**

- block retrieval update

**Verify:**

- compute \( y’ = \text{hash}(F’) \)
- check \( y = y’ \)
Merkle hash tree on relation

Client stores root hash locally

h_ε = f(h_0 || h_1 )

h_0 = f(h_00 || h_01 )

h_00 = f(h_000 || h_001 )

h_01 = f(h_010 || h_011 )

h_1 = f(h_10 || h_11 )

h_10 = f(h_100 || h_101 )

h_11 = f(h_110 || h_111 )

h_000 = f(t_1 )

h_001 = f(t_2 )

h_010 = f(t_3 )

h_011 = f(t_4 )

h_100 = f(t_5 )

h_101 = f(t_6 )

h_110 = f(t_7 )

h_111 = f(t_8 )

EMP
age name dept

25 joe sales
30 bob admin
32 mary tech
32 jen admin
36 peter hr
41 paul mgmt
43 john hr
51 harry mgmt

[Merkle 1989] [Devanbu, et al. 2001]
Verifiable query (1)

CLIENT: Give me the record for ‘peter’

SERVER: answer is $t_5$
hash path is $h_{101}$, $h_{11}$, $h_0$

CLIENT: verify by computing
new root hash $h'_{\epsilon}$
check $h'_{\epsilon} = h_{\epsilon}$

This proves \((36,\text{Peter},\text{hr})\) was in original database and is unmodified
Verifiable query (2)

CLIENT: Return all employees with age between 35 and 40

SERVER: answer is \( t_5 \). Also return \( t_4, t_6 \).
hash path is \( h_{010}, h_{00}, h_{11} \)

CLIENT: verify by computing
new root hash \( h' \)
check \( h' = h_\epsilon \)

This proves peter is the only employee
between 35 and 50, and unmodified
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

- Smart client, oblivious server
- Server
  - Relational representation of hash tree
  - Query definition, index selection
- Client
  - For queries: reconstruct tree, hash
  - For inserts: translate into tuple updates
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><strong>Query</strong></td>
<td>2.0 ms</td>
<td>.4 ms</td>
<td>5.0</td>
</tr>
<tr>
<td><strong>Range query</strong></td>
<td>6.1 ms</td>
<td>1.3 ms</td>
<td>4.7</td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td>8.3 ms</td>
<td>.8 ms</td>
<td>10</td>
</tr>
</tbody>
</table>

500,000 tuples

Preliminary results
Conclusion

• Integrity of data stored in database systems is critical
• In practice, clients may not trust the server to enforce integrity
• Our techniques can offer strong integrity guarantees with reasonable performance penalty.
Questions ?