Sharing with Limited Trust: An Attack-Tolerant Service in the Durham e-Demand Project

Erica Y. Yang, Jie Xu, and Keith H. Bennett
Department of Computer Science
University of Durham

4 September, 2003
The Problem

- How can we securely and dependably retrieve information from an information provider within an untrusted Grid environment?

- Well, we need to
  - Protect the privacy of users against untrusted Grid nodes (or Grid service providers)
  - Effectively detect or mask any job tampering against both intentional and unintentional faults
  - Obtain correct results (information) even in the presence of malicious attacks
Privacy Issues

- **Identity privacy**
  - Example: “keep Erica’s identity private when she votes.”
  - Representative approaches: Crowds (AT&T), Chaum’s work on Mix network.

- **Location/author privacy**
  - Example: “don’t ask where this document comes from”
  - Representative approaches: Rewebber, FreeNet

- **Intention privacy – our research focus**
  - Example: “ask for Erica’s medical history.”
  - Representative approaches: Private Information Retrieval (PIR)
  - Private and sensitive intention is kept secret to both communication channels and service providers
Application Domains

- Privacy-critical applications
  - Credential services in the Grid, e.g., online Certification Authority (CA)
  - Medical information services
  - Financial information services

- Information sharing among untrusted parties
  - Dynamic collaboration over the Grid
  - Ad-hoc collaboration

- Global and remote computation
  - Mobile agent applications
The Challenges

- Trusted third party
- Trusted computing base
- Theoretical models for PIR
- Practical implementations of PIR (PPIR)
- Detection or masking of job tampering
- Tolerance to malicious attacks
Our Solution: TIR

- We present an attack-Tolerant Information Retrieval scheme (TIR), which is an extension to the PIR scheme, with a new model for malicious faults
System Model

What is the input?

User

The Grid

Network

query₁

answer₁

query₂

answer₂

queryₖ

answerₖ

replica₁

replica₂

replicaₖ

Adversary

Client

Input

Result
The TIR Scheme

Four types of function:
- Query function
- Answer function
- Reconstruction function
- Result verification function*

The system ensures both privacy protection and fault tolerance, provided that:

\[ k = t + 1 + f, \text{ where} \]
- \( k \) – total number of replicas,
- \( t \) – maximum number of replicas in collusion (that can communicate with each other),
- \( f \) – maximum number of corrupted replicas, and
- \( t = 1 \) and \( f = 0 \).

*Without using a verification function, \( k \) can be in the order of \( 5f \)!
TIR: Result Verification

- Probabilistic result verification function

  - Whenever \( t + 1 \) answers for a given job are returned from replicas, a result can be reconstructed. But its correctness is subject to either fault masking by replication or fault detection by a result verification function.

  - In TIR incorrect results can be detected by our result verification function with a probability arbitrarily close to one.

  - The existence of at least a group of \( (t + 1) \) correct answers guarantees the success of result reconstruction in the end.
TIR: Implementation Model

1. <INPUT, ?, s, b>
2. If needed, start the identifier resolution Protocol to obtain the identifier i.
3. Generate a random index set
   \[ RIS = MF(b, i, r) \]
   where \(|RIS| = b\)
4. Generate queries
   \[ Q_1 = QF(i, R) \]
   \[ ... \]
   \[ Q_k = QF(i, R) \]
5. <QUERY, RIS, s, Qk>
6. Generate view \( V(RIS, a) \)
7. Compute answer
   \[ A_k = AF(V, Q_k) \]
8. <ANSWER, Ak>
9. Reconstruct Result
   \[ = RF(A_1, ..., A_{j-1}, A_{j+1}) \]
10. <RESULT, Result>

?: keyword of intended record.
s: schema, where \(|s| = a\).
b: number of records in a view.
r: a set of random numbers
R: a set of random numbers.

MF: meta function.
QF: query function.
AF: answer function.
RF: reconstruction function.
Implementation

- Java J2SDK 1.4.0
- Both replicas and clients are multithreaded
- Communication channels are implemented based on TCP/IP
Experimental Environment

- **Client-side**
  - Time-sharing Sun Sparc E450 with four 250Mhz processors running SunOS 5.8.

- **Servers (up to 5)**
  - 400 MHz Pentium IIs (celeron).
  - RedHat Linux (6.0 or 7.2).
  - 64 Mbytes RAM, and
  - 4 Gigabyte hard disk
  - MySQL 3.23

- **The Durham Campus LAN**
## (Malicious) Attack Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Attack Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. OneRecord</td>
<td>Modify only one record in a view</td>
</tr>
<tr>
<td>2. OneRecord-oneChar</td>
<td>Modify one character of a record in a view</td>
</tr>
<tr>
<td>3. RecordSet</td>
<td>Modify a set of records in a view</td>
</tr>
<tr>
<td>4. Answer</td>
<td>Modify several characters in an answer before sending back to the client</td>
</tr>
<tr>
<td>5. Answer-oneChar</td>
<td>Modify one character in an answer before sending back to the client</td>
</tr>
</tbody>
</table>
Measurements

TPT = TPreQ + TProQ + TPR.

- TPreQ: time taken to prepare queries
- TProQ: server-side query processing time
- TPR: time taken to perform reconstruction
- TPT: total processing time
Experimental Results (1):
Breakdown of costs for various operations in fault-free cases

<table>
<thead>
<tr>
<th>Phrase (normal)</th>
<th>Three replicas (msec)</th>
<th>Percent of TPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Preparation</td>
<td>327</td>
<td>48%</td>
</tr>
<tr>
<td>Server side Computation</td>
<td>275</td>
<td>40%</td>
</tr>
<tr>
<td>Result Reconstruction</td>
<td>84</td>
<td>12%</td>
</tr>
<tr>
<td>Total Processing Time (TPT)</td>
<td>687</td>
<td>100%</td>
</tr>
</tbody>
</table>
Experimental Results (2):

Query processing

Time taken to process queries in fault-free cases (n = 3,000).

Time taken to process queries in faulty cases (n=3,000).
Experimental Results (3):
Performance in the presence of malicious attacks

<table>
<thead>
<tr>
<th>num. of replicas</th>
<th>Normal</th>
<th>OneRecord</th>
<th>OneRecord-oneChar</th>
<th>RecordSet</th>
<th>Answer</th>
<th>Answer-oneChar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three replicas</td>
<td>687</td>
<td>709</td>
<td>741</td>
<td>716</td>
<td>699</td>
<td>676</td>
</tr>
<tr>
<td></td>
<td>3%</td>
<td>increased</td>
<td>8%</td>
<td>4%</td>
<td>2%</td>
<td>-1%</td>
</tr>
<tr>
<td>Five replicas</td>
<td>753</td>
<td>991</td>
<td>952</td>
<td>947</td>
<td>980</td>
<td>963</td>
</tr>
<tr>
<td></td>
<td>32%</td>
<td>increased</td>
<td>26%</td>
<td>26%</td>
<td>30%</td>
<td>28%</td>
</tr>
</tbody>
</table>

Performance Results of Different Attack Categories in milliseconds (n = 46,000)
Strengths of TIR

- Make little assumption of the Grid (Neither trusted servers nor trusted communication channels are required.)

- No conventional cryptography operations are involved (Cryptography can be very costly; e.g. public-key cryptosystems.)

- No information can be obtained by an adversary even in the presence of malicious attacks (This is important to tackle the issue with dynamic collaboration between rapidly changing parties)
Conclusions

- TIR addresses three trust-related problems: privacy protection, error detection and result verification

- We are among the first to actually implement PIR (IBM and Dartmouth College had some implementations of PPIR, but very limited results were provided)

- An effective implementation of TIR is provided

- Good performance is achieved

- TIR performs well even in the presence of malicious attacks

- We are in the process of applying TIR to real-world Grid applications
Demo

- If you are interested, please see our system demo, which includes
  - Normal situations
  - Simulated attack situations