Query Execution Assurance for Outsourced Databases

Radu Sion
(sion@cs.stonybrook.edu)
http://www.cs.stonybrook.edu/~sion

Secure Systems Lab

Computer Sciences
Stony Brook University
→ ⊕ data outsourcing
⊗ query completeness
⊗ searching
⊗ secure co-processor
data outsourcing

**client**
- PDA
- personal email user
- file-client

**server**
- email server
- PostgreSQL
- file-server
data outsourcing: challenges

Untrusted server:
- lazy: incentives to perform less
- curious: incentives to acquire information
- malicious:
  - denial of service
  - incorrect results
  - possibly compromised

Why is this hard?
- how?
- arbitrary expressivity
- overheads
  - network
  - computational costs

What do we do?
- query assurances
- full privacy
  - of queries (even encrypted)
  - of access patterns
- data confidentiality
data outsourcing → query completeness
searching
secure co-processor
querying with completeness: why ?!

Client requires quantifiable assurances that query results are complete and correct, for arbitrary query types in the presence of a server that could be ...

... lazy (we do this *here*)

... and/or fully malicious (!)
P. Golle and I. Mironov, "Uncheatable Distributed Computations", RSA 2001 (Cryptographer's track)
query completeness proofs (lazy server)

A challenge token (computed by client) is sent together with the batch of queries. Upon return, batch execution is proved by \((x==x')\).

\[
C(Q, x, \varepsilon) = \{H(\varepsilon || \rho(Q_x)), \varepsilon\}
\]
secure query interface (SQi)
SQi: client interface

- Data Client
- Security Controls
- Legacy Query Interface
- Extended Query Interface
- Event Callback Management
- Query Pre-processor
- Fast Crypto
- Security Logic Execution Proofs
- Transparent Query Batch Scheduler
- Persistent Query Store
- Client Data Source Adapter
- Plugin Handler
- Privacy Manager

The diagram illustrates the client interface components and their interactions, focusing on security controls and event callback management.
SQi: server

from
client

Incoming Query Queue

Security Logic Execution Proofs

Query Post-processor

Fast Crypto

Server Data Source Adapter

Plugin Handler

Privacy Manager Agent

sqi.server.QueryServer

Legacy Data Source
success probability of cheating

The behavior of $P'(w, r, f)$ (fake tokens) plotted against $P_c(w, r)$ (client-side result checking mechanism) showing that the query execution proof mechanism (with fake tokens) significantly decreases the ability to "get away" with less work.
execution times

(a) Execution times behave naturally linear in the size of the input, (b) Execution time and network overheads behavior with increasing segment size ($r = 1$).
Overheads with increasing number of challenge tokens per batch: (a) execution overhead increases (b) network overhead decreases and eventually becomes negative.
beyond laziness

- client-side result checking
  - weaker assurances of a stronger type 😊
- secure hardware (we’ll see later)
- etc. ?
data outsourcing
query completeness
→ searching
secure co-processor
searching: fun for sure, but important?

Selected scenarios

- compromised server (e.g. network context)
- secure email server
  - do not allow sysadmin to read email 😊
- secure networked file system
  - unable to deploy forensics (without data owner consent)
- secures (from commercial competition):
  - company data
  - data access patterns

Sample: “return all emails containing ‘John’ and ‘lunch’”
searching: fun for sure, but important?!

Challenges

- result assurances
- completeness
- correctness
- confidentiality of data
- obliviousness
  - privacy of searches
  - no correlation leaks
- overheads
  - computational
  - network
  - storage constrained client
- dynamic (updates)
Deploying a modified version of computational PIR targeted at a server-side indexing structure to achieve complete privacy.
data outsourcing
query completeness
searching
→ secure co-processor
e.g. IBM 4758 (4764)
architecture overview (4758)
trust propagation (4758)

- Software:
  - Applications
  - Environment/OS
  - Kernel
  - Loaders

- Firmware:
  - Post
  - Miniboot

- Hardware:
  - Processor
  - Flash, RAM, ROM
  - Locks
  - Tamper-responding unit
  - Crypto functions
A secure co-processor on the data management side may allow for significant leaps in expressivity for queries where privacy and completeness assurance are important.
For conjunctive keyword searches on document (email, files) servers, oblivious search index structures could be queried in secure memory achieving a novel zero-leak query model.
**scpu: hash-join (with privacy)**

Hash-JOIN could be naturally accommodated.
For Merge-JOIN, order-preserving encryption primitives could be deployed to minimize the amount of data parsing required in the sorting phase.
scpu: what about general semantics?

How do we approach the problem of arbitrary query expressivity with strong computational (at least) privacy?

Let's look at things we don't "believe" in 😊 ...
sample “wouldn’t do”: SCPU=client proxy

Good idea? not so sure!
scpu: some things we are afraid to do

- Process entire queries on SCPU (!)
- Dedicate (one) SCPU per query or equivalent
  - e.g., limit TPS by SCPU TPS
- Synchronize CPU with SCPU
  - e.g., block main CPU until SCPU completes
- Transfer $\geq O(n)$ on SCPU-CPU bus (!)
- Anything else un-smart 😊
selected related research (SCPU)


“Practical server privacy with secure coprocessors”, IBM Systems Journal 2001, S. W. Smith, D. Safford


A. Iliev, S.W. Smith, "Prototyping an Armored Data Vault: Rights Management on Big Brother's Computer.", Privacy-Enhancing Technology 2002


Related research at IBM TJ Watson (Bishwaranjan Bhattacharjee a.o.)
cat /proc/lunchtime

Thank You 😊