Machine-Learning-Based Data Compression
and Secure Oblivious Computation

John Mitchell
Jeremy Planul, Joe Zimmerman
Stanford University
Cloud computing
Homomorphic encryption

\[ \text{Compute function } f \text{ on encrypted data} \]
Secret sharing

For $n,k$-secret sharing, secure if $<k$ servers collude
Paradigm

- Secret sharing monad (library) on “cloud” platforms
- Replicated Haskell runtime

Domain-specific language embedded in Haskell

Debugging Environment (no crypto)

Standard Haskell runtime

Homomorphic encryption monad (library)

Optimized custom crypto runtime

- Write general code once using generic monad
- Execute code later on chosen platform
Sample application scenarios

- **SpamFilter**
  - Search encrypted email for public spam indicators
  - Modify encrypted email appropriately, if desired
  - Can we also generate spam indicators this way?

- **Collision avoidance**
  - Initial low-accuracy public location information
  - High-accuracy private info if collision possible

- **MapLocation**
  - Route from secret starting location to secret final location, using public map
Map Location

- **Public map**
- **Private location**
  - Plan route from private start to private end location
  - Could include “detours” for secrecy?
- **Cloud algorithm**
  - Server learns what data is read from map data structure
  - Minimize queries for efficiency
  - Maximize queries for secrecy
- **Secrecy goals**
  - Hide start and end?
  - Minimize probability that any point can be guessed with error $\varepsilon$?
Efficiency of oblivious computing

- Algorithms on plaintext
  - $\sim 10^9$ (FHE)
  - $\sim 10^3$–$10^5$ (SMC)
- Algorithms on hidden data
Efficiency picture: refined

Algorithms on plaintext

~$10^9$ (FHE)

~$10^3$–$10^5$ (SMC)

“Obliviousness” – naively: $S(n)$ ($10^3$, $10^6$, ...?)

Algorithms on hidden data
Efficiency picture: refined

Algorithms on plaintext

\[ \sim 10^9 \text{(FHE)} \]

Data values are hidden (encrypted or shared): if we could see them, we could break security!

"Obliviousness"

\[ \sim 10^3 - 10^5 \text{(SMC)} \]

Algorithms on hidden data
Efficiency picture: refined

Algorithms on plaintext

$\sim 10^9$ (FHE) \quad \sim 10^3 - 10^5$ (SMC)

Computation must be *data-oblivious*: no matter what the input data, we must perform the same operations.

"Obliviousness"

Algorithms on hidden data
Oblivious algorithms

- Data “secret” (encrypted) or “public” (clear)
- Algorithm must be oblivious w.r.t. secret data
- Two different criteria depending on setting
  - Data-independent (FHE)
    - No dependence of control flow on secret data
    - e.g. Boolean circuits, sorting networks
  - Data-oblivious (SMC)
    - Control flow is a function of data and random bits; must be independent of secret data (as distribution)
    - Equivalently: “reveal” operations from secret to public allowed, as long as they preserve secrecy
Making programs oblivious

Results for general oblivious simulation are well-known
- Simple simulation optimal for data-independent (no-reveal) algorithms (c * T(n) * S(n))
- For data-oblivious algorithms (with reveal), Oblivious RAM problem well-studied (c * T(n) * polylog(S(n)))

What about special cases?
Making programs oblivious

- Simple simulation \((c \times T(n) \times S(n))\) is optimal for data-independent RAM programs, but not for Turing machines
  - Pippenger-Fischer, 1979: \(O(c \times T(n) \times \log S(n))\)
- Key is locality: TM tape head can only move left or right by one cell each step
  - Do not pay full cost of obliviousness if access pattern already has some structure
Back to map routing ...

- The client wants to go from a point A to a point B
- The server has a map and wants to give directions to the client

- The client does not want the server to learn anything about A and B
- The server does not want the client to learn anything about the map, apart from the route
Hierarchical approach

Need to read local map

Possible approach

- Read all main highways
- Compute inter-city route publicly
  - “Secret” selection from public list
- Read some number $n$ of city maps
- Compute local route (secretly)
- Reveals city with $1/n$ probability
Possible approaches locally

1. Run a shortest path algorithm \( O(N^2 \cdot \log(N)) \), obliviously \( O(N^2) \):
   - \( O(N^4 \cdot \log(N)) \) (times cost of operations on encrypted data)

2. Propagate distances by updating every node in the map using the distances of its neighbors: \( O(N^3) \)

- Do we need to touch every memory location at every step? \( O(N^2) \)
- No, we can do better if our program accesses locations with a pattern:
  Oblivious Turing tapes (Pippenger & Fischer 1979)
  \( O(t \cdot \log(t)) \) cost for \( t \) steps
Our choice

- We embed the map on a 2D-grid,
- precompute some data at for each point, and
- use an heuristic $f(source\_data, \ destination\_data)$ to find the next vertex.

- Complexity: $O(N^2 \times representation\_size)$
Oblivious routing: embedding on grid
Oblivious routing: locality
Oblivious map routing: solution

- We embed the map on a 2D-grid,
- precompute some data at for each point, and
- use an heuristic $f(source\_data, destination\_data)$ to find the next vertex.

- Complexity: $O(N^2 \times \text{representation\_size})$
Demo/Video
Summary

- To compute on hidden data, must make programs oblivious
- General solutions well-understood, but significant overhead
- In specific cases, can do much better!
  - Example: oblivious TM simulation [Pippenger, Fischer]

New results
- Composition of oblivious data structures
- Navigation on road maps

Intuition: exploit structured memory access pattern in algorithms - or locality in problems