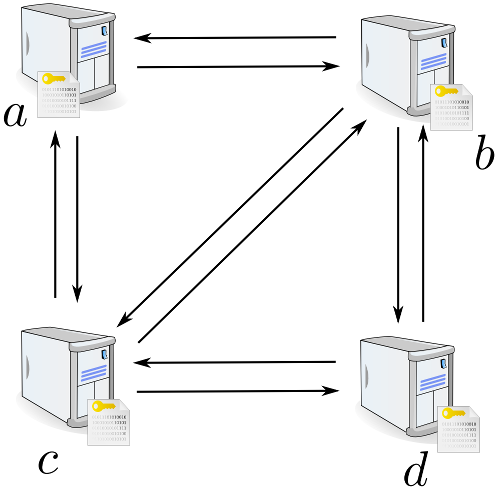
Efficient, Oblivious Data Structures for MPC

Marcel Keller and *Peter Scholl*

Department of Computer Science
University of Bristol

11 December 2014

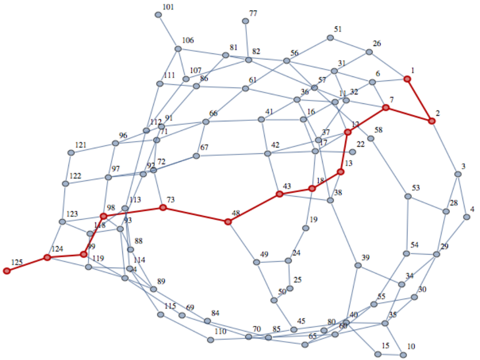
Secure Multi-Party Computation



Goal: compute $f(a, b, c, d)$

Example application

Privacy-preserving shortest path algorithm



Start/destination remain private to server holding map.

Secret sharing-based MPC

Additive secret sharing:

$[x]$: P_i holds x_i (no information on x)

$$x = x_1 + \cdots + x_n \in \mathbb{F}$$

Need all n shares to reconstruct secret.

Arithmetic Black Box:

- Add($[x], [y]$): output $[x + y]$ (local operation)
- Mul($[x], c$): output $[c \cdot x]$ (local operation)
- Mul($[x], [y]$): output $[x \cdot y]$ (send $O(1)$ $\mathbb{F}$-element)
- Reveal($[x]$): output x (send $O(1)$ $\mathbb{F}$-element)

Cost metric: $|\text{comms}|$ + local comp.

Beyond circuits

ABB gives evaluation of arithmetic circuits or binary circuits.

Most programs aren't written as circuits:

- How about array lookup with secret shared index?
- Dijkstra's algorithm?
- RAM programs?

Beyond circuits

ABB gives evaluation of arithmetic circuits or binary circuits.

Most programs aren't written as circuits:

- How about array lookup with secret shared index?
- Dijkstra's algorithm?
- RAM programs?

Goal: augment ABB with oblivious data structures

Results overview

Obliv. data structure		Based on	Complexity
Array		Demux [LDDA12]	$O(N)$
Dictionary	$\{$	N comparison circuits	$O(N \cdot \ell)$
		'Binary search' circuit	$O(N + \ell \cdot \log N)$
Array	$\{$	SCSL ORAM	$O(\log^4 N)$
		Path ORAM	$O(\log^3 N)$
Priority queue	$\{$	Array	$O(\log^4 N)$
		Modified Path ORAM	$O(\log^3 N)$

N: # items

ℓ: length of keys (for dict.)

Simple Oblivious Array/Dictionary

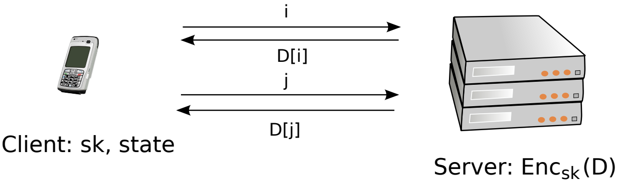
Compute N comparisons:

$$[c_0] = ([i] \stackrel{?}{=} 0),\ [c_1] = ([i] \stackrel{?}{=} 1),\ ,\ldots, [c_{N-1}] = ([i] \stackrel{?}{==} N-1)$$

$$x_{[i]} = \sum_j [c_j] \cdot [x_j]$$

Comparison cost: $O(\ell)$ comms/computation (constant round)
Total: $O(N \cdot \ell)$

Oblivious RAM



Goal: hide *access pattern* (i, j) from server.

Access pattern must be randomized, polylog(N) overhead

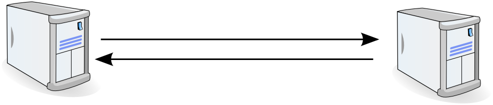
[Gol90, GO97]

Key observation

ORAM $+$ MPC for circuits $\Rightarrow$ polylog(N) oblivious array for MPC

[OS97, DMN11]

Oblivious array using ORAM



c_1 c_2

D_1 D_2

secret share D

secret share client state,

- Replace encryption with secret sharing (c.f. [DMN11])
- Execute instructions client/server within MPC
- Reveal client's address queries – secure by ORAM simulation

Related: client-server model using Yao [GKK+12, AHMR14]

Oblivious array using ORAM

Challenge: design MPC circuit for ORAM instructions

Tree ORAM schemes

Tree ORAM [SCSL11]:

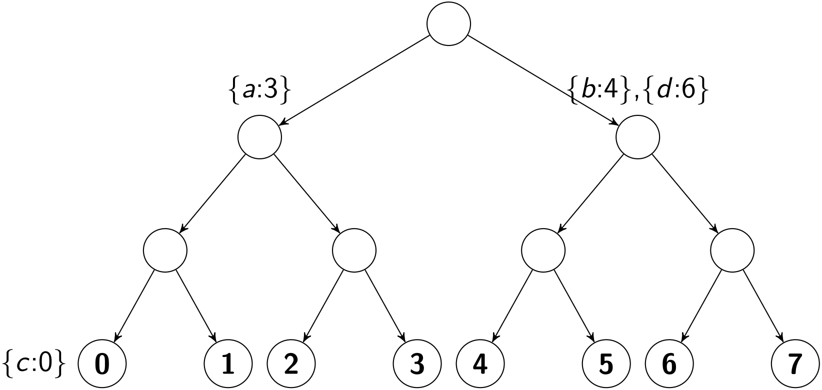
- Simple, tree-based construction
- $O(\log^3 N)$ overhead
- Worse asymptotics than previous, but much more practical

Path ORAM (Stefanov et al. CCS '13):

- Same structure as [SCSL11]
- New 'path eviction' method
- $O(\log^2 N)$ overhead

Similar works: [GGH+13, CLP14]

Tree-based ORAM



Index	a	b	c	d
Leaf	3	4	0	6

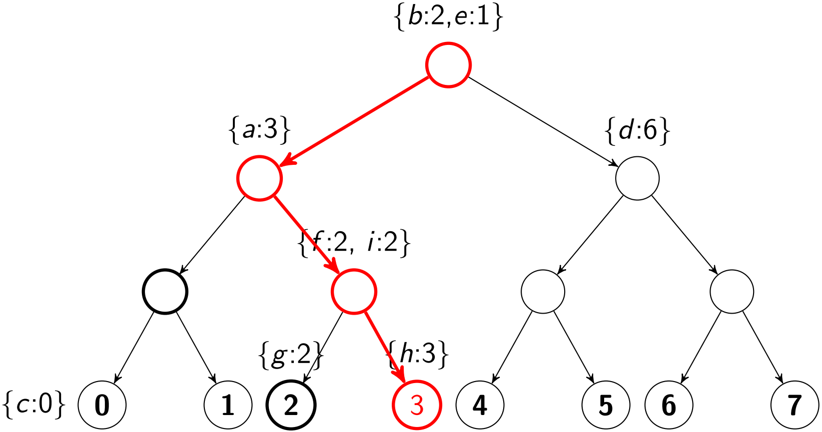
Invariant: x lies on path from root to Leaf(x)

Each node is bucket of fixed size $Z = 2$

Path ORAM eviction

Choose random leaf

Push entries as far down path as possible

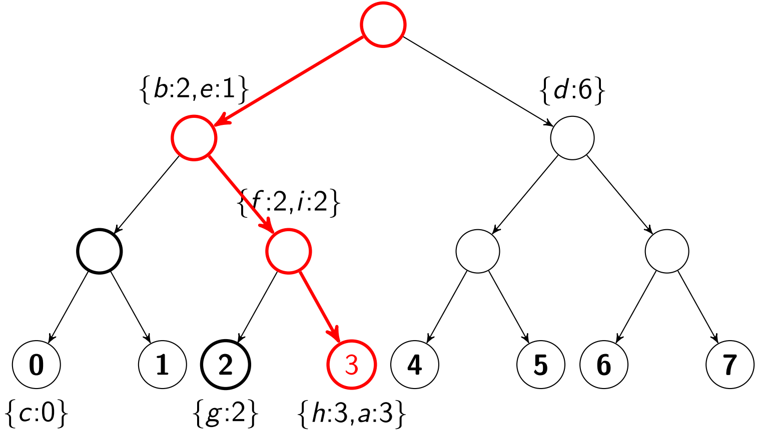


Index	a	b	c	d	e	f	g	h	i
Leaf	3	2	0	6	1	3	5	2	5

Path ORAM eviction

Choose random leaf

Push entries as far down path as possible



Index	a	b	c	d	e	f	g	h	i
Leaf	3	2	0	6	1	3	5	2	5

University of BRISTOL

Path Eviction in MPC

Eviction leaf: ℓ^*
Entry in path: leaf ℓ
Calculate level entry ends up at:

- Least Common Ancestor of ℓ, ℓ^*
  - First bit where ℓ, ℓ^* differ
  - $\equiv$ first 1 in $\mathsf{BitDec}(\ell \oplus \ell^*)$
- Adjust LCA to account for bucket overflows

$O(\log N)$ comp. per entry, for path + stash size $O(\log N)$

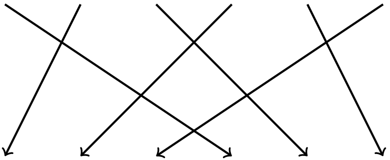
$$\Rightarrow O(\log^2 N)$$

Now need to assign levels (more complex)

Path eviction in MPC

$$([E_1], [\mathsf{lev}_1]) \quad ([E_2], [\mathsf{lev}_2]) \quad \ldots \quad ([E_n], [\mathsf{lev}_n])$$

First idea: oblivious shuffle with permutation networks



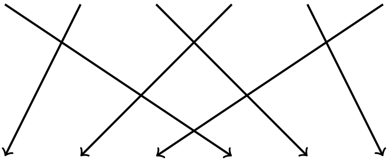
$$([E_{\pi(1)}], [\mathsf{lev}_{\pi(1)}]) \quad ([E_{\pi(2)}], [\mathsf{lev}_{\pi(2)}]) \quad \ldots \quad ([E_{\pi(n)}, [\mathsf{lev}_{\pi(n)}])$$

Reveal $\mathsf{lev}_{\pi(i)}$ and place entry $[E_i]$ in bucket

Path eviction in MPC

$$([E_1], [\mathsf{lev}_1]) \quad ([E_2], [\mathsf{lev}_2]) \quad \ldots \quad ([E_n], [\mathsf{lev}_n])$$

First idea: oblivious shuffle with permutation networks



$$([E_{\pi(1)}], [\mathsf{lev}_{\pi(1)}]) \quad ([E_{\pi(2)}], [\mathsf{lev}_{\pi(2)}]) \quad \ldots \quad ([E_{\pi(n)}, [\mathsf{lev}_{\pi(n)}])$$

Reveal $\mathsf{lev}_{\pi(i)}$ and place entry $[E_i]$ in bucket

Problem: some entries may be empty (dummy)

- If E_i is empty then $\mathsf{lev}_i = \perp$
- Reveals # of empty entries

Solution: requires oblivious sorting – $O(\log N \log\log^2 N)$

Oblivious Array from Path ORAM: summary

Single eviction cost: $O(\log^2 N)$ comms/comp, $O(\log N)$ rounds

Read/write cost: $O(\log N)$

$\Downarrow \times O(\log N)$ levels recursion

Total cost: $O(\log^3 N)$

In practice: approx. 30% LCA comp, 30% sorting, 30% shuffling

Application: Dijkstra's algorithm

- $|V|, |E|$ public
- Graph structure secret
- Maintain distance to nodes in oblivious priority queue Q

```
for each edge
    if new vertex
        v = Q.pop()
    [ ... ]
    Q.DecKey(v, [ ... ])
```

Dijkstra complexity

Ordinary complexity: $O(|E| + |V| \log |V|)$

MPC: $O(|V| \log^3 |V| + |E|(\log^3 |E| + \log^3 |V|))$
Sparse graphs where $|E| = O(|V|)$:

- $O(|V| \log^3 |V|)$

First MPC implementation of Dijkstra with sublinear overhead.

Implementation

Implemented using SPDZ MPC protocol:

- Information-theoretic online phase
- SHE for 'preprocessing' phase
  - Independent of inputs
  - Can be done in advance

Timings

Oblivious array: < 100ms (online) for size 1 million with Path ORAM

Path ORAM beats $O(N)$ solutions for sizes > 1000.

Dijkstra: ≈ 5000s for graph with 1000 nodes

Conclusion

Oblivious data structures in MPC are practical.

Open problems:

- More data structures; RAM programs
- Better ORAM for MPC? Circuit ORAM [WCS14]

Conclusion

Oblivious data structures in MPC are practical.

Open problems:

- More data structures; RAM programs
- Better ORAM for MPC? Circuit ORAM [WCS14]

Thanks for listening!