6.892 Computing on Encrypted Data	December 09, 2013
Lecture 9	
Lecturer: Vinod Vaikuntanathan	*Scribe: Nathan Rittenhouse*

1 Oblivious Database Access

It is often the case that a client wants to obtain data from a server but hide the replies and the access patterns. Two models, private information retrieval and oblivious are used to achieve this end. The server is assumed to be an honest but curious party and the client is assumed to be free from attackers. An attacker has access to the communication channel and wants to learn communication patterns.

2 Private Information Retrieval

2.1 Primitives

PIRQuery(i) A message sent from the client to the server that contains the index of the desired database entry.

PIRReply An encrypted reply to the query returned by the server.

PIRDecode A method used by the client to decode the server's response.

2.2 Correctness

All queries should succeed with probability $(p = 1)$.

2.3 Security [1]

Lemma 1. *For all* $(i, i^{'})$: *PIRQuery(i)* $\overset{c}{\approx}$ *PIRQuery(*$i^{'}$*)*

Lemma 2. *For all* $(i_1, \dots i_j, \dots, i_t)$: *PIRQuery(i)* $\overset{c}{\approx}$ *PIRQuery(*$i^{'}$*)*

Constraints

- In an unencrypted setting, the number of bits for the query number is $\Omega(\lg(N))$, which will not allow for security under information theoretic assumptions.
- To prevent information leakage, all database entries must be touched. For that reason, we primarily focus on reducing the communication complexity.

	Single Server	Many Servers
Unconditional (semantic security)	$\mathcal{O}(N)$ (server must read entire DB)	$\mathcal{O}(N)^{\lg \lg(N)}$
With cryptographic assumptsions	Server must touch every entry. Assuming difficulty of quadratic residues: $\mathcal{O}(\sqrt{N})$. FHE: $\mathcal{O}(\lg(N))$	See multi-server section

3 FHE Databases

Example: $f_{DB}(x_1, \ldots, x_t) = \sum_{i \in (i_1, \ldots, i_t)} DB[i](x_1 - i_1) \ldots (x_t - i_t)$
Protocol for FHE DB queries

1. Client sends (FHEEnc(*seed*), PseudoRandom(*seed*)$\oplus i$)
2. Server recomputes the PseudoRandom(*seed*) homomorphically
3. Server performs homomorphic exclusive or to obtain i
4. Server runs the above query to obtain the query response

3.1 Multi-Server FHE DBs

1	2	3	4
See single-server section	$\mathcal{O}(\sqrt[3]{N})$	$\mathcal{O}(N^{\frac{1}{\lg\lg(N)}})$	$\mathcal{O}(\sqrt{N})$

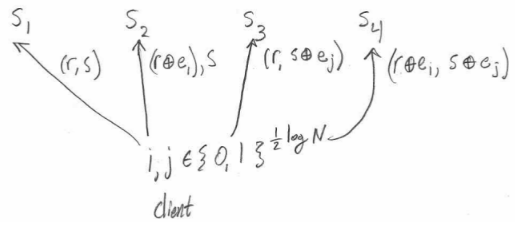
Best communication complexity lower bound: $\mathcal{O}(\lg(N))$

3.2 4-Server Scheme

We first encode our database as our matrix. Queries take the form of choosing the entry at the intersection of the ith row and jth column.

e_i is the row vector which selects entries of the row we wish to select

e_j is the vector that when transposed will properly select the column



$e_i \cdot DB \cdot e_j^T = DB(i, j)$

4 Oblivious RAM

In the oblivious RAM model, a client communicates with a server, much like in the previous model. N bits total are stored in the memory.

CPU is defined as the client

Memory is defined as the server

4.1 Primitives

read(i) reads the memory at index i

write(j, m) writes value m to index j

4.2 Assumptions

1. The client's memory is protected
2. The values written to memory are encrypted.
3. The server is honest-but-curious - it snoops memory reads but is not actively mangling the data returned to the client.

4.3 Security

Any length-two subsequences of database accesses should be statistically indistinguishable.

5 Simple ORAM Construction

CPU creates a permutation $\pi : [N] \rightarrow [N]$ that remaps each memory location to another memory location. Although this construction randomizes individual accesses, it will still reveal repeated access sequences.

The setup time for this scheme is $\theta(N)$ to write down a permutation array.

6 Goldreich-Ostrovsky ORAM Construction[2]

Key ideas: add a cache, which maps the pre-permutation index to the actual memory value. Add garbage accesses when the lookup address is cached.

6.1 Data Structures

The server's memory is segmented into:

1. N entries for the permutation
2. N entries for the actual memory values
3. $\sqrt{N}$ entries for the cache
4. $\sqrt{N}$ entries for garbage indices

To create the memory index permutation on the server:

1. Zero out the cache
2. Create permutation $\pi^{'}$
3. Write $\pi^{'}$ to the permutation segment
4. Sort $\pi^{'}$ on the server with data oblivious sorting algorithm ($\mathcal{O}(N\lg(N))$)

6.2 Protocol

Method for obscuring the locations of reads and writes

1. Client downloads the cache
2. If the cache contains the ith entry then the client has found the index
3. If the client has found the index, it accesses π(last_garbage_index+1)
4. Append (i, v) to the cache if our value was not in the cache or $(i_{garbage}, v_{garbage})$ if it was

6.3 Overhead

Memory N entries to hold permutation, $2 \cdot \sqrt{N}$ for cache and dummy cache

Computational overhead $\mathcal{O}(\sqrt{N})$ per access and $\mathcal{O}(N \lg(N))$ accesses until a re-permutation. Thus accesses have $\Theta(\sqrt{N})$ complexity, amortized.

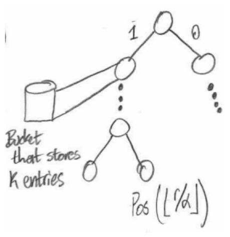
6.4 Problems

1. Cache overflows require a relatively high amount of work of $\mathcal{O}(N \lg(N))$
2. Reads are $\mathcal{O}(\sqrt{N})$
3. Assuming the existence of pseudo-random functions and one way functions. That assumption is needed because in this model, it is assumed that the CPU does not have enough memory to store a permutation.

7 Chung-Pass [3] ORAM Scheme

Advantages over prior schemes

1. No cryptographic assumptions
2. Worst case analysis for lookup times rather than amortized.



7.1 Definitions

N - number of bits stored in the memory

k - the number of blocks in a bucket

α - the number of blocks

7.2 Data structures

1. Memory blocks of length α are stored at the $\pi(i)$th location
2. Each memory block is described by a tuple: (block number, position descriptor, data for memory block). Size: $\lg \frac{N}{\alpha} + \lg(N) + \alpha$
3. CPU has $\frac{n}{\alpha}$ buckets that can store k memory-block tuples
4. A Huffman-tree like strucutre is employed to resolve indices to data

7.3 read(r)

1. Read entire path from root to $Pos(\lfloor \frac{r}{\alpha} \rfloor)$. Output mem[$r$] if r exists, otherwise output $\perp$.
2. Create a new remapping $pos^{'}$ for r by adding $\lfloor \frac{r}{\alpha} \rfloor$ to the root of the tree.
3. Pick a random pos^*, move each common ancestor pos_i of pos and $pos^{'}$ to the bucket at the least common ancestor of pos^* and pos_i

7.4 Overhead

Memory overhead $\frac{N}{\alpha} \cdot k \cdot \alpha \approx k \cdot N \cdot \log_\alpha(N)$

Computational overhead for each read and write: $\lg(\frac{N}{\alpha}) \cdot k \cdot \lg_\alpha(N) = \text{polylog}(N) = \omega(\log^2(N))$

7.5 Oblivious-ness Argument

Obliviousness: Each decision in each read and write is conditioned on history that is comprised of random paths.

7.6 Overflow Resistance

Overflow resistance: The probability of any $pos^{'}$ being a child of location $\gamma \leq 2^{-\frac{k}{2})} \leq 2^{\omega(\log(N))}$.

References

[1] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998. [Online]. Available: http://people.csail.mit.edu/madhu/papers/1995/pir-journ.pdf

[2] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, pp. 431–473, May 1996. [Online]. Available: http://eprint.iacr.org/2010/366.pdf

[3] K.-M. Chung and R. Pass, "A simple oram," Cryptology ePrint Archive, Report 2013/243, 2013, http://eprint.iacr.org/.