Lecture 4:

Private Information Retrieval...

Extensions

MIT - 6.893
Fall 2020
Henry Corrigan-Gibbs

Plan

* Discuss
  - Most important priv/sec issues out there.
  - What would you like to see covered?

* Recap: PIR

* Longer DB rows

* PIR by keywords
  + functionality

* Stretch break

* Batch PIR
  + efficiency on server

Logistics

- iPad wins!
- HW 1 due 9/18 @ 5pm via Gradescope (HW out 9/22)
- Anonymity on Piazza

Recap: Private Information Retrieval

Read a row of a database w/o database learning which row you read.

→ Should be surprising that this is even possible!

Two flavors: multi-server + no assumptions (info theoretic security)
− 2+ non-colluding servers
(Where do you get 2 non-colluding Googles?)

single-server + one server (easier to deploy)
− use relatively heavy crypto (public-key)

To give a sense of the performance:

latest 1-server ≈ 80 MB/s server throughput
2-server ≈ 35,000 MB/s " "

!

Properties: 1. Correctness

2. Security. [1-server] $\forall n \in \mathbb{N} \quad \forall i, i' \in [n]$

$$\{\text{Query}(1^n, i)\} \approx_c \{\text{Query}(1^n, i')\}.$$

All ideas we will see today make "black-box" use of the underlying PIR scheme.

→ Apply to both the single- and multi-server settings.

PIR for longer DB rows.

On Monday, each row was 1 bit long.

Now, each row is ℓ bits long.

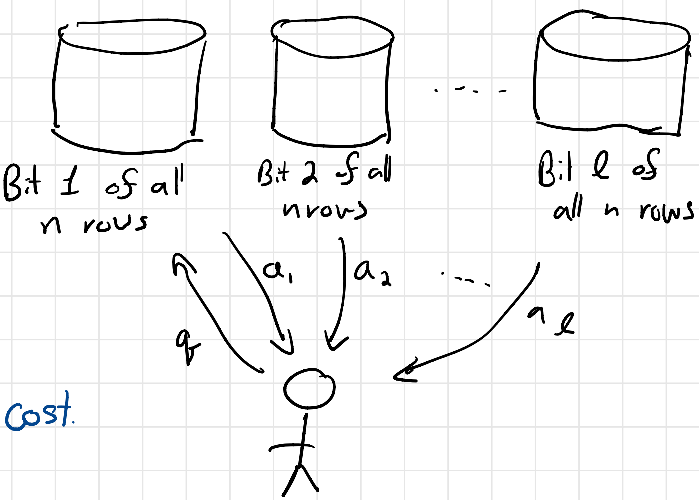
Naive solution: View DB as a single $n \cdot \ell$-bit string.

Comm cost: Upload $U(n) \longrightarrow \ell \cdot U(n \cdot \ell)$

Dnload $D(n) \longrightarrow \ell \cdot D(n \cdot \ell)$.

Scheme from Monday: $O(\ell^{3/2}\sqrt{n})$ cost.

Better idea: Think of client as fetching bit $i \in [n]$ from ℓ distinct databases.



Upload: $U(n)$

Download: $\ell \cdot D(n)$

$\sqrt{n} + \ell\sqrt{n}$

$= O(\ell \cdot \sqrt{n})$ cost.

When ℓ grows large can trade upload for download to decrease cost further.

PIR by keywords (Chor, Gilboa, Naor '97)

Standard PIR: DB is $x \in \{0,1\}^n$
Client holds $i \in [n]$
Client wants $x_i \in \{0,1\}$.

PIR by Keywords: DB is $\{key_1, \ldots, key_n\}$

ℓ-bit strings

Client holds string $s \in \{0,1\}^{\ell}$
Client wants to know $s \overset{?}{\in} \{key_1, \ldots, key_n\}$

→ Extension to (key, value) lookups is almost immediate ←

Key-value API may be useful (e.g. for DNS).

↳ Not the end of the story... fuzzy text search would be even more useful.

We will show:

$\boxed{\text{k-server PIR scheme communication } C(n, \ell)} \Longrightarrow \boxed{\text{k-server PIR by keywords scheme with communication } O(\log n) + C(n, O(\log n)) + C(3n, \ell)}$

For natural scheme $O(C(n, \ell))$.

Stretch Break

Construction

Idea: Clever use of hashing.

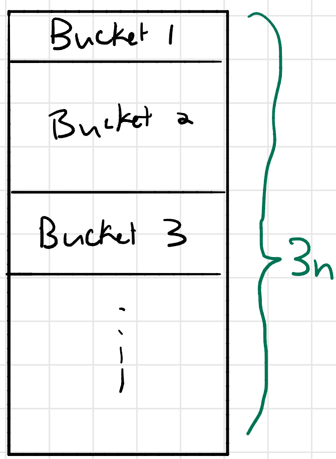
Let $H : \{0,1\}^{\ell} \to [n]$ be a hash fn that we model as random oracle* almost

Not necessary here, but simplifies the analysis

Two steps:

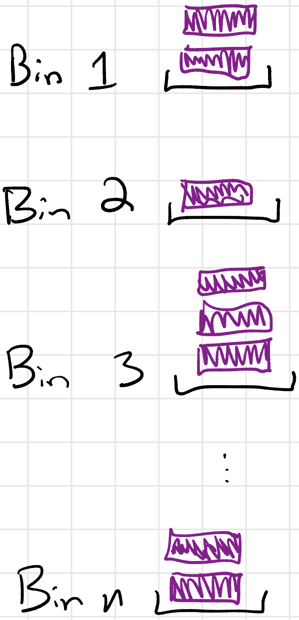
1. Assign each key to one of n buckets.
   (Using hashing)

2. Store the contents of all n buckets in an array of size $\leq 3n$.
   (Using hashing again)



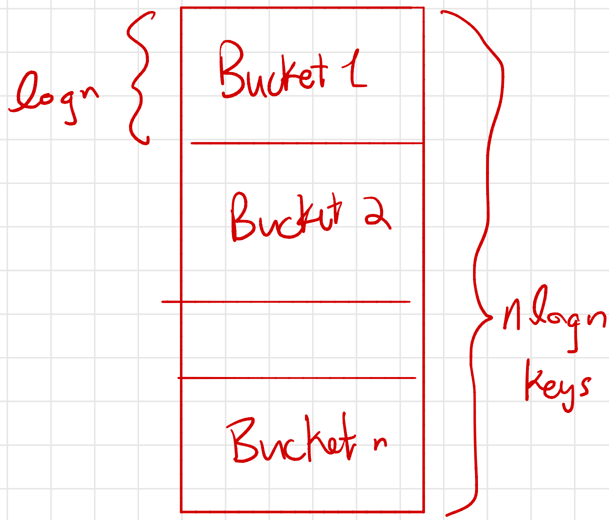
Step 1: Assign keys to buckets.

Just hash each key with H.



Problem: Each bucket will contain a different # of keys.

Standard balls-in-bins result: $O(\log n)$ bins in heaviest-loaded bucket.



Step 2: Store all buckets in an array of size $3n$.

For each bucket $b \in \{1, \dots, n\}$, with load b_n, find a hash fn

$$h_b : \{0,1\}^{\ell} \to [n_b^2]$$

s.t. $\nexists$ keys k, k' in bucket b w/ $h_b(k) = h_b(k')$.

(Assume that we can find such an h_b.)

POINTER TABLE

1	h_1	$offset_1$
2	h_2	$offset_2$
3	h_3	$offset_3$
4	h_4	$offset_4$
	$\vdots$	
n	h_n	$offset_n$

DATA TABLE

$offset_1 \to$	000000	
	key_7	$\leftarrow offset_1 + h_1(key_7)$
$offset_2 \to$	key_{231}	
$offset_3 \to$	$\vdots$	
$\vdots$		
$offset_n \to$	$\vdots$	
	key_{54}	

How lookups work

* Client downloads description of H.

* Given string s, client locally computes bucket index b as

$$b \leftarrow H(s). \qquad (b \in [n])$$

* Client uses one PIR query at index b to POINTER TABLE to fetch

$$(h_b, \text{offset}_b).$$

* Client makes a second PIR query at index $\text{offset}_b + h_b(s)$ to DATA TABLE to fetch s (if exists).

Need to show

* How to construct $h_1, \dots, h_n$

* That DATA TABLE isn't too large ($\leq 3n$ keys)

Efficiency

Description of H .... $O(\log n)$ bits.

1 PIR query to table of n rows, $O(\log n)$ bits each.

1 PIR query to table of $3n$ rows, ℓ bits each.

I. Constructing $h_1, \dots, h_b$

Need $h_b : \{0,1\}^\ell \to [n_b^2]$ such that n_b distinct values in $\{0,1\}^\ell$ map to distinct values under h_b.

A random oracle h_b will satisfy this property w/ probability $\geq 1/2$.

Probability of collision

$$\binom{n_b}{2} \cdot \left(\frac{1}{n_b^2}\right) = \frac{n_b(n_b-1)}{2\, n_b^2} < 1/2.$$

Lazy approach: Given a big random oracle $H : \{0,1\}^* \to \{0,1\}^n$,

Construct a small random oracle as

$$h_b(x) := H(\text{salt}_b,\ x\ ) \pmod{n_b^2}.$$

for $\text{salt}_b \in \{0,1\}^n$. (Half of salts will work.)

Danger! There are some subtleties here, but this general strategy will work.

(Non-lazy approach uses pairwise independent hashing.)

II. Showing that data table isn't too big.

Let $H: \{0,1\}^{\ell} \to [n]$ be a random fn.

Then let C be number of pairs (k,k') such that $h(k) = h(k')$.
$k \neq k'$

$$\mathbb{E}[C] = \binom{n}{2} \cdot \left(\frac{1}{n}\right) \leq n/2.$$

By Markov's inequality,

$$\Pr[C > n] \leq \frac{n}{2}\left(\frac{1}{n}\right) \leq 1/2.$$

So, if we look at two choices of H (on average), we will find one w/ $C \leq n$.

Salted random oracle.

We want to bound

$$\sum_{i=1}^{n} b_i^2 = \sum_{i=1}^{n} b_i + \sum_{i=1}^{n} b_i(b_i - 1)$$

$$= n + 2 \cdot \sum_{i=1}^{n} \binom{b_i}{2}.$$

$$= n + 2 \cdot C.$$

$$\leq 3n.$$

Batching PIR

(See Ishai, Kushilevitz, Ostrovsky, Sahai 2004)

All PIR schemes we've seen so far have $\Omega(n)$ server-side computational cost.

Idea: If client wants to make q queries, all at once to the same DB, server can answer <u>all</u> queries at computational cost $\approx n$.

→ Example: DNS lookups,
mapping phone #s to public keys
...

Strategy: Again, use hashing.

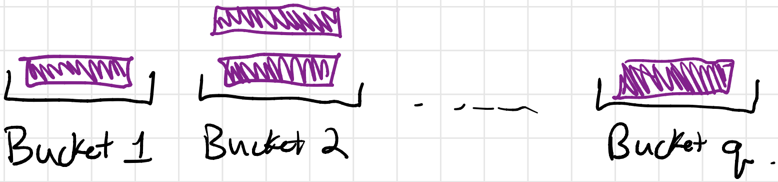
Client $(i_1, \ldots, i_q \in [n])$ Server $(x \in \{0,1\}^n)$

1. Client and server choose a hash fn $H: [n] \to [q]$. (Again, random oracle)

↳ Maps each data item x_i into bucket $H(i)$.

↳ q buckets.

2.



Standard balls-in-bins analysis shows that w.p. $\leq 2^{-\lambda}$, at most $\lambda \log q$ of the client's desired indices fall into any one bucket.

(Not too many collisions.)

3. Client & server view each bucket as a separate database.

Client makes $\lambda \log q$ PIR queries to each bucket.

$\Rightarrow$ Enough to recover $(x_{i_1}, \ldots, x_{i_q})$.

Efficiency

- If original PIR scheme has server time $T(n)$, q-query batch scheme has

$$T_\lambda'(n,q) = \underbrace{(\lambda \log q)}_{\#\text{ queries}} \cdot \underbrace{q}_{\#\text{ buckets}} \cdot \underbrace{T(n/q)}_{\text{Time per query per bucket}}$$

$$T_\lambda'(n,q) = \underbrace{(\lambda \log q)\, n}_{\text{Within a } \lambda \log q \text{ factor of optimal.}}$$

- Original PIR scheme has comm $C(n)$

$$C_\lambda'(n,q) = (\lambda \log q) \cdot q \cdot C(n/q)$$

If $C(n) = \sqrt{n} \Rightarrow \tilde{O}(\lambda \sqrt{nq})$.

More complicated constructions can improve on the $\lambda \log q$ factor.