Lecture 5: Offline/Online PIR

MIT - 6.893
Fall 2020
Henry Corrigan-Gibbs
Plan

- Recap: Batch PIR (+ PIR by keywords)
- Computation in PIR
- Stretch break
- Offline/Online PIR

Logistics

- Last chance for HW1: today 3pm
- HW2 Posted
  - Start early!
  - I am not joking!
  - Due 10/2 3pm via Gradescope
- Breakout sessions:
  - will aim for 1/wk
- Guest lecture on W... please look at reading & bring one Q.
- Computer issues...
Recap:

PIR by keywords.

\[ 0 \in \{0,1\}^k \]

\[ 1 \leq s \in \text{DB} \]

\[ 0 \circ \circ \]

Bottom Line: Cost of PIR by keywords

\[ \leq \]

Cost of normal PIR

Approach: Hash twice...
- Once into buckets (few collisions)
- Once within buckets (no collisions)
Recap: Batch PIR

Idea: Answer $g$ queries at server-side cost of answering 1 query.

Observation:

$$\frac{1 \text{ query to DB of size } n}{q \text{ queries to DB of size } \frac{n}{g}}.$$ 

Strategy: If client wants to make $q$ queries, partition DB into $g$ chunks at random.

As long as clients' desired bits fall in diff chunks $\Rightarrow$ can recover.

$\Rightarrow$ A little more work gets correctness w/ $\leq 1$ but negl prob.
**Server-Side Computation in PIR**

BIM'04: If PIR servers store DB in original form, servers must probe every DB bit in responding to client's query.

Intuition: Servers don't touch bit *⇒* Client is probably not reading bit *⇒* Takes some work to make precise.

How to get around?

1. Batching: Amortize cost of linear scan over many queries.
2. Preprocessing: Server does linear scan in a preprocessing phase.

⇒ Per client (today) — offline/online
⇒ Per database (HW) — "PIR w/ preprocessing"

⇒ Active area of research. Let's to do still...
Offline - Online PIR (with D. Kogan)

- Will discuss two - server setting ... also makes sense in single - server setting.

Idea: Push heavy work to an offline phase ... takes place before client even knows which DB element it wants.

- Push heavy linear scan to a more convenient time (out of critical path).

- Servers can still store DB in unmodified form

- Other approaches that you'll see on HW blow up server storage

  \[ n \times \text{bit DB} \rightarrow n^3 \times \text{bit preprocessed data structure} \]

Notation: \( O(\mathcal{A}(n)) = O(\mathcal{A}(n) \cdot \text{poly log}(n)) \)
Will present a two-server scheme with computational security (PRG)

\[ \text{comm } \mathcal{O}(\sqrt{n}) \]
\[ \text{online time } \mathcal{O}(\sqrt{n}) \]

We show that for PIR schemes in which servers store DB in unmodified form

\[ \text{comm. } S \quad \text{online time } T \Rightarrow S \cdot T \geq \mathcal{O}(n). \]

⇒ This \( \sqrt{n} \) scheme is optimal u.r.t. these parameters.

* Can get info-theoretic security.
* Can get perfect correctness.
* Can reduce client running time.
* Can reduce online communication.

} See paper for many of these extensions.
OFFLINE (think: overnight batch job)

```
```

ONLINE (think: when you browse web)

```
```
PIR Security & correctness properties are as before.

Construction: Offline Phase

1. Choose a random partition of \{1, ..., n\} into \sqrt{n} sets \(S_1, ..., S_{\sqrt{n}}\), each of size \(\sqrt{n}\).

2. Server sends parity of DB bits in each set

\[ h_1 = \sum_{i \in S_1} x_i \mod 2 \]
\[ h_\sqrt{n} = \sum_{i \in S_\sqrt{n}} x_i \mod 2 \]

3. Client stores his and S's as hint.

Construction: Online Phase

1. Find set \(S_j\) s.t. \(i \in S_j\).

2. Construct set \(S_j' \times \text{see next page}\) send to server

3. Reply w/ parity of DB bits in set.

\[ a = \sum_{i \in S_j} x_i \]

4. Compute \(x_i \leftarrow h_j \oplus a\).
How client constructs $S'$ ...

Flip a coin $\mathcal{U}$, $p_{\text{heads}} = \frac{\sqrt{n} - 1}{n}$

1. If heads, choose random $i \in S \setminus \{i\}$
   Set $S' = S \setminus \{i\}$
   Output "fail"

2. If tails, set $S' = S \setminus \{i\}$.

Why this works...

Correctness: with probability $1 - P_{\text{heads}} \approx 1 - \frac{1}{\sqrt{n}}$ ...

$S' = S_i \setminus \{i\}$, so client outputs

$$h_j \oplus a = \sum_{x \in S_j} \sum_{x' \in S'} x \oplus x' \pmod{2}$$

$$= \sum_{x \in S_j} \sum_{x' \in S_j} x \oplus x' \pmod{2}$$

$= x_i$.

$\Rightarrow$ Scheme fails w.p. $\approx \frac{1}{\sqrt{n}}$.

Repeat the whole thing $\lambda$ times in parallel to ensure that 1 execution succeed w.p. $\geq 1 - \left(\frac{1}{\sqrt{n}}\right)^\lambda \geq 1 - 2^{-\lambda}$. 

Security

(Left server)

Sees only the random partition $S_1, \ldots, S_n$ is independent of client's index $i$.

(Right server)

Claim: Set $S'$ is a set of $\sqrt{n-1}$ random elements chosen w.o. replacement from $[n]$. Here is one (funny) way to sample a set $S'$ of $\sqrt{n-1}$ random elements from $[n]$.

Is $i \in S'$?

No $\frac{u.p.}{1 - \frac{\sqrt{n-1}}{n}}$

Output random set of $\sqrt{n-1}$ elements not containing $i$.

Yes $u.p. \frac{\sqrt{n-1}}{n}$

Output random set of $\sqrt{n-1}$ elements containing $i$.

This is exactly how client constructs $S'$.

$\Rightarrow S'$ is a random set, indep of $i$.

$\Rightarrow$ Info-theoretic security.
**Efficiency**

- Need to repeat $A$ times to drive failure prob $\rightarrow 0$.

- Offline cost
  - Server time: $\lambda$ random reads
  - Comm: upload: $1$ PDU seed
  - Storage: $\lambda \sqrt{n}$ parity bits

- Online time
  - Server: $\lambda \sqrt{n}$ random reads
  - Comm: $\lambda \sqrt{n}$ bits $\leftarrow$ can reduce to $\text{poly}(A, \log n)$
  - Client time: ???

- Standard PIN server: $n/2$ XORS over DB in linear scan
- Off-line PIN offline server: $1/n$ XORS in random order

  $\Rightarrow$ $-128x$ cost $\frac{1}{2}$ of $\lambda$
  $\Rightarrow$ $-10x$ cost $\frac{1}{2}$ of random reads

Not clear how to reduce.
Extensions

- Can reuse one hint for many subsequent online queries (requires tweaks)

$$\frac{\tilde{O}(n)}{q} + \tilde{O}(\sqrt{n}) \Rightarrow \tilde{O}(\sqrt{n}) \text{ for } q \text{ large}.$$
**OFFLINE**

```
```

```
```

```
i\in\{0,1\}^n
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```
Moral of the story:

1. Can reduce online server time in PIR using preprocessing.
   \[\Rightarrow\] Still new things to say about an old problem.

2. Still not clear whether these techniques will work in practice at scale.

3. Competition is no privacy.
   \[\Rightarrow\] How can we ever outperform that?

\[\uparrow\] This is the important question to think about as you are working on crypto + privacy tech.