Communication-Computation Trade-offs in PIR

Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth and Kevin Yeo

Google
Humboldt-Universität zu Berlin
Outline

- What is Private Information Retrieval (PIR)?
- Our Contributions
  - MulPIR: Improved Communication
  - Gentry-Ramzan Improvements
  - Sparse PIR
Private Information Retrieval

Index \( i \) was downloaded.
Private Information Retrieval

What was the requested index?
Efficiency Considerations
Efficiency Considerations

Client Computation

\[ \text{Client} \rightarrow \ldots \rightarrow \text{Server} \]

\[ \text{Computation} \]

\[ B_1 \rightarrow B_2 \rightarrow \ldots \rightarrow B_n \]
Communication-Computation Trade-offs

Communication

Computation

Communication

Computation
What is the best trade-off?

- Completely depends on context
- Typically, client computation must be small as querier is a user device
  - Example: Mobile phones querying a cloud storage provider
- Can estimate best trade-off using monetary costs
  - Example: Cloud computing prices
Homomorphic Encryption-based PIR

\[ E(0), ..., E(0), E(1), E(0), ..., E(0) \]

\[ E(0 \cdot B_1 + ... 0 \cdot B_{i-1} + 1 \cdot B_i + 0 \cdot B_{i+1} + ... 0 \cdot B_n) = E(B_i) \]
Recursion for PIR

\[
\begin{array}{cccc}
\text{Enc}(0) & \text{Enc}(0) & \text{Enc}(1) & \text{Enc}(0) \\
B_1 & B_2 & B_3 & B_4 \\
B_5 & B_6 & B_7 & B_8 \\
B_9 & B_{10} & B_{11} & B_{12} \\
B_{13} & B_{14} & B_{15} & B_{16} \\
\end{array}
\]
MulPIR: Improved Communication

Improve upload by up to 75% and download up to 80% with minimal computational increases over prior SealPIR implementation of Angel, Chen, Laine, Setty ‘18.
MulPIR: Improved Communication

Improve upload by up to 75% and download up to 80% with minimal computational increases over prior SealPIR implementation of Angel, Chen, Laine, Setty ‘18.

1. Use secret key encryption on client-side.
2. Replace long randomness with a short PRG seed.
3. Compress downloaded ciphertext using modulus switching.
4. Improved oblivious expansion.
5. Leverage multiplicative homomorphism.
Expandable Randomness using PRG

- Private key encryption is of the form \((c_0, c_1)\) where each element in \(R/qR\).
- \(c_0\) is a uniformly random element independent of public and private keys.
- Replace \(c_0\) with a PRG seed \(S\).
- Reduces upload by half already!
Improved Oblivious Expansion

SealPIR introduced the notion of oblivious expansion. Instead of a single ciphertext per bit, encrypt multiple bits per ciphertext.

**Without Oblivious Expansion:**

\[ E(0), \ldots, E(0), E(1), E(0), \ldots, E(0) \] with \( N \) ciphertexts

**With Oblivious Expansion:**

\[ E(0, \ldots, 0), E(0, \ldots, 0, 1, 0, \ldots, 0), E(0, \ldots, 0) \] with \( < N \) ciphertexts depending on parameters.

Server will obliviously expand compressed vector.
Limits of SealPIR's Oblivious Expansion

Limitation: To-be-compressed bit vector must have Hamming weight $\leq 1$. 
Limits of SealPIR’s Oblivious Expansion

Enc(0)  Enc(0)  Enc(1)  Enc(0)

$B_1$  $B_2$  $B_3$  $B_4$

$B_5$  $B_6$  $B_7$  $B_8$

$B_9$  $B_{10}$  $\boxed{B_{11}}$  $B_{12}$

$B_{13}$  $B_{14}$  $B_{15}$  $B_{16}$
Limits of SealPIR's Oblivious Expansion

Limitation: To-be-compressed bit vector must have Hamming weight $\leq 1$.

Limitation with Recursion: Must compress two sets of vectors separately. In the worst case, lots of wasted space.
Limits of SealPIR’s Oblivious Expansion

Limitation: To-be-compressed bit vector must have Hamming weight <= 1.

Limitation with Recursion: Must compress two sets of vectors separately. In the worst case, lots of wasted space.
Improved Oblivious Expansion

Improvement: To-be-compressed bit vector can have arbitrary Hamming weight.

Improvement with Recursion: Compress all uploaded bit vectors into one ciphertext.
Improved Oblivious Expansion

**Improvement**: To-be-compressed bit vector can have arbitrary Hamming weight.

**Improvement with Recursion**: Compress all uploaded bit vectors into one ciphertext.

Less Wasted Space
Improved Oblivious Expansion

Observation: Server oblivious expansion is linear in the plaintext space. Compression and expansion work for arbitrary vectors (not just bit vectors!).

Less Wasted Space
## Experimental Evaluation

Table 3: Communication and CPU costs (in ms) of SealPIR and MulPIR (recursion $d = 2$) for a database of $n$ elements of 288B.

<table>
<thead>
<tr>
<th>Database size $n$</th>
<th>SealPIR [3] ($d = 2$)</th>
<th>SealPIR [3] ($d = 3$)</th>
<th>MulPIR ($d = 2$)</th>
<th>MulPIR ($d = 3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual number of rows after packing</td>
<td>262144 1048576 4194304</td>
<td>262144 1048576 4194304</td>
<td>262144 1048576 4194304</td>
<td>262144 1048576 4194304</td>
</tr>
<tr>
<td>Client Query</td>
<td>19 19 19</td>
<td>19 19 19</td>
<td>172 192 213</td>
<td>126 128 161</td>
</tr>
<tr>
<td>Server Expand</td>
<td>145 294 590</td>
<td>33 55 90</td>
<td>391 783 1610</td>
<td>396 395 841</td>
</tr>
<tr>
<td>Server Respond</td>
<td>1020 3520 12891</td>
<td>1136 3519 11554</td>
<td>1919 5213 16307</td>
<td>3268 11677 30501</td>
</tr>
<tr>
<td>Upload (kB)</td>
<td>61.4 61.4 61.4</td>
<td>92.2 92.2 92.2</td>
<td>122 122 122</td>
<td>130 130 130</td>
</tr>
<tr>
<td>Download (kB)</td>
<td>307 307 307</td>
<td>1966 1966 1966</td>
<td>119 119 119</td>
<td>130 130 130</td>
</tr>
<tr>
<td>Server Cost (US cents)</td>
<td>0.0033 0.0040 <strong>0.0067</strong> 0.017 0.017 0.020 <strong>0.0026</strong> <strong>0.0036</strong> 0.0069 0.0031 0.0054 0.011</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Present improvements to Gentry-Ramzan PIR to enable tunable communication-computation trade-offs. Reduces server computation by up to 85% for larger communication sizes.
## Experimental Evaluation

### Table 5: Communication and computation costs for PIR protocols for two databases, without recursion.

<table>
<thead>
<tr>
<th></th>
<th># chunks</th>
<th>Communication (kB)</th>
<th>Computation (ms)</th>
<th>Server Cost (US cents)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>upload</td>
<td>download</td>
<td>C.Setup</td>
</tr>
<tr>
<td>1MB database: 5000 elements of 288B.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MulPIR</td>
<td>1</td>
<td>14</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>Gentry–Ramzan (1 generator)</td>
<td>5</td>
<td>0.5</td>
<td>1.3</td>
<td>0</td>
</tr>
<tr>
<td>Client-Aided Gentry–Ramzan (15 generators)</td>
<td>5</td>
<td>4.1</td>
<td>1.3</td>
<td>0</td>
</tr>
<tr>
<td>Client-Aided Gentry–Ramzan (50 generators)</td>
<td>5</td>
<td>13.1</td>
<td>1.3</td>
<td>0</td>
</tr>
<tr>
<td>Client-Aided Gentry–Ramzan (100 generators)</td>
<td>5</td>
<td>25.8</td>
<td>1.3</td>
<td>0</td>
</tr>
<tr>
<td>Damgård–Jurik (s = 1)</td>
<td>1</td>
<td>1,480</td>
<td>0.6</td>
<td>40,636</td>
</tr>
<tr>
<td>ElGamal</td>
<td>72</td>
<td>280</td>
<td>8</td>
<td>283</td>
</tr>
<tr>
<td>Private File Download – 3GB database: 10,000 elements of 307kB.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MulPIR</td>
<td>100</td>
<td>79.4</td>
<td>1,385</td>
<td>0</td>
</tr>
<tr>
<td>Client-Aided Gentry–Ramzan (50 generators)</td>
<td>4,955</td>
<td>13.1</td>
<td>1,259</td>
<td>6</td>
</tr>
<tr>
<td>Damgård–Jurik (s = 1)</td>
<td>1,060</td>
<td>2,960</td>
<td>614</td>
<td>≈ 80,000</td>
</tr>
<tr>
<td>ElGamal</td>
<td>76,800</td>
<td>280</td>
<td>4,300</td>
<td>≈ 300</td>
</tr>
</tbody>
</table>

Median over 10 computations. The timings indicated with ≈ have been estimated on a smaller number of chunks to finish in a reasonable amount of time.
Sparse PIR to (Dense) PIR Transformation

Generic transformation from Sparse PIR to (Dense) PIR.
What is Sparse PIR?
What is Sparse PIR?

Key $k_i$

$B_i$

$k_1$

$B_1$

$k_2$

$B_2$

$...$

$...$

$k_n$

$B_n$
Cuckoo Hashing

\[ k_1, k_2, \ldots, k_n \rightarrow B_1, B_2, \ldots, B_n \]
Sparse PIR using Dense PIR

Key $k_i$
Questions?