SimplePIR: Simple and Fast Single-Server Private Information Retrieval

Alexandra Henzinger
MIT
Matthew M. Hong
MIT
Henry Corrigan-Gibbs
MIT
Sarah Meiklejohn
Google
Vinod Vaikuntanathan
MIT

USENIX Security 2023
Private information retrieval [CGKS95, KO97]

Goal: Privately read an entry from a remote database

Correctness: learns $i$-th entry in $D$, if the server is honest

Privacy: learns nothing about $i$, even if the server is malicious

index $i \in \{1, \ldots, N\}$

database $D \in \{0,1\}^N$
Private information retrieval [CGKS95, KO97]

Goal: Privately read an entry from a remote database

\[ i \in \{1, \ldots, N\} \]

\[ D \in \{0,1\}^N \]

\[ \ll N \text{ communication} \]

[CMS99, BG16, DG16]

However, PIR inherently requires lots of server-side computation.

- Database read with no privacy: \( O(1) \) time
- Database read with privacy: \( \geq N \) time* (even with crypto) [BIM04, PY22]
25 years of work on single-server PIR

Server throughput (GB/s/core) vs. Year

- KO + Pailler
- SealPIR
- XPIR
- FastPIR
- OnionPIR
- MulPIR
- SimplePIR
- DoublePIR
- FrodoPIR
- Spiral

This talk

1. New tool: Fast linearly homomorphic encryption

2. Our results
   - SimplePIR: High throughput
   - DoublePIR: Less communication

3. Evaluation
Starting point: a classic PIR scheme [KO97]

wants entry at position $(i, j)$

$D \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}}$

 Shackles denotes linearly homomorphic encryption
Starting point: a classic PIR scheme [KO97]

wants entry at position \((i, j)\)

\[ D \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}} \]

\(\sqrt{N}\) denotes linearly homomorphic encryption
Starting point: a classic PIR scheme \([\text{KO97}]\)

A person wants entry at position \((i, j)\).

\[
\sqrt{N} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}
\]

\[
D \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}}
\]

\[
\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\]

\(\text{denotes linearly homomorphic encryption}\)
Starting point: a classic PIR scheme [KO97]

wants entry at position \((i,j)\)

\[\sqrt{N}\]

\[
\begin{array}{ccc}
0 & 1 & j \\
0 & 0 & 0 \\
\end{array}
\]

\[
D \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}}
\]

\[
\begin{array}{ccc}
0 & 1 & j \\
0 & 0 & 0 \\
\end{array}
\times
\]

\[=\]

\[\text{decrypts to get}
\]

\[\text{\(\text{\#} \text{ denotes linearly homomorphic encryption}\)}\]
Starting point: a classic PIR scheme [KO97]

- Wants entry at position $(i, j)$
- Decrypts to get

How fast can we multiply an encrypted vector by a plaintext matrix?

\[ D \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}} \times 0 1 0 0 0 0 0 0 0 0 1 j \]

\[ = \]

\[ \sqrt{N} \]

\[ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \end{bmatrix} \]

\( \text{\(\square\)}\) denotes linearly homomorphic encryption
New tool: Fast linearly homomorphic encryption

Regev encryption [Reg09] of a message $\mu \in \mathbb{Z}^{\sqrt{N}}$ with 1-byte entries:

$$\mu = \begin{pmatrix} A \\ 1024, \end{pmatrix} \begin{pmatrix} b \\ 1 \end{pmatrix}$$

Random, fixed matrix (32-bit entries)  Message dependent (32-bit entries)
New tool: Fast linearly homomorphic encryption

Regev encryption [Reg09] of a message $\mu \in \mathbb{Z}^{\sqrt{N}}$ with 1-byte entries:

\[
\begin{align*}
D \times \mu & = \\
D \times \begin{pmatrix} D \times A & D \times b \end{pmatrix} & = \\
& = D\mu,
\end{align*}
\]

Random, fixed matrix (32-bit entries)  
Message dependent (32-bit entries)
New tool: Fast linearly homomorphic encryption

Regev encryption [Reg09] of a message $\mu \in \mathbb{Z}^{\sqrt{N}}$ with 1-byte entries:

$$D \times \mu = \begin{pmatrix} D \times A \\ D \times b \end{pmatrix} = D \mu$$

Our observations:

- Precompute $D \times A$ once
  
  99.9% of the work

- The per-message work is $D \times b$
  
  one 32-bit mul per entry in $D$

$\Rightarrow$ Multiplying an encrypted vector with a plaintext matrix is cheap with preprocessing.
1. New tool: Fast linearly homomorphic encryption
2. Our results
   - SimplePIR: High throughput
   - DoublePIR: Less communication
3. Evaluation
SimplePIR: fast PIR with one-time preprocessing

Offline

\[ \sqrt{N} \times A \]

A

\[ \sqrt{N} \times D \times A \]

Hint

Online

wants entry at position \((i,j)\)

\[ b \]

1

\[ D b \]

1

\[ D \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}} \]
SimplePIR: fast PIR with one-time preprocessing

Offline

A

\[ D \times A \]

\[ \sqrt{N} \]

Hint

Online

wants entry at position \((i, j)\)

\[ b \]

\[ \sqrt{N} \]

\[ Db \]

\[ \sqrt{N} \]

\[ D \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}} \]

After downloading the hint, can make any number of queries cheaply!
**Result 1: SimplePIR**

Assuming LWE, we build single-server PIR where, on an \(N\)-byte database,
- the client downloads a one-time “hint” of size \(4096 \cdot \sqrt{N}\) bytes,
- the per-query communication is \(8 \cdot \sqrt{N}\) bytes, and
- the server performs \(N\) 32-bit adds and muls per query [10 GB/s/core].

**Result 2: DoublePIR**

On databases with 1-byte entries, we can shrink the hint to a constant 16 MB with a small decrease in throughput [7 GB/s/core].
Result 1: SimplePIR

Assuming LWE, we build single-server PIR where, on an $N$-byte database,

- the client downloads a one-time “hint” of size $4096 \cdot \sqrt{N}$ bytes,
- the per-query communication is $8 \cdot \sqrt{N}$ bytes, and
- the server performs $N$ 32-bit adds and muls per query [10 GB/s/core].

Result 2: DoublePIR

On databases with 1-byte entries, we can shrink the hint to a constant 16 MB with a small decrease in throughput [7 GB/s/core].
This talk

1. New tool: fast linearly homomorphic encryption

2. Our results
   - SimplePIR: High throughput
   - DoublePIR: Less communication

3. Evaluation
Implementation

Open-source code at github.com/ahenzinger/simplepir

- 1,600 lines of Go/C for SimplePIR and DoublePIR
- No external libraries
- For speed: SIMD instructions, optimized memory access locality

Experimental setup

- Server runs on an AWS c5n.metal instance
- All experiments on a 1 GB database, with a single thread of execution
SimplePIR is the fastest known single-server PIR

PIR costs on a 1 GB database, with entries of increasing size.
SimplePIR is almost as fast as two-server PIR

PIR costs on a 1 GB database, with entries of increasing size.
SimplePIR saves compute at the cost of communication.

PIR costs on a 1 GB database, with the optimal entry size.
With SimplePIR, PIR can be fast... what’s next?

1. Reduce the communication cost.

2. Deploy PIR to protect our privacy.

- This paper: DoublePIR to privately audit webpage certificates
- Blyss demo: DoublePIR to privately check for password compromise
- SOSP 2023: SimplePIR for private web search [HDCZ23]
With SimplePIR, PIR can be fast… what’s next?

1. Reduce the communication cost.

2. Deploy PIR to protect our privacy.

- This paper: DoublePIR to privately audit webpage certificates
- Blyss demo: DoublePIR to privately check for password compromise
- SOSP 2023: SimplePIR for private web search [HDCZ23]
- Almost as fast as 2-server PIR
- Moderate communication
- 1,600 lines of code

Server throughput (GB/s/core)

- KO + Pailler
- SealPIR
- XPIR
- FastPIR
- OnionPIR
- MulPIR
- SimplePIR
- DoublePIR
- FrodoPIR
- Spiral

Paper: eprint.iacr.org/2022/949
Code: github.com/ahenzinger/simplepir
Independent demo: playground.blyss.dev/passwords

Alexandra Henzinger
ahenz@csail.mit.edu