SPIRAL: Fast High-Rate Single-Server Private Information Retrieval

Samir Menon and David Wu
Private Information Retrieval (PIR)

Does not learn index $i$
Private Information Retrieval (PIR)

Our focus: single-server setting

Basic building block in many privacy-preserving protocols

- Metadata-private messaging
- Contact discovery
- Safe browsing
- Private DNS
- Private contact tracing
- Private navigation

record $i$

$r_1, r_2, \ldots, r_N$
Efficiency Metrics

1. **Query size**

2. **Server Throughput**

   \[
   \frac{\text{database size}}{\text{server computation time}}
   \]

   "measures how fast the server can respond as a function of database size"
Efficiency Metrics

1. **Query size**
   - query
   - response
   - Without preprocessing, server must perform a linear scan over the database

2. **Server Throughput**
   - database size
   - server computation time
   - “measures how fast the server can respond as a function of database size”
Efficiency Metrics

1. Query size
   - "measures communication overhead in responses"

2. Server Throughput
   - \[ \frac{\text{database size}}{\text{server computation time}} \]
   - "measures how fast the server can respond as a function of database size"

3. Rate
   - \[ \frac{\text{record size}}{\text{response size}} \]

4. Public parameter size

Client generates a **reusable** set of public parameters
The SPIRAL Family of PIR Protocols

Techniques to translate between FHE schemes enables new trade-offs in single-server PIR

Automatic parameter selection based on database configuration

<table>
<thead>
<tr>
<th>Base version of SPIRAL</th>
<th>Streaming versions of SPIRAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query size: 14 KB</td>
<td>Rate: 0.81</td>
</tr>
<tr>
<td>Rate: 0.41</td>
<td>Throughput: 1.9 GB/s</td>
</tr>
<tr>
<td>Throughput: 333 MB/s</td>
<td>Cost: 3.4× larger public parameters (17 MB)</td>
</tr>
</tbody>
</table>

4.5× smaller responses
2.1× higher
2.9× higher

(Database with $2^{14}$ records of size 100 KB)

Best previous protocol:
Rate: 0.24
Throughput: 158 MB/s

3.4× smaller responses
12.3× higher
The SPIRAL Family of PIR Protocols

Techniques to translate between FHE schemes enable new trade-offs in single-server PIR.

Automatic parameter selection based on database configuration.

**Base version of SPIRAL**
- **Query size:** 14 KB
- **Rate:** 0.41
- **Throughput:** 333 MB/s

(Database with $2^{14}$ records of size 100 KB)

**Cost:** 3.4× larger public parameters (17 MB)

**Streaming versions of SPIRAL**
- **Rate:** 0.81
- **Throughput:** 1.9 GB/s

3.4× smaller responses
12.3× higher

**Best previous protocol:**
- **Rate:** 0.24
- **Throughput:** 158 MB/s

Higher throughput than running software AES over database (Primary operation: 64-bit integer arithmetic)
The SPIRAL Family of PIR Protocols

Techniques to translate between FHE schemes enable new trade-offs in single-server PIR.

Automatic parameter selection based on database configuration.

**Base version of SPIRAL**

- **Query size:** 14 KB
- **Rate:** 0.41
- **Throughput:** 333 MB/s

(Database with $2^{14}$ records of size 100 KB)

**Cost:** 3.4× larger public parameters (17 MB)

**Streaming versions of SPIRAL**

- **Rate:** 0.81
- **Throughput:** 1.9 GB/s

3.4× smaller responses, 12.3× higher

**Best previous protocol:**

- **Rate:** 0.24
- **Throughput:** 158 MB/s

Cost of privately streaming a 2 GB movie from database of $2^{14}$ movies estimated to be 1.9× more expensive than no-privacy baseline (based on AWS compute costs).
Starting point: a $\sqrt{N}$ construction ($N =$ number of records)

Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix.
PIR from Homomorphic Encryption

Starting point: a $\sqrt{N}$ construction ($N =$ number of records)

Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

Encrypt a 0/1 vector indicating the row containing the desired record

Homomorphically compute product between query vector and database matrix
**PIR from Homomorphic Encryption**

**Starting point:** a $\sqrt{N}$ construction ($N = \text{number of records}$)

<table>
<thead>
<tr>
<th>$r_{11}$</th>
<th>$r_{12}$</th>
<th>$r_{13}$</th>
<th>$r_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_{21}$</td>
<td>$r_{22}$</td>
<td>$r_{23}$</td>
<td>$r_{24}$</td>
</tr>
<tr>
<td>$r_{31}$</td>
<td>$r_{32}$</td>
<td>$r_{33}$</td>
<td>$r_{34}$</td>
</tr>
<tr>
<td>$r_{41}$</td>
<td>$r_{42}$</td>
<td>$r_{43}$</td>
<td>$r_{44}$</td>
</tr>
</tbody>
</table>

Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

Encrypt a 0/1 vector indicating the row containing the desired record

$\begin{align*}
    &\times 0 + \\
    &\times 0 + \\
    &\times 1 + \\
    &\times 0 + \\
\end{align*}$

Database is in the clear, so *additive* homomorphism suffices
PIR from Homomorphic Encryption [KO97]

Starting point: a $\sqrt{N}$ construction ($N =$ number of records)

Client decrypts to learn records

Encrypt a 0/1 vector indicating the row containing the desired record

Response size: $\sqrt{N} \cdot \text{poly}(\lambda)$

Homomorphically compute product between query vector and database matrix
PIR from Homomorphic Encryption

Starting point: a $\sqrt{N}$ construction ($N =$ number of records)

Client decrypts to learn records

Encrypt a 0/1 vector indicating the row containing the desired record

Response size: $\sqrt{N} \cdot \text{poly}(\lambda)$

Homomorphically compute product between query vector and database matrix

(KO97)
PIR from Homomorphic Encryption

Beyond $\sqrt{N}$ communication: view the database as **hypercube**

**Approach:** Use homomorphic multiplication

Gentry-Halevi [GH19]
OnionPIR [MCR21]
SPIRAL: Composing FHE Schemes

Follows Gentry-Halevi blueprint of composing two lattice-based FHE schemes:

FHE ciphertexts are noisy encodings
Homomorphic operations increase noise; more noise = larger parameters = less efficiency

Scheme 1: Regev’s encryption scheme [Reg04]
    High-rate; only supports additive homomorphism

Scheme 2: Gentry-Sahai-Waters encryption scheme [GSW13]
    Low rate; supports homomorphic multiplication (with additive noise growth)

Goal: get the best of both worlds
Regev encoding of a scalar $m \in R$:

- Secret key allows recovery of noisy version of original message
- To support decryption of “small” values $t \in R_p$, we encode $t$ as $(q/p)t$
- Decryption recovers noisy version of $(q/p)t$ and rounding yields $t$

$$rate = \frac{\log p}{2 \log q} < \frac{1}{2}$$

OnionPIR: rate = 0.24

All elements are polynomials in the ring $R = \mathbb{Z}[x]/(x^d + 1)$ where $d = 2^k$
Matrix Regev Encodings (over Rings)

**Idea:** “Reuse” encryption randomness

rate = \( \frac{n^2 \log p}{n(n+1) \log q} = \frac{n^2}{n^2 + n \log q} \)

Additively homomorphic:

\( S^T C_1 \approx M_1 \)
\( S^T C_2 \approx M_2 \)
\( S^T (C_1 + C_2) \approx M_1 + M_2 \)

All elements are polynomials in the ring \( R = \mathbb{Z}[x]/(x^d + 1) \) where \( d = 2^k \)
**Gentry-Sahai-Waters Encodings**

**GSW encoding** of a bit $\mu \in \{0,1\}$:

$$R_q^{n \times (n+1)} \quad R_q^{(n+1) \times n} \quad \approx \quad \begin{array}{c} \mu \end{array} \quad R_q^{n \times (n+1)} \quad R_q^{(n+1) \times m}$$

**Gadget matrix** \([MP12]\):

$$G = \begin{bmatrix} g^T & \ddots & \end{bmatrix}$$

$$g^T = \begin{bmatrix} 1 & 2 & 2^2 & \cdots & 2^{\lfloor \log_2 q \rfloor} \end{bmatrix}$$

“Powers-of-2” matrix

$$m = (n + 1) \log q$$

Construction will use other decomposition bases

All elements are polynomials in the ring $R = \mathbb{Z}[x]/(x^d + 1)$ where $d = 2^k$
GSW encoding of a bit $\mu \in \{0, 1\}$:

$$G \approx \begin{bmatrix} S^T & C \end{bmatrix} \begin{bmatrix} R_q^{n \times (n+1)} & R_q^{(n+1) \times n} \end{bmatrix} \begin{bmatrix} \mu \end{bmatrix} \begin{bmatrix} R_q^{n \times (n+1)} & R_q^{(n+1) \times m} \end{bmatrix}$$

**Main property:** for every vector $\mathbf{v} \in \mathbb{Z}_q^{n+1}$, can define $G^{-1}(\mathbf{v}) \in \{0, 1\}^m$ where $GG^{-1}(\mathbf{v}) = \mathbf{v}$

"binary decomposition"

All elements are polynomials in the ring $R = \mathbb{Z}[x]/(x^d + 1)$ where $d = 2^k$
Gentry-Sahai-Waters Encodings

**GSW encoding** of a bit $\mu \in \{0,1\}$:

\[
R_q^{n \times (n+1)} R_q^{(n+1) \times n} \approx \begin{bmatrix} S^T & C \end{bmatrix} \mu S^T G
\]

**Gadget matrix** [MP12]:

\[
G = \begin{bmatrix} g^T & \vdots & g^T \end{bmatrix}
\]

\[
g^T = [1, 2, 2^2, \ldots, 2^{\lfloor \log_2 q \rfloor}]
\]

"Powers-of-2" matrix

**Construction will use other decomposition bases**

**Rate**: $\frac{1}{d(n+1)^2 \log q}$

**Concretely**: $d = 2048, n \geq 1, q = 2^{56}$

All elements are polynomials in the ring $R = \mathbb{Z}[x]/(x^d + 1)$ where $d = 2^k$
Regev-GSW Homomorphism

\[ S^T C_{\text{Reg}} \approx M \]

\[ S^T C_{\text{GSW}} \approx \mu S^T G \]

\[ S^T C_{\text{GSW}} G^{-1}(C_{\text{Reg}}) \approx \mu S^T C_{\text{Reg}} \approx \mu M \]

With noise terms:
\[ S^T C_{\text{GSW}} G^{-1}(C_{\text{Reg}}) = \mu M + E_{\text{GSW}} G^{-1}(C_{\text{Reg}}) + \mu E_{\text{Reg}} \]

**Asymmetric noise growth:** if all GSW ciphertexts are “fresh,” then noise accumulation is additive in the number of multiplications

\[ C_{\text{GSW}} G^{-1}(C_{\text{Reg}}) \] is a Regev encoding of \( \mu M \)
The Gentry-Halevi Blueprint

Database is represented as $2^{\nu_1} \times 2 \times 2 \times \cdots \times 2$ hypercube

Query contains $2^{\nu_1}$ matrix Regev ciphertexts

Indicator for index along first dimension

Query contains $\nu_2$ GSW ciphertexts

Indicator for index along subsequent dimensions

Response is a single matrix Regev ciphertext

Each GSW ciphertext participates in only one multiplication with a Regev ciphertext!
The Gentry-Halevi Blueprint

Database is represented as $2^{\nu_1} \times 2 \times 2 \times \cdots \times 2$ hypercube

**Drawback:** large queries

Can compress using polynomial encoding method of Angel et al. [ACLS18]

Estimated size: 4 MB/ciphertext

Estimated query size: 30 MB

Query contains $2^{\nu_1}$ matrix Regev ciphertexts

Indicator for index along first dimension

Query contains $\nu_2$ GSW ciphertexts

Indicator for index along subsequent dimensions
The Gentry-Halevi Blueprint

Database is represented as $2^{\nu_1} \times 2 \times 2 \times \cdots \times 2$ hypercube

**Drawback:** large queries

Can compress using polynomial encoding method of Angel et al. [ACLS18]

Query contains $2^{\nu_1}$ matrix Regev ciphertexts

Indicator for index along first dimension

Query contains $\nu_2$ GSW ciphertexts

Indicator for index along subsequent dimensions

SealPIR query size: 66 KB

Estimated query size: 30 MB

[GH19]
**High-level:** Gentry-Halevi approach with *scalar* Regev ciphertexts \((n = 1)\)

Leverages Chen et al. approach \([CCR19]\) to “assemble” GSW ciphertext using Regev-GSW multiplication

Regev ciphertexts can be packed using polynomial encoding method \([ACLS18, CCR19]\)

Use of scalar Regev ciphertexts reduces the rate to \(\approx 0.24\)
(over 4× response overhead)
“Best of both worlds”: Small queries (as in OnionPIR) with the high rate/throughput of the Gentry-Halevi scheme

<table>
<thead>
<tr>
<th>Query size:</th>
<th>14 KB</th>
<th>2000× smaller than Gentry-Halevi (4.5× smaller than OnionPIR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate:</td>
<td>0.41</td>
<td>2.1× higher than OnionPIR</td>
</tr>
<tr>
<td>Throughput:</td>
<td>333 MB/s</td>
<td>2.9× higher than OnionPIR</td>
</tr>
</tbody>
</table>

(Database with $2^{14}$ records of size 100 KB)

Cost: 3.4× larger public parameters for extra translation keys

Leverage simple key-switching techniques for query and response compression

Scalar Regev → Matrix Regev
Matrix Regev → GSW

Query compression

Scalar Regev → Matrix Regev
Response compression
(for large records)
Scalar Regev $\rightarrow$ Matrix Regev

**Input:** encoding $c$ where $s_1^T c \approx m$

**Output:** encoding $C$ where $S_2^T C \approx mI_n$

\[ S_2^T = \begin{bmatrix} -\tilde{s}_0 & \cdots & -\tilde{s}_0 \end{bmatrix}, \quad I_n \]

\[ S_2^T C = mI_n \]

\[ c^T = [c_0 \mid c_1] \in R_q^2 \]

Can replace with $S_2$ with arbitrary secret key using standard key-switching techniques.
Goal: use Regev encodings to construct $C$ such that $S^T C \approx \mu S^T G$

$S^T = [-s \mid I_n] \in R^{n \times (n+1)}$

$G = \begin{bmatrix} g^T \\ \vdots \\ g^T \end{bmatrix}$

$\mu S^T G = \begin{bmatrix} -\mu s g^T \\ \mu I_n \\ 2\mu I_n \\ 2^2 \mu I_n \\ \vdots \\ 2^t \mu I_n \end{bmatrix}$

$C = \begin{bmatrix} A \\ B_0 \\ B_1 \\ B_2 \\ \vdots \\ B_t \end{bmatrix}$

Break $C$ into blocks
Matrix Regev → GSW

Goal: use Regev encodings to construct $C$ such that $S^T C \approx \mu S^T G$

$S^T C = S^T A S^T B_0 S^T B_1 S^T B_2 \ldots S^T B_t$

$\mu S^T G = -\mu S g^T \mu I_n 2\mu I_n 2^2 \mu I_n \ldots 2^t \mu I_n$

$B_0, \ldots, B_t$ are matrix Regev ciphertexts encrypting $\mu I_n, 2\mu I_n, \ldots, 2^t \mu I_n$

Can derive from scalar Regev encodings of $\mu, 2\mu, \ldots, 2^t \mu$
Goal: use Regev encodings to construct $C$ such that $S^T C \approx \mu S^T G$

Write $S^T = [-s | I_n]$

Let $s_{\text{Reg}}$ be the key for a Regev encoding scheme

Construct key-switching matrix $W$:

$$S^T W \approx -s \left( s_{\text{Reg}}^T \otimes g^T \right)$$

Let $c_0, \ldots, c_t$ be encodings of $\mu, \ldots, 2^t \mu$ under $s_{\text{Reg}}$: $s_{\text{Reg}}^T c_i \approx 2^i \mu$

Let $C = [c_0 \mid \ldots \mid c_t]$

Then, $S^T W g^{-1}(C) \approx -s \left( s_{\text{Reg}}^T \otimes g^T \right) g^{-1}(C) \approx -s[\mu \mid \ldots \mid 2^t \mu] = -\mu s g^T$
Matrix Regev → GSW

**Goal:** use Regev encodings to construct $C$ such that $S^T C \approx \mu S^T G$

\[
S^T C = \begin{bmatrix} S^T A & S^T B_0 & S^T B_1 & \cdots & S^T B_t \end{bmatrix}
\]

\[
\mu S^T G = \begin{bmatrix} -\mu s g^T & \mu I_n & 2\mu I_n & 2^2 \mu I_n & \cdots & 2^t \mu I_n \end{bmatrix}
\]

Write $S^T = [-s \mid I_n]$

Let $s_{\text{Reg}}$ be the key for a Regev encoding scheme

Construct key-switching matrix $W$:

\[
S^T W \approx -s \left( s_{\text{Reg}}^T \otimes g^T \right)
\]

Let $c_0, \ldots, c_t$ be encodings of $\mu, \ldots, 2^t \mu$ under $s_{\text{Reg}}$: $s_{\text{Reg}}^T c_i \approx 2^i \mu$

Define $A = W g^{-1}(C)$

Then, $S^T W g^{-1}(C) \approx -s \left( s_{\text{Reg}}^T \otimes g^T \right) g^{-1}(C) \approx -s [\mu \mid \cdots \mid 2^t \mu] = -\mu s g^T$
Matrix Regev $\rightarrow$ GSW

Scalar Regev to Matrix Regev

$Wg^{-1}([c_0| \cdots |c_t])$

Concatenate blocks to obtain GSW encoding of $\mu$

$\begin{align*}
-\mu sg^T \\
\mu I_n \\
\vdots \\
2^t \mu I_n
\end{align*}$

Ciphertext contains $(n+1)^2(t+1)$ elements of $R_q$

Takeaway: instead of sending $(n+1)^2(t+1)$ ring elements per GSW ciphertext, only need to send $2(t+1)$

scalar Regev encodings: elements of $R_q^2$

matrix Regev encodings: elements of $R_q^{(n+1)\times n}$
### Further Compression via Polynomial Encodings

[ACLS18, CCR19]: let \( f(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_t \cdot x^t \) with \( t < d \)

| \( c_0 \) | \( \mu \) |
| \( c_1 \) | \( 2\mu \) |
| \( \vdots \) | \( \vdots \) |
| \( c_t \) | \( 2^t \mu \) |

Expands a Regev encoding of a polynomial into Regev encodings of its coefficients

**Takeaway:** We can pack \((\mu, 2\mu, \ldots, 2^t \mu)\) into a single polynomial

As long as \( t + 1 < d \), client and communicate a GSW ciphertext with a single Regev encoding (2 ring elements)

**Cost:** additional (reusable) public parameters needed for Regev-to-GSW translation

\((n+1)^2(t+1)\) ring elements

2 ring elements
Query Expansion in Spiral

Database is represented as $2^{\nu_1} \times \underbrace{2 \times 2 \times \cdots \times 2}_{2^{\nu_2}}$ hypercube

Query contains $2^{\nu_1}$ matrix Regev ciphertexts

Indicator for index along first dimension

Query contains $\nu_2$ GSW ciphertexts

Indicator for index along subsequent dimensions

1. Compress into scalar Regev encodings

2. Pack scalars into single polynomial

Compress into scalar

Pack scalars into single polynomial
Query Expansion in Spiral

Trade-off: larger public parameters, smaller queries

- **SealPIR:** 3 MB
- **OnionPIR:** 5 MB
- **SPIRAL:** 18 MB
- **SealPIR:** 66 KB
- **OnionPIR:** 63 KB
- **Gentry-Halevi:** \(\approx\) 30 MB
- **SPIRAL:** 14 KB

Moving costs from online to offline phase

**SPIRAL** also achieves higher rate and throughput
Response Compression via Modulus Switching

PIR response consists of a single matrix Regev encoding

\[
\begin{pmatrix}
-s \\ I_n
\end{pmatrix} \approx \frac{q}{p} M
\]

Modulus \( q \) must be large enough to support target number of homomorphic operations

\[
\text{rate} \propto \frac{\log p}{\log q}
\]

Standard technique in FHE: \textit{modulus reduction}

Rescale ciphertext by \( \frac{q'}{q} \) where \( q' < q \)

\[
\text{rate} \propto \frac{\log p}{\log q'}
\]

Rescaling introduces small amount of noise (from rounding)

\textbf{This work:} Observe that rounding error \( E \) is scaled by \( \begin{pmatrix} -s \\ I_n \end{pmatrix} \)
Response Compression via Modulus Switching

PIR response consists of a single matrix Regev encoding

\[
\begin{bmatrix}
-s \\
I_n
\end{bmatrix} C \approx \frac{q}{p} \cdot M
\]

Modulus \( q \) must be large enough to support target number of homomorphic operations

\[
\text{rate} \propto \frac{\log p}{\log q}
\]

Standard technique in FHE: *modulus reduction*

Rescale ciphertext by \( \frac{q'}{q} \) where \( q' < q \)

\[
\text{rate} \propto \frac{\log p}{\log q'}
\]

Rescaling introduces small amount of noise (from rounding)

**This work:** Observe that rounding error \( E \) is scaled by \( [-s I_n] \)

Error scaled by \( -s \)

Error scaled by \( I_n \)
PIR response consists of a single matrix Regev encoding

\[
\begin{bmatrix} -s & I_n \end{bmatrix} C \approx \frac{q}{p} \cdot M
\]

Observation: At least half of the error components are scaled by identity matrix!

Approach: Use two different moduli to rescale the ciphertext

Standard technique in FHE: \textit{modulus reduction}

Rescale ciphertext by \( \frac{q'}{q} \) where \( q' < q \)

rate \( \propto \frac{\log p}{\log q'} \)

Rescaling introduces small amount of noise (from rounding)

This work: Observe that rounding error \( E \) is scaled by \( [-s \mid I_n] \)

Error scaled by \( -s \)

Error scaled by \( I_n \)
PIR response consists of a single matrix Regev encoding

Standard technique in FHE: *modulus reduction*

Rescale ciphertext by \( \frac{q'}{q} \) where \( q' < q \)

rate \( \propto \frac{\log p}{\log q'} \)

Rescaling introduces small amount of noise (from rounding)

**Observation:** At least half of the error components are scaled by identity matrix!

**Approach:** Use two different moduli to rescale the ciphertext

**This work:** Observe that rounding error \( E \) is scaled by \([ -s \mid I_n ]\)

\( [-s \mid I_n] \) Error scaled by \(-s\)

\( e_0^T \) Error scaled by \( I_n \)
Response Compression via Modulus Switching

PIR response consists of a single matrix Regev encoding

\[ C = \begin{pmatrix} c_0^T \\ c_1 \end{pmatrix} \]

Rescale by \( q_2 / q \)

Rescale by \( q_1 / q \)

Observation: At least half of the error components are scaled by identity matrix!

Approach: Use two different moduli to rescale the ciphertext

rate = \( \frac{n^2 \log p}{n^2 \log q_1 + n \log q_2} \)

- SealPIR: 0.01
- Gentry-Halevi (estimated): 0.44
- OnionPIR: 0.24

Overall rate: 0.34 (with vanilla modulus switching)
0.81 (with split modulus switching)

This work: Observe that rounding error \( E \) is scaled by \( [−s \mid I_n] \)

Error scaled by \( −s \)

Error scaled by \( I_n \)
Vanilla SPIRAL

record $i$

public parameters

Key-switching matrices for ciphertext expansion and translation
Vanilla SPIRAL

public parameters

query

Single scalar Regev encoding of a polynomial

Homomorphic expansion

\[ 0 \quad I_n \quad 0 \quad 0 \]

\[ 0 \quad 1 \quad 1 \]
Vanilla SPIRAL

public parameters
query

Homomorphic expansion

Regev encodings for first dimension
GSW encodings for subsequent dimensions

First dimension processing
Regev-GSW folding
Vanilla SPIRAL

Many parameter choices in SPIRAL:
- Plaintext matrix dimension
- Plaintext modulus
- Decomposition bases for key-switching
- Database arrangement

Trade-offs in public parameter size, query size, server throughput, and rate

Use estimated running time + compute cost to choose parameters for an input database configuration

Automatic parameter selection tool
# Basic Comparisons

<table>
<thead>
<tr>
<th>Database Configuration</th>
<th>Public Param. Size</th>
<th>Query Size</th>
<th>Response Size</th>
<th>Server Compute</th>
<th>Rate</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^{18} records 30 KB records (7.9 GB database)</td>
<td>3 MB</td>
<td>66 KB</td>
<td>3 MB</td>
<td>74.91 s</td>
<td>0.24</td>
<td>149 MB/s</td>
</tr>
<tr>
<td>2^{18} records 50 KB records (15.8 GB database)</td>
<td>5 MB</td>
<td>63 KB</td>
<td>5 MB</td>
<td>52.7 s</td>
<td>0.36</td>
<td>322 MB/s</td>
</tr>
</tbody>
</table>

Database configuration preferred by OnionPIR:

- reduce query size by 4.5×
- reduce response size by 2×
- reduce compute time by 2×
- increase public parameter size by 3.6×
Basic Comparisons (with Larger Records)

Throughput for 100 GB database ($2^{20}$ records):

- **SPIRAL**: 310 MB/s (322 s)
- **SealPIR**: 102 MB/s (977 s)
- **FastPIR**: 189 MB/s (528 s)
- **OnionPIR**: 122 MB/s (817 s)

SPIRAL also has smaller query size and response size, but larger public parameters.

All measurements based on single-thread/single-core processing.

Server cost is linear in database size.
**Basic Comparisons (with Larger Records)**

**Client costs:**
- Generating reusable public parameters is the most expensive operation, but still $< 1$ s
- Query generation and response decoding are fast (30 ms and $< 1$ ms)

**Server costs:**
- Query expansion typically takes $\approx 1$ second (less than 1.5% of overall compute when number of records is large)
- Parameter selection favors configurations that evenly distributes the work between first layer processing and ciphertext folding

(see paper for detailed microbenchmarks)
The Streaming Setting: SPIRAL STREAM

**Streaming setting:** same query reused over multiple databases

- Private video stream (database $D_i$ contains $i^{th}$ block of media) [GCMSAW16]
- Private voice calls (repeated polling of the same “mailbox”) [AS16, AYAAG21]

**Goal:** minimize online costs (i.e., server compute, response size)

Consider larger public parameters or query size (amortized over lifetime of stream)

- **Matrix Regev encodings**
  
  \[
  \begin{array}{cccc}
  0 & 1 & 0 & 0 \\
  \end{array}
  \xrightarrow{f}
  \begin{array}{cccc}
  0 & \mathbb{I}_n & 0 & 0 \\
  \end{array}
  \]

- **GSW encodings**
  
  \[
  \begin{array}{cccc}
  0 & 0 & 0 & 0 \\
  \end{array}
  \xrightarrow{1, z, z^2, z^3}
  \begin{array}{cccc}
  0 & 1 & 1 \\
  \end{array}
  \]

**SPIRAL query expansion**
The Streaming Setting: SPIRALSTREAM

Removing the initial expansion **significantly** reduces the noise growth from query expansion

- Decreases size of public parameters (no more automorphism keys)
- Better control of noise growth ⇒ higher server throughput and higher rate
- Larger queries (more Regev encodings)

### SPIRALSTREAM query

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>z</td>
<td>z^2</td>
<td>z^3</td>
</tr>
<tr>
<td>1</td>
<td>z</td>
<td>z^2</td>
<td>z^3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Matrix Regev encodings

| 0 | I_n | 0 | 0 |

GSW encodings

| 0 | 1 | 1 |

**SPIRALSTREAM query expansion**
# The Streaming Setting: SPIRALSTREAM

<table>
<thead>
<tr>
<th>Database</th>
<th>Metric</th>
<th>OnionPIR</th>
<th>SPIRAL</th>
<th>SPIRALSTREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{18}$ records 30 KB records (7.9 GB database)</td>
<td>Public Param. Size</td>
<td>5 MB</td>
<td>18 MB</td>
<td>3 MB</td>
</tr>
<tr>
<td></td>
<td>Query Size</td>
<td>63 KB</td>
<td>14 KB</td>
<td>15 MB</td>
</tr>
<tr>
<td></td>
<td>Response Size</td>
<td>127 KB</td>
<td>84 KB</td>
<td>62 KB</td>
</tr>
<tr>
<td></td>
<td>Server Compute</td>
<td>52.7 s</td>
<td>24.5 s</td>
<td>9.0 s</td>
</tr>
<tr>
<td></td>
<td><strong>Rate:</strong></td>
<td>0.23</td>
<td>0.36</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td><strong>Throughput:</strong></td>
<td>149 MB/s</td>
<td>322 MB/s</td>
<td>874 MB/s</td>
</tr>
</tbody>
</table>

- 25% reduction in response size
- 2.7× increase in throughput
The Streaming Setting: SPIRALSTREAM

Streaming throughput: ignoring query expansion costs, assuming optimal record size for each system

Peaks at \( \approx 1.5 \text{ GB/s} \) (over \( 7 \times \) faster than previous constructions)
Higher Rate via Response Packing: SPIRALPACK

Can we further reduce response size?

\[
\text{rate} = \frac{n^2 \log p}{n \log q_2 + n^2 \log q_1} \quad q_1 = 4p
\]

Increasing the plaintext dimension \( n \) increases the rate

**SPIRAL** and **SPIRALSTREAM** use \( n = 2 \)

Higher values of \( n \) increases computational cost

Each Regev encoding is a \((n + 1) \times n\) matrix, so number of ring operations per homomorphic operation scale with \( O(n^3) \) [Not using fast matrix multiplications here]

**SPIRALPACK**: Perform homomorphic operations with \( n = 1 \) and pack responses
Higher Rate via Response Packing: SPIRALPACK

Plaintext space: $R_p^{n \times n}$

Each record is $n \times n$ matrix

Split database into $n^2$ databases

$i^{th}$ database contains $i^{th}$ entry of record (elements of $R_p$)

Response consists of $n^2$ Regev encodings

Better throughput

Worse rate
Higher Rate via Response Packing: SPIRALPACK

Variant of scalar Regev to matrix Regev transformation

Requires publishing $n$ key-switching matrices

Packing done only at the very end (cost does not scale with number of records)

$n^2$ Regev ciphertexts with dimension 1

Consists of $2n^2$ ring elements

$1$ Regev ciphertext with dimension $n$

Consists of $n(n + 1)$ ring elements

SPIRALPACK: higher throughput and rate (for sufficiently large records), larger public parameters
Higher Rate via Response Packing: SPIRALPACK

<table>
<thead>
<tr>
<th>Database</th>
<th>Metric</th>
<th>OnionPIR</th>
<th>SPIRAL</th>
<th>SPIRALSTREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{18}$ records</td>
<td>Public Param. Size</td>
<td>5 MB</td>
<td>18 MB → 18 MB</td>
<td>3 MB → 16 MB</td>
</tr>
<tr>
<td>30 KB records</td>
<td>Query Size</td>
<td>63 KB</td>
<td>14 KB → 14 KB</td>
<td>15 MB → 30 MB</td>
</tr>
<tr>
<td>(7.9 GB database)</td>
<td>Response Size</td>
<td>127 KB</td>
<td>84 KB → 86 KB</td>
<td>62 KB → 96 KB</td>
</tr>
<tr>
<td></td>
<td>Server Compute</td>
<td>52.7 s</td>
<td>24.5 s → 17.7 s</td>
<td>9.0 s → 5.3 s</td>
</tr>
</tbody>
</table>

- Small records ⇒ can only take advantage of low packing dimension
- Higher throughputs since homomorphic operations cheaper
- Responses larger due to extra noise from response packing
Higher Rate via Response Packing: SPIRALPACK

<table>
<thead>
<tr>
<th>Database Size</th>
<th>Metric</th>
<th>OnionPIR</th>
<th>SPIRAL</th>
<th>SPIRALSTREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{18}$</td>
<td>Public Param. Size</td>
<td>5 MB</td>
<td>18 MB → 18 MB</td>
<td>3 MB → 16 MB</td>
</tr>
<tr>
<td>30 KB records</td>
<td>Query Size</td>
<td>63 KB</td>
<td>14 KB → 14 KB</td>
<td>15 MB → 30 MB</td>
</tr>
<tr>
<td>(7.9 GB</td>
<td>Response Size</td>
<td>127 KB</td>
<td>84 KB → 86 KB</td>
<td>62 KB → 96 KB</td>
</tr>
<tr>
<td>Database</td>
<td>Server Compute</td>
<td>52.7 s</td>
<td>24.5 s → 17.7 s</td>
<td>9.0 s → 5.3 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database Size</th>
<th>Metric</th>
<th>OnionPIR</th>
<th>SPIRAL</th>
<th>SPIRALSTREAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{14}$</td>
<td>Public Param. Size</td>
<td>5 MB</td>
<td>17 MB → 47 MB</td>
<td>1 MB → 24 MB</td>
</tr>
<tr>
<td>100 KB records</td>
<td>Query Size</td>
<td>63 KB</td>
<td>14 KB → 14 KB</td>
<td>8 MB → 30 MB</td>
</tr>
<tr>
<td>(1.6 GB</td>
<td>Response Size</td>
<td>508 KB</td>
<td>242 KB → 188 KB</td>
<td>208 KB → 150 KB</td>
</tr>
<tr>
<td>Database</td>
<td>Server Compute</td>
<td>14.4 s</td>
<td>4.92 s → 4.58 s</td>
<td>2.4 s → 1.2 s</td>
</tr>
</tbody>
</table>

| Rate:         | 0.20                    | 0.41 → 0.53 | 0.48 → 0.67 |
| Throughput:   | 114 MB/s                | 333 MB/s → 358 MB/s | 683 MB/s → 1.4 GB/s |

With 100 KB records, higher rate **and** throughput in exchange for larger public parameters
Streaming throughput: ignoring query expansion costs, assuming optimal record size for each system.
Packing in the Streaming Setting

**Streaming throughput:** ignoring query expansion costs, assuming optimal record size for each system

Packing outperforms non-packed protocol for streaming settings

1.94 GB/s and a rate of 0.81 (125 MB public parameter and 30 MB query)

Memory bandwidth on system: $\approx 10$ GB/s

---

**Diagram:**

- **Spiral**
- **Spiral Pack**
- **Spiral Stream**
- **Spiral Stream Pack**
- **Seal PIR**
- **Fast PIR**
- **Onion PIR**
Packing in the Streaming Setting

**Streaming throughput:** ignoring query expansion costs, assuming optimal record size for each system

Cost of privately streaming a 2 GB movie from database of $2^{14}$ movies estimated to be 1.9× more expensive than no-privacy baseline (based on AWS compute costs)

Previously: $\approx 17\times$ more expensive

Packing outperforms non-packed protocol for streaming settings
A Systematic Way to Explore PIR Trade-Offs

Parameter selection tool can be used to minimize online cost with constraints on public parameter and query sizes

(Database configuration: $2^{14} \times 100$ KB database)
The SPIRAL Family of PIR

Techniques to translate between FHE schemes enables new trade-offs in single-server PIR

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar Regev → Matrix Regev</td>
<td>Query compression</td>
</tr>
<tr>
<td>Regev → GSW</td>
<td>Response compression (for large records)</td>
</tr>
</tbody>
</table>

Automatic parameter selection to choose lattice parameters based on database configuration

**Base version of SPIRAL**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query size</td>
<td>14 KB</td>
<td>4.5× smaller</td>
</tr>
<tr>
<td>Rate</td>
<td>0.41</td>
<td>2.1× higher</td>
</tr>
<tr>
<td>Throughput</td>
<td>333 MB/s</td>
<td>2.9× higher</td>
</tr>
</tbody>
</table>

(Database with $2^{14}$ records of size 100 KB)

**Streaming versions of SPIRAL**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate</td>
<td>0.81</td>
<td>3.4× smaller</td>
</tr>
<tr>
<td>Throughput</td>
<td>1.9 GB/s</td>
<td>12.3× higher</td>
</tr>
</tbody>
</table>

Improvements primarily due to query and response compression
## The SPIRAL Family of PIR

Techniques to translate between FHE schemes enables new trade-offs in single-server PIR

- **Scalar Regev → Matrix Regev**
- **Regev → GSW**

- **Query compression**
- **Response compression**
  (for large records)

Automatic parameter selection to choose lattice parameters based on database configuration

### Base version of SPIRAL

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query size</td>
<td>14 KB</td>
<td>4.5× smaller</td>
</tr>
<tr>
<td>Rate</td>
<td>0.41</td>
<td>2.1× higher</td>
</tr>
<tr>
<td>Throughput</td>
<td>333 MB/s</td>
<td>2.9× higher</td>
</tr>
</tbody>
</table>

(Database with $2^{14}$ records of size 100 KB)

### Streaming versions of SPIRAL

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate</td>
<td>0.81</td>
<td>3.4× smaller</td>
</tr>
<tr>
<td>Throughput</td>
<td>1.9 GB/s</td>
<td>12.3× higher</td>
</tr>
</tbody>
</table>

Improvements primarily due to fine-tuning scheme parameters for database configuration
Future Directions

Leveraging FHE composition in other privacy-preserving systems

- Private set intersection (PSI)
- Oblivious RAM (ORAM)

Hardware acceleration for higher throughput

Leveraging preprocessing to achieve *sublinear* server computation

**Paper:** [https://eprint.iacr.org/2022/368](https://eprint.iacr.org/2022/368)

**Code:** [https://github.com/menonsamir/spiral](https://github.com/menonsamir/spiral)

Thank you!