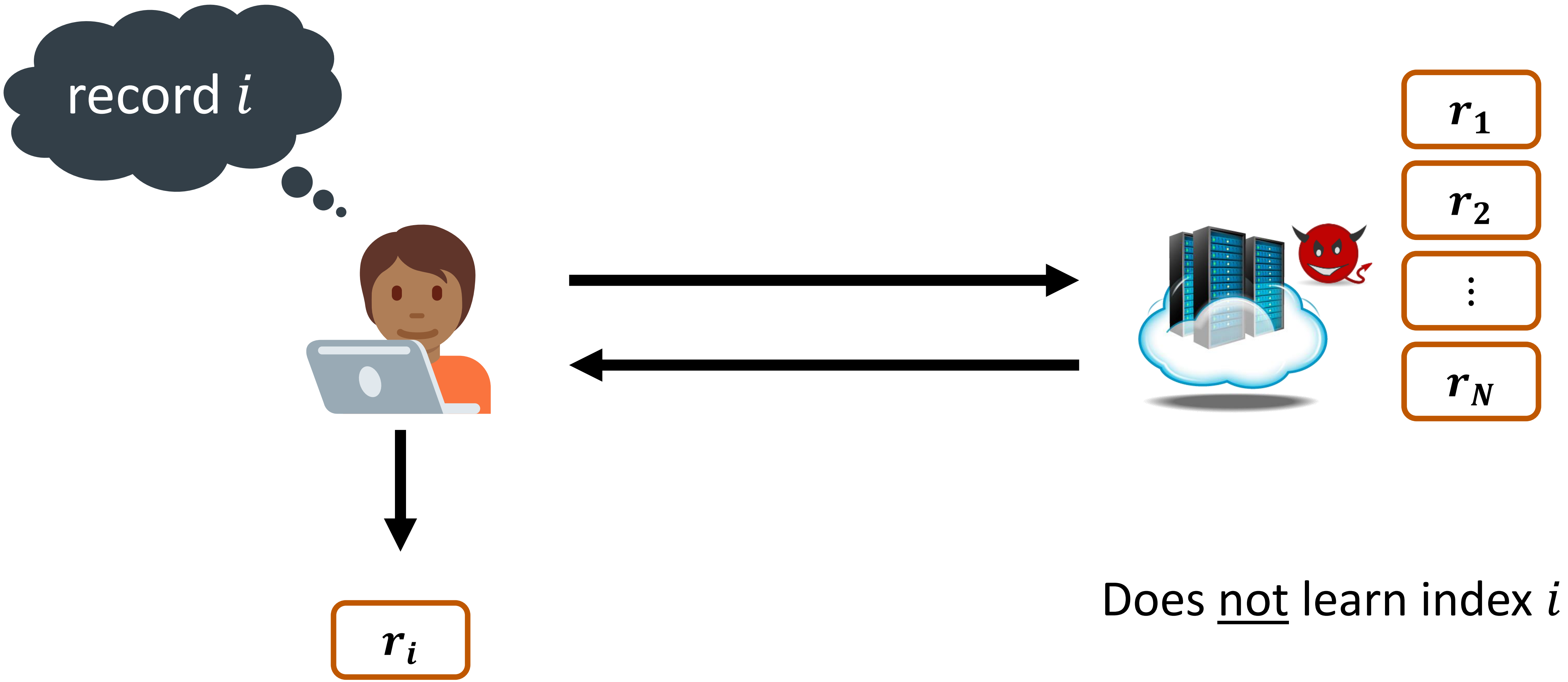
SPIRAL: Fast High-Rate Single-Server Private Information Retrieval

Samir Menon and David Wu

Private Information Retrieval (PIR)

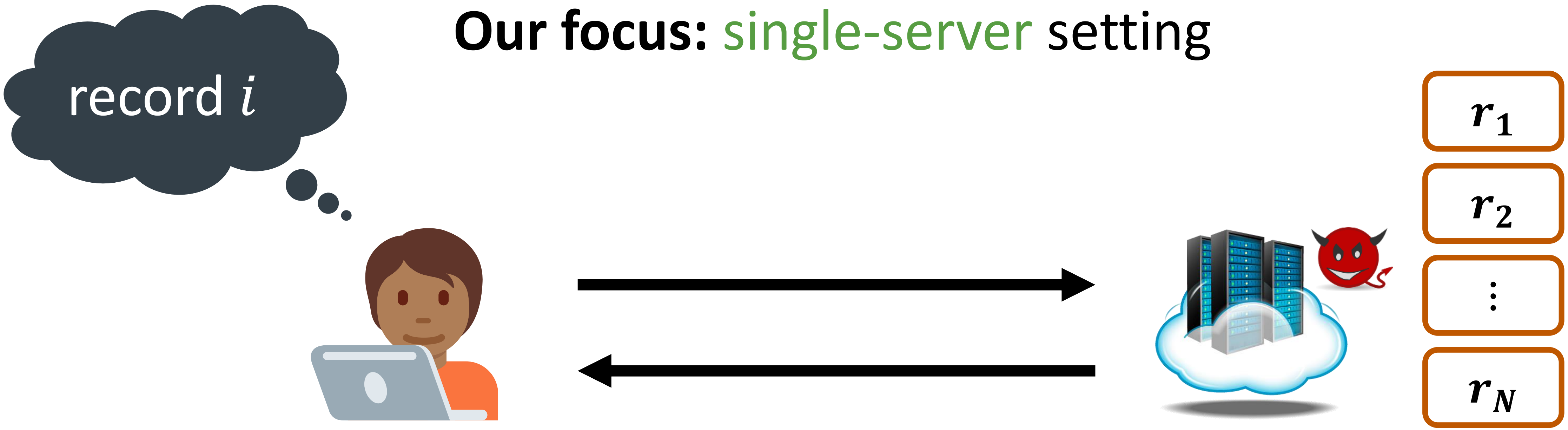
[CGKS95]



Private Information Retrieval (PIR)

[CGKS95]

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

Efficiency Metrics

1 **Query size**

query →

← response

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**

$$\frac{\text{database size}}{\text{server computation time}}$$

“measures how fast the server can respond as a function of database size”

Efficiency Metrics

Client generates a ***reusable*** set of public parameters

public parameters →

4 Public parameter size

1 Query size

query →

response ←

3 Rate

$$\frac{\text{record size}}{\text{response size}}$$

"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"

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

Base version of SPIRAL

Query size:	14 KB	4.5× smaller
Rate:	0.41	2.1× higher
Throughput:	333 MB/s	2.9× higher

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

Cost: 3.4× larger public parameters (17 MB)

Streaming versions of SPIRAL

Rate:	0.81	3.4× smaller responses
Throughput:	1.9 GB/s	12.3× higher

Best previous protocol:

Rate:	0.24
Throughput:	158 MB/s

The SPIRAL Family of PIR Protocols

Techniques to translate between FHE sch

Automatic parameter selection based on

Higher throughput than running software AES over database
(Primary operation: 64-bit integer arithmetic)

Base version of SPIRAL

Query size:	14 KB	4.5× smaller
Rate:	0.41	2.1× higher
Throughput:	333 MB/s	2.9× higher

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

Cost: 3.4× larger public parameters (17 MB)

Streaming versions of SPIRAL

Rate:	0.81	3.4× smaller responses
Throughput:	1.9 GB/s	12.3× higher

Best previous protocol:

Rate:	0.24
Throughput:	158 MB/s

The SPIRAL Family of PIR Protocols

Techniques to translate between FHE schemes enable

Automatic parameter selection based on database co

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)

Base version of SPIRAL

Query size:	14 KB	4.5× smaller
Rate:	0.41	2.1× higher
Throughput:	333 MB/s	2.9× higher

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

Cost: 3.4× larger public parameters (17 MB)

Streaming versions of SPIRAL

Rate:	0.81	3.4× smaller responses
Throughput:	1.9 GB/s	12.3× higher

Best previous protocol:

Rate:	0.24
Throughput:	158 MB/s

PIR from Homomorphic Encryption

[KO97]

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

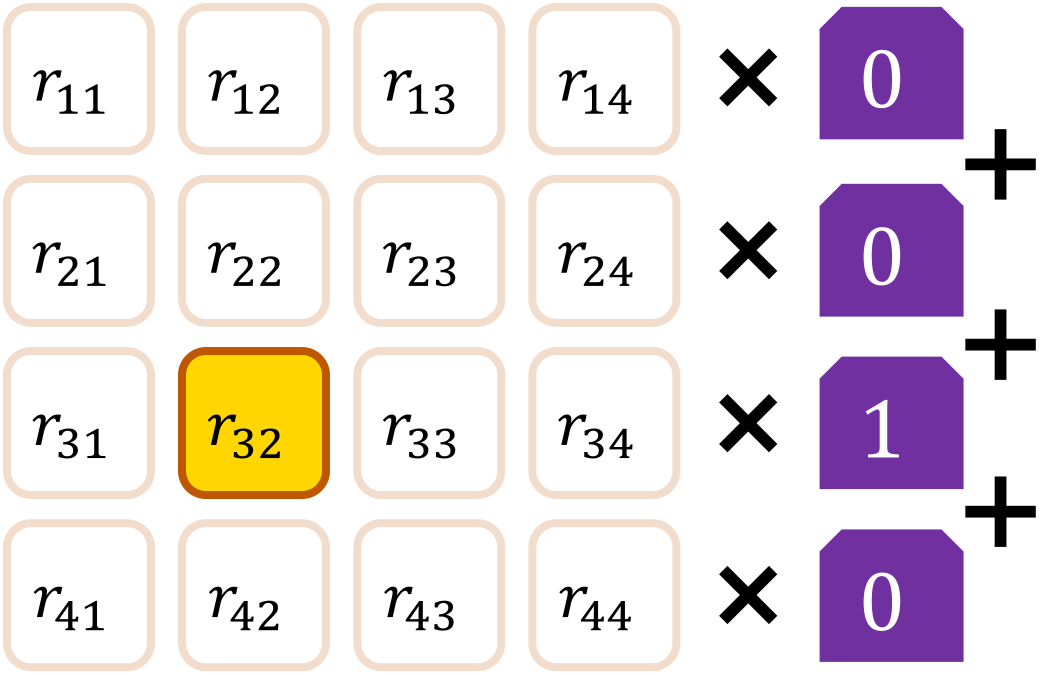
r_{11}	r_{12}	r_{13}	r_{14}
r_{21}	r_{22}	r_{23}	r_{24}
r_{31}	r_{32}	r_{33}	r_{34}
r_{41}	r_{42}	r_{43}	r_{44}

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

PIR from Homomorphic Encryption

[KO97]

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



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

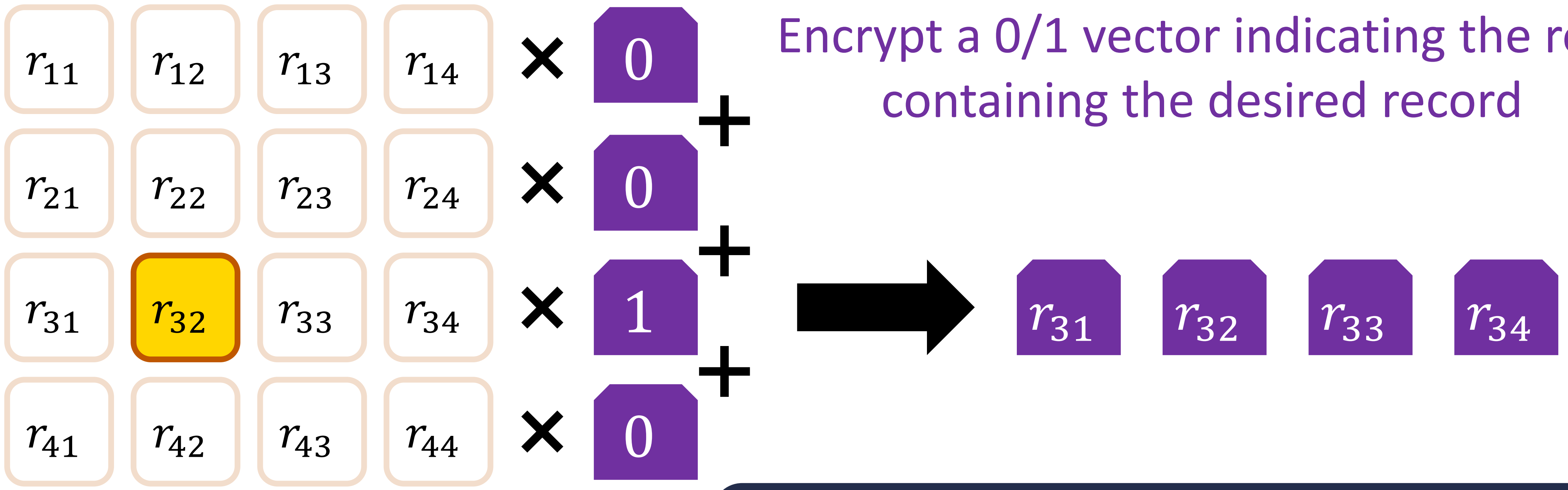
Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

Homomorphically compute product between query vector and database matrix

PIR from Homomorphic Encryption

[KO97]

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



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

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

Database is in the clear, so *additive* homomorphism suffices

PIR from Homomorphic Encryption

[KO97]

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

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

Client decrypts to learn records

r_{31} r_{32} r_{33} r_{34}

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

Homomorphically compute product between query vector and database matrix

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

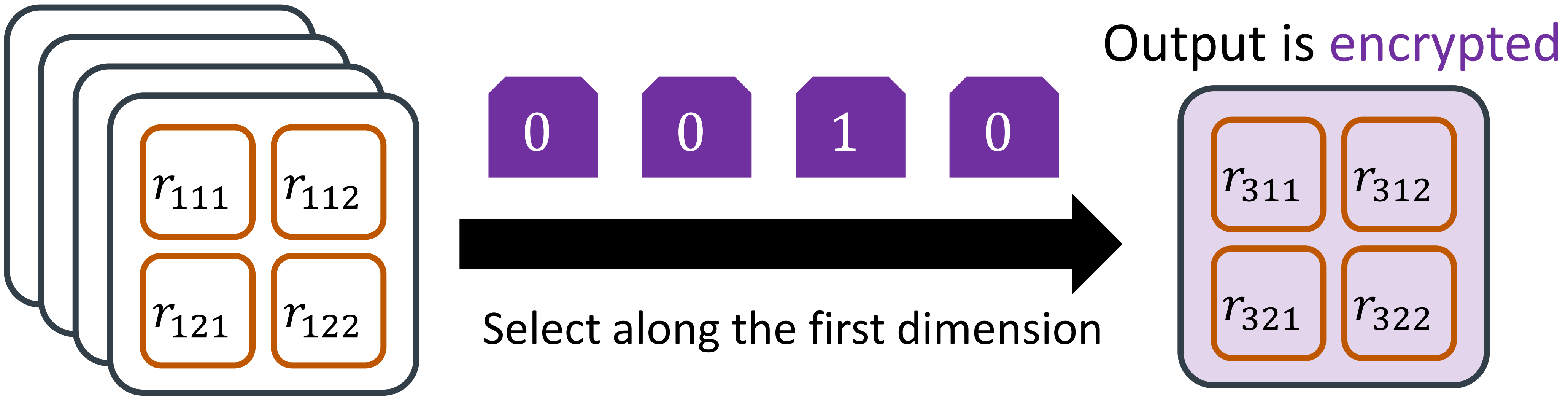
ciphertext size
(λ is security parameter)

Homomorphically compute product between query vector and database matrix

PIR from Homomorphic Encryption

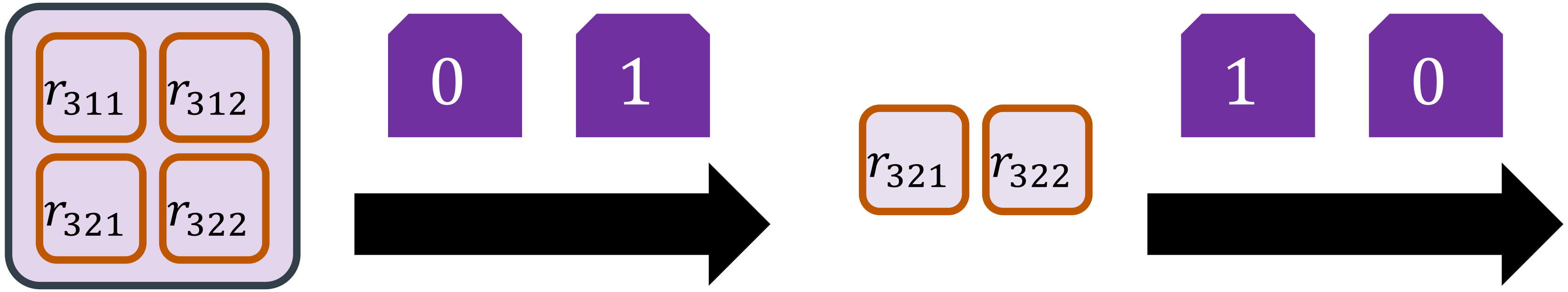
[KO97]

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]

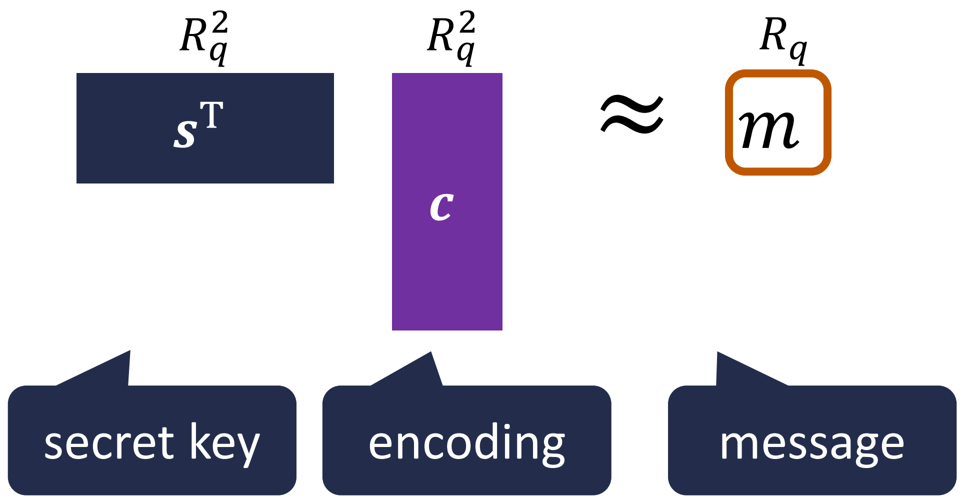
Low rate; supports homomorphic multiplication (with additive noise growth)

Goal: get the best of *both* worlds

Regev Encodings (over Rings)

[Reg04, LPR10]

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

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

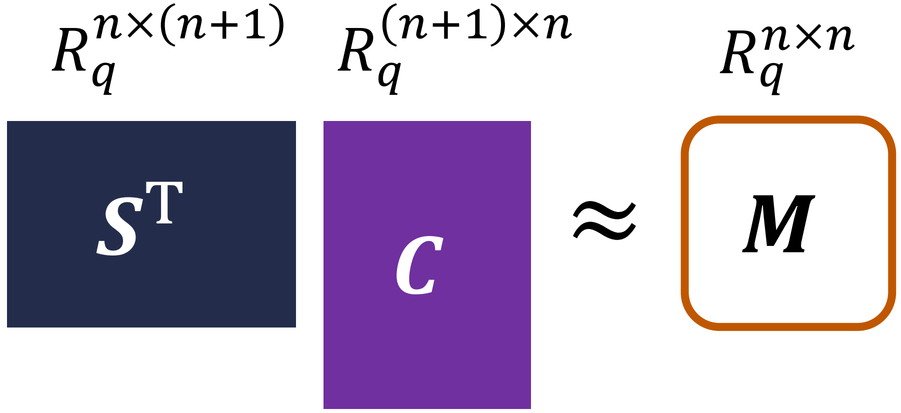
OnionPIR: rate = 0.24

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

Matrix Regev Encodings (over Rings)

[PVW08, LPR10]

Regev encoding of a matrix $\boldsymbol{M} \in R_q^{n\times n}$:



Idea: "Reuse" encryption randomness

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

Additively homomorphic:

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_1 \approx \boldsymbol{M}_1$$

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_2 \approx \boldsymbol{M}_2$$

$$\boldsymbol{S}^{\mathrm{T}}(\boldsymbol{C}_1 + \boldsymbol{C}_2) \approx \boldsymbol{M}_1 + \boldsymbol{M}_2$$

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

Gentry-Sahai-Waters Encodings

[GSW13]

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

$$\underbrace{\boldsymbol{S}^{\mathrm{T}}}_{R_q^{n\times(n+1)}} \underbrace{\boldsymbol{C}}_{R_q^{(n+1)\times n}} \approx \mu \underbrace{\boldsymbol{S}^{\mathrm{T}}}_{R_q^{n\times(n+1)}} \underbrace{\boldsymbol{G}}_{R_q^{(n+1)\times m}}$$

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

Gadget matrix [MP12]:

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

$$\boldsymbol{g}^{\mathrm{T}} = [1 \quad 2 \quad 2^2 \quad \cdots \quad 2^{\lfloor \log_z q \rfloor}]$$

"Powers-of-2" matrix

Construction will use other decomposition bases

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

Gentry-Sahai-Waters Encodings

[GSW13]

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

$$\underbrace{\boldsymbol{s}^{\mathrm{T}}}_{R_q^{n\times(n+1)}} \underbrace{\boldsymbol{C}}_{R_q^{(n+1)\times n}} \approx \mu \cdot \underbrace{\boldsymbol{s}^{\mathrm{T}}}_{R_q^{n\times(n+1)}} \underbrace{\boldsymbol{G}}_{R_q^{(n+1)\times m}}$$

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

Gadget matrix [MP12]:

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

$$\boldsymbol{g}^{\mathrm{T}} = [1 \quad 2 \quad 2^2 \quad \cdots \quad 2^{\lfloor \log_2 q \rfloor}]$$

"Powers-of-2" matrix

Construction will use other decomposition bases

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

"binary decomposition"

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

Gentry-Sahai-Waters Encodings

[GSW13]

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

$$\underbrace{\boldsymbol{s}^{\mathsf{T}}}_{R_q^{n\times(n+1)}} \underbrace{\boldsymbol{C}}_{R_q^{(n+1)\times n}} \approx \mu \underbrace{\boldsymbol{s}^{\mathsf{T}}}_{R_q^{n\times(n+1)}} \underbrace{\boldsymbol{G}}_{R_q^{(n+1)\times m}}$$

$$\text{rate} = \frac{1}{d(n+1)^2 \log q}$$

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

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

Gadget matrix [MP12]:

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

$$\boldsymbol{g}^{\mathsf{T}} = [1 \quad 2 \quad 2^2 \quad \cdots \quad 2^{\lfloor \log_z q \rfloor}]$$

"Powers-of-2" matrix

Construction will use other decomposition bases

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

Regev-GSW Homomorphism

[CGGI18]

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_{\mathrm{Reg}} \approx \boldsymbol{M}$$

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_{\mathrm{GSW}} \approx \mu\boldsymbol{S}^{\mathrm{T}}\boldsymbol{G}$$

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_{\mathrm{GSW}}\boldsymbol{G}^{-1}(\boldsymbol{C}_{\mathrm{Reg}}) \approx \mu\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_{\mathrm{Reg}} \approx \mu\boldsymbol{M}$$

With noise terms:

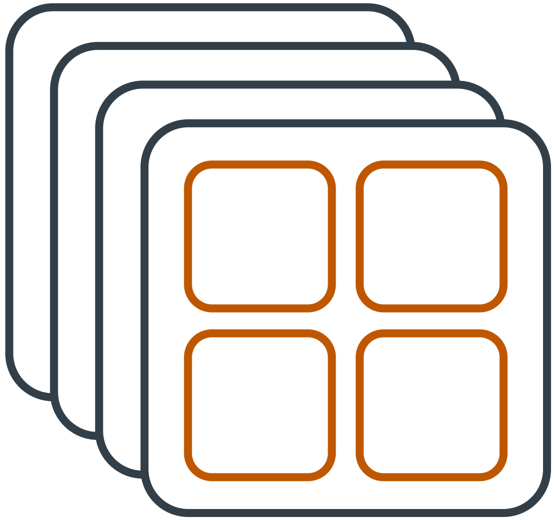
$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_{\mathrm{GSW}}\boldsymbol{G}^{-1}(\boldsymbol{C}_{\mathrm{Reg}}) = \mu\boldsymbol{M} + \boldsymbol{E}_{\mathrm{GSW}}\boldsymbol{G}^{-1}(\boldsymbol{C}_{\mathrm{Reg}}) + \mu E_{\mathrm{Reg}}$$

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

$\boldsymbol{C}_{\mathrm{GSW}}\boldsymbol{G}^{-1}(\boldsymbol{C}_{\mathrm{Reg}})$ is a Regev encoding of $\mu\boldsymbol{M}$

The Gentry-Halevi Blueprint

[GH19]

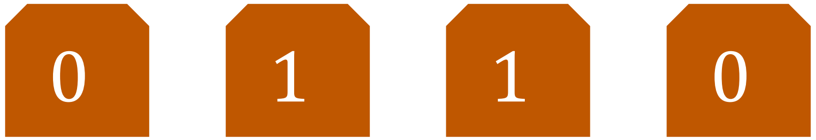


Database is represented as $2^{\nu_1} \times \underbrace{2 \times 2 \times \cdots \times 2}_{2^{\nu_2}}$ hypercube

Query contains 2^{ν_1} matrix Regev ciphertexts

Indicator for index along first dimension

Query contains ν_2 GSW ciphertexts



Indicator for index along subsequent dimensions

Each GSW ciphertext participates in only one multiplication with a Regev ciphertext!

Response is a single matrix Regev ciphertext

The Gentry-Halevi Blueprint

[GH19]

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

Drawback: large queries

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

Query contains 2^{ν_1} matrix Regev ciphertexts

0 I_n 0 0 0 0

Indicator for index along first dimension

Estimated size: 4 MB/ciphertext

Estimated query size: 30 MB

Query contains ν_2 GSW ciphertexts

0 1 1 0

Indicator for index along subsequent dimensions

The Gentry-Halevi Blueprint

[GH19]

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

Drawback: large queries

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

Query contains 2^{ν_1} matrix Regev ciphertexts

0	I_n	0	0	0	0

Indicator for index along first dimension

SealPIR query size: 66 KB

Estimated query size: 30 MB

Query contains ν_2 GSW ciphertexts

0	1	1	0

Indicator for index along subsequent dimensions

OnionPIR

[MCR21]

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 ≈ 0.24 (over $4\times$ response overhead)

This Work: Translating Between Regev and GSW

“Best of both worlds”: Small queries (as in OnionPIR) with the high rate/throughput of the Gentry-Halevi scheme

Query size:	14 KB	2000× smaller than Gentry-Halevi (4.5× smaller than OnionPIR)
Rate:	0.41	2.1× higher than OnionPIR
Throughput:	333 MB/s	2.9× higher than OnionPIR

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

Comparable improvements for other database configurations; more speed-ups in streaming setting

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 → Matrix Regev

Input: encoding $\boldsymbol{c}$ where $\boldsymbol{s}_1^{\mathrm{T}}\boldsymbol{c} \approx m$

$\boldsymbol{s}_1^{\mathrm{T}} = [-\tilde{s}_0 \mid 1] \in R_q^2$

$\boldsymbol{c}^{\mathrm{T}} = [c_0 \mid c_1] \in R_q^2$

Output: encoding $\boldsymbol{C}$ where $\boldsymbol{S}_2^{\mathrm{T}}\boldsymbol{C} \approx m\mathbf{I}_n$

$$\boldsymbol{S}_2^{\mathrm{T}} = \left[\begin{array}{c|c} -\tilde{s}_0 & \\ \vdots & \boldsymbol{I}_n \\ -\tilde{s}_0 & \end{array}\right]$$

$$\boldsymbol{C} = \left[\begin{array}{ccc} c_0 & \cdots & c_0 \\ \hline & c_1\boldsymbol{I}_n & \end{array}\right]$$

$$\boldsymbol{S}_2^{\mathrm{T}}\boldsymbol{C} = m\boldsymbol{I}_n$$

Can replace with $\boldsymbol{S}_2$ with arbitrary secret key using standard key-switching techniques

Matrix Regev → GSW

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

$$\boldsymbol{S}^{\mathrm{T}} = [-\boldsymbol{s} \mid \mathbf{I}_n] \in R_q^{n \times (n+1)}$$

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

rearrange →

$\boldsymbol{g}^{\mathrm{T}}$	$\mathbf{0}$				
$\mathbf{0}$	$\mathbf{I}_n$	$2\mathbf{I}_n$	$2^2\mathbf{I}_n$	$\cdots$	$2^t\mathbf{I}_n$

$t = \log q$

$\mu \boldsymbol{S}^{\mathrm{T}}\boldsymbol{G} =$

$-\mu \boldsymbol{s} \boldsymbol{g}^{\mathrm{T}}$	$\mu\mathbf{I}_n$	$2\mu\mathbf{I}_n$	$2^2\mu\mathbf{I}_n$	$\cdots$	$2^t\mu\mathbf{I}_n$

$\boldsymbol{C} =$

$\boldsymbol{A}$	$\boldsymbol{B}_0$	$\boldsymbol{B}_1$	$\boldsymbol{B}_2$	$\cdots$	$\boldsymbol{B}_t$

Break $\boldsymbol{C}$ into *blocks*

Matrix Regev → GSW

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

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C} = \left[\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{A} \ \middle|\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_0 \ \middle|\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_1 \ \middle|\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_2 \ \middle|\ \cdots \ \middle|\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_t\ \right]$$

$$\mu\boldsymbol{S}^{\mathrm{T}}\boldsymbol{G} = \left[\ -\mu \boldsymbol{s}\boldsymbol{g}^{\mathrm{T}} \ \middle|\ \mu\mathbf{I}_n \ \middle|\ 2\mu\mathbf{I}_n \ \middle|\ 2^2\mu\mathbf{I}_n \ \middle|\ \cdots \ \middle|\ 2^t\mu\mathbf{I}_n\ \right]$$

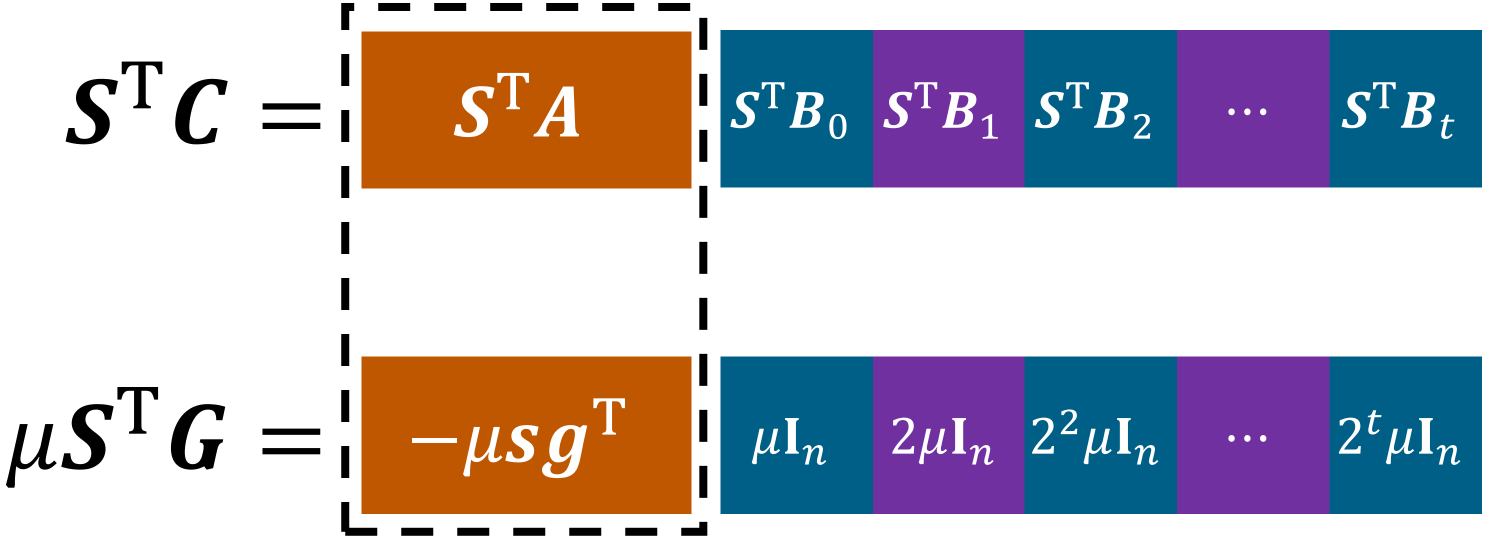
$\boldsymbol{B}_0, \dots, \boldsymbol{B}_t$ are matrix Regev ciphertexts encrypting $\mu\mathbf{I}_n, 2\mu\mathbf{I}_n, \dots, 2^t\mu\mathbf{I}_n$

⬇

Can derive from scalar Regev encodings of $\mu, 2\mu, \dots, 2^t\mu$

Matrix Regev → GSW

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



Write $\boldsymbol{S}^{\mathrm{T}} = [-\boldsymbol{s} \mid \mathbf{I}_n]$

Let $\boldsymbol{s}_{\mathrm{Reg}}$ be the key for a Regev encoding scheme

Construct key-switching matrix $\boldsymbol{W}$:

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{W} \approx -\boldsymbol{s}\left(\boldsymbol{s}_{\mathrm{Reg}}^{\mathrm{T}} \otimes \boldsymbol{g}^{\mathrm{T}}\right)$$

Let $\boldsymbol{c}_0, \dots \boldsymbol{c}_t$ be encodings of $\mu, \dots, 2^t\mu$ under $\boldsymbol{s}_{\mathrm{Reg}}$: $\boldsymbol{s}_{\mathrm{Reg}}^{\mathrm{T}}\boldsymbol{c}_i \approx 2^i\mu$

Let $\boldsymbol{C} = [\boldsymbol{c}_0 \mid \cdots \mid \boldsymbol{c}_t]$

Then, $\boldsymbol{S}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{g}^{-1}(\boldsymbol{C}) \approx -\boldsymbol{s}\left(\boldsymbol{s}_{\mathrm{Reg}}^{\mathrm{T}} \otimes \boldsymbol{g}^{\mathrm{T}}\right)\boldsymbol{g}^{-1}(\boldsymbol{C}) \approx -\boldsymbol{s}[\,\mu \mid \cdots \mid 2^t\mu\,] = -\mu\boldsymbol{s}\boldsymbol{g}^{\mathrm{T}}$

Matrix Regev → GSW

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

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C} = \left[\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{A} \ \middle|\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_0 \ \middle|\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_1 \ \middle|\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_2 \ \middle|\ \cdots \ \middle|\ \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_t\ \right]$$

$$\mu\boldsymbol{S}^{\mathrm{T}}\boldsymbol{G} = \left[\ -\mu\boldsymbol{s}\boldsymbol{g}^{\mathrm{T}} \ \middle|\ \mu\mathbf{I}_n \ \middle|\ 2\mu\mathbf{I}_n \ \middle|\ 2^2\mu\mathbf{I}_n \ \middle|\ \cdots \ \middle|\ 2^t\mu\mathbf{I}_n\ \right]$$

Write $\boldsymbol{S}^{\mathrm{T}} = [-\boldsymbol{s} \mid \mathbf{I}_n]$

Let $\boldsymbol{s}_{\mathrm{Reg}}$ be the key for a Regev encoding scheme

Construct key-switching matrix $\boldsymbol{W}$:

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{W} \approx -\boldsymbol{s}\left(\boldsymbol{s}_{\mathrm{Reg}}^{\mathrm{T}} \otimes \boldsymbol{g}^{\mathrm{T}}\right)$$

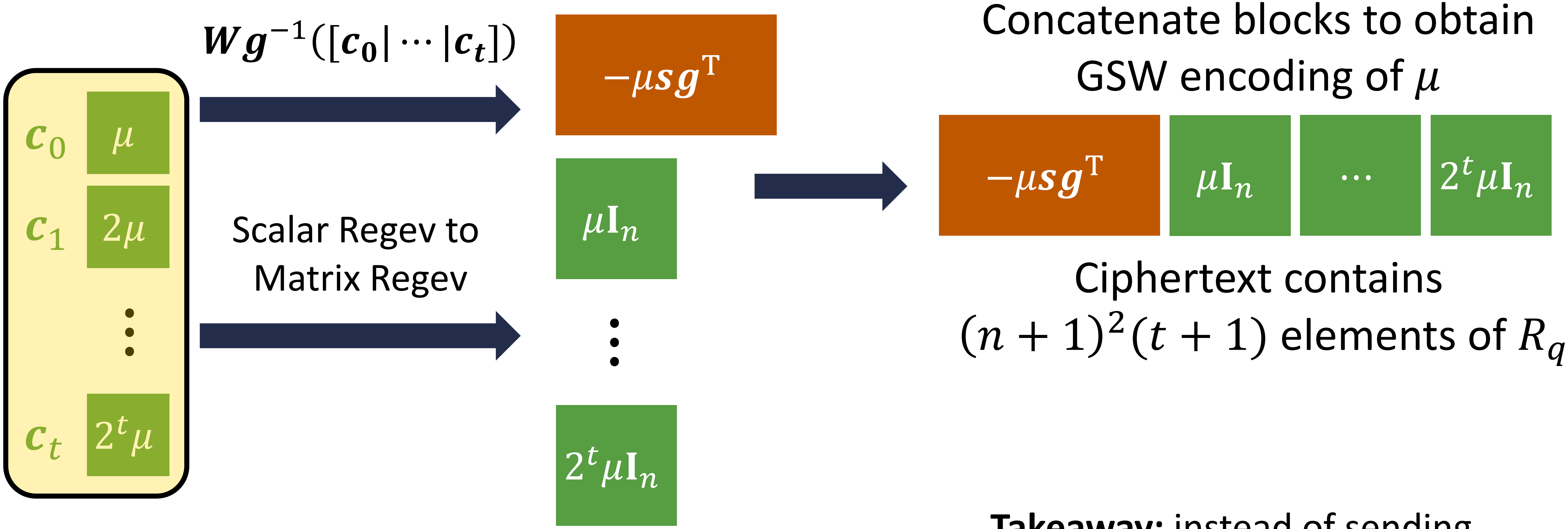
Let $\boldsymbol{c}_0$, .. under $\boldsymbol{s}_{\mathrm{Reg}}$: $\boldsymbol{s}_{\mathrm{Reg}}^{\mathrm{T}}\boldsymbol{c}_i \approx 2^i\mu$

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

Let $\boldsymbol{C} =$

Then, $\boldsymbol{S}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{g}^{-1}(\boldsymbol{C}) \approx -\boldsymbol{s}\left(\boldsymbol{s}_{\mathrm{Reg}}^{\mathrm{T}} \otimes \boldsymbol{g}^{\mathrm{T}}\right)\boldsymbol{g}^{-1}(\boldsymbol{C}) \approx -\boldsymbol{s}[\,\mu \mid \cdots \mid 2^t\mu\,] = -\mu\boldsymbol{s}\boldsymbol{g}^{\mathrm{T}}$

Matrix Regev → GSW

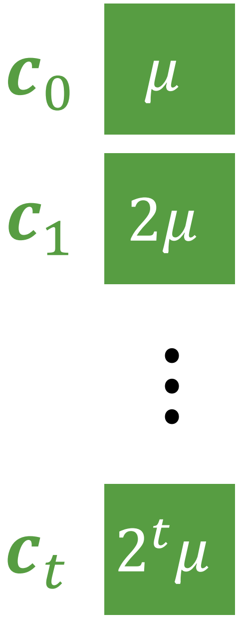


scalar Regev encodings: elements of R_q^2

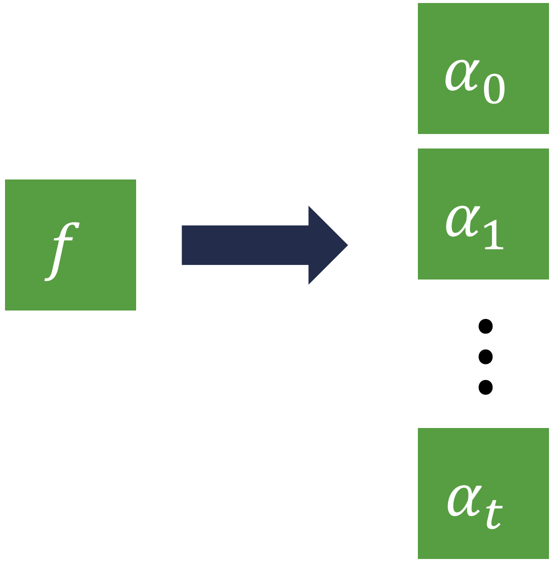
matrix Regev encodings: elements of $R_q^{(n+1)\times n}$

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

Further Compression via Polynomial Encodings



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



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

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

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)

$(n+1)^2(t+1)$ 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^{ν_1} matrix Regev ciphertexts

0 | I_n | 0 | 0

Indicator for index along first dimension

1. Compress into scalar Regev encodings

0
1
0
0

Query contains ν_2 GSW ciphertexts

0 | 1 | 1 | 0

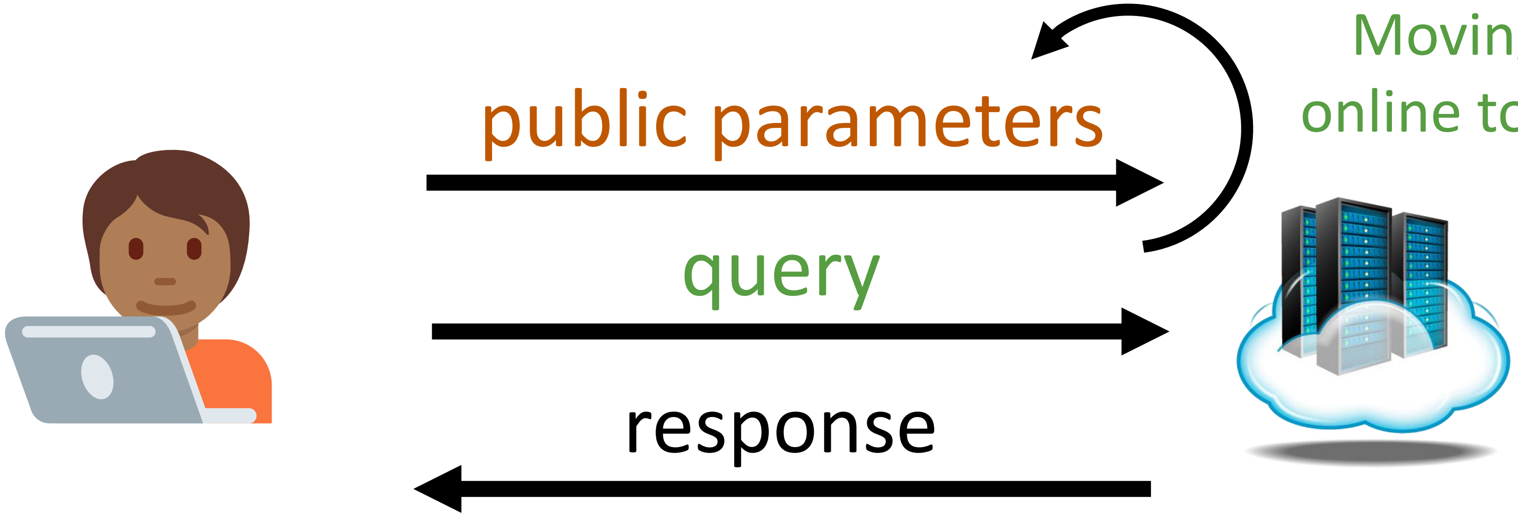
Indicator for index along subsequent dimensions

2. Pack scalars into single polynomial

0	0	0	0
1	2	2^2	2^3
1	2	2^2	2^3
0	0	0	0

f

Query Expansion in Spiral



Moving costs from online to offline phase

offline and one-time cost

online cost

SPIRAL also achieves higher rate and throughput

Trade-off: larger public parameters, smaller queries

SealPIR:	3 MB
OnionPIR:	5 MB
SPIRAL:	18 MB

SealPIR:	66 KB	Gentry-Halevi:	≈30 MB
OnionPIR:	63 KB	**SPIRAL:**	14 KB

Response Compression via Modulus Switching

PIR response consists of a single matrix Regev encoding

$$[-\boldsymbol{s} \mid \mathbf{I}_n] \; \boldsymbol{C} \approx \frac{q}{p} \cdot \boldsymbol{M}$$

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 $\boldsymbol{E}$ is scaled by $[-\boldsymbol{s} \mid \mathbf{I}_n]$

$$[-\boldsymbol{s} \mid \mathbf{I}_n] \; \boldsymbol{E}$$

Response Compression via Modulus Switching

PIR response consists of a single <u>matrix</u> Regev encoding

$$[-\boldsymbol{s} \mid \mathbf{I}_n] \; \boldsymbol{C} \approx \frac{q}{p} \cdot \boldsymbol{M}$$

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 $\boldsymbol{E}$ is scaled by $[-\boldsymbol{s} \mid \mathbf{I}_n]$

$$[-\boldsymbol{s} \mid \mathbf{I}_n] \begin{bmatrix} \boldsymbol{e}_0^{\mathrm{T}} \\ \boldsymbol{E}_1 \end{bmatrix}$$

$\boldsymbol{e}_0^{\mathrm{T}}$: Error scaled by $-\boldsymbol{s}$

$\boldsymbol{E}_1$: Error scaled by $\mathbf{I}_n$

Response Compression via Modulus Switching

PIR response consists of a single matrix Regev encoding

$$[-\boldsymbol{s} \mid \mathbf{I}_n] \; \boldsymbol{C} \approx \frac{q}{p} \cdot \boldsymbol{M}$$

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: *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 $\boldsymbol{E}$ is scaled by $[-\boldsymbol{s} \mid \mathbf{I}_n]$

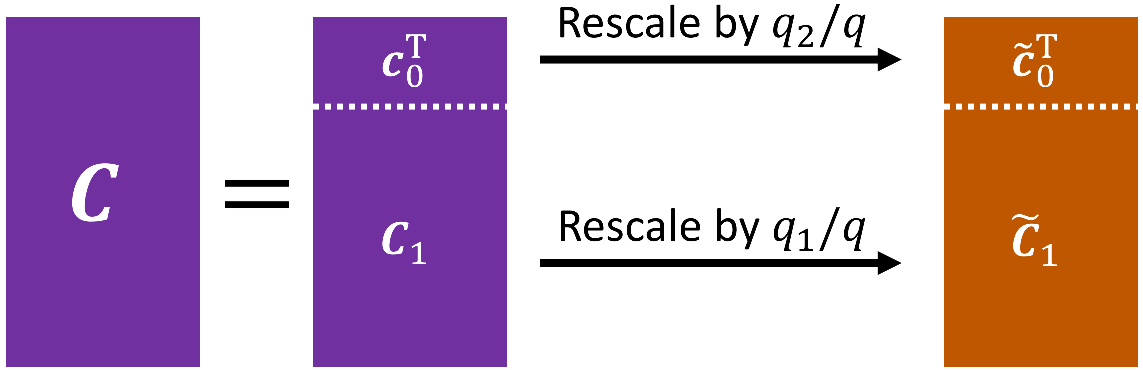
$$[-\boldsymbol{s} \mid \mathbf{I}_n] \begin{bmatrix} \boldsymbol{e}_0^{\mathrm{T}} \\ \boldsymbol{E}_1 \end{bmatrix}$$

$\boldsymbol{e}_0^{\mathrm{T}}$: Error scaled by $-\boldsymbol{s}$

$\boldsymbol{E}_1$: Error scaled by $\mathbf{I}_n$

Response Compression via Modulus Switching

PIR response consists of a single <u>matrix</u> Regev encoding



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

Approach: Use two different moduli to rescale the ciphertext

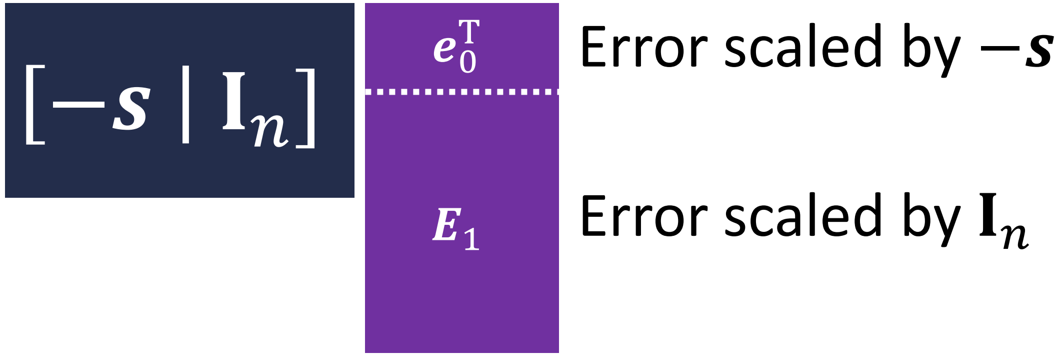
Standard technique in FHE: *modulus reduction*

Rescale ciphertext by $\frac{q'}{q}$ where $q' < q$

$$\text{rate} \propto \frac{\log p}{\log q'}$$

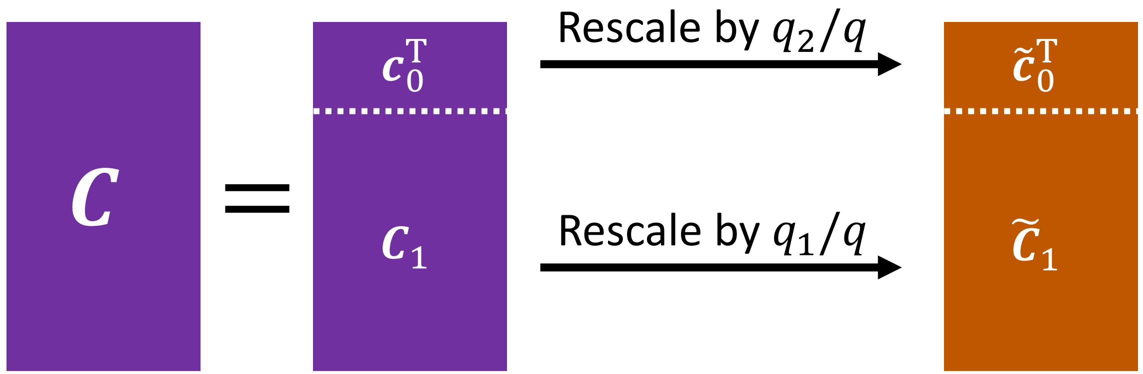
Rescaling introduces small amount of noise (from rounding)

This work: Observe that rounding error $\boldsymbol{E}$ is scaled by $[-\boldsymbol{s} \mid \mathbf{I}_n]$



Response Compression via Modulus Switching

PIR response consists of a single matrix Regev encoding



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

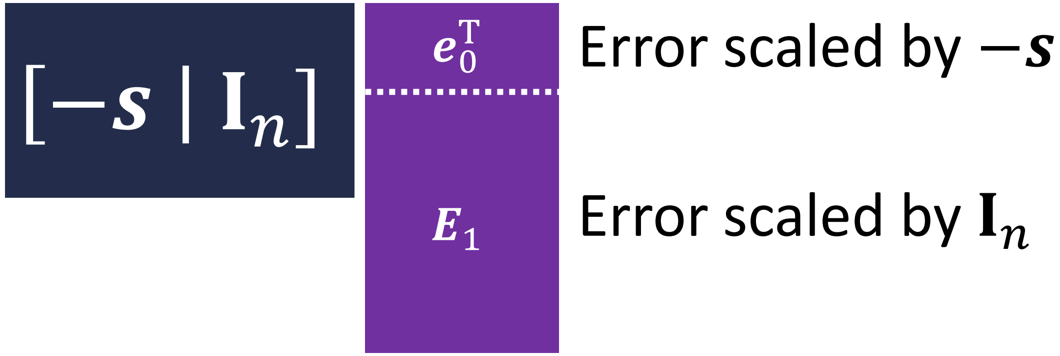
Approach: Use two different moduli to rescale the ciphertext

$$\text{rate} = \frac{n^2 \log p}{n^2 \log q_1 + n \log q_2}$$

- 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)

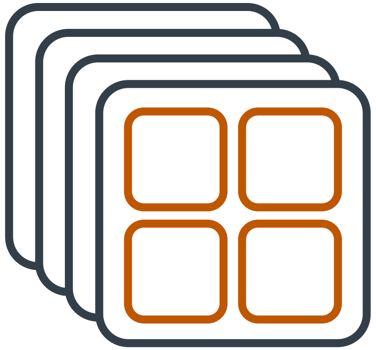
This work: Observe that rounding error $\boldsymbol{E}$ is scaled by $[-\boldsymbol{s} \mid \mathbf{I}_n]$



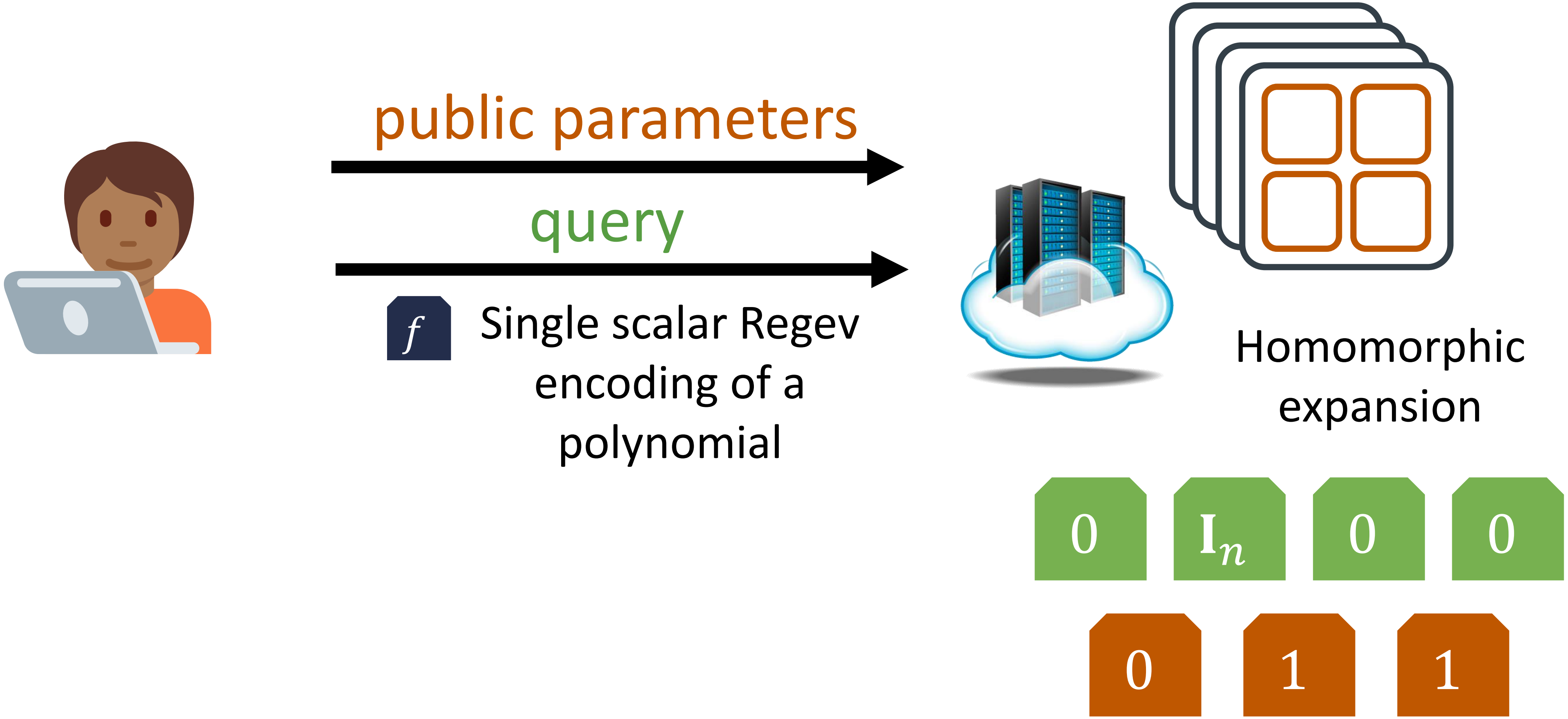
Vanilla SPIRAL

public parameters

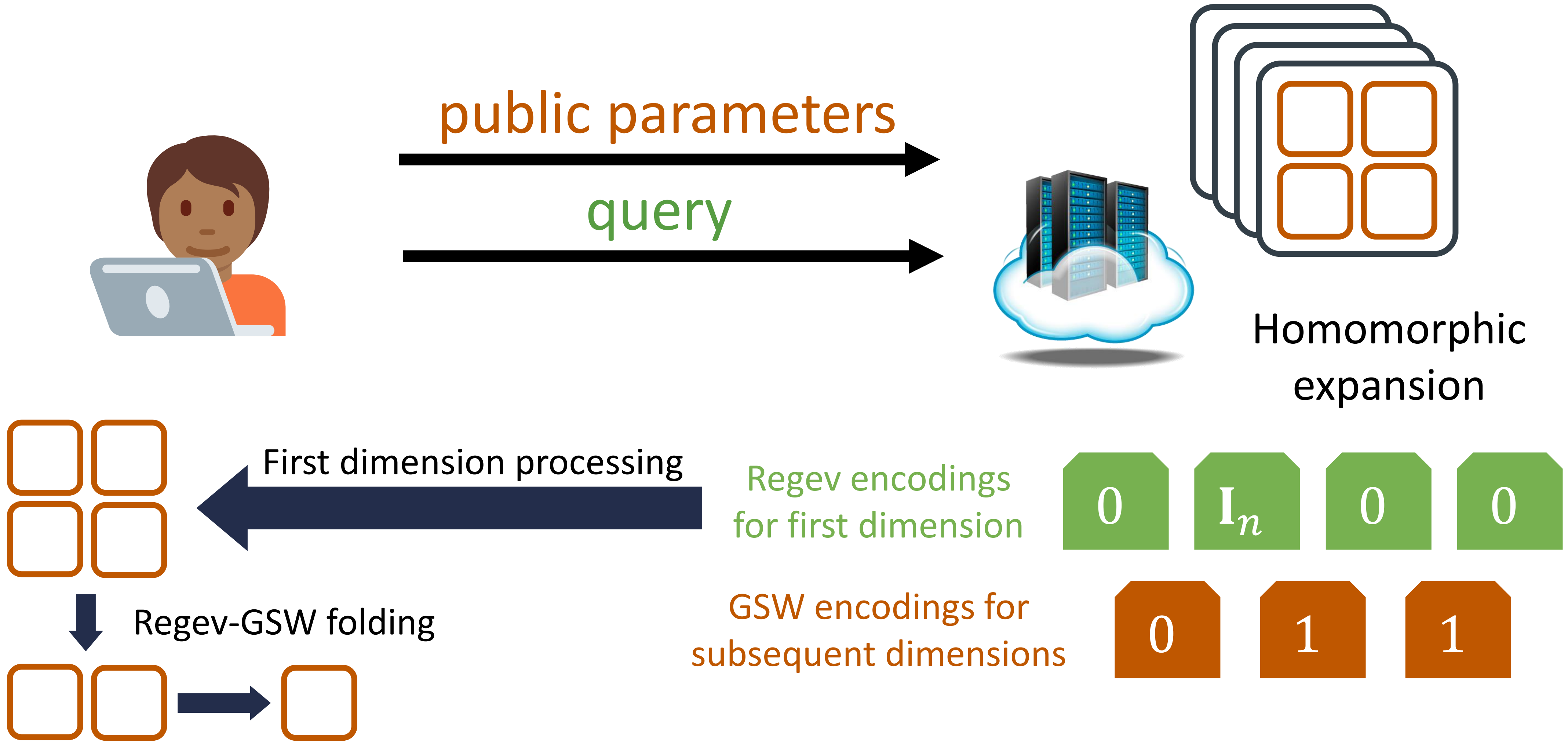
Key-switching matrices for ciphertext expansion and translation



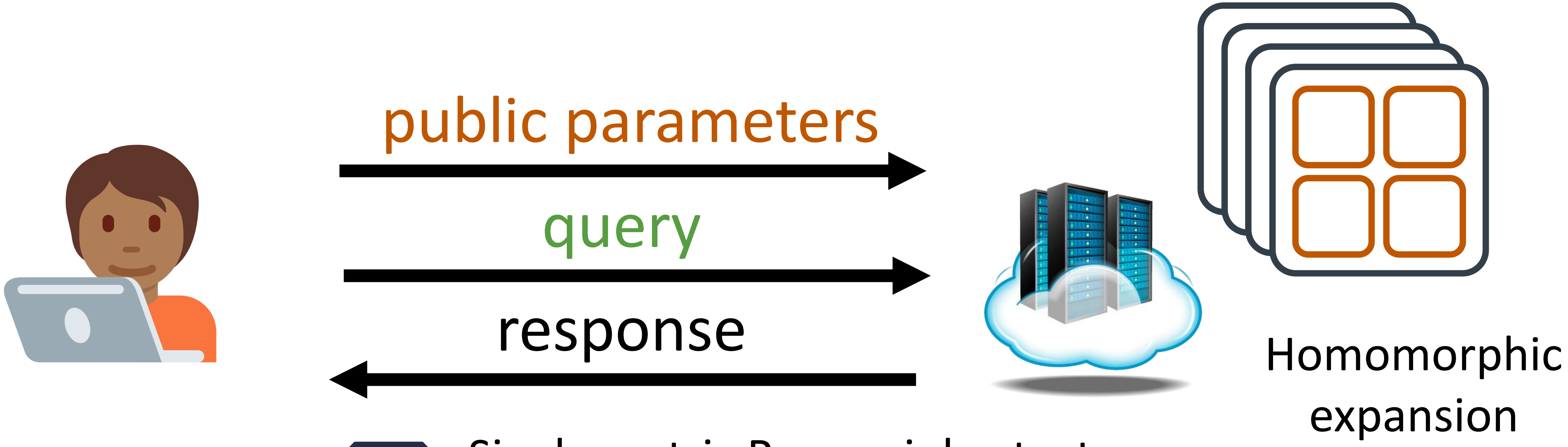
Vanilla Spiral



Vanilla SPIRAL



Vanilla SPIRAL



r_i Single matrix Regev ciphertext (with modulus reduction)

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

Database	Metric	SealPIR	FastPIR	OnionPIR	SPIRAL
2^{18} records 30 KB records (7.9 GB database)	Public Param. Size	3 MB	1 MB	5 MB	18 MB
	Query Size	66 KB	8 MB	63 KB	14 KB
	Response Size	3 MB	262 KB	127 KB	84 KB
	Server Compute	74.91 s	50.5 s	52.7 s	24.5 s
			Rate:	0.24	0.36
			Throughput:	149 MB/s	322 MB/s

Database configuration preferred by OnionPIR

Compared to OnionPIR:

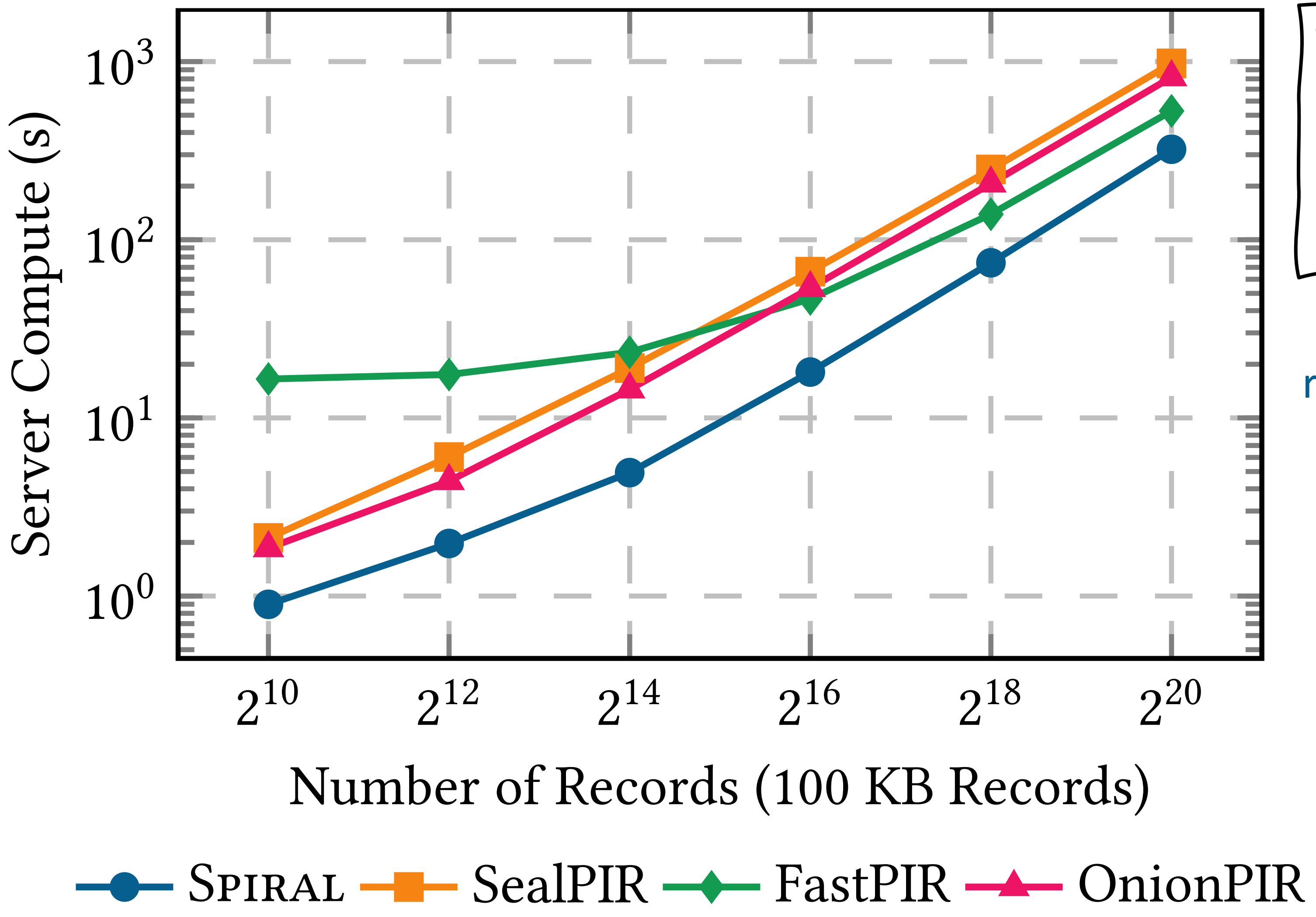
reduce query size by 4.5×

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):

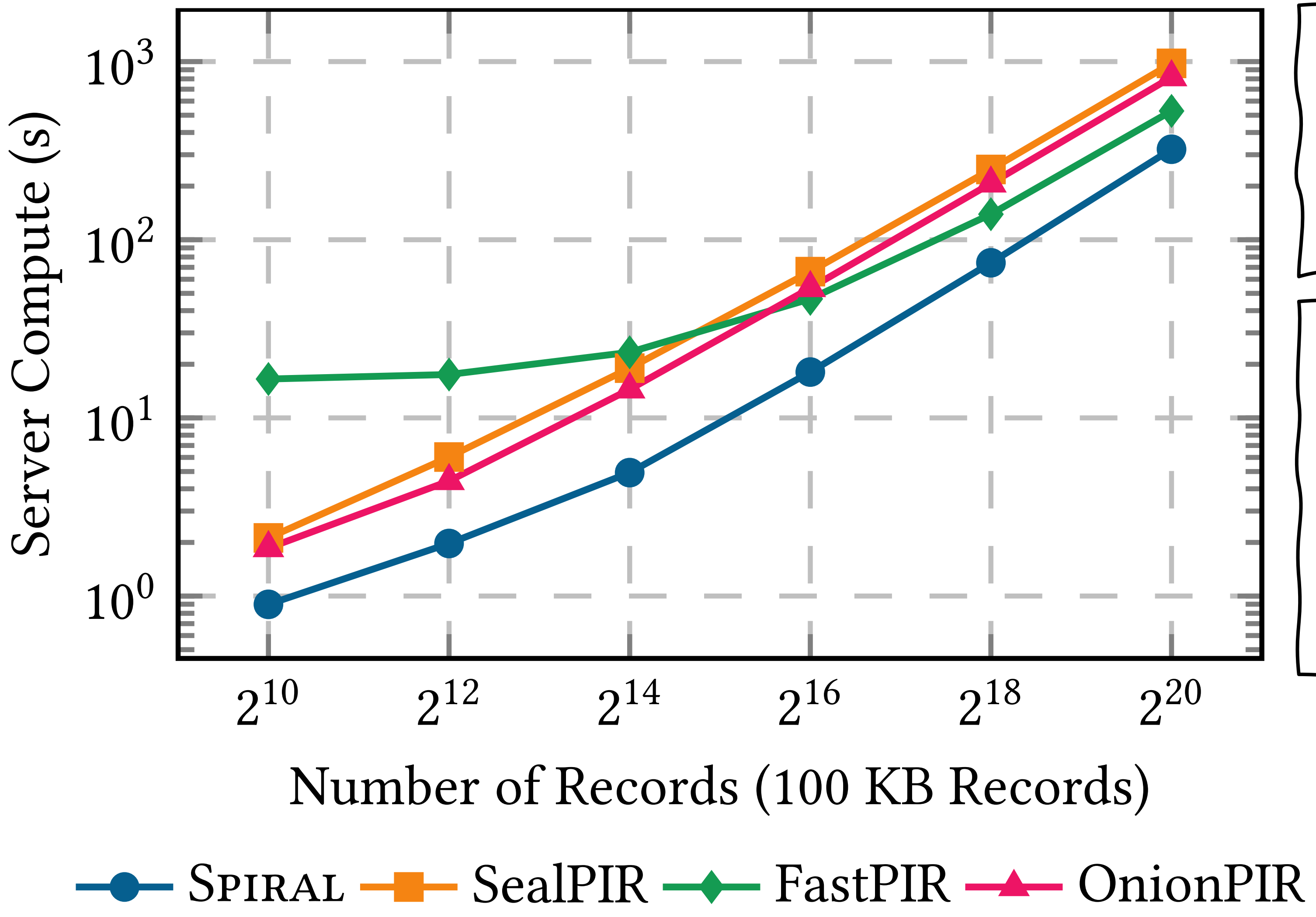
- Spiral: 310 MB/s (322 s)
- SealPIR: 102 MB/s (977 s)
- FastPIR: 189 MB/s (528 s)
- OnionPIR: 122 MB/s (817 s)

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 <u>linear</u> 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 ≈ 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*)

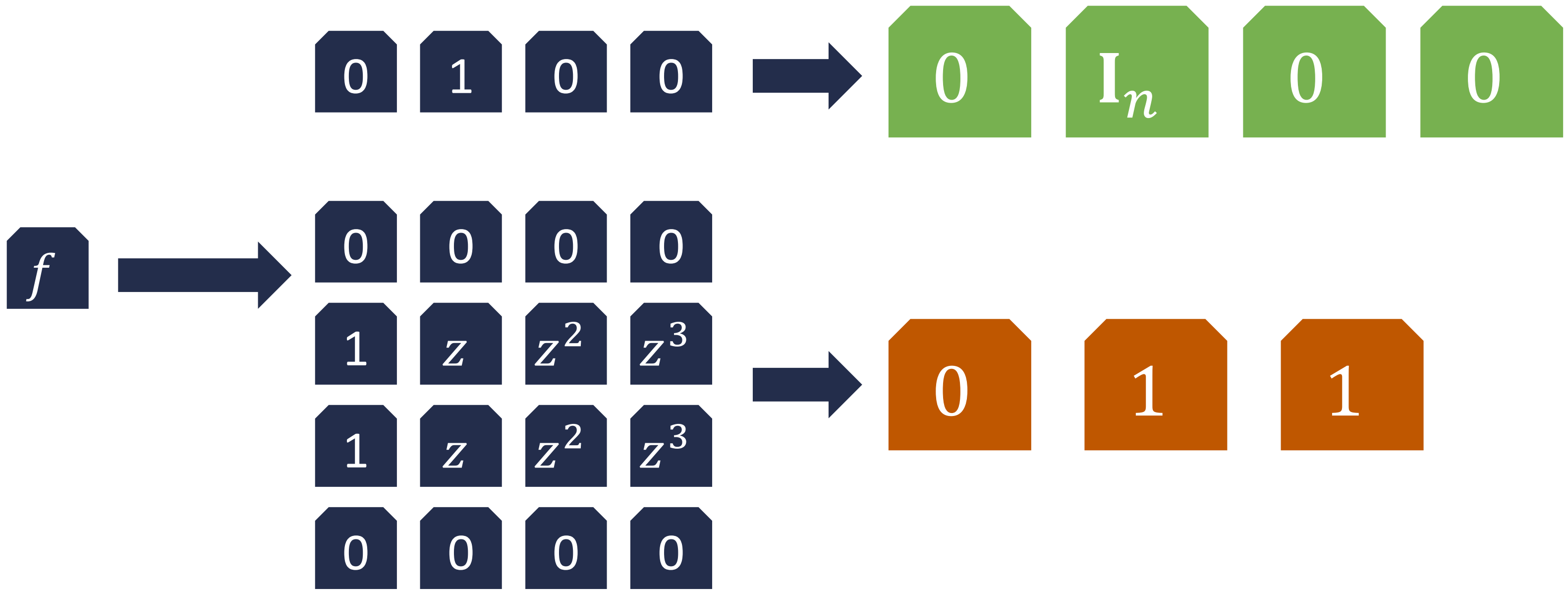
The Streaming Setting: SPIRALSTREAM

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

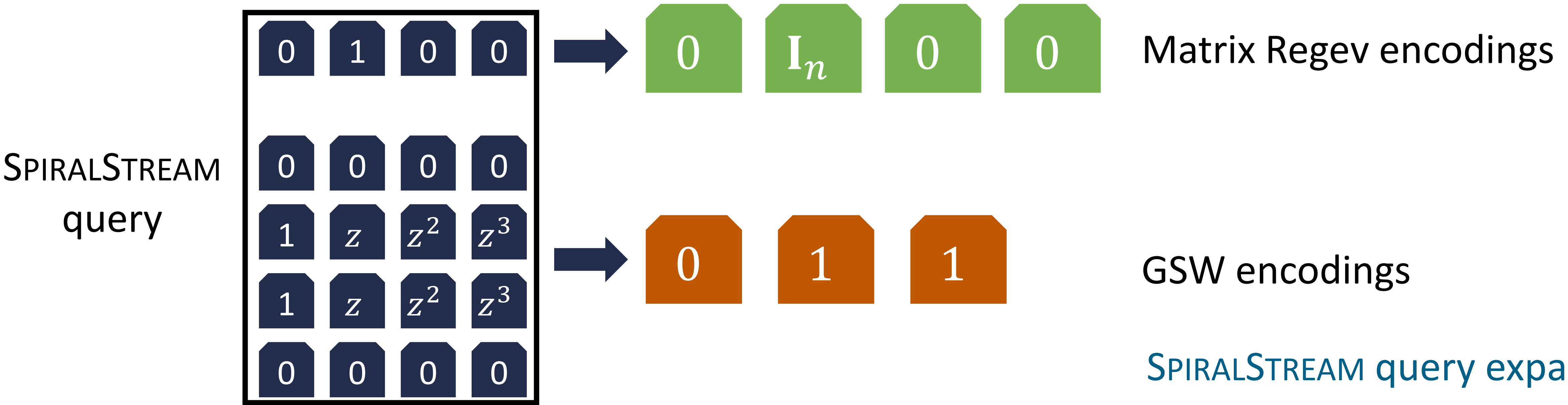
GSW encodings

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 expansion

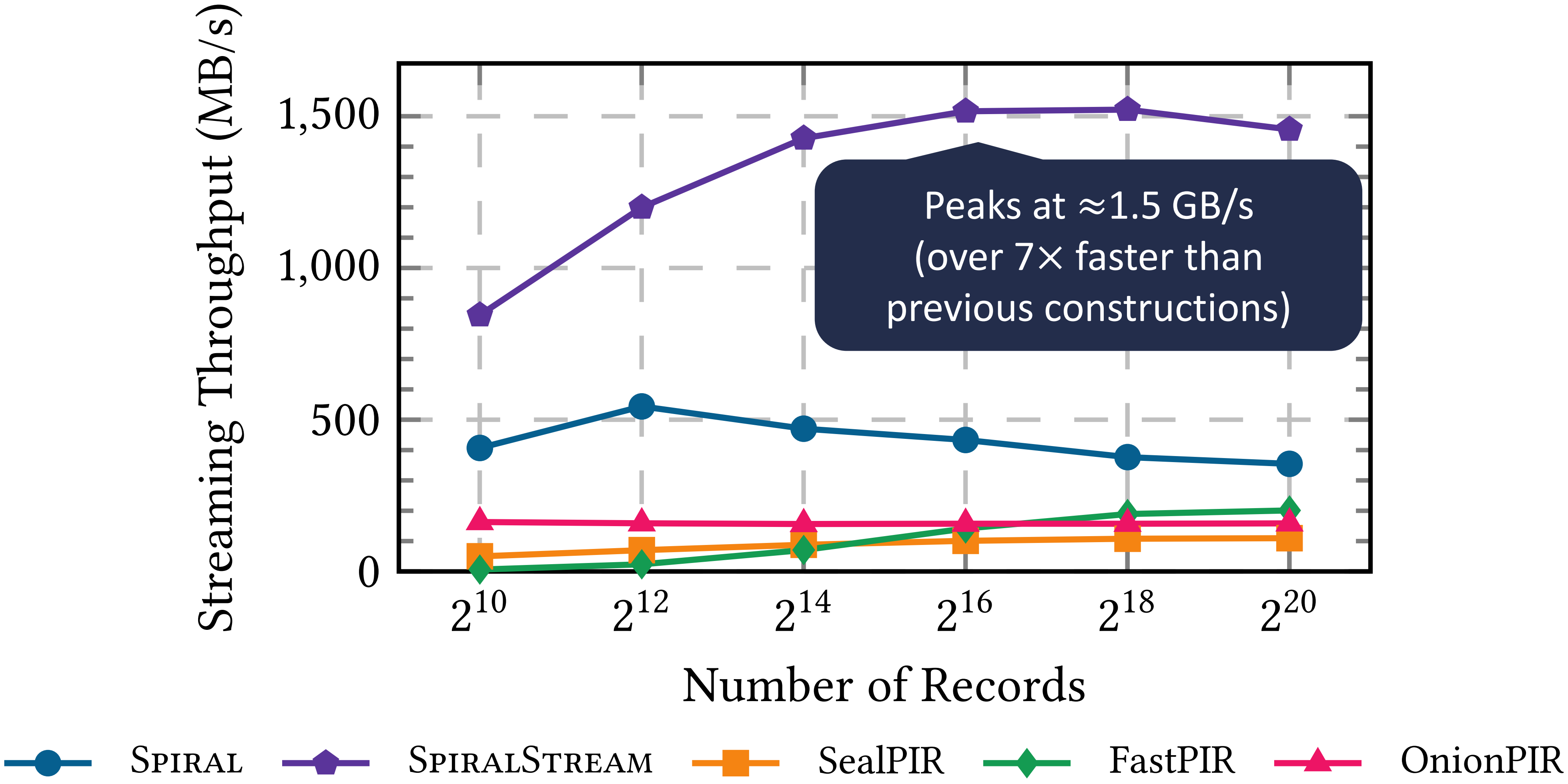
The Streaming Setting: SpiralStream

Database	Metric	OnionPIR	Spiral	SpiralStream
2^{18} records 30 KB records (7.9 GB database)	Public Param. Size	5 MB	18 MB	3 MB
	Query Size	63 KB	14 KB	15 MB
	Response Size	127 KB	84 KB	62 KB
	Server Compute	52.7 s	24.5 s	9.0 s
	Rate:	0.23	0.36	0.48
	Throughput:	149 MB/s	322 MB/s	874 MB/s

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



Higher Rate via Response Packing: SpiralPack

Can we further reduce response size?

$$\text{rate} = \frac{n^2 \log p}{n \log q_2 + n^2 \log q_1} \qquad q_1 = 4p$$

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

SPIRAL

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

Each record is
$n \times n$ matrix

SPIRALPACK

Split database into n^2 databases

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

Better throughput
Worse rate

Response consists of n^2 Regev encodings

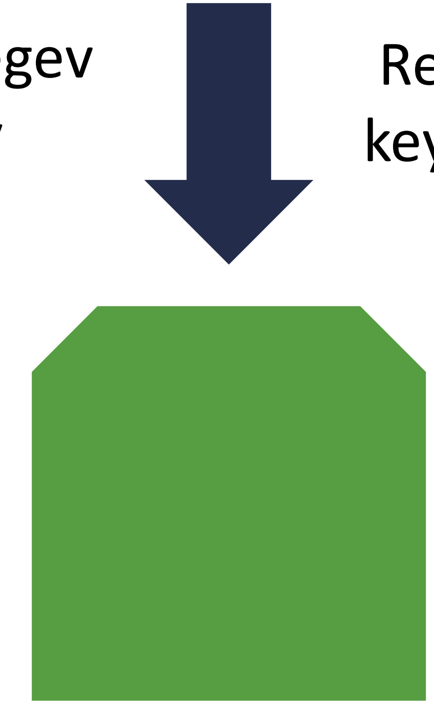
Higher Rate via Response Packing: SpiralPack

n^2 Regev ciphertexts
with dimension 1

Consists of $2n^2$ ring elements

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)

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

Database	Metric	OnionPIR	Spiral	SpiralStream
2^{18} records 30 KB records (7.9 GB database)	Public Param. Size	5 MB	18 MB → 18 MB	3 MB → 16 MB
	Query Size	63 KB	14 KB → 14 KB	15 MB → 30 MB
	Response Size	127 KB	84 KB → 86 KB	62 KB → 96 KB
	Server Compute	52.7 s	24.5 s → 17.7 s	9.0 s → 5.3 s

- 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

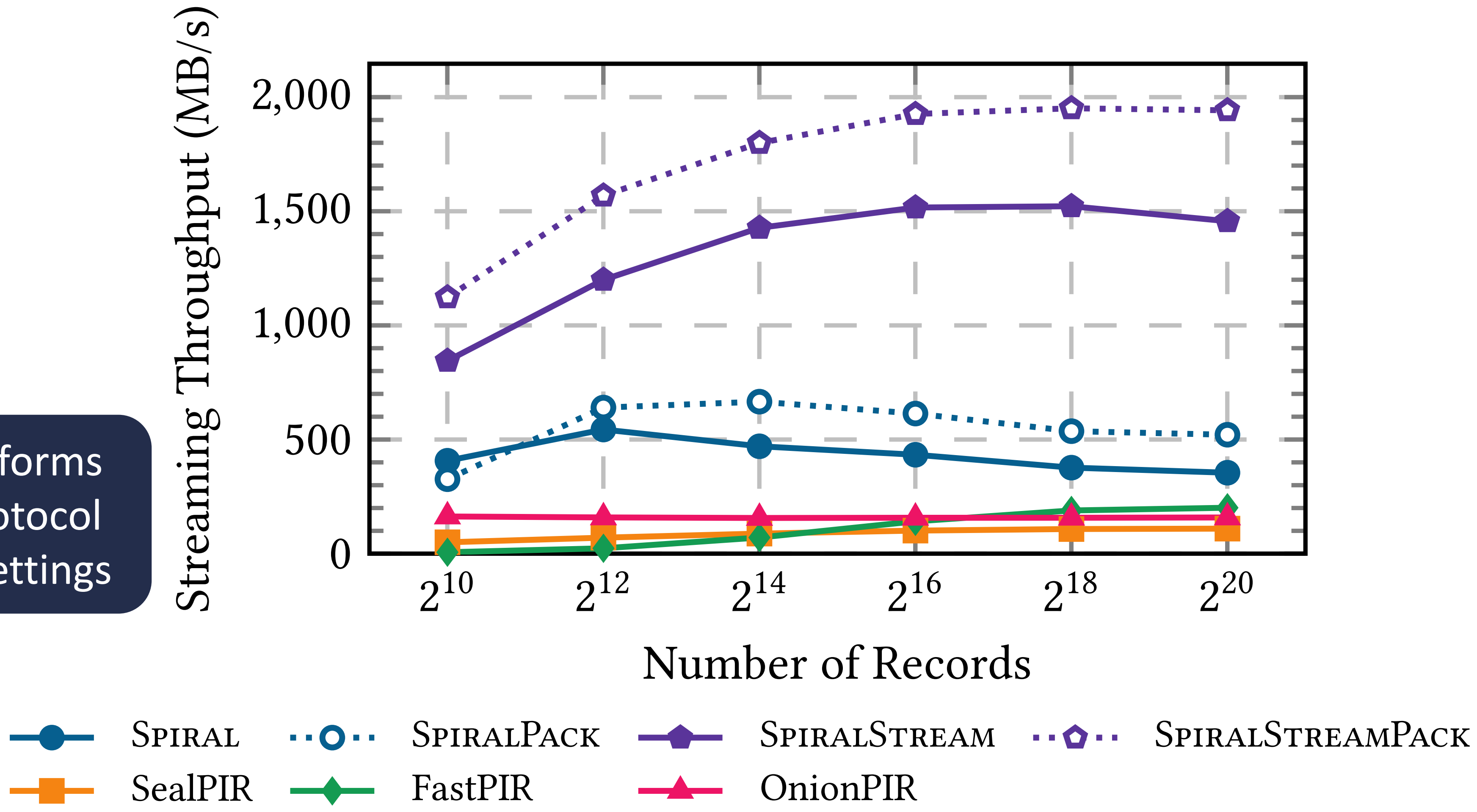
Database	Metric	OnionPIR	Spiral	SpiralStream
2^{18} records 30 KB records (7.9 GB database)	Public Param. Size	5 MB	18 MB → 18 MB	3 MB → 16 MB
	Query Size	63 KB	14 KB → 14 KB	15 MB → 30 MB
	Response Size	127 KB	84 KB → 86 KB	62 KB → 96 KB
	Server Compute	52.7 s	24.5 s → 17.7 s	9.0 s → 5.3 s
2^{14} records 100 KB records (1.6 GB database)	Public Param. Size	5 MB	17 MB → 47 MB	1 MB → 24 MB
	Query Size	63 KB	14 KB → 14 KB	8 MB → 30 MB
	Response Size	508 KB	242 KB → 188 KB	208 KB → 150 KB
	Server Compute	14.4 s	4.92 s → 4.58 s	2.4 s → 1.2 s
	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

Packing in the Streaming Setting

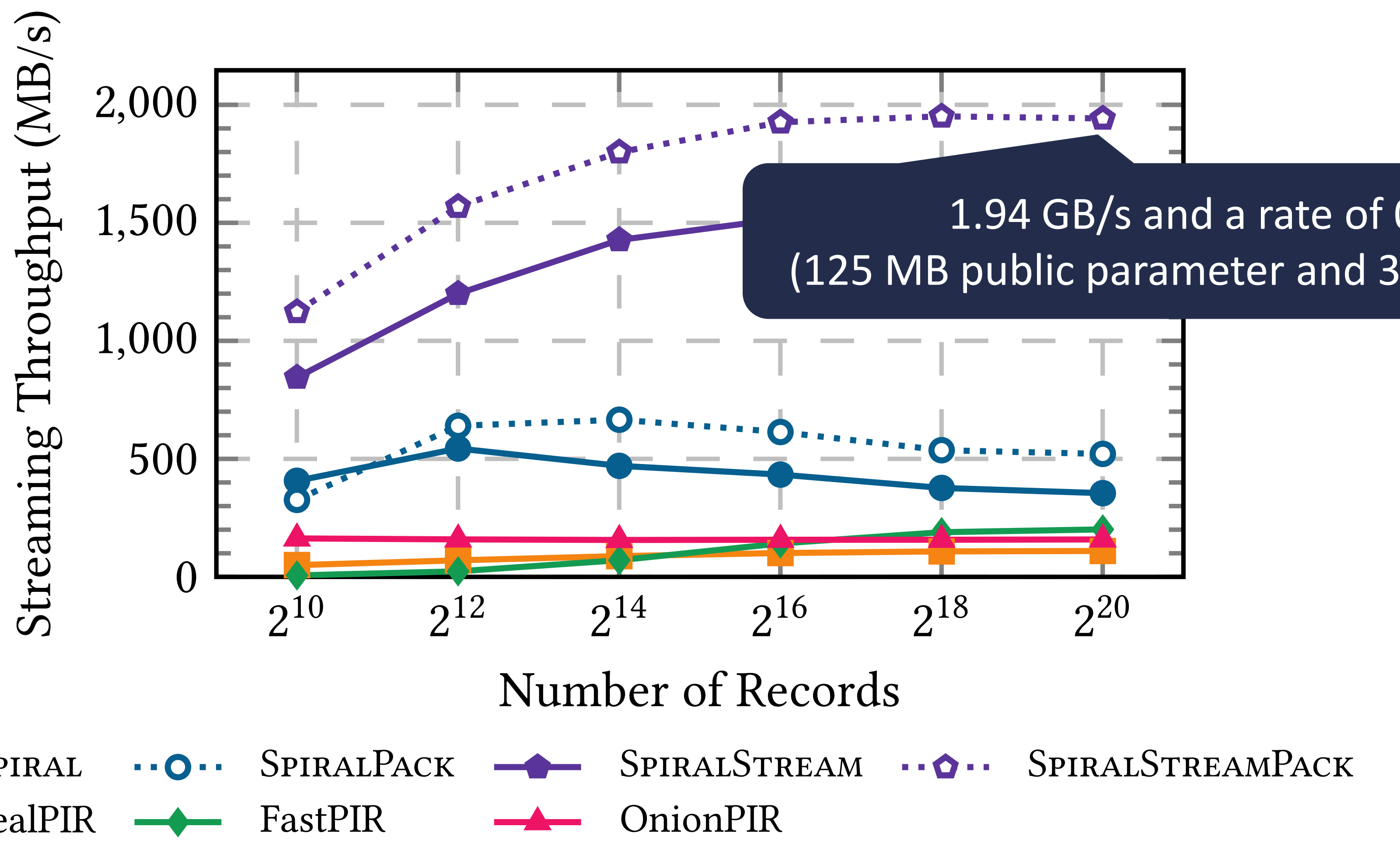
Streaming throughput: ignoring query expansion costs, assuming optimal record size for each system

Packing outperforms non-packed protocol for streaming settings



Packing in the Streaming Setting

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

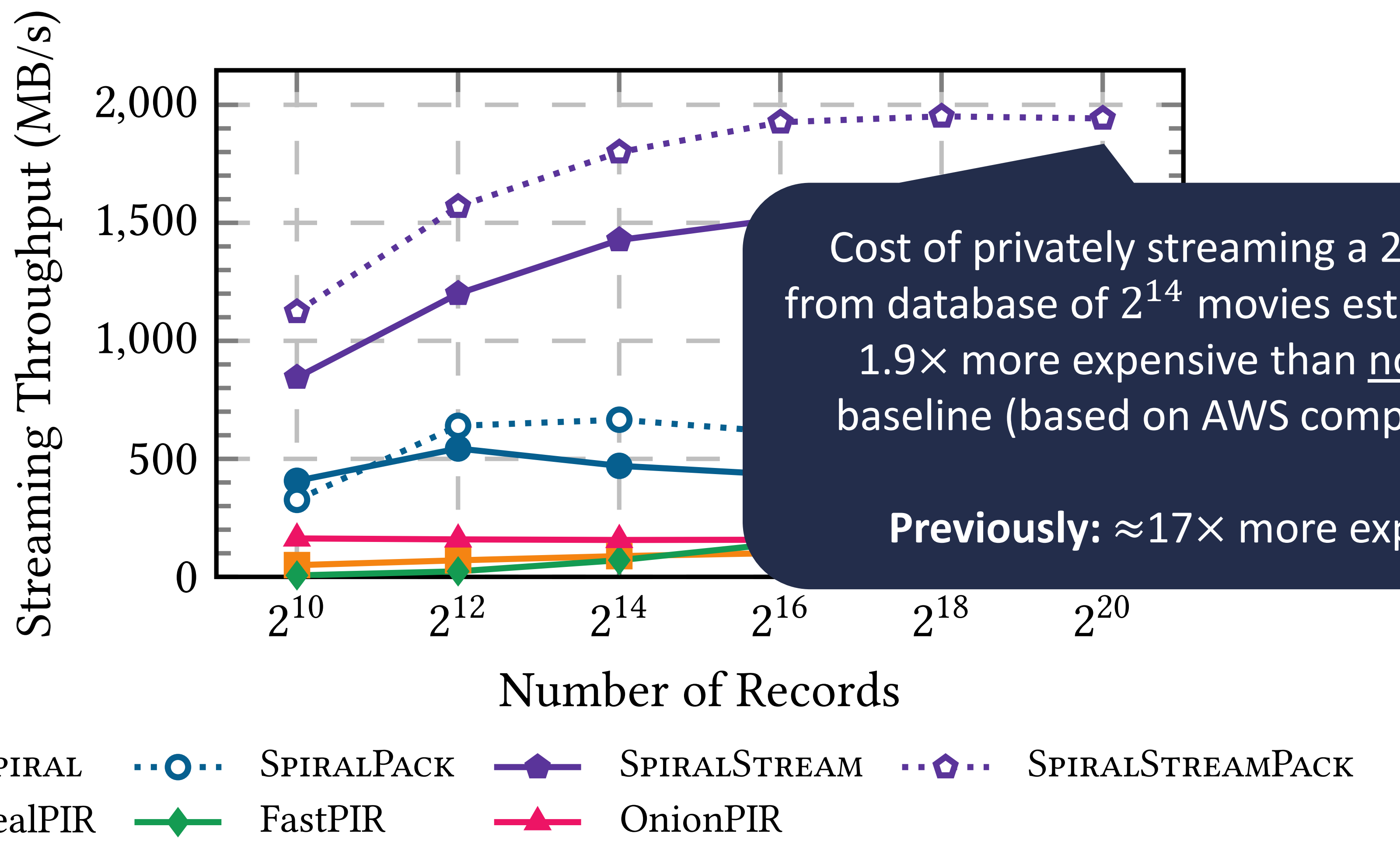


1.94 GB/s and a rate of 0.81
(125 MB public parameter and 30 MB query)

Packing outperforms non-packed protocol for streaming settings

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

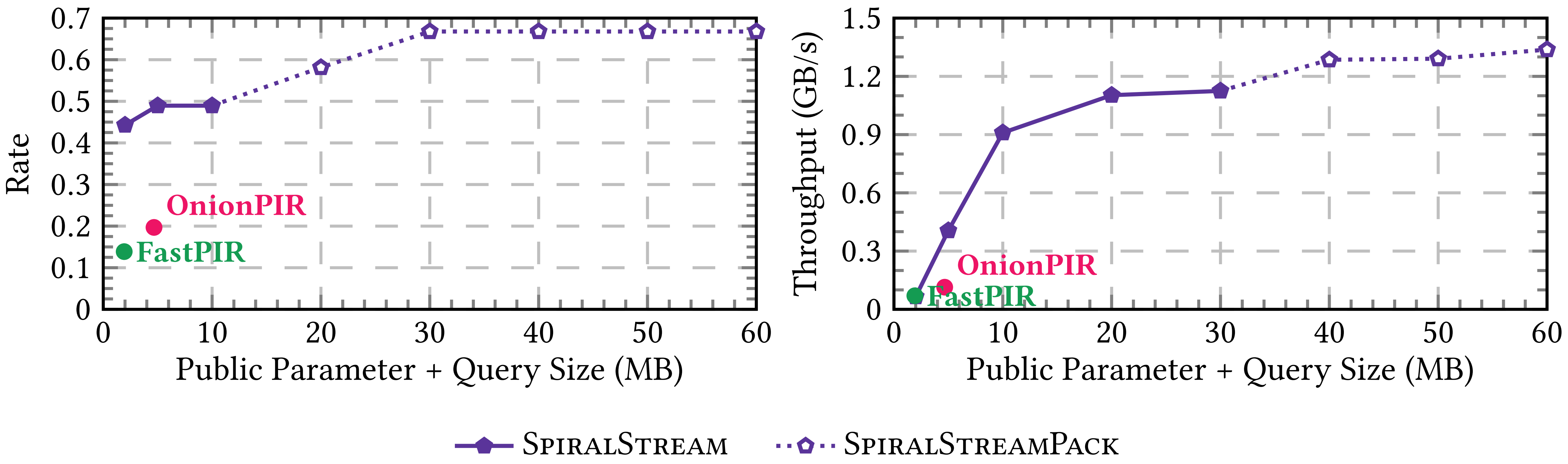
Cost of privately streaming a 2 GB movie from database of 2^{14} movies estimated to be 1.9× more expensive than no-privacy baseline (based on AWS compute costs)

Previously: ≈17× more expensive

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

Scalar Regev → Matrix Regev
Regev → GSW

Query compression

Scalar Regev → Matrix Regev

Response compression
(for large records)

Automatic parameter selection to choose lattice parameters based on database configuration

Base version of SPIRAL

Query size:	14 KB	4.5× smaller
Rate:	0.41	2.1× higher
Throughput:	333 MB/s	2.9× higher

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

Streaming versions of SPIRAL

Rate:	0.81	3.4× smaller
Throughput:	1.9 GB/s	12.3× higher

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

Scalar Regev → Matrix Regev

Response compression
(for large records)

Automatic parameter selection to choose lattice parameters based on database configuration

Base version of SPIRAL

Query size:	14 KB	4.5× smaller
Rate:	0.41	2.1× higher
Throughput:	333 MB/s	2.9× higher

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

Streaming versions of SPIRAL

Rate:	0.81	3.4× smaller
Throughput:	1.9 GB/s	12.3× higher

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`
Code: `https://github.com/menonsamir/spiral`

Thank you!