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

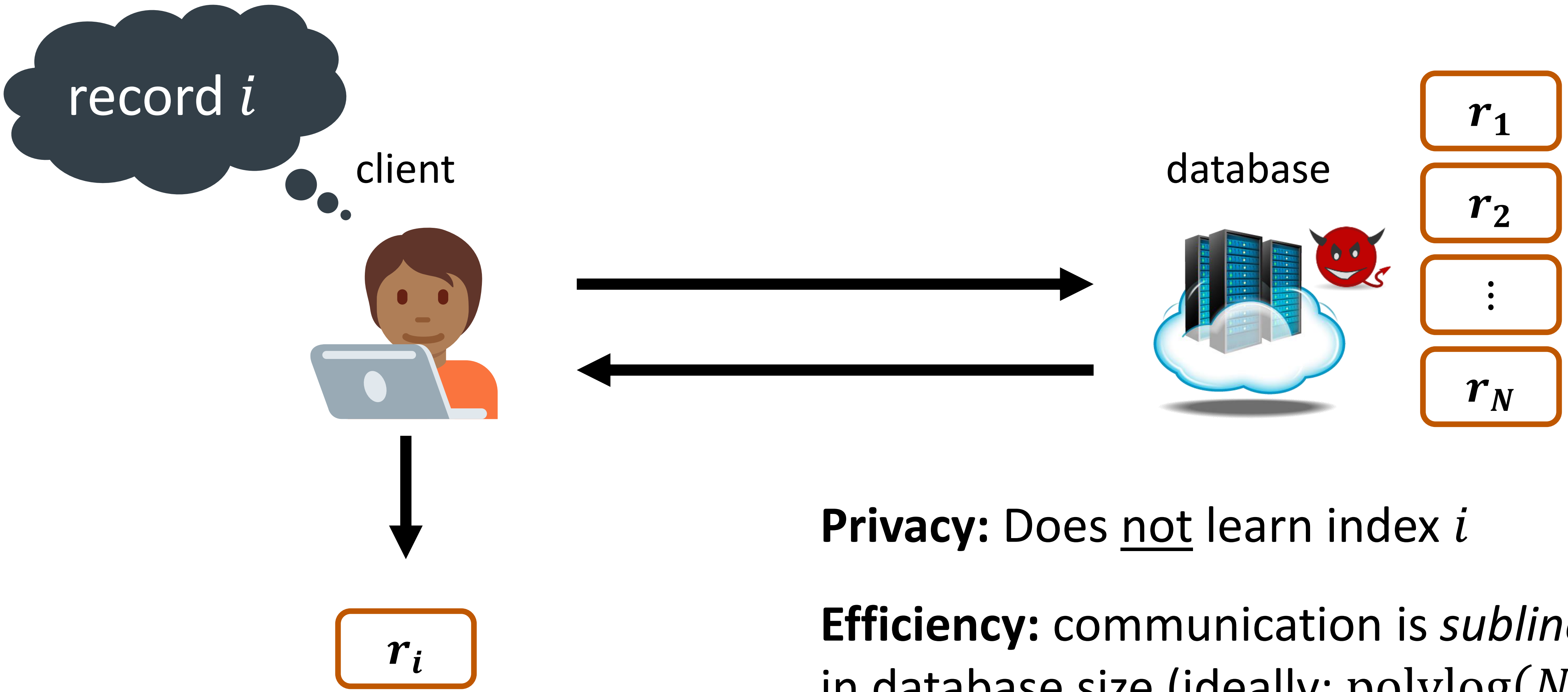
Samir Menon and David Wu

July 2023

IEEE Security and Privacy ("Oakland"), 2022

Private Information Retrieval (PIR)

[CGKS95]



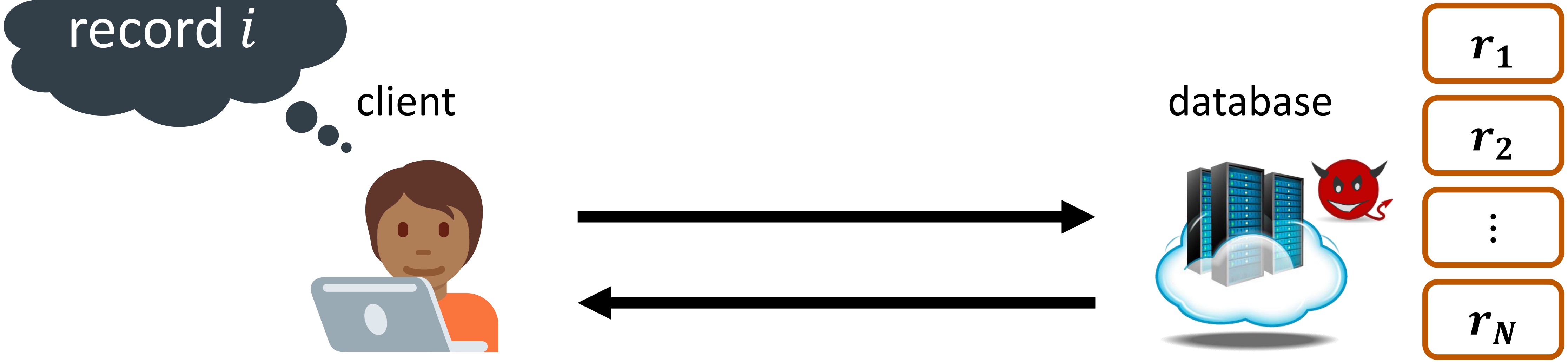
Privacy: Does <u>not</u> learn index i

Efficiency: communication is *sublinear* in database size (ideally: $\mathrm{polylog}(N)$)

Private Information Retrieval (PIR)

[CGKS95]

This talk: single-server setting



Basic building block in many privacy-preserving protocols

- Metadata-private messaging
- Contact discovery
- Private content delivery
- Private DNS
- Private contact tracing
- Private navigation

PIR Metrics of Interest

Rate: *"measures communication overhead in responses"*

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

Throughput: *"measures how fast the server can respond as a function of database size"*

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

Constructions do not rely on server preprocessing so server always performs a <u>linear</u> scan

Goal: make linear scan as fast as possible

The SPIRAL Family of PIR Protocols

Leverages techniques to translate between homomorphic encryption schemes

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)

Independent of database **and** query and can be reused across multiple queries

Comparisons against schemes that do not require server preprocessing (i.e., server hints)

In particular, these exclude subsequent schemes such as FrodoPIR [DPC23], SimplePIR [HHCMV23], and Piano [ZPSZ23]

The SPIRAL Family of PIR Protocols

Leverages techniques to translate between homomorphic encryption schemes

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)

Independent of database **and** query and can be reused across multiple queries

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)

$$\begin{matrix} r_{11} & r_{12} & r_{13} & r_{14} & \times & 0 & \\ & & & & & & + \\ r_{21} & r_{22} & r_{23} & r_{24} & \times & 0 & \\ & & & & & & + \\ r_{31} & r_{32} & r_{33} & r_{34} & \times & 1 & \\ & & & & & & + \\ r_{41} & r_{42} & r_{43} & r_{44} & \times & 0 & \end{matrix}$$

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

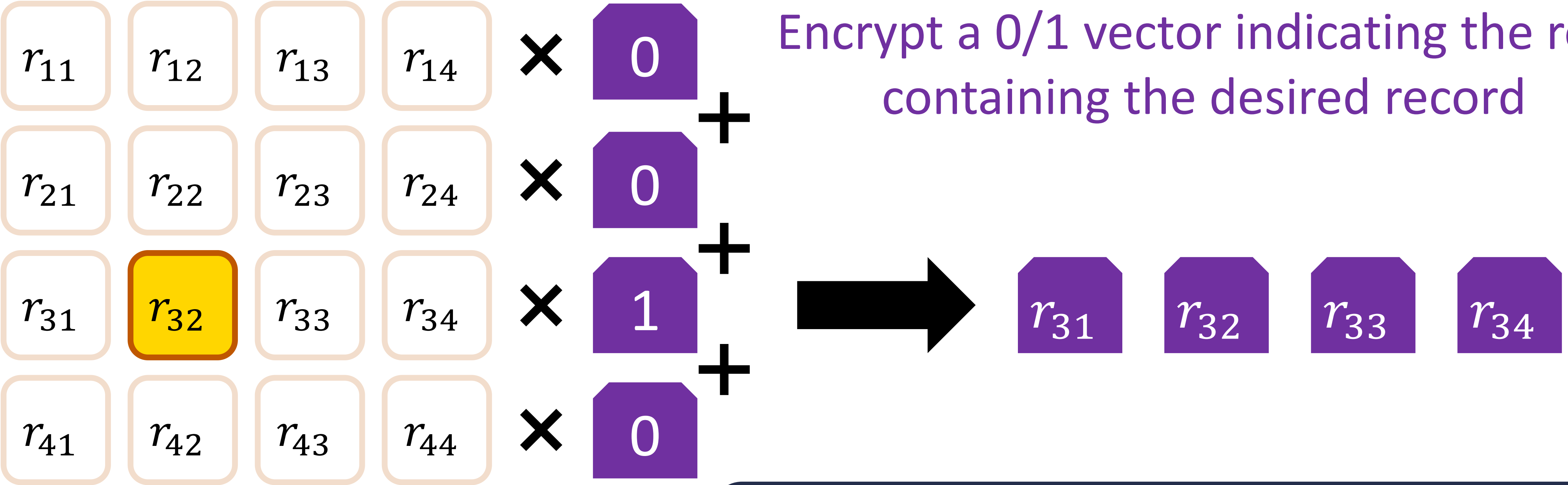
Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

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

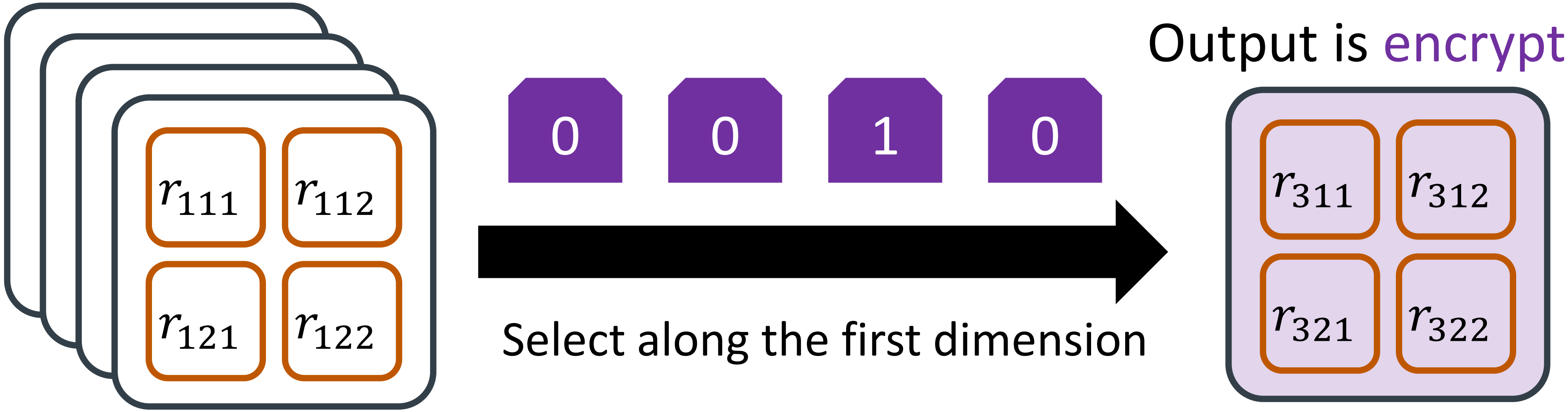
Response size: $O_\lambda(\sqrt{N})$

Homomorphically compute product between query vector and database matrix

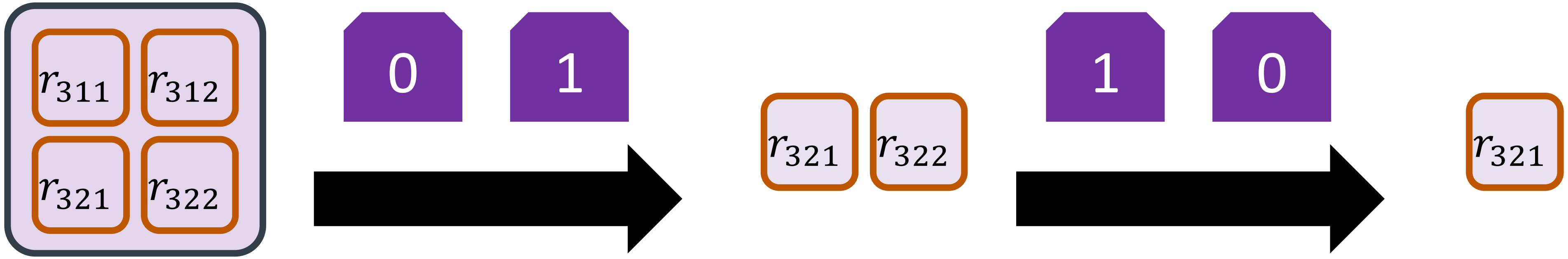
PIR from Homomorphic Encryption

[KO97]

Sub-$\sqrt{N}$ communication: view the database as hypercube



Approach: Use homomorphic multiplication [GH19, PT20, ALPRSSY21, MCR21]



SPIRAL: Composing FHE Schemes

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

Ciphertexts in lattice-based schemes are noisy encodings

Homomorphic operations increase noise; more noise = larger parameters = less efficiency

Scheme 1: Regev's encryption scheme [Reg04]

Small ciphertexts (amortized); only supports additive homomorphism

18 KB plaintext	⇒	43 KB ciphertext	(2.4× expansion)
1 MB plaintext	⇒	1.3 MB ciphertext	(1.3× expansion)

allows the use of smaller lattice dimension and modulus

Scheme 2: Gentry-Sahai-Waters encryption scheme [GSW13]

Large ciphertexts; supports homomorphic multiplication (with additive noise growth)

1 bit plaintext	⇒	2.5 **MB** ciphertext

Can we get the best of both worlds?

SPIRAL: Composing FHE Schemes

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

Ciphertexts in lattice-based schemes are noisy encodings

Homomorphic operations increase noise; more noise = larger parameters = less efficiency

Scheme 1: Regev's encryption scheme [Reg04]

Small ciphertexts (amortized); only supports additive homomorphism

18 KB plaintext	⇒	43 KB ciphertext	(2.4× expansion)
1 MB plaintext	⇒	1.3 MB ciphertext	(1.3× expansion)

allows the use of smaller lattice dimension and modulus

Scheme 2: Gentry-Sahai-Waters encryption scheme [GSW13]

Large ciphertexts; supports homomorphic multiplication (with additive noise growth)

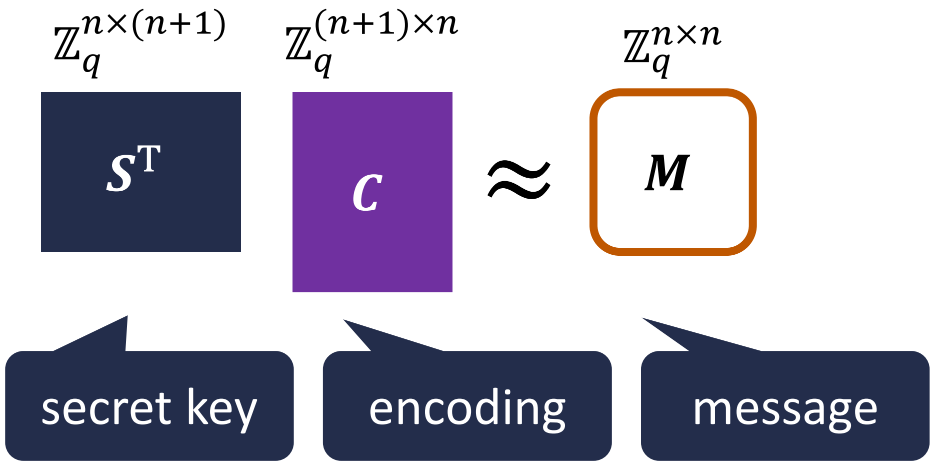
1 bit plaintext	⇒	2.5 **MB** ciphertext

SPIRAL: Use GSW for homomorphic multiplication, Regev for communication

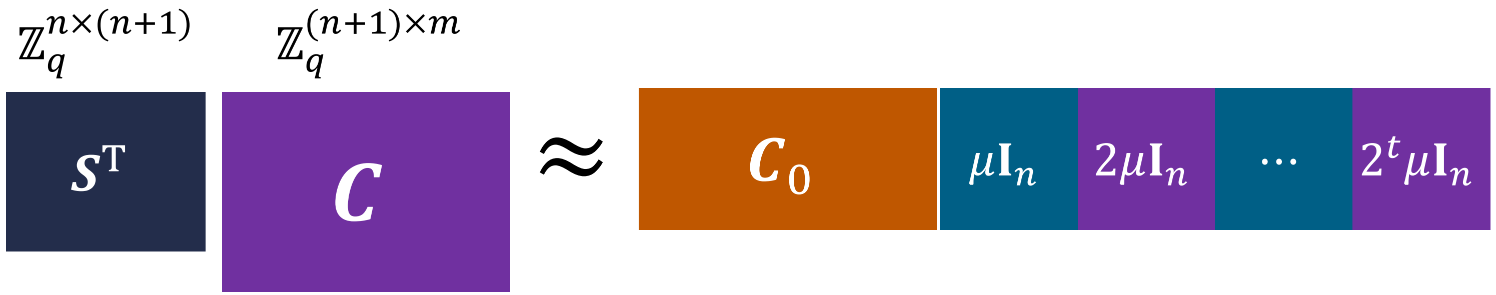
Regev-GSW Homomorphisms

[CGGI18]

Regev encoding of $\boldsymbol{M} \in \mathbb{Z}_q^{n \times n}$:



GSW encoding of $\mu \in \mathbb{Z}_q$:



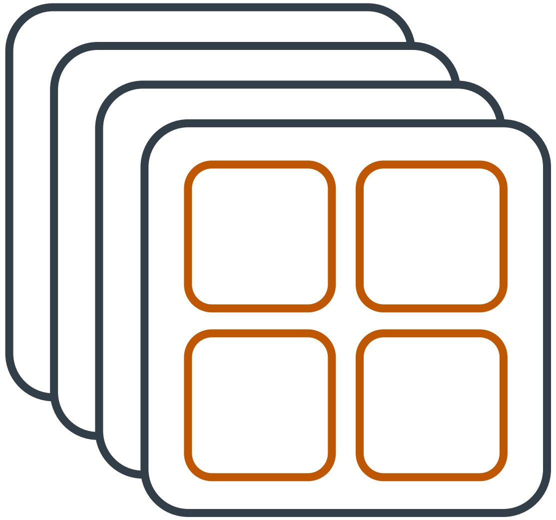
Redundant encoding of μ

Key property: given Regev encoding of message $\boldsymbol{M}$ and GSW encoding of scalar μ, can efficiently derive a Regev encoding of $\mu \cdot \boldsymbol{M}$

Simplification: *actual scheme operates over polynomial rings*

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} Regev encodings

0 1 0 0 0 0

Technically uses matrices

Indicator for index along first dimension

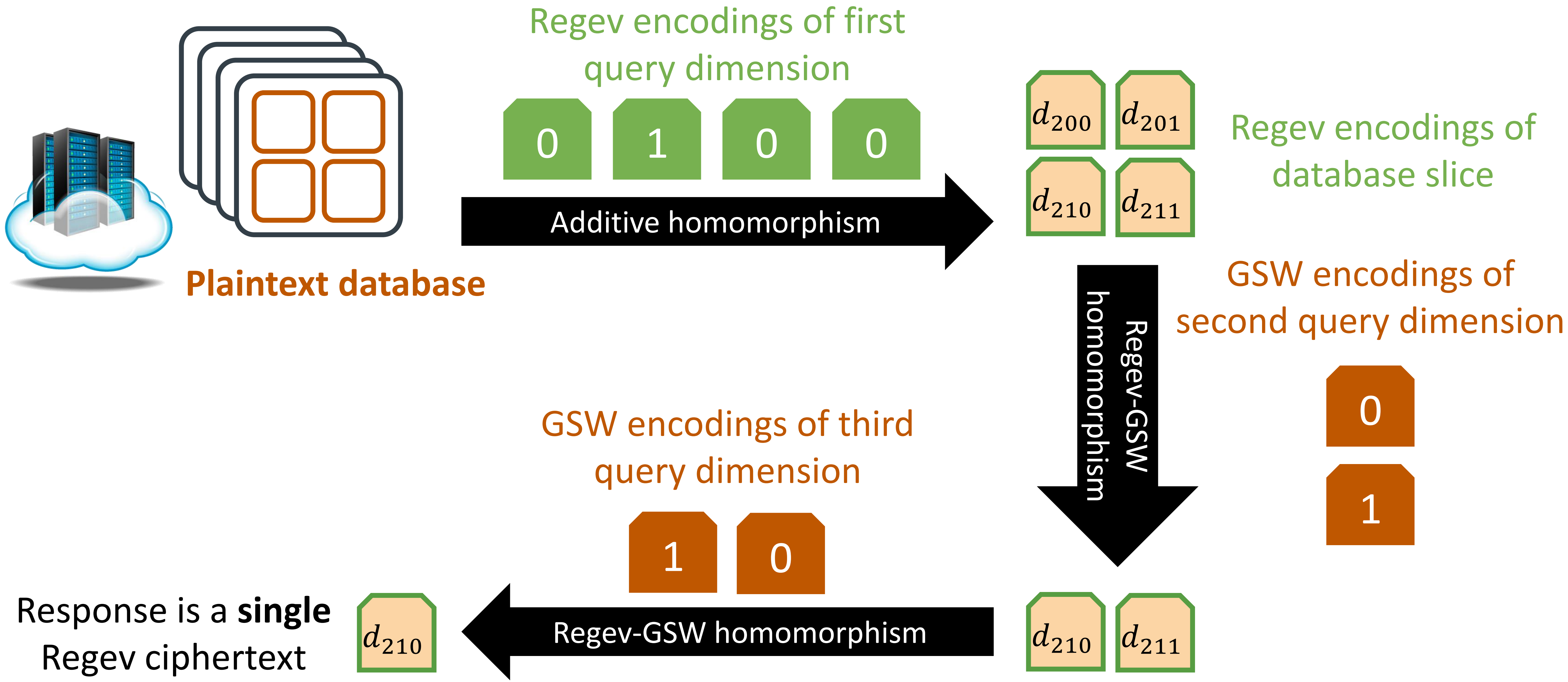
Query contains ν_2 GSW encodings

0 1 1 0

Indicator for index along subsequent dimensions

The Gentry-Halevi Blueprint

[GH19]



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

Query contains 2^{ν_1} Regev encodings

0 1 0 0 0 0

Indicator for index along first dimension

Query contains ν_2 GSW encodings

0 1 1 0

Indicator for index along subsequent dimensions

Estimated size:
4 MB/ciphertext

Estimated query size:
30 MB

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

Query contains 2^{ν_1} Regev encodings

0 1 0 0 0 0

Indicator for index along first dimension

Query contains ν_2 GSW encodings

0 1 1 0

Indicator for index along subsequent dimensions

SealPIR query size:
66 KB

Estimated query size:
30 MB

The SPIRAL Protocol

Key idea: Expand Regev encodings into GSW encodings

OnionPIR [MCR21]: use Regev-GSW homomorphism for the **scalar** case

This work: General toolkit to translate between scalar/matrix Regev and GSW

Transformations useful for query compression and response packing

Assembling GSW Encodings

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

$$\mu\boldsymbol{S}^{\mathrm{T}}\boldsymbol{G} = \begin{bmatrix} \boldsymbol{C}_0 & \mu\mathbf{I}_n & 2\mu\mathbf{I}_n & 2^2\mu\mathbf{I}_n & \cdots & 2^t\mu\mathbf{I}_n \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}_0 & \boldsymbol{B}_1 & \boldsymbol{B}_2 & \cdots & \boldsymbol{B}_t \end{bmatrix}$$

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

Assembling GSW Encodings

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

$$\mu\boldsymbol{S}^{\mathrm{T}}\boldsymbol{G} = \left[\begin{array}{c|ccccc} \boldsymbol{C}_0 & \mu\mathbf{I}_n & 2\mu\mathbf{I}_n & 2^2\mu\mathbf{I}_n & \cdots & 2^t\mu\mathbf{I}_n \end{array}\right]$$

$$\approx \qquad\qquad \approx$$

$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C} = \left[\begin{array}{c|ccccc} \boldsymbol{S}^{\mathrm{T}}\boldsymbol{A} & \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_0 & \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_1 & \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_2 & \cdots & \boldsymbol{S}^{\mathrm{T}}\boldsymbol{B}_t \end{array}\right]$$

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

Leverage “key-switching”

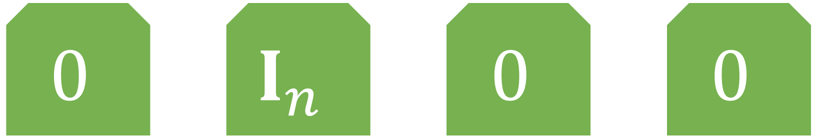
Standard Regev encodings of $\mu, 2\mu, \ldots, 2^t\mu$

[see paper for details]

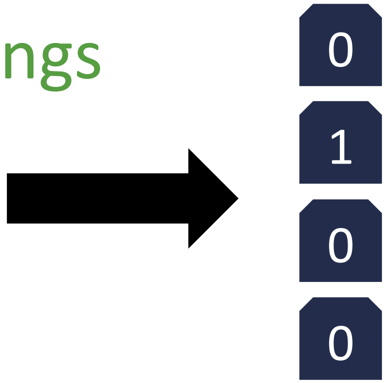
Query Compression in SPIRAL

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 encodings

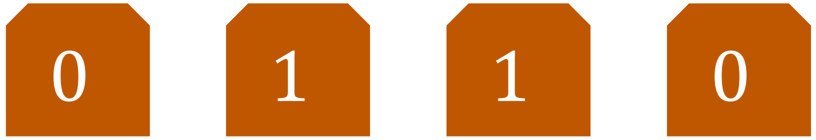


Indicator for index along first dimension



1 Compress into scalar Regev encodings

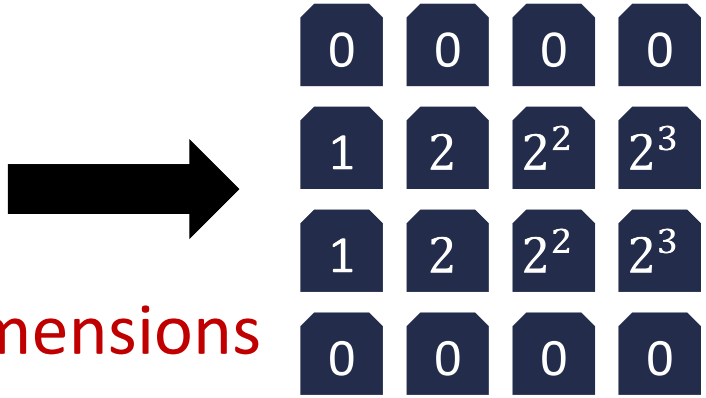
Query contains ν_2 GSW encodings

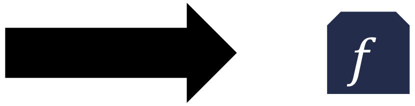


Indicator for index along subsequent dimensions

2 Pack scalars into <u>single</u> encodings

[ACLS18, CCR19]



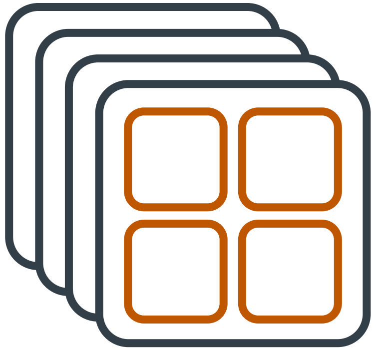


Similar techniques possible for *response* compression [see paper]

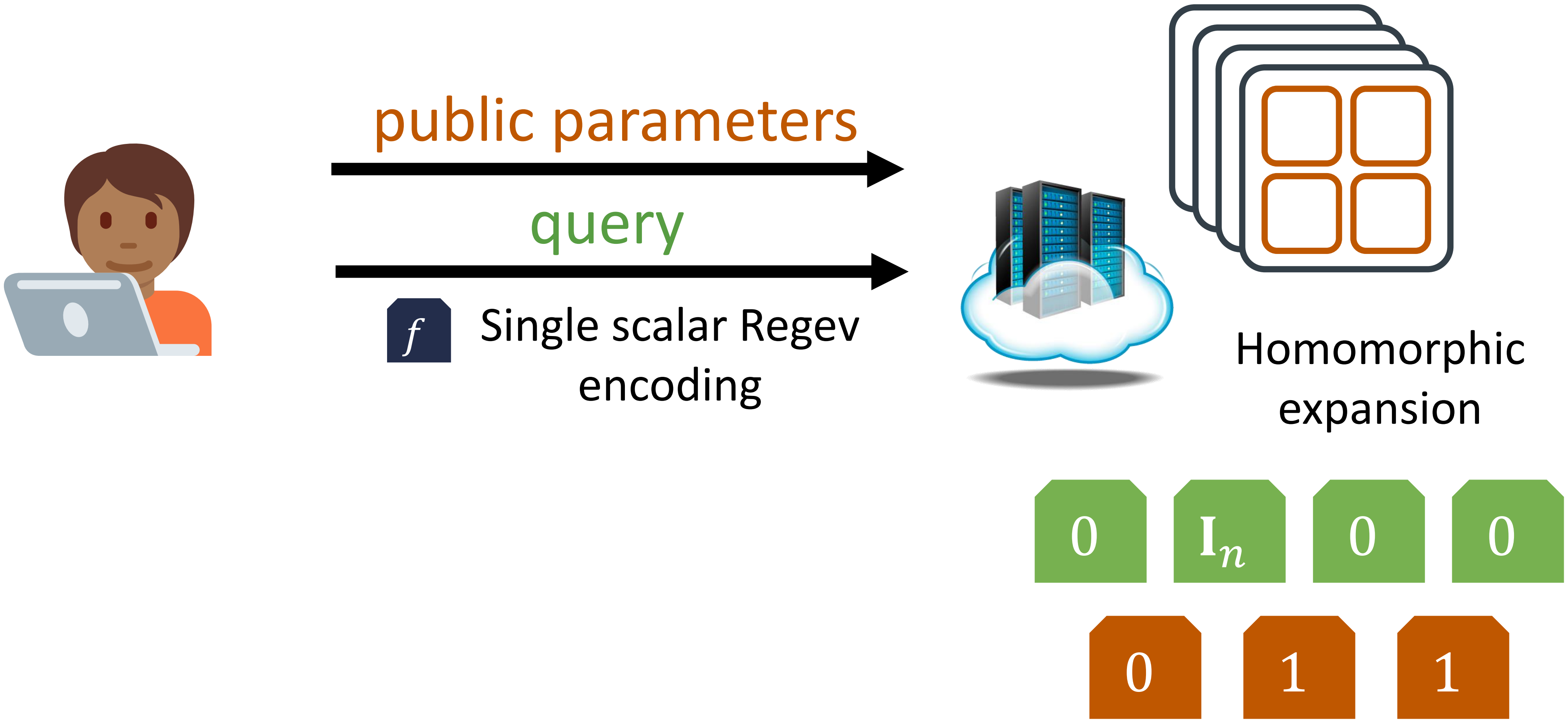
The SPIRAL Protocol

public parameters

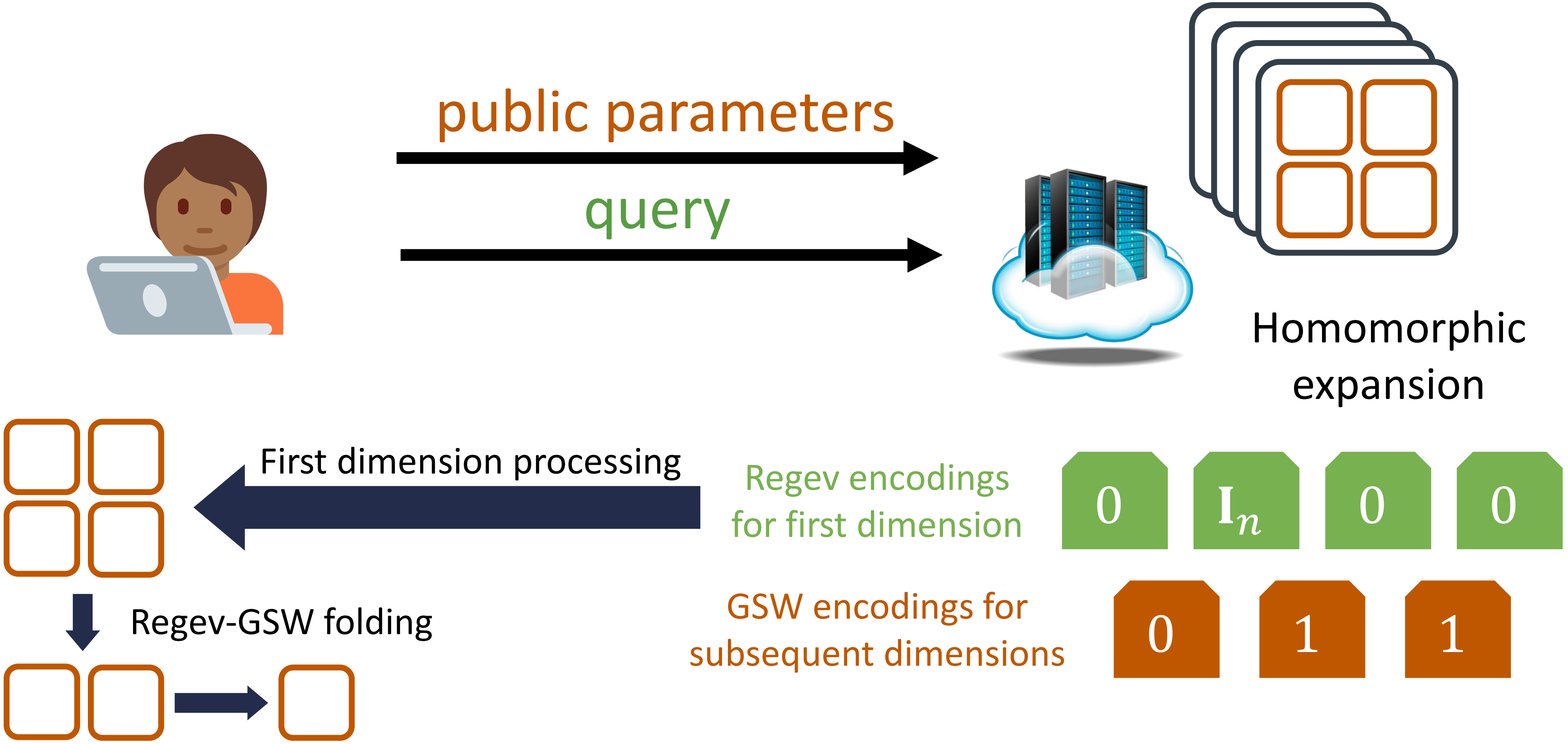
Key-switching matrices for ciphertext expansion and translation



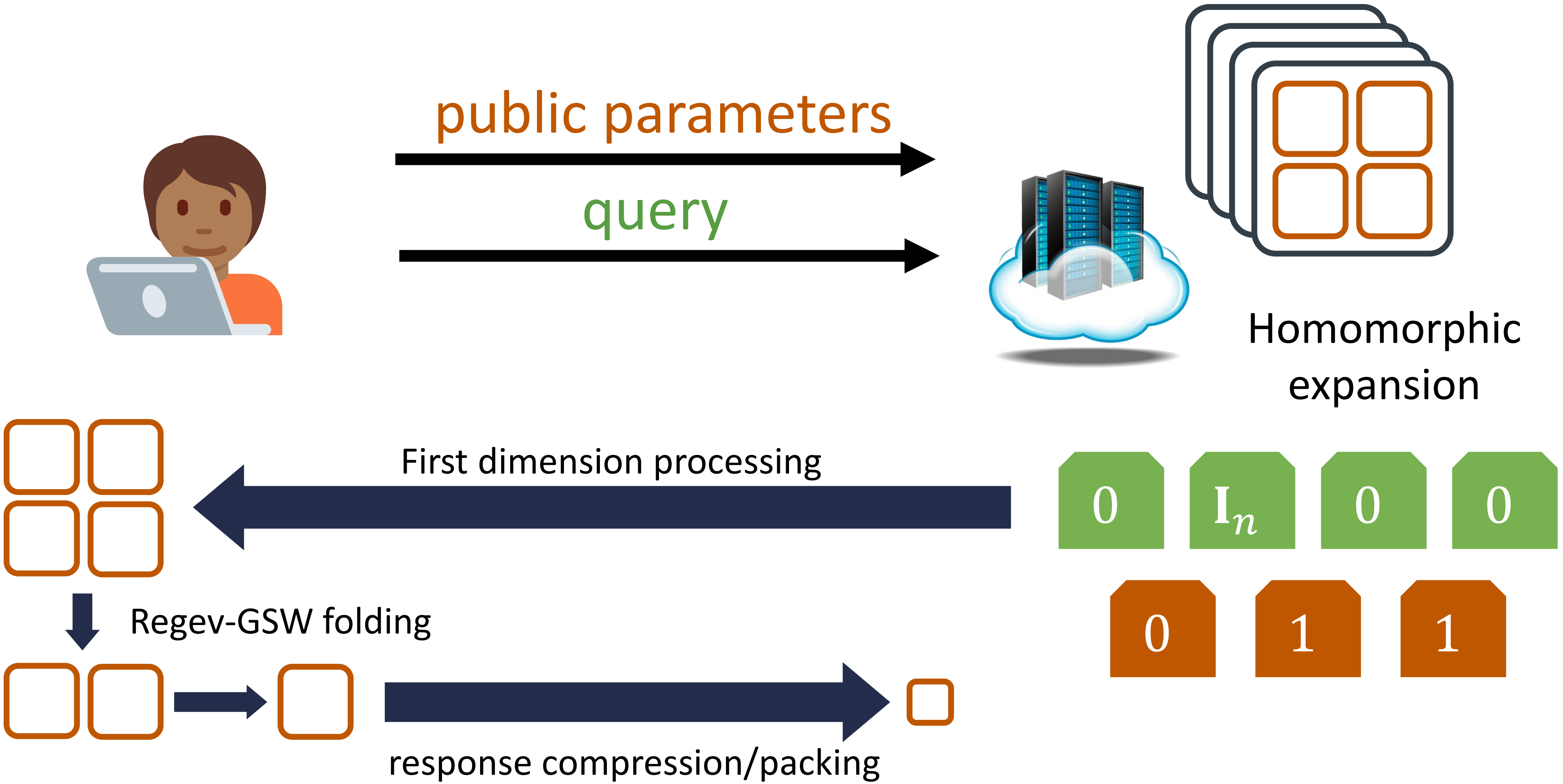
The SPIRAL Protocol



The SPIRAL Protocol



The SPIRAL Protocol



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×

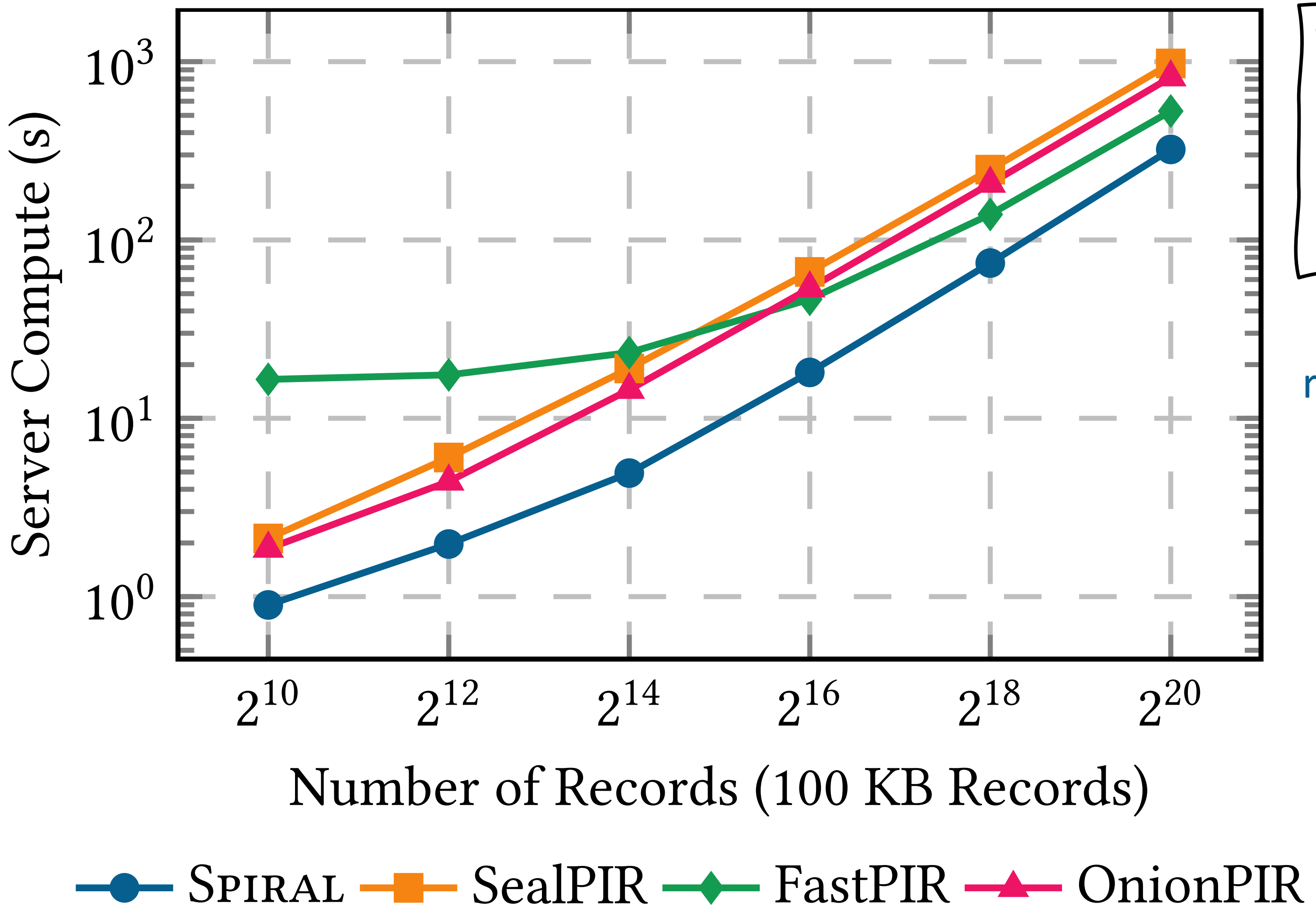
reduce compute time by 2×

increase public parameter size by 3.6×

Comparisons against schemes that do not require server preprocessing (i.e., server hints)

In particular, these exclude subsequent schemes such as FrodoPIR, SimplePIR, and Piano

Basic Comparisons (with Large 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

The Streaming Setting

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]

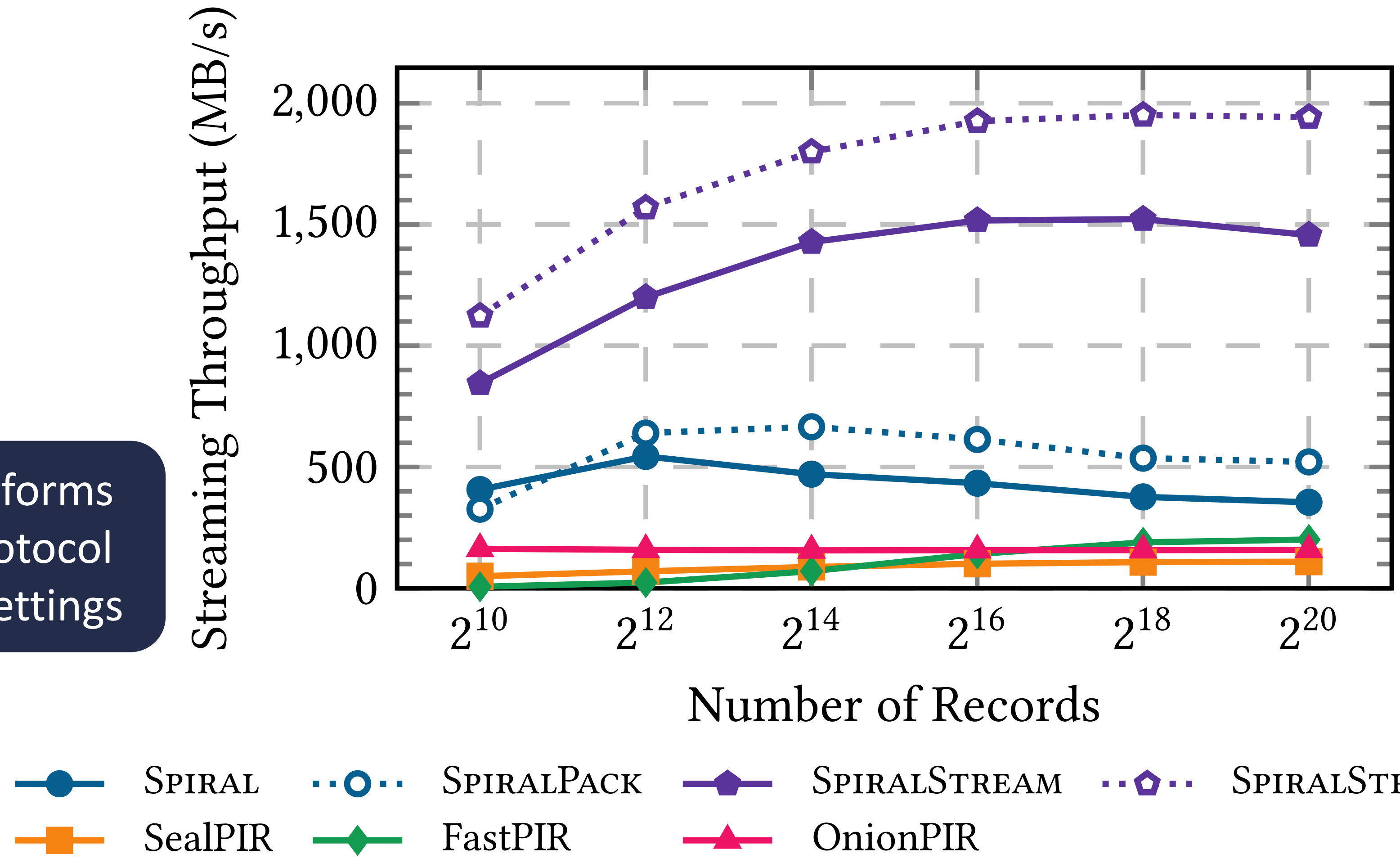
Goal: minimize online costs (i.e., server compute, response size)

- Consider larger public parameters or query size (amortized over lifetime of stream)
- **Approach:** send all of the Regev encodings (and only use Regev-GSW translation)

The Streaming Setting

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

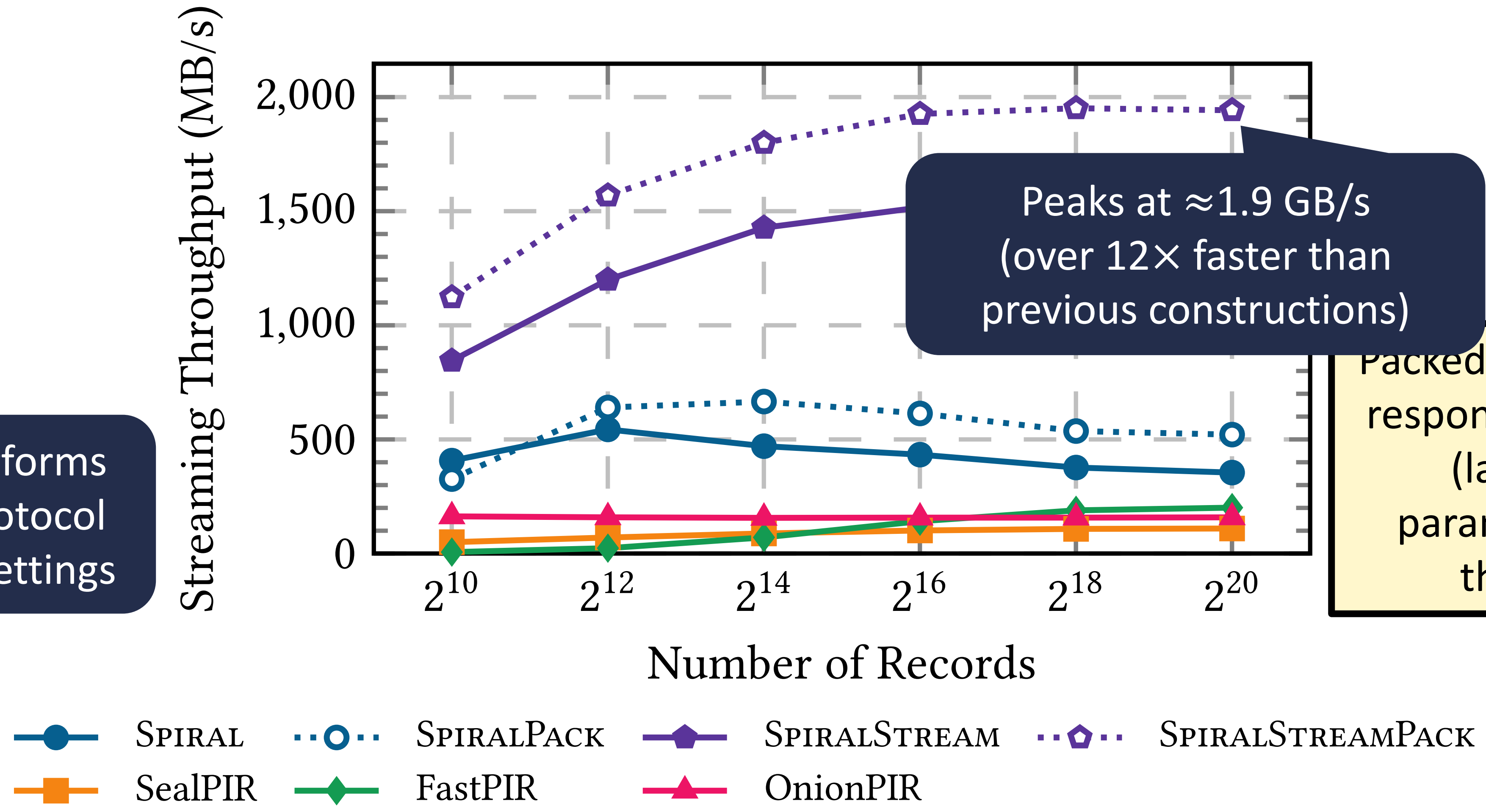
Packing outperforms non-packed protocol for streaming settings



Packed versions rely on response compression (larger public parameters, higher throughput)

The Streaming Setting

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



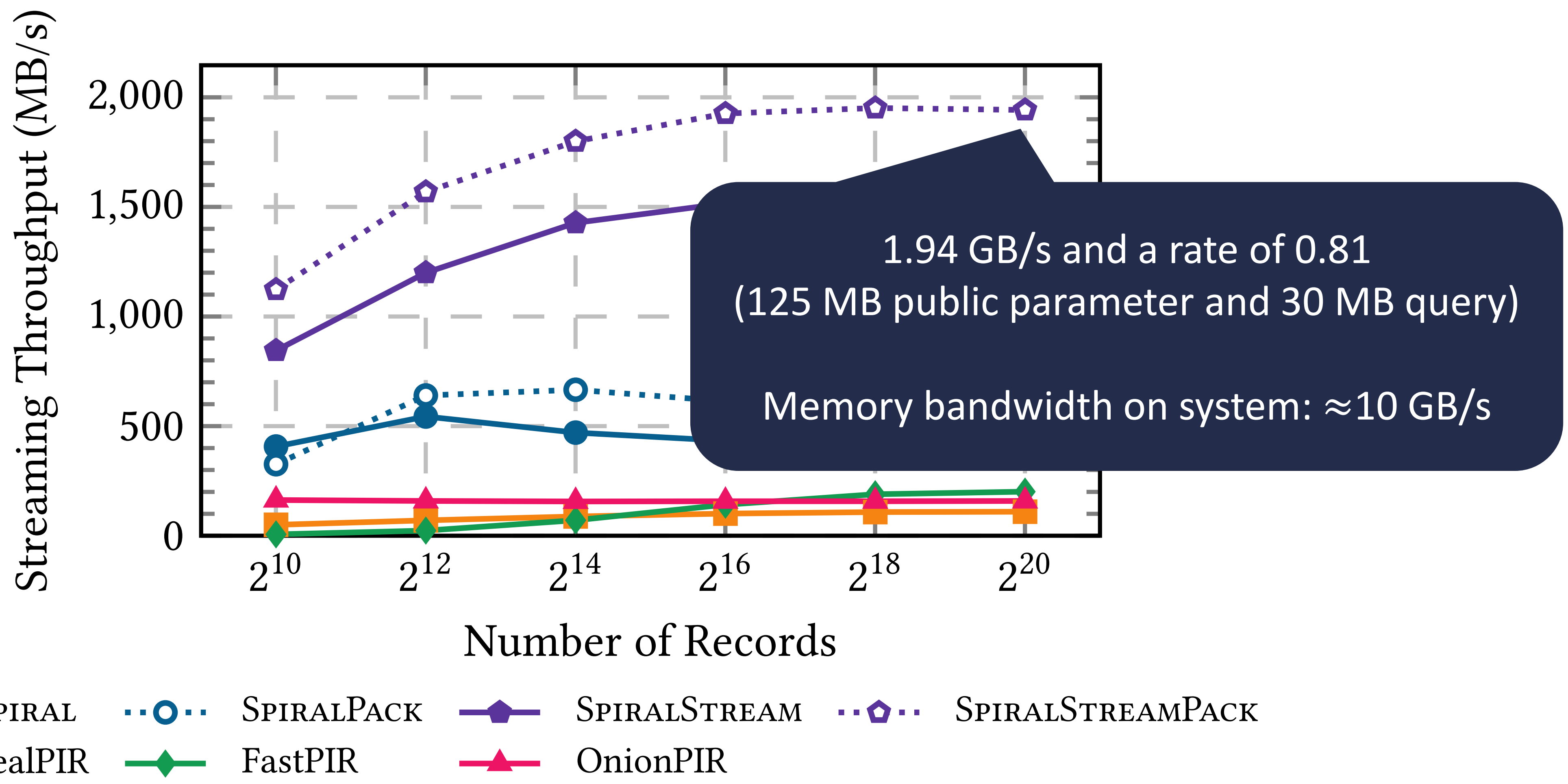
Peaks at ≈1.9 GB/s (over 12× faster than previous constructions)

Packing outperforms non-packed protocol for streaming settings

Packed versions rely on response compression (larger public parameters, higher throughput)

The Streaming Setting

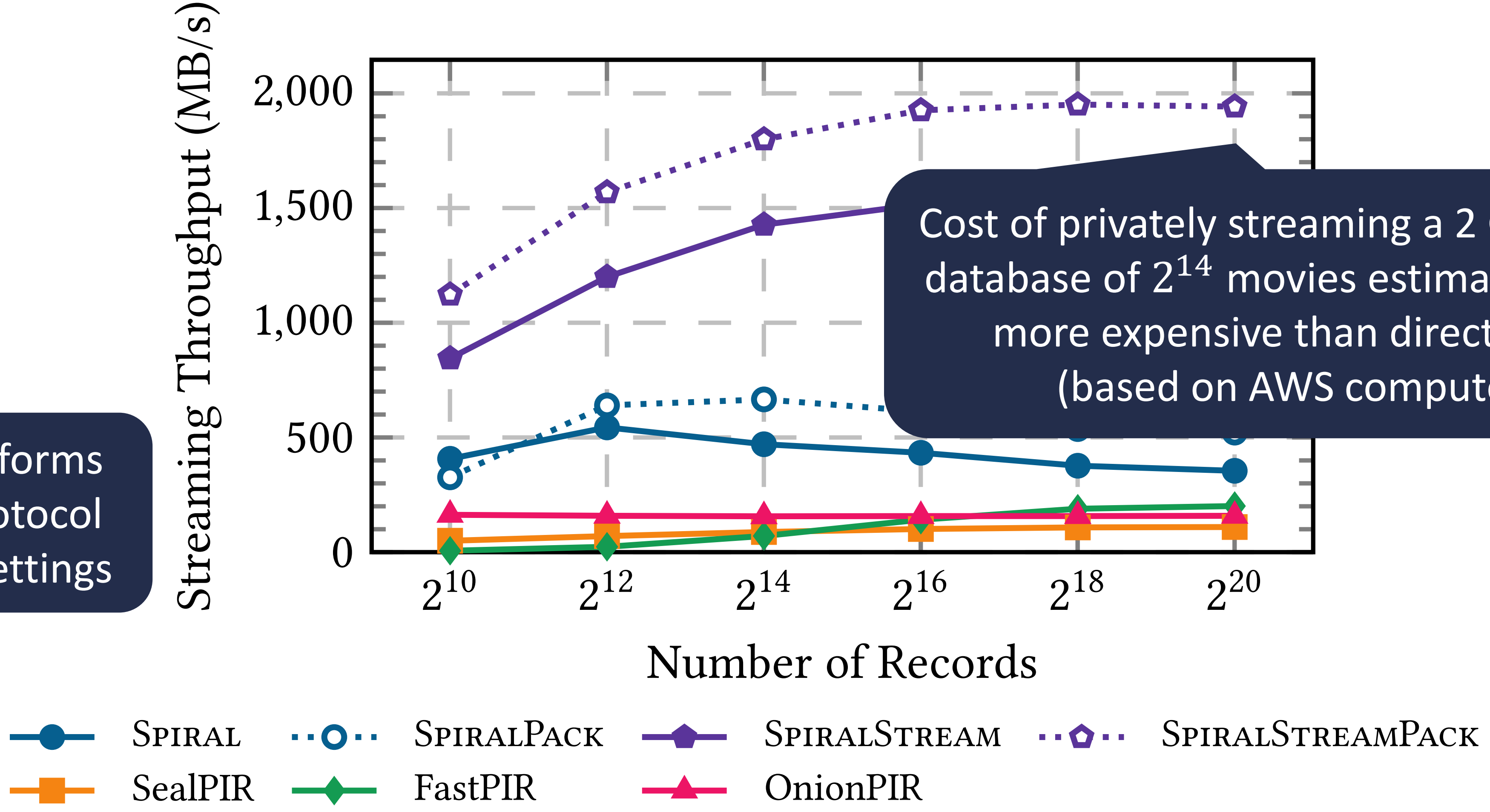
Streaming throughput: ignoring query expansion costs, assuming optimal record size for each system



Packing outperforms non-packed protocol for streaming settings

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 direct download (based on AWS compute costs)

Packing outperforms non-packed protocol for streaming settings

The SPIRAL Family of PIR

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

- Used for both query compression and response compression

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

Some Recent Developments in PIR

Server preprocessing (client downloads hint at beginning of protocol)

- SimplePIR, DoublePIR [HHCMV23]
  - Very high throughput (nearly memory bandwidth!)
  - Well suited for databases with small records (a few bits)
- Piano [ZPSZ23]
  - **Sublinear** server computational costs (can scale better to databases that are $>$100 GB)
  - Preprocessing phase requires *streaming* the entire database

Server preprocessing (*without* hint)

- Doubly-efficient PIR [LMW23]
  - Server encodes the database to answer queries in sublinear time
  - Concrete efficiency not yet clear

Some Recent Developments in PIR

Server preprocessing (client downloads hint at

SimplePIR, DoublePIR [HHCMV23]

Very high throughput (nearly memory band

Well suited for databases with small records

Piano [ZPSZ23]

Sublinear server computational costs (can scale better to databases that are >100 GB)

Preprocessing phase requires *streaming* the entire database

Server preprocessing (*without* hint)

Doubly-efficient PIR [LMW23]

Server encodes the database to answer que

Concrete efficiency not yet clear

Many other directions!

- Protocols for batch queries [MR23]
- Supporting keyword search [PSY23]
- Authenticating the response [CNCWF23]

Takeaway: PIR is an exciting area to work in with many different trade-offs to explore!

The SPIRAL Family of PIR

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

- Used for both query compression and response compression

Automatic parameter selection to choose lattice parameters based on database configuration

Paper: `https://eprint.iacr.org/2022/368`
Code: `https://github.com/menonsamir/spiral-rs`
Demo: `https://spiralwiki.com`

Thank you!