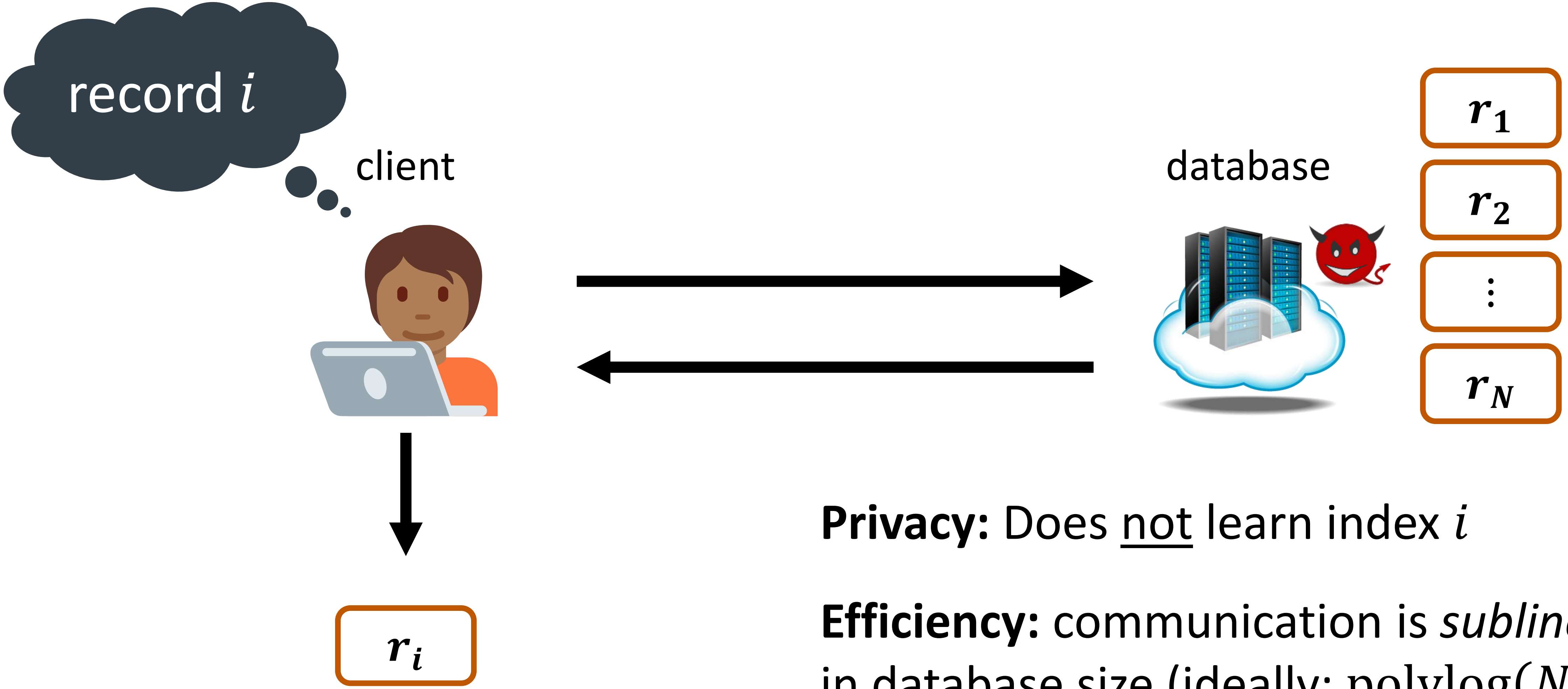
Recent Advancements in Private Information Retrieval

David Wu

based on joint works with Alexander Burton and Samir Menon

Private Information Retrieval (PIR)

[CGKS95]

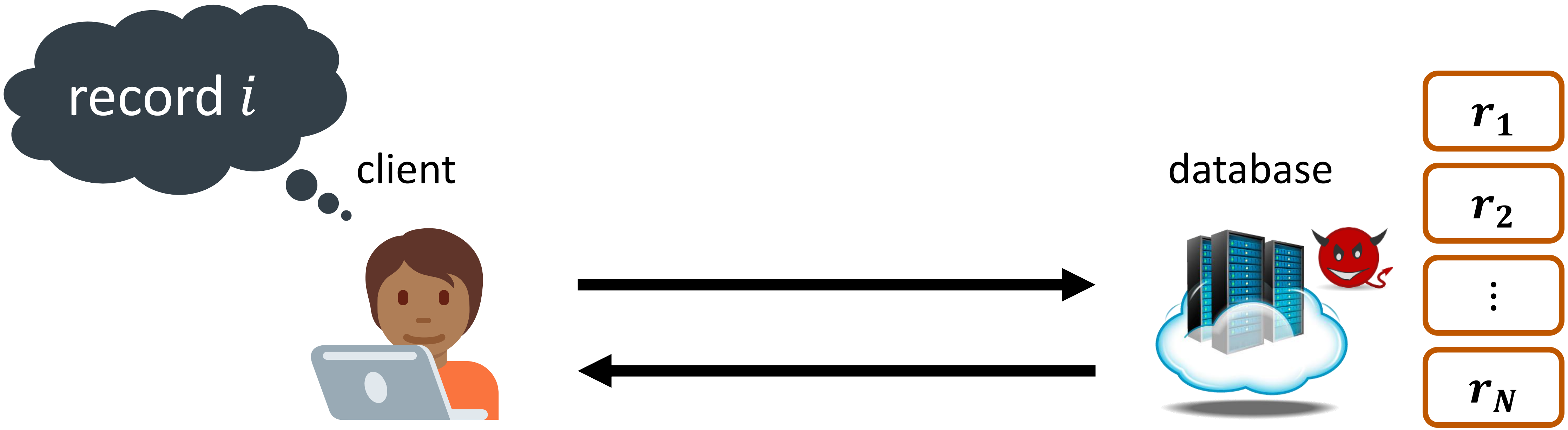


Privacy: Does not learn index i

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

Private Information Retrieval (PIR)

[CGKS95]

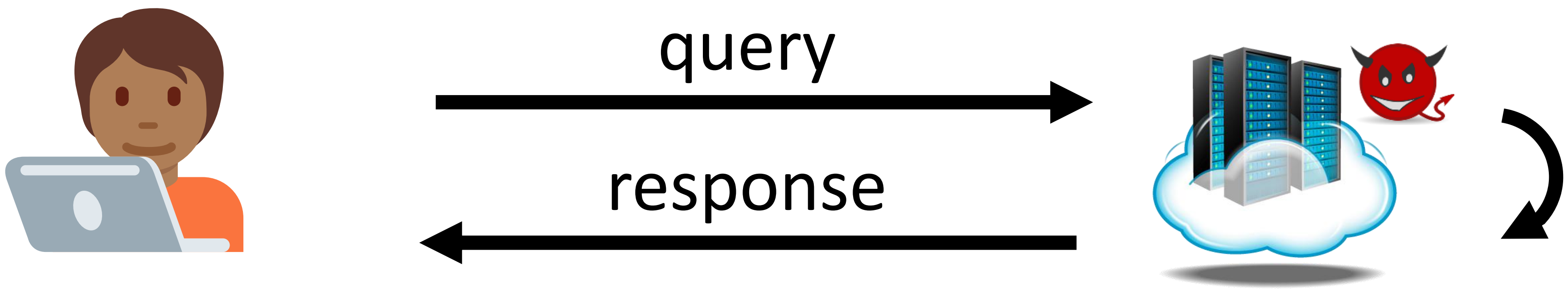


Basic building block in many privacy-preserving protocols

- Metadata-private messaging
- Contact discovery
- Private contact tracing
- Certificate transparency auditing
- Private web search
- Private DNS
- Private content delivery
- Private navigation
- Password breach checking

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**

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

“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

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

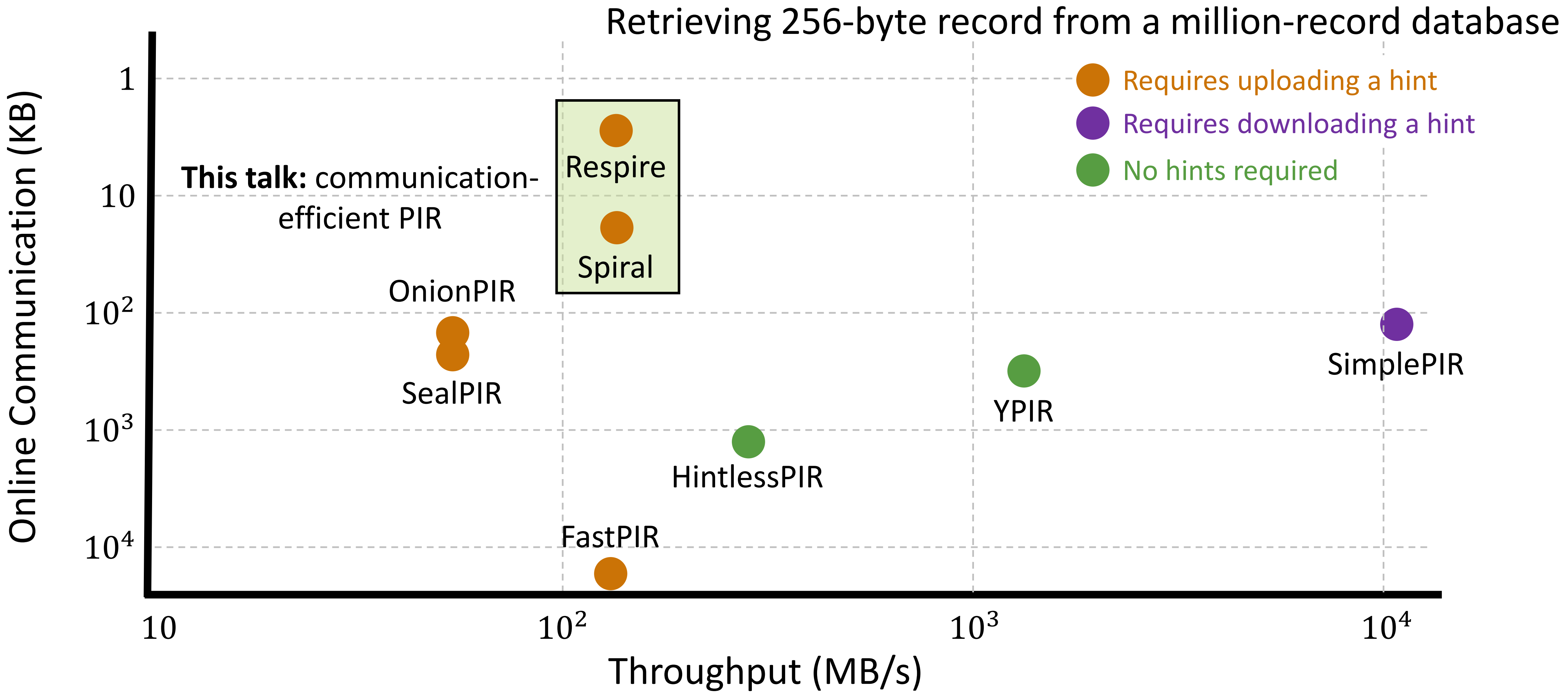
"measures communication overhead in responses"

2 Server Throughput

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

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

Communication/Computation Trade-offs in PIR



PIR from Homomorphic Encryption

[KO97]

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

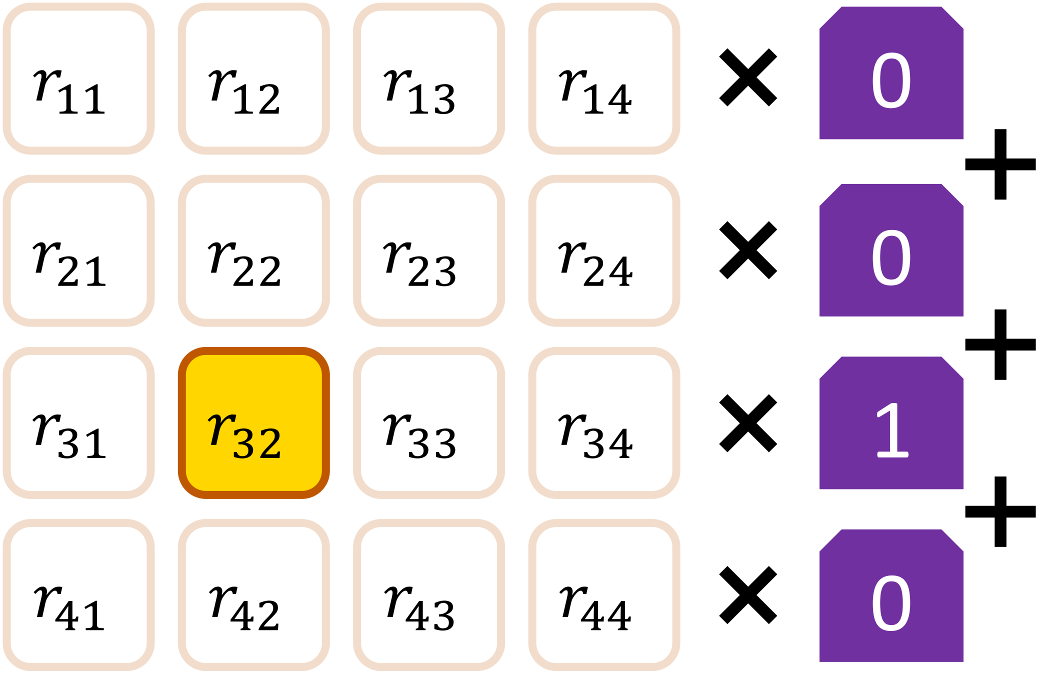
r_{11}	r_{12}	r_{13}	r_{14}
r_{21}	r_{22}	r_{23}	r_{24}
r_{31}	r_{32}	r_{33}	r_{34}
r_{41}	r_{42}	r_{43}	r_{44}

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

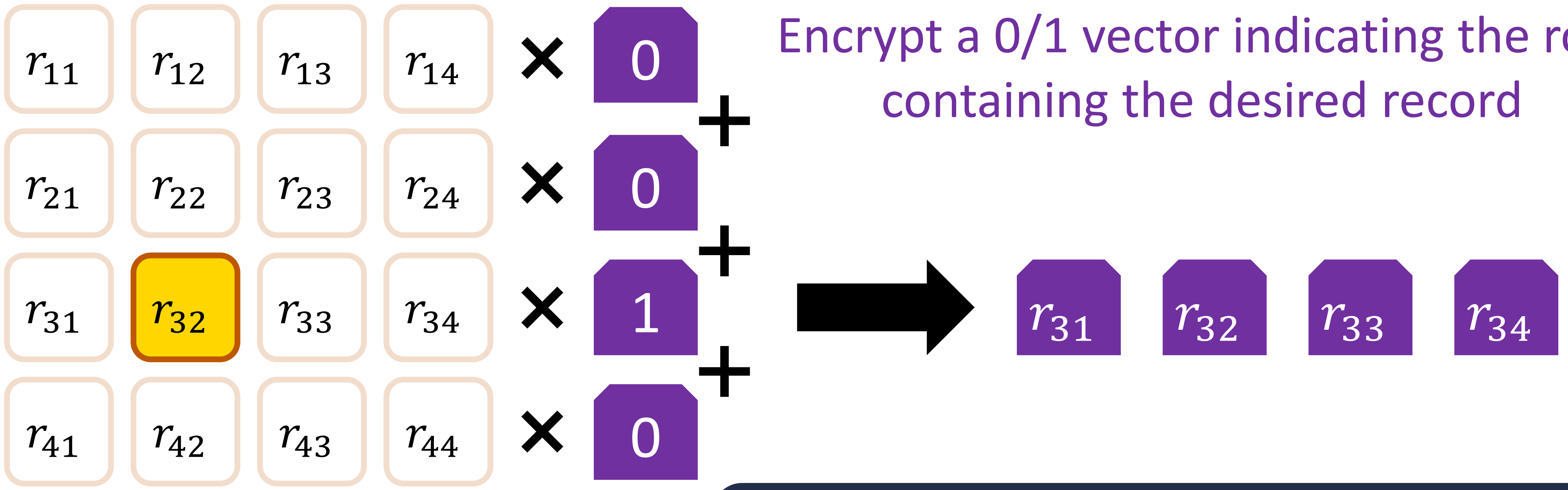
Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

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}

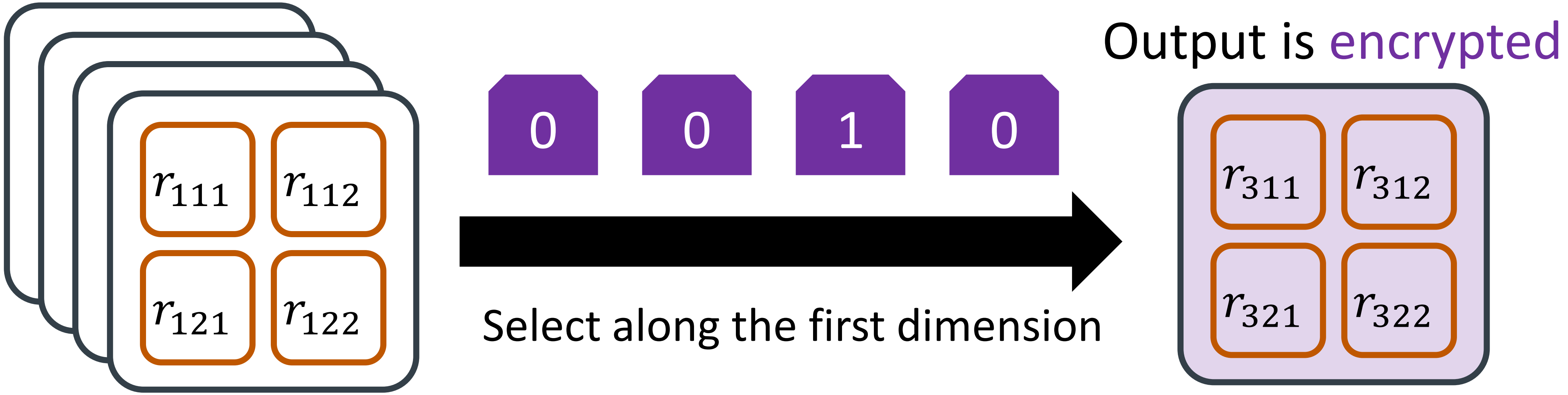
Response size: $O_\lambda(\sqrt{N})$

Homomorphically compute product between query vector and database matrix

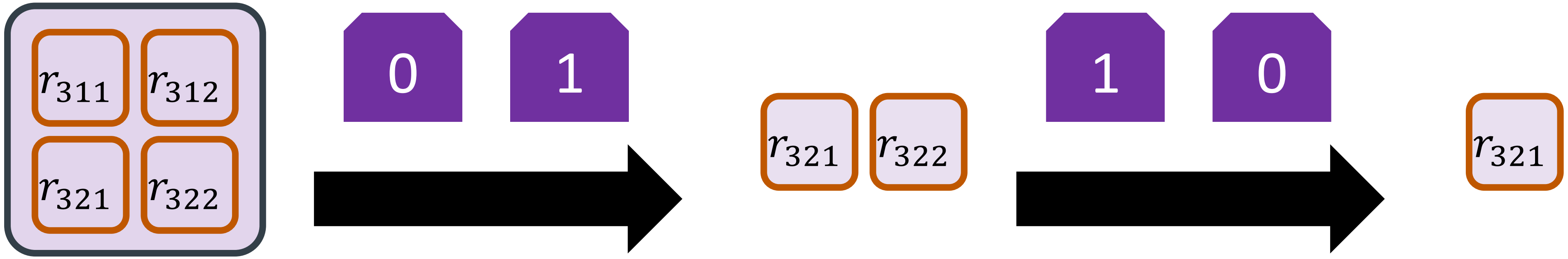
PIR from Homomorphic Encryption

[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)

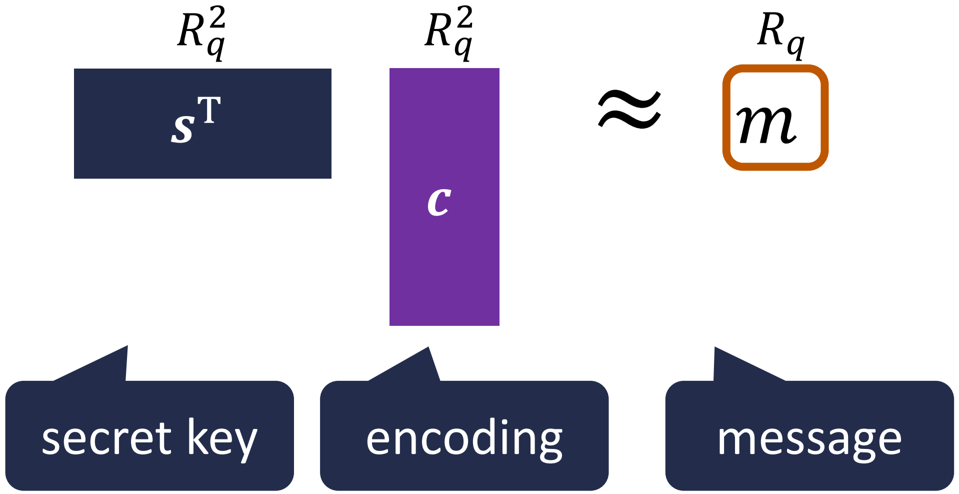
1 bit plaintext	⇒	2.5 **MB** ciphertext

SPIRAL: Use GSW for homomorphic multiplication, Regev for communication

Regev 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

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

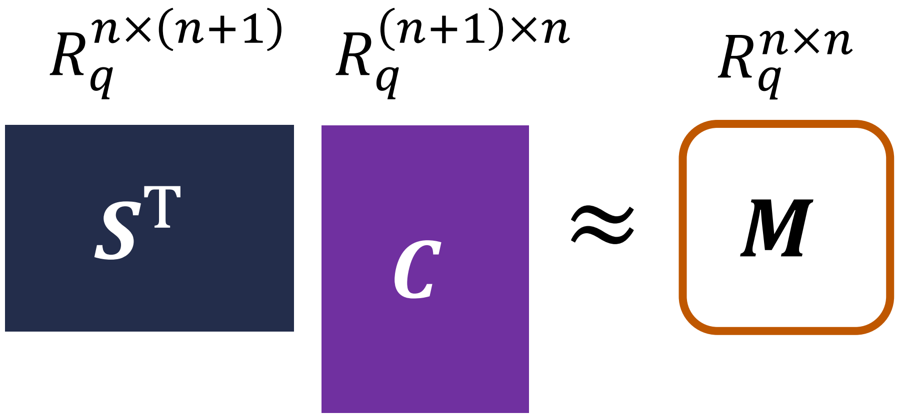
OnionPIR: rate = 0.24

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:

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

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

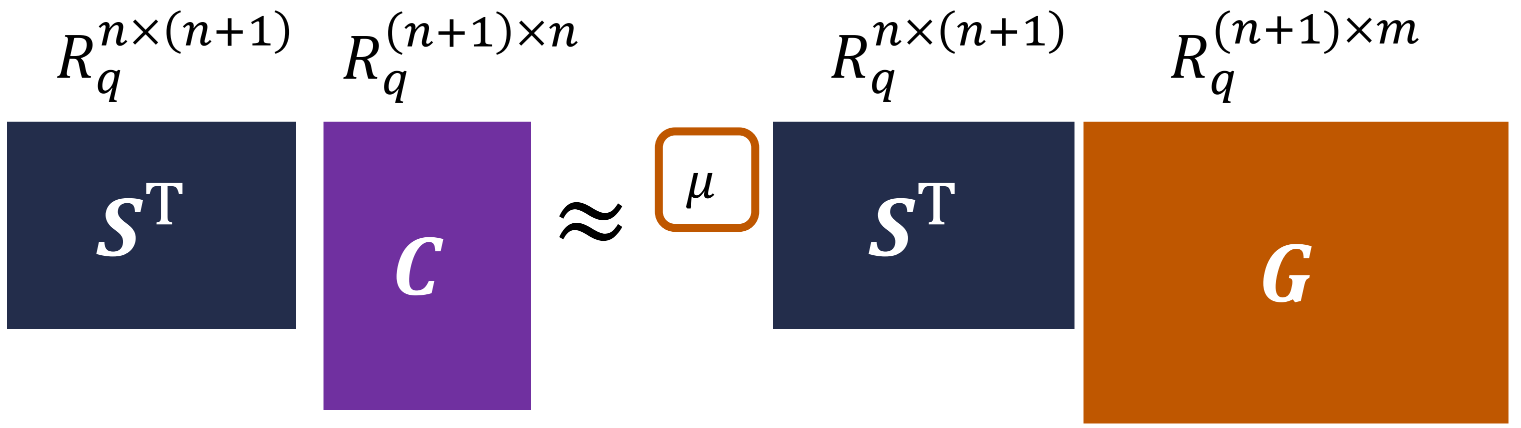
$$\boldsymbol{S}^{\mathrm{T}}(\boldsymbol{C}_1 + \boldsymbol{C}_2) \approx \boldsymbol{M}_1 + \boldsymbol{M}_2$$

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 $\mu \in \{0,1\}$:



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

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

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 $\mu \in \{0,1\}$:

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

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

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

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

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

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

Regev-GSW Homomorphism

[CGGI18]

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

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

$$\Downarrow$$

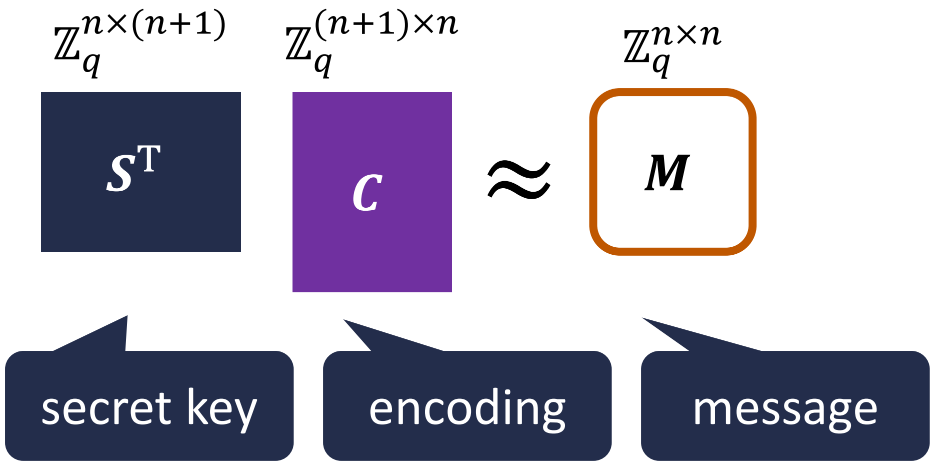
$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_{\mathrm{GSW}}\boldsymbol{G}^{-1}\left(\boldsymbol{C}_{\mathrm{Reg}}\right) \approx \mu\boldsymbol{S}^{\mathrm{T}}\boldsymbol{C}_{\mathrm{Reg}} \approx \mu\boldsymbol{M}$$

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

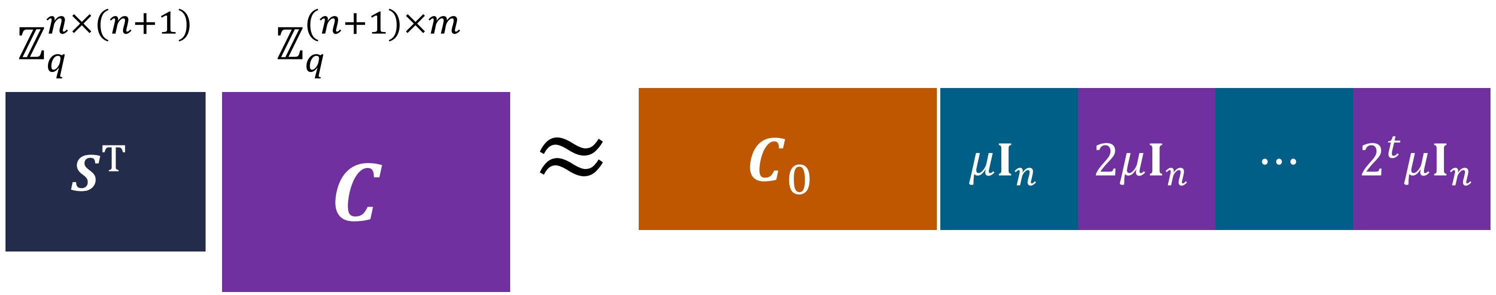
Regev-GSW Homomorphism

[CGGI18]

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



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



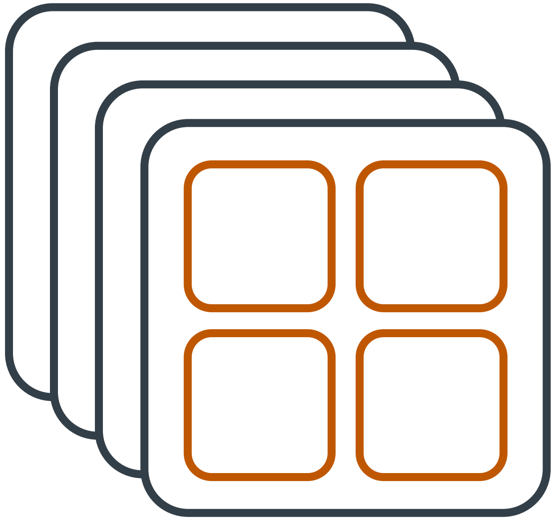
Redundant encoding of μ

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

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

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

0	I_n	0	0	0	0

Indicator for index along first dimension

Query contains ν_2 GSW ciphertexts

0	1	1	0

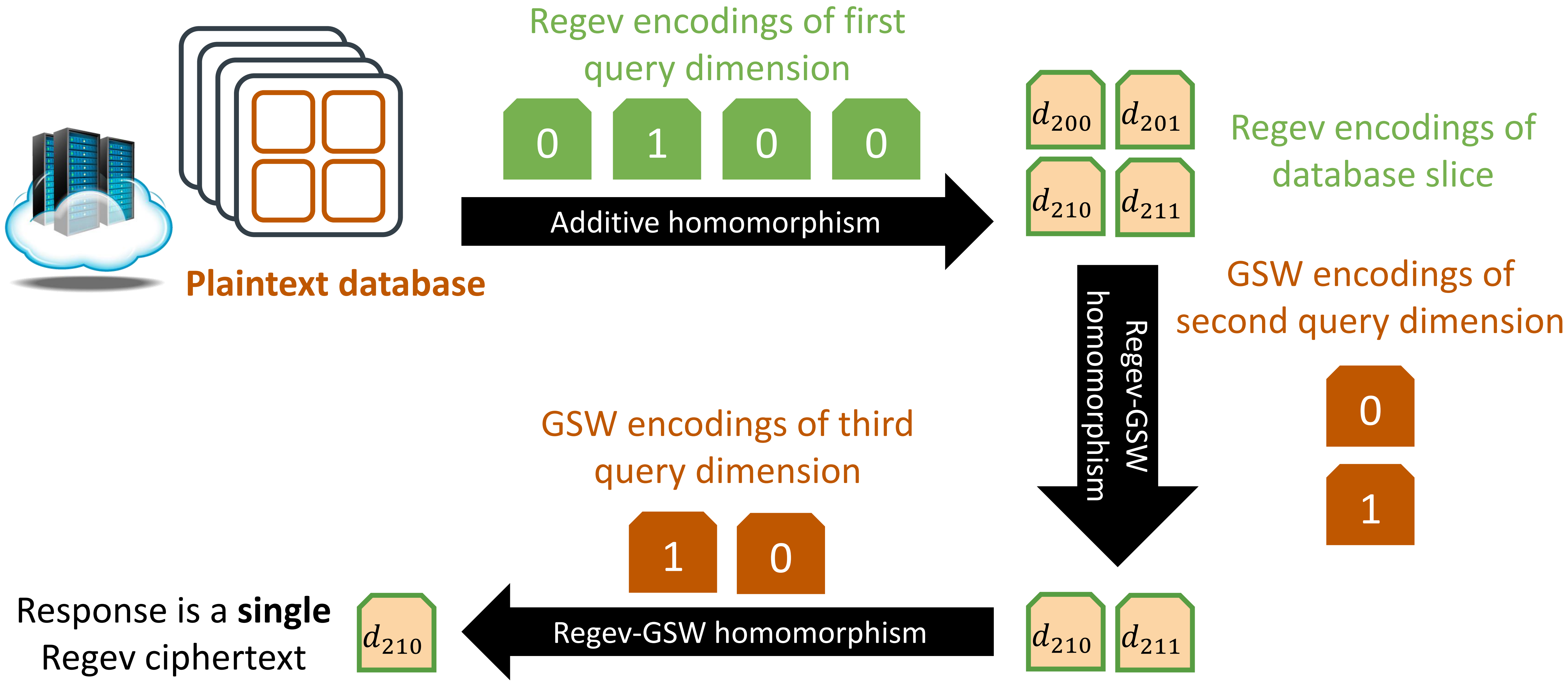
Indicator for index along subsequent dimensions

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

Response is a single matrix Regev ciphertext

The Gentry-Halevi Blueprint

[GH19]



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 $\quad$ I_n $\quad$ 0 $\quad$ 0 $\quad$ 0 $\quad$ 0

Indicator for index along first dimension

Estimated size:
4 MB/ciphertext

Estimated query size:
30 MB

Query contains ν_2 GSW ciphertexts

0 $\quad$ 1 $\quad$ 1 $\quad$ 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

The SPIRAL Protocol

Key idea: Expand Regev encodings into GSW encodings

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

SPIRAL: General toolkit to translate between Regev and GSW

Transformations useful for query compression and response packing

Assembling GSW Encodings

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

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

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

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

Assembling GSW Encodings

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

$$\mu\boldsymbol{S}^{\mathrm{T}}\boldsymbol{G} = \left[\; \boldsymbol{C}_0 \;\middle|\; \mu\mathbf{I}_n \;\middle|\; 2\mu\mathbf{I}_n \;\middle|\; 2^2\mu\mathbf{I}_n \;\middle|\; \cdots \;\middle|\; 2^t\mu\mathbf{I}_n \;\right]$$

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

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

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

Leverage “key-switching”

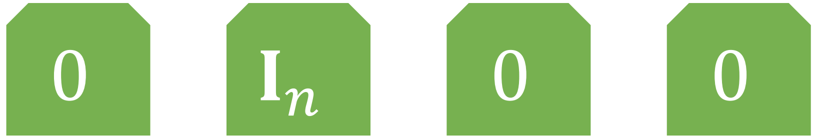
Standard Regev encodings of $\mu, 2\mu, \dots, 2^t\mu$

[see paper for details]

Query Compression in Spiral

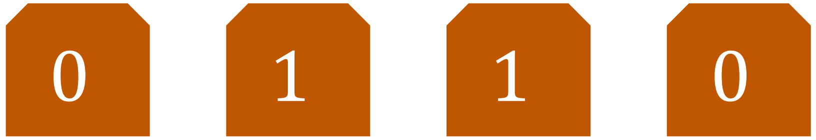
Database is represented as $2^{\nu_1} \times \underbrace{2 \times 2 \times \cdots \times 2}_{2^{\nu_2}}$ hypercube

Query contains 2^{ν_1} matrix Regev encodings

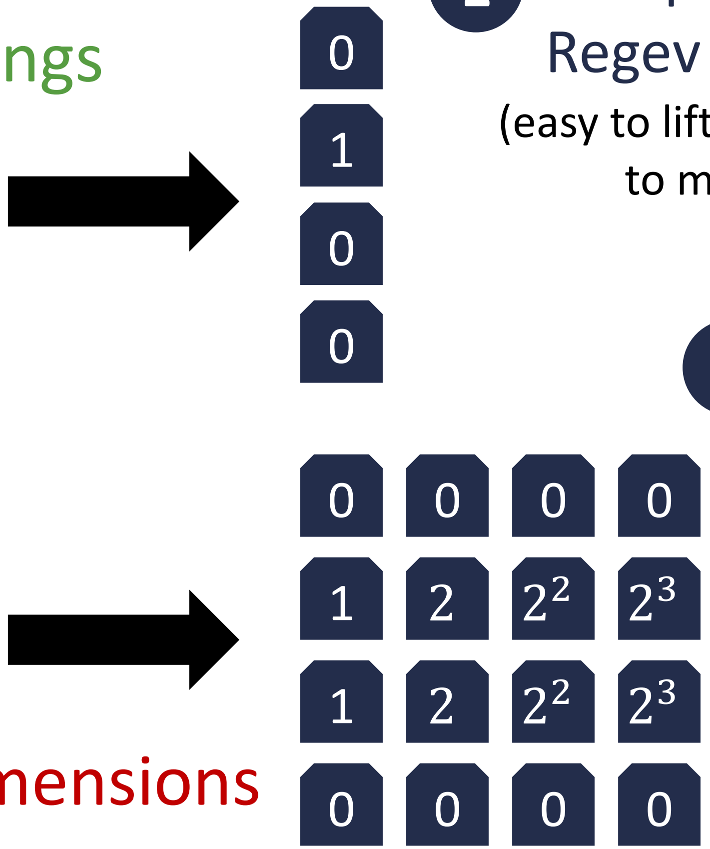


Indicator for index along first dimension

Query contains ν_2 GSW encodings



Indicator for index along subsequent dimensions

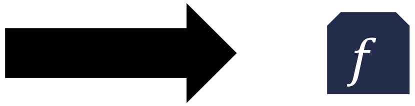


1 Compress into scalar Regev encodings

(easy to lift from scalar encoding to matrix encoding)

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

[ACLS18, CCR19]

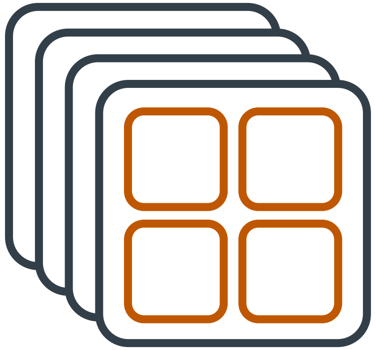


Similar techniques possible for *response* compression [see paper]

The SPIRAL Protocol

public parameters

Key-switching matrices for ciphertext expansion and translation

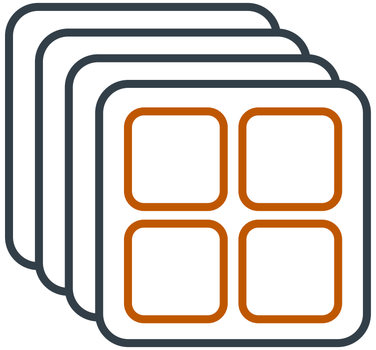


The SPIRAL Protocol

public parameters

query

f Single scalar Regev encoding

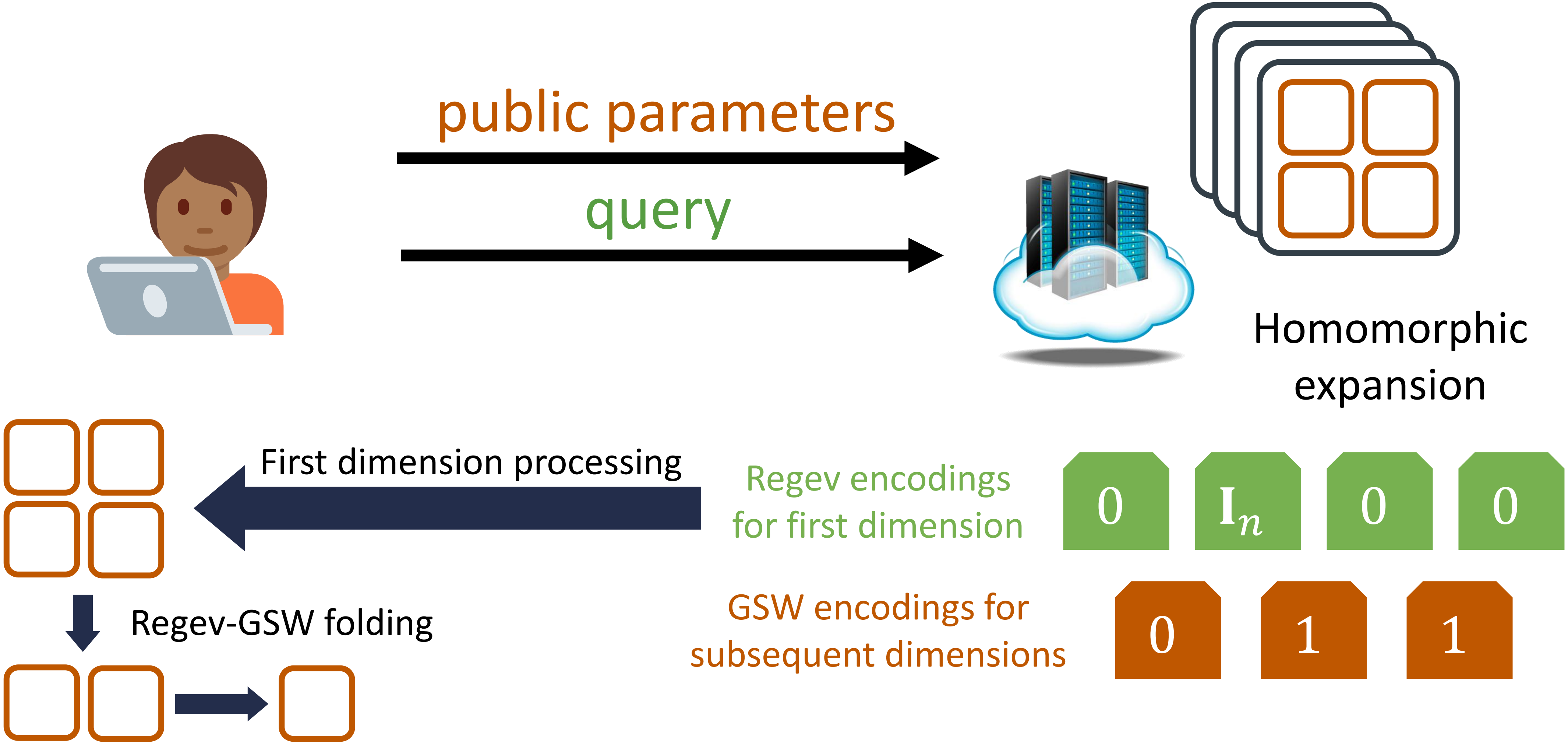


Homomorphic expansion

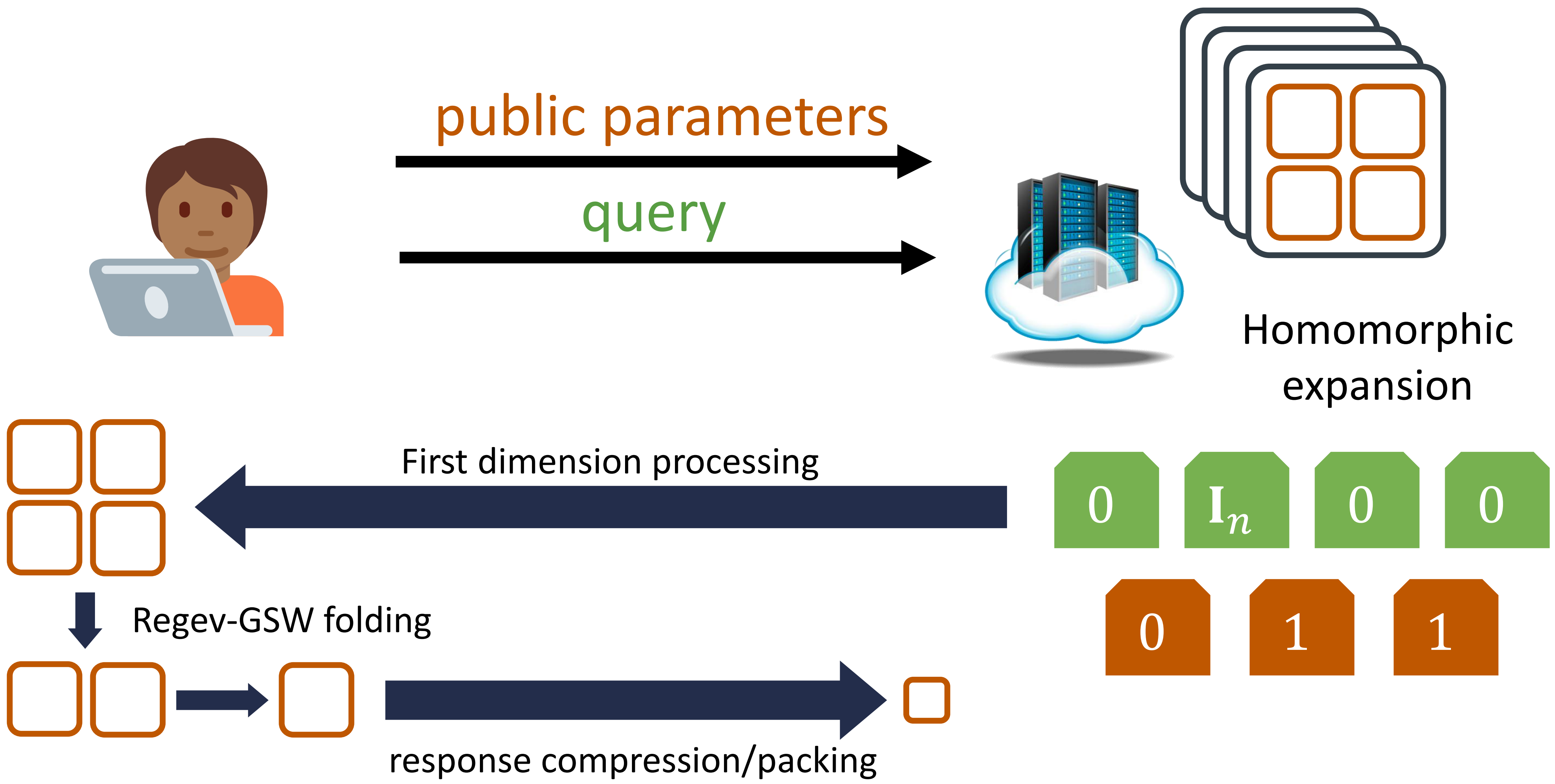
0 I_n 0 0

0 1 1

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:

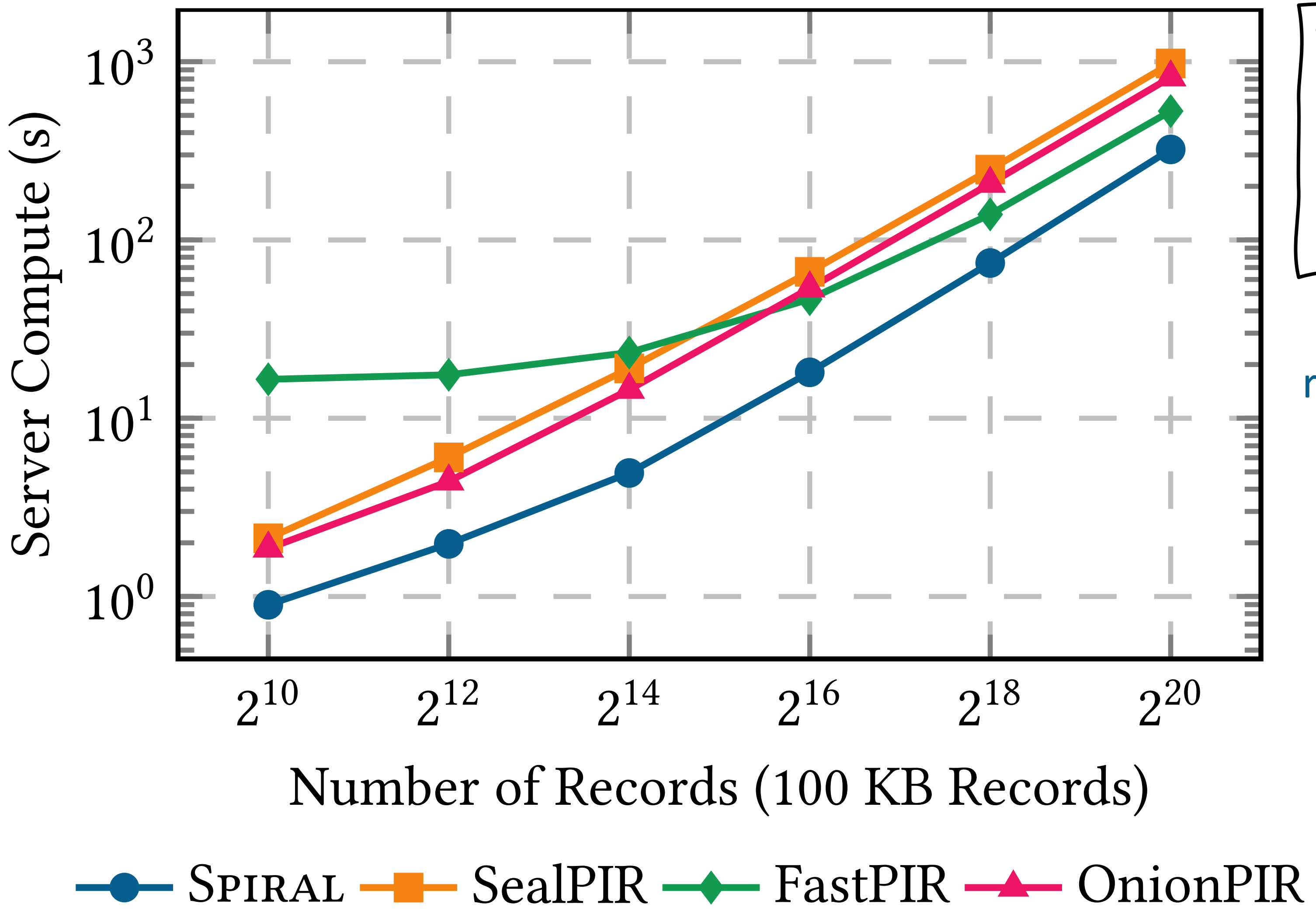
reduce query size by 4.5×
reduce response size by 2×
reduce compute time by 2×

increase public parameter size by 3.6×

Comparisons against other communication-efficient schemes (i.e., ones that do not have 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

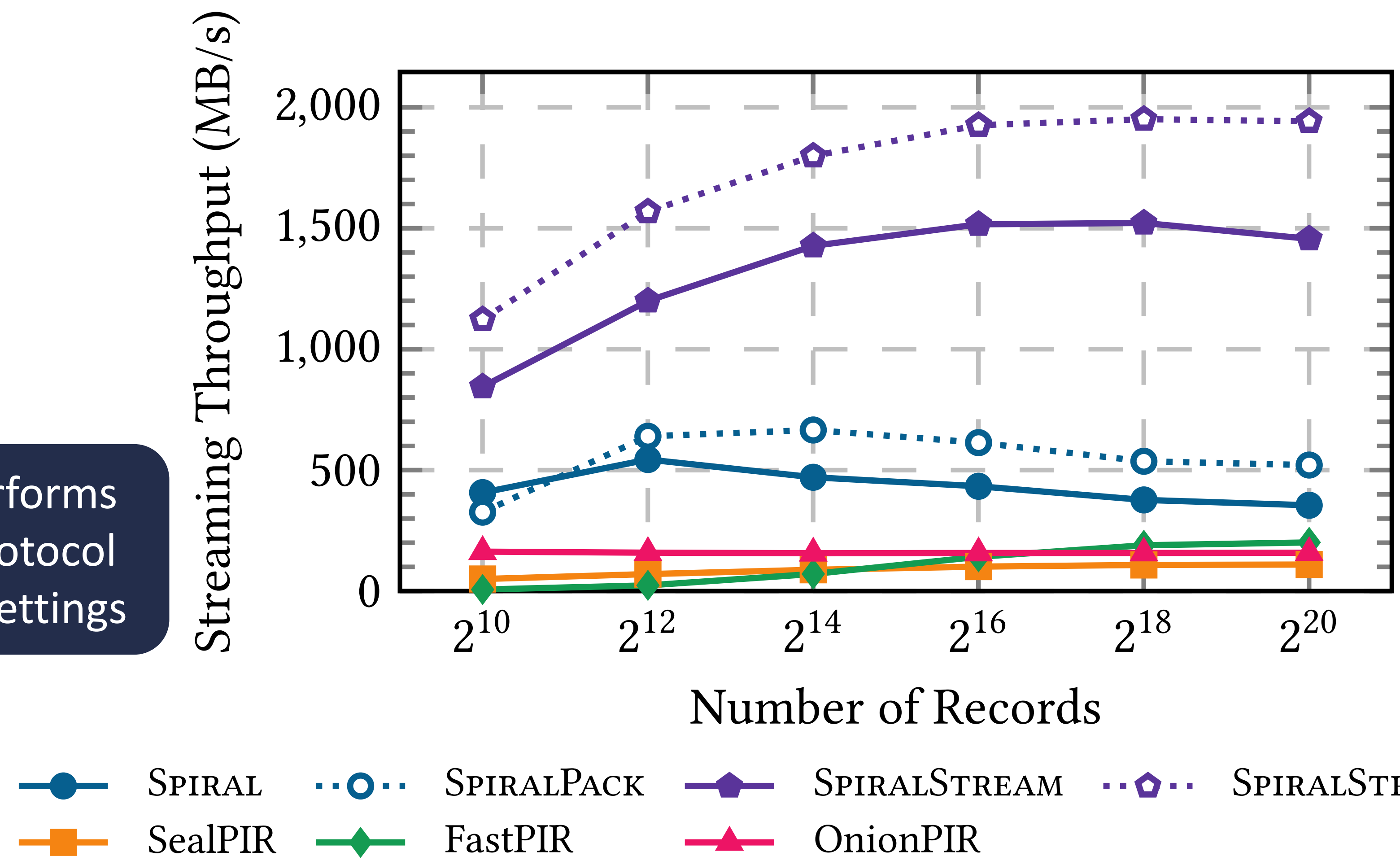
- Private video stream (database D_i contains i^{th} block of media) [GCMSAW16]
- Private voice calls (repeated polling of the same "mailbox") [AS16, AYAAG21]

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

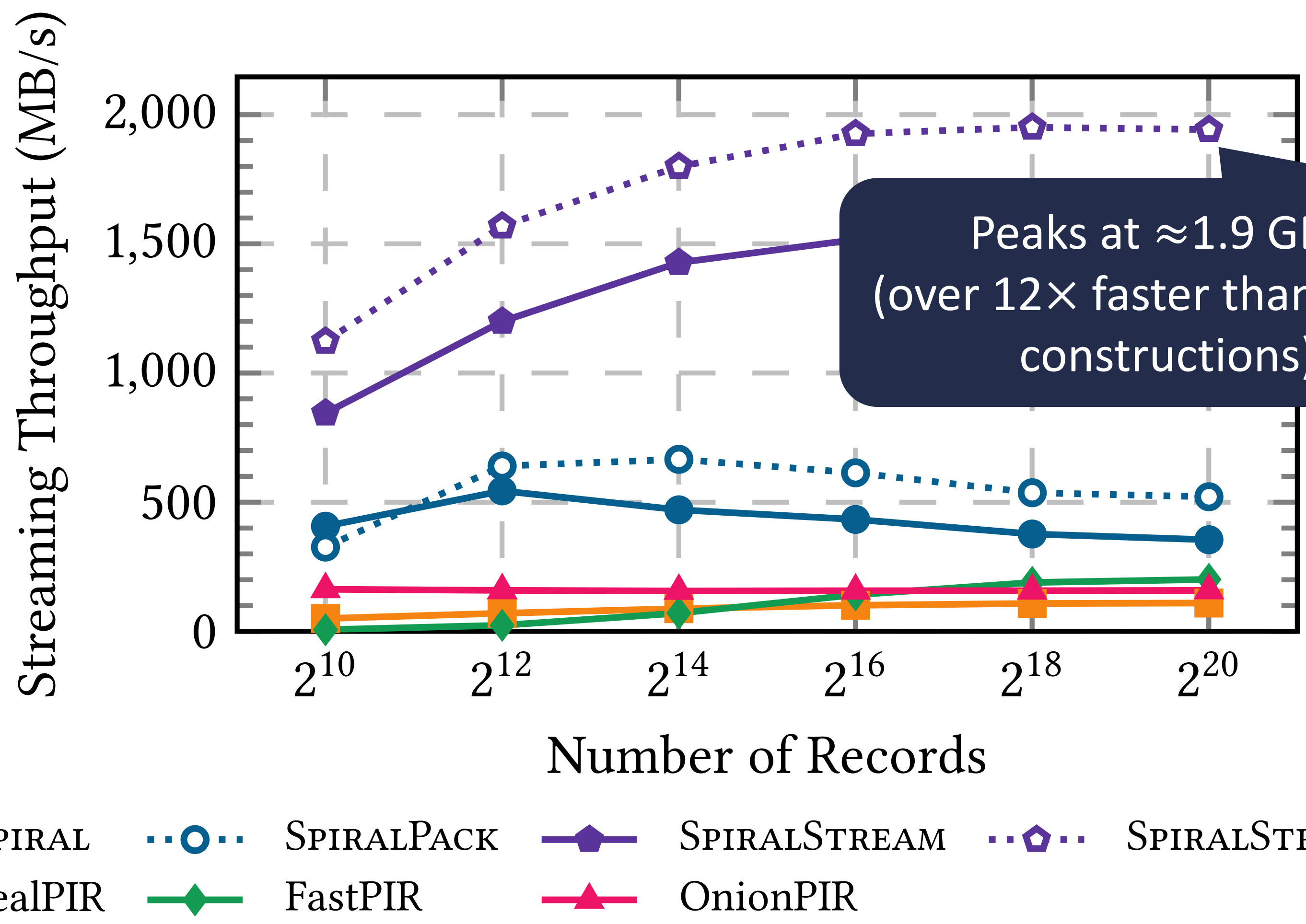


Packing outperforms non-packed protocol for streaming settings

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



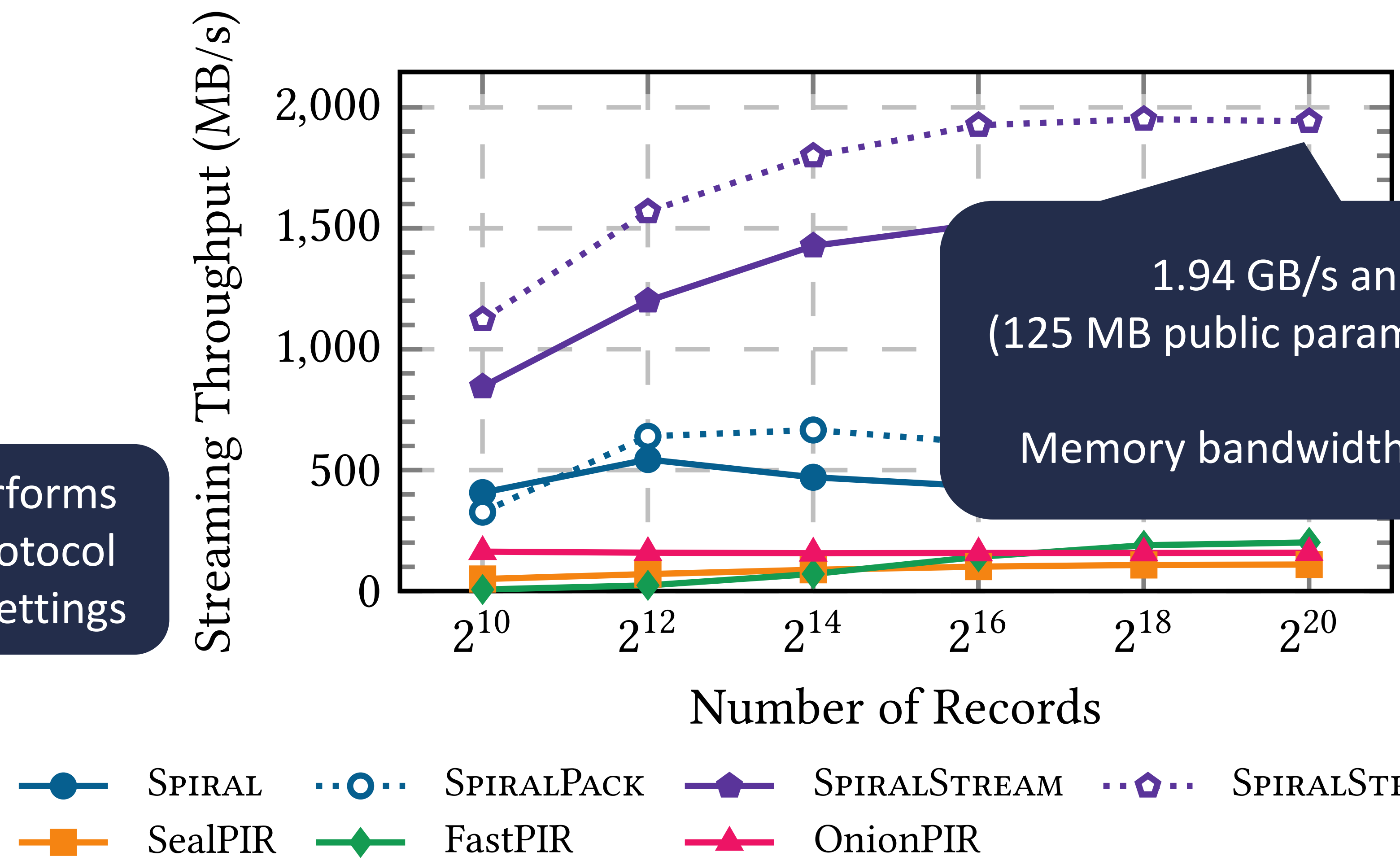
Peaks at ≈1.9 GB/s (over 12× faster than earlier constructions)

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



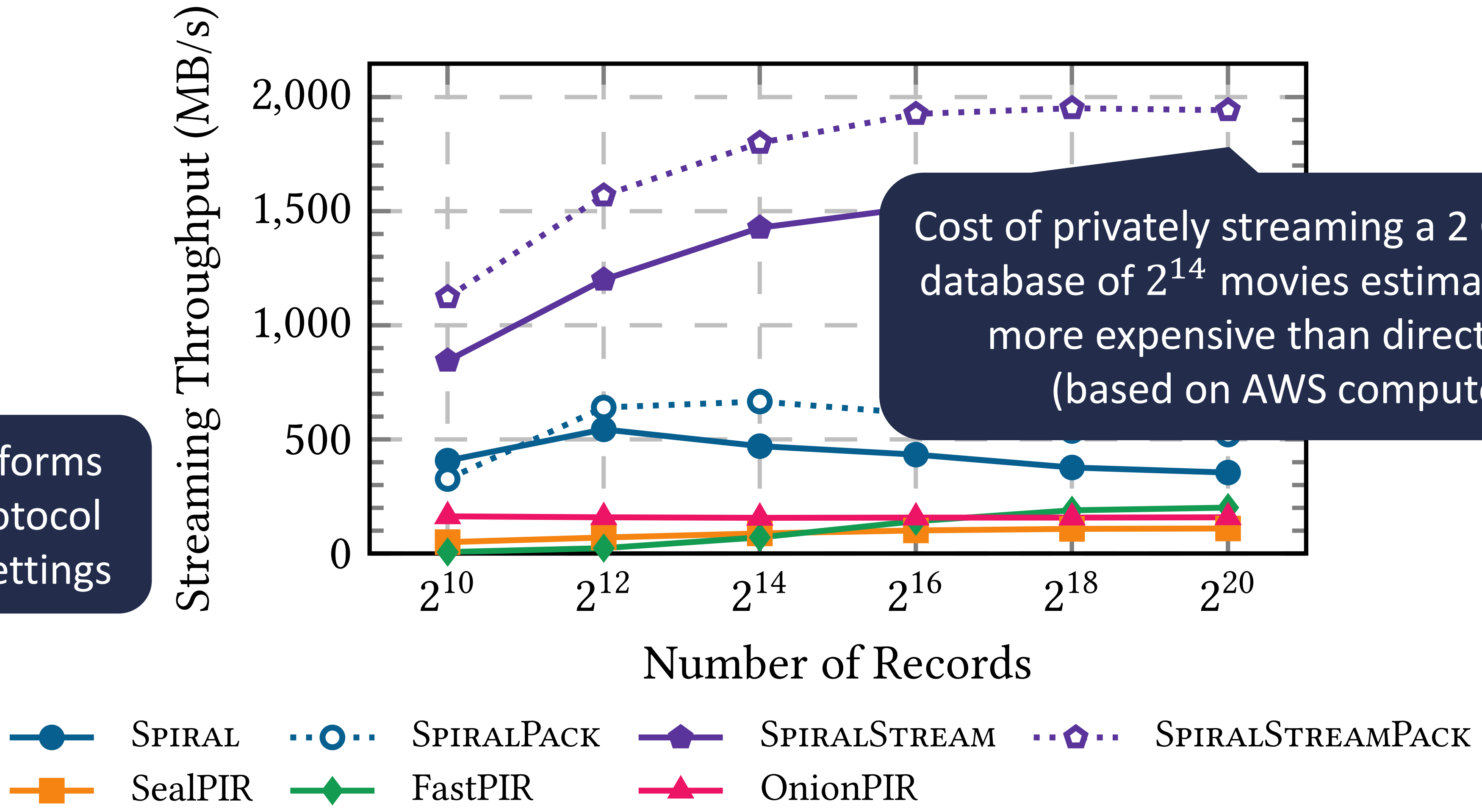
1.94 GB/s and a rate of 0.81
(125 MB public parameter and 30 MB query)

Memory bandwidth on system: ≈10 GB/s

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

RESPIRE: The Small Record Setting

Suppose database has **small** records (~ 256 bytes)

Query size:	16 KB
Response size:	20 KB
Throughput:	200 MB/s

Both queries and responses are much larger than the record!

Reason: LWE ciphertexts are **big**

Recall that query consists of (packed) Regev ciphertext (at least one element of R_q)

- $R = \mathbb{Z}[x]/(x^d + 1)$
- For correctness + security, need $d \sim 2048$ and $q \sim 2^{56}$
- Single ciphertext already ≥ 14 KB

Can we reduce communication when records are small?

RESPIRE: The Small Record Setting

Suppose database has **small** records ($\sim$ 256 bytes)

Query size:	16 KB
Response size:	20 KB
Throughput:	200 MB/s

RESPIRE

Query size:	4.1 KB	3.9× smaller
Response size:	2.0 KB	10× smaller
Throughput:	204 MB/s	

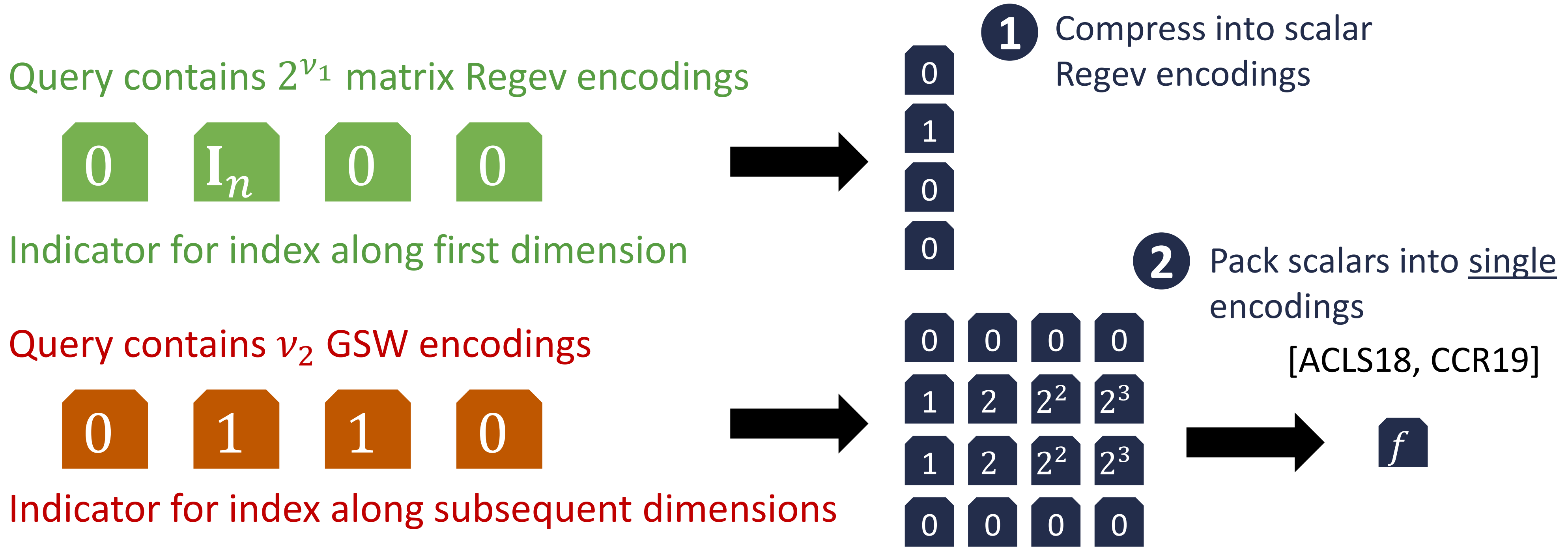
Reason: LWE ciphertexts are **big**

Recall that query consists of (packed) Regev ciphertext (at least one element of R_q)

- $R = \mathbb{Z}[x]/(x^d + 1)$
- For correctness + security, need $d \sim 2048$ and $q \sim 2^{56}$
- Single ciphertext already ≥ 14 KB

Can we reduce communication when records are small?

Query Expansion, Revisited



When database is small, we only need to pack a **small** number of coefficients into an encoding

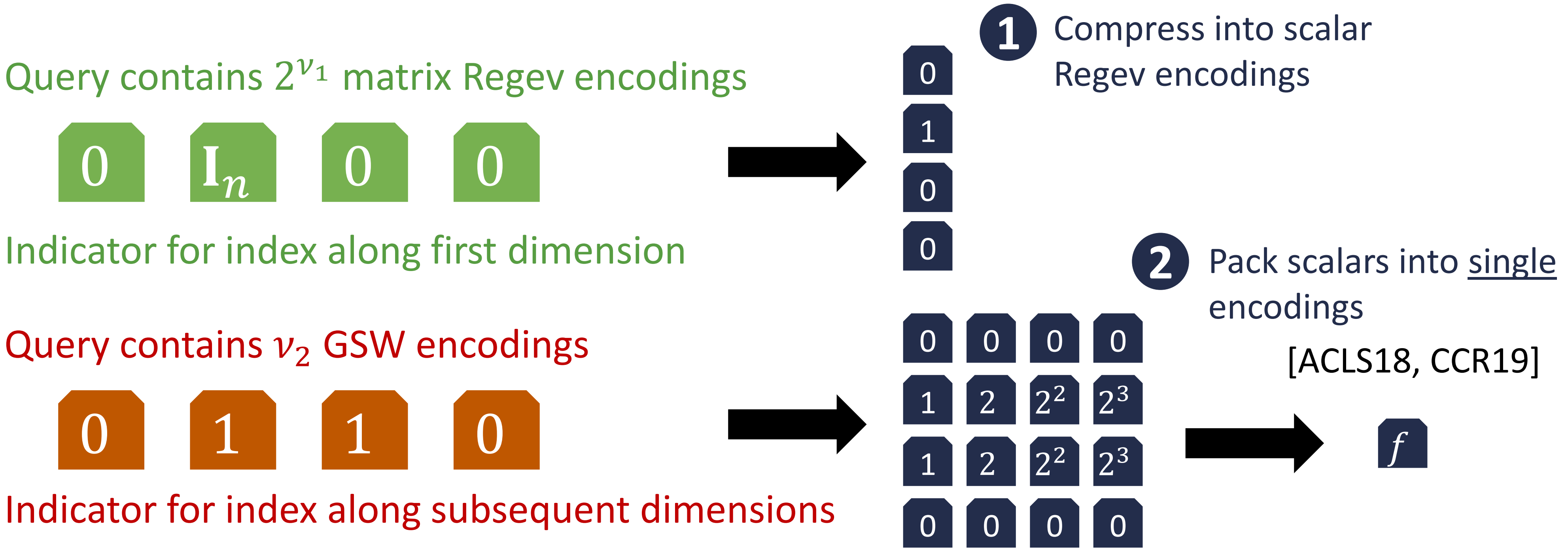
Each plaintext value is a polynomial of degree d and can hold d values in $\mathbb{Z}_q$

$1 + x + x^3$

1	1	0	1	0	0	0	0

$d = 8$

Query Expansion, Revisited



When database is small, we only need to pack a **small** number of coefficients into an encoding

Each plaintext value is a polynomial of degree d and can hold d values in $\mathbb{Z}_q$

$1 + x + x^3$

1	1	0	1	0	0	0	0

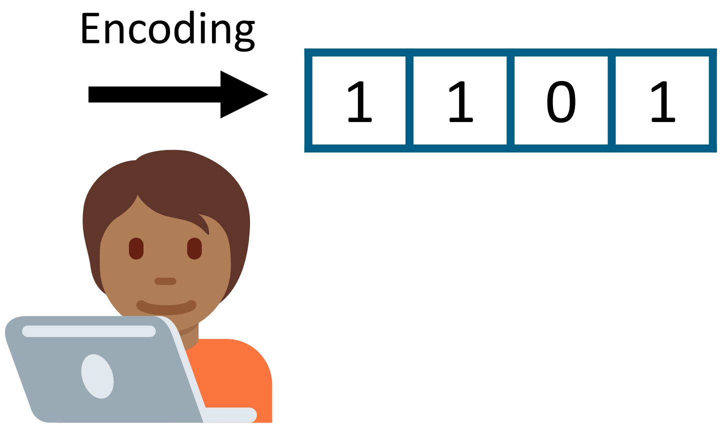
$d = 8$ **Unused space!**

Respire Query Compression

Let d be the ring dimension

If we want to encode (i.e., pack h independent values into a single ciphertext), it suffices to communicate a vector of dimension h rather than d

$$h = 4$$
$$d = 8$$

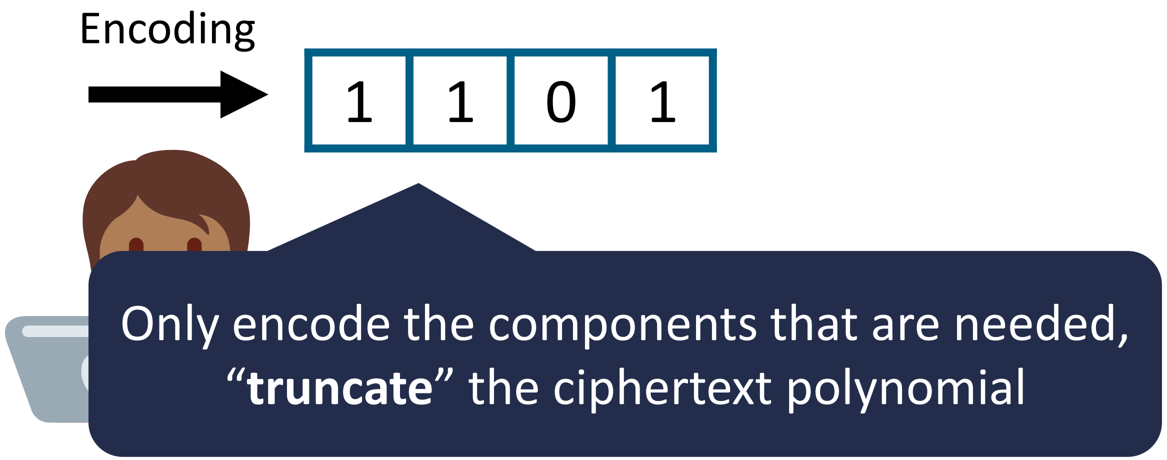


RESPIRE Query Compression

Let d be the ring dimension

If we want to encode (i.e., pack h independent values into a single ciphertext), it suffices to communicate a vector of dimension h rather than d

$$h = 4$$
$$d = 8$$

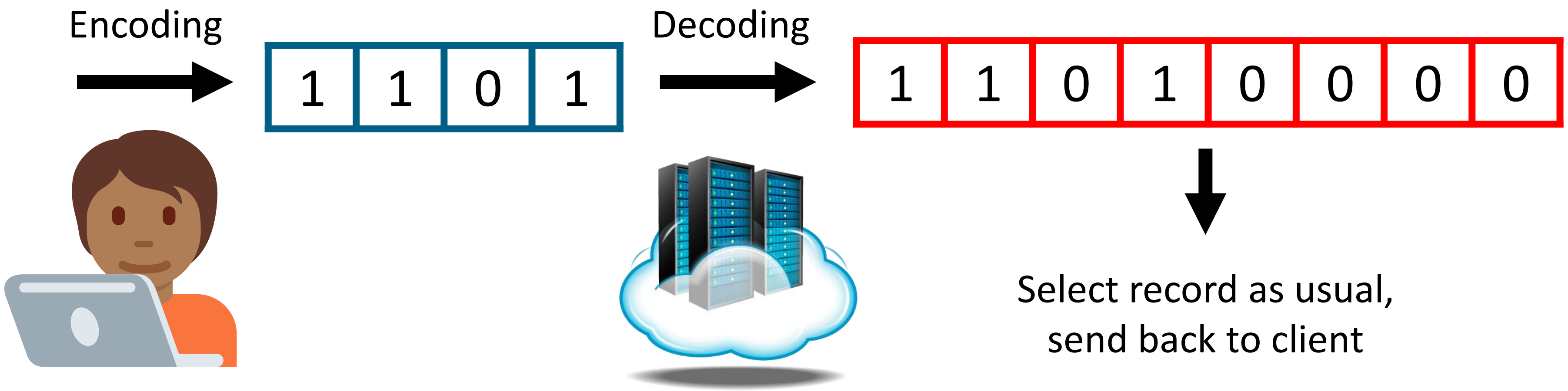


RESPIRE Query Compression

Let d be the ring dimension

If we want to encode (i.e., pack h independent values into a single ciphertext), it suffices to communicate a vector of dimension h rather than d

$$h = 4$$
$$d = 8$$

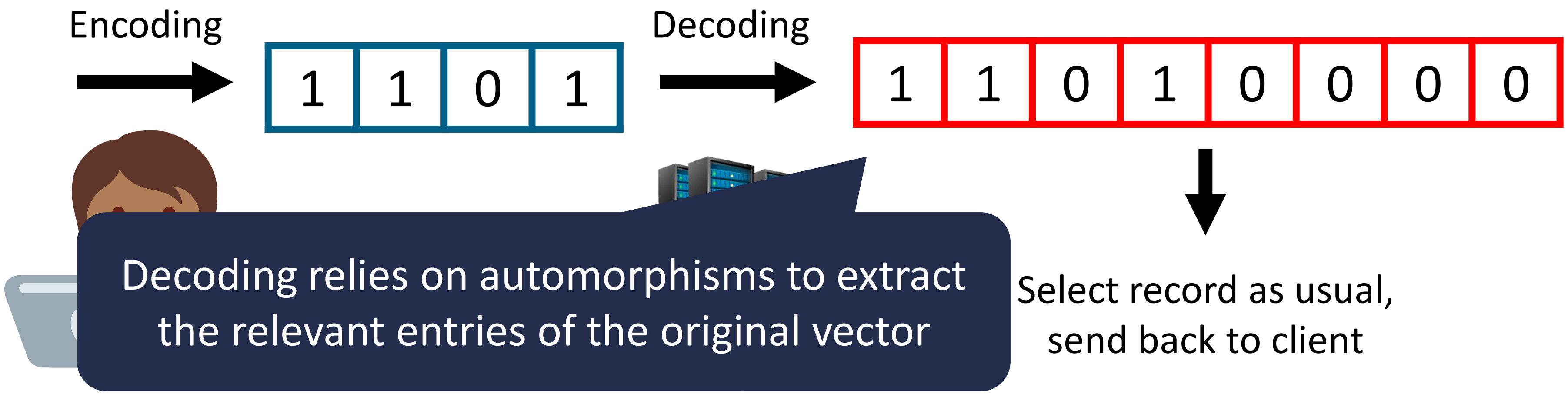


RESPIRE Query Compression

Let d be the ring dimension

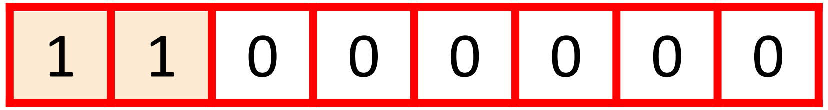
If we want to encode (i.e., pack h independent values into a single ciphertext), it suffices to communicate a vector of dimension h rather than d

$$h = 4$$
$$d = 8$$

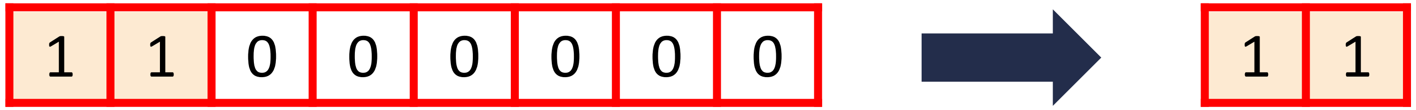


RESPIRE Response Compression

Let d be the ring dimension



Suppose record is *much* smaller than a single ring element



"Ring switching" [BGV12, GHPS12]: translate ciphertext over big ring to a ciphertext over a *subring*

RESPIRE

Query size:	4.1 KB
Response size:	2.0 KB
Throughput:	204 MB/s

(1 million 256 byte records)

Both query and response is "smaller" than standard RLWE ciphertext!

More Recent Developments in PIR

Server preprocessing (client downloads hint at beginning of protocol)

- SimplePIR, DoublePIR [HHCMV23]
  - Very high throughput (nearly memory bandwidth!)
  - Suitable for databases with small records (a few bits), but has a large hint (hundred of MB)
- HintlessPIR [LMRS24], YPIR [MW24]
  - SimplePIR without the hint (by leveraging bootstrapping/key-switching)
  - Comparable throughput (for big databases), slightly more communication
- Piano [ZPSZ23]
  - **Sublinear** server computational costs (can scale better to databases that are >100 GB)
  - Preprocessing phase requires *streaming* the entire database

More Recent Developments in PIR

Server preprocessing (*without* hint)

Doubly-efficient PIR [LMW23]

Server encodes the database to answer queries in sublinear time

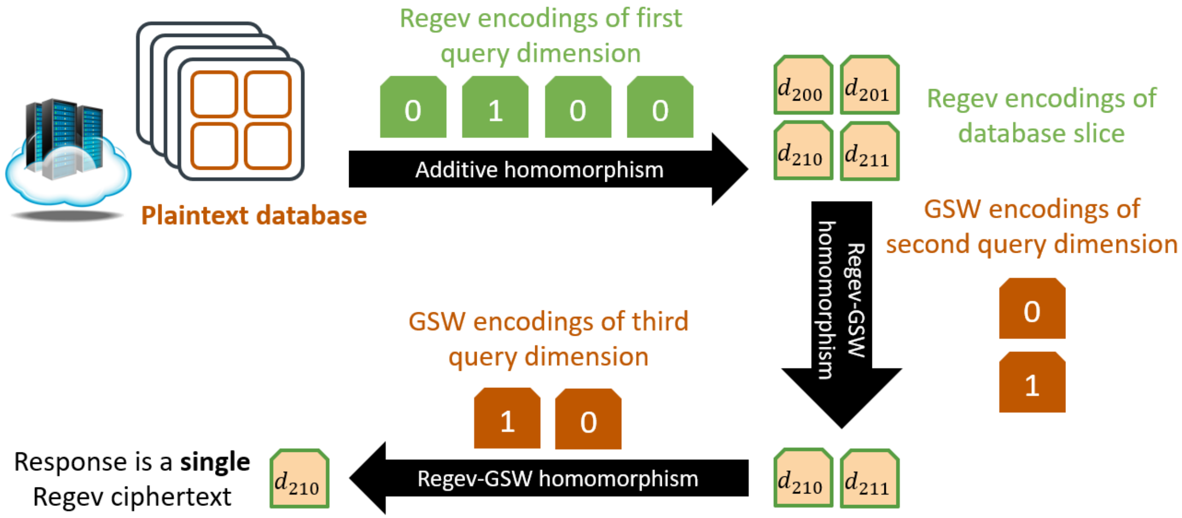
Concrete efficiency not yet clear

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!

SPIRAL and RESPIRE



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

Useful for both query compression and response compression

SPIRAL: https://eprint.iacr.org/2022/368.pdf
RESPIRE: https://eprint.iacr.org/2024/1165.pdf
Code: https://github.com/menonsamir/spiral-rs

Thank you!