Private Information Retrieval: Opportunities and Challenges

David Wu

June 2025

based on joint work with Samir Menon

Private Information Retrieval (PIR)

[CGKS95]

This talk: focus only on **single-server** setting

record i

client

database

$\boldsymbol{r_1}$

$\boldsymbol{r_2}$

$\vdots$

$\boldsymbol{r_N}$

$\boldsymbol{r_i}$

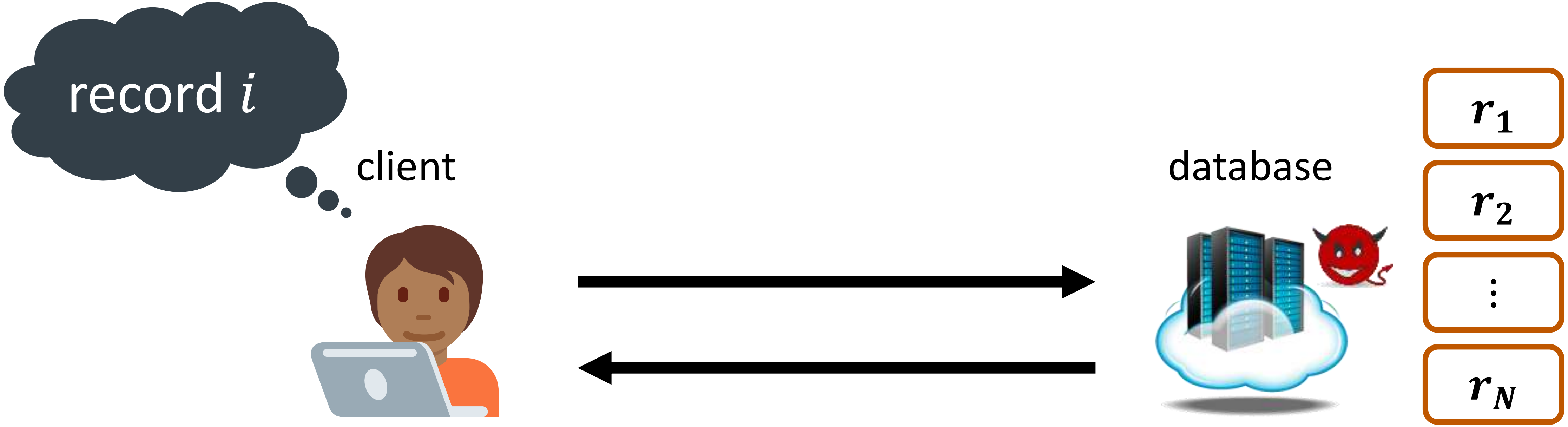
Correctness: Client learns r_i

Privacy: Server does <u>not</u> learn 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

Application to Certificate Transparency

[LLK13, Lau14]

Let's Encrypt

COMODO CYBERSECURITY

digicert

certificate authorities

certificate

SCT

signed certificate timestamp

Google

Meta

CLOUDFLARE

certificate transparency log server

Goal: monitor issuance of certificates and detect rogue certificates

Approach:

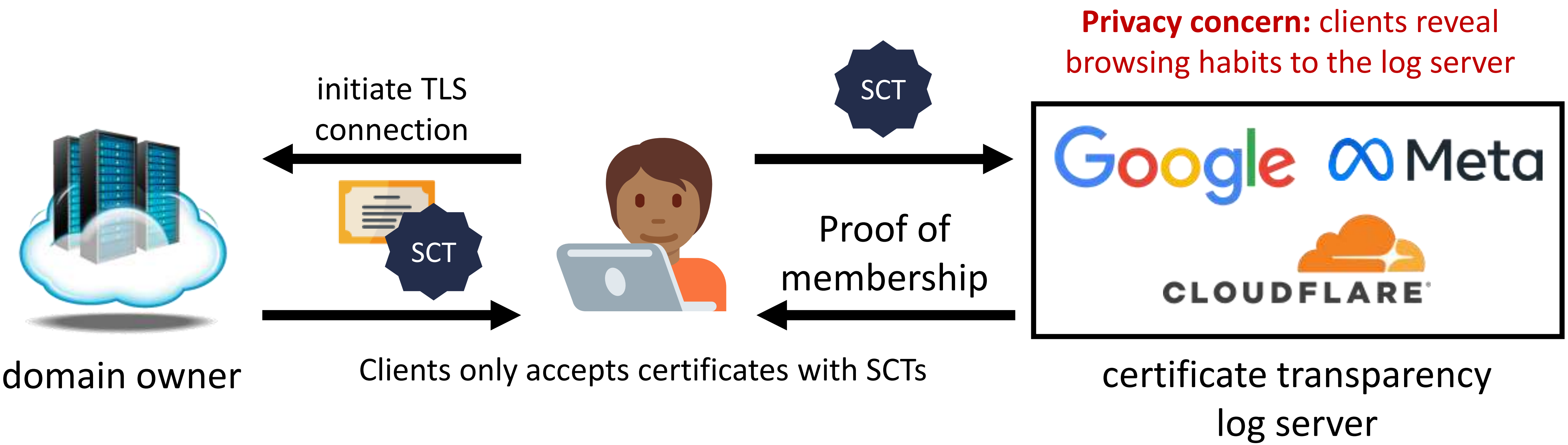
- When certificate authority (CA) issues certificate, it deposits it into a log server and receives a signed certificate timestamp (SCT)
- Servers can check log server to see all certificates issued for their domain name

Application to Certificate Transparency

[LLK13, Lau14]

A valid SCT means that the certificate was deposited into a log server

But is the log server honest? Clients will periodically **audit** log server to check that SCT is actually present



Goal: monitor issuance of certificates and detect rogue certificates

Today: Chrome reveals a few bits of the hash of the SCT (k-anonymity-based solution)

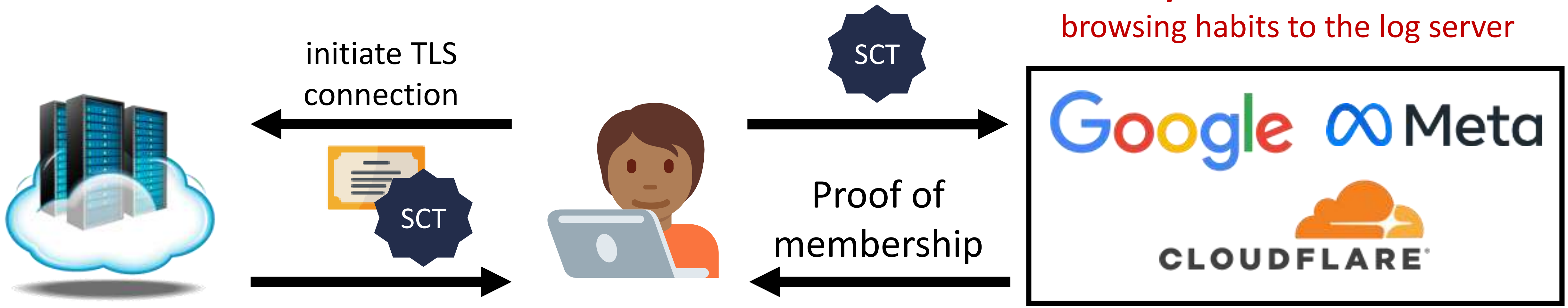
PIR for Certificate Transparency

[LG15, KOR19, HHCMV23]

A valid SCT means that the certificate was deposited into a log server

But is the log server honest? Clients will periodically **audit** log server to check that SCT is actually present

Privacy concern: clients reveal browsing habits to the log server



Proposal: client checks SCT membership using PIR

[HHCMV23]: Server maintains a Bloom filter of the set of SCTs deposited

To audit, client reads **one** bit of the Bloom filter (high false positive rate, but handled by relying on many clients)

How Efficient is PIR?

On the Computational Practicality of Private Information Retrieval

Radu Sion *
Network Security and Applied Cryptography Lab
Computer Sciences, Stony Brook University
sion@cs.stonybrook.edu

Bogdan Carbunar
Pervasive Platforms and Architectures
Motorola Labs
carbunar@motorola.com

Abstract

We explore the limits of single-server computational private information retrieval (PIR) for the purpose of preserving client access patterns leakage. *We show that deployment of non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude less time-efficient than trivially transferring the entire database. We stress that these results are beyond existing knowledge of mere "impracticality" under unfavorable assumptions. They rather reflect an inherent limitation with respect to modern hardware, likely the result of a communication-cost centric protocol design. We argue that this is likely to hold on non-specialized traditional hardware in the foreseeable future. We validate our reasoning in an experimental setup on modern off-the-shelf hardware. Ultimately, we hope our results will stimulate practical designs.*

Here we discuss single-server computational PIR *for the purpose of preserving client access patterns leakage*. We show that deployment of non-trivial single server private information retrieval protocols on real hardware of the recent past would have been orders of magnitude more time-consuming than trivially transferring the entire database. The deployment of computational PIR would in fact *increase* overall execution time, as well as the probability of *forward* leakage, when the deployed present trapdoors become eventually vulnerable – e.g., today's queries will be revealed once factoring of today's values will become possible in the future.

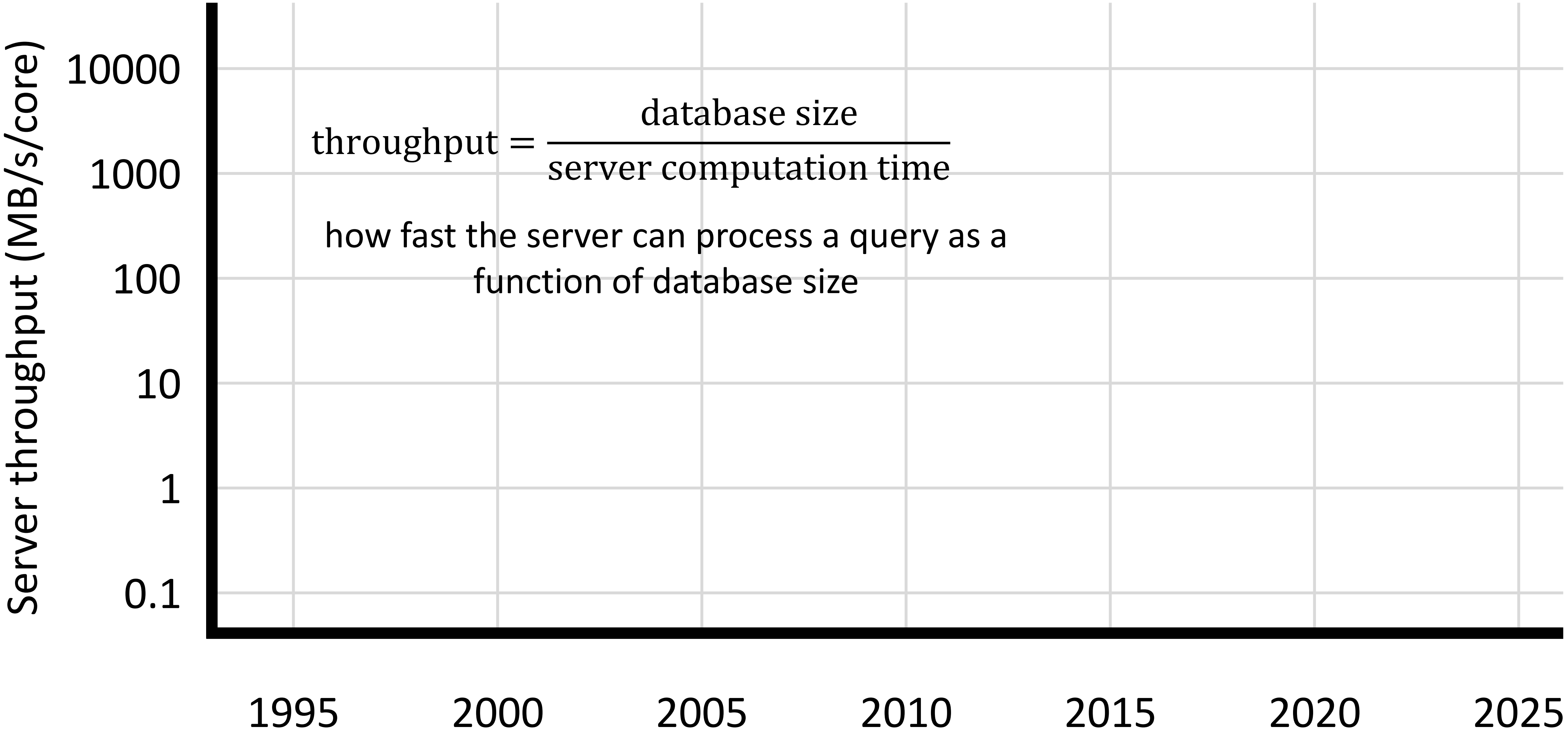
We stress that this is beyond existing knowledge of mere "impracticality" under unfavorable assumptions. On real hardware, *no* existing non-trivial single server PIR protocol could have possibly had outperformed the trivial client-to-server transfer of records in the past, and is likely not to do so in the future either. This is due to the fact that on any

NDSS 2007

Take-away (2007): PIR schemes are too expensive and better to just have client download the database; need **new** constructions

Recurring theme in cryptography: powerful tools, but often (concretely) expensive

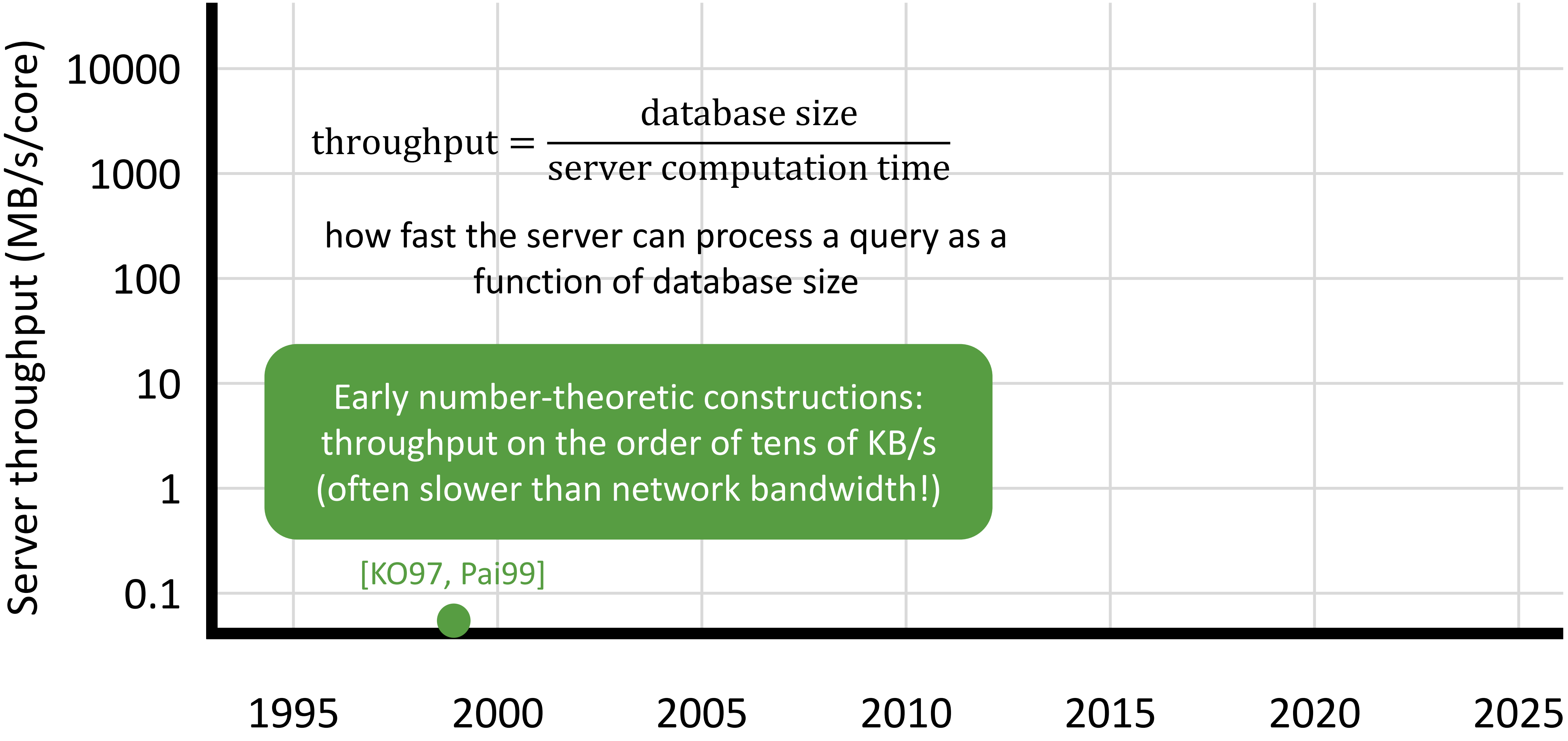
25 Years of PIR Research



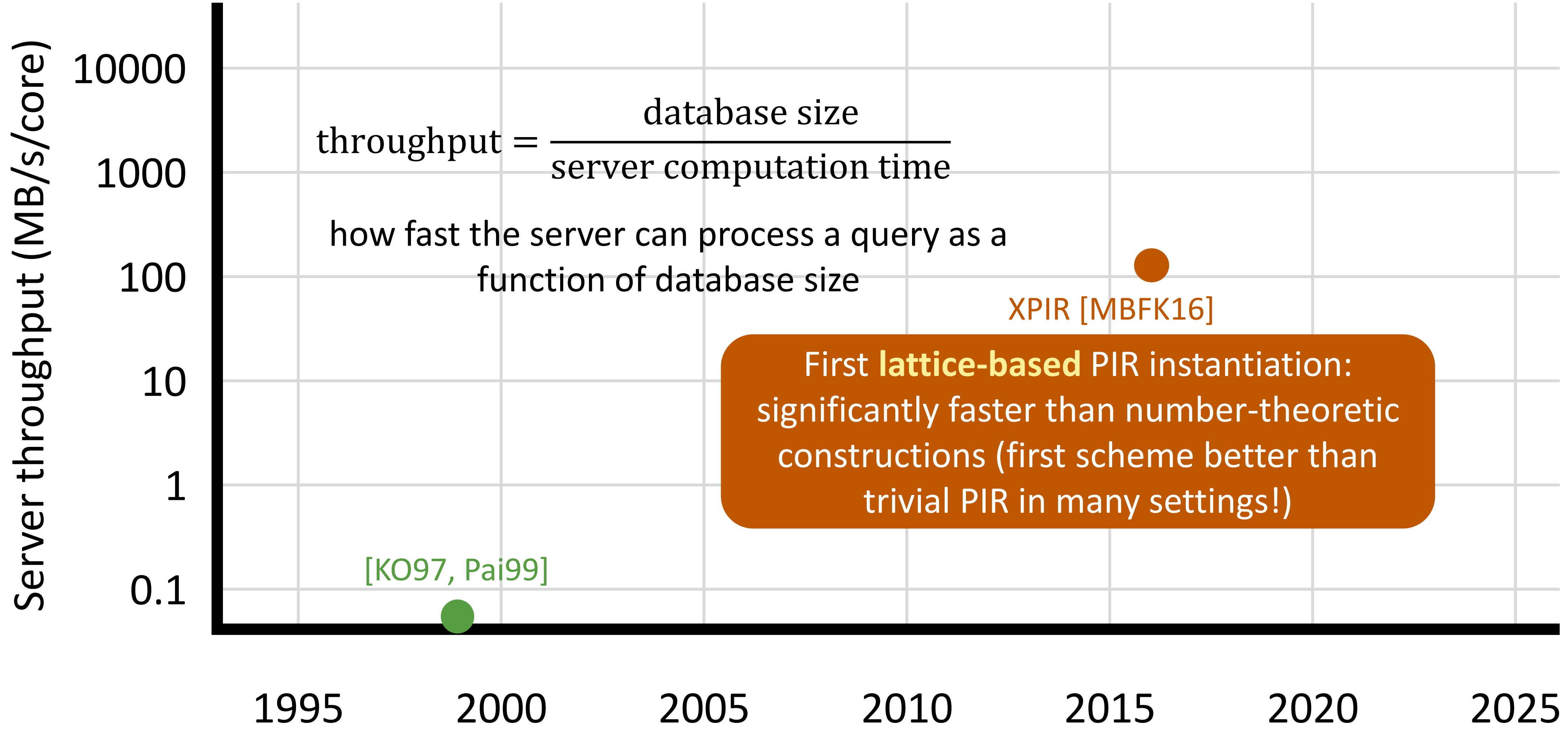
$$\text{throughput} = \frac{\text{database size}}{\text{server computation time}}$$

how fast the server can process a query as a function of database size

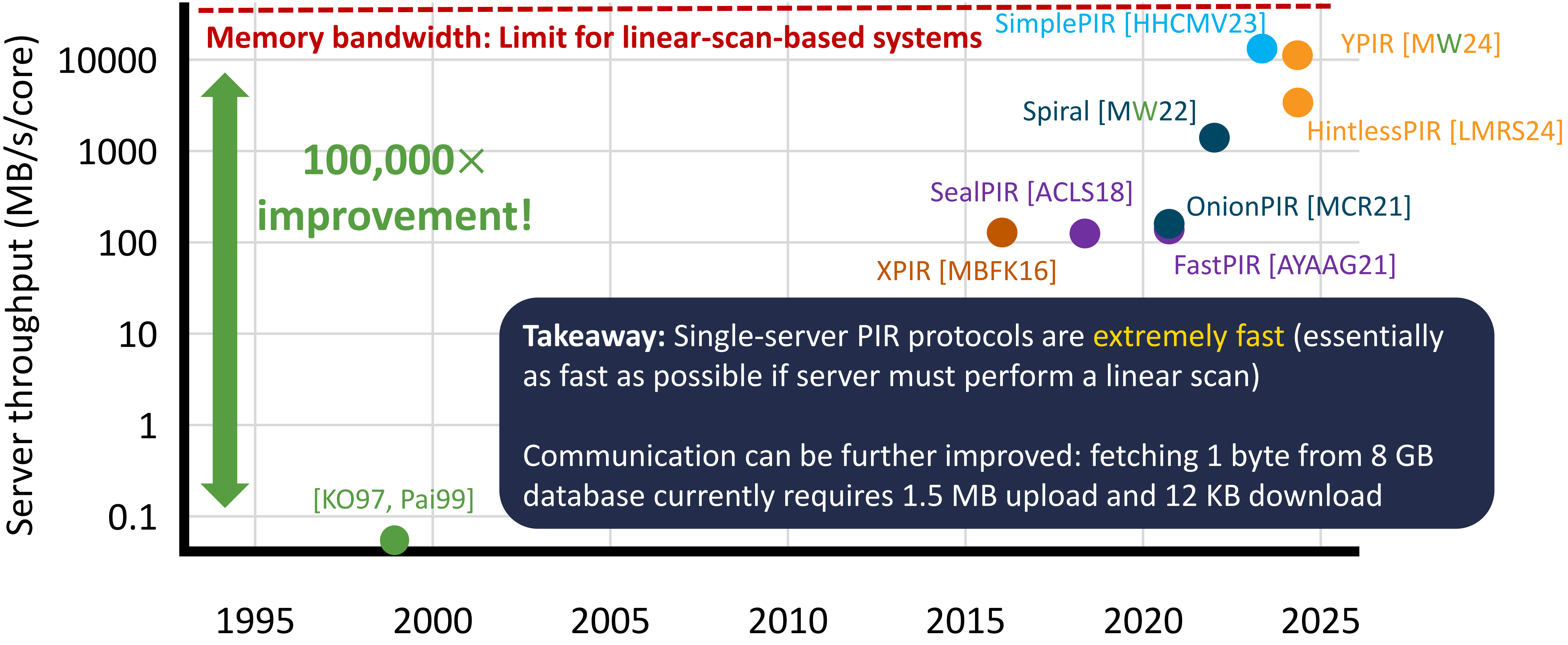
25 Years of PIR Research



25 Years of PIR Research



25 Years of PIR Research



Takeaway: Single-server PIR protocols are extremely fast (essentially as fast as possible if server must perform a linear scan)

Communication can be further improved: fetching 1 byte from 8 GB database currently requires 1.5 MB upload and 12 KB download

PIR from Homomorphic Encryption

[KO97]

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

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

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)

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

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

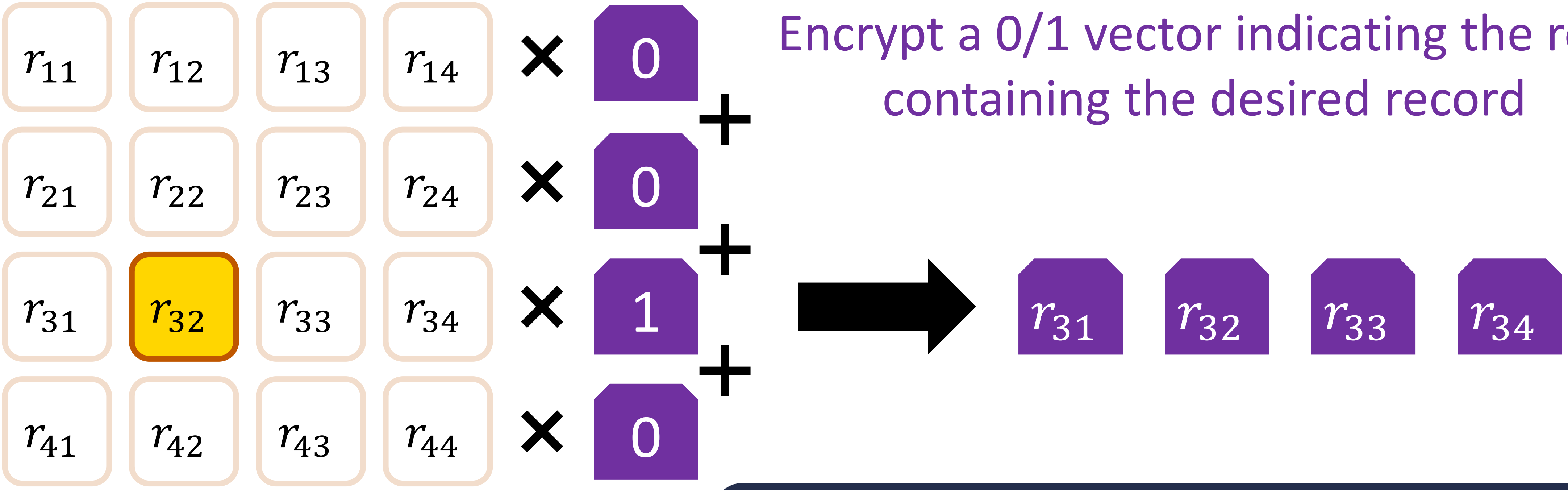
Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

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}

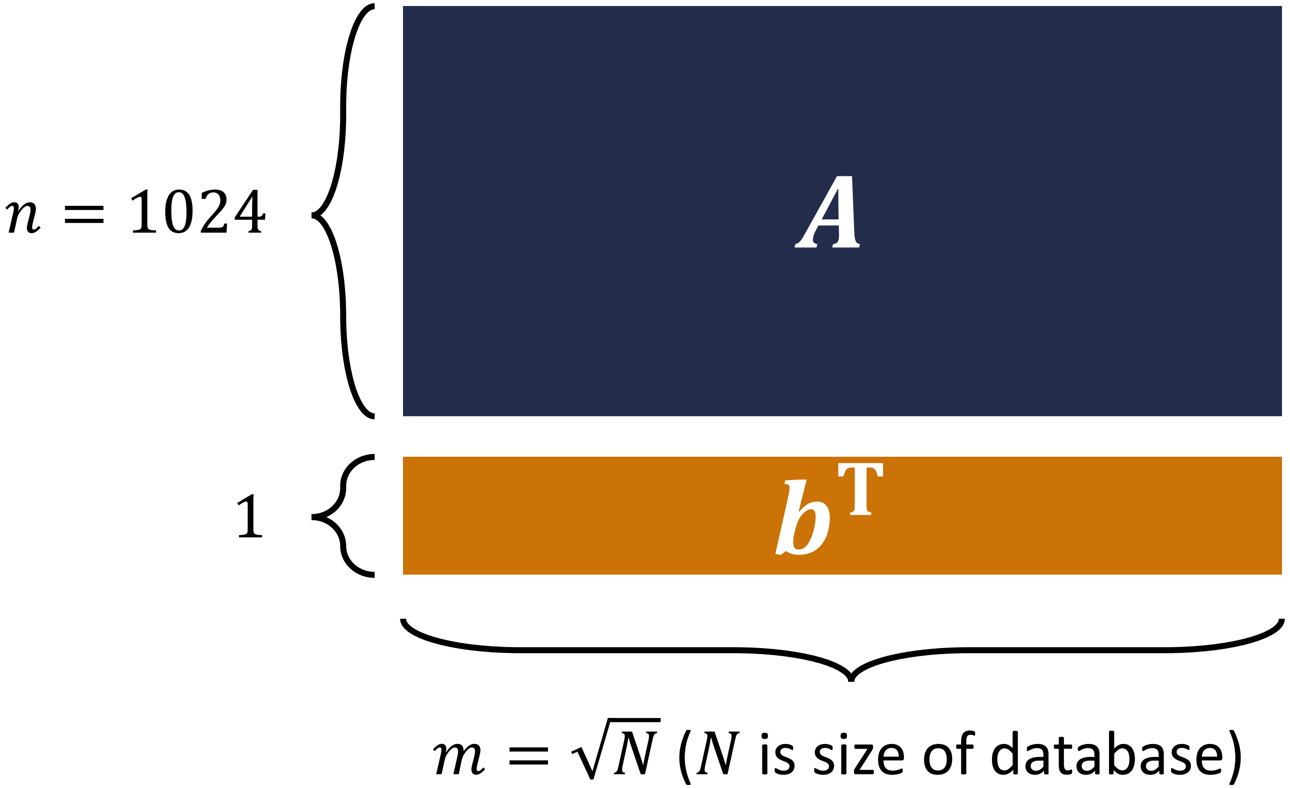
Response size: $O_\lambda(\sqrt{N})$

Homomorphically compute product between query vector and database matrix

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Building block: Regev's linearly homomorphic encryption from LWE [Reg05]



Elements of A are random 32-bit integers

b is a ciphertext (encrypts a $\sqrt{N}$-dimension bit vector)

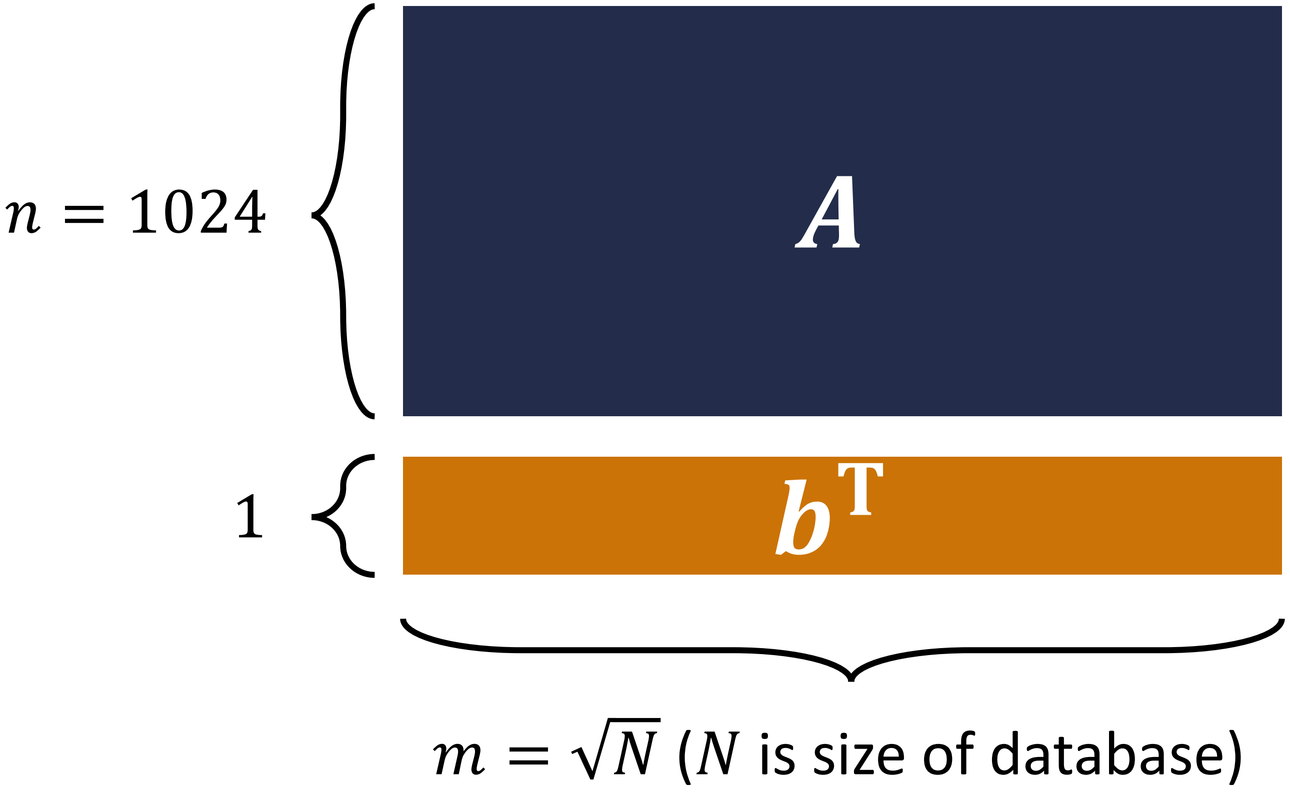
Query structure in SimplePIR

All components are elements of $\mathbb{Z}_q = \mathbb{Z}_{2^{32}}$ (i.e., 32-bit integers)

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Building block: Regev's linearly homomorphic encryption from LWE [Reg05]



Concretely:

- Suppose plaintext space (i.e., record size) is 1 byte (element of $\mathbb{Z}_p = \mathbb{Z}_{2^8}$)
- Secret key is $\boldsymbol{s} \in \mathbb{Z}_q^n$

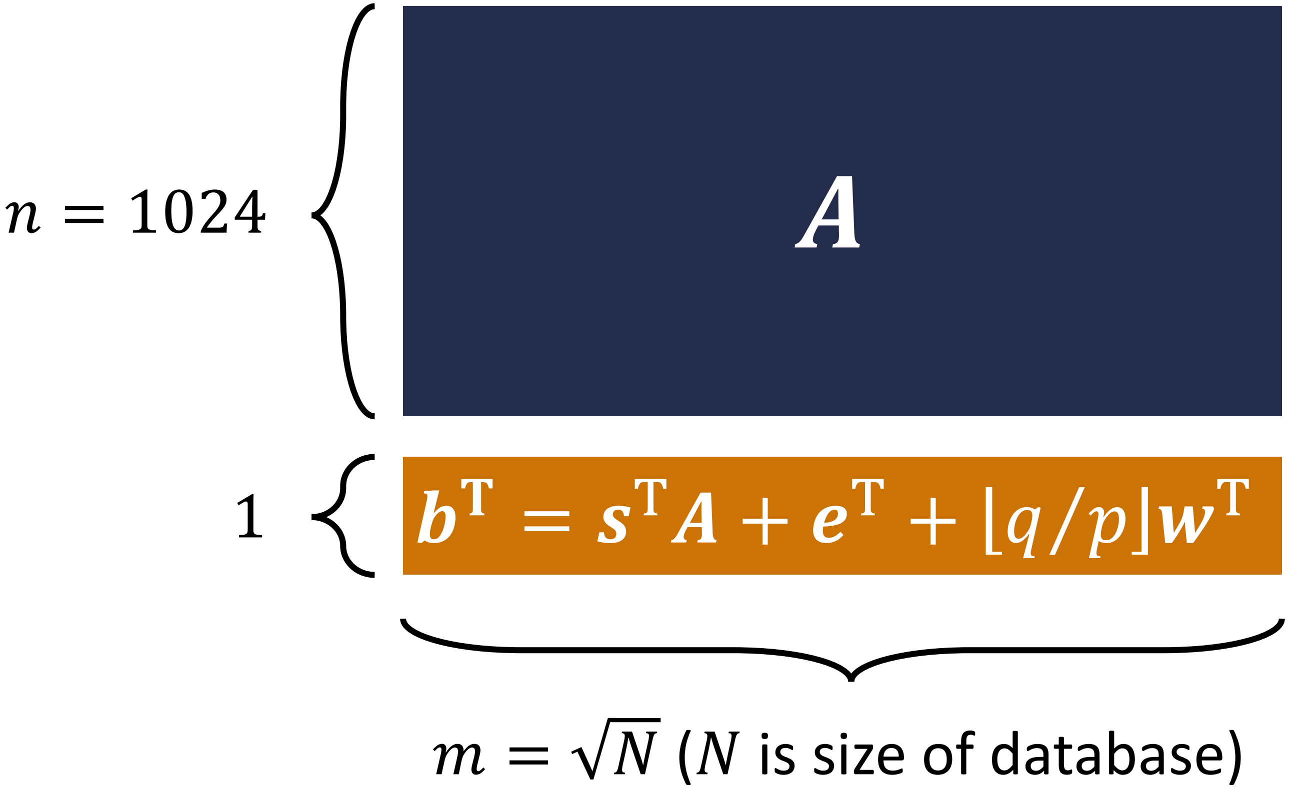
Query structure in SimplePIR

All components are elements of $\mathbb{Z}_q = \mathbb{Z}_{2^{32}}$ (i.e., 32-bit integers)

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Building block: Regev's linearly homomorphic encryption from LWE [Reg05]



Concretely:

- Suppose plaintext space (i.e., record size) is 1 byte (element of $\mathbb{Z}_p = \mathbb{Z}_{2^8}$)
- Secret key is $\boldsymbol{s} \in \mathbb{Z}_q^n$
- Ciphertext (encrypting $\boldsymbol{w}$) is
  $$\boldsymbol{b}^{\mathrm{T}} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{A} + \boldsymbol{e}^{\mathrm{T}} + \lfloor q/p \rfloor \boldsymbol{w}^{\mathrm{T}}$$
  where $\boldsymbol{e}^{\mathrm{T}}$ is small error vector
- To decrypt, compute
  $$\boldsymbol{b}^{\mathrm{T}} - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{e}^{\mathrm{T}} + \lfloor q/p \rfloor \boldsymbol{w}^{\mathrm{T}}$$
  and round

Ciphertext structure in Regev

All components are elements of $\mathbb{Z}_q = \mathbb{Z}_{2^{32}}$ (i.e., 32-bit integers)

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Building block: Regev's linearly homomorphic encryption from LWE [Reg05]

$n = 1024$ $\quad \boldsymbol{A}$

1 $\quad \boldsymbol{b}^{\mathrm{T}} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{A} + \boldsymbol{e}^{\mathrm{T}} + \lfloor q/p \rceil \boldsymbol{w}^{\mathrm{T}}$

$m = \sqrt{N}$ (N is size of database)

$\times$

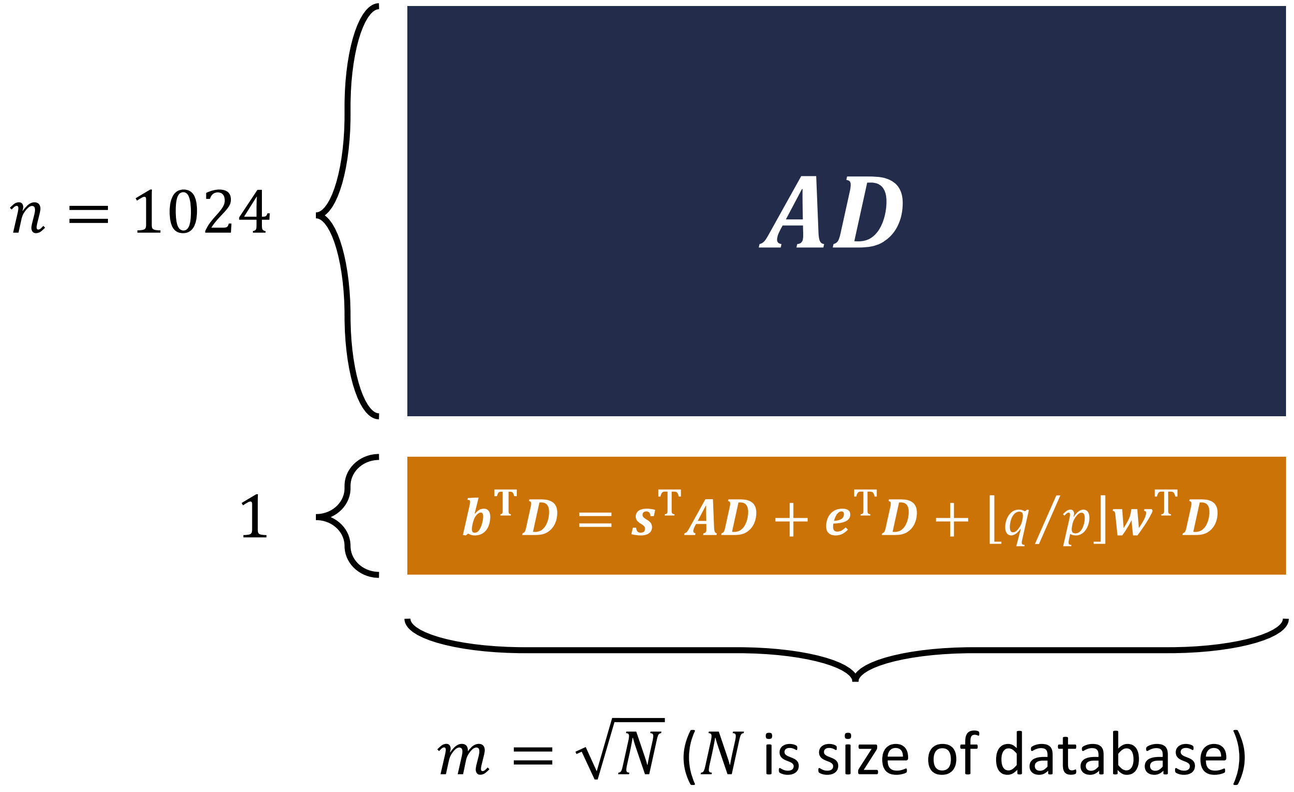
Database $\boldsymbol{D} \in \mathbb{Z}_p^{m \times m}$

All components are elements of $\mathbb{Z}_q = \mathbb{Z}_{2^{32}}$ (i.e., 32-bit integers)

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Building block: Regev's linearly homomorphic encryption from LWE [Reg05]



Decryption:

$$\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{AD} \approx \lfloor q/p \rfloor \boldsymbol{w}^{\mathrm{T}}\boldsymbol{D}$$

Suppose $\boldsymbol{w}$ is an indicator vector (i.e., $w_j = 1$ and $w_i = 0$ if $i \neq j$)

Then $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{D}$ is j^{th} row of $\boldsymbol{D}$

All components are elements of $\mathbb{Z}_q = \mathbb{Z}_{2^{32}}$ (i.e., 32-bit integers)

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

To recover record in row j:

$q = 2^{32}, p = 2^{8}, n = 2^{10}$

$\boldsymbol{A}$

$\boldsymbol{b}^{\mathrm{T}} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{A} + \boldsymbol{e}^{\mathrm{T}} + \lfloor q/p \rfloor \boldsymbol{w}^{\mathrm{T}}$

Query size:
$(n+1)\sqrt{N}$ elements over $\mathbb{Z}_q$

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

$\boldsymbol{AD}$

$\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}$

Response size:
$(n+1)\sqrt{N}$ elements over $\mathbb{Z}_q$

Server computation:
$2(n+1)N$ operations over $\mathbb{Z}_q$

2000 operations per byte of database!

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Key insight in SimplePIR: $\boldsymbol{A}$ is query-independent so move computation of $\boldsymbol{AD}$ offline

$\boldsymbol{A}$

$\boldsymbol{b}^{\mathrm{T}} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{A} + \boldsymbol{e}^{\mathrm{T}} + \lfloor q/p \rfloor \boldsymbol{w}^{\mathrm{T}}$

Query size:
$(n+1)\sqrt{N}$ elements over $\mathbb{Z}_q$

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

$\boldsymbol{AD}$

$\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}$

Response size:
$(n+1)\sqrt{N}$ elements over $\mathbb{Z}_q$

Server computation:
$2(n+1)N$ operations over $\mathbb{Z}_q$

2000 operations per byte of database!

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Key insight in SimplePIR: $\boldsymbol{A}$ is query-independent so move computation of $\boldsymbol{AD}$ offline

$\boldsymbol{AD}$

offline phase

online phase

Database

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

$\boldsymbol{b}^{\mathrm{T}} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{A} + \boldsymbol{e}^{\mathrm{T}} + \lfloor q/p \rfloor \boldsymbol{w}^{\mathrm{T}}$

$\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}$

Query and response size:
$\sqrt{N}$ elements over $\mathbb{Z}_q$

Server computation:
$2N$ operations over $\mathbb{Z}_q$

2 operations per byte of database!

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Key insight in SimplePIR: $\boldsymbol{A}$ is query-independent so move computation of $\boldsymbol{AD}$ offline

Limitation: hint is **large** ($n \times$ larger than response) and needs to be updated whenever database changes

$\boldsymbol{AD}$

offline phase

online phase

$\boldsymbol{b}^{\mathrm{T}} = \boldsymbol{s}^{\mathrm{T}}\boldsymbol{A} + \boldsymbol{e}^{\mathrm{T}} + \lfloor q/p \rceil \boldsymbol{w}^{\mathrm{T}}$

$\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}$

Database

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

On modern processors, can process query at *memory bandwidth* (best possible if protocol performs linear-scan)

2 operations per byte of database!

Query and response size:
$\sqrt{N}$ elements over $\mathbb{Z}_q$

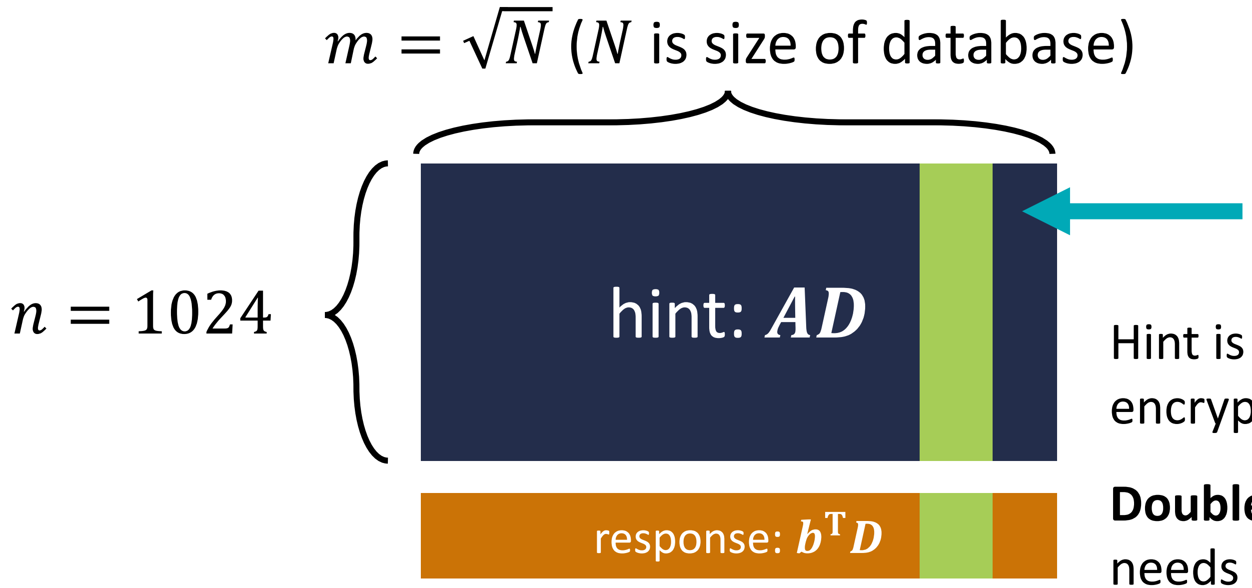
PIR with *Silent* Preprocessing

Silent preprocessing: allow **offline** preprocessing, but **no communication**

No need to manage hints; better suited for dynamic databases

Approach: *compress* the hint and include as part of the response [HintlessPIR; LMRS24] [YPIR; MW24]

Why is the SimplePIR hint so large?



j^{th} column of hint is the "randomness" for the ciphertext that encrypts the j^{th} record

Hint is large because Regev encryption has poor rate: encrypting one $\mathbb{Z}_p$ element requires $(n+1)$ $\mathbb{Z}_q$ elements

DoublePIR: use PIR to retrieve column j of hint; this still needs a hint for the (smaller) database $\boldsymbol{AD}$

SimplePIR using Polynomial Rings

Common technique to get better rate: use polynomial rings

Plaintext space: $\mathbb{Z}_p \longrightarrow R_p = \mathbb{Z}_p[x]/\left(x^d + 1\right)$

degree-d polynomials with coefficients in $\mathbb{Z}_p$

Ciphertext space:

$$\mathbb{Z}_q^{n+1} \longrightarrow R_q^2 = \left(\mathbb{Z}_q[x]/\left(x^d + 1\right)\right)^2$$

security relies on ring learning with errors (RLWE)

$$\text{rate} = \frac{\text{size of plaintext}}{\text{size of ciphertext}} = \frac{\log p}{(n+1)\log q} \longrightarrow \frac{\log p}{2\log q}$$

rings allow packing more data into each ciphertext

Recall $n = 2^{10} = 1024$, so using rings gives over 500× reduction in size

SimplePIR using Polynomial Rings

Common technique to get better rate: use polynomial rings

Plaintext space: $\mathbb{Z}_p \longrightarrow R_p = \mathbb{Z}_p[x]/\left(x^d + 1\right)$

degree-d polynomials with coefficients in $\mathbb{Z}_p$

Ciphertext space:

$$\mathbb{Z}_q^{n+1} \longrightarrow R_q^2 = \left(\mathbb{Z}_q[x]/\left(x^d + 1\right)\right)^2$$

security relies on ring learning with errors (RLWE)

SimplePIR over polynomial rings incurs $\approx \log q / \log p \approx 4$ overhead (due to blow-up in size of database representation)

$$\longrightarrow \frac{\log p}{2 \log q}$$

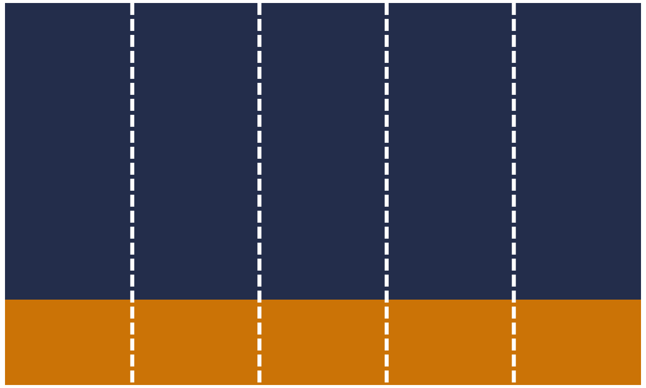
rings allow packing more data into each ciphertext

Recall $n = 2^{10} = 1024$, so using rings gives over 500× reduction in size

YPIR: Lightweight Hint Compression

[MW24]

YPIR approach: pack LWE ciphertexts into an RLWE ciphertext



Each column is an LWE ciphertext encrypting a record $z_i \in \mathbb{Z}_p$

Limitation: LWE ciphertext has **poor** rate

Observation: when $\boldsymbol{A}$ is a structured matrix then each column is also an RLWE ciphertext that encrypts a polynomial whose **constant** coefficient is z_i

For example, i^{th} column might decrypt to the polynomial

$$z_i + r_1 x + r_2 x^2 + \cdots + r_{n-1} x^{n-1}$$

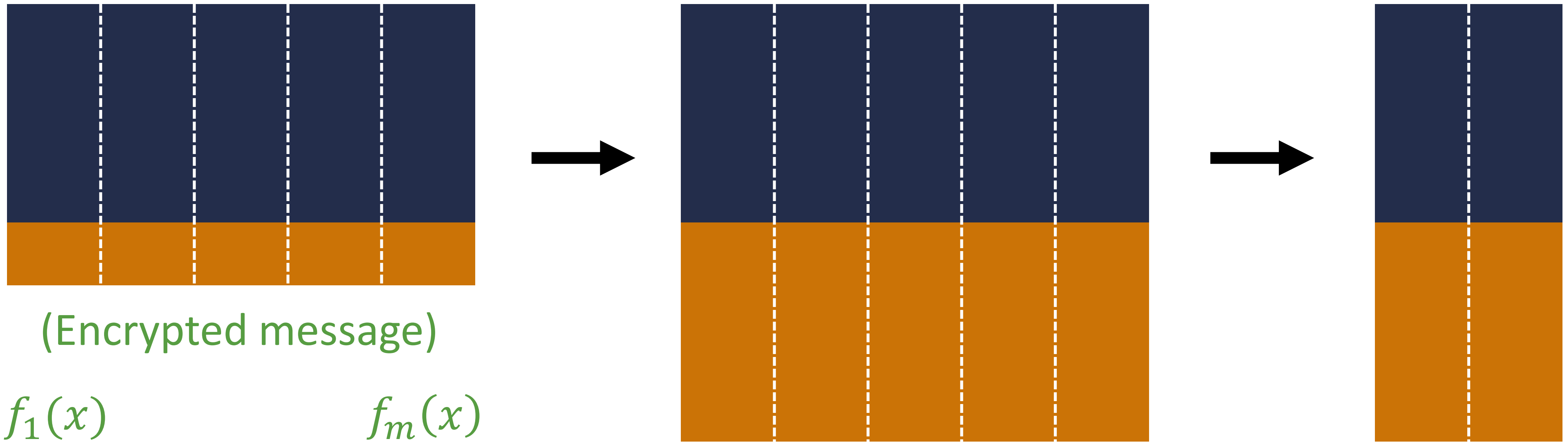
where $r_1, \ldots, r_{n-1}$ are (arbitrary) coefficients and $n = 1024$

Observation [CDKS21]: When $n = 2^d$, there is a simple homomorphic procedure that maps encryption of polynomial f to an encryption of $f(0)$

YPIR: Lightweight Hint Compression

[MW24]

YPIR approach: replace nested encryption with an LWE-to-RLWE packing technique



(Encrypted message)

$f_1(x)$ $f_m(x)$

$f_1(0) = z_1$ $f_m(0) = z_m$

z_1 z_2 z_3 $\cdots$ z_m

$\sum_{i \in [n]} z_i x^{i-1}$ $\sum_{i \in [n]} z_{i+n} x^{i-1}$

Step 1: Homomorphically extract constant term

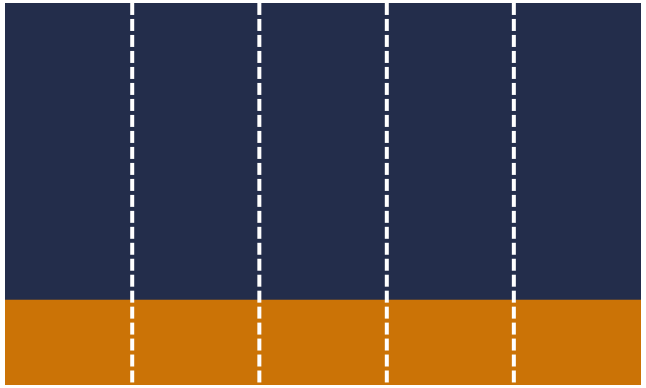
Each column now encrypts a **constant** polynomial (equal to the record)

Step 2: Homomorphically evaluate the linear function $\sum_{i \in [n]} z_i x^{i-1}$ on blocks of n columns

YPIR: Lightweight Hint Compression

[MW24]

YPIR approach: replace nested encryption with an LWE-to-RLWE packing technique



(Encrypted message)

$f_1(x)$ $f_m(x)$

$f_1(0) = z_1$ $f_m(0) = z_m$

Step 1: Homomorphically extract constant term

SimplePIR response size:

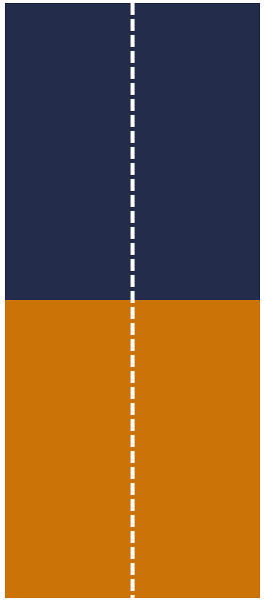
$(n+1)m$ elements of $\mathbb{Z}_q$

Packed RLWE response size:

$\frac{m}{n} \cdot 2n = 2m$ elements of $\mathbb{Z}_q$

Recall: $n \geq 2^{10}$, so substantial reduction in practice

Increases server compute, but observe that packing is applied to the SimplePIR **response**, which has size $O_\lambda(\sqrt{N})$



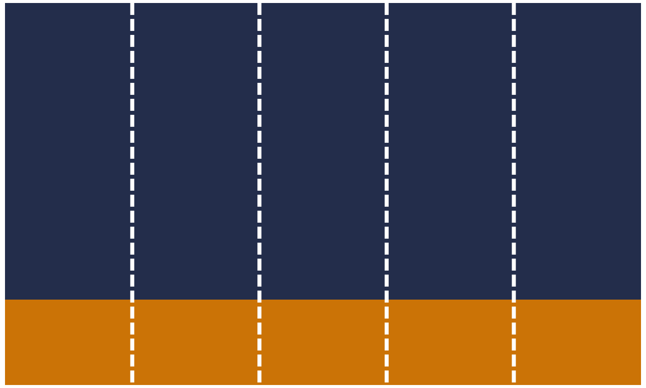
$\sum_{i \in [n]} z_i x^{i-1}$ $\sum_{i \in [n]} z_{i+n} x^{i-1}$

Step 2: Homomorphically evaluate the linear function $\sum_{i \in [n]} z_i x^{i-1}$ on blocks of n columns

YPIR: Lightweight Hint Compression

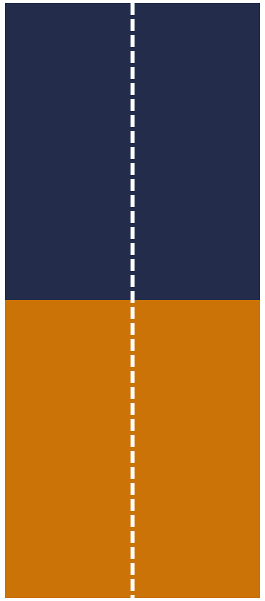
[MW24]

YPIR approach: replace nested encryption with an LWE-to-RLWE packing technique



SimplePIR response size:
$(n+1)m$ elements of $\mathbb{Z}_q$
Packed RLWE response size:
$\frac{m}{n} \cdot 2n = 2m$ elements of $\mathbb{Z}_q$

Recall: $n \geq 2^{10}$, so substantial reduction in practice



$\sum_{i\in[n]} z_i x^{i-1}$ $\sum_{i\in[n]} z_{i+n} x^{i-1}$

If we apply packing to DoublePIR (SimplePIR with 1 recursive step), then packing is applied to $O_\lambda(1)$ size response

Increases server compute, but observe that packing is applied to the SimplePIR **response**, which has size $O_\lambda(\sqrt{N})$

Step 2: Homomorphically evaluate the linear function $\sum_{i\in[n]} z_i x^{i-1}$ on blocks of n columns

YPIR: Lightweight Hint Compression

[MW24]

Compute:

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

AD

No communication!

offline phase

online phase

$b^{\mathrm{T}} = s^{\mathrm{T}}A + e^{\mathrm{T}} + \lfloor q/p \rfloor w^{\mathrm{T}}$

packing keys (key-switching)

packed response

PIR for Certificate Transparency Auditing

Setup: 5 billion SCTs, encoded as Bloom filter with 2^{36} bits (8 GB)

SCT audit consists of a single PIR query

[HHCMV23] uses DoublePIR

		DoublePIR	HintlessPIR	YPIR
Offline	Download	14 MB	–	–
Online	Upload	960 KB	1.4 MB	1.5 MB
	Download	12 KB	1.7 MB	12 KB
	Throughput	12.5 GB/s	4.9 GB/s	11.6 GB/s

1.6× larger queries and 93% of the throughput of fastest scheme

PIR for Certificate Transparency Auditing

Setup: 5 billion SCTs, encoded as Bloom filter with 2^{36} bits (8 GB)

SCT audit consists of a single PIR query

Assuming each client performs 20 SCT audits each week (based on client making 10^4 TLS connections and auditing $1/500$ fraction of connections) – achieves detection rate of $1/1000$ (Chrome's current approach)

Weekly server costs based on AWS cost model (free inbound communication) to support 1000 clients:

		DoublePIR	**DoublePIR**	**HintlessPIR**	**YPIR**	
	SCT Update Frequency	*Weekly*	*Daily*	*Daily*	*Daily*	
Server Costs	Communication	\$1.25	\$8.63	\$3.22	\$0.02	
	Computation	\$0.19	\$0.25	\$0.49	\$0.21	
	Total	\$1.44	\$8.88	\$3.71	\$0.23	84% cheaper!

PIR for Certificate Transparency Auditing

Setup: 5 billion SCTs, encoded as Bloom filter with 2^{36} bits (8 GB)

SCT audit consists of a single PIR query

Assuming each client performs 20 SCT audits each week (based on client making 10^4 TLS connections and auditing $1/500$ fraction of connections) – achieves detection rate of $1/1000$ (Chrome's current approach)

Weekly server costs based on AWS cost model (free inbound communication) to support 1000 clients:

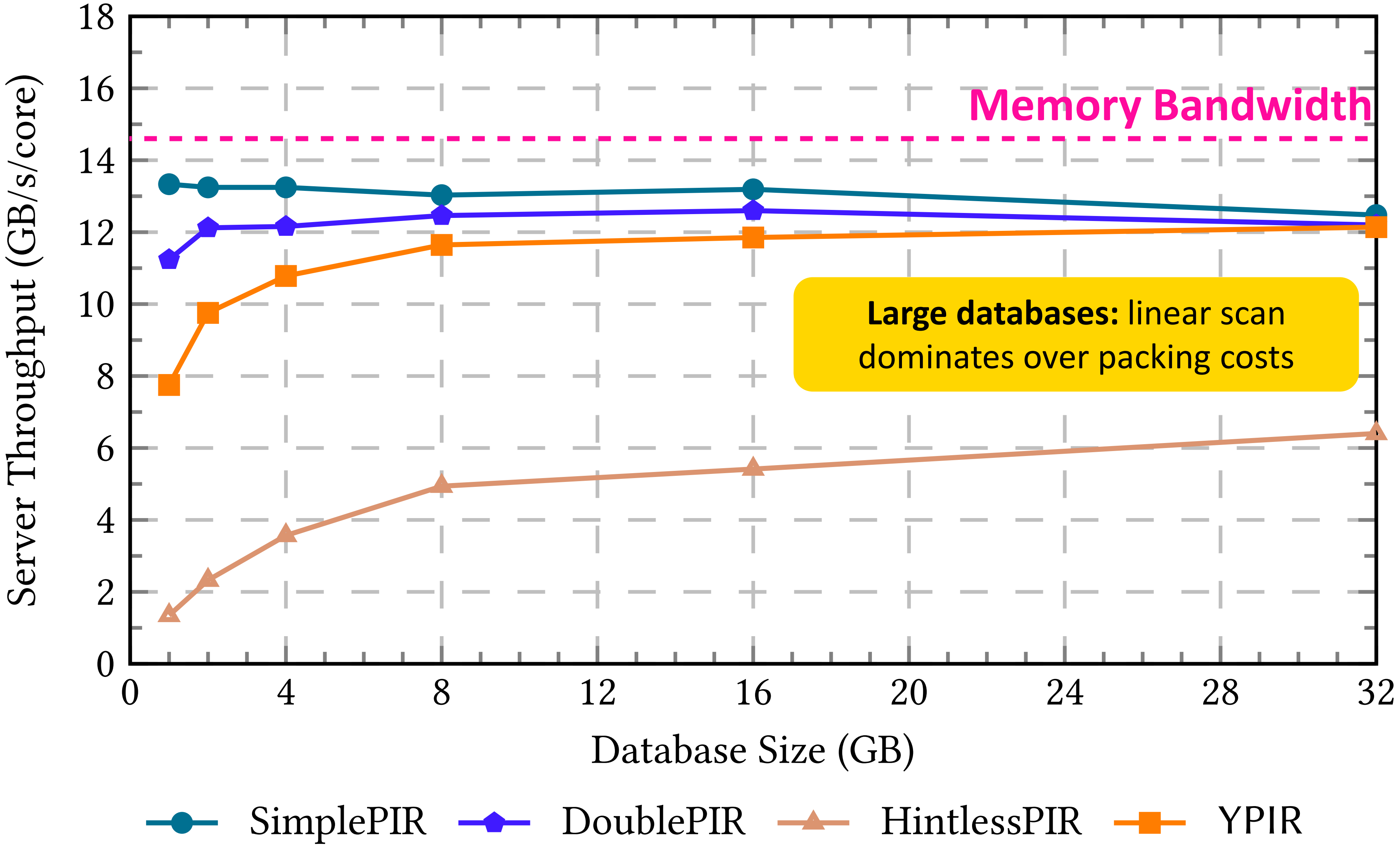
		DoublePIR	**DoublePIR**	**HintlessPIR**	**YPIR**
	SCT Update Frequency	*Weekly*	*Daily*	*Daily*	*Daily*
Server Costs	Communication				\$0.02
	Computation				\$0.21
	Total				\$0.23

k-anonymity (Chrome): 2.3 MB total communication per client per week

YPIR: 29 MB total communication per client per week (12.6× higher)

84% cheaper!

PIR Throughput



Retrieving Larger Records

Application: Password breach monitoring

Consider database with SHA-256 hashes of 250 million compromised passwords

Database size is 8 GB, partition into 250,000 records of size 32 KB

Technically, this is keyword PIR, but can be reduced to PIR via hashing

		SimplePIR	HintlessPIR	YPIR
Offline	Download	362 MB	–	–
Online	Upload	362 KB	1.4 MB	1.3 MB
	Download	362 KB	1.7 MB	228 KB
	Throughput	11 GB/s	5 GB/s	5 GB/s

Extra computational overhead from packing transformation

Recent Improvements (for Private SCT Auditing)

Distributional PIR [LHC25]**:** Better performance if we have prior information on query distribution (e.g., private SCT auditing)

- Distributional PIR + YPIR for private SCT auditing: reduces computational costs by 12× and communication by 3×
- **Cryptographic privacy** at 4× communication cost over Chrome's k-anonymity-based approach

System architecture for private SCT auditing [HPW25]**:** Consider better data structures and system architecture to support private SCT auditing – "*seem to bring wide SCT auditing to the brink of practicality*"

Existing single-server PIR protocols are fast enough to support some privacy-preserving applications at scale

Recent progress: Apple's use of PIR for private caller ID lookup and private image search

Recent Developments in PIR

The bottleneck for linear-scan-based PIR is the memory bandwidth, and recent schemes essentially hit this limit – will **not** cut it when database is 100 GB or 1 TB or even larger

Piano [ZPSZ23], QuarterPIR [GZS24], [RMS24], Plinko [HPPY25], [WR25]

- **Sublinear** server computational costs (can scale better to databases that are >100 GB)
- Preprocessing phase requires *streaming* the entire database (and client storing some state)
- Can avoid streaming the database in the two-server model (but rely on non-collusion assumption)

Doubly-Efficient PIR

[CHR17, BIPW17, LMW23]

- Server performs one-time encoding of the database
- In online phase, server can then answer queries by reading $\mathrm{polylog}(N)$ bits of the encoded value (no client-specific state needed)

Implication: private data access is essentially free (in an asymptotic sense)

Communication:

Without privacy: $\log N$ bits

With privacy: $\log N + \tilde{O}(\lambda)$ bits (where λ is a security parameter) [BV11]

Computation:

Without privacy: 1 probe

With privacy: $\mathrm{poly}(\lambda, \log N)$ probes [LMW23]

Doubly-Efficient PIR

[CHR17, BIPW17, LMW23]

- Server performs one-time encoding of the database
- In online phase, server can then answer queries by reading $\mathrm{polylog}(N)$ bits of the encoded value (no client-specific state needed)

Limitation: Still far from practical

For 350 MB database [OPPW24]:

- Size of server encoding is 412 PB (2^{59} bytes)
- Encoding time: $>$ 8 million core-years ($>$ \$4 billion)
- Communication: 573 MB (larger than database)
- Computation: 4.3 core-weeks (\$39/query)

Open problem: Practical doubly-efficient PIR (interesting even in multi-server setting)

Open Problems

Reduce concrete communication costs of PIR with silent preprocessing

Current approaches all have $O(\sqrt{N})$ communication – can we get to $\mathrm{polylog}(N)$ with similar (concrete) computational overhead

Practical doubly-efficient PIR (single-server ***or*** multi-server)

What will it take to deploy PIR in practical systems?

Recent progress: Apple's use of PIR for private caller ID lookup and private image search

Thank you!