Hintless Single-Server PIR

Baiyu Li [1] Daniele Micciancio [2] Mariana Raykova [1]
Mark Schultz-Wu [2,3]

[1]Google

[2]University of California San Diego

19 August 2024

[3]Work performed at Google

Contents

Single-Server PIR

Private Information Retrieval

Client$(i \in [m])$		**Server**$(\mathrm{db} \in \Omega^m)$
$\mathrm{qu} \leftarrow \mathrm{Query}(i)$		
	$\xrightarrow{\mathrm{qu}}$	
		$\mathrm{rsp} \leftarrow \mathrm{Response}(\mathrm{qu}, \mathrm{db})$
	$\xleftarrow{\mathrm{rsp}}$	
$\mathrm{db}_* \leftarrow \mathrm{Recover}(\mathrm{rsp}, i)$		

Single-Server PIR

Private Information Retrieval

Client$(i \in [m])$		**Server**$(\mathrm{db} \in \Omega^m)$
$\mathrm{qu} \leftarrow \mathrm{Query}(i)$		
	$\xrightarrow{\mathrm{qu}}$	
		$\mathrm{rsp} \leftarrow \mathrm{Response}(\mathrm{qu}, \mathrm{db})$
	$\xleftarrow{\mathrm{rsp}}$	
$\mathrm{db}_* \leftarrow \mathrm{Recover}(\mathrm{rsp}, i)$		

- **Correct**: $\mathrm{db}_* = \mathrm{db}[i]$

Single-Server PIR

Private Information Retrieval

Client$(i \in [m])$		**Server**$(\mathsf{db} \in \Omega^m)$
$\mathsf{qu} \leftarrow \mathsf{Query}(i)$		
	$\xrightarrow{\mathsf{qu}}$	
		$\mathsf{rsp} \leftarrow \mathsf{Response}(\mathsf{qu}, \mathsf{db})$
	$\xleftarrow{\mathsf{rsp}}$	
$\mathsf{db}_* \leftarrow \mathsf{Recover}(\mathsf{rsp}, i)$		

- **Correct**: $\mathsf{db}_* = \mathsf{db}[i]$
- **Secure**: $\forall i, j \in [m]$, $\{\mathsf{Query}(i)\} \approx_c \{\mathsf{Query}(j)\}$

Single Server PIR

- Trivial: Server transmits db to the Client

Single Server PIR

- Trivial: Server transmits db to the Client
  - Info-theoretically secure + correct

Single Server PIR

- Trivial: Server transmits db to the Client
  - Info-theoretically secure + correct
  - Bandwidth $O(|\text{db}|)$, compute $O(|\text{db}|)$

Single Server PIR

- Trivial: Server transmits db to the Client
  - Info-theoretically secure + correct
  - Bandwidth $O(|\text{db}|)$, compute $O(|\text{db}|)$
- HE-based: Client sends $\text{Enc}(i)$ (and maybe ek), server hom. computes $i \mapsto \text{db}[i]$
  - Secure+Correct if HE is (IND-CPA) Secure+Correct

Single Server PIR

- Trivial: Server transmits db to the Client
  - Info-theoretically secure + correct
  - Bandwidth $O(|\mathrm{db}|)$, compute $O(|\mathrm{db}|)$
- HE-based: Client sends $\mathrm{Enc}(i)$ (and maybe ek), server hom. computes $i \mapsto \mathrm{db}[i]$
  - Secure+Correct if HE is (IND-CPA) Secure+Correct
- Metrics to Optimize:
  - Bandwidth

Single Server PIR

- Trivial: Server transmits db to the Client
  - Info-theoretically secure + correct
  - Bandwidth $O(|\mathrm{db}|)$, compute $O(|\mathrm{db}|)$
- HE-based: Client sends $\mathrm{Enc}(i)$ (and maybe ek), server hom. computes $i \mapsto \mathrm{db}[i]$
  - Secure+Correct if HE is (IND-CPA) Secure+Correct
- Metrics to Optimize:
  - Bandwidth
  - Server Compute

Single Server PIR

- Trivial: Server transmits db to the Client
  - Info-theoretically secure + correct
  - Bandwidth $O(|\mathrm{db}|)$, compute $O(|\mathrm{db}|)$
- HE-based: Client sends $\mathrm{Enc}(i)$ (and maybe ek), server hom. computes $i \mapsto \mathrm{db}[i]$
  - Secure+Correct if HE is (IND-CPA) Secure+Correct
- Metrics to Optimize:
  - Bandwidth
  - Server Compute
  - Preprocessing Costs

PIR from HE: SOTA

- SimplePIR and DoublePIR [HHCMV Usenix23]
  - Pros: Extremely fast (online) server processing
  - Cons: Large queries, require a database-dependent "hint" (which requires expensive server pre-processing to generate)

PIR from HE: SOTA

- SimplePIR and DoublePIR [HHCMV Usenix23]
  - Pros: Extremely fast (online) server processing
  - Cons: Large queries, require a database-dependent “hint” (which requires expensive server pre-processing to generate)
    - $2^{20} \times 256$B (286MB) Database has hint 1/3 DB size (SimplePIR), and $25\times$ DB size (DoublePIR)

PIR from HE: SOTA

- SimplePIR and DoublePIR [HHCMV Usenix23]
  - Pros: Extremely fast (online) server processing
  - Cons: Large queries, require a database-dependent "hint" (which requires expensive server pre-processing to generate)
    - $2^{20} \times 256$B (286MB) Database has hint 1/3 DB size (SimplePIR), and $25\times$ DB size (DoublePIR)

PIR from HE: SOTA

- SimplePIR and DoublePIR [HHCMV Usenix23]
  - Pros: Extremely fast (online) server processing
  - Cons: Large queries, require a database-dependent "hint" (which requires expensive server pre-processing to generate)
    - $2^{20} \times 256$B (286MB) Database has hint 1/3 DB size (SimplePIR), and 25$\times$ DB size (DoublePIR)
- Spiral [MW SSP22]
  - Pros: Small queries
  - Cons: $\approx 10\times$ slower, moderately-sized evaluation keys

PIR from HE: SOTA

- SimplePIR and DoublePIR [HHCMV Usenix23]
  - Pros: Extremely fast (online) server processing
  - Cons: Large queries, require a database-dependent "hint" (which requires expensive server pre-processing to generate)
    - $2^{20} \times 256$B (286MB) Database has hint 1/3 DB size (SimplePIR), and $25\times$ DB size (DoublePIR)
- Spiral [MW SSP22]
  - Pros: Small queries
  - Cons: $\approx 10\times$ slower, moderately-sized evaluation keys
- TiptoePIR [HDCZ SOSP23] (Concurrent to our work)
  - Pros: Removes the hint from SimplePIR!
  - Cons: $+\Omega(n^2 \log q)$ ($\approx$ 23MB) per-query bandwidth cost

LWE Encryption

- Ciphertexts: $[A, \vec{b}]$ s.t. $\vec{b} - A \cdot \vec{s} \approx \Delta \vec{m}$

LWE Encryption

- Ciphertexts: $[A, \vec{b}]$ s.t. $\vec{b} - A \cdot \vec{s} \approx \Delta \vec{m}$
  - $\|\vec{s}\|_\infty$ small,

LWE Encryption

- Ciphertexts: $[A, \vec{b}]$ s.t. $\vec{b} - A \cdot \vec{s} \approx \Delta\vec{m}$
  - $\|\vec{s}\|_\infty$ small,
  - $A \in \mathbb{Z}_q^{m \times n}$ can be reused

LWE Encryption

- Ciphertexts: $[A, \vec{b}]$ s.t. $\vec{b} - A \cdot \vec{s} \approx \Delta \vec{m}$
  - $\|\vec{s}\|_\infty$ small,
  - $A \in \mathbb{Z}_q^{m \times n}$ can be reused
  - Easily supports hom. computation of $\vec{x} \mapsto \mathrm{db} \cdot \vec{x}$

LWE Encryption

- Ciphertexts: $[A, \vec{b}]$ s.t. $\vec{b} - A \cdot \vec{s} \approx \Delta \vec{m}$
  - $\|\vec{s}\|_\infty$ small,
  - $A \in \mathbb{Z}_q^{m \times n}$ can be reused
  - Easily supports hom. computation of $\vec{x} \mapsto \text{db} \cdot \vec{x}$
    - $\text{db} \cdot [A, \vec{b}]$ has large component $H := \text{db} \cdot A$

LWE Encryption

- Ciphertexts: $[A, \vec{b}]$ s.t. $\vec{b} - A \cdot \vec{s} \approx \Delta \vec{m}$
  - $\|\vec{s}\|_\infty$ small,
  - $A \in \mathbb{Z}_q^{m \times n}$ can be reused
  - Easily supports hom. computation of $\vec{x} \mapsto \mathrm{db} \cdot \vec{x}$
    - $\mathrm{db} \cdot [A, \vec{b}]$ has large component $H := \mathrm{db} \cdot A$
  - General moduli q

LWE Encryption

- Ciphertexts: $[A, \vec{b}]$ s.t. $\vec{b} - A \cdot \vec{s} \approx \Delta \vec{m}$
  - $\|\vec{s}\|_\infty$ small,
  - $A \in \mathbb{Z}_q^{m \times n}$ can be reused
  - Easily supports hom. computation of $\vec{x} \mapsto \mathrm{db} \cdot \vec{x}$
    - $\mathrm{db} \cdot [A, \vec{b}]$ has large component $H := \mathrm{db} \cdot A$
  - General moduli $q \in \{2^{32}, 2^{64}\}$

SimplePIR's Hint

- SimplePIR (and DoublePIR) requires clients to download a hint $H := \mathsf{db} \cdot A$

SimplePIR's Hint

- SimplePIR (and DoublePIR) requires clients to download a hint $H := \mathsf{db} \cdot A$
- Clients only use this to compute $\vec{s} \mapsto H \cdot \vec{s}$ for LWE key $\vec{s}$

SimplePIR's Hint

- SimplePIR (and DoublePIR) requires clients to download a hint $H := \mathrm{db} \cdot A$
- Clients only use this to compute $\vec{s} \mapsto H \cdot \vec{s}$ for LWE key $\vec{s}$
  - Almost a PIR query, which is solved trivially

SimplePIR's Hint

- SimplePIR (and DoublePIR) requires clients to download a hint $H := \text{db} \cdot A$
- Clients only use this to compute $\vec{s} \mapsto H \cdot \vec{s}$ for LWE key $\vec{s}$
  - Almost a PIR query, which is solved trivially
- Our Work: Use non-trivial (albeit technically straightforward) RLWE-based PIR instead

SimplePIR's Hint

- SimplePIR (and DoublePIR) requires clients to download a hint $H := \text{db} \cdot A$
- Clients only use this to compute $\vec{s} \mapsto H \cdot \vec{s}$ for LWE key $\vec{s}$
  - Almost a PIR query, which is solved trivially
- Our Work: Use non-trivial (albeit technically straightforward) RLWE-based PIR instead
  - Speed up with novel precomputation argument

SimplePIR's Hint

- SimplePIR (and DoublePIR) requires clients to download a hint $H := \mathrm{db} \cdot A$
- Clients only use this to compute $\vec{s} \mapsto H \cdot \vec{s}$ for LWE key $\vec{s}$
  - Almost a PIR query, which is solved trivially
- Our Work: Use non-trivial (albeit technically straightforward) RLWE-based PIR instead
  - Speed up with novel precomputation argument
  - Also a PIR with $O(\sqrt[3]{|\mathrm{db}|})$ bandwidth scaling (TensorPIR)

Why Keep SimplePIR?

- $T_{\mathsf{SimplePIR}}(|\mathrm{db}|) = 2|\mathrm{db}|$ operations in $\mathbb{Z}_q$

Why Keep SimplePIR?

- $T_{\mathsf{SimplePIR}}(|\mathsf{db}|) = 2|\mathsf{db}|$ operations in $\mathbb{Z}_q$
- $T_{\mathsf{NTTlessPIR}}(|\mathsf{db}|) = O(|\mathsf{db}|)$ operations in $\mathbb{Z}_{q^*}$

Why Keep SimplePIR?

- $T_{\mathsf{SimplePIR}}(|\mathsf{db}|) = 2|\mathsf{db}|$ operations in $\mathbb{Z}_q$
- $T_{\mathsf{NTTlessPIR}}(|\mathsf{db}|) = O(|\mathsf{db}|)$ operations in $\mathbb{Z}_{q^*}$
  - for $q^* \notin \{2^{32}, 2^{64}\}$

Why Keep SimplePIR?

- $T_{\mathsf{SimplePIR}}(|\mathrm{db}|) = 2|\mathrm{db}|$ operations in $\mathbb{Z}_q$
- $T_{\mathsf{NTTlessPIR}}(|\mathrm{db}|) = O(|\mathrm{db}|)$ operations in $\mathbb{Z}_{q^*}$
  - for $q^* \notin \{2^{32}, 2^{64}\}$
  - $\approx 10\times$ slower than $q \in \{2^{32}, 2^{64}\}$

Why Keep SimplePIR?

- $T_{\text{SimplePIR}}(|\text{db}|) = 2|\text{db}|$ operations in $\mathbb{Z}_q$
- $T_{\text{NTTlessPIR}}(|\text{db}|) = O(|\text{db}|)$ operations in $\mathbb{Z}_{q^*}$
  - for $q^* \notin \{2^{32}, 2^{64}\}$
  - $\approx 10\times$ slower than $q \in \{2^{32}, 2^{64}\}$
- $T_{\text{HintlessPIR}}(|\text{db}|) = T_{\text{SimplePIR}}(|\text{db}|) + T_{\text{NTTlessPIR}}(n\sqrt{|\text{db}|})$

Why Keep SimplePIR?

- $T_{\text{SimplePIR}}(|\text{db}|) = 2|\text{db}|$ operations in $\mathbb{Z}_q$
- $T_{\text{NTTlessPIR}}(|\text{db}|) = O(|\text{db}|)$ operations in $\mathbb{Z}_{q^*}$
  - for $q^* \notin \{2^{32}, 2^{64}\}$
  - $\approx 10\times$ slower than $q \in \{2^{32}, 2^{64}\}$
- $T_{\text{HintlessPIR}}(|\text{db}|) = T_{\text{SimplePIR}}(|\text{db}|) + T_{\text{NTTlessPIR}}(n\sqrt{|\text{db}|})$
  - $2|\text{db}|$ operations in $\mathbb{Z}_q$, and $O(n\sqrt{|\text{db}|})$ operations in $\mathbb{Z}_{q^*}$.

Column-Major Matrix-Vector Multiplication

$$\begin{pmatrix} \vdots & \vdots & \dots & \vdots \\ \vec{a}_1 & \vec{a}_2 & \dots & \vec{a}_n \\ \vdots & \vdots & \dots & \vdots \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vdots \\ \vdots \\ \vec{s}_n \end{pmatrix} = \sum_{i \in [n]} \vec{s}_i \cdot \begin{pmatrix} \vdots \\ \vec{a}_i \\ \vdots \end{pmatrix}$$

Column-Major Matrix-Vector Multiplication

$$\begin{pmatrix} \vdots & \vdots & \dots & \vdots \\ \vec{a}_1 & \vec{a}_2 & \dots & \vec{a}_n \\ \vdots & \vdots & \dots & \vdots \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vdots \\ \vdots \\ \vec{s}_n \end{pmatrix} = \sum_{i \in [n]} \vec{s}_i \cdot \begin{pmatrix} \vdots \\ \vec{a}_i \\ \vdots \end{pmatrix}$$

- Can encrypt $\{\vec{s}_i\}_i$, compute above with linearly homomorphic encryption (e.g. RLWE)

Column-Major Matrix-Vector Multiplication

$$\begin{pmatrix} \vdots & \vdots & \dots & \vdots \\ \vec{a}_1 & \vec{a}_2 & \dots & \vec{a}_n \\ \vdots & \vdots & \dots & \vdots \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vdots \\ \vdots \\ \vec{s}_n \end{pmatrix} = \sum_{i \in [n]} \vec{s}_i \cdot \begin{pmatrix} \vdots \\ \vec{a}_i \\ \vdots \end{pmatrix}$$

- Can encrypt $\{\vec{s}_i\}_i$, compute above with linearly homomorphic encryption (e.g. RLWE)
  - What TiptoePIR does

Column-Major Matrix-Vector Multiplication

$$\begin{pmatrix} \vdots & \vdots & \dots & \vdots \\ \vec{a}_1 & \vec{a}_2 & \dots & \vec{a}_n \\ \vdots & \vdots & \dots & \vdots \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vdots \\ \vdots \\ \vec{s}_n \end{pmatrix} = \sum_{i \in [n]} \vec{s}_i \cdot \begin{pmatrix} \vdots \\ \vec{a}_i \\ \vdots \end{pmatrix}$$

- Can encrypt $\{\vec{s}_i\}_i$, compute above with linearly homomorphic encryption (e.g. RLWE)
  - What TiptoePIR does
- Con: Requires n RLWE ciphertexts, $\Theta(n^2 \log q)$ bandwidth

Diagonally-Dominant Matrix-Vector Multiplication

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vec{s}_3 \end{pmatrix}$$

Diagonally-Dominant Matrix-Vector Multiplication

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vec{s}_3 \end{pmatrix} = \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{21}\vec{s}_1 + a_{22}\vec{s}_2 + a_{23}\vec{s}_3 \\ a_{31}\vec{s}_1 + a_{32}\vec{s}_2 + a_{33}\vec{s}_3 \end{pmatrix}$$

Diagonally-Dominant Matrix-Vector Multiplication

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vec{s}_3 \end{pmatrix} = \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{21}\vec{s}_1 + a_{22}\vec{s}_2 + a_{23}\vec{s}_3 \\ a_{31}\vec{s}_1 + a_{32}\vec{s}_2 + a_{33}\vec{s}_3 \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{22}\vec{s}_2 + a_{23}\vec{s}_3 + a_{21}\vec{s}_1 \\ a_{33}\vec{s}_3 + a_{31}\vec{s}_1 + a_{32}\vec{s}_2 \end{pmatrix}$$

Diagonally-Dominant Matrix-Vector Multiplication

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vec{s}_3 \end{pmatrix} = \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{21}\vec{s}_1 + a_{22}\vec{s}_2 + a_{23}\vec{s}_3 \\ a_{31}\vec{s}_1 + a_{32}\vec{s}_2 + a_{33}\vec{s}_3 \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{22}\vec{s}_2 + a_{23}\vec{s}_3 + a_{21}\vec{s}_1 \\ a_{33}\vec{s}_3 + a_{31}\vec{s}_1 + a_{32}\vec{s}_2 \end{pmatrix}$$

$$= \sum_i \mathrm{diag}_i(A) \circ \mathrm{rot}^i(\vec{s})$$

Diagonally-Dominant Matrix-Vector Multiplication

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vec{s}_3 \end{pmatrix} = \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{21}\vec{s}_1 + a_{22}\vec{s}_2 + a_{23}\vec{s}_3 \\ a_{31}\vec{s}_1 + a_{32}\vec{s}_2 + a_{33}\vec{s}_3 \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{22}\vec{s}_2 + a_{23}\vec{s}_3 + a_{21}\vec{s}_1 \\ a_{33}\vec{s}_3 + a_{31}\vec{s}_1 + a_{32}\vec{s}_2 \end{pmatrix}$$

$$= \sum_i \mathrm{diag}_i(A) \circ \mathrm{rot}^i(\vec{s})$$

- Small overhead if one can hom. evaluate $\vec{s} \mapsto \mathrm{rot}(\vec{s})$

Diagonally-Dominant Matrix-Vector Multiplication

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} \vec{s}_1 \\ \vec{s}_2 \\ \vec{s}_3 \end{pmatrix} = \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{21}\vec{s}_1 + a_{22}\vec{s}_2 + a_{23}\vec{s}_3 \\ a_{31}\vec{s}_1 + a_{32}\vec{s}_2 + a_{33}\vec{s}_3 \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}\vec{s}_1 + a_{12}\vec{s}_2 + a_{13}\vec{s}_3 \\ a_{22}\vec{s}_2 + a_{23}\vec{s}_3 + a_{21}\vec{s}_1 \\ a_{33}\vec{s}_3 + a_{31}\vec{s}_1 + a_{32}\vec{s}_2 \end{pmatrix}$$

$$= \sum_i \mathrm{diag}_i(A) \circ \mathrm{rot}^i(\vec{s})$$

- Small overhead if one can hom. evaluate $\vec{s} \mapsto \mathrm{rot}(\vec{s})$
  - Our Work: Use precomputation to make this fast

Contents

RLWE Notation

- $R_{n,q} := \mathbb{Z}_q[x]/(x^n + 1)$ for $n = 2^k$

RLWE Notation

- $R_{n,q} := \mathbb{Z}_q[x]/(x^n + 1)$ for $n = 2^k$
- For $q \equiv 1 \bmod 2n$ (NTT friendly q), exists a ring isomorphism

$$\mathsf{NTT} : (R_{n,q}, +, *) \to (\mathbb{Z}_q^n, +, \circ)$$

computable in $O(n \log n)$ $\mathbb{Z}_q$-ops

RLWE Notation

- $R_{n,q} := \mathbb{Z}_q[x]/(x^n + 1)$ for $n = 2^k$
- For $q \equiv 1 \bmod 2n$ (NTT friendly q), exists a ring isomorphism

$$\mathsf{NTT} : (R_{n,q}, +, *) \to (\mathbb{Z}_q^n, +, \circ)$$

  computable in $O(n \log n)$ $\mathbb{Z}_q$-ops
- Use standard RLWE (and gadget RLWE) encryption with plaintext packing

$\diamond$-Product: Standard Domain

$\diamond$-Product

Let $\vec{\mathrm{ct}} \in (R_{n,q} \times R_{n,q})^\ell$ be a collection of ℓ RLWE ciphertexts. For a polynomial $a \in R_{n,q}$, define

$$a \diamond \vec{\mathrm{ct}} = \sum_{i \in [\ell]} \mathrm{GadgetDecompose}(a)_i * \vec{\mathrm{ct}}_i$$

$\diamond$-Product: Standard Domain

$\diamond$-Product

Let $\vec{\mathrm{ct}} \in (R_{n,q} \times R_{n,q})^\ell$ be a collection of ℓ RLWE ciphertexts. For a polynomial $a \in R_{n,q}$, define

$$a \diamond \vec{\mathrm{ct}} = \sum_{i \in [\ell]} \mathrm{GadgetDecompose}(a)_i * \vec{\mathrm{ct}}_i$$

- Fundamental Operation for RLWE-based HE

$\diamond$-Product: Standard Domain

$\diamond$-Product

Let $\vec{\mathrm{ct}} \in (R_{n,q} \times R_{n,q})^\ell$ be a collection of ℓ RLWE ciphertexts. For a polynomial $a \in R_{n,q}$, define

$$a \diamond \vec{\mathrm{ct}} = \sum_{i \in [\ell]} \mathrm{GadgetDecompose}(a)_i * \vec{\mathrm{ct}}_i$$

- Fundamental Operation for RLWE-based HE
  - Main part of gadget-based key-switching

$\diamond$-Product: Standard Domain

$\diamond$-Product

Let $\vec{\mathrm{ct}} \in (R_{n,q} \times R_{n,q})^\ell$ be a collection of ℓ RLWE ciphertexts. For a polynomial $a \in R_{n,q}$, define

$$a \diamond \vec{\mathrm{ct}} = \sum_{i \in [\ell]} \mathrm{GadgetDecompose}(a)_i * \vec{\mathrm{ct}}_i$$

- Fundamental Operation for RLWE-based HE
  - Main part of gadget-based key-switching
    - And GSW/FHEW/TFHE/etc.

$\diamond$-Product: Standard Domain

$\diamond$-Product

Let $\vec{\text{ct}} \in (R_{n,q} \times R_{n,q})^\ell$ be a collection of ℓ RLWE ciphertexts. For a polynomial $a \in R_{n,q}$, define

$$a \diamond \vec{\text{ct}} = \sum_{i \in [\ell]} \text{GadgetDecompose}(a)_i * \vec{\text{ct}}_i$$

- Fundamental Operation for RLWE-based HE
  - Main part of gadget-based key-switching
    - And GSW/FHEW/TFHE/etc.
  - Implies hom. rotation $\vec{s} \mapsto \text{rot}(\vec{s})$

$\diamond$-Product: Standard Domain

$\diamond$-Product

Let $\vec{\mathrm{ct}} \in (R_{n,q} \times R_{n,q})^\ell$ be a collection of ℓ RLWE ciphertexts. For a polynomial $a \in R_{n,q}$, define

$$a \diamond \vec{\mathrm{ct}} = \sum_{i\in[\ell]} \mathrm{GadgetDecompose}(a)_i * \vec{\mathrm{ct}}_i$$

- Fundamental Operation for RLWE-based HE
  - Main part of gadget-based key-switching
    - And GSW/FHEW/TFHE/etc.
  - Implies hom. rotation $\vec{s} \mapsto \mathrm{rot}(\vec{s})$
- As written, is $O(n^2 \log q)$ time due to $*$

$\diamond$-Product: NTT Domain

- If we assume $a, \vec{\mathsf{ct}}$ start in NTT domain $(\widehat{a}, \widehat{\vec{\mathsf{ct}}})$, and want result in NTT domain, instead must compute

$$\widehat{a \diamond \vec{\mathsf{ct}}} = \sum_{i \in [\ell]} \mathsf{NTT}(\mathsf{GadgetDecompose}(\mathsf{NTT}^{-1}(\widehat{a}))_i) \circ \widehat{\vec{\mathsf{ct}}_i}$$

$\diamond$-Product: NTT Domain

- If we assume $a, \vec{\mathsf{ct}}$ start in NTT domain $(\widehat{a}, \widehat{\vec{\mathsf{ct}}})$, and want result in NTT domain, instead must compute

$$\widehat{a \diamond \vec{\mathsf{ct}}} = \sum_{i \in [\ell]} \mathsf{NTT}(\mathsf{GadgetDecompose}(\mathsf{NTT}^{-1}(\widehat{a}))_i) \circ \widehat{\vec{\mathsf{ct}}}_i$$

- Have to switch back to coefficient domain to compute the coefficients

$$\{\mathsf{NTT}(\mathsf{GadgetDecompose}(\mathsf{NTT}^{-1}(\widehat{a}))_i)\}_{i \in [\ell]}$$

$\diamond$-Product: NTT Domain

- If we assume $a, \vec{\text{ct}}$ start in NTT domain $(\widehat{a}, \widehat{\vec{\text{ct}}})$, and want result in NTT domain, instead must compute

$$\widehat{a \diamond \vec{\text{ct}}} = \sum_{i \in [\ell]} \text{NTT}(\text{GadgetDecompose}(\text{NTT}^{-1}(\widehat{a}))_i) \circ \widehat{\vec{\text{ct}}_i}$$

- Have to switch back to coefficient domain to compute the coefficients

$$\{\text{NTT}(\text{GadgetDecompose}(\text{NTT}^{-1}(\widehat{a}))_i)\}_{i \in [\ell]}$$

- Precompute?

Composable Preprocessing

- For RLWE ctxt $\mathsf{ct} = [a, b]$, define

$$\alpha(\mathsf{ct}) = a, \qquad \beta(\mathsf{ct}) = b$$

Composable Preprocessing

- For RLWE ctxt $\mathsf{ct} = [a, b]$, define

$$\alpha(\mathsf{ct}) = a, \qquad \beta(\mathsf{ct}) = b$$

- These are homomorphic with respect $\{+, *\}$, e.g.

$$\alpha(\mathsf{ct}_0 + \mathsf{ct}_1) = \alpha(\mathsf{ct}_0) + \alpha(\mathsf{ct}_1)$$

Composable Preprocessing

- For RLWE ctxt $\mathsf{ct} = [a, b]$, define

$$\alpha(\mathsf{ct}) = a, \qquad \beta(\mathsf{ct}) = b$$

- These are homomorphic with respect $\{+, *\}$, e.g.

$$\alpha(\mathsf{ct}_0 + \mathsf{ct}_1) = \alpha(\mathsf{ct}_0) + \alpha(\mathsf{ct}_1)$$

- More interestingly, $\alpha(c \diamond \vec{\mathsf{ct}}) = c \diamond \alpha(\vec{\mathsf{ct}})$ is also homomorphic

Composable Preprocessing

- For RLWE ctxt $\mathsf{ct} = [a, b]$, define

$$\alpha(\mathsf{ct}) = a, \qquad \beta(\mathsf{ct}) = b$$

- These are homomorphic with respect $\{+, *\}$, e.g.

$$\alpha(\mathsf{ct}_0 + \mathsf{ct}_1) = \alpha(\mathsf{ct}_0) + \alpha(\mathsf{ct}_1)$$

- More interestingly, $\alpha(c \diamond \vec{\mathsf{ct}}) = c \diamond \alpha(\vec{\mathsf{ct}})$ is also homomorphic
  - Precompute $\alpha(\mathsf{Eval}_{\mathsf{ek}}(C, \{\mathsf{ct}_i\}_i))$ for C built from $\{+, *, \diamond\}$ ($2\times$ speedup)

Composable Preprocessing

- For RLWE ctxt $\mathsf{ct} = [a, b]$, define

$$\alpha(\mathsf{ct}) = a, \qquad \beta(\mathsf{ct}) = b$$

- These are homomorphic with respect $\{+, *\}$, e.g.

$$\alpha(\mathsf{ct}_0 + \mathsf{ct}_1) = \alpha(\mathsf{ct}_0) + \alpha(\mathsf{ct}_1)$$

- More interestingly, $\alpha(c \diamond \vec{\mathsf{ct}}) = c \diamond \alpha(\vec{\mathsf{ct}})$ is also homomorphic
  - Precompute $\alpha(\mathsf{Eval}_{\mathsf{ek}}(C, \{\mathsf{ct}_i\}_i))$ for C built from $\{+, *, \diamond\}$ ($2\times$ speedup)
- Hom. rotations only call GadgetDecompose on $\alpha(\mathsf{ct})$

Composable Preprocessing

- For RLWE ctxt $\mathsf{ct} = [a, b]$, define

$$\alpha(\mathsf{ct}) = a, \qquad \beta(\mathsf{ct}) = b$$

- These are homomorphic with respect $\{+, *\}$, e.g.

$$\alpha(\mathsf{ct}_0 + \mathsf{ct}_1) = \alpha(\mathsf{ct}_0) + \alpha(\mathsf{ct}_1)$$

- More interestingly, $\alpha(c \diamond \vec{\mathsf{ct}}) = c \diamond \alpha(\vec{\mathsf{ct}})$ is also homomorphic
  - Precompute $\alpha(\mathsf{Eval}_{\mathsf{ek}}(C, \{\mathsf{ct}_i\}_i))$ for C built from $\{+, *, \diamond\}$ ($2\times$ speedup)
- Hom. rotations only call GadgetDecompose on $\alpha(\mathsf{ct})$
  - Precompute all NTTs ($O(\log n)\times$ speedup)

NTTlessPIR

- Use (standard) Diagonally-Dominant matrix-vector multiplication algorithm

NTTlessPIR

- Use (standard) Diagonally-Dominant matrix-vector multiplication algorithm
  - With client-side CRT interpolation to handle NTT-unfriendly $q \in \{2^{32}, 2^{64}\}$

NTTlessPIR

- Use (standard) Diagonally-Dominant matrix-vector multiplication algorithm
  - With client-side CRT interpolation to handle NTT-unfriendly $q \in \{2^{32}, 2^{64}\}$
- Apply composable preprocessing to preprocess all NTTs that occur

NTTlessPIR

- Use (standard) Diagonally-Dominant matrix-vector multiplication algorithm
  - With client-side CRT interpolation to handle NTT-unfriendly $q \in \{2^{32}, 2^{64}\}$
- Apply composable preprocessing to preprocess all NTTs that occur
- Net Result: Can hom. compute $(A, \mathsf{Enc}_{\vec{s}}(\vec{x})) \mapsto \mathsf{Enc}_{\vec{s}}(A \cdot \vec{x})$ within a multiplicative constant of the cost in plaintext

Composable Preprocessing Limitations

- Doesn't seem to allow hom. multiplication

Composable Preprocessing Limitations

- Doesn't seem to allow hom. multiplication
  - Can non-composably preprocess algorithms still

Composable Preprocessing Limitations

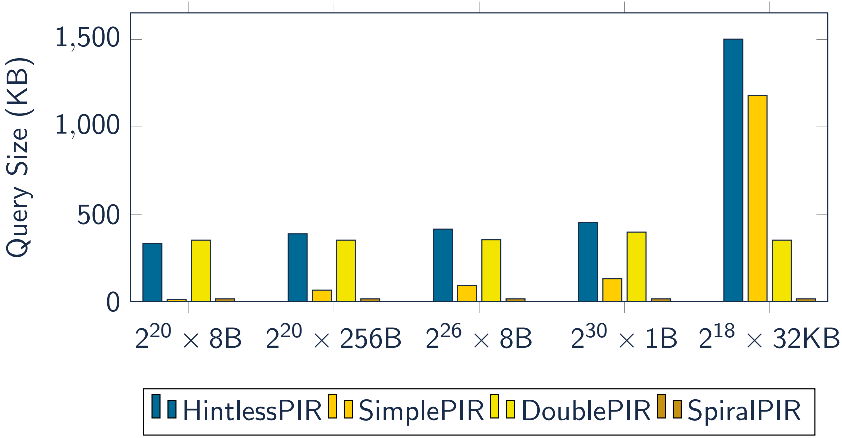
- Doesn't seem to allow hom. multiplication
  - Can non-composably preprocess algorithms still (TensorPIR)

Contents

Evaluation

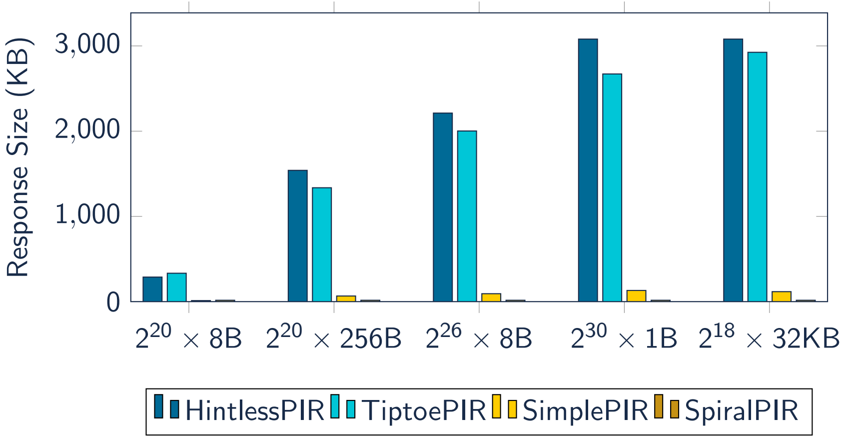
- Implemented our protocols
  - `https://github.com/google/hintless_pir`
- Compared HintlessPIR to other HE-based PIR
  - SimplePIR, DoublePIR, TiptoePIR, SpiralPIR

Bandwidth: Query Size



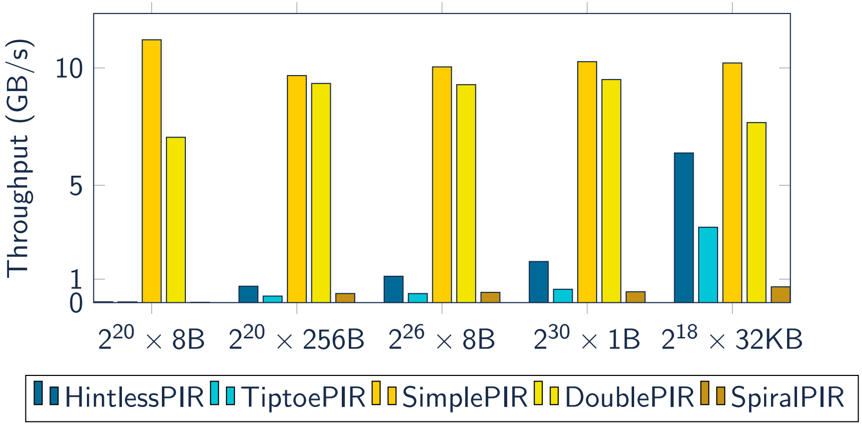
- TiptoePIR: ≈ 23MB for all database sizes

Bandwidth: Response Size



- DoublePIR's response depends only on record size. Ranges from 44KB (1B records) to 1273MB (32KB records)

Throughput


Throughput (GB/s)
10
5
1
0
$2^{20} \times$ 8B
$2^{20} \times$ 256B
$2^{26} \times$ 8B
$2^{30} \times$ 1B
$2^{18} \times$ 32KB
HintlessPIR
TiptoePIR
SimplePIR
DoublePIR
SpiralPIR

Hint Size

Database	SimplePIR	DoublePIR
$2^{20} \times$ 8B(8MB)	200% (16MB)	3025% (242MB)
$2^{20} \times$ 256B(286MB)	34% (92MB)	2573% (6897MB)
$2^{26} \times$ 8B(537MB)	24% (131MB)	45% (242MB)
$2^{30} \times$ 1B(1074MB)	17% (185MB)	2.8% (31MB)
$2^{18} \times$ 32KB(8389MB)	2% (185MB)	10418% (874000MB)