Private information retrieval (PIR): basic building block for privacy-preserving protocols

client (i) $\qquad\qquad$ server $(d_1, \ldots, d_N)$

$\longrightarrow$

$\longleftarrow$

$\downarrow$

d_i

<u>Requirements</u>: client learns the desired database record d_i

server does <u>not</u> learn anything (even if server is malicious!)

↳ We do <u>not</u> require privacy for server's database (<u>not</u> OT)

<u>Trivial PIR</u>: download the full database

<u>Goal</u>: Minimize communication from server to client

Trivial PIR: $O(N)$ communication

<u>Applications</u>: private DNS lookups

safe browsing

private contact tracing

contact discovery

anonymous messaging

<u>Two-server setting</u>: assume database is replicated across two servers (that do not collude)

$S_1 \; (d_1, \ldots, d_N)$ $\qquad\qquad$ $S_2 \; (d_1, \ldots, d_N)$

↳ $y_1 \leftarrow \sum_{j \in [N]} d_j f_1(j)$ $\qquad\qquad$ $y_2 \leftarrow \sum_{j \in [N]} d_j f_2(j)$

f_1 $\qquad$ y_1 $\qquad$ f_2 $\qquad$ y_2

client (i)

$(f_1, f_2) \leftarrow \text{DPF.Share}(1^\lambda, i)$

$\downarrow$

$d_i \leftarrow y_1 \oplus y_2$

<u>Query size</u>: $O(\log N) \cdot \text{poly}(\lambda)$

<u>Response size</u>: $2 \cdot |d_i| = O(|d_i|)$

$\Big\}$ Essentially optimal with respect to N

(in fact, can extend this to nearly optimal rate: $|\text{response}| = |d_i| + \text{poly}(\lambda)$)

$\uparrow$

<u>Concretely</u>: a few extra bits (e.g., 1)

<u>Limitation</u>: server-side work is <u>linear</u> in database size (possible to amortize with preprocessing)

In multi-server setting, we can also obtain information-theoretic constructions with $o(N)$ communication

↳ Question closely related to locally-decodable codes

<u>Single-server setting</u>: database is hosted on a <u>single</u> server

server $(d_1, \ldots, d_N)$

$ct' \leftarrow \text{FHE.Eval}(f_{d_1, \ldots, d_N}, ct)$

ct $\qquad$ ct'

client (i)

$(pk, sk) \leftarrow \text{FHE.Setup}(1^\lambda)$

$ct \leftarrow \text{FHE.Encrypt}(i)$

$$f_{d_1, \ldots, d_N}(i) = \sum_{j \in [N]} d_j \cdot \mathbb{1}\{i = j\}$$

<u>Query size</u>: $O(\log N) \cdot \text{poly}(\lambda)$

<u>Response size</u>: $|d_i| \cdot \text{poly}(\lambda, \log N)$

↳ Can remove $\log N$ dependence with bootstrapping

FHE-based approach described above is not likely to have good <u>concrete</u> efficiency

↳ Requires $O(\log N)$ multiplications for each database item

↳ Typically $N \sim 2^{20}$ (or 2^{30}) so multiplicative depth is high → poor concrete efficiency

Concretely-efficient PIR schemes follow Kushilevitz-Ostrovsky framework:

$$\begin{bmatrix} d_{11} & \cdots & d_{1\ell} \\ \vdots & \ddots & \vdots \\ d_{\ell 1} & \cdots & d_{\ell\ell} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \longrightarrow \begin{bmatrix} d_{1i} \\ \vdots \\ d_{\ell i} \end{bmatrix}$$

← row i (containing desired record)

$\ell = \sqrt{N}$

Database available in the clear

Query encrypted under FHE

Encryption of column i of database

Query + response size $\sim O(\sqrt{N})$

↳ Can decrease either by recursive composition or by homomorphic multiplication → basis of concretely-efficient constructions

Open question: Further reduce compute via database encoding?

↳ Server computation is still linear

(Recent work shows how to get to $\sqrt{N}$ amortized with preprocessing)

Improving the concrete efficiency of lattice-based schemes

Standard Regev encryption: $\quad pk = A = \begin{bmatrix} \bar{A} \\ \bar{s}^T \bar{A} + e^T \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$

large public key ~ typically $O(n^2 \log q)$ – quadratic in lattice dimension

$$sk = s^T = [-\bar{s}^T \mid 1] \in \mathbb{Z}_q^n$$

$$ct = Ar + \begin{bmatrix} 0^{n-1} \\ \mu \cdot \lfloor \frac{q}{p} \rceil \end{bmatrix} \in \mathbb{Z}_q^n$$

need a vector of dimension n to encrypt a single scalar (high overhead)

To improve efficiency, we can instead work over polynomial rings

$\mathbb{Z}[x]$: ring of polynomials with integer coefficients

$\mathbb{Z}[x]/(x^{2^d}+1)$: ring of polynomials modulo $x^{2^d}+1$ (cyclotomic polynomial)

In particular $x^{2^d} = -1 \pmod{x^{2^d}+1}$

We can view LWE as working over the ring $R = \mathbb{Z}$. Now, we consider the ring $R = \mathbb{Z}[x]/(x^{2^d}+1)$.

RLWE assumption: Sample $a \xleftarrow{R} R_q$ $\quad (R_q = R/qR = \mathbb{Z}_q[x]/(x^{2^d}+1))$

$s \xleftarrow{R} R_q$

$e \leftarrow \chi$ $\quad$ (distribution over R_q where polynomials have small coefficients) ← analog of discrete Gaussian

$u \xleftarrow{R} R_q$

The RLWE assumption says that following distributions are computationally indistinguishable:

$$(a,\ sa+e) \quad \text{and} \quad (a,\ u)$$

We can view ring multiplication as a matrix-vector multiplications

$a = 2x^3 + x^2 - 3x + 1$

$s = x^3 - 2x + 2$

$R = \mathbb{Z}[x]/(x^4+1)$

$(2x^3+x^2-3x+1)(x^3-2x+2)$

first row is uniform $\quad$ secret is uniform

$$\begin{bmatrix} 1 & -3 & 1 & 2 \\ -2 & 1 & -3 & 1 \\ -1 & -2 & 1 & -3 \\ 3 & -1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -2 \\ 2 \end{bmatrix} = \begin{array}{c} \begin{bmatrix} 3 \\ 6 \\ -9 \\ 9 \end{bmatrix} \end{array} \begin{array}{l} x^3 \\ x^2 \\ x \\ 1 \end{array}$$

LWE with a <u>structured</u> matrix A

can compute $a \cdot s$ in $O(d \log d)$ time using FFT (for suitable q) } faster than matrix-vector multiplication in LWE

Regev encryption over rings: $pk = (a, b)$ where

$$a \xleftarrow{R} R_q, \quad s \xleftarrow{R} R_q, \quad e \leftarrow \chi \quad \rightsquigarrow \quad b = sa + e$$

$$sk = s$$

$$ct = (ar,\ br + \mu \cdot \lfloor \tfrac{q}{p} \rceil) \quad \text{where } r \leftarrow \chi \text{ and } \mu \in R_q$$

Advantages over vanilla Regev: Shorter public keys, faster key-generation

Better ciphertext rate: need 2 R_q elements to encrypt 1 R_p element

Drawback: More structured assumption (ring multiplication is commutative!)

↳ Does not have reductions to worst case lattice problems

Most constructions based on standard lattices can be translated directly to the ring setting (with better concrete efficiency)

Exploiting structure in the ring setting:

Suppose we work over the ring $R = \mathbb{Z}[x]/(x^{2^d}+1)$ and suppose we chose the plaintext modulus $p = 1 \pmod{2^{d+1}}$.

Then, we can show that the polynomial $x^{2^d}+1$ factors mod p as

$$x^{2^d} + 1 = \prod_{i \in [2^d]} (x - \alpha_i) \pmod{p}$$

for $\alpha_1, \ldots, \alpha_{2^d} \in \mathbb{Z}_p$. Then, by Chinese Remainder Theorem (CRT):

$$R/pR = \mathbb{Z}_p[x]/(x^{2^d}+1) \cong \mathbb{Z}_p[x]/(x-\alpha_1) \times \cdots \times \mathbb{Z}_p[x]/(x-\alpha_{2^d})$$
$$\cong \mathbb{Z}_p^{2^d}$$

Plaintext space is isomorphic to $\mathbb{Z}_p^{2^d}$

↳ Addition in R_p corresponds to component-wise addition in $\mathbb{Z}_p^{2^d}$

↳ Multiplication in R_p corresponds to component-wise multiplication in $\mathbb{Z}_p^{2^d}$

} SIMD (single instruction multiple data) support for homomorphic evaluation

We can encrypt a vector of 2^d integers

↳ Each homomorphic operation simultaneously computes on all 2^d elements

Reducing ciphertext size via modulus switching:

When we use FHE to evaluate a circuit C, parameters have to be chosen so that accumulated error is smaller than $q/2p$.

Useful technique: Perform all computations with respect to a modulus q and then rescale final ciphertext to a smaller modulus $q' < q$:

$$s^T c = \lfloor \tfrac{q}{p} \cdot \mu \rceil + e \pmod{q}$$

Replace $c \mapsto \lfloor \frac{q'}{q} \cdot c \rceil$ (interpret c' as element of $\mathbb{Z}_{q'}$)

To analyze this, we consider an expression over the rationals:

$$s^T c = \lfloor \tfrac{q}{p} \cdot \mu \rceil + e + kq \quad \text{for some integer } q$$

We can write

$$c' = \lfloor \tfrac{q'}{q} \cdot c \rceil = \tfrac{q'}{q} \cdot c + e' \quad \text{where } \|e\| < \tfrac{1}{2} \text{ (over the rationals)}$$

Then,

$$s^T c' = \frac{q'}{q} \cdot s^T c + s^T e'$$
$$= \frac{q'}{q} \left[\left(\frac{q}{p} \cdot \mu + e'' \right) + e + kq \right] + s^T e'$$

$$= \frac{q'}{p}\mu + \frac{q'}{q}(e'' + e) + kq' + s^T e'$$

$$= \frac{q'}{p}\mu + \underbrace{\frac{q'}{q}(e'' + e) + s^T e'}_{\text{require that these}} \pmod{q'}$$

will need that secret key components are small (i.e., sampled from error distribution)

components are smaller than $q'/2p$ [observe original error e is scaled down by q'/q]

Take-away: After performing operations, can rescale the ciphertexts to smaller modulus q' (reduces communication concretely)