# Private Information Retrieval

Linear homomorphism turns out to be a useful property for building cryptosystems that can do operations on encrypted data

Today, we will be looking at querying remote databases without revealing what we are searching for

- Suppose a server holds an $N$ bit database $D \in \mathbb{Z}_2^N$
- A user has an index $i$ and wants to discover $D_i$ without revealing $i$ to the server

**Private Information Retrieval Scheme**

A PIR scheme is a triple of randomized algorithms `(Query, Answer, Reconstruct)` that meet the following properties, where $N$ is the length of the database:

- **Correctness**: for any index $i$, database $D$, and security parameter $n$, the probability of `Reconstruct(Answer(D, qu), st)` is at least $1 - \mathrm{negl}(n)$ where `(qu, st)` are given by `Query(n, i)`
- **Security**: for any two indices $i, j$, the queries corresponding to $i$ and $j$ are are computationally indistinguishable
- **Succintness**: The total bit length of the client's query and the server's answer is less than $N$
    - That is, the total number of bits communicated is smaller than $N$
    - This is to rule out trivial schemes where the user downloads the whole database

## Square-Root PIR

Key idea is that the server will represent its $N$ bit database as a matrix $D \in \mathbb{Z}_2^{\sqrt{N} \times \sqrt{N}}$

- The user wants to read bits at 2D locations $(i, j)$
- Assume we are given a linearly-homomorphic encryption scheme `(Gen, Enc, Dec)`

**Protocol**:

- $\mathrm{Query}(n, (i, j))$:
    - Build the unit vector $u_j$ that consists of all 0s except for a single 1 at position $j$
    - Sample a secret $\mathrm{sk}$ with $\mathrm{Gen}(n)$
    - Output the query vector $\mathrm{qu} \leftarrow \mathrm{Enc}(\mathrm{sk}, u_j)$ along with client-held state $\mathrm{st} \leftarrow (\mathrm{sk}, i)$
- $\mathrm{Answer}(D, \mathrm{qu})$:
    - Output the answer vector $\mathrm{ans} \leftarrow D \cdot \mathrm{qu}$
- $\mathrm{Reconstruct}(\mathrm{ans}, \mathrm{st})$:
    - Decrypt the answer vector as $v \leftarrow \mathrm{Dec}(\mathrm{sk}, \mathrm{ans})$
    - Parse the client-held state $(\mathrm{sk}, i)$
    - Output the $i$ th entry of $v$

Note that `Query` will run on the client, not the server

- The server is only in charge of addressing `Answer`

**Correctness**:

- If the encryption scheme is linearly homomorphic, we have that $\mathrm{ans}$ is:
    - $D \cdot \mathrm{Query}(n, (i, j)) = D \cdot \mathrm{enc}(\mathrm{sk}, u_j) = \mathrm{Enc}(\mathrm{sk}, D \cdot u_j)$
    - Decoding and getting the $i$ th bit will then get $D_{ij}$

**Security**:

- Follows from the CPA-security of the encryption scheme

**Succintness**:

- `Query` consists of $\sqrt{N}$ ciphertexts, as does the output of `Answer`

## Naive Implementation

We attempt to implement with the secret key encryption scheme from LWE

- `Query`:
  - We build the unit vector $u_j$ as described above
  - We sample:
    - A random matrix $A \in \mathbb{Z}_q^{\sqrt{N} \times n}$
    - A secret key vector $s \in \mathbb{Z}_q^n$
    - A random error vector $e \in \chi^{\sqrt{N}}$
  - Output:
    - Query: $(A, A \cdot a + e + \lfloor q/2 \rfloor \cdot u_j)$
    - Client-held state: $(s, i)$
- `Answer`:
  - Parse the query as $(A, b)$
  - Output the answer $(D \cdot A, D \cdot b)$
- `Reconstruct`:
  - Parse the client-held state as $(s, i)$
  - Parse the answer as $(H, c)$
  - Compute $v = c - H \cdot s$
  - Round the $i$ th entry of $v$ to the nearest multiple of $\lfloor q/2 \rfloor$ and divide by $\lfloor q/2 \rfloor$

In practice we use:

- $n \approx 1024$
- $q \approx 2^{16}$
- Total communication cost would be $10^4 \cdot \sqrt{N}$ bits

## Optimizations

- We can use the same $A$ matrix as long as we use a random $s$ and $e$ each time
  - We can even use $A$ as a public parameter known to all users
  - The $A$ matrix now no longer needs to be sent around
- The server can precompute $D \cdot A$
- The user can prefetch the value of $D \cdot A$

These make the communication size independent of $n$

- The PIR scheme now uses $\log q \cdot \sqrt{N}$ bits, which is around $16\sqrt{N}$ bits