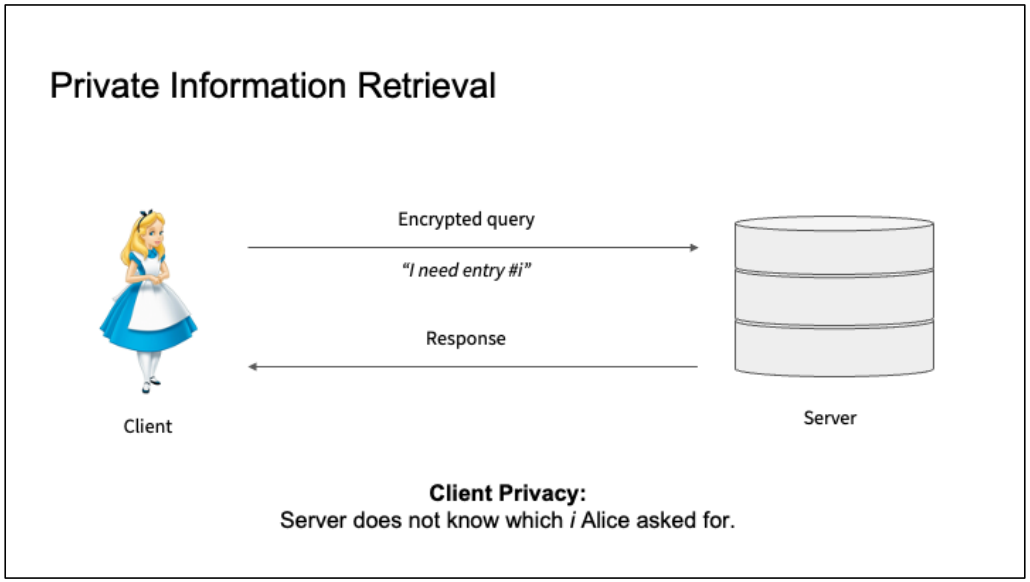
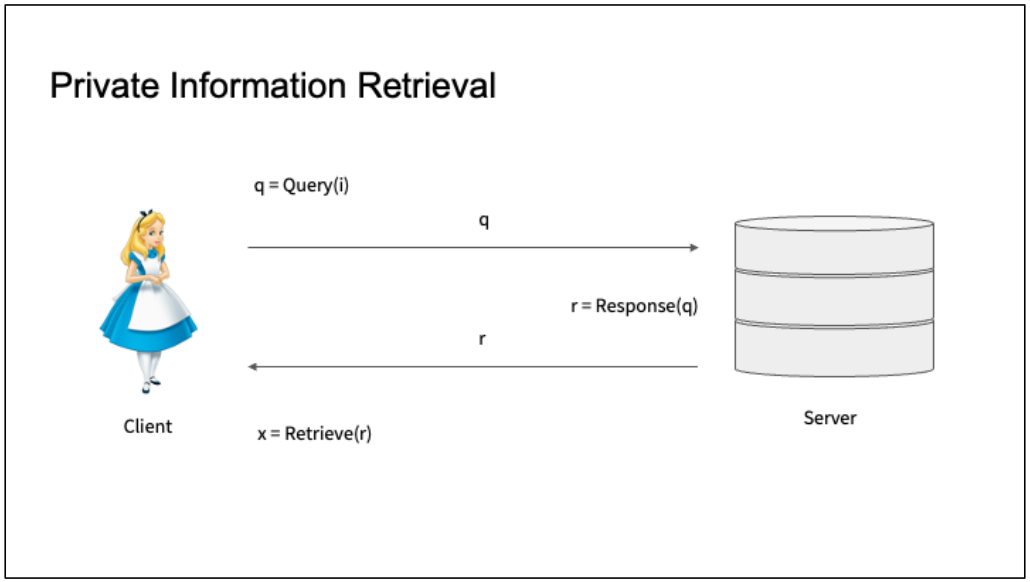
Friday Live Exercises

Homomorphic Encryption

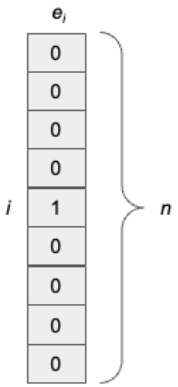


Notice how the setup is a bit different than what seen in class. Here the server has data, however, this data can be public (PGP keys for example).



Trivial solution - send the full dataset. This satisfies perfect client privacy, but obviously not great for utility.

Attempt I



How to use FHE and the indicator vector e_i to implement the protocol?

q = Query(i)

r = Response(q)

x = Retrieve(r)

In fact, first we need to specify what kind of FHE scheme we are dealing with. Let us assume the following scheme:

- Key generation procedure: sk, pk = KeyGen()
- Encryption algorithm that supports bit messages: c = Enc(pk, m), where m is a bit.
- The homomorphic evaluation supports mixed ciphertext-plaintext operations c_3 = Add(c_1, c_2) and c_3 = Mult(c_1, c_2), where c_1, c_2 can be either a ciphertext or plaintext.
- Decryption algorithm: m = Dec(sk, c)

Next, we need to figure out which function we want to evaluate on private inputs. Ideally, we want a function f(i) that takes the requested index i, performs some computations on the database, and returns the corresponding entry. We want to be able to compute this function only using Additions and Multiplications, which are the supported homomorphic operations in our scheme.

This exact signature, f(i), might be hard to implement, but using the indicator vector e_i is actually simpler. Assume that the database is a n x m matrix X. A record X_i is a row in this matrix. Then, to obtain the requested row X_i we need to compute for each column j = 1, …, m:

1. $t_i = e_i \times X_{*j}$, where $v_1 \times v_2$ is entry-wise multiplication
2. $r_j = t_1 + t_2 + \ldots + t\_m$, where $v_1 + v_2$ is entry-wise addition

As a result, we get a vector r = (r_1, r_2, …, r_m) corresponding to the requested row.

```
res_j := 0 for j = 1..m
for i in 1..n:
  for j in 1..m:
     r_ij = e_i * X_ij
    res_j += r_ij
Output res = (res_j)_{j=1}^m
```

This function is easy to represent using entry-wise Additions and Multiplications, therefore can be easily evaluated homomorphically.

The entire protocol is:

1. Setup: Alice generates a keypair sk, pk = KeyGen()
2. Query(i): q = Enc(pk, e_i) entrywise.
3. Response(q): Homomorphically compute the function above.
4. Retrieve(r): x = Dec(sk, r) entrywise.

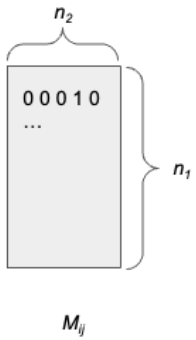
Other versions are possible. In particular, if the HE scheme supports batching — encrypting multiple bits into a single ciphertext — this would make for a more efficient scheme. The evaluated function would be different as we would work with vectors not single elements.

Attempt I. Analysis

1. What is the threat model (e.g., honest, honest-but-curious, malicious server)
2. What is the computation and communication cost of the system in terms of n (database size) and m (record size)?
3. What is the multiplicative depth?

1. We are secure against an honest-but-curious server. A malicious server could give wrong responses or deny the service.
2. Computation: We did n * m homomorphic multiplications and (n-1) * m homomorphic additions.
   Communication: n ciphertexts for the query + m ciphertexts for the response.
   Note that communication is linear in the size of the database. This is not good, as it is not far from the trivial PIR solution: send the whole database.
3. The depth is 1.

Attempt II. Sublinear communication



How to use FHE and the indicator matrix M_{ij} to implement the protocol?

q = Query(i)

r = Response(q)

x = Retrieve(r)

We use the same FHE scheme as before.

We can use a different function to save some communication. First, we reshape the data matrix X into a three-dimensional tensor [n_1, n_2, m], where n_1 and n_2 are such that n_1 x n_2 = n, e.g., sqrt(n). Then we can get the requested record using two index vectors e_i and e_j of size n_1 and n_2 respectively: Multiply e_i by rows, and then sum them up. The result is a matrix n_2 x m matrix T. Finally, multiple e_j by T and sum again.

Attempt II. Analysis

1. What is the threat model (e.g., honest, honest-but-curious, malicious server)
2. What is the computation and communication cost of the system in terms of *n* (database size) and *m* (record size)?
3. What is the multiplicative depth?

1. Same as before
2. Computation: Need $n_1 \times n_2 \times m = n \times m$ multiplications and $n_2 \times m$ additions for the first part ($e_i \times \ldots$). Then, $n_2 \times m$ multiplications, $n_2 \times m$ additions for the second ($e_j \times T$). Communication: $n_1 + n_2$ query ciphertexts, and m response ciphertexts.
3. The depth is 2.

Server privacy

Do the protocols achieve server privacy? (The client only learns information about the requested record)

1. Honest-but-curious client
2. Malicious client

A malicious client can misbehave and send an array of 1’s for example.