A Fast Single Server Private Information Retrieval Protocol with Low Communication Cost

Changyu Dong, University of Strathclyde

Liqun Chen, HP Labs

changyu.dong@strath.ac.uk, *liqun.chen@hp.com*

Single Server Private Information Retrieval

- Single server: a server hosts an n-bit database $\mathbf{x} = \mathbf{x}_1\mathbf{x}_2...\mathbf{x}_n$
- Information retrieval: a client wants to retrieve the ith bit in the database
- Private: the server knows nothing about i
- Correctness: the client knows $\mathbf{x}_i$
  - The client is allowed to retrieve more than just $\mathbf{x}_i$ from the server's database

A Trivial Solution

- The server sends the whole database to the client.
  - No computation needed
  - But consumes a lot of bandwidth.
- A non-trivial solution must have less communication overhead than the trivial one
  - Bandwidth consumption $<$ n-bit
  - A long line of research started in 1990s.
  - Most previous work focused only on how to reduce the communication cost

An Efficiency Paradox

- In 2007 Sion et al pointed out that all non-trivial single server PIR protocols were actually slower than the trivial one.
  - Less communication, more computation
  - For each bit, the computation time is much more than simply transmitting the bit through network
- The trivial solution can beat the fastest protocol in a network with merely 300 kbps bandwidth.
- Computation efficiency is also key to make single server PIR practical.

Our Protocol: How to Retrieve the Bit?

- The server organises the database into a vector of bit blocks.
- $\mathbf{x}_j$ is in the jth row.
- The client creates a query $\mathbf{q}$ as a bit vector, all bits are 0 except the jth bit.
- The client sends $\mathbf{q}$ to the server
- The server multiplies $\mathbf{q}$ with X, the result is the jth row and sends it to the client.
- The client checks the answer and finds the bit.

$$\mathbf{q}\cdot X = \begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}\cdot\begin{bmatrix}101\\111\\001\\010\end{bmatrix} = 0\cdot 101+0\cdot 111+1\cdot 001+0\cdot 010 = \begin{bmatrix}001\end{bmatrix}$$

Our Protocol: How to Make It Private?

- The client uses a fully homomorphic encryption to encryption the query string.
- An FHE scheme allows addition and multiplication to be performed on encrypted data.
  - $E_{pk}(m_1) + E_{pk}(m_2) = E_{pk}(m_1 + m_2)$
  - $E_{pk}(m_1) + m_2 = E_{pk}(m_1 + m_2)$
  - $E_{pk}(m_1) \cdot E_{pk}(m_2) = E_{pk}(m_1 \cdot m_2)$
  - $E_{pk}(m_1) \cdot m_2 = E_{pk}(m_1 \cdot m_2)$

Our Protocol: How to Make It Private?

- The server can retrieve the bit homomorphically

$$\begin{bmatrix} E_{pk}(0) & E_{pk}(0) & E_{pk}(1) & E_{pk}(0) \end{bmatrix} \cdot \begin{bmatrix} 101 \\ 111 \\ 001 \\ 010 \end{bmatrix}$$

$$\begin{aligned} &= E_{pk}(0) \cdot 101 + E_{pk}(0) \cdot 111 + E_{pk}(1) \cdot 001 + E_{pk}(0) \cdot 010 \\ &= E_{pk}(0 \cdot 101) + E_{pk}(0 \cdot 111) + E_{pk}(1 \cdot 001) + E_{pk}(0 \cdot 010) \\ &= E_{pk}(0 \cdot 101 + 0 \cdot 111 + 1 \cdot 001 + 0 \cdot 010) \\ &= E_{pk}(001) \end{aligned}$$

Wait a Minute...

- The protocol is correct and private, but not efficient!
  - Communication wise, the cost can be higher than sending the whole database.
  - Computation wise, FHE schemes are slow.

Our Protocol: How to Make It Efficient

- Two ideas:
  - We compress the query string before sending it to the server.
  - We do homomorphic operations on thousands of bits in the database simultaneously.

Reduce communication cost

- The client first compresses **q** then encrypts it.
  - Requirements: good compression ratio and a simple decompression algorithm.
- The intuition: to fold the string recursively.
- A n-bit string can be folded into $2 \log n$ bit.
- Unfold is also recursive and works well with FHE.
- A tree structure is used to make the process deterministic.

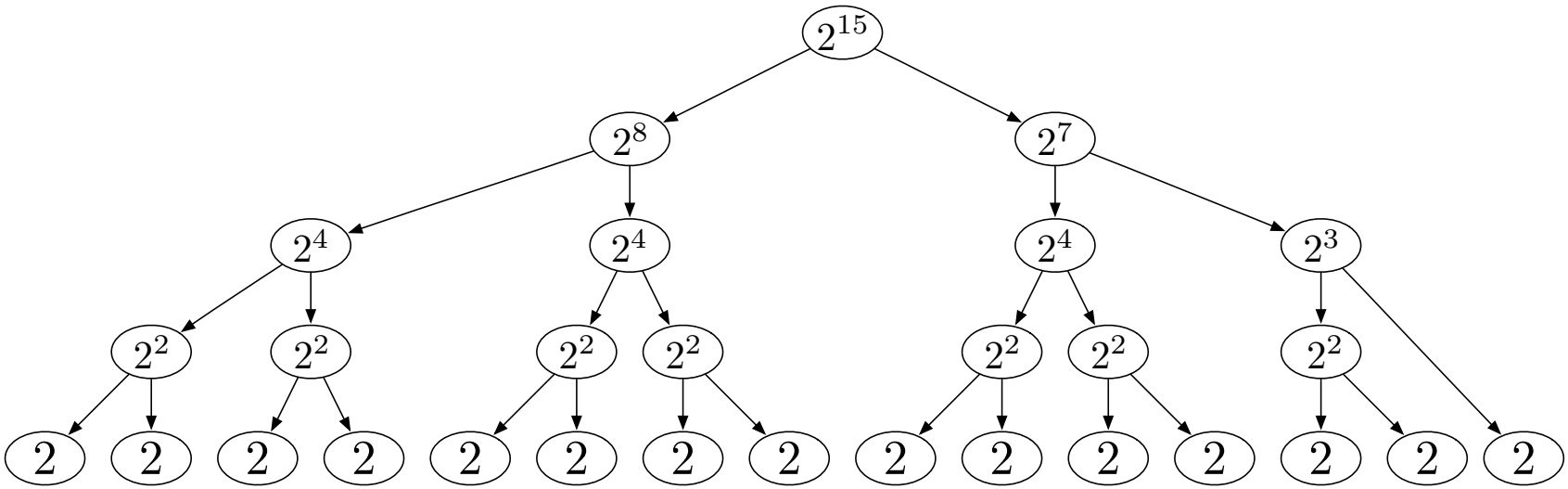
	0	1	0	0
0	0	0	0	0
0	0	0	0	0
1	0	1	0	0
0	0	0	0	0

q = 00000000 01000000

u = 0010

v = 0100

A Folding Tree



Improve Efficiency: SIMD

- Use the SIMD feature of the BGV FHE scheme
  - Single Instruction Multiple Data
- Encrypt a vector of plaintexts rather than a single one
- Homomorphic operations are now component-wise

$$
\begin{aligned}
E_{pk}([m_1 \quad \dots \quad m_l]) + E_{pk}([m'_1 \quad \dots \quad m'_l]) &= E_{pk}([m_1 + m'_1 \quad \dots \quad m_l + m'_l]) \\
E_{pk}([m_1 \quad \dots \quad m_l]) + [m'_1 \quad \dots \quad m'_l] &= E_{pk}([m_1 + m'_1 \quad \dots \quad m_l + m'_l]) \\
E_{pk}([m_1 \quad \dots \quad m_l]) \cdot E_{pk}([m'_1 \quad \dots \quad m'_l]) &= E_{pk}([m_1 \cdot m'_1 \quad \dots \quad m_l \cdot m'_l]) \\
E_{pk}([m_1 \quad \dots \quad m_l]) \cdot [m'_1 \quad \dots \quad m'_l] &= E_{pk}([m_1 \cdot m'_1 \quad \dots \quad m_l \cdot m'_l])
\end{aligned}
$$

- The vector encrypted can be rotated (circular shift)

$$E_{pk}([m_1 \quad \dots \quad m_l]) << i = E_{pk}([m_{i+1} \quad \dots \quad m_l \quad m_1 \quad \dots \quad m_i])$$

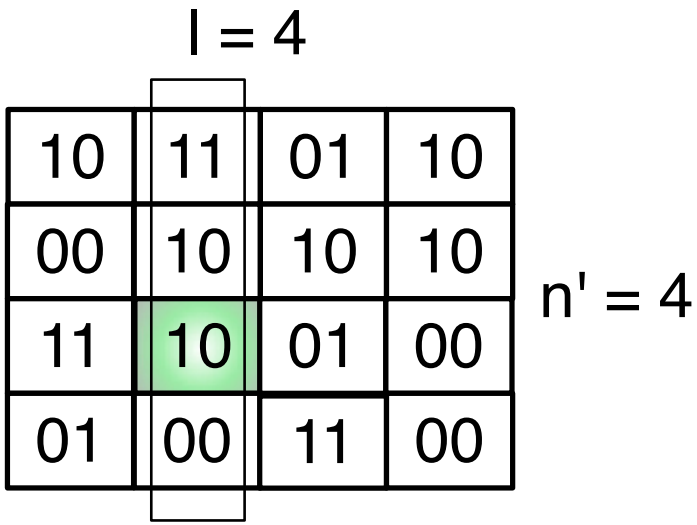
Further Reduce the Bandwidth Consumption

- After folding the query string, the compressed query string can be encrypted as a single ciphertext using the SIMD feature.
- In stead of $2 \log n$ ciphertexts, now the client only needs to send one ciphertext.
- This only adds a very small overhead
  - The client needs to rotate the compressed query string before encrytion.
  - The server needs to rotate the ciphertext before unfolding.
  - The rotation are necessary to unfold the query string.

Database Representation

- The database is now an $n' \times l$ matrix, each element is a d-bit block
- Each column corresponds to a plaintext slot in the SIMD ciphertext.

$$\mathbf{x} = 10110110001010101110010001001100$$



The Client Side

- The client creates a query string **q**.
  - **q** contains the information about which row to retrieve.
  - If the block to be retrieved is in the jth row, then the jth bit in **q** is 1.
- The client folds the query string into **s**.
- The client rotates the compressed query string **s** and obtains **s'**.
  - The rotation adds information about which column to retrieve.
  - If the block to be retrieved is in the kth column, then the first bit in **s** becomes the kth bit in **s'**.
- The client encrypts **s'** and sends the ciphertext to the server.

To retrieve the block at row 3, col 2

$\mathbf{q} = 0010$

fold

$\mathbf{s} = 0110$

rotate

$\mathbf{s}' = 0011$

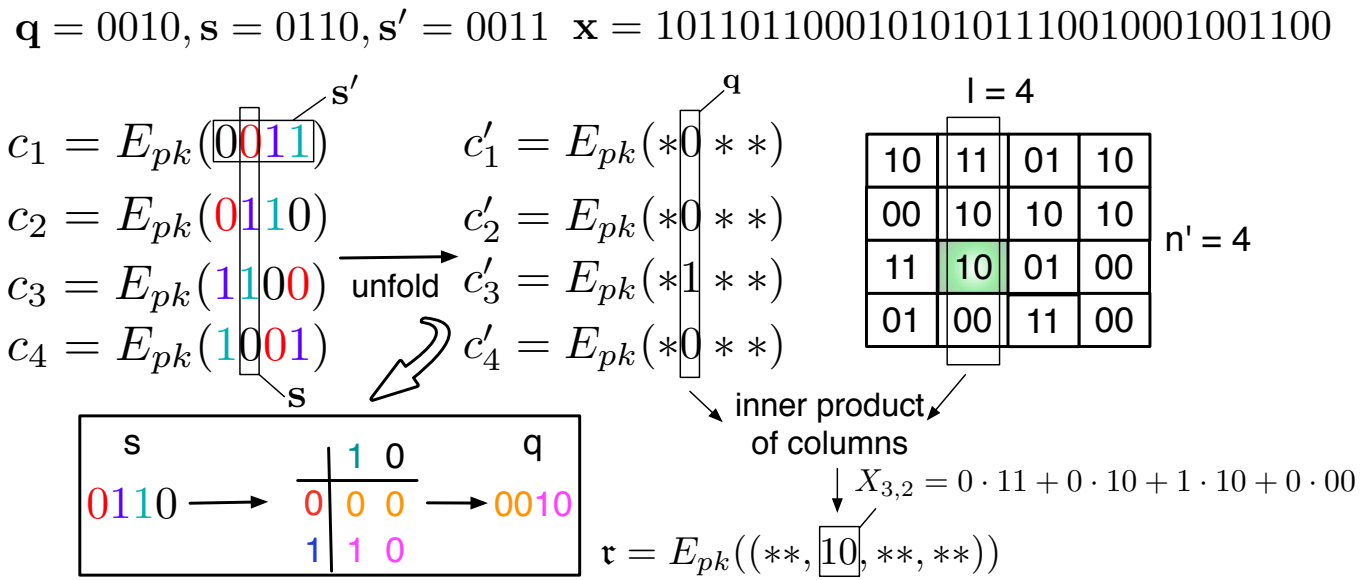
l = 4

10	11	01	10
00	10	10	10
11	10	01	00
01	00	11	00

n' = 4

The Server Side

- The server first rotates the ciphertext, to get $2 \log n'$ ciphertexts
- then unfolds it,
- then computes the answer
- Essentially, it runs in parallel l instances of the protocol we mentioned earlier.



Efficiency

- Communication wise, 1 ciphertext + 1 ciphertext (query and reply)
- Server side computation, $< 2n' + 3\sqrt{n'}$ homomorphic multiplications
  - $n' = \frac{n}{l \cdot d}$, e.g. $n' = \frac{n}{8190}$ in our experiment.
  - $2n'$ "raw multiplications" – efficient
  - $< 3\sqrt{n'}$ "full multiplications" – less efficent
  - If convert to 64-bit modular multiplications, the upper-bound is $12n$ for sufficient large n. (128-bit security)
  - Per bit cost is significantly lower than previous best protocol 12n 64-bit mul vs n 3072-bit mul
- Client side computation: 1 encryption and 1 decryption.

Experiments

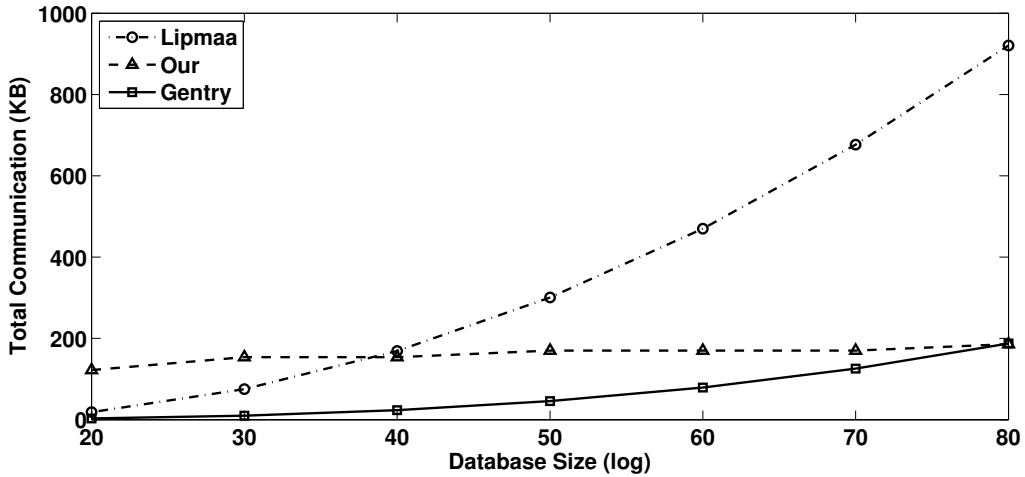
- Prototype in C++, using HElib
- Security parameter 128-bit security
- Database size 2^{25} bits (4 MB), 2^{30} bits (128 MB), 2^{35} bits (4 GB)
- On a MacBook Pro laptop, 2.2 GHz quad-core CPU, 16 GB RAM

Performance Comparison

- The most computationally efficient PIR protocol is by Kushilevitz, which needs n 3072-bit mod multiplications

DB size (bit)	2^{25}	2^{30}	2^{35}
Kushilevitz Time (s)	277.46	8878.72	284,119.04
Improvement	12.5 $\times$	49.2 $\times$	90.5 $\times$

- Communication wise, our protocol is less efficient than the previous best one, better than the second best one
  - Asymptotically, our protocol is better, but due to the large ciphertext size in the BGV scheme, it is less efficient unless the database is impractically large.
  - But the difference is small, only less than 200 KB



Conclusion

- A single server PIR protocol based on the BGV FHE
  - It is fast
  - It has low communication cost
  - FHE-based protocols can be practical in certain cases.
- Future work
  - Further improve performance of PIR, e.g. by amortizing costs over multiple-queries.
  - Other protocols suitable for FHE?