CS670: Assignment 1 Solutions

Prashant Kumar, 231110036

March 5, 2024

Solution-1:

Assumption and Notations:

Given that there are 2 servers S_0 and S_1 and C clients having a string $\alpha_i \in \{0,1\}^n$ and the servers want to learn the number of clients who hold a certain string σ while maintaining the privacy of each client's string.

1. Protocol steps :

To implement the protocol I use the concept of the Distributed Point Functions such that each client generates the DPF keys that will help the server to learn about the σ privately.

Client side:

- At first, Each client C_i generates the DPF keys (k_{i0}, k_{i1}) with the following properties

$$(k_{i0}, k_{i1}) \leftarrow Gen(\lambda, \alpha_i, 1)$$

$$\begin{aligned} &\text{eval}(s, k_{i0}) + \text{eval}(s, k_{i1}) = 1, \text{ iff } s = \alpha_i; \\ &\text{otherwise, } \text{eval}(s, k_{i0}) + \text{eval}(s, k_{i1}) = 0. \end{aligned}$$

- Now each client C_i sends their generated DPF keys (k_{i0}, k_{i1}) to server S_0 and S_1 respectively.

Server side:

- Server S_0 receives k_{i0} of each client C_i then it calculates $\text{eval}(\sigma, k_{i0})$ for each received k_{i0} and stores the addition of all results, so essentially server S_0 is having $eval(\sigma, k_{10}) + eval(\sigma, k_{20}).... + eval(\sigma, k_{C0})$ or

$$\sum_{i=1}^{C} (\text{eval}(\sigma, k_{i0}))$$

- Similarly, Server S_1 receives k_{i1} of each client C_i then it calculates $\text{eval}(\sigma, k_{i1})$ for each received k_{i1} and stores the addition of all results, so again the server S_1 is having $eval(\sigma, k_{10}) + eval(\sigma, k_{20}).... + eval(\sigma, k_{C0})$ or

$$\sum_{i=1}^{C} (\text{eval}(\sigma, k_{i1}))$$

Server Computation:

- At last, both servers will take the addition of their stored intermediate results which will result in the count of clients having string σ.

- So, Server S_0 and S_1 collaboratively calculates:

$$\text{count} = \sum_{i=1}^{C} (\text{eval}(\sigma, k_{i0})) + \sum_{i=1}^{C} (\text{eval}(\sigma, k_{i1}))$$

$$\text{count} = \sum_{i=1}^{C} (\text{eval}(\sigma, k_{i0})) + \text{eval}(\sigma, k_{i1}))$$

- Now the value obtained in the count variable is the required result which is the number of clients having string σ.

2. Proof of correctness: 1. DPF Properties:
The DPF keys (k_{i0}, k_{i1}) generated by each client C_i have the property:

$$\text{eval}(s, k_{i0}) + \text{eval}(s, k_{i1}) = 1 \quad \text{iff} \quad s = \alpha_i$$

$$\text{eval}(s, k_{i0}) + \text{eval}(s, k_{i1}) = 0 \quad \text{otherwise}$$

This property ensures that the DPF keys are valid only for the specific client's string α_i so by this property keys will add to 1 only if $\alpha_i and \sigma$ are the same.

Now assume that **count** is the incorrect count of σ learned by the server. This can only be the case when either $eval(s, k_{i0}) + eval(s, k_{i1}) = 1$ even if $\alpha_i \neq s$ or $eval(s, k_{i0}) + eval(s, k_{i1}) = 0$ when $\alpha_i = s$, but both of these cases are contradictory without assumption of DPF so these cases can never occur and if these cases never occur then count will always have correct value so this essentially means that this protocol is always correct.

Solution-2:

Malicious Database Holder

The honest database holder (owner) can use the concept of digital certificates to prevent the malicious database holder from altering contents, following are the protocol steps he can perform to do this-

1. **Pre-Processing:**

   - Database owner purchases a digital certificate issued by a trusted Certificate Authority which includes a public key linked to his identity, Note that this key is publicly known to everyone.
     P_b: Public key of the honest database holder
   - The owner generates a private key associated with his identity, this will be used in signing any document/data.
     P_r: Private key of the honest database holder

2. **Outsourcing the data:**

   - Now the owner will sign the contents of the database (with some publicly known padding in the contents) before outsourcing data for storage to an untrusted party, note that this step can be done on blocks of the database.
     $Sign(P_r, B)$: Signing block B of database before outsourcing
   - owner outsources the signed data to untrusted third-party storage.

3. **Client Request:**

   - Now a client can fetch the data from this untrusted database and verify the data by the publicly known key of the honest owner, this will give the actual data that can be verified by a publicly known padding.
     $Verify(P_b, Sign, B)$: Verification of signature of obtained data

4. **Verification of response:**

   - The third party can not modify the contents of the database because he can not sign with the private key of the honest owner and if he modifies the contents without signing with the private key then the padding in the actual content will change and he will get caught.

A Two Server PIR protocol

1. A simple Protocol for PIR:
Assumptions:

- Two servers S_0 and S_1 hold replicas of the database and Servers do not communicate with each other.
- Servers holding databases are trusted.

Protocol Steps:
i) Client Request: The client generates two random XOR shares r_0 and r_1, where $r_0 \oplus r_1 = e_i$, and then sends r_0 and r_1 to the servers S_0 and S_1.

ii) Server Response: Each server computes the response of the received query vector r_0 and r_1 with the database contents b and responds with the result $< r_0.b >$ and $< r_1.b >$

iii) Computing result: The client receives the responses from both servers and performs XOR on responses to obtain the i-th index of the database.

$$< r_0.b > \oplus < r_1.b >= b_i$$

Explanation:
The protocol is **correct** because it obtains the i-th index because of the linear property of XOR shares which states that if $r_0 \oplus r_1 = e_i$ then $< r_0.b > \oplus < r_1.b >=< b.e_i >= b_i$ Also this can be easily proven by taking b as a common from both the responses and protocol **ensures privacy** because of the assumptions that both the servers do not communicate with each other so the query vector seems random to each server.

2. Working of this protocol with malicious server:

No, the aforementioned protocol would not be correct in the presence of a malicious server that manipulates the database.

Proof: Suppose server S_1 is malicious and it modifies the contents of database to b' from b.
Now when the protocol executes the new responses of servers will be $< r_0.b >$ and $< r_1.b' >$ and the corresponding client calculation for the target index will be $< r_0.b > \oplus < r_1.b' >$.
Again assume, $< r_0.b > \oplus < r_1.b' >= b_i$
then $< r_0.b > \oplus < r_1.b' >=< r_0.b > \oplus < r_1.b >$
again taking XOR by $< r_0.b >$ on both sides, $< r_1.b' >=< r_1.b >$

clearly for LHS and RHS to be equal b should be equal to the b' but this does not hold because it is assumed that the database is modified. Hence this simple PIR protocol doesn't work in case of a malicious server.

3. Modification in protocol to detect the foul play:

Notations and Assumptions:

- Two Servers S_0 and S_1 are having Databases b and b' here b' may or may not be equal to b.
- e_i is the standard basis vector.

- r_0, r_1 are shares of standard basis vector such that $r_0 \oplus r_1 = e_i$.
- Again α_0, α_1 are two shares of standard basis vector scaled by α such that $\alpha_0 \oplus \alpha_1 = \alpha \cdot e_i$ and $\alpha_0 \neq \alpha \cdot r_0$ (the reason for this is explained in proving the correctness part).

Protocol Steps:

i)Client Query:

- Client sends r_0, r_1 and then (after first response) again shares α_0, α_1 chosen with aforementioned conditions to servers S_0 and S_1 respectively, Note that since client chooses the α_0, α_1 by scaling of std. vector by α so client knows the α.

3/10

ii)Server Response:

- Servers S_0 and S_1 calculates the result for respective shares that they got, and servers first respond with $< r_0 \cdot b >$, $< r_1 \cdot b' >$ respectively for shares of r and then response with $< \alpha_0 \cdot b >$, $< \alpha_1 \cdot b' >$ for shares of $\alpha \cdot e_i$.

Will XOR really work here? You are having XOR and multiplication which are not distributrive.

iii)Client computation to detect foul play:

- Client calculates the result $R_1 =< r_0 \cdot b > \oplus < r_1 \cdot b' >$ by the first response of the server, and then calculates the result $R_2 =< \alpha_0 \cdot b > \oplus < \alpha_1 \cdot b' >$ by the second response.
- Now the client calculates if $\alpha \cdot R_1 = R_2$ holds then the client concludes that the database is not modified otherwise database is modified.

Proof of correctness:

The client determines the result based on the fact that if $\alpha \cdot R_1 = R_2$ then the database is not modified otherwise it is claimed that the database is modified, so to prove the correctness of the protocol proving that when $b' = b$ then $\alpha \cdot R_1 = R_2$ always holds and if $b' \neq b$ (means database is modified) then $\alpha \cdot R_1 = R_2$ does not hold will be sufficient.

Your bigger idea is correct, but, you should have "+" rather than XOR.

Now taking equation $\alpha \cdot R_1 = R_2$ and substituting the values:

$$\alpha \cdot R_1 = R_2$$
$$\alpha(< r_0 \cdot b > \oplus < r_1 \cdot b' >) =< \alpha_0 \cdot b > \oplus < \alpha_1 \cdot b' >$$

This equation is trivial when $b = b'$ so for values other than this when this equation holds-
Assume $(< r_0 \cdot b > \oplus < r_1 \cdot b' >) = k$, then equation becomes

$$\alpha \cdot k =< \alpha_0 \cdot b > \oplus < \alpha_1 \cdot b' >$$

Now consider one of the possible shares of α that is $\alpha_0 = \alpha \cdot r_0$ and $\alpha_1 = \alpha \cdot r_1$. Substituting these values,

This equation does not hold true.

$$\alpha \cdot k = < \alpha \cdot r_0 \cdot b > \oplus < \alpha \cdot r_1 \cdot b' >$$
$$\alpha \cdot k = \alpha \cdot (< r_0 \cdot b > \oplus < r_1 \cdot b' >)$$
$$\alpha \cdot k = \alpha \cdot k$$

Hence, for the equation $\alpha \cdot R_1 = R_2$ to hold, either b=b' or $\alpha_0 = \alpha \cdot r_0$ or $\alpha_1 = \alpha \cdot r_1$ should be the case.

So the equation $\alpha \cdot R_1 = R_2$ only holds if either database is not modified or shares of α are $\alpha_0 = \alpha \cdot r_0$ and $\alpha_1 = \alpha \cdot r_1$ that is why client is generating shares such that this condition is never satisfied then it leaves only one way to hold the equation that is when database it not modified this again implies whenever database it modified this equation will not hold.

Again to prove it formally assume this equation holds and the database is modified ($b \neq b'$) then this indicates-

$$\alpha(< r_0 \cdot b > \oplus < r_1 \cdot b' >) = < \alpha_0 \cdot b > \oplus < \alpha_1 \cdot b' >$$

Given that ($b \neq b'$) so by the above conclusion, this can only hold when $\alpha_0 = \alpha \cdot r_0$ or $\alpha_1 = \alpha \cdot r_1$ and this is our assumption that shares of α can't satisfy this, so by proof of contradiction it means this equation will never hold if the database is modified.

2/5

Solution-3:

Replication is needed

Why do we need that many transtripts.

If there are no computational assumptions then a trivial solution involving a single server can be downloading the whole database, however, there is no non-trivial solution involving a single server with no computational assumptions that guarantees PIR.

Proof: Assume the existence of a non-trivial Private Information Retrieval (PIR) protocol with a communication cost of t bits for a database of N bits, where $t < N$ since the protocol is non-trivial.

Now, considering the protocol cost is t bits, there exist a maximum of 2^t possible communication transcripts. This implies that the client can upload/download a maximum of 2^t different transcripts.

Since the database is N bits, there are a total of 2^N possible transcripts that the database can have. For a non-trivial PIR protocol to guarantee differentiation of all 2^N possible transcripts, the total number of transcripts generated by the protocol must be greater than or equal to the total possible transcripts of the database.

This condition leads to the inequality $2^t \geq 2^N$, which is contradictory because it is known that $t < N$. Thus, by proof of contradiction, it can be concluded that such a protocol does not exist.

why is t < N, what will go wrong if otherwise?

SPIR

Assumptions:

- There is a one-bit database held by the server and the contents of the database are $(x_1, x_2, ..., x_n)$.
- Servers holding database is trusted and it does not modify the content of the database.

1. CPIR Protocol based on QRA: CPIR Protocol is a single server PIR protocol based on Quadratic residue assumption. Given that a is a quadratic residue modulo m if $z^2 \equiv a \pmod{N}$, and without knowing factors of N it is a hard problem to decide that a given number is QR or QNR.
The protocol scheme is as follows-

i) Pre-processing:

- The user/client wants to download the i-th index of the database privately, he starts by choosing 2 large prime numbers p_1,p_2 and multiplies them such that $N = p_1 \cdot p_2$ note that while N can be known to the server but these prime factors are only known to the user.

ii) Client query:

- Now user picks n random numbers $(a_1, a_2, a_3..., a_{i-1}, b_i, a_{i+1}..., a_n)$ less than N where number b_i is QNR and all other numbers are QR then User sends these numbers to the server.

iii) Server response:

- Notice that the server can not decide which number is QNR (or QR) so the server can not differentiate between the numbers and these numbers are received as $(u_1, u_2, u_3..., u_{i-1}, u_i, u_{i+1}..., u_n)$.
- Server then computes and response with $R \leftarrow u_1^{x_1} \cdot u_2^{x_2} \cdot u_3^{x_3} ... \cdot u_n^{x_n}$ where $(x_1, x_2, ..., x_n)$ are the contents of database held by the server.

iv) Client calculation:

- Now the user receives the response R and calculate the i-th bit as- if R is QR then $x_i = 0$ otherwise $x_i = 1$ (in case of R being QNR)

Explanation for why protocol works:

- This protocol works primarily on the computational assumption that the server can not decide that a given number is QR or QNR because Determining whether a number is a quadratic residue modulo a composite number without knowing its prime factors is challenging due to the inherent complexity of factoring large composite numbers.
- This protocol also works because of properties of QR such that $QR \cdot QR = QR$ and $QR \cdot QNR = QNR$, essentially these properties help in decoding the actual i-th bit of the database by using the fact that if i-th bit is set then QNR at i-th bit in the query of the database will be multiplied by the other QRs present in the query and thus result will be QNR.

2. Is the Quadratic Residue based PIR protocol SPIR?
NO, Quadratic residue-based PIR protocols are not necessarily SPIR because in some cases extra information other than x_i can be inferred.

Proof: Assuming there exists a Quadratic Residue-based SPIR protocol that is designed for obtaining x_i, it basically means that the client query in the protocol is using QNR only at i-th place. Now consider the following scenario:

- The client chooses a prime QR number p at index j such that $j \neq i$ and p is co-prime with all other numbers in the query.
- server receives this query and computes the response for the given query like mentioned in the protocol
- Since p is co-prime so if the index j is 1 only then p will be a multiple of the response calculated by the server.

- Now the response that the client gets from the server will be divisible by p if the database on the server contains 1 at index j otherwise not divisible, this helps the client in learning of x_j.

This is contradictory because the protocol was assumed to be SPIR so it should not allow learning anything other than x_i, but the client can easily learn the content of the database at x_j while following the protocol so Quadratic Residue-based PIR protocol is not SPIR.

Solution-4:

1. Cloned the repository using the command:
`git clone https://github.com/avadapal/cs670-iitk-2024.git`

2. I used **make** command described in the make file to run the code and generate the binary file.

3/5:
The output can be
+ or - 2

3. The output for the current code consists of large numbers for each index from 0 to 1023. However, the expected result should be 0 for all indices except the target index, which is 12 in the given code. This discrepancy arises because the code is an implementation of the dpf algorithm for Private Information Retrieval (PIR), and the correct behavior is to evaluate to 0 for all indices except at index 12.

4. Modified the **traverse** function in dpf.h by inserting the correct parameters **(s and cw)** to XOR based on advice bit which resulted in the expected output and now code gives the value 2 at index 12 and 0 elsewhere, also I have added relevant comments in the code itself.