CS670 Assignment-1

Chayan Kumawat - 220309

Program:	B.Tech
Branch:	CSE
Batch:	2026
Date of Sub:	05.03.2034 (DD-MM-YYYY)

Question 1

Application of DPFs

Problem Statement

- Two servers, S_0 and S_1, exist.
- There are C clients, each holding a string $\alpha_i \in \{0,1\}^n$.
- Servers aim to learn the count of clients holding a specific string σ.
- Clients wish to keep their strings secret.

Client-Side Key Generation

- Clients generate DPF keys using a generation algorithm (denoted as `gen`).
- The generation algorithm `gen` takes the client's string (s) and a bit value 1 as parameters and outputs two keys, k_0 and k_1.

  $$Gen(s_i, 1) = \{k_0, k_1\}$$

  And defining this will ensure that

  $$\text{Eval}(k_0, s) + \text{Eval}(k_1, s) = \begin{cases} 1, & \text{for } s = s_i \\ 0, & \text{for } s \neq s_i \end{cases}$$

- s_i is the targeted string.

Client-Server Communication

- Clients communicate with servers using an authenticated, integrity-preserving channel.
- Each client randomly sends one key (k_0 or k_1) to each server.

Server Calculation

- Each server computes the sum of evaluations for all received keys corresponding to the queried string σ.
- Let result_0 and result_1 denote the sums obtained by servers S_0 and S_1, respectively.

  For Server S_0:

  $$\text{result}_0 = \text{Eval}(k_{i1}, \sigma) + \text{Eval}(k_{i2}, \sigma) + \ldots + \text{Eval}(k_{in}, \sigma)$$

  For Server S_1:

  $$\text{result}_1 = \text{Eval}(k_{1-i1}, \sigma) + \text{Eval}(k_{1-i2}, \sigma) + \ldots + \text{Eval}(k_{1-in}, \sigma)$$

Count Calculation

- The total count of clients holding string σ is determined by adding result_0 and result_1.

  $$\text{result}_0 + \text{result}_1 = m$$

  Here the m is the number of clients having the string σ.

Privacy Preservation

- Servers only share their calculated values of result_0 and result_1.
- The privacy of clients' strings is maintained as:
  - To infer a client's string, both $\texttt{Eval}(k_i, \sigma)$ and $\texttt{Eval}(k_{1-i}, \sigma)$ are required.
  - Since these values are distributed across different servers, no single server possesses complete information about any client's string.
  - This ensures that servers cannot deduce which client holds the string σ, preserving client privacy.

Conclusion

By following this protocol, servers can accurately determine the count of clients holding a specific string σ while ensuring the privacy of clients' strings through the use of DPF keys and distributed evaluation functions. This approach allows for secure computation without compromising individual client confidentiality.

Question 2

Malicious Database Holder

To prevent unauthorized alterations to the database content, it's crucial to establish a structured approach:

1. Create a shared key, denoted as 'k', between the database creator and all clients. This key enables the generation of Message Authentication Codes (MACs), ensuring data integrity.

2. Implement asymmetric encryption, which necessitates two distinct keys:

   - The encryption key, identified as $k_{\text{encryption}}$ (referred to as k_{en}), encrypts data. (Known only to the database creator)
   - The decryption key, referred to as $k_{\text{decryption}}$ (referred to as k_{d}), decrypts data. (All clients must know it, even the malicious database holder)

With these keys generated and distributed to the desired holders, the following setup is implemented for all database entries (d_{i}):

1. Use the shared key k to generate the MAC of all database entries d_{i}.

2. Use an asymmetric encryption scheme using the public key k_{en} for encryption and the private key k_{d} for decryption. Create a "checksum" ciphertext $Cipher_i = E_{k_{\text{en}}}(MAC_k(d_i))$.

3. Append $Cipher_i$ to d_i.

The malicious server responds to a query with d_i appended with $E_{k_{\text{en}}}(MAC_k(d_i))$ i.e., $d_i||E_{k_{\text{en}}}(MAC_k(d_i))$.

Client Side:
The client will first separate d_i and $Cipher_i$. Then, using the public decryption key k_d, the client retrieves $MAC_k(d_i)$ from $E_{k_{\text{en}}}(MAC_k(d_i))$. The client generates a MAC using the same key k used by the database creator and compares this MAC with the decrypted MAC appended to the text. This enables the client to detect any tampering by the malicious database holder.

Proof of Correctness of the Protocol:
The protocol's correctness relies on the assumption that the server does not have access to the key k_{en}, thus preventing it from encrypting data. If the server modifies a record d_i to d_i', it would need to compute $E_{k_{\text{en}}}(MAC_k(d_i'))$ to append it to the end to deceive the client. However, the server cannot do this. Even if it knows the key for the MAC, it can only calculate the $MAC(d_i)$ value, which cannot be further encrypted and then sent to the client. This demonstrates the correctness of the protocol.

A Two Server PIR protocol

1: Simple Protocol

When a client needs to retrieve the i-th index from a database, it sends a query to the server indicating this specific index. In standard settings, this query is represented as $\langle 0, 0, \ldots, 1_i, \ldots, 0, 0\rangle$, while the desired data is $\langle 0, 0, \ldots, d_i, \ldots, 0, 0\rangle$.
Our approach involves creating XOR additive shares of the query, sending them to the servers, retrieving two database vectors, and XORing them together.
Let's denote the two additive shares as S_1 and S_2. We aim for these two vectors:

$$S_1 \oplus S_2 = \langle 0, 0, \ldots, 1_i, \ldots, 0, 0\rangle$$

Utilizing the XOR properties:

$$S_2 = S_1 \oplus \langle 0, 0, \ldots, 1_i, \ldots, 0, 0\rangle$$

Initially, we generate a random query share S_1 of size n. Then, using the above property, we compute S_2. Thus, we have two additive shares S_1 and S_2 where $S_1 \oplus S_2 = \langle 0, 0, \ldots, 1_i, \ldots, 0, 0\rangle$.
Next, we send these shares to two replicas of the database, requesting $\langle S_1\rangle \cdot \langle DB\rangle$ and $\langle S_2\rangle \cdot \langle DB\rangle$.
The client then calculates the XOR of the two responses:

$$(\langle S_1\rangle \cdot \langle DB\rangle) \oplus (\langle S_2\rangle \cdot \langle DB\rangle) = (\langle S_1\rangle \oplus \langle S_2\rangle) \cdot \langle DB\rangle)$$

This computation yields:

$$\begin{aligned} &\Rightarrow \langle 0, 0, \ldots, 1_i, \ldots, 0, 0\rangle \cdot \langle DB\rangle \\ &\quad \Rightarrow \langle 0, 0, \ldots, d_i, \ldots, 0, 0\rangle \end{aligned}$$

Thus, we successfully retrieve the d_i element from the database. This protocol can be extended to retrieve multiple indexes. For instance, if we need the i-th, j-th, and k-th indexes, we calculate $\langle S_2\rangle$ using $\langle \ldots, 1_i, \ldots, 1_j, \ldots, 1_k, \ldots\rangle$ and the randomly generated $\langle S_1\rangle$.

2: Will the Simple protocol still work in one server malicious

The protocol outlined above relies on the assumption that the process of correctly computing the dot product with the database and generating the response is executed flawlessly.
However, in scenarios where one of the servers is malicious, there exists a risk of compromising the integrity of our response from the database. A malicious server might manipulate the data it holds, deviating from the expected database content.

Let's consider the case where the second server is malicious. In such a scenario, we obtain responses from both servers, denoted as $\langle S_1\rangle \cdot \langle DB\rangle$ and $\langle S_2\rangle \cdot \langle DB'\rangle$, where DB' represents the potentially tampered database by the malicious server.
The combined result,

$$(\langle S_1\rangle \cdot \langle DB\rangle) \oplus (\langle S_2\rangle \cdot \langle DB'\rangle)$$

may not yield the expected outcome of

$$\langle 0, 0, \ldots, d_i, \ldots, 0, 0\rangle$$

Consequently, this straightforward protocol proves inadequate when confronted with the presence of a malicious server.

In essence, the protocol's vulnerability emerges when trust in the servers' integrity is breached, resulting in potential alterations to the database content by malicious actors. Such deviations jeopardize the reliability and accuracy of the computed responses, highlighting the need for more robust security measures in handling sensitive data interactions.

3: Modified Protocol

To enhance the integrity of the two-server protocol, we propose a modification. In this revised protocol, the client will randomly select an element α from the real numbers in the field F . The client then multiplies this α with their original query, denoted as$\langle 0, 0, 0, \ldots, 1_i, \ldots, 0, 0, 0\rangle$.
Consequently, the client obtains two queries:

$$\langle 0, 0, 0, \ldots, 1_i, \ldots, 0, 0, 0\rangle \text{ and } \langle 0, 0, 0, \ldots, \alpha_i, \ldots, 0, 0, 0\rangle$$

These queries serve distinct purposes: one for retrieving data from the database and the other for data authentication.

We employ the same additive sharing scheme to generate two shares for each query: $\langle S_1^1\rangle$, $\langle S_2^1\rangle$ for the original query, and $\langle S_1^\alpha\rangle$, $\langle S_2^\alpha\rangle$ for the modified query.
Subsequently, both servers compute the dot product of the received queries and send back responses in the form of

$$\langle S_1^1\rangle \cdot \langle DB\rangle, \langle S_1^\alpha\rangle \cdot \langle DB\rangle \text{ and } \langle S_2^1\rangle \cdot \langle DB'\rangle, \langle S_2^\alpha\rangle \cdot \langle DB'\rangle$$

We then verify whether

$$\alpha \cdot ((\langle S_1^1\rangle \cdot \langle DB\rangle) \oplus (\langle S_2^1\rangle \cdot \langle DB'\rangle)) = (\langle S_1^\alpha\rangle \cdot \langle DB\rangle) \oplus (\langle S_2^\alpha\rangle \cdot \langle DB'\rangle)$$

If this condition holds true, it indicates that the database remains unaltered. This verification process enables the client to ascertain data integrity without requiring additional information beyond the original protocol, thereby effectively resolving the issue.

Proof of Correctness of the Protocol :

Detection of Foul Play:

Forward Implication:

If the database is altered, then the dot products computed by the servers will differ from what they should be if the database were intact. This alteration would lead to the verification condition failing.

Backward Implication:

If the verification condition fails, it implies that the dot products computed by the servers do not match the expected values, indicating a discrepancy between the actual database contents and the responses provided by the servers. Therefore, the database must have been altered.

Preservation of Data Integrity:

Forward Implication:

If the database remains unaltered, the dot products computed by the servers will correspond to the correct responses. In this scenario, the verification condition will hold true.

Backward Implication:

If the verification condition holds true, it implies that the dot products computed by the servers match the expected values, indicating consistency between the actual database contents and the responses provided by the servers. Therefore, the database remains unaltered.

By proving both the forward and backward implications for each property, we establish the correctness of the modified protocol, ensuring both detection of foul play and preservation of data integrity.

Question 3

To Prove : To demonstrate that without computational assumptions, a single-server Private Information Retrieval (PIR) scheme cannot ensure privacy, we'll utilize the Pigeonhole Principle.

1. **Setting the Scene:** Let's denote the number of bits in the database as n, and the length of the query as x. For a database of size n, there are 2^x possible query possibilities, resulting in a total of 2^n possible queries.

2. **Applying the Pigeonhole Principle:** Considering that there are 2^n possible queries and only a finite number of possible responses (transcripts), with each response corresponding to a specific query, we apply the Pigeonhole Principle. If we have 2^s possible transcripts (N') in the protocol with communication cost s (where $s < n$), it follows that $2^s < 2^n$.

3. **Conclusion from the Pigeonhole Principle:** The Pigeonhole Principle states that if we have more "pigeons" (possible transcripts) than "pigeonholes" (possible queries), there must be at least two pigeons (transcripts) in the same hole (corresponding to the same query). This implies that there exist at least two different databases, d_1 and d_2, that yield the same response to the user for a given query.

4. **Privacy Violation:** As the server's response is solely based on the requested item and not on the identity of the requester, having two databases that yield the same response violates the privacy guarantee of PIR. Since PIR aims to retrieve each database entry without revealing the requested item, this ambiguity undermines its privacy-preserving property.

5. **Final Conclusion:** In essence, this reasoning shows that without computational assumptions, a single-server PIR scheme fails to ensure privacy. The existence of multiple databases leading to the same response demonstrates a fundamental flaw in achieving information-theoretic privacy in PIR with a single server.

SPIR

1 : Formal Protocol Description

The Protocol

The protocol is designed as follows:

1. The client generates quadratic residues modulo N and $M-1$ entries, where N is a large prime number denoted as $\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_M$. Here, M is the size of the database. The crucial assumption of this protocol is that the prime number N is not disclosed to the server holders. It is assumed that the servers hold the strings $S_1, S_2, \ldots, S_M$, which are unknown to the clients.

2. The client sends the query elements of length M in a specific order, where the non-residue β_i corresponds to the bit in the database that the client wants to retrieve. The sequence sent is $\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \beta_i, \alpha_{i+1}, \ldots, \alpha_M$.

3. Upon receiving the query from the client in the form of $U_1, U_2, \ldots, U_M$, the server, lacking knowledge of the prime number N chosen by the initiator (which helps maintain the main principle of PIR), cannot distinguish between residues and non-residues.

4. The responders compute the value of $Response = U_1^{S_1} \cdot U_2^{S_2} \cdot \ldots \cdot U_M^{S_M}$ and send this result to the client.

5. Upon receiving *Response*, the client checks whether *Response* is a quadratic residue modulo N as follows:
$$Response = \begin{cases} \text{Quadratic residue} & \text{if } S_i = 0 \\ \text{Quadratic non-residue} & \text{if } S_i = 1 \end{cases}$$

Proof of Correctness of the Protocol

The proof of correctness of the protocol relies on the following properties:

1. $QR \cdot QR = QR$
2. $QR \cdot QNR = QNR$
3. $QNR \cdot QNR = QR$

From these properties, it is evident that the product

$$R = U_1^{S_1} \cdot U_2^{S_2} \cdot \ldots \cdot U_{i-1}^{S_{i-1}} \cdot U_{i+1}^{S_{i+1}} \cdot \ldots \cdot U_M^{S_M}$$

is a quadratic residue, irrespective of the exponents (that is, the strings of the database at the server), as

$$U_1, U_2, \ldots, U_{i-1}, U_{i+1}, \ldots, U_M$$

are all quadratic residues, and the nature of the *Response* only depends on the term $\beta_i^{S_i}$.

$$\beta_i^{S_i} = \begin{cases} \text{Quadratic residue} & \text{if } S_i = 0 \\ \text{Quadratic non-residue} & \text{if } S_i = 1 \end{cases}$$

$$Response = R \cdot \beta_i^{S_i}$$

This implies that,

$$\Rightarrow Response = \begin{cases} \text{Quadratic residue} & \text{if } S_i = 0 \\ \text{Quadratic non-residue} & \text{if } S_i = 1 \end{cases}$$

Thus, this concludes that the client gets the value of S_i without the server knowing the interest of the client (as it doesn't know about N, so quadratic and quadratic non-residues are indistinguishable) and without the client knowing extra information.

2 : Proving that Quadratic Residue based PIR protocol SPIR

We have to prove that the client will not be able to know any sight of other information about the other database entries if they use the protocol for the ith index. They will only be able to know the database entry at the ith index, that is S_i.

We will engage in a simple thought activity and show that even if there is a change in the data at other indices of the database, say $j \neq i$, S_j, then the response sent by the client will be exactly the same in both cases. Therefore, if we prove this, we can say that the client will not have any more information than the index the client used the protocol for.

Consider two cases for the database configuration:

$$\langle S_1, S_2, \ldots, S_{j-1}, 0, S_{j+1}, \ldots, S_n \rangle \text{ and } \langle S_1, S_2, \ldots, S_{j-1}, 1, S_{j+1}, \ldots, S_n \rangle$$

Let the product computed in both cases by the server to send to the client be:

$$u_1^{S_1} \cdot u_2^{S_2} \ldots \cdot u_j^0 \cdot u_n^{S_n} \text{ and}$$
$$u_1^{S_1} \cdot u_2^{S_2} \ldots \cdot u_j^1 \cdot u_n^{S_n}$$

The client will not be able to distinguish between them because, when there is a change in the nature of the response even after changing the bit string at the database with index j. This is indeed the case; please carefully observe that u_j^0 and u_j^1 are both quadratic residues modulo N. According to the fundamental properties, there will be no change in the nature with multiplication with Quadratic Residues. Thus, the nature of the two responses will remain the same, either Quadratic Residue or Quadratic Non-Residue. Thus, the client doesn't have any differentiating factor for the two above responses.

This means that for strings at the database at indices other than i, the client is not able to distinguish the responses, thus no extra information than the entry at the ith entry is revealed to the client.

Question 4

- Successfully forked and cloned the repository.
- However, upon running the executable file, it became evident that it produces incorrect output.
- – Referring to the dpf.cpp file, the specified target index is 12 with a target value of 2. The expected output should consist of the reconstruction value and its one's complement.
  - Ideally, for all indices except 12, the output should be 0 0. The reconstruction process is computed as $Eval(k_i, \sigma) - Eval(k_{1-i}, \sigma)$, which yields either 2 or -2.
  - So at targeted index it should be $\{2, (2^{64} - 2)\}$ or $\{(2^{64} - 2), 2\}$.
- The parameters under discussion are 's' and 'cw'.