DATA PRIVACY & SECURITY ASSIGNMENT 4

Problem 1:

1. **Can n players generate s? Why or Why not?**

Ans: In the Chapter 8 slides, we have gone through (n,n) secret splitting tells that it works by dividing a secret into n parts, by giving single share to each participant. The secret can only be recovered when all n members come altogether, and if even one share is missing, the secret stays completely protected.

Let me say the secret is **s**, as given in the question there are **n** players (P_1, P_2,.... P_n)

The dealer will be first creating the n − 1 random binary strings let them be like r_1, r_2,.... r_{n-1}.

Lets say last share be $r_n = s \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus r_{n-1}$ where $\oplus$ it is the bitwise XOR.

As per the question, if all n shares are available, XORing will be $r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus r_n$

Now lets substitute the $r_n = s \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus r_{n-1}$.

$$r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus (s \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus r_{n-1})$$

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

$In\ this\ equation\ r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus r_{n-1} \oplus (s \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus r_{n-1})$ we have seen each random value appearing twice... they are going to be 0 as we know from the above XOR table like $r_1 \oplus r_1 = 0\ like\ that\ for\ till\ r_{n-1}. The\ original\ secret\ that\ is\ left\ is$ **s**.

Now the final equation will look like $s \oplus 0 \oplus 0 \oplus 0 \oplus 0 \ldots\ldots \oplus 0$. From the above XOR table $1 \oplus 0 = 1, s \oplus 0\ going\ to\ be$ **s**. **The final answer is going to be s**.

$$s \oplus 0 \oplus 0 \oplus 0 \oplus 0 \ldots\ldots \oplus 0 = s$$

Finally, when all n participants combine their shares, all the random values cancel each other, and the original secret s we got as solution.

2. **Can any n−1 players generate s? Why or why not?**

Ans: No, I would say any set of n-1 players cannot generate the secret.

Lets have we have **n** shares, r_1, r_2,.... r_n. Lets say r_i is one share that is missing.

The remaining XOR result will be $r_1 \oplus r_2 \oplus \ldots\ldots \oplus r_{i-1} \oplus r_{i+1} \oplus \ldots\ldots \oplus r_n$.

As we already that $r_n = s \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus r_{n-1}$.

The equation will become $r_1 \oplus r_2 \oplus \ldots\ldots \oplus r_{i-1} \oplus r_{i+1} \oplus \ldots\ldots \oplus s \oplus r_1 \oplus r_2 \oplus r_3 \oplus r_4 \ldots\ldots \oplus r_{n-1}$. Now without knowing the missing r_i it is impossible to know about the secret s.

Each bit of the secret is equally likely to be 0 or 1, so without the missing share, we can't know about the secret. Thus we can say that, any n−1 participants cannot know anything, as the missing share completely hides the secret.

Shamir (k,n)-threshold secret sharing.

1. 10 Points. You set up a (k,n) = (2,30) Shamir threshold scheme, working mod the prime p = 101. Two of the shares are (1,13) and (3,12). Another person received the share (2,*). What is M ? What is the value of * ?

Ans:

In the question they have given that, k as 2 and the polynomial degree should be k-1 then it will be 1. So, the equation will become $s(x) = M + a_1x \text{ Mod}(P)$. In the question they have given shares as (1,13) and (3,12). We need to find M.

For the x=1 and x =3, the equations will be like **$M + a_1(1) \equiv 13 \text{ Mod}(101)$ and $M + a_1(3) \equiv 12 \text{ Mod}(101)$.**

Lets say **$M + a_1(3) \equiv 12 \text{ Mod}(101)$** as equation 1 and **$M + a_1(1) \equiv 13 \text{ Mod}(101)$** as equation 2.

Lets subtract equation 1 and equation 2.

$M + a_1(3) – (M - a_1(1)) \equiv 12 \text{ Mod}(101) – 13 \text{ Mod}(101)$

$3 a_1 - a_1 \equiv (12-13) \text{ Mod}(101)$

$2a_1 \equiv (-1) \text{ Mod}(101)$

-1 is equivalent to 100 mod(101) as because 101 is our modulus.

Because -1 + 101 = 100 and **$-1 \equiv 100 \text{ Mod}(101)$**

$2a_1 \equiv (100) \text{ Mod}(101)$

Now lets test for **a_1 as** 51.

$2*x \equiv 1 \text{ Mod}(101)$

when know 2*51 = 102 when we divide 102 by 101 remainder will be 1.

By this we got to know that 51 is the modular inverse of 2.

$2*51 \equiv 1 \text{ Mod}(101)$

We can't divide by 2 in mod 101, so we multiply by its modular inverse. The modular inverse of 2 mod 101 is 51, since $2 * 51 \equiv 1 (\text{mod} 101)$.

We have this equation **$2a_1 \equiv (100) \text{ Mod}(101)$,** lets multiply both sides with 51.

$(2*51)a_1 \equiv (51*100) \text{ Mod}(101)$, but we got to know that $2*51 \equiv 1$

$a_1 \equiv (51*100) \text{ Mod}(101)$

$a_1 \equiv (5100) \text{ Mod}(101)$,

We know that, 101*50 as 5050, as we have 5100, remainder will be 50.

$5100 \equiv (50) \text{ Mod}(101)$,

Finally, **a_1 = 50.**

M + a_1(1) ≡ 13 Mod(101)

We have got **a_1 as 50, M + 50 ≡ 13 Mod(101)**

M ≡ (13-50) Mod(101)

M ≡ (-37) Mod(101).

We know that modular values must be positive within 0 to 100, we add 101 to −37.
That gives 64,

M ≡ (64) Mod(101).

So, finally M is 64.

For the share (2,*) as x is 2.

S(2) = M + **a_1(2) ≡ 64 + 50(2) ≡ 100 + 64 ≡ 164**

We know that modular values must be positive within 0 to 100, lets subtract 101 from 164.
That gives 63.

s(2) ≡ **63 Mod(101).**

From this we got third share as (2,63).

2. **In a (3,5) Shamir secret sharing scheme with modulus p = 17, the following were given to Alice, Bob, Charles: (1,8), (3,10), (5,11). Calculate the corresponding Lagrange interpolating polynomial, and identify the secret M.**

Ans:
In the given question, they have given that (k,n) = (3,5) as Shamir secret Sharing and p as 17. In the question they have given that, k as 3 and the polynomial degree should be k-1 then it will be 2. So, the equation will become s(x) = (a_0 + a_1x + a_2x^2) Mod(P).

The secret as M = a_0.

As we know that general interpolation formula is s(0)= $\sum_{i=1}^{3} L_i(0) * y_i$

We need to calculate $L_i(0)$ the lagrange basis polynomial that we need to evaluated at x=0.

$$L_i(0) = \frac{(0 - x_j)(0 - x_k)}{(x_i - x_j)(x_i - x_k)} \, mod(17)$$

We need to calculate the $L_1(0), L_2(0), L_3(0)$.

Now lets calculate the $L_1(0)$.

Lets calculate for the first share (x_1, y_1) = (1,8)

For i = 1, x_2 = 3, x_3 = 5 it means x_j = 3 and x_k=5

$$L_1(0) = \frac{(0 - 3)(0 - 5)}{(1 - 3)(1 - 5)} \, mod(17)$$

$$L_1(0) = \frac{15}{(-2)*(-4)} \, mod(17)$$

$$L_1(0) = \frac{15}{8} \, mod(17)$$

When we are doing the modular arithmetic, we can't do division directly.

So instead of dividing by 8, we multiply by its modular inverse, a number that gives remainder 1 when multiplied by 8 (mod 17).

When we test, $8 * 15 = 120 \equiv 1(\text{mod}17)$, so the inverse of 8 (mod 17) is 15.

$$L_1(0) = 15 * 15 \, mod(17)$$

$$L_1(0) = 225 \, mod(17)$$

We know that , 17*13 = 221, so the remainder will be 4.

$$L_1(0) = 4.$$

Now lets calculate the $L_2(0)$.

Lets calculate for the first share (x_2, y_2) = (3,10)

For $x_1 = 1$, $x_2 = 3$, $x_3 = 5$ it means $x_j = 1$ and $x_k = 5$

$$L_2(0) = \frac{(0-1)(0-5)}{(3-1)(3-5)} \, mod(17)$$

$$L_2(0) = \frac{5}{(2)*(-2)} \, mod(17)$$

$$L_2(0) = \frac{5}{-4} \, mod(17)$$

When we are doing the modular arithmetic, we can't do division directly.

So instead of dividing by -4, we will convert that into positive equivalent and then we multiply by its modular inverse,

To convert -4 mod(17) to a positive number, -4+17 = 13

So -4 ≡ 13 mod(17)

So instead of dividing by 13, we multiply by its modular inverse, a number that gives remainder 1 when multiplied by 13 (mod 17).

When we test, $13 * 4 = 52 \equiv 1(\text{mod}17)$, so the inverse of 13 (mod 17) is 4.

$$L_2(0) = 5 * 4 \, mod(17)$$

$$L_2(0) = 20 \, mod(17)$$

We know that , 17*1 = 17, 20 – 17 is 3, so the remainder will be 3

$$L_2(0) = 3.$$

Now lets calculate the $L_3(0)$.

Lets calculate for the first share (x_3, y_3) = (5,11)

For $x_1 = 1$, $x_2 = 3$, $x_3 = 5$ it means $x_j = 1$ and $x_k = 3$

$$L_3(0) = \frac{(0-1)(0-3)}{(5-1)(5-3)} \, mod(17)$$

$$L_3(0) = \frac{3}{(4)*(2)} \, mod(17)$$

$$L_3(0) = \frac{3}{8} \, mod(17)$$

When we are doing the modular arithmetic, we can't do division directly.

So instead of dividing by 8, we multiply by its modular inverse, a number that gives remainder 1 when multiplied by 8 (mod 17).

When we test, $8 * 15 = 120 \equiv 1(\text{mod}17)$, so the inverse of 8 (mod 17) is 15.

$$L_3(0) = 15 * 3 \, mod(17)$$

$$L_3(0) = 45 \, mod(17)$$

We know that , 17*2= 34, so the remainder will be 11.

$$L_3(0) = 11.$$

We got $L_1(0) = 4$, $L_2(0) = 3$, $L_3(0) = 11$ and given $y_1 = 8$, $y_2 = 10$, $y_3 = 11$,

$s(0) = L_1(0) * y_1 + L_2(0) * y_2 + L_3(0) * y_3$

$s(0) = 4*8 + 3*10 + 11*11$

s(0) = 32+30+ 121

s(0) = 183 ≡ 13 mod(17)

When we take mod 17 to the 183 then the remainder will be 13 (because 17*10 = 170)

So **S(0) = 13.**

Finally we got **secret (M) as 13** and **$L_1(0) = 4$, $L_2(0) = 3$, $L_3(0) = 11$.**

Problem 2 (30 Points):

1. **5 Points.**

Ans:

No, it is not a ZKP protocol. Actually, this scenario fails the of zero-knowledge because the camera turns Peggy's private proof into something that can be shown to some other. In a real zero-knowledge proof, the verifier only get to knows (learns) that the claim is true and nothing else. But if a video shows Victor's random questions and Peggy always giving the right answers, it becomes evidence that she will be knowing the secret. Anyone watching that video can see that her chances of guessing correctly every time are mostly impossible, so the video gives away more information than it should.

This recording will basically allow Victor to prove to others that Peggy knows the secret, which is not at all how the zero-knowledge proof protocol is supposed to play. In a correct situation, Victor should only be convinced personally and should not be able to convince anyone else. Once the interaction is recorded, it will become public proof instead of a private check, it means protocol is no longer truly zero-knowledge.

2. **5 Points.**

Ans:

No, this is not a zero-knowledge proof protocol. In this scenario, Peggy is directly showing Victor that she knows the magic word by going in through path A and coming out from path B in front of him. This means Victor sees her use the secret, which will get to know the knowledge instead of just proving it. In a correct zero-knowledge proof, the verifier should only be convinced that the prover knows the secret but should not learn or see anything that exposes it.

Here, since Victor witnesses the whole process in one trial, it fails the zero-knowledge property. Peggy's action will give away the secret instead of hiding it, so this is not a valid zero-knowledge proof.

The Sudoku Game (20 Points).

Ans:

In this situation, Alice want to prove to bob that she has solved the sudoku puzzle and has a valid solution but without revealing the actual numbers.

Protocol:

Alice wants to show Bob she solved a Sudoku puzzle correctly without revealing her complete answer. To do this, she takes the solved Sudoku grid and covers each cell with a card placed face down so that no numbers are visible. Bob can see the original unsolved puzzle that both already know, but he cannot see Alice's completed solution. This setup ensures that Alice's numbers always remain hidden all the time.

Next, Bob randomly chooses what he wants to verify. He can ask Alice to show one full row, one full column, or one 3*3 block. When Bob chooses his choice, Alice only lifts the cards for that section. Bob then checks that all the numbers from 1 to 9 appear exactly once in that row, column, or block, and that the numbers match with the puzzle's original clues. Once Bob finishes checking, Alice covers the numbers again, and Bob repeats the same process several times with different sections. After multiple checks, Bob becomes convinced that Alice's solution is correct, even though he never gets to see the complete grid.

Completeness:
If Alice really has the correct Sudoku solution, she will always be able to show Bob that every row, column, or block he chooses is valid. Each time Bob checks, he'll see that the numbers follow all Sudoku rules. After several checks, Bob will be sure that Alice's solution is correct.

Soundness:
If Alice is lying or doesn't have the right solution, she won't be able to pass all of Bob's random checks. Now or then, Bob will pick a row, column, or block that has a mistake like a missing or repeated number. The more times Bob checks, the harder it becomes for Alice to cheat without being caught.

Zero-Knowledge:
Bob only sees small parts of the Sudoku each time, not the whole grid. Because of this, he learns nothing about Alice's full solution only that it's valid. In fact, Alice could easily make fake examples that look the same to Bob without giving away any real numbers. This means Bob knows she solved it but doesn't know how to learn.

Conclusion:

This approach allows Alice to prove she solved the Sudoku without disclosing answers like a genuine solver passes all checks (completeness), a fake is caught (soundness), and the solution remains hidden (zero-knowledge).

Problem 3 (40 Points):

1. **Protocol design (15 Points).**

Ans:

Private Information Retrieval (PIR) Protocol Design

Background:

Private Information Retrieval (PIR) is a cryptographic protocol that lets clients access specific database items without revealing which ones. Using homomorphic encryption, it tries servers compute on encrypted queries without knowing the requested record. PIR relies on secure encryption schemes to maintain client privacy.

Protocol Design

The PIR protocol is implemented using the SHEEP framework, which supports fully homomorphic encryption backends such as SEAL, HElib, and TFHE. The system consists of a client and a server and the client generates an encryption key pair, while the server holds an encrypted version of the database.

Description of the Protocol:

In the **Setup Phase**, the server stores the database $D = [d_1, d_2, \ldots, d_n]$. The client generates a public-private key pair, shares the public key with the server, and keeps the private key secret. This setup enables the server to compute on encrypted data while preserving the client's privacy.

In the **Query Phase,** 7 steps are involved:

1. **Selection Vector of the Client:**
   The client creates a binary vector $s = [s_1, s_2, \ldots, s_n]$, where $s_i = 1$ which marks the desired record's index, and all other $s_j = 0$.
2. **Encryption:**
   Each bit of the selection vector is encrypted using the public key, forming $\mathrm{Enc}(s) = [\mathrm{Enc}(s_1), \mathrm{Enc}(s_2), \ldots, \mathrm{Enc}(s_n)]$.
3. **Transmission to Server:**
   The encrypted selection vector $Enc(s)$is sent to the server.
4. **Homomorphic Computation:** The server performs the encrypted dot product operation by utilizing the homomorphic properties inherent in the encryption scheme.
$$\mathrm{Enc}(r) = \sum_{j=1}^{n} \mathrm{Enc}(d_j) \odot \mathrm{Enc}(s_j)$$
5. **Result Simplification:**
   Since all $s_j = 0$ except one $s_i = 1$, the result simplifies to $\mathrm{Enc}(r) = \mathrm{Enc}(d_i)$.
6. **Response to Client:**
   The server sends the encrypted result $\mathrm{Enc}(r)$back to the client.
7. **Decryption:**
   The client uses its private key to decrypt the ciphertext and obtain the requested data.

$$r = \mathrm{Dec}(\mathrm{Enc}(r)) = d_i$$

Details of the Implementation:

The SHEEP framework uses a basic dot product circuit to multiply each database item by its selection bit and sum the outputs. All these computations are carried out using homomorphic operations. Its primary operations include:

- **Encryption:** $\mathrm{Enc}(\mathrm{plaintext})$
- **Decryption:** $\mathrm{Dec}(\mathrm{ciphertext})$
- **Homomorphic Addition:** $\mathrm{Enc}(\mathrm{a}) + \mathrm{Enc}(\mathrm{b}) = \mathrm{Enc}(\mathrm{a} + \mathrm{b})$
- **Homomorphic Multiplication:** $\mathrm{Enc}(\mathrm{a}) \odot \mathrm{Enc}(\mathrm{b}) = \mathrm{Enc}(\mathrm{a} * \mathrm{b})$

The protocol may be run in "Clear" mode without encryption to verify its accuracy before applying real encryption.

Execution Example:

For the database $D = [2,4,6,8,10,1,3]$, each query uses a binary selection vector with a single 1.

Query Index (i)	Selection Vector	Computed Dot Product	Retrieved Value
0	[1, 0, 0, 0, 0, 0, 0]	$2*1+4*0+6*0+8*0+10*0+1*0+3*0$	2
1	[0, 1, 0, 0, 0, 0, 0]	$2*0+4*1+6*0+8*0+10*0+1*0+3*0$	4
2	[0, 0, 1, 0, 0, 0, 0]	$2*0+4*0+6*1+8*0+10*0+1*0+3*0$	6
3	[0, 0, 0, 1, 0, 0, 0]	$2*0+4*0+6*0+8*1+10*0+1*0+3*0$	8
4	[0, 0, 0, 0, 1, 0, 0]	$2*0+4*0+6*0+8*0+10*1+1*0+3*0$	10
5	[0, 0, 0, 0, 0, 1, 0]	$2*0+4*0+6*0+8*0+10*0+1*1+3*0$	1
6	[0, 0, 0, 0, 0, 0, 1]	$2*0+4*0+6*0+8*0+10*0+1*0+3*1$	3

Each result matches the database, confirming that the PIR protocol retrieves data accurately and maintains privacy.

Justification using Theory:

The protocol works due to the **linearity property** of homomorphic encryption.
When computing the dot product

$\mathrm{Enc}(r) = \mathrm{Enc}(d_1 s_1 + d_2 s_2 + \ldots + d_n s_n)$, all terms where $s_j = 0$

cancel out, leaving only the encrypted desired element $\mathrm{Enc}(d_i)$.
This property ensures correctness (the client receives the correct record), privacy (the query index stays hidden), and non-leakage (the server learns nothing about access patterns or indexes).

Conclusion:

This PIR protocol will enable clients to query databases privately via homomorphic encryption, so the server will never learn and know which record was accessed. Using the SHEEP framework, it delivers privacy, accuracy, and security for efficient private data retrieval.

2. Implementation (25 Points). This simple dot-product protocol for a database of a specific size can be run by the the SHEEP platform.

Ans:

hw4-3-pir_sheep.py

```
#!/usr/bin/env python


import sys

import os


sheep_frontend_path = os.path.join(os.path.dirname(__file__), "SHEEP-master", "frontend")

if os.path.exists(sheep_frontend_path):

  sys.path.insert(0, sheep_frontend_path)

  from pysheep import sheep_client

else:

  print(f"Error: SHEEP-master folder not found at {sheep_frontend_path}")

  print("Please ensure SHEEP-master folder is in the same directory as this script")

  sys.exit(1)


def create_dot_product_circuit(n):

  circuit_lines = [

    "INPUTS database selection",

    "CONST_INPUTS rotate_1",
```

```
        "OUTPUTS result",
        "database selection MULTIPLY prod_r0"
    ]

    for i in range(n - 1):
        circuit_lines.append(f"prod_r{i} rotate_1 ROTATE prod_r{i+1}")
        if i == 0:
            circuit_lines.append(f"prod_r0 prod_r1 ADD prod_s1")
        else:
            circuit_lines.append(f"prod_s{i} prod_r{i+1} ADD prod_s{i+1}")

    if n > 1:
        circuit_lines.append(f"prod_s{n-1} ALIAS result")
    else:
        circuit_lines.append(f"prod_r0 ALIAS result")

    return "\n".join(circuit_lines)

def main():
    database = [2, 4, 6, 8, 10, 1, 3]
    n = len(database)

    print(f"Database contains {n} elements: {database}\n")

    try:
        contexts = sheep_client.get_available_contexts()
        if contexts["status_code"] != 200:
            print("Cannot connect to SHEEP server")
            print("Make sure the server is running on http://localhost:34568/SheepServer")
            return
    except Exception as e:
```

```
    print(f"Cannot connect to SHEEP server: {e}")
    print("Make sure the server is running on http://localhost:34568/SheepServer")
    return

circuit = create_dot_product_circuit(n)

available_contexts = contexts["content"]
if "SEAL_BFV" in available_contexts:
    context = "SEAL_BFV"
elif "SEAL_CKKS" in available_contexts:
    context = "SEAL_CKKS"
elif len(available_contexts) > 0:
    context = available_contexts[0]
else:
    print("No encryption contexts available")
    return

print("Retrieving each element from the database:\n")

for i in range(n):
    sheep_client.new_job()
    sheep_client.set_context(context)
    sheep_client.set_input_type("int16_t")
    sheep_client.set_circuit_text(circuit)
    sheep_client.set_const_inputs({"rotate_1": -1})

    selection = [0] * n
    selection[i] = 1

    print(f"Retrieving element at position {i}:")
    print(f"  Selection vector: {selection}")
```

```
sheep_client.set_inputs({
   "database": database,
   "selection": selection
})

run_result = sheep_client.run_job()
if run_result["status_code"] != 200:
   print(f"  ERROR: Job execution failed - {run_result['content']}")
   continue

results = sheep_client.get_results()
if results["status_code"] != 200:
   print(f"  ERROR: Failed to get results - {results['content']}")
   continue

result_content = results["content"]
if "outputs" not in result_content:
   print(f"  ERROR: No outputs in results")
   continue

output_values = result_content["outputs"]["result"]
if isinstance(output_values[0], str):
   retrieved_value = int(output_values[0].split(',')[0])
else:
   retrieved_value = int(output_values[0])

if "cleartext check" in result_content:
   is_correct_cleartext = result_content["cleartext check"].get("is_correct", False)
   if not is_correct_cleartext:
      print(f"  WARNING: Cleartext check failed - circuit may be incorrect")
```

```
    expected_value = database[i]
    is_correct = (retrieved_value == expected_value)

    print(f"  Retrieved value: {retrieved_value}")
    print(f"  Expected value:  {expected_value}")
    if is_correct:
      print(f"  Correct!")
    else:
      print(f"  Incorrect!")

if __name__ == "__main__":
  main()
```

Implementation notes:

For the implementation to run the PIR demo, first install the dependencies by running **pip install -e SHEEP-master/frontend** and **pip install requests.**

Then, open two separate terminal windows: in the first terminal, start the server by running the **SHEEP server** using Docker (**cd SHEEP-master && docker-compose up sheep-server**) or the local build **(cd SHEEP-master/backend/build && ./bin/run-sheep-server**), and keep it running. In the second terminal, execute python hw4-3-pir_sheep.py (or use hw4-3-RUN_DEMO.bat on Windows) to run the demo. The demo will retrieve each element from the database D = [2, 4, 6, 8, 10, 1, 3] using encrypted selection vectors and display the results showing the retrieved value, expected value, and whether each retrieval was correct.

```
@Ranjithv280502 ➜ /workspaces/CS528-Assignment4/SHEEP-master (main) $ docker
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  bake        Build from a file
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Authenticate to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information
```

```
@Ranjithv280502 ➜ /workspaces/CS528-Assignment4 (main) $ cd /workspaces/CS528-Ass
ignment4/SHEEP-master
docker-compose up -d sheep-server
WARN[0000] /workspaces/CS528-Assignment4/SHEEP-master/docker-compose.yml: the att
ribute `version` is obsolete, it will be ignored, please remove it to avoid poten
tial confusion
[+] Building 190.4s (19/19) FINISHED
 => [internal] load local bake definitions                                  0.0s
 => => reading from stdin 418B                                              0.0s
 => [internal] load build definition from Dockerfile                        0.0s
 => => transferring dockerfile: 1.07kB                                      0.0s
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (lin  0.0s
 => [internal] load metadata for docker.io/turingsheep/sheep_base:v3        0.5s
 => [auth] turingsheep/sheep_base:pull token for registry-1.docker.io       0.0s
 => [internal] load .dockerignore                                           0.0s
 => => transferring context: 2B                                             0.0s
 => [ 1/11] FROM docker.io/turingsheep/sheep_base:v3@sha256:09af5e9044aa  117.3s
 => => resolve docker.io/turingsheep/sheep_base:v3@sha256:09af5e9044aa14b3  0.0s
 => => sha256:09af5e9044aa14b3ad34986007460f791962aee7c4 11.25kB / 11.25kB  0.0s
 => => sha256:9ff7e2e5f967fb9c4e8099e63508ab0dddebe3f8 43.77MB / 43.77MB  189.7s
 => => sha256:59856638ac9f32d4caa0f5761b2597fe251642786fdfe1b9 848B / 848B  0.1s
 => => sha256:fc0ac7d4c43db0a3c21fc367864eff02fbbf160ae8 13.49kB / 13.49kB  0.0s
```

```
@Ranjithv280502 ➜ /workspaces/CS528-Assignment4 (main) $ cd /workspaces/CS528-Ass
ignment4/SHEEP-master
docker-compose up -d sheep-server
 => => transferring context: 474.86kB                                       0.1s
 => [ 2/11] ADD backend/applications SHEEP/backend/applications             0.4s
 => [ 3/11] ADD backend/cmake SHEEP/backend/cmake                           0.0s
 => [ 4/11] ADD backend/CMakeLists.txt SHEEP/backend/CMakeLists.txt         0.0s
 => [ 5/11] ADD backend/include SHEEP/backend/include                       0.0s
 => [ 6/11] ADD backend/src SHEEP/backend/src                               0.0s
 => [ 7/11] ADD backend/tests SHEEP/backend/tests                           0.1s
 => [ 8/11] RUN rm -fr SHEEP/backend/build; mkdir SHEEP/backend/build       0.6s
 => [ 9/11] RUN cd SHEEP/backend/build; export CC=gcc-7; export CXX=g++-7;  3.9s
 => [10/11] RUN cd SHEEP/backend/build; make VERBOSE=1 run-sheep-server    43.1s
 => [11/11] WORKDIR SHEEP/backend/build                                     0.0s
 => exporting to image                                                     24.0s
 => => exporting layers                                                    24.0s
 => => writing image sha256:940d7f3f20dd373e29bd8408fceac490e89f7636a40e50  0.0s
 => => naming to docker.io/library/sheep-master-sheep-server                0.0s
 => resolving provenance for metadata file                                  0.0s
[+] Running 3/3
 ✓ sheep-server                  Built                                      0.0s
 ✓ Network sheep-master_sheep_nw  Creat...                                  0.1s
 ✓ Container sheep_sheep-server_1  Star...                                  1.1s
```

Outputs:

```
@Ranjithv280502 ➜ /workspaces/CS528-Assignment4 (main) $ python hw4-3-pir_sheep.p
y
Database contains 7 elements: [2, 4, 6, 8, 10, 1, 3]

Retrieving each element from the database:

Retrieving element at position 0:
  Selection vector: [1, 0, 0, 0, 0, 0, 0]
  Retrieved value: 2
  Expected value:  2
  Correct!
Retrieving element at position 1:
  Selection vector: [0, 1, 0, 0, 0, 0, 0]
  Retrieved value: 4
  Expected value:  4
  Correct!
Retrieving element at position 2:
  Selection vector: [0, 0, 1, 0, 0, 0, 0]
  Retrieved value: 6
  Expected value:  6
  Correct!
```

```
y
  Expected value:  6
  Correct!
Retrieving element at position 3:
  Selection vector: [0, 0, 0, 1, 0, 0, 0]
  Retrieved value: 8
  Expected value:  8
  Correct!
Retrieving element at position 4:
  Selection vector: [0, 0, 0, 0, 1, 0, 0]
  Retrieved value: 10
  Expected value:  10
  Correct!
Retrieving element at position 5:
  Selection vector: [0, 0, 0, 0, 0, 1, 0]
  Retrieved value: 1
  Expected value:  1
  Correct!
Retrieving element at position 6:
  Selection vector: [0, 0, 0, 0, 0, 0, 1]
  Retrieved value: 3
  Expected value:  3
  Correct!
```

Verification of results:

Position	Retrieved	Expected	Status
0	2	2	Correct
1	4	4	Correct
2	6	6	Correct
3	8	8	Correct
4	10	10	Correct
5	1	1	Correct
6	3	3	Correct

Conclusion:

This PIR protocol will enable clients to query databases privately via homomorphic encryption, so the server will never learn and know which record was accessed. Using the SHEEP framework, it delivers privacy, accuracy, and security for efficient private data retrieval.