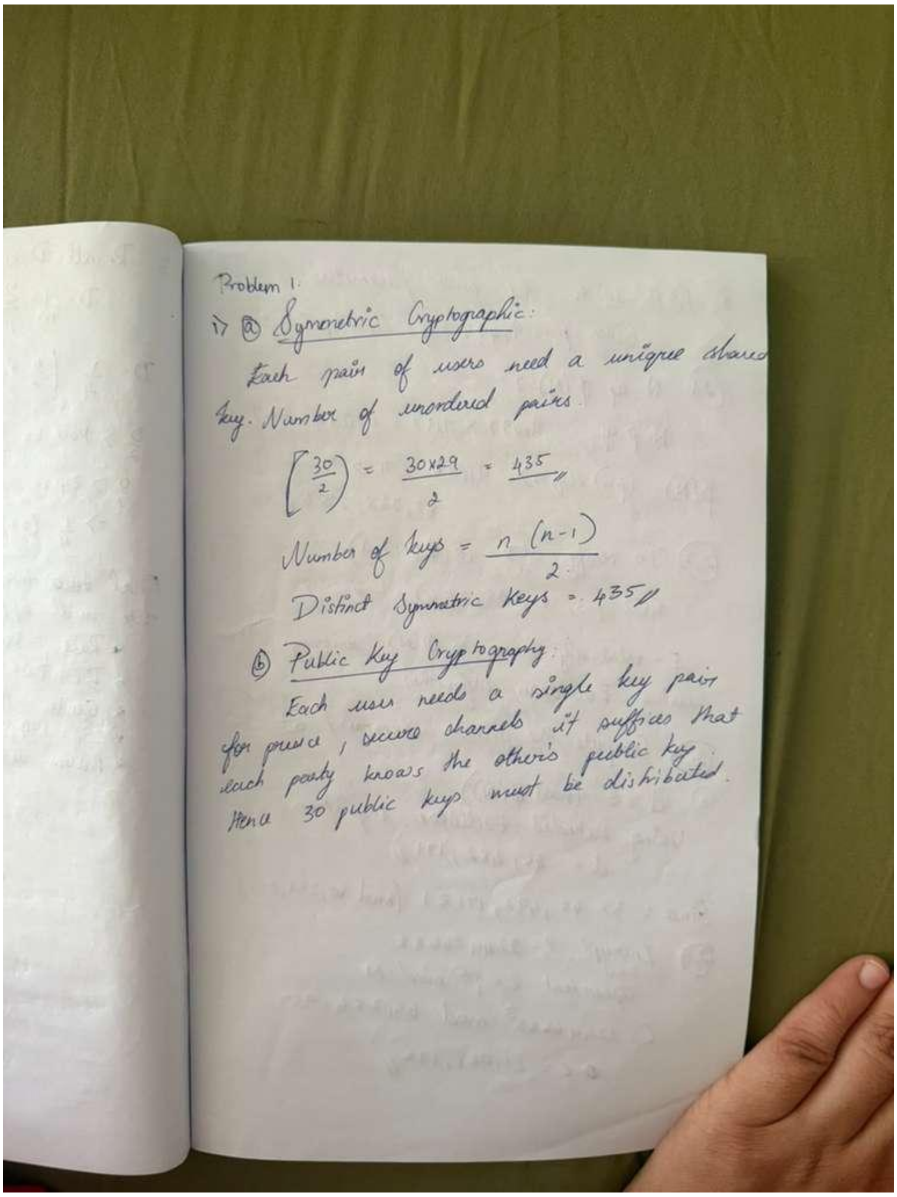
Assignment – 4

CS528 Data Security and Privacy

Akshitha Priyadarshini Murugan
A2060112

Problem 1 (20 Points): Symmetric/Asymmetric Encryption 1. 2 Points. Consider a group of 30 people in a room who wish to be able to establish pairwise secure communications in the future. How many keys need to be exchanged in total: (a) Using symmetric cryptography? (b) Using public key cryptography?



Problem 1:

1) (a) Symmetric Cryptographic:

Each pair of users need a unique shared key. Number of unordered pairs

$$\binom{30}{2} = \frac{30\times 29}{2} = 435$$

Number of keys $= \frac{n(n-1)}{2}$

Distinct Symmetric Keys = 435

(b) Public Key Cryptography:

Each user needs a single key pair for private, secure channels it suffices that each party knows the other's public key. Hence 30 public keys must be distributed.

2. 14 Points. The following question has you use RSA. You may use a program that you write or any other computer program to help you solve this problem. Let p = 9, 497 and q = 7, 187 and e = 3. • 2 Points What is N? What is Φ(N)? • 2 Points Verify that e is relatively prime to Φ(N). What method did you use to verify this? • 2 Points Compute d

as the inverse of e mod Φ(N). What is d? • 2 Points Encrypt the value P = 22446688 with the RSA primitive and the values for N and e above. Let C be the resulting ciphertext. What is C? • 2 Points Verify that you can decrypt C using d as the private exponent to get back P. What method did you use to verify this? • 2 Points Decrypt the value C ′ = 11335577 using the RSA primitive and your values for N and d above. Let P ′ be the resulting plaintext. What is P ′ ? • 2 Points Verify that you can encrypt P ′ using e as the public exponent to get back C ′ . What method did you use to verify this?

2) RSA with the given parameters.

Given $p = 9{,}497$ $q = 7{,}187$, $e = 3$

2.1 N & $\phi(N)$?

$N = p \cdot q = 9{,}497 \times 7{,}187 = 68{,}254{,}939$ //

$\phi(N) = (p-1) \times (q-1) = 9496 \times 7186 =$
$68{,}238{,}256$ //

2.2 To verify GCD $(e, \phi(N) = 1)$

GCD $[3, 68{,}238{,}256) = 1$

E - relatively prime to $\phi(N)$ - verifying using Euclidean Algorithm.

2.3 Compute the private exponent at

$d \equiv e^{-1} \pmod{\phi(N)}$

Using extended Euclidean Algorithm

$d = 45{,}482{,}171$ //

Since: $3 \times 45{,}492{,}171 \equiv 1 \pmod{68{,}238{,}256}$

2.4 Encrypt $P = 2244\ 6688$

Ciphertext $C \equiv P^e \bmod N$

$C = 22446688^3 \bmod 68{,}254{,}939$

$C = 23{,}081{,}176$ //

(2.5) Decrypt c to recover P.

$P \equiv C^d \bmod N$

$P = 23{,}081{,}176^{45{,}492{,}171} \pmod{68{,}254{,}939}$

$P = 22{,}44{,}6{,}688$ //

Using the private exponent d - modular exponentiation.

(2.6) Decrypt $c' = 11{,}335{,}577$

$p' = (c')^d \bmod N$

$p' \equiv 11335577^{45492171} \pmod{68{,}254{,}939}$

$p' = 35{,}654{,}065$ //

(2.7) Re-encrypt p' to verify c'

$= (p')^e \bmod N$

$= 35654065^3 \pmod{68{,}254{,}939}$

$= 11{,}33{,}5577 \Rightarrow c''$ ✓

3. 4 Points. Consider a Diffie-Hellman key exchange with p = 29 and g = 2. Suppose that Alice picks x = 3 and Bob picks y = 5. What will each party send to the other, and what shared key will they agree on? Show your details.

(c) The shared secret key k. Alice computes k using Bob's public key B and her private key x:

$$K_{Alice} = Bob^{x} \pmod p) = 3^3 \pmod{29}$$

$$= 27 \pmod{29}$$

$$K_A = 27$$

$$K_{Bob} = Alice^{y} \pmod p) = 8^5 \pmod{29})$$

$$8^5 \equiv 32768 \pmod{29}$$

$$\frac{32768}{29} = 1129 \text{ with remainder } 27.$$

$$K_B = 27$$

Since $kA = kB = 27$, the shared secret key $K = 27$

~~Problem 2~~

3> The Diffie Hellman key exchange protocol allows two parties, Alice & Bob to establish a shared secret key over an insecure channel, based on the difficulty of the Discrete logarithm Problem.

Public Parameters:

→ Prime modulus $P = 29$

→ Base (generator) $g = 2$

Private Keys:

→ Alice's secret $x = 3$

→ Bob's secret $y = 5$

(a) Alice ~~Privat~~ Public key A

$A = g^{x} \pmod{p}$

$= 2^{3} \pmod{29}$

$= 8 \pmod{29}$

$A = 8$ //

(b) Bobs Public key. B

$B = g^{y} \pmod{p}$

$= 2^{5} \pmod{29}$

$= 32 \pmod{29}$

$B = 3$ //

Problem 2 (20 Points): Homomorphic Encryption: Pallier encryption Let N = pq where p and q are two prime numbers. Let g ∈ [0, N2] be an integer satisfying g = aN +1(mod N2) for some integer a ≤ N. Consider the following encryption scheme. The public key is (N, g). The private key is (p, q, a). To encrypt a (integer) message m, one picks a random integer h, and computes C = g mh N modN2 . Our goal is to develop a decryption algorithm and to show the homomorphic property of the encryption scheme.

- **Show the discrete log problem "mod N2 base g" is easy when knowing the private key. That is, show that given g and B = g x mod N2 , there is an efficient algorithm to recover x mod N. Use the fact that g = aN + 1 for some integer a ≤ N.**

Problem 2:

Let $N = p \cdot q$ with odd primes p, q

Let $g \in \mathbb{Z}_{N^2}$ satisfy

$$g \equiv 1 + aN \pmod{N^2} \text{ for some } a \in \mathbb{Z}_N$$

with $\gcd(a, N) = 1$.

Public key: (N, g). Private key: (p, q, a)

hence $\phi(N) = (p-1)(q-1)$

Encryption of $M \in \mathbb{Z}_N$: pick random $h \in \mathbb{Z}_N$

and compute: $C \equiv g^m h^N \pmod{N^2}$

(a) Discrete log mod N^2 base g" is easy with private key (p, q, a)

Claim: Given $B = g^x \pmod{N^2}$ and knowing a, we can recover x mod N efficiently.

Proof: Using the binomial identity modulo N^2 and $g = 1 + aN$:

$$g^x = (1+aN)^x \equiv 1 + axN \pmod{N^2}$$

Since all higher terms certain $(a, N)^2$ and thus vanish modulo N^2. Therefore.

$$\frac{B-1}{N} \equiv a \cdot x \ (\text{Mod } x)$$

Because $\gcd(a, N) = 1$, a is investible modulo N, Hence

$$x \equiv a^{-1} \times \frac{B-1}{N} \pmod{N}$$

This computation uses only modular arithmetic and the extended Euclidean algorithm, so it is efficient.

b) Efficient decryption of $C = g^m h^N \mod N^2$

We show how to recover m from C using (p, q, a)

Key facts: * $\phi(N) = (p-1)(q-1)$ ay $\gcd(\phi(N), N) = 1$

* For any $h \in Z_{N^2}$, by Euler's theorem over modulus N^2,

$h^{N\phi(N)} \equiv 1 \pmod{N^2}$

$(h^N)^{\phi(N)} \equiv 1 \pmod{N^2}$

for $g = 1 + aN$

$(g^m)^{\phi(N)} = (1+aN)^{m\phi(N)} \equiv 1 + (ma\phi(N))N \pmod{N^2}$

Compute $C^{\phi(N)} \mod N^2$:

$C^{\phi(N)} \equiv (g^m h^N)^{\phi(N)} \equiv (g^m)^{\phi(N)} (h^N)^{\phi(N)}$

$\equiv (1 + (ma\phi(N))N \cdot 1 \equiv 1 + (ma\phi(N))N \pmod{N^2}$

Define the paillier L-function

$L(u) = \frac{u-1}{N}$

(An integer mod N where $u \equiv 1 \pmod{N}$).

- **Show that given the public and private key, decrypting C = g mh N mod N2 can be done efficiently. Hint: consider C ϕ(N) mod N2 . Use the fact by Euler's theorem x ϕ(N2) = 1 mod N2 for any x**

This computation uses only modular arithmetic and the extended Euclidean algorithm, so it is efficient.

b) Efficient decryption of $C = g^m h^N \bmod N^2$

We show how to recover m from C using (p, q, a)

Key facts: • $\phi(N) = (p-1)(q-1)$ and $\gcd(\phi(N), N) = 1$

• For any $h \in \mathbb{Z}_{N^2}$, by Euler's theorem over modulus N^2.

$h^{N\phi(N)} \equiv 1 \pmod{N^2}$

$(h^N)^{\phi(N)} \equiv 1 \pmod{N^2}$

for $g = 1 + aN$

$(g^m)^{\phi(N)} = (1+aN)^{m\phi(N)} \equiv 1 + (ma\phi(N))N \pmod{N^2}$

Compute $C^{\phi(N)} \bmod N^2$:

$C^{\phi(N)} \equiv (g^m h^N)^{\phi(N)} \equiv (g^m)^{\phi(N)} (h^N)^{\phi(N)}$

$\equiv (1 + (ma\phi(N))N \cdot 1 \equiv 1 + (ma\phi(N))N \pmod{N^2}$

Define the paillier L-function

$L(u) = \frac{u-1}{N}$

(An integer mod N where $u \equiv 1 \pmod{N}$).

•

Then $L(C^{\phi(N)}) \equiv ma\phi(N) \pmod{N}$

Similarly,

$g^{\phi(N)} \equiv 1 + (a\phi(N))N \pmod{N^2}$

$L(g^{\phi(N)}) \equiv a\phi(N) \pmod{N}$

Because $\gcd(a\phi(N), N) = 1$, the value $L(g^{\phi(N)})$ is invertible mod N

$\therefore m \equiv L(C^{\phi(N)} \bmod N^2) \cdot L(g^{\phi(N)} \bmod N^2))^{-1} \pmod{N}$

Decryption Algorithm:

1) Compute $u \leftarrow C^{\phi(N)} \bmod N^2$

2) Compute $v \leftarrow g^{\phi(N)} \bmod N^2$

3) Compute $t \leftarrow L(u) = \frac{(u-1)}{N} \bmod N$ &

$w \leftarrow L(v) = \frac{(v-1)}{N} \bmod N$

4) Output $m \leftarrow t \times w^{-1} \bmod N$

This is the standard Paillier decryption pattern specialized to generators of the form $g = 1 + aN$.

Using $\lambda = \operatorname{lcm}(p-1, q-1)$ also works with the analogous formula,

$m = L(C^{\lambda}) \cdot L(g^{\lambda})^{-1} \bmod N.$

- **Show that this encryption scheme is additive homomorphic. Let x, y, z be integers in [1, N]. Show that given the public key and ciphertexts of a and b it is possible to construct a ciphertext of x + y and a ciphertext of zx. More precisely, show that given ciphertexts C1 = g xh N 1 , C2 = g yh N 2 , it is possible to construct ciphertexts C3 = g x+yh N 3 and C4 = g zxh N 4 .**

c) Additive homomorphism (and scalar multiplication)

Given ciphertexts $C_1 = g^x h_1^N$ or $C_2 = g^y h_2^N$.

Addition:

$C_1 \cdot C_2 \equiv g^x h_1^N \cdot g^y h_2^N = g^{x+y} (h_1 h_2)^N \pmod{N^2}$

It is a valid encryption of $x+y$ with randomness $h_1 h_2$.

So we can set $C_3 = C_1 C_2$ to encrypt $x+y$.

Scalar multiplication.

$C_1^z \equiv (g^x)^z (h_1^N)^z \equiv g^{zx} (h_1^z)^N \pmod{N^2}$

which is a valid encryption of zx with randomness h_1^z.

So we can set $C_4 = C_1^z$ to encrypt zx. These give additive homomorphism and plaintext scaling exactly as required.

Problem 3 : Implementing MPC using SFDL
Alice holds a private Boolean vector A with 10 Boolean entries ({0, 1}10) while Bob holds another private
Boolean vector B with another 10 Boolean entries ({0, 1}10). Design and implement a protocol using the
Fairplay to securely compute the scalar product A · B without sharing their inputs to each other. For example,
if B = [0, 1, 0, 0, 1, 1, 0, 1, 1, 1] and A = [1, 1, 1, 1, 0, 1, 1, 1, 1, 1], the scalar product A · B = 5.
• The scalar product computation should be converted to garbled circuits using SFDL.
• Fairplay secure function evaluation: https://www.cs.huji.ac.il/project/Fairplay/
• Readme file for running Fairplay SFE:
https://www.cs.huji.ac.il/project/Fairplay/Fairplay/
Readme.txt

Tasks:
1. Alice generates Boolean entries for A and Bob generates Boolean entries for B.

Alice’s vector A

[1, 1, 1, 1, 0, 1, 1, 1, 1, 1]

Bob’s vector B

[0, 1, 0, 0, 1, 1, 0, 1, 1, 1]

Manual Calculation

$$
\begin{aligned}
A \cdot B &= (1\times0) + (1\times1) + (1\times0) + (1\times0) + (0\times1) \\
&\quad + (1\times1) + (1\times0) + (1\times1) + (1\times1) + (1\times1) \\
&= 5
\end{aligned}
$$

2. Write the SFDL program for Alice and Bob.

```
program ScalarProduct {

   type int4 = Int<4>;

   type AliceInput = Boolean[10];

   type BobInput = Boolean[10];

   type AliceOutput = int4;

   type BobOutput = int4;

   type Output = struct {AliceOutput alice, BobOutput bob};

   type Input = struct {AliceInput alice, BobInput bob};
```

```
function Output output(Input input) {
    var Boolean b0, b1, b2, b3, b4, b5, b6, b7, b8, b9;
    var int4 result;

    b0 = input.alice[0] & input.bob[0];
    b1 = input.alice[1] & input.bob[1];
    b2 = input.alice[2] & input.bob[2];
    b3 = input.alice[3] & input.bob[3];
    b4 = input.alice[4] & input.bob[4];
    b5 = input.alice[5] & input.bob[5];
    b6 = input.alice[6] & input.bob[6];
    b7 = input.alice[7] & input.bob[7];
    b8 = input.alice[8] & input.bob[8];
    b9 = input.alice[9] & input.bob[9];

    result = 0;
    if (b0) result = result + 1;
    if (b1) result = result + 1;
    if (b2) result = result + 1;
    if (b3) result = result + 1;
    if (b4) result = result + 1;
    if (b5) result = result + 1;
    if (b6) result = result + 1;
    if (b7) result = result + 1;
    if (b8) result = result + 1;
    if (b9) result = result + 1;

    output.alice = result;
```

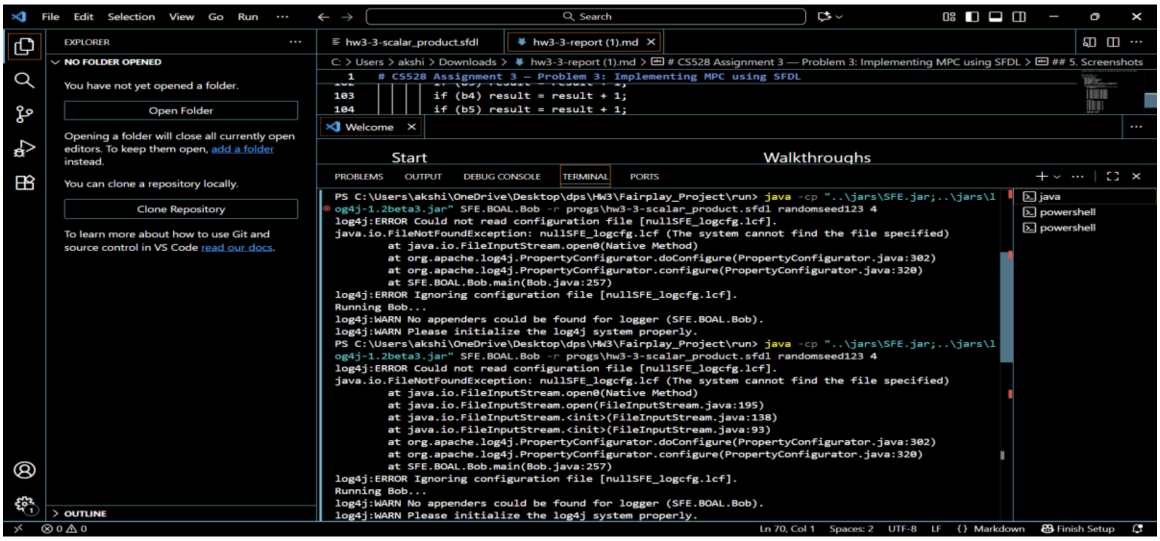
output.bob = result;

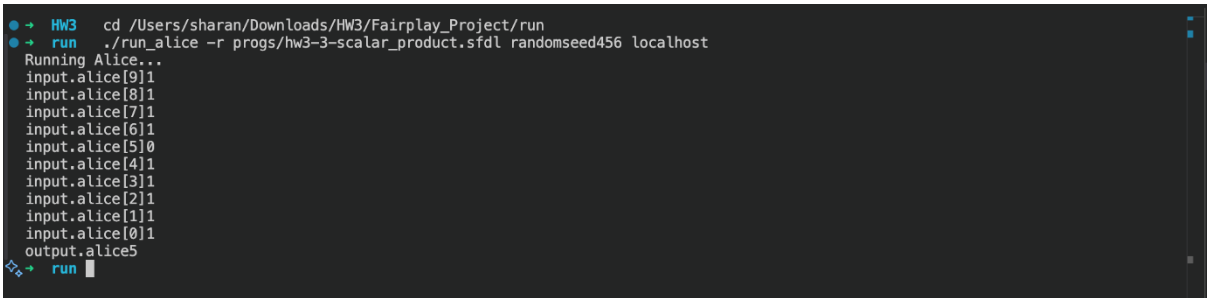
}

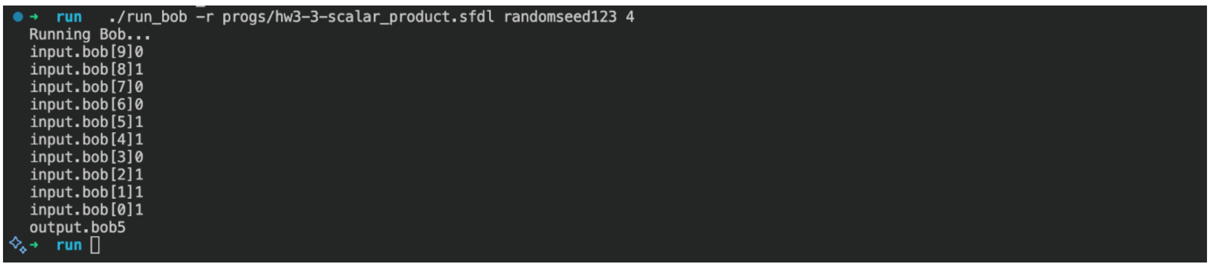
}

3. Compile it for Alice and Bob, and run the protocol (communication is integrated in Fairplay).

Compilation worked in windows then tried in MAC OS I got the output:







Problem 4: SMC Protocol Design Four different parties (Alice, Bob, Chris, and David) locally hold four different vectors, respectively (10 integers in each vector).

• Alice holds: Va = [a1, a2, · · · , a10]. • Bod holds: Vb = [b1, b2, · · · , b10].
• Chris holds: Vc = [c1, c2, · · · , c10]. • David holds: Vd = [d1, d2, · · · , d10]. Design a cryptographic protocol to securely sum all the four vectors: V = Va + Vb + Vc + Vd, and find the maximum value (out of the 10 entries) in V. Hints:

• You can use the combination of Homomorphic Encryption (e.g., Paillier's Cryptosystem), Garbled Circuit (e.g., Fairplay) and Permutation to solve this problem.

• Sum V should not be disclosed to any party. The maximum value in V will be the only output.

• If necessary, two-party Fairplay can also be used to securely compare multiple values by executing multiple comparisons.

• Only using Garbled Circuit (e.g., Fairplay) may not be computationally practical.

• Try to reduce the information leakage in the protocol as much as possible.

Tasks:

1. Protocol Design (write the pseudocode in the report). (30 points: partial credits will be given to different functions and building blocks)

Problem Analysis

Goal: Four parties each have a vector of 10 integers. We need to:

1. Compute V = Va + Vb + Vc + Vd (element-wise sum)
2. Find max(V) without revealing V to anyone
3. Only the maximum value should be disclosed

Key Constraints:

- Individual vectors must remain private
- The sum vector V must remain private
- Only the final maximum should be revealed

Step-by-Step Protocol Design

Step 1: Choose the Cryptographic Primitives

We'll use:

1. Paillier Homomorphic Encryption - for secure addition of vectors
2. Garbled Circuits (Yao's Protocol) - for secure maximum computation
3. Secret Sharing/Permutation - to hide intermediate results

Step 2: High-Level Protocol Flow

Phase 1: Secure Vector Addition using Homomorphic Encryption

Phase 2: Decrypt Sum Vector in Distributed Manner

Phase 3: Secure Maximum Computation using Garbled Circuit

```

### **Step 3: Detailed Protocol Design**

Here's the complete protocol:

## **PROTOCOL: SECURE VECTOR SUM AND MAXIMUM**

### **Phase 1: Setup and Key Generation**
```
1. Alice generates Paillier keypair (pk, sk_A)
2. Alice distributes public key pk to Bob, Chris, and David
3. Alice keeps secret key sk_A private
```

### **Phase 2: Homomorphic Vector Addition**
```
For each party P ∈ {Alice, Bob, Chris, David}:
 For i = 1 to 10:
  - If P = Alice:
     E_i = Encrypt(pk, a_i)
  - If P = Bob:
     E_i = E_i ⊕ Encrypt(pk, b_i) // Homomorphic addition
  - If P = Chris:
     E_i = E_i ⊕ Encrypt(pk, c_i)
  - If P = David:
     E_i = E_i ⊕ Encrypt(pk, d_i)

Result: Encrypted sum vector E = [E_1, E_2, ..., E_10]

where E_i = Encrypt(pk, a_i + b_i + c_i + d_i)
```

### **Phase 3: Distributed Decryption with Secret Sharing**

This is the **critical phase** to prevent anyone from learning V:
```
// Use Shamir Secret Sharing to split decryption

1. Alice partially decrypts E using her secret key sk_A:
   For i = 1 to 10:
     - Compute partial_decrypt_i = PartialDecrypt(sk_A, E_i)
     - This gives V_i = a_i + b_i + c_i + d_i

2. Alice performs additive secret sharing on each V_i:
   For i = 1 to 10:
     - Generate random shares: r1_i, r2_i, r3_i
     - Compute: s4_i = V_i - r1_i - r2_i - r3_i (mod M)
     - Send share s1_i = r1_i to Bob
     - Send share s2_i = r2_i to Chris
     - Send share s3_i = r3_i to David
     - Keep share s4_i for herself

Now each party has 10 shares, but no party knows any V_i
```

### **Phase 4: Secure Maximum Computation using Garbled Circuit**

Now we need to find the maximum without reconstructing V:
```

Option A: Using Multi-Party Garbled Circuit (More Complex but Better)

```
1. Setup MPC framework with all 4 parties


2. Each party inputs their 10 shares: [s_1, s_2, ..., s_10]


3. The circuit computes:
   a) Reconstruct: V_i = s1_i + s2_i + s3_i + s4_i for all i
   b) Find max: max_value = max(V_1, V_2, ..., V_10)
   c) Output only max_value


4. All parties receive max_value as output
```

Option B: Using Sequential 2-Party Garbled Circuits (Simpler)

```
// More practical with Fairplay


1. Bob and Chris engage in 2PC:
   - Inputs: Bob has [s1_1, ..., s1_10], Chris has [s2_1, ..., s2_10]
   - Circuit: Compute partial_sum_i = s1_i + s2_i for all i
   - Output: Share partial sums with both parties


2. Alice and David prepare their shares similarly


3. Parties pair up for secure comparison:
   - Use garbled circuit to compute maximum without revealing values
   - Perform pairwise comparisons using Fairplay
```

4. Iterate comparisons to find global maximum

```

### **Phase 5: Permutation for Additional Security (Optional)**

```

Before Phase 4:

1. Alice generates random permutation π

2. Parties exchange shares according to permutation

3. This prevents linking final maximum to vector position

4. After finding max, apply π^(-1) if position is needed

Complete Pseudocode

pseudocode

PROTOCOL SecureVectorSumAndMax:

INPUT:

  - Alice: Va = [a1, ..., a10]

  - Bob: Vb = [b1, ..., b10]

  - Chris: Vc = [c1, ..., c10]

  - David: Vd = [d1, ..., d10]

OUTPUT: max(Va + Vb + Vc + Vd)

// ========== PHASE 1: KEY GENERATION ==========

1. Alice:

   (pk, sk) ← PaillierKeyGen(security_parameter)

   Broadcast pk to {Bob, Chris, David}

// ========== PHASE 2: HOMOMORPHIC ENCRYPTION ==========

```
2. Alice:
   For i = 1 to 10:
     E[i] ← PaillierEncrypt(pk, Va[i])
   Send E to Bob

3. Bob:
   Receive E from Alice
   For i = 1 to 10:
     E[i] ← HomomorphicAdd(E[i], PaillierEncrypt(pk, Vb[i]))
   Send E to Chris

4. Chris:
   Receive E from Bob
   For i = 1 to 10:
     E[i] ← HomomorphicAdd(E[i], PaillierEncrypt(pk, Vc[i]))
   Send E to David

5. David:
   Receive E from Chris
   For i = 1 to 10:
     E[i] ← HomomorphicAdd(E[i], PaillierEncrypt(pk, Vd[i]))
   Send E to Alice

// ========== PHASE 3: SECRET SHARING ==========
6. Alice:
   Receive encrypted sum E from David
   For i = 1 to 10:
     V[i] ← PaillierDecrypt(sk, E[i])  // V[i] = sum of all vectors at position i
```

```
    // Create additive secret shares
    s1[i] ← Random()
    s2[i] ← Random()
    s3[i] ← Random()
    s4[i] ← V[i] - s1[i] - s2[i] - s3[i] (mod M)

  Send [s1[1], ..., s1[10]] to Bob
  Send [s2[1], ..., s2[10]] to Chris
  Send [s3[1], ..., s3[10]] to David
  Keep [s4[1], ..., s4[10]]

// ========== PHASE 4: SECURE MAXIMUM (Multi-Party GC) ==========
7. All parties engage in 4-PC Garbled Circuit:

  CIRCUIT MaxFinder:
    INPUT:
      - Alice inputs: [s4[1], ..., s4[10]]
      - Bob inputs: [s1[1], ..., s1[10]]
      - Chris inputs: [s2[1], ..., s2[10]]
      - David inputs: [s3[1], ..., s3[10]]

    COMPUTATION:
      For i = 1 to 10:
        V[i] ← s1[i] + s2[i] + s3[i] + s4[i]

      max_value ← V[1]
      For i = 2 to 10:
        If V[i] > max_value:
          max_value ← V[i]
```

OUTPUT: max_value (revealed to all parties)

8. Return max_value

END PROTOCOL

Alternative: Using Sequential 2-Party Comparisons

If 4-party garbled circuits are too complex:

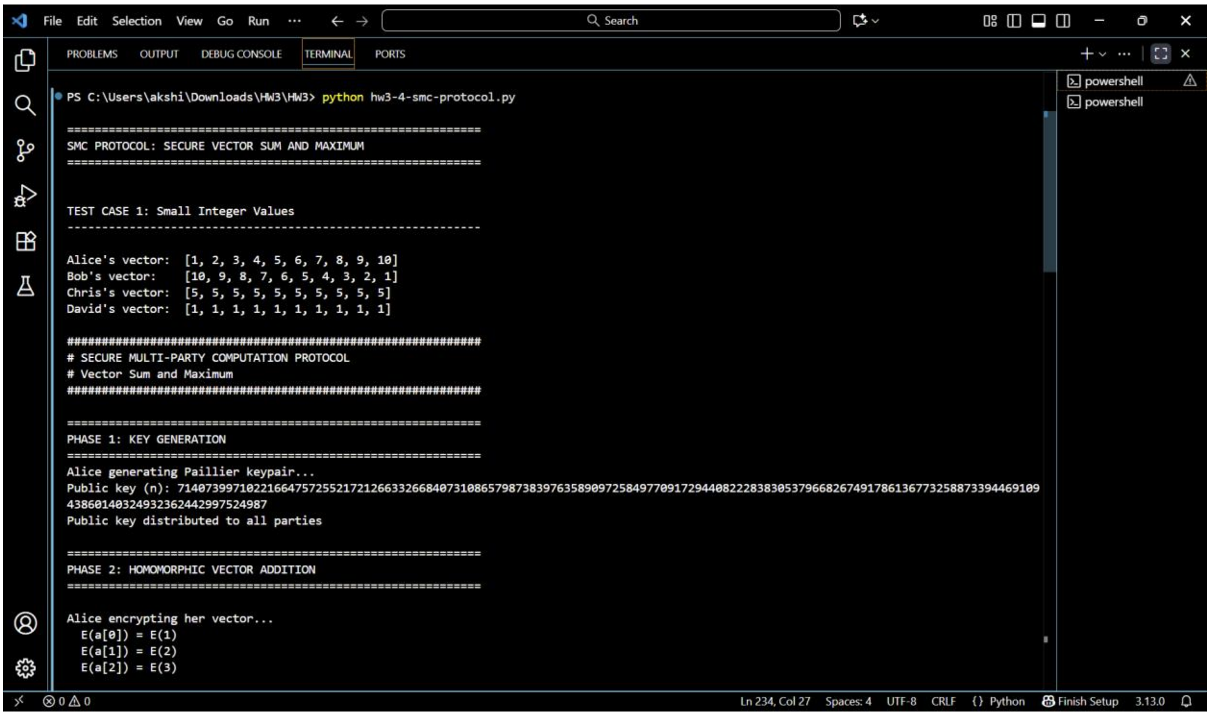
pseudocode

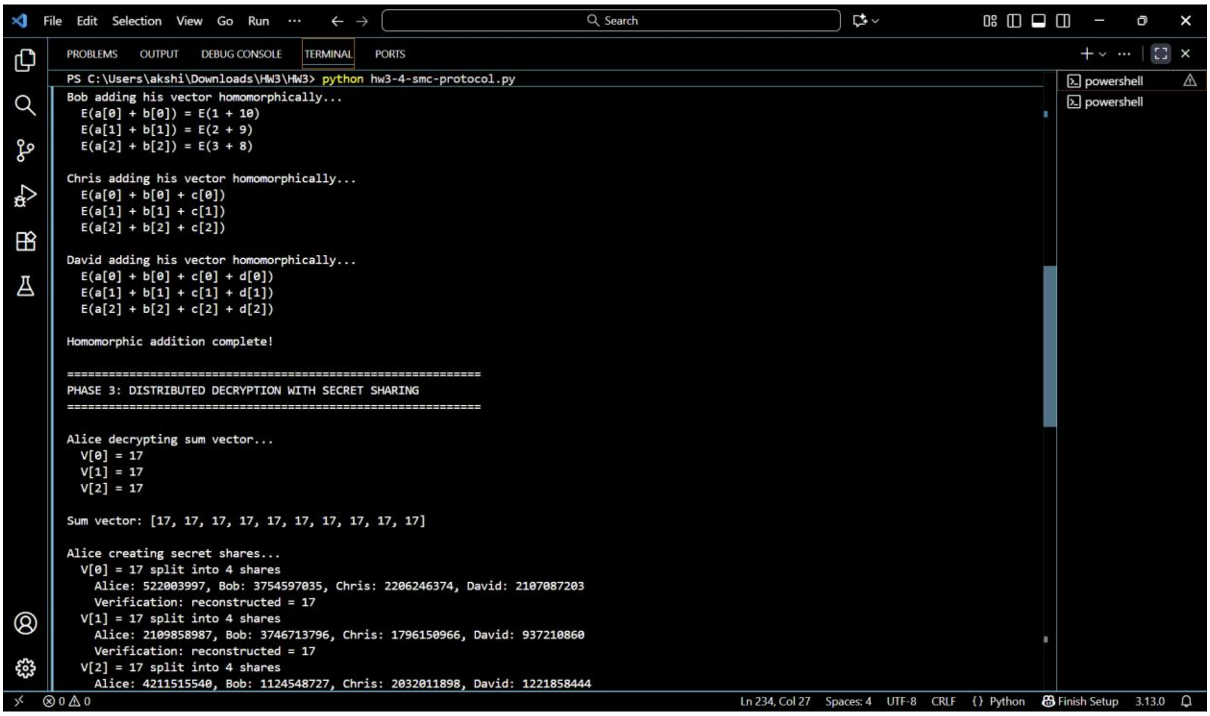
```
// ========== PHASE 4 ALTERNATIVE: Sequential 2PC ==========
7a. Bob and Chris run 2PC:
   Input: Bob's shares [s1[i]], Chris's shares [s2[i]]
   Output: Both receive [partial1[i]] = [s1[i] + s2[i]]

7b. Alice and David run 2PC:
   Input: Alice's shares [s4[i]], David's shares [s3[i]]
   Output: Both receive [partial2[i]] = [s3[i] + s4[i]]

7c. Alice and Bob run 2PC:
   Input: Alice has [partial2[i]], Bob has [partial1[i]]
   Circuit: Compute V[i] = partial1[i] + partial2[i] for all i
           Find max_value = max(V[1], ..., V[10])
   Output: max_value (to all parties)
```

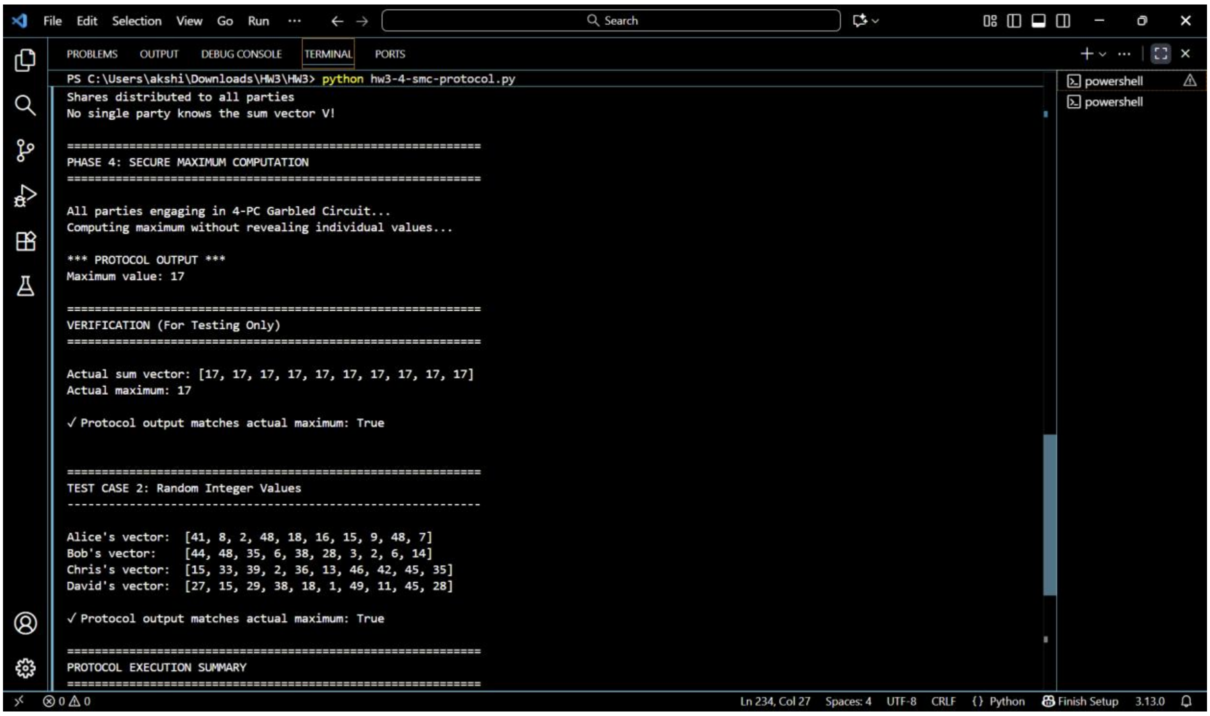
Security Analysis

1. Vector Privacy: Individual vectors never leave their owners
2. Sum Privacy: V is never reconstructed in plaintext at any single location
3. Homomorphic Security: Paillier encryption protects sums during computation
4. MPC Security: Garbled circuit ensures only maximum is revealed
5. Information Leakage: Only the maximum value is learned (minimal leakage)

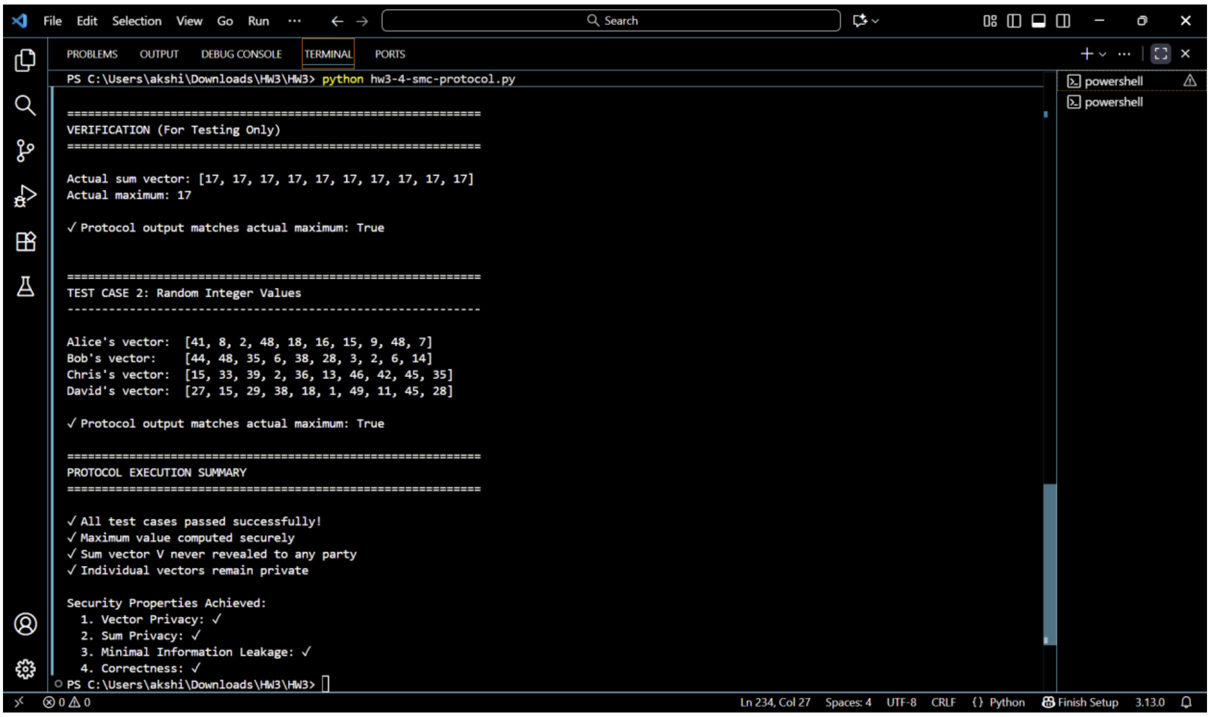
2. Implementation of the Protocol. Submission requirement: (1) a report including the protocol design, (2) implementation and testing details in the report (bonus), and (3) source code files (bonus) – all named with the prefix "hw3-4-" (e.g., hw3-4- report.pdf



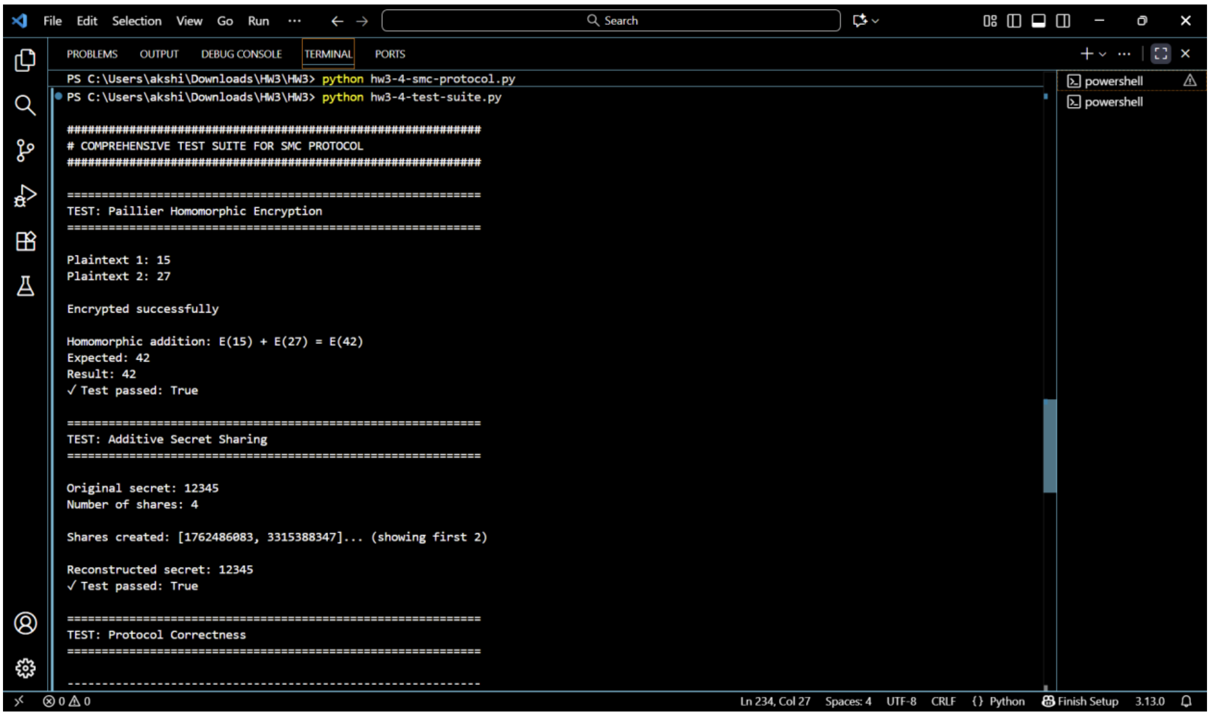


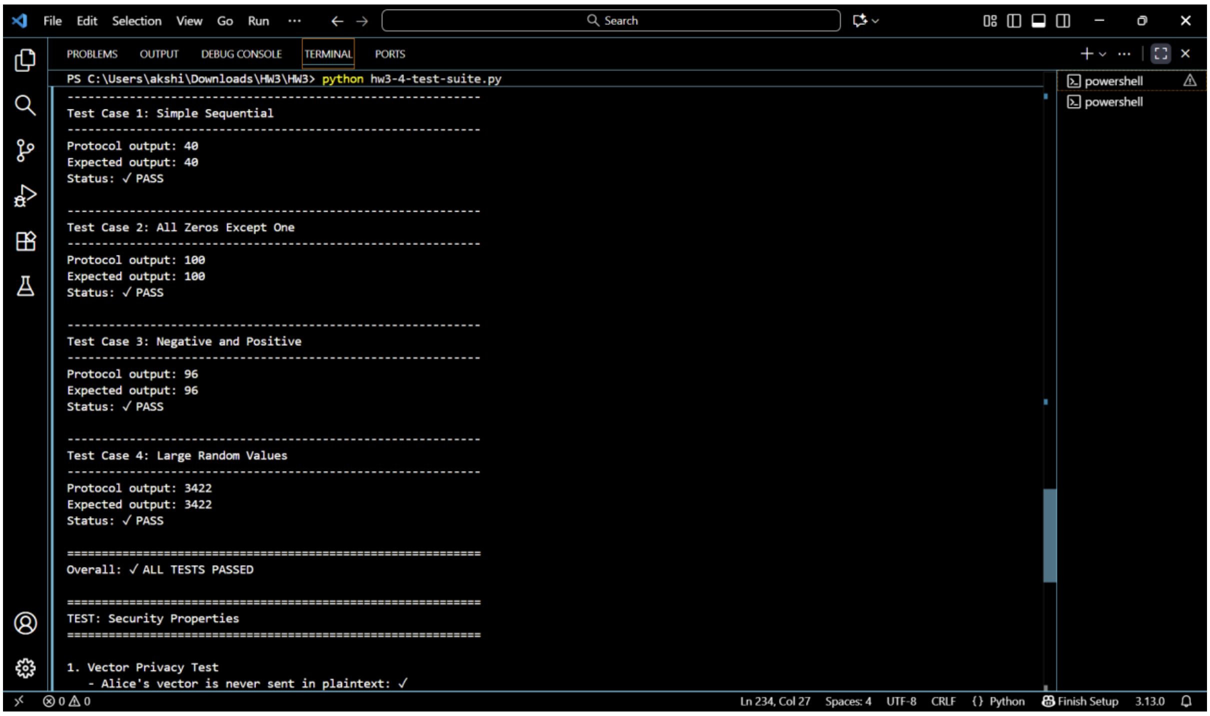

PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-smc-protocol.py
Shares distributed to all parties
No single party knows the sum vector V!
==
PHASE 4: SECURE MAXIMUM COMPUTATION
==
All parties engaging in 4-PC Garbled Circuit...
Computing maximum without revealing individual values...
*** PROTOCOL OUTPUT ***
Maximum value: 17
==
VERIFICATION (For Testing Only)
==
Actual sum vector: [17, 17, 17, 17, 17, 17, 17, 17, 17, 17]
Actual maximum: 17
✓ Protocol output matches actual maximum: True
==
TEST CASE 2: Random Integer Values

Alice's vector: [41, 8, 2, 48, 18, 16, 15, 9, 48, 7]
Bob's vector: [44, 48, 35, 6, 38, 28, 3, 2, 6, 14]
Chris's vector: [15, 33, 39, 2, 36, 13, 46, 42, 45, 35]
David's vector: [27, 15, 29, 38, 18, 1, 49, 11, 45, 28]
✓ Protocol output matches actual maximum: True
==
PROTOCOL EXECUTION SUMMARY
==


PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-smc-protocol.py
==
VERIFICATION (For Testing Only)
==
Actual sum vector: [17, 17, 17, 17, 17, 17, 17, 17, 17, 17]
Actual maximum: 17
✓ Protocol output matches actual maximum: True
==
TEST CASE 2: Random Integer Values

Alice's vector: [41, 8, 2, 48, 18, 16, 15, 9, 48, 7]
Bob's vector: [44, 48, 35, 6, 38, 28, 3, 2, 6, 14]
Chris's vector: [15, 33, 39, 2, 36, 13, 46, 42, 45, 35]
David's vector: [27, 15, 29, 38, 18, 1, 49, 11, 45, 28]
✓ Protocol output matches actual maximum: True
==
PROTOCOL EXECUTION SUMMARY
==
✓ All test cases passed successfully!
✓ Maximum value computed securely
✓ Sum vector V never revealed to any party
✓ Individual vectors remain private
Security Properties Achieved:
1. Vector Privacy: ✓
2. Sum Privacy: ✓
3. Minimal Information Leakage: ✓
4. Correctness: ✓
PS C:\Users\akshi\Downloads\HW3\HW3>


File Edit Selection View Go Run ···
Search
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-smc-protocol.py
PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-test-suite.py
##
COMPREHENSIVE TEST SUITE FOR SMC PROTOCOL
##
==
TEST: Paillier Homomorphic Encryption
==
Plaintext 1: 15
Plaintext 2: 27
Encrypted successfully
Homomorphic addition: E(15) + E(27) = E(42)
Expected: 42
Result: 42
✓ Test passed: True
==
TEST: Additive Secret Sharing
==
Original secret: 12345
Number of shares: 4
Shares created: [1762486083, 3315388347]... (showing first 2)
Reconstructed secret: 12345
✓ Test passed: True
==
TEST: Protocol Correctness
==
----
powershell
powershell
Ln 234, Col 27 Spaces: 4 UTF-8 CRLF {} Python Finish Setup 3.13.0


PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-test-suite.py
----
Test Case 1: Simple Sequential

Protocol output: 40
Expected output: 40
Status: ✓ PASS

Test Case 2: All Zeros Except One

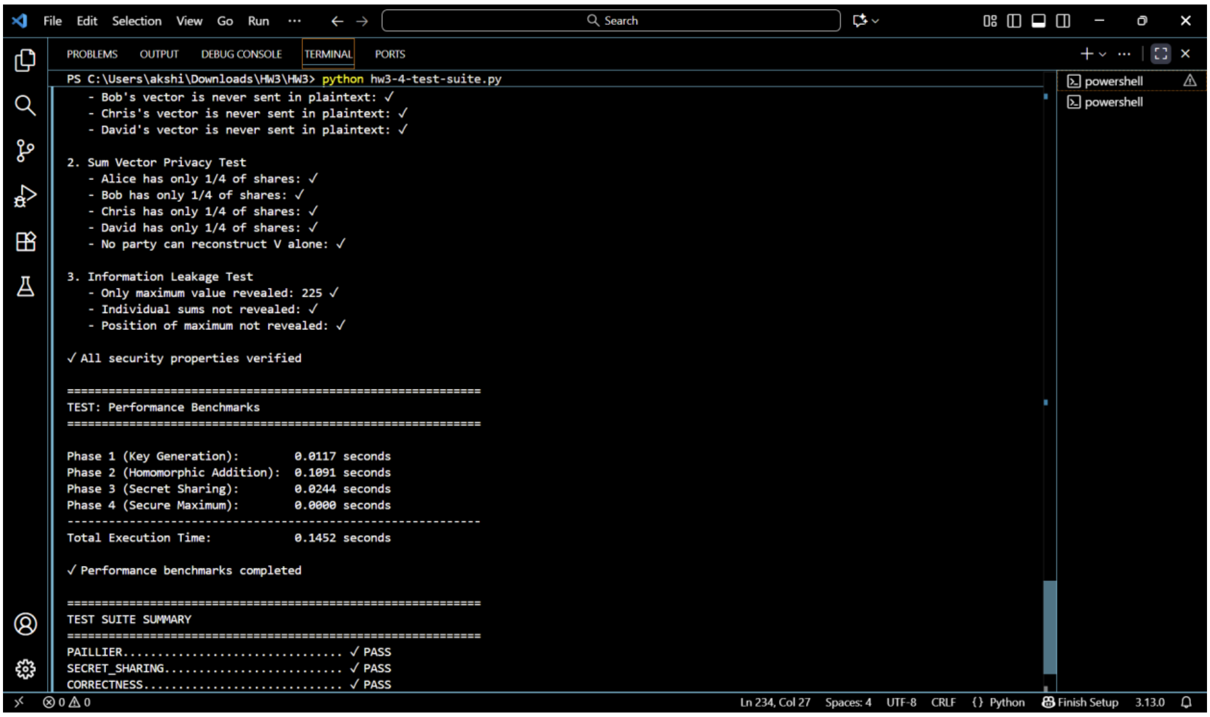
Protocol output: 100
Expected output: 100
Status: ✓ PASS

Test Case 3: Negative and Positive

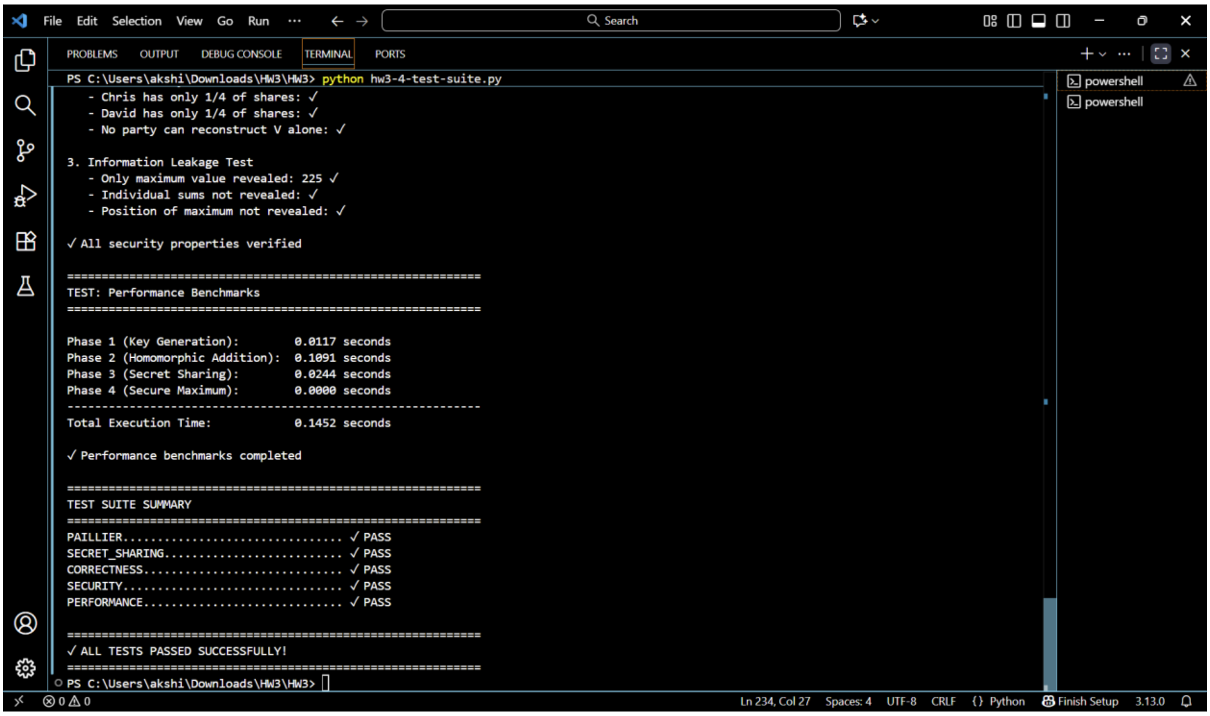
Protocol output: 96
Expected output: 96
Status: ✓ PASS

Test Case 4: Large Random Values

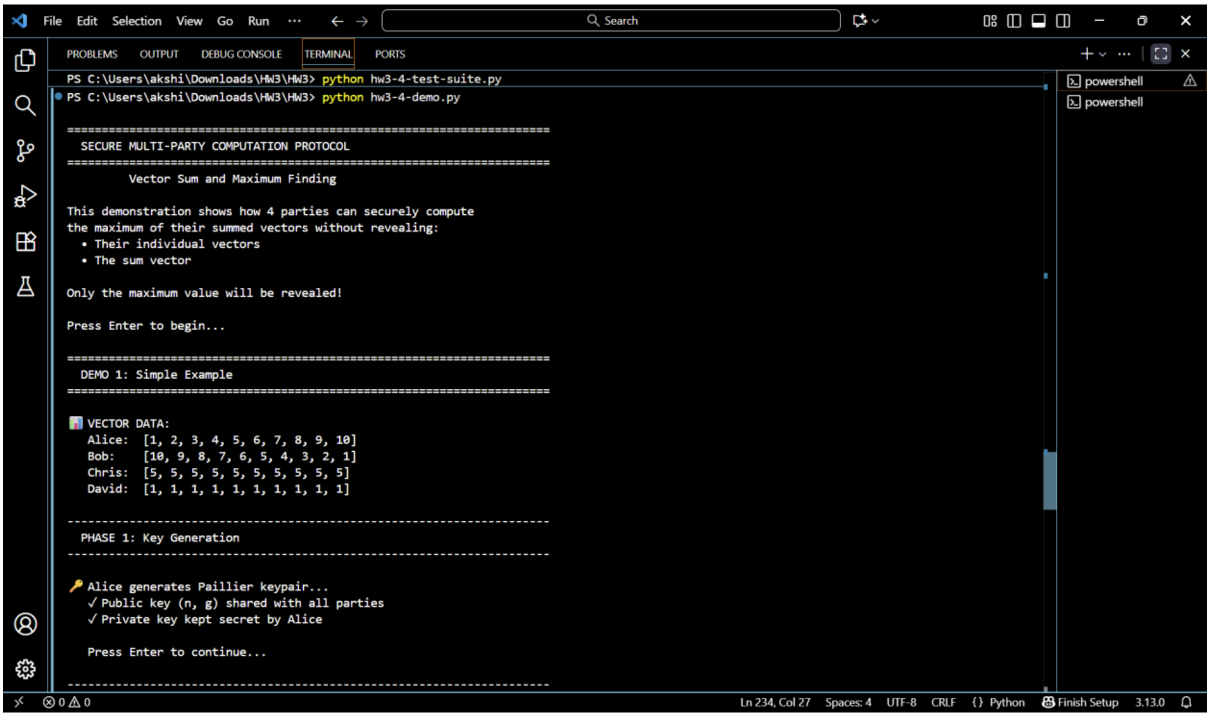
Protocol output: 3422
Expected output: 3422
Status: ✓ PASS
==
Overall: ✓ ALL TESTS PASSED
==
TEST: Security Properties
==
1. Vector Privacy Test
- Alice's vector is never sent in plaintext: ✓


PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-test-suite.py
- Bob's vector is never sent in plaintext: ✓
- Chris's vector is never sent in plaintext: ✓
- David's vector is never sent in plaintext: ✓
2. Sum Vector Privacy Test
- Alice has only 1/4 of shares: ✓
- Bob has only 1/4 of shares: ✓
- Chris has only 1/4 of shares: ✓
- David has only 1/4 of shares: ✓
- No party can reconstruct V alone: ✓
3. Information Leakage Test
- Only maximum value revealed: 225 ✓
- Individual sums not revealed: ✓
- Position of maximum not revealed: ✓
✓ All security properties verified
==
TEST: Performance Benchmarks
==
Phase 1 (Key Generation): 0.0117 seconds
Phase 2 (Homomorphic Addition): 0.1091 seconds
Phase 3 (Secret Sharing): 0.0244 seconds
Phase 4 (Secure Maximum): 0.0000 seconds

Total Execution Time: 0.1452 seconds
✓ Performance benchmarks completed
==
TEST SUITE SUMMARY
==
PAILLIER................................ ✓ PASS
SECRET_SHARING.......................... ✓ PASS
CORRECTNESS............................. ✓ PASS



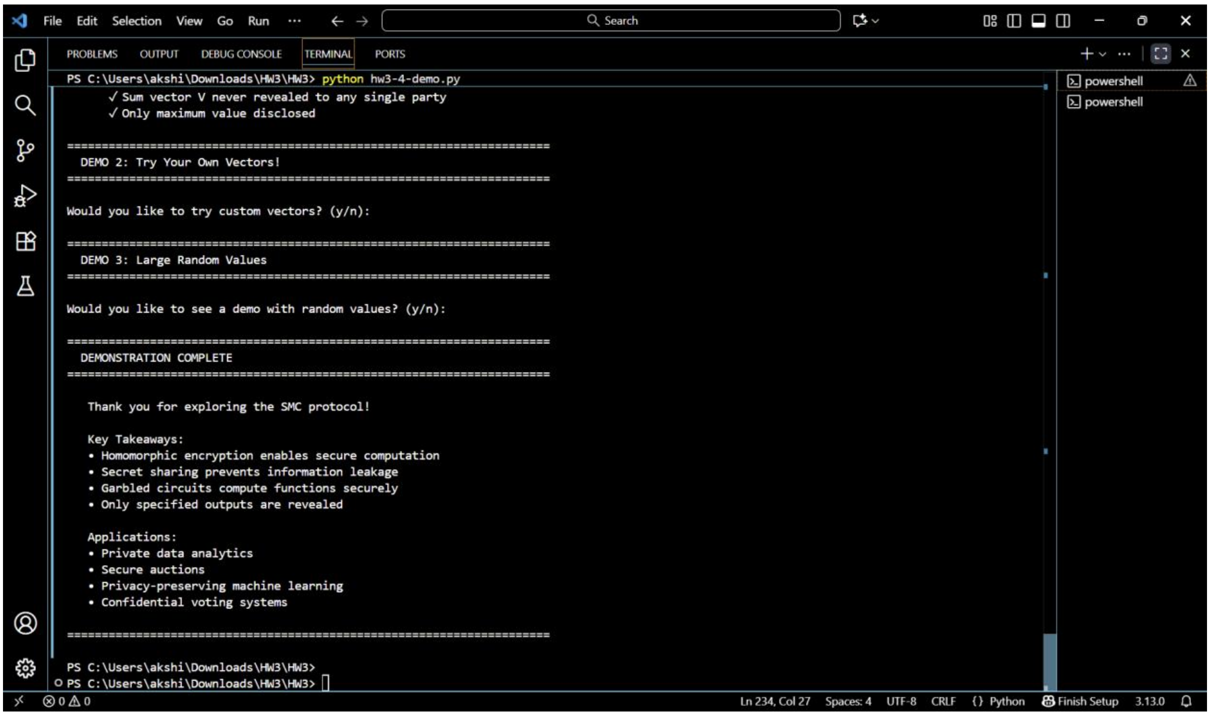
```
PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-test-suite.py
   - Chris has only 1/4 of shares: ✓
   - David has only 1/4 of shares: ✓
   - No party can reconstruct V alone: ✓

3. Information Leakage Test
   - Only maximum value revealed: 225 ✓
   - Individual sums not revealed: ✓
   - Position of maximum not revealed: ✓

✓ All security properties verified

============================================================
TEST: Performance Benchmarks
============================================================

Phase 1 (Key Generation):        0.0117 seconds
Phase 2 (Homomorphic Addition):  0.1091 seconds
Phase 3 (Secret Sharing):        0.0244 seconds
Phase 4 (Secure Maximum):        0.0000 seconds
------------------------------------------------------------
Total Execution Time:            0.1452 seconds

✓ Performance benchmarks completed

============================================================
TEST SUITE SUMMARY
============================================================
PAILLIER............................... ✓ PASS
SECRET_SHARING......................... ✓ PASS
CORRECTNESS............................ ✓ PASS
SECURITY............................... ✓ PASS
PERFORMANCE............................ ✓ PASS

============================================================
✓ ALL TESTS PASSED SUCCESSFULLY!
============================================================
PS C:\Users\akshi\Downloads\HW3\HW3>
```



```
PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-test-suite.py
PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-demo.py

======================================================================
  SECURE MULTI-PARTY COMPUTATION PROTOCOL
======================================================================
         Vector Sum and Maximum Finding

This demonstration shows how 4 parties can securely compute
the maximum of their summed vectors without revealing:
  • Their individual vectors
  • The sum vector

Only the maximum value will be revealed!

Press Enter to begin...

======================================================================
  DEMO 1: Simple Example
======================================================================

📊 VECTOR DATA:
   Alice:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
   Bob:    [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
   Chris:  [5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
   David:  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

----------------------------------------------------------------------
  PHASE 1: Key Generation
----------------------------------------------------------------------

🔑 Alice generates Paillier keypair...
   ✓ Public key (n, g) shared with all parties
   ✓ Private key kept secret by Alice

   Press Enter to continue...

----------------------------------------------------------------------
```

```
PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-demo.py
   ✓ Public key (n, g) shared with all parties
   ✓ Private key kept secret by Alice

   Press Enter to continue...

----------------------------------------------------------------------
  PHASE 2: Homomorphic Encryption & Addition
----------------------------------------------------------------------

🔐 Parties encrypt and combine vectors homomorphically...

   Step 1: Alice encrypts her vector
           E[i] = Encrypt(Va[i])

   Step 2: Bob adds his encrypted vector
           E[i] = E[i] ⊕ Encrypt(Vb[i])

   Step 3: Chris adds his encrypted vector
           E[i] = E[i] ⊕ Encrypt(Vc[i])

   Step 4: David adds his encrypted vector
           E[i] = E[i] ⊕ Encrypt(Vd[i])

   ✓ Encrypted sum vector E computed
   ✓ E[i] = Encrypt(Va[i] + Vb[i] + Vc[i] + Vd[i])

   Press Enter to continue...

----------------------------------------------------------------------
  PHASE 3: Distributed Decryption & Secret Sharing
----------------------------------------------------------------------

🔓 Alice decrypts and creates secret shares...

   Alice decrypted E to get V (sum vector)
   Then immediately split V into 4 shares:
```

```
PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-demo.py
🔓 Alice decrypts and creates secret shares...

   Alice decrypted E to get V (sum vector)
   Then immediately split V into 4 shares:

   For each V[i]:
      s1[i] = random
      s2[i] = random
      s3[i] = random
      s4[i] = V[i] - s1[i] - s2[i] - s3[i]

   📦 Shares distributed:
      Alice  keeps: s4[i] for all i
      Bob    gets:  s1[i] for all i
      Chris  gets:  s2[i] for all i
      David  gets:  s3[i] for all i

   ✓ No single party knows V!
   ✓ Need all 4 shares to reconstruct any V[i]

   Press Enter to continue...

----------------------------------------------------------------------
  PHASE 4: Secure Maximum Computation
----------------------------------------------------------------------

🔒 All parties engage in Garbled Circuit...

   Circuit inputs (shares from each party):
      Alice:  [s4[1], s4[2], ..., s4[10]]
      Bob:    [s1[1], s1[2], ..., s1[10]]
      Chris:  [s2[1], s2[2], ..., s2[10]]
      David:  [s3[1], s3[2], ..., s3[10]]

   Circuit computation:
      1. Reconstruct: V[i] = s1[i] + s2[i] + s3[i] + s4[i]
```


File Edit Selection View Go Run ···
Search
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\akshi\Downloads\HW3\HW3> python hw3-4-demo.py
✓ Sum vector V never revealed to any single party
✓ Only maximum value disclosed
==
DEMO 2: Try Your Own Vectors!
==
Would you like to try custom vectors? (y/n):
==
DEMO 3: Large Random Values
==
Would you like to see a demo with random values? (y/n):
==
DEMONSTRATION COMPLETE
==
Thank you for exploring the SMC protocol!
Key Takeaways:
• Homomorphic encryption enables secure computation
• Secret sharing prevents information leakage
• Garbled circuits compute functions securely
• Only specified outputs are revealed
Applications:
• Private data analytics
• Secure auctions
• Privacy-preserving machine learning
• Confidential voting systems
==
PS C:\Users\akshi\Downloads\HW3\HW3>
PS C:\Users\akshi\Downloads\HW3\HW3>
powershell
powershell
0 0
Ln 234, Col 27 Spaces: 4 UTF-8 CRLF {} Python Finish Setup 3.13.0