HW3-4: Secure Multi-Party Computation Protocol

Finding Maximum of Summed Vectors without Revealing the Sum

Student: Mohamed Trigui
Course: Data Privacy
Assignment: HW3-4 — SMC Protocol Design

October 16, 2025

1 Problem Statement

1.1 Inputs

Four parties hold private integer vectors:

- Alice holds $V_a = [a_1, \ldots, a_{10}]$
- Bob holds $V_b = [b_1, \ldots, b_{10}]$
- Chris holds $V_c = [c_1, \ldots, c_{10}]$
- David holds $V_d = [d_1, \ldots, d_{10}]$

1.2 Goal

$$V = V_a + V_b + V_c + V_d, \qquad \texttt{max\_value} = \max\{V[1], \ldots, V[10]\}.$$

1.3 Security Requirements

1. Output: only `max_value` is revealed to all parties.
2. The sum vector V remains secret.
3. Input privacy: V_a, V_b, V_c, V_d remain private.
4. Minimize leakage during computation.

2 Protocol Overview

2.1 Key Insight

We must securely sum vectors, find the maximum without revealing the sum vector, and combine techniques: additive homomorphic encryption (Paillier), random permutation, and garbled circuits.

2.2 Protocol Architecture

```
Phase 1: Homomorphic Sum
 Use Paillier to compute encrypted sums Enc(V[i])

Phase 2: Permutation for Privacy
 Randomly permute Enc(V[i]) to break index-value linkage

Phase 3: Distributed Decryption
 Collaboratively (or centrally) decrypt permuted values

Phase 4: Maximum Finding
 Use garbled circuits to reveal only max_value
```

3 Detailed Protocol Design

3.1 Setup Phase

Participants: the four data holders and a coordinator (can be any party). All agree on security parameters.

Initialization:

1. Alice generates Paillier key pair (pk, sk).
2. Alice distributes pk to Bob, Chris, David; all verify pk.

Rationale: Paillier provides additive homomorphism: $\text{Enc}(a) \cdot \text{Enc}(b) = \text{Enc}(a + b)$.

3.2 Phase 1: Homomorphic Vector Summation

Objective: compute $\text{Enc}(V[i])$ with $V[i] = a_i + b_i + c_i + d_i$.

Listing 1: Homomorphic summation

```
# Alice (key owner)
for i in 1..10:
    enc_alice[i] = Encrypt(pk, Va[i])
send enc_alice -> Bob

# Bob
for i in 1..10:
    enc_bob[i] = Encrypt(pk, Vb[i])
    enc_ab[i]  = enc_alice[i] * enc_bob[i]
send enc_ab -> Chris

# Chris
for i in 1..10:
    enc_chris[i] = Encrypt(pk, Vc[i])
    enc_abc[i]   = enc_ab[i] * enc_chris[i]
send enc_abc -> David

# David
for i in 1..10:
```

```
    enc_david[i] = Encrypt(pk, Vd[i])
    enc_sum[i]   = enc_abc[i] * enc_david[i]
# enc_sum[i] = Enc(Va[i] + Vb[i] + Vc[i] + Vd[i])
```

Result: David holds $\{\text{Enc}(V[1]), \ldots, \text{Enc}(V[10])\}$.

3.3 Phase 2: Random Permutation

Objective: hide index-value correspondence.

Listing 2: Permutation step

```
# David
pi = RandomPermutation({1..10})
for i in 1..10:
    enc_perm[i] = enc_sum[pi(i)]
send enc_perm -> Alice
```

Permutation π is known only to David, destroying positional linkage.

3.4 Phase 3: Collaborative Decryption

Goal: decrypt without revealing V's structure.

Listing 3: Threshold-style outline

Option A: Secret sharing / threshold decryption (preferred).

```
# Each party holds a key share; combine partial decryptions
for i in 1..10:
    share_alice[i] = PartialDecrypt(sk_alice, enc_perm[i])
    share_bob[i]   = PartialDecrypt(sk_bob,   enc_perm[i])
    share_chris[i] = PartialDecrypt(sk_chris, enc_perm[i])
    share_david[i] = PartialDecrypt(sk_david, enc_perm[i])
    V_perm[i] = Combine(share_alice[i], share_bob[i],
                        share_chris[i], share_david[i])
```

Listing 4: Simple decryption

Option B: Simple centralized decryption (if acceptable).

```
# Alice decrypts permuted ciphertexts (order unknown to her)
for i in 1..10:
    V_perm[i] = Decrypt(sk, enc_perm[i])
broadcast V_perm to all
```

All parties obtain $V_{\text{perm}} = [V_{\pi(1)}, \ldots, V_{\pi(10)}]$.

3.5 Phase 4: Secure Maximum Finding

Objective: compute $\max(V_{\text{perm}})$ via garbled circuits.

Listing 5: Tournament via GC

Method 1: Centralized (simpler).

```
# Alice & Bob run Fairplay GC comparisons
max_1 = GC_Max(V_perm[1], V_perm[2])
max_2 = GC_Max(V_perm[3], V_perm[4])
max_3 = GC_Max(V_perm[5], V_perm[6])
max_4 = GC_Max(V_perm[7], V_perm[8])
max_5 = GC_Max(V_perm[9], V_perm[10])

max_12   = GC_Max(max_1, max_2)
max_34   = GC_Max(max_3, max_4)
max_1234 = GC_Max(max_12, max_34)
final_max = GC_Max(max_1234, max_5)
```

Listing 6: Distributed comparisons

Method 2: Distributed (more balanced).

```
max_AB = Fairplay_FindMax(Alice: V_perm[1..5], Bob:   V_perm[1..5])
max_CD = Fairplay_FindMax(Chris: V_perm[6..10], David: V_perm[6..10])
final_max = Fairplay_Max(Alice: max_AB, Chris: max_CD)
```

Complexity: 9 pairwise comparisons (tournament). Each comparison uses $O(\log n)$ gates for n-bit integers.

4 Complete Protocol Pseudocode

```
PROTOCOL: SecureMaxOfSums
INPUT: Alice(Va), Bob(Vb), Chris(Vc), David(Vd)
OUTPUT: max(Va + Vb + Vc + Vd)

# PHASE 1: SETUP
Alice:
    (pk, sk) = PaillierKeyGen(security_param)
    broadcast pk

# PHASE 2: HOMOMORPHIC SUM
... (as in Phase 1 above)

# PHASE 3: PERMUTATION
David:
    pi = RandomPermutation(1..10)
    E_perm = Permute(E_sum, pi)
    send E_perm -> Alice

# PHASE 4: DECRYPTION (Option A or B)
Alice (Option B):
    V_perm = [Decrypt(sk, c) for c in E_perm]
    broadcast V_perm

# PHASE 5: SECURE MAXIMUM
def TournamentMax(values):
    if len(values) == 1: return values[0]
    if len(values) == 2: return GarbledCircuitMax(values[0], values[1])
```

```
    mid = len(values)//2
    return GarbledCircuitMax(
        TournamentMax(values[:mid]),
        TournamentMax(values[mid:])
    )

All:
    max_value = TournamentMax(V_perm)
    output max_value
```

5 Security Analysis

5.1 Privacy Guarantees

- Phase 1: inputs remain encrypted; Paillier's semantic security protects values.
- Phase 2: permutation breaks index–value linkage.
- Phase 3: decryption reveals values but not positions; threshold variant avoids single-party advantage.
- Phase 4: only the maximum is revealed; GC hides intermediate comparison bits.

5.2 Information Leakage

Revealed: the final maximum value (intended). If using the simple variant, the permuted values become public. Hidden: individual inputs, original V, positional correspondence, and GC internals. Leakage can be minimized with threshold decryption and GC over encrypted encodings.

5.3 Adversary Model

Semi-honest parties. Potential collusion risk: David (knows π) + Alice (decrypts) could reconstruct V. Mitigation: threshold decryption and distributed permutation.

6 Optimization and Variants

6.1 Enhanced Privacy Variant

Use threshold Paillier (e.g., 3-of-4):

```
(pk, {sk_i}) = ThresholdPaillierKeyGen(4, 3)
# Homomorphic sum as before
# Decrypt via PartialDecrypt + CombineShares
```

6.2 Efficiency Improvements

Batch encryptions and parallel GC sessions:

```
E_alice = PaillierEncryptVector(pk, Va)  # batching
# Run multiple GC_Max comparisons in parallel
```

6.3 Alternative: Pure MPC

A pure Fairplay approach requires 4-party MPC or chained 2-party MPC; circuits grow large for 40 additions and maximum.

7 Complexity Analysis

7.1 Computation

Per party: $O(10)$ encryptions, $O(10)$ homomorphic multiplications; decryption $O(10)$ for decrypting party; tournament $O(9)$ GC comparisons.

7.2 Communication

- Rounds: ≈ 11 (setup, sum, permutation, decrypt, tournament).
- Messages: $O(10)$ ciphertext transfers across phases; $O(\log 10)$ GC exchanges.

7.3 Comparison

Approach	Computation	Communication	Privacy
Hybrid (ours)	$O(10n)$	$O(10) + O(\log 10)$	Good
Pure Fairplay	Large circuits	Fewer rounds	Best
Trusted Third Party	Minimal	Minimal	None

8 Implementation Considerations

8.1 Libraries and Tools

Paillier: Python `phe`, or Java libraries (2048-bit keys). Garbled circuits: Fairplay, EMP-toolkit, SCALE-MAMBA. Secure channels (TLS) and MACs for transport.

8.2 Testing Strategy

```
# Test 1
Va=[1,2,3,4,5,6,7,8,9,10]
Vb=[1]*10; Vc=[0]*10; Vd=[0]*10
# Expected max = 11

# Test 2
Va=[5]*10; Vb=[3]*10; Vc=[2]*10; Vd=[0]*10
# Expected max = 10

# Test 3 (negatives)
Va=[-5,-3,10,2,1,0,4,6,8,9]
Vb=[5,3,-2,-1,0,1,2,3,4,5]
Vc=[0]*10; Vd=[0]*10
# Expected max = 14
```

9 Conclusion

9.1 Summary

A hybrid SMC protocol that securely sums private vectors via Paillier, hides structure through permutation, and reveals only the maximum via garbled circuits. It minimizes leakage and distributes trust.

9.2 Key Innovations

Hybrid HE+GC design, permutation layer to hide V, and options for threshold decryption and distributed comparisons.

9.3 Limitations and Future Work

Single-point permutation knowledge (David), central decryption in simple variant, sequential homomorphic passes. Future: threshold decryption, distributed permutation, parallelization, and malicious security via ZK proofs.

10 References

1. Paillier, P. (1999). *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes.* EUROCRYPT.

2. Yao, A. C. (1986). *How to Generate and Exchange Secrets.* IEEE FOCS.

3. Damgård, I., & Jurik, M. (2001). *A Generalisation and Some Applications of Paillier.* PKC.

4. Lindell, Y., & Pinkas, B. (2009). *Secure Multiparty Computation for Privacy-Preserving Data Mining.* JPC.

5. Malkhi, D., Nisan, N., Pinkas, B., & Sella, Y. (2004). *Fairplay.* USENIX Security.