



Commands + Code + Text ▶ Run all ▾

Connect ▾



```
[ ] # -----  
# Part I: MPC Sum using Additive Secret Sharing  
# -----  
  
import secrets  
  
# Private values (example given in assignment: total = 220)  
values = {  
    "Ahmed": 100,  
    "Ali": 70,  
    "Rashida": 50  
}  
  
participants = list(values.keys())  
n = len(participants)  
  
# 1) Each participant splits their private value into random shares  
shares = {}  
for owner, val in values.items():  
    s1 = secrets.randrange(200)  
    s2 = secrets.randrange(200)  
    s3 = val - (s1 + s2)  
    shares[owner] = [s1, s2, s3]  
  
print("Shares (hidden in real MPC):")  
for owner in shares:  
    print(owner, shares[owner], "sum =", sum(shares[owner]))  
print()  
  
# 2) Distribute shares & each participant sums received shares  
partial_sums = {p: 0 for p in participants}  
  
for i, receiver in enumerate(participants):  
    collected = [shares[owner][i] for owner in participants]  
    partial_sums[receiver] = sum(collected)  
    print(f"{receiver} received {collected} -> partial sum = {partial_sums[receiver]}")  
print()  
  
# 3) Global sum = sum of partial sums  
global_sum = sum(partial_sums.values())  
print("Reconstructed global sum:", global_sum)  
  
# 4) Check correctness  
assert global_sum == sum(values.values())  
print("✓ Correct total =", global_sum, "(No one revealed private values!)")
```



```
... Shares (hidden in real MPC):  
Ahmed [90, 106, -96] sum = 100  
Ali [90, 164, -184] sum = 70  
Rashida [126, 133, -209] sum = 50  
  
Ahmed received [90, 90, 126] -> partial sum = 306  
Ali received [106, 164, 133] -> partial sum = 403  
Rashida received [-96, -184, -209] -> partial sum = -489  
  
Reconstructed global sum: 220  
✓ Correct total = 220 (No one revealed private values!)
```



```
[ ] # -----  
# Part II: Shamir's Secret Sharing (k, n)  
# -----  
  
import random  
from sympy import symbols, expand  
  
# Parameters  
secret = 1234 # The secret to protect  
k = 3 # Threshold
```

```

k = 3                      # threshold
n = 5                      # Number of shares
prime = 2089                # A prime > secret

# Generate random polynomial coefficients (degree k-1)
coeffs = [secret] + [random.randint(1, prime-1) for _ in range(k-1)]

def f(x):
    """Evaluate polynomial at x."""
    total = 0
    for i, c in enumerate(coeffs):
        total += c * (x**i)
    return total % prime

# Generate shares
shares = [(i, f(i)) for i in range(1, n+1)]
print("Shares (x, y):")
for s in shares:
    print(s)

# Reconstruction with Lagrange Interpolation
def reconstruct(selected_shares):
    total = 0
    for i, (x_i, y_i) in enumerate(selected_shares):
        li = 1
        for j, (x_j, _) in enumerate(selected_shares):
            if i != j:
                li *= (x_j * pow(x_j - x_i, -1, prime)) % prime
        total = (total + y_i * li) % prime
    return total

print("\nReconstructed secret from any 3 shares:")
rec = reconstruct(shares[:3])
print("Recovered secret =", rec)
print("✓ Correct!" if rec == secret else "✗ Error")

```

... Shares (x, y):
(1, 58)
(2, 140)
(3, 1480)
(4, 1989)
(5, 1667)

Reconstructed secret from any 3 shares:
Recovered secret = 1234
✓ Correct!

```

[ ] # -----
# Part III: Simple Garbled Circuit (AND Gate)
# -----

import secrets

# Inputs
A = 1    # Omar's bit
B = 0    # Nancy's bit

# Random labels for each wire
L0_A = secrets.token_hex(8)
L1_A = secrets.token_hex(8)

L0_B = secrets.token_hex(8)
L1_B = secrets.token_hex(8)

L0_OUT = secrets.token_hex(8)
L1_OUT = secrets.token_hex(8)

# Garbled table for AND gate
garbled_table = {
    (L0_A, L0_B): L0_OUT,  # 0 AND 0 = 0
    (L0_A, L1_B): L0_OUT,  # 0 AND 1 = 0
    (L1_A, L0_B): L0_OUT,  # 1 AND 0 = 0
    (L1_A, L1_B): L1_OUT  # 1 AND 1 = 1
}

# Each party receives only their label

```

```
omar_label = L1_A if A == 1 else L0_A
nancy_label = L1_B if B == 1 else L0_B

# Evaluator computes output without knowing A or B
output_label = garbled_table[(omar_label, nancy_label)]

output_bit = 1 if output_label == L1_OUT else 0

print("Omar's input label:", omar_label)
print("Nancy's input label:", nancy_label)
print("Output:", output_bit, "(A AND B)")
```

▼
Omar's input label: ffb9315ebe875856
Nancy's input label: e816bc6a0d6d5450
Output: 0 (A AND B)

Colab paid products - [Cancel contracts here](#)

{ } Variables  Terminal

