In [ ]:

```
import numpy as np
from math import sqrt
from scipy.interpolate import lagrange
import random
from smpc_secrets import RandPoly


# calculate the Euclidean distance between two vectors
# standard, insecure way
# each row is the set of feataures for a node
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)):
        distance += (row1[i] - row2[i]) ** 2
    return sqrt(distance)


# generate a function for each feature in the array
def generate_functions(features_arr):
    all_functions = []
    for feature in range(len(features_arr)):
        func = RandPoly(
            name=f"f{feature}",
            n=1,
            R=[
                (i, x)
                for i, x in enumerate(
                    list([features_arr[feature], random.randint(2, 250)])
                )
            ],
        ).poly
        all_functions.append(func)
    return all_functions


# generate shares for each feature function for a given node_id
def generate_shares(func_array, node_id):
    shares = []
    for func in func_array:
        shares.append(func(node_id))
    return shares


# calculate distances between each feature in the arrays
def get_feature_distances(arr1, arr2):
    distances = []
    for feature_a, feature_b in zip(arr1, arr2):
        dist = (feature_a - feature_b) ** 2
        distances.append(dist)
    return distances


# add all the elements of the array
def sum_distances(arr):
    return sum(arr)


# Get the final distance
```

```python
# i.e.: Given, S(1), S(2), and S(3), calculate S(0)
def reconstruct(shares):
    x = np.arange(1, len(shares) + 1)
    y = shares
    f = lagrange(x, y)
    return f(0)


# Maria's version of SPMC distance calculation
def simplified_calc(alice_data, bob_data):
    # Alice's functions
    assert len(alice_data) == len(bob_data), "feature length mismatch!"

    # get private functions
    alice_functions = generate_functions(alice_data)
    bob_functions = generate_functions(bob_data)

    # get shares
    alice_personal_shares = generate_shares(alice_functions, 1)
    alice_from_bob = generate_shares(bob_functions, 1)

    bob_personal_shares = generate_shares(bob_functions, 2)
    bob_from_alice = generate_shares(alice_functions, 2)

    server_from_alice = generate_shares(alice_functions, 3)
    server_from_bob = generate_shares(bob_functions, 3)

    # compute distance for each feature
    alice_distances = get_feature_distances(alice_personal_shares, alice_from_bob)
    bob_distances = get_feature_distances(bob_personal_shares, bob_from_alice)
    server_distances = get_feature_distances(server_from_alice, server_from_bob)

    # get sum of distances
    alice_sum = sum_distances(alice_distances)
    bob_sum = sum_distances(bob_distances)
    server_sum = sum_distances(server_distances)

    # print("Sums: ", alice_sum, bob_sum, server_sum)

    dist_squared = reconstruct([alice_sum, bob_sum, server_sum])

    alice_bob_distance = np.sqrt(dist_squared)

    return alice_bob_distance


def compare_computations(a, b):
    print("\n************Comparing computations************")
    print(f"a: {a}\nb: {b}")
    print("\nInsecure distance calc: ", euclidean_distance(a, b))
    print("Secure distance calc:", simplified_calc(a, b))


# TEST
compare_computations([0, 0], [0, 0])
compare_computations([1, 1], [4, 4])
compare_computations([1, 5, 9, 17], [72, 5, 18, 16])
compare_computations([1, 2, 3, 4, 5, 6, 7, 8, 9], [11, 12, 13, 14, 15, 16, 17, 18, 19]
```