Shamir's Secret Sharing

Problem Breakdown:

Shamir's Secret Sharing Scheme is a cryptographic technique that allows a secret value to be split into multiple shares such that:

1. The secret can be reconstructed if at least `k` shares are known.
2. Any knowledge of fewer than `k` shares provides no information about the secret.

Your solution is implementing this by using **Lagrange Interpolation**, which allows us to reconstruct a polynomial from given points.

Breakdown of the Provided Code:

1. **Parsing Input Data**
    - The JSON input consists of:
        - `keys`: Defines `n` (total shares) and `k` (minimum required shares to reconstruct the secret).
        - A dictionary where each key represents a share identifier, and each value contains a `base` and `value`.
    - The code converts these values into decimal form for computation.
2. **Lagrange Interpolation Function**
    - The function `lagrange_interpolation(points)` constructs a polynomial from given points `(x, y)` using:

      $$P(x) = \sum_{i=0}^{k-1} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

    - The `sympy` library is used to expand and simplify the polynomial.
3. **Generating Points for Interpolation**
    - The decimal values from the input are mapped into `(index, value)` pairs.
    - Example from Test Case 1:

```
points = [(1, 4), (2, 7), (3, 12), (6, 39)]  # Converted to decimal
```

4. **Generating Polynomials from Combinations of `k` Points**
    - Since any `k` points are sufficient to reconstruct the polynomial, all possible subsets of `k` points are generated using `itertools.combinations`.
    - The polynomial is computed for each subset.
5. **Validating the Polynomial**
    - The script checks if a polynomial correctly reconstructs a known value `(x_test, y_test)`.

Explanation with Test Case 1:

```
{
  "keys": { "n": 4, "k": 3 },
  "1": { "base": "10", "value": "4" },
  "2": { "base": "2", "value": "111" },
  "3": { "base": "10", "value": "12" },
  "6": { "base": "4", "value": "213" }
}
```

Step 1: Convert to Decimal Values

```
(1, 4)   # Base 10
(2, 7)   # Base 2 → Decimal 7
(3, 12)  # Base 10
(6, 39)  # Base 4 → Decimal 39
```

Step 2: Compute Lagrange Interpolation Polynomial

```
P(x) = a_0 + a_1*x + a_2*x^2
```

The coefficients are computed using interpolation, ensuring that the polynomial passes through the given points.

Step 3: Verify the Polynomial

The script evaluates the polynomial at
`x_test=1` and checks if the computed value matches `y_test=4`.

Solution Implementation:

```
import json
from sympy import symbols, expand
from itertools import combinations

def lagrange_interpolation(points):
    x = symbols('x')
    polynomial = 0
    for i in range(len(points)):
        xi, yi = points[i]
        term = yi
        for j in range(len(points)):
            if i != j:
                xj, _ = points[j]
                term *= (x - xj) / (xi - xj)
        polynomial += term
    return expand(polynomial)

with open('input.json') as file:
    data = json.load(file)

n = data['keys']['n']
k = data['keys']['k']

points = []
for key, value in data.items():
    if key.isdigit():
        base = int(value['base'])
        decimal_value = int(value['value'], base)
        points.append((int(key), decimal_value))

print("Points:", points)

combinations_of_points = combinations(points, k)
polynomials = []

for combo in combinations_of_points:
    polynomial = lagrange_interpolation(combo)
    polynomials.append(polynomial)
```

```
    print(f'Polynomial for {combo}: {polynomial.simplify()}')

print("Generated polynomials:", polynomials)

x_test, y_test = 1, 4  # Example verification
x = symbols('x')
for poly in polynomials:
    if poly.subs(x, x_test) == y_test:
        print("Found the correct polynomial:", poly)
        break
```

Conclusion:

- The approach effectively reconstructs the polynomial, allowing us to determine the original secret.
- The method ensures that knowledge of fewer than `k` points provides no information about the secret, aligning with Shamir's Secret Sharing principles.