## Homomorphic Encryption and Polynomial Validation

Homomorphic Encryption (HE) allows computations to be performed on encrypted data without decrypting it. In this solution, I use CKKS encryption, which supports approximate arithmetic on encrypted floating-point numbers, ensuring privacy while still enabling operations like addition and multiplication on sensitive data.

## Polynomial Validation in Homomorphic Encryption

Polynomial validation involves using a polynomial to verify if values in a shared list match values from a suspicious list. A polynomial is generated from the suspicious values, and if the polynomial evaluates to zero when applied to a value from the shared list, it indicates a match. This technique ensures privacy since the comparisons are performed over encrypted data.

## The Value Polynomial Validation in Homomorphic Encryption

**Efficient Matching**: Instead of comparing each value individually, polynomial validation allows multiple suspicious values to be represented in a single mathematical expression. This provides a more elegant and efficient method of performing encrypted comparisons.

**Privacy-Preserving Search:** Polynomial validation ensures that the actual suspicious values are never exposed, even during the matching process. By performing all operations on encrypted data, this method enhances data security and confidentiality.

**Scalability:** Although there are limitations in the degree of polynomial that can be used, polynomial validation has the potential to scale more efficiently than individual encrypted comparisons. The use of a single polynomial to represent multiple values helps reduce the number of comparisons needed, enhancing computational efficiency.

**Compact Representation:** Polynomial validation provides a compact way of encoding the entire suspicious list into a single polynomial, allowing for condensed encrypted operations instead of numerous pairwise comparisons. This reduces the need for multiple homomorphic operations, which can be costly.

## How the solution works

**Generate Polynomial:** A polynomial is generated from the suspicious list. For example, if the suspicious list contains values [a, b, c], the polynomial is (x - a) \* (x - b) \* (x - c).

**Encrypt Shared Values:** Each value in the shared list is encrypted using CKKS encryption.

**Evaluate Polynomial:** The polynomial is evaluated on the encrypted shared values. If the result of the polynomial evaluation is close to zero, it indicates that the shared value matches a suspicious value.

**Decryption and Matching:** The result of the polynomial evaluation is decrypted, and if it’s close to zero, the shared value is considered a match.

## Code Workflow

**Polynomial Generation:**

A polynomial is created with the suspicious values as roots. This polynomial will evaluate to zero for any value in the suspicious list.

**Encrypt Shared Values:**

Each shared value is encrypted using CKKS encryption, allowing computations on encrypted data while keeping the values confidential.

**Evaluate Polynomial:**

The polynomial is evaluated for each encrypted shared value. Homomorphic operations like multiplication and addition are used to compute the polynomial evaluation while the data remains encrypted.

**Decryption and Comparison:**

After the polynomial is evaluated, the result is decrypted. If the result is close to zero, the shared value is identified as a match to the suspicious list.

## Issues Faced and Resolutions

**Scale Overflow in CKKS:**

***Problem:*** CKKS encryption uses a scale that grows with every multiplication. With high-degree polynomials, this scale grew too large, leading to overflow errors.

***Resolution*:** reduced the polynomial degree by limiting the number of terms used from the suspicious list and applied rescaling to control the size of coefficients during evaluation.

**Precision Loss**

***Problem***: CKKS introduces noise during operations, and the decrypted results were sometimes far from the expected values, leading to mismatches.

***Resolution***: simplified the polynomial by using linear validation (i.e., single-term polynomials) and applied dynamic tolerances to handle precision loss.

**False Negatives:**

***Problem***: In some cases, matching values were missed because the decrypted results were not close enough to zero.

***Resolution***: adjusted the matching tolerance dynamically based on the magnitude of the result to more accurately identify matches.

## Limitations of the Code

**Limited Polynomial Degree:**

Reducing the polynomial degree helps avoid scale overflow but limits the number of suspicious values that can be processed simultaneously. This restricts the complexity of the polynomial we can handle.

**Precision Issues:**

CKKS encryption introduces noise, and precision loss can cause incorrect results, especially when dealing with high-degree polynomials. This was addressed by reducing polynomial complexity and applying dynamic tolerances, but precision issues still pose a limitation.

**Scalability:**

While polynomial validation condenses multiple comparisons into a single operation, it can still be computationally expensive when dealing with large datasets, especially when multiple homomorphic operations are required.

## Conclusion

Polynomial validation adds significant value to Homomorphic Encryption by providing a compact, efficient, and privacy-preserving method for matching encrypted data. By representing multiple suspicious values in a single polynomial, polynomial validation enhances computational efficiency while maintaining data confidentiality. However, managing precision and controlling scale in CKKS encryption presents challenges, particularly when dealing with complex polynomials, and limitations remain in terms of scalability and precision. Despite these challenges, the use of polynomial validation in HE is a powerful technique for privacy-preserving data matching.