**Handover Document: Homomorphic Encryption and Polynomial Validation**

**Introduction**

This document provides an overview of the Homomorphic Encryption and Polynomial Validation solution. It explains the methodology, code workflow, challenges faced, resolutions implemented, and limitations of the solution. This handover ensures continuity in development and maintenance of the code.

**Overview of Homomorphic Encryption (HE)**

Homomorphic Encryption (HE) enables computations on encrypted data without decrypting it. In this solution, **CKKS encryption** is utilised, which supports approximate arithmetic on encrypted floating-point numbers. It ensures privacy while allowing operations such as addition and multiplication on sensitive data.

**Polynomial Validation in Homomorphic Encryption**

Polynomial validation is a method for verifying if values from a shared list match values from a suspicious list. A polynomial is generated from the suspicious values, and when evaluated on a shared value, if the result is zero, it indicates a match. This technique is privacy-preserving as the comparisons occur over encrypted data.

**Benefits of Polynomial Validation in HE**

* **Efficient Matching:** Instead of comparing each value individually, polynomial validation represents multiple suspicious values in a single mathematical expression, enhancing the efficiency of encrypted comparisons.
* **Privacy-Preserving Search:** The suspicious values are not exposed during matching. All operations are conducted on encrypted data, ensuring confidentiality.
* **Scalability:** Polynomial validation can scale more efficiently than individual comparisons, despite some limitations with polynomial degree.
* **Compact Representation:** The suspicious list is encoded into a single polynomial, reducing the need for multiple homomorphic operations.

**How the Solution Works**

1. **Generate Polynomial:** A polynomial is generated from the suspicious list. For instance, if the suspicious list contains values [a, b, c], the polynomial is (x − a) \* (x − b) \* (x − c).
2. **Encrypt Shared Values:** The values in the shared list are encrypted using CKKS encryption.
3. **Evaluate Polynomial:** The polynomial is evaluated on the encrypted shared values. If the result is close to zero, it indicates a match.
4. **Decryption and Matching:** After decryption, if the result is close to zero, the shared value is identified as a match.

**Code Workflow**

**1. Polynomial Generation**

* A polynomial is generated using the suspicious values as roots.
* The polynomial evaluates to zero for any value that matches a suspicious value.

**2. Encrypt Shared Values**

* The shared list is encrypted using CKKS encryption, allowing computations while keeping the data confidential.

**3. Evaluate Polynomial**

* The polynomial is evaluated for each encrypted shared value using homomorphic operations such as addition and multiplication.

**4. Decryption and Comparison**

* After evaluation, the result is decrypted. If it is close to zero, the value is a match with the suspicious list.

**Issues Faced and Resolutions**

**1. Scale Overflow in CKKS**

* **Problem:** The CKKS scale grows with each multiplication, and high-degree polynomials resulted in overflow.
* **Resolution:** Reduced the polynomial degree by limiting the number of terms used and applied rescaling to control coefficient sizes during evaluation.

**2. Precision Loss**

* **Problem:** CKKS introduces noise, and some decrypted results were far from the expected values.
* **Resolution:** Simplified the polynomial by using linear validation (single-term polynomials) and applied dynamic tolerances to manage precision loss.

**3. False Negatives**

* **Problem:** Some matching values were missed as the decrypted results were not sufficiently close to zero.
* **Resolution:** Adjusted the matching tolerance dynamically based on the result’s magnitude to more accurately identify matches.

**Limitations of the Code**

**1. 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 that can be handled.

**2. Precision Issues**

* CKKS encryption introduces noise that can affect results, especially with complex polynomials. Precision issues persist even with simplifications and dynamic tolerances.

**3. Scalability**

* While polynomial validation optimises comparison operations, handling large datasets or numerous homomorphic operations can be computationally expensive.

**Conclusion**

Polynomial validation enhances Homomorphic Encryption by providing a compact, efficient, and privacy-preserving method for matching encrypted data. Despite challenges related to precision and scalability, it is a powerful technique for privacy-preserving data matching. Continuous optimisation around polynomial degree and precision management is crucial to improving performance and scalability.

For further development, the focus should be on:

1. Exploring ways to handle more complex polynomials without sacrificing precision.
2. Investigating new strategies to improve scalability when handling larger datasets.