**Project Report\_Order-Preserving Encryption (OPE)**

**Team 6: Priyanka Vyas**

**1.Explanation about order preserving encryption the way it works (2 pts);**

Order-Preserving Encryption (OPE) is a cryptographic technique that encrypts data in such a way that the order of the plaintext is preserved in the ciphertext. In simpler terms, if we have two plaintext values, say ‘x’ and ‘y’, such that ‘x < y’, then their encrypted counterparts ‘Enc(x)’ and ‘Enc(y)’ will also maintain this order, i.e., ‘Enc(x) < Enc(y)’. This property is crucial for certain types of database operations like sorting and range queries, which can be performed on the encrypted data without needing to decrypt it. The working principle of OPE involves mapping plaintext values to ciphertext values while preserving their order. This is typically achieved using various cryptographic algorithms and techniques. However, it's important to note that OPE is inherently less secure than other forms of encryption because it leaks some information about the plaintext, specifically the order of the values. Therefore, its use is context-dependent, balancing the need for functionality in encrypted databases against the security requirements.

**2.Order-Preserving Encryption (OPE)** is a type of encryption method that maintains the order of the plaintext values in the encrypted form. This means that if we have two values, say A and B, where A is less than B, then after applying OPE, the encrypted value of A will still be less than the encrypted value of B. This property is particularly useful in database encryption for performing range queries, sorting, and comparison operations directly on encrypted data without needing to decrypt it first.

The source code demonstrates the use of a very simple OPE scheme. It multiplies a numeric value by a fixed key (OPE\_KEY) to encrypt it and divides the encrypted value by the same key to decrypt it. This simplistic approach preserves the order of the values. For instance, if OPE\_KEY = 10, encrypting 5 gives 50, and encrypting 10 gives 100, preserving the order (5 < 10 and 50 < 100).

**3.Applying OPE on the "Weight" Attribute**

In the context of the source code, OPE is applied to the "Weight" attribute of patient records in a database. This implementation facilitates range queries on encrypted data, allowing the database to retain the functionality of querying based on weight without compromising the confidentiality of the data.

**3.1Application Process:**

**Encryption:** When entering new patient data, the "Weight" attribute is encrypted using the OPE method. In this specific implementation, it involves multiplying the weight by a predefined key (OPE\_KEY). When a new patient record is inserted, the weight attribute is encrypted using simple\_ope\_encrypt function, which multiplies the weight by the OPE\_KEY.

**Storage:** The encrypted value of the weight is stored in the database. This method protects the actual weight data, preserving patient privacy.

**Range Queries**: To execute a query within a certain weight range, the query parameters (minimum and maximum weights) are first encrypted using the OPE method. The query is then run against these encrypted values, accurately returning the desired range of records due to the order-preserving nature of OPE. To perform a range query (e.g., find patients within a certain weight range), the query bounds are first encrypted using the same OPE method. The query is then executed against these encrypted bounds. Since the order is preserved, this returns the correct range of records.

**Decryption:** When it's necessary to view or use the actual weight values, the encrypted data can be decrypted by reversing the encryption process. In this implementation, this means dividing by the OPE\_KEY. When reading the data, the encrypted weight can be decrypted back to the original weight using simple\_ope\_decrypt, dividing by the OPE\_KEY.

**4. Design and Implementation Details:**

A diagram of data processing

Description automatically generated

**Fig 1: Secure DBaaS System Architecture diagram with OPE implementation**

**4.1. Design Overview:**

The Secure DBaaS System Architecture diagram in figure 1 with OPE implementation, the primary design goal is to securely store sensitive patient data, like weight, while still allowing specific database operations like sorting and range queries. OPE is chosen for its ability to preserve the order of numeric values post-encryption, which is key to enabling these operations on encrypted data. The design aims to enable secure storage of sensitive data (like patient weight) while allowing certain database operations (like range queries) without compromising data privacy. The use of OPE facilitates this by keeping the relative order of the encrypted data identical to that of the original data.

**4.2. Implementation Details:**

* **Simple OPE Scheme:** The OPE scheme multiplies/divides the numeric value by a fixed key (OPE\_KEY). This is a straightforward approach but not highly secure against sophisticated attacks, as patterns in the data could be discerned.
* **Data Encryption/Decryption:** simple\_ope\_encrypt and simple\_ope\_decrypt functions handle the encryption and decryption of weight values. These functions, handle the conversion of the weight attribute to and from its encrypted form.
* **Range Queries:** Encrypted range values are used in SQL queries to fetch relevant records. The OPE allows for encrypted range values to be directly used in SQL queries, enabling accurate data retrieval without decryption. The returned data is accurate due to the order-preserving nature of the encryption.
* **Database Integration:** The encrypted weight is stored alongside other patient data. Other sensitive data such as gender and age are encrypted using AES for stronger security.
* **Security Considerations:** While simple OPE is used for weight, it's important to note that this method is not highly secure and is used here for demonstration purposes. In real-world applications, more complex and secure OPE methods should be considered.

**5. Testing Evaluation and Validation (Output Analysis):**

In the source code, an OPE\_KEY is defined (as 10). This key is used in the simple OPE functions (simple\_ope\_encrypt and simple\_ope\_decrypt) to encrypt and decrypt the weight data of patients. The encryption is done by multiplying the weight by the OPE\_KEY, and decryption is the inverse process, dividing the encrypted value by the OPE\_KEY. The key choice ensures that the relative order of the weights is maintained even after encryption. The range query functionality is demonstrated in the function test\_ope\_range\_query. Here, a weight range is defined (e.g., between 70.0 and 80.0). The range is then encrypted using the same OPE method, resulting in a new range of encrypted values. When a range query is executed on the encrypted database using these encrypted values, it returns the correct set of records that fall within the original weight range. This is possible because the order of the values is preserved even after encryption. The output of the source code, especially after running the range query, demonstrates that the system can correctly retrieve records based on a specified range of weights, even though the weights are stored in an encrypted form in the database. This is significant because it allows for the performance of certain types of database queries without compromising the privacy and security of the sensitive data (in this case, patient weights). For instance, if the output shows patient records with weights between 70.0 and 80.0, it means that the OPE and the range query are functioning as expected - the system is able to identify and retrieve records that fall within the specified range of weights, despite the weights being stored in an encrypted format.

**6. Security and Privacy-Related Limitation:**

1. An attacker with access to the encrypted data might deduce the OPE\_KEY or gain insights into the plaintext values due to the linear relationship between the plaintext and ciphertext. This compromises the confidentiality of the data and poses a significant privacy risk.
2. Since OPE preserves the order of the encrypted values, it inherently leaks some information about the underlying plaintext data.

**7. Potential Vulnerabilities**

1. The use of a static OPE\_KEY across the dataset increases vulnerability. If this key is compromised, all encrypted data can be easily decrypted.
2. The current error handling during encryption and decryption is minimal, potentially leading to unhandled exceptions or leaks of sensitive information.
3. It is learned that while this method is straightforward and demonstrates the concept of OPE, it is not highly secure and might be susceptible to pattern analysis attacks.
4. The approach lacks additional layers of security, such as noise addition or more complex encryption algorithms, that could provide better protection against these advanced methods of attack.

**8. Improvements**

Implementing secure padding techniques and checks can safeguard against padding oracle attacks in AES encryption

**9. Source Code:**

Enclosed Source code file: Order-Preserving Encryption (OPE)\_source\_code.ipynb

Gitlink: <https://github.com/vpriyanca/Data-Security-Project>