Table of Contents

[**What is Homomorphic Encryption**? 2](#_Toc175838801)

[ Encryption 2](#_Toc175838802)

[ Computation 2](#_Toc175838803)

[ Decryption 2](#_Toc175838804)

[**How does Homomorphic Encryption** **work?** 2](#_Toc175838805)

[ Encryption 2](#_Toc175838806)

[ Homomorphic Operations 2](#_Toc175838807)

[ Decryption 2](#_Toc175838808)

[ Mathematical Foundation 2](#_Toc175838809)

[ Security 3](#_Toc175838810)

[ Applications 3](#_Toc175838811)

[**Using Homomorphic Encryption for Suspect List Search: A Step-by-Step Example** 3](#_Toc175838812)

[ Encrypt Both Lists 3](#_Toc175838813)

[ Perform the Search 3](#_Toc175838814)

[ Return Encrypted Results 4](#_Toc175838815)

[ Challenges and Considerations 4](#_Toc175838816)

[**What is Semi-Honest Model?** 4](#_Toc175838817)

[**How Homomorphic Encryption is related to Semi-honest Protocols?** 5](#_Toc175838818)

[ Use of Homomorphic Encryption in Semi-Honest Protocols 5](#_Toc175838819)

[ How It Works? 5](#_Toc175838820)

[ Why Homomorphic Encryption Fits the Semi-Honest Model 5](#_Toc175838821)

[ Application to List Matching 5](#_Toc175838822)

[ Conclusion 6](#_Toc175838823)

[FNP04 Protocol 6](#_Toc175838824)

[ Problem Setup 6](#_Toc175838825)

[ Core Idea: Polynomial Representation of Sets 6](#_Toc175838826)

[ Privacy Considerations: 7](#_Toc175838827)

[ Security Model: 7](#_Toc175838828)

[ Applications: 7](#_Toc175838829)

[ Summary: 7](#_Toc175838830)

[FNP04 and homomorphic encryption 8](#_Toc175838831)

[Similarities: 8](#_Toc175838832)

[Differences: 8](#_Toc175838833)

[How They Could Interact 8](#_Toc175838834)

[ Summary: 9](#_Toc175838835)

# **What is Homomorphic Encryption**?

**Homomorphic Encryption** allows performing computations on encrypted data without decrypting it, ensuring that sensitive data remains secure even during processing.

* Encryption: The data (plaintext) is encrypted using a homomorphic encryption scheme, resulting in a ciphertext.
* Computation: Operations such as addition or multiplication are performed directly on the ciphertexts. The key property of HE is that these operations on ciphertexts correspond to operations on the underlying plaintexts.
* Decryption: After the computations are completed, the resulting ciphertext can be decrypted to obtain the result in plaintext.

# **How does Homomorphic Encryption** **work?**

* Encryption**:** 
  + Start with a plaintext message m.
  + Encrypt the message using a homomorphic encryption scheme. The result is an encrypted message or ciphertext c=Enc(m).
* Homomorphic Operations**:**
  + If you want to add two encrypted values: c1 = Enc(m1) and c2=Enc(m2). you can perform this directly on the ciphertexts: c3=c1⊕c2=Enc(m1+m2).
  + Similarly, multiplication can be performed: c4 = c1⊗c2 = Enc(m1×m2).

## Decryption

* After performing the desired homomorphic operations on the ciphertexts, you can decrypt the result.
* If c3 was the result of adding two encrypted messages, the decryption m3=Dec(c3) will yield m1+m2 ​.
* This means that even though the operations were performed on encrypted data, the result, when decrypted, matches what would have been obtained if the operations were performed on the plaintext data.

## Mathematical Foundation

* Homomorphic encryption schemes are typically built on complex mathematical structures like lattices, rings, or groups, which allow operations to be mirrored in both the encrypted and unencrypted domains.
* For example, in some lattice-based schemes, encryption might involve adding a small noise term to the data. During computations, this noise increases but can be managed or "reduced" using techniques specific to the scheme.

## Security

* The security of homomorphic encryption relies on the difficulty of certain mathematical problems (e.g., lattice problems), which make it infeasible for attackers to decrypt the data without the appropriate key.
* Because operations are performed on ciphertexts without revealing the underlying plaintext, the data remains confidential throughout the computation process.

## Applications

* Privacy-Preserving Computations: Allowing users to outsource computations (e.g., to a cloud server) while keeping their data private.
* Secure Data Sharing: Enabling computations on shared data without exposing the data itself.
* Encrypted Machine Learning: Performing machine learning on encrypted datasets to maintain privacy.

# **Using Homomorphic Encryption for Suspect List Search: A Step-by-Step Example**

## Encrypt Both Lists

Let's say we have:

* **List A (Suspicious List):** ["Alice", "Bob", "Charlie"]
* **List B (Reference List):** ["Eve", "Bob", "Dave"]

Using homomorphic encryption, we encrypt each element in both lists:

* **Encrypted List A:** [ Enc (Alice), Enc (Bob), Enc (Charlie)]
* **Encrypted List B:** [ Enc (Eve), Enc (Bob), Enc (Dave)]

Each Enc () function encrypts the data, making it unreadable without the decryption key.

## Perform the Search

For each element in the encrypted suspicious list (List A), we want to compare it with each element in the encrypted reference list (List B). Since homomorphic encryption allows computations on encrypted data, we can perform operations like subtraction to check for equality.

* For Enc (Alice) in List A, compute the difference with each element in List B:
  + Enc (Alice) – Enc (Eve)
  + Enc (Alice) – Enc (Bob)
  + Enc (Alice) – Enc (Dave)
* If the result of any of these operations, after decryption, equals zero, it means "Alice" was found in List B.

Since all operations are done on encrypted data, the server performing these operations cannot see the actual names, only encrypted results.

## Return Encrypted Results

The results of the comparisons are still encrypted:

* For example, if Enc (Alice) - Enc (Eve) is not zero, it indicates no match.
* If Enc (Bob) - Enc (Bob) equals zero, it indicates a match.

The encrypted results are returned to the data owner, who can then decrypt them to see which suspicious names from List A are present in List B.

## Challenges and Considerations

* Performance:
  + Homomorphic encryption is computationally intensive. Searching for matches between two large lists can be slow due to the complexity of performing operations on encrypted data.
  + Optimizations, such as using batched encryption (processing multiple values together) or specific homomorphic encryption schemes optimized for comparisons, can help improve performance.
* Noise Accumulation:
  + Each homomorphic operation adds noise to the ciphertext. After a certain number of operations, the noise can make the ciphertext undecryptable. Managing this noise is crucial for ensuring correct results.
* Security:
  + Homomorphic encryption ensures that no party performing the search learns anything about the plaintext data, preserving privacy. Only the data owner, with the decryption key, can interpret the results.
* Use Case:
  + This method is suitable for privacy-sensitive applications where direct comparison of plaintext lists is not an option. For example, it could be used in scenarios involving sensitive data like medical records, financial information, or surveillance lists.

# **What is Semi-Honest Model?**

* In the semi-honest model, parties involved in a protocol are assumed to follow the protocol correctly (i.e., they behave honestly), but they might try to learn additional information from the data they receive during the protocol execution.
* The parties are "curious" and may analyse any intermediate data to gain extra information, but they do not deviate from the agreed-upon protocol.

# **How Homomorphic Encryption is related to Semi-honest Protocols?**

The concept of using homomorphic encryption for comparing two lists, such as finding a suspicious person in another list, is closely related to **semi-honest protocols** (also known as "honest-but-curious" protocols) in secure multi-party computation (MPC). Here's how they are connected:

## Use of Homomorphic Encryption in Semi-Honest Protocols

* Homomorphic encryption can be employed in semi-honest protocols to ensure that sensitive data remains private even when other parties might be curious.
* In the context of searching or comparing lists, a semi-honest protocol would involve two or more parties who wish to compare their lists (e.g., a suspicious list against another) without revealing their data to each other.

## How It Works?

* **Party A** encrypts their list using a homomorphic encryption scheme and sends the encrypted data to **Party B**.
* **Party B** performs the required computations (e.g., comparing the encrypted elements) directly on the encrypted data and returns the results to **Party A**.
* **Party A** decrypts the results to determine which elements, if any, match between the lists.
* Throughout this process, **Party B** does not learn anything about **Party A’s** data, ensuring that the semi-honest assumption is respected.

## Why Homomorphic Encryption Fits the Semi-Honest Model

* **Confidentiality**: Homomorphic encryption ensures that intermediate computations reveal no useful information to the party performing them. Even though **Party B** sees the encrypted data and the results of operations on this data, they cannot decrypt or understand the actual content.
* **Correctness**: Since the parties follow the protocol as specified, they do not attempt to decrypt the data or alter the results, adhering to the semi-honest assumption.
* **Security**: In a semi-honest protocol, even if a party is curious, they cannot learn anything beyond what they are supposed to because the data remains encrypted during processing.

## Application to List Matching

* When using homomorphic encryption for list matching in a semi-honest setting, both parties (e.g., law enforcement and a company holding sensitive data) can collaborate to identify if a suspicious person is in the company's database without either party learning unnecessary information about the other's data.
* This ensures that the company’s list remains private, and the law enforcement agency only learns if there is a match, without gaining access to the full list.

## Conclusion

Homomorphic encryption provides a powerful tool in semi-honest protocols by allowing computations on encrypted data without revealing the underlying information. This aligns with the semi-honest model, where parties are trusted to follow the protocol but might be curious to learn more, ensuring that sensitive information remains secure during the process.

# FNP04 Protocol

FNP04 refers to a specific cryptographic protocol introduced in the 2004 paper titled "Privacy-Preserving Matching for Statistical Databases" by Moni Naor and Benny Pinkas. The FNP04 protocol is designed for privacy-preserving matching, allowing two parties to compute the intersection of their private sets without revealing any other information about the sets.

The protocol uses **polynomial evaluation** as its core technique, leveraging properties of polynomials to achieve this privacy-preserving set intersection.

## Problem Setup

* **Parties Involved**:
  + **Party A**: Holds a private set A= {a1, a2, …, an}.
  + **Party B**: Holds a private set B= {b1, b2, …, bm}.
* **Goal**:
  + Both parties want to find the intersection A∩B without revealing their respective sets to each other.

## Core Idea: Polynomial Representation of Sets

* **Polynomial Construction by Party A:**
  + Party A represents its set A as a polynomial PA(x), where the roots of the polynomial correspond to the elements of set A. Specifically, Party A constructs: (x)=
* This means that for each element ai in set AAA, the polynomial PA(x) will equal zero when x=ai. For example, if A = {2,3}, then:
* PA(x)=(x−2) (x−3) = x^2 - 5x + 6
* **Oblivious Polynomial Evaluation by Party B:**
  + Party B has its own set B with elements {b1,b2,…bm). Party B wants to check if any elements of set B are also in set A without revealing its elements to Party A and without learning anything about set A other than the intersection.
* **Process:**
  + Party B wants to evaluate PA(x) for each bj​ to see if the result is zero (indicating that bj is in set A).
  + To preserve privacy, this evaluation must be done in such a way that Party A does not learn which elements are being evaluated, and Party B does not learn anything about the polynomial PA(x) other than the result of the evaluation.
* **Oblivious Transfer:**
  + The OT protocol is employed to securely perform the polynomial evaluation while maintaining the privacy of both parties.
  + Party B uses OT to obtain the values PA(bj) for each element bj in its set B without Party A learning which elements are being evaluated.
  + Party B only receives the evaluation result PA(bj), without gaining any additional information about the polynomial PA(x) itself.
* **Result**:
  + After evaluating PA(bj) for each bj ​ in B, Party B can determine the intersection A∩B.

### Privacy Considerations:

* **Party A's Privacy**: Party A's set A is represented by a polynomial, but Party B only receives evaluations of this polynomial and does not learn the actual elements of A.
* **Party B's Privacy**: Party B evaluates the polynomial PA(x) on its elements without revealing those elements to Party A.

### Security Model:

The FNP04 protocol operates under the **semi-honest model** (honest-but-curious model), where both parties follow the protocol correctly but may try to learn additional information from the messages they receive.

### Applications:

* **Privacy-Preserving Data Mining**: FNP04 can be used in scenarios where two parties want to find common data entries (e.g., common customers or common research subjects) without sharing their full databases.
* **Secure Multi-Party Computation**: It serves as a building block for more complex protocols that require secure computation over private data.

### Summary:

FNP04 is a privacy-preserving protocol that allows two parties to compute the intersection of their sets without revealing any other information. The protocol relies on representing one party's set as a polynomial and then allowing the other party to perform secure evaluations of this polynomial. This method ensures that both parties' data remains confidential throughout the process, making it a powerful tool for privacy-preserving set operations.

# FNP04 and homomorphic encryption

The FNP04 protocol, while using polynomial evaluation for privacy-preserving matching, is not directly based on homomorphic encryption. However, both FNP04 and homomorphic encryption share a common goal: performing computations on private data without revealing that data. Here's how FNP04 relates to homomorphic encryption and how they differ:

## Similarities:

* **Privacy-Preserving Computations**:
  + Both FNP04 and homomorphic encryption allow for computations on data that is meant to remain private. In FNP04, this involves evaluating a polynomial representing a private set, while in homomorphic encryption, it involves performing mathematical operations (like addition or multiplication) on encrypted data.
* **Security Models**:
  + Both techniques are often used within the semi-honest (honest-but-curious) security model, where parties follow the protocol but may try to learn additional information.

## Differences:

* **Underlying Techniques**:
  + **FNP04**: The FNP04 protocol is primarily based on polynomial representation and oblivious transfer. Party A encodes its set as a polynomial, and Party B uses a secure protocol to evaluate this polynomial without revealing its inputs.
  + **Homomorphic Encryption**: Homomorphic encryption is a cryptographic method where encrypted data can be directly manipulated. Operations performed on the encrypted data correspond to operations on the plaintext data, allowing complex computations without ever decrypting the data during the process.
* **Focus on Specific Tasks**:
  + **FNP04**: The protocol is specifically designed for privacy-preserving set intersection, where the goal is to find common elements between two private sets without revealing any other information.
  + **Homomorphic Encryption**: More general in its application, homomorphic encryption can be used for various computations on encrypted data, not just set intersections.
* **Performance and Complexity**:
  + **FNP04**: Generally more efficient for the specific task of set intersection, as it doesn't require the computational overhead associated with fully homomorphic encryption.
  + **Homomorphic Encryption**: Can be computationally expensive, especially when dealing with fully homomorphic encryption, which allows for arbitrary operations on encrypted data.

## How They Could Interact

* **Homomorphic Encryption in Set Intersection**:
  + You could theoretically use homomorphic encryption to perform set intersection in a way similar to FNP04, where both sets are encrypted, and the intersection is computed directly on the encrypted data. However, this might be less efficient than using FNP04 for this specific task.
* **Enhancing FNP04**:
  + Homomorphic encryption could be used in conjunction with FNP04 to enhance security or support more complex operations, but this would increase computational overhead.

### Summary:

The FNP04 protocol is related to homomorphic encryption in the broader context of secure and privacy-preserving computation. However, it uses different cryptographic techniques (like polynomial evaluation and oblivious transfer) tailored for the specific task of set intersection, whereas homomorphic encryption provides a more general framework for computations on encrypted data.