ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dipartimento di Scienze ed Ingegneria Informatica

Corso di Informatica

A Study on Applications of Homomorphic Encryption in Privacy-Preserving Protocols

Relatore:
Prof. FEDERICO MONTORI

Presentata da:
DIEGO BARBIERI

Correlatore:
Prof. SAVERIO GIALLORENZO

Sessione di Luglio 2025
Anno Accademico 2024/2025

Location

Zero-Trust

Network Protocols

Privacy

Homomorphic Encryption

A me,
Alla fortuna che mi sono creato
e al mio (di)ego

Abstract

This thesis presents a comprehensive study on the application of homomorphic encryption (HE) in privacy-preserving communication protocols, with a particular focus on location-based services. We analyze and compare their effectiveness and performance to identify the practical limitations they face in real-world scenarios. A central part of this work explores different encoding strategies (such as z-order) that enable location data to be securely processed under homomorphic encryption schemes in a grid. We examine how these strategies impact both the privacy guarantees and the computational overhead of the protocols.

As a case study, we design and implement a protocol that allows mobile clients to discover nearby parking spots without revealing their precise location to the server. Special attention is given to the integration of HE in publish/subscribe and request/response paradigms and to the trade-offs that would arise. The protocol is designed starting from the previous analysis of existing solutions, such as LA-MQTT, and incorporates homomorphic encryption to ensure that the server can process location queries without accessing sensitive data. We evaluate the performance of our protocol in terms of computational efficiency and communication overhead, comparing it with existing solutions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, location-based services have become increasingly prevalent in our daily lives. From finding nearby accommodations or points of interest to accessing services tailored to our geographical context, location awareness is playing an essential role in modern applications. However, this convenience comes at a cost: the necessity of sharing users' location data with third parties, often leading to significant privacy concerns.

Exposing our location could lead to unwanted tracking, profiling, and even physical harm. Therefore, it is crucial to adopt technologies that can preserve user privacy even in adversarial settings. For this purpose, homomorphic encryption (HE) emerges as a promising solution that allows computations to be performed on encrypted data without revealing the underlying plain text. This means that sensitive information, such as our location, can be processed without exposing it to the server. This type of encryption is becoming extremely popular in fields such as cloud computing, where data is often stored and processed by third-party providers. At the same time, we are witnessing a shift towards zero-trust protocols, where no middleman is trusted and there is no central authority that can be relied upon to handle sensitive data.

The focus of this thesis explores the applications of homomorphic encryption in privacy-preserving protocols, considered in the context of location-based services. In particular, I present a detailed analysis of how to securely encode location data in a grid system, allowing for efficient processing under homomorphic encryption schemes. A concrete application of this approach, and the primary use case examined in this work, is the privacy-preserving discovery of nearby parking spots.

The motivation for creating a privacy-preserving protocol comes from the necessity of implementing this type of mechanism inside the Location Aware MQTT (LA-

MQTT) protocol[M+22]. LA-MQTT is an extension of MQTT, optimized to work with geofence data and location data sources. The protocol was originally meant to be used with other types of privacy standards that would only obfuscate the client's position but not hide it completely.

The thesis is structured as follows: in Chapter 2, I provide the necessary background on homomorphic encryption and location-based services, including a review of existing solutions and their limitations. In Chapter 3, I present the system architecture of the proposed protocol, detailing how homomorphic encryption is integrated into the publish/subscribe and request/response paradigms. In Chapter 4, I evaluate the performance of the protocol in terms of computational efficiency and communication overhead, comparing it with existing solutions. Finally, in Chapter 5, I summarize the findings and discuss potential future work.

Chapter 2

Background

This chapter will discuss the theoretical background needed to understand the protocol presented in this thesis. We will cover the main network protocols used in distributed systems, such as Request-Response and Publish-Subscribe, and how they can be applied to IoT devices. We will also discuss Universal Location Referencing, Space-Filling Curves, and Privacy-Preserving techniques, such as Homomorphic Encryption, which are the foundation of the solution proposed.

2.1 Request-Response Protocol

Request-response is a fundamental communication pattern in distributed systems where a client sends a request to a server and waits for a response. This synchronous communication model is characterized by its simplicity and direct interaction between parties, making it suitable for operations requiring immediate feedback and confirmation.

2.1.1 HTTP

HTTP (Hypertext Transfer Protocol) is the foundation of data communication on the World Wide Web. It operates as a request-response protocol, allowing clients to request resources from servers and receive responses. HTTP supports various methods (GET, POST, PUT, DELETE) for different types of operations and is extensible through headers and status codes.

HTTPS

HTTPS is a version of HTTP built on the SSL/TLS protocol, providing secure communication over a computer network. It encrypts data exchanged between clients and servers, ensuring confidentiality and integrity. HTTPS is essential for protecting sensitive information from eavesdropping and tampering.

One of the key features of HTTPS is its use of certificates to authenticate the server, ensuring that clients are communicating with the intended entity. This is possible through a relatively high usage of computational resources, which is a trade-off for the enhanced security it provides.

2.1.2 Request-Response in IoT devices

HTTPS is a commonly used choice also for IoT devices. However, its usability is often limited due to several factors:

- **Resource Constraints**[M+23]: Encrypting and decrypting certificate standards (RSA, EEC, AES) can be computationally expensive, which is a significant concern for IoT devices with limited processing power and memory.
- **Lack of Secure Firmware Updates**[BNT19]: Many IoT devices do not support secure firmware updates, making it difficult to patch vulnerabilities in the HTTPS implementation.
- **Weak or Nonexistent Certificate Validation**[PDVRC21]: Many IoT devices do not validate server certificates properly, leading to potential vulnerabilities.

This limitation has led to the development of alternative protocols and standards that are more suitable for IoT devices, such as CoAP (see Section 2.1.3) and MQTT (see Section 2.2.1). These protocols are designed to be lightweight and efficient, making them more suitable for resource-constrained devices.

2.1.3 CoAP

CoAP (Constrained Application Protocol) is a specialized web transfer protocol designed for constrained devices and low-power networks[SHB14]. It operates over UDP, making it lightweight and suitable for IoT applications. CoAP supports request-response interactions similar to HTTP but is optimized for low-bandwidth and high-latency environments.

The protocol presents several key features:

- Web protocol fulfilling M2M requirements in constrained environments.

- UDP-based with support for multicast.
- Low header overhead and parsing complexity.
- URI and Content-type support.
- Simple proxy and caching capabilities.
- A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- Security binding to Datagram Transport Layer Security (DTLS)

Due to its design, CoAP can replicate the RESTful architecture of HTTP, supporting methods such as GET, POST, PUT, and DELETE. Additionally, CoAP provides an *observe* mechanism that enables clients to subscribe to resources and receive notifications upon changes.

Henceforth, references to HTTP requests could be interpreted as referring to a **CoAP request**. Moreover, my protocol is designed to be compatible with IoT devices where bandwidth, data rate, and power consumption are critical factors. CoAP's lightweight nature and efficient use of resources make it a suitable choice for such applications.

2.2 Publish-Subscribe Protocol

Publish-Subscribe (Pub/Sub) is an asynchronous messaging pattern where senders (publishers) categorize messages into topics without the knowledge of the receivers (subscribers). Subscribers express interest in specific topics and receive messages published on those topics. This decoupled architecture enables scalable and flexible communication in distributed systems.

2.2.1 MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight, open-source messaging protocol designed for constrained devices and low-bandwidth, high-latency networks. It implements the publish-subscribe pattern over TCP/IP, providing three quality service levels for message delivery and supporting various security features.

The protocol defines three main network entities:

- **Message Broker**: The central component that manages message routing between publishers and subscribers. It receives messages from publishers and forwards them to subscribers based on their subscriptions.

- **Publisher**: A client that sends messages to the broker on specific topics.
- **Subscriber**: A client that expresses interest in specific topics and receives messages published to those topics by the broker.

MQTT Quality of Service Levels

MQTT provides three quality of service (QoS) levels to ensure message delivery reliability:

- **QoS 0 (At most once)**: The message is delivered at most once, with no acknowledgment from the receiver. This level is suitable for applications where occasional message loss is acceptable.
- **QoS 1 (At least once)**: The message is guaranteed to be delivered at least once, with acknowledgment from the receiver. This level ensures that messages are not lost but may result in duplicates.
- **QoS 2 (Exactly once)**: The message is guaranteed to be delivered exactly once, using a four-step handshake process. This level provides the highest reliability but incurs more overhead.

MQTT Security

As we mentioned, one of the key features of MQTT is the possibility to scale the protocol to fit the needs of the application. This is possible by using different security features, such as TLS/SSL for secure communication, authentication mechanisms to verify client identities, and access control lists to restrict topic access. These features help protect against unauthorized access and ensure the integrity of messages exchanged between clients.

MQTT Workflow

Generally, the MQTT workflow starts with the client establishing a connection to the broker, using a TCP/IP connection with optional TLS/SSL security. Once connected, the client can publish messages to specific topics or subscribe to topics of interest. The broker then routes messages to subscribers based on their subscriptions (see Figure 2.1).

2.2.2 LA-MQTT

LA-MQTT[M+22] extends the standard MQTT protocol by incorporating location-based features. It enables spatial queries and location-aware message routing, mak-

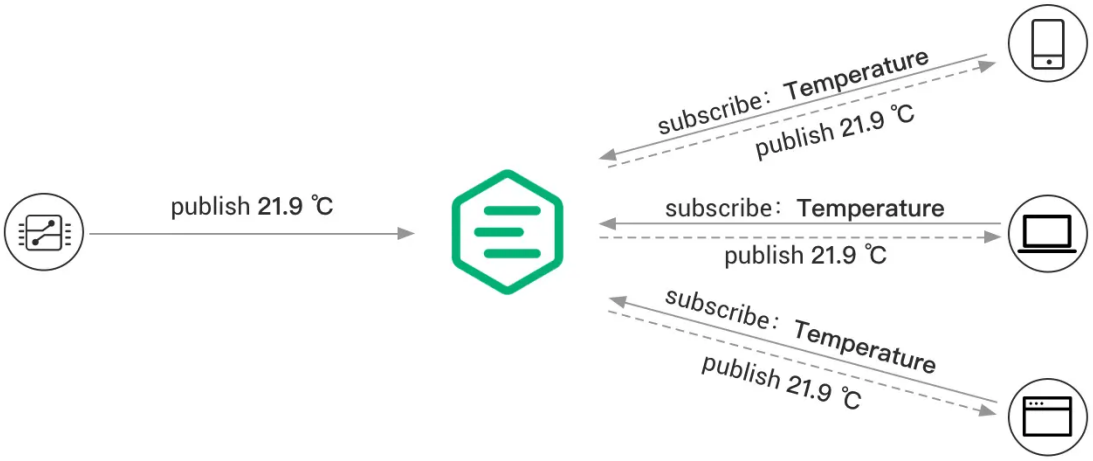

Figure 2.1: MQTT Workflow

ing it particularly suitable for IoT applications requiring geographical context in message distribution.

The protocol is first introduced to resolve the limitations of traditional MQTT in handling location-based data.

API	**Subject**	**MQTT OP**	**Topic**	**Payload**
Position publish	MC	Publish	GPS_-DATA	$\{position : (P_i)\}$
Topic subscription	MC	Subscribe	$C(i, t_s)$	*
	MC	Publish	MC_SUB	$\{mc : i, topic : t_s\}$
Geo fence publish	LDS	Publish	GEO_-FENCE_-DATA	$\{topic : t_s, content : c_s, region : g_s, event : e_s\}$
Content publish	Backend	Publish	$C(i, t_s)$	$\{content : c_s\}$

Table 2.1: The LA-MQTT Publish-subscribe Operations

Table 2.1 summarizes the main operations of the LA-MQTT protocol, highlighting the interactions between clients (MC), location data sources (LDS), and the backend system.

Those operations include:

- **Position Publish**: MC publishes their GPS data to the broker, allowing other clients to receive updates on their positions.

- **Topic Subscription**: MCs subscribe to specific topics, enabling them to receive messages related to their areas of interest.
- **Geo fence Publish**: LDSs publish geo fence data, which includes the topic, content, region, and event associated with the Geo fence.
- **Content Publish**: The backend publishes content related to the subscribed topics, forwarding it to the subscribed MCs.

LA-MQTT integrates two privacy-preserving strategies within its client-side architecture: The first strategy is based on randomized location perturbation. This method applies controlled noise to the GPS coordinates before transmission. Specifically, for a given GPS value P_i, a user-defined number of decimal digits is preserved, while the remaining digits are randomly replaced. This approach balances between:

- Privacy Preservation (PP): Higher randomness enhances anonymity.
- Spatial Precision (SP): Excessive perturbation can degrade the accuracy of spatial notifications.

The second strategy involves the use of dummy updates. Here, the MC alternates between sending real and synthetic (dummy) location data. In each sequence of updates, only one is the actual position; the others are randomly generated or trajectory-based decoys.

2.3 Universal Location Referencing

Universal Location Referencing provides standardized methods for encoding and representing geographical locations. These systems ensure consistent and unambiguous location representation across different applications and platforms.

2.3.1 Background: Distance Measures Between Vectors

In many applications, especially related to positioning and spatial data, it is essential to measure the similarity or dissimilarity between vectors. This is particularly relevant in fields such as machine learning, computer vision, and geographic information systems. To analyze similarity or dissimilarity between vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, three standard distance metrics are:

Cosine similarity:

$$\mathrm{CS}(\mathbf{x}, \mathbf{y}) = \frac{\sum_i x_i y_i}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}.$$

This measures the angle between vectors and is scale-invariant.

Euclidean distance:

$$\mathrm{ED}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_i (x_i - y_i)^2}.$$

It represents the straight-line distance but involves a non-linear square root.

Manhattan distance:

$$\mathrm{MD}(\mathbf{x}, \mathbf{y}) = \sum_i |x_i - y_i|.$$

Also known as the ℓ_1 norm, it sums up absolute differences.

2.3.2 Cantor Pairing

Cantor Pairing is a mathematical technique that uniquely maps two natural numbers to a single natural number (Figure 2.2). This bijective function $\pi : \mathbb{N} \to \mathbb{N}$ is particularly useful in computer science for combining two coordinates into a single value while maintaining the ability to recover the original coordinates.

More formally, the Cantor pairing function is defined as:

$$\pi(x, y) = \frac{(x + y)(x + y + 1)}{2} + y$$

Although this function does not preserve algebraic properties, it provides some unique properties derived from the fact that it segments the two-dimensional space into a zig-zag pattern.

$$\pi(x, y) + 1 = \pi(x - 1, y + 1)$$

Moreover, we also need to define the behavior of the function when one hits the boundaries of the first quadrant:

$$\pi(x, 0) + 1 = \pi(x + 1, 0)$$

At last, we denote the starting point of the Cantor pairing function as $\pi(0, 0) = 0$. This means that the function starts at the origin of the two-dimensional space and maps it to zero in the one-dimensional space. The inverse of the Cantor pairing function can be computed as follows:

$$\pi^{-1}(z) = \left(\frac{n(n+1)}{2} - z, z - \frac{n(n+1)}{2} \right)$$

Where n is the largest integer such that $\frac{n(n+1)}{2} \leq z$. This allows us to retrieve the original coordinates (x, y) from the single value z.

This function is widely used in computer science, particularly in data structures and algorithms, where it is necessary to map multi-dimensional data to a single dimension for efficient storage and retrieval. It is also used in various applications such as database indexing, spatial data representation, and cryptography.

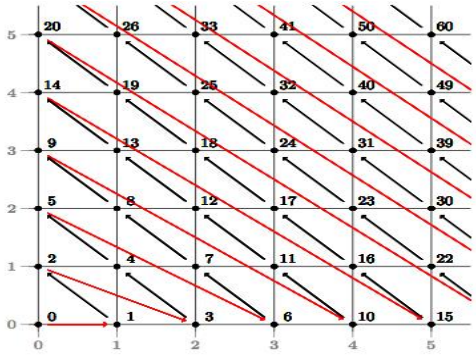

Figure 2.2: Visualization of the Cantor pairing function mapping two-dimensional coordinates to a single value

2.4 Space Filling Curves

Space-filling curves are mathematical curves that pass through every point in a multi-dimensional space. They provide a way to map multi-dimensional data to a single dimension while preserving spatial locality, making them valuable for spatial indexing and data organization.

The most common space-filling curves include (Figure 2.3):

- **Z-order Curve**
- **Hilbert Curve**

The Hilbert Curve is particularly notable for its ability to preserve locality, meaning that points that are close in multi-dimensional space remain close in the one-dimensional representation.

On the other hand, the Z-order Curve, preserves the order of positions in a grid-like manner, making it suitable for applications requiring efficient spatial queries.

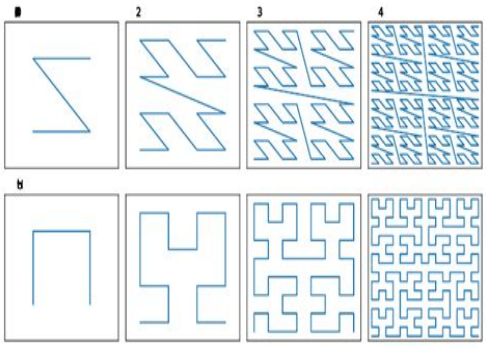

Figure 2.3: Comparison of Z-order and Hilbert curves in two-dimensional space

Thus, it's possible to reduce the problem of finding matching points to finding the maximum prefix of two-bit strings.

2.4.1 Z-order Curve

As mentioned, the Z-order curve is one of the most widely used space-filling curves. It maps multi-dimensional data into a single dimension, similar to the Cantor encoding function. Conversely, it has some useful properties, when applied to spatial data, such as preserving locality and allowing efficient range queries.

Z-order Encoding

The encoding process for Z-order involves interleaving the bits of the coordinates of a point in a multi-dimensional space. For example, given a point with coordinates (x, y), the Z-order encoding can be represented as:

$$Z(x, y) = \sum_{i=0}^{n} (x_i \cdot 2^{2i} + y_i \cdot 2^{2i+1})$$

Where:

- x_i and y_i are the bits of the binary representation of the coordinates x and y.
- n is the number of bits used to represent each coordinate.

The Z-order curve can also be applied to encode vectors with dimensions greater than two. In this case, the encoding process involves interleaving the bits of all coordinates in a similar manner.

Z-order Decoding

The decoding process retrieves the original coordinates from the Z-order encoded value. Given a Z-order value Z, the decoding can be performed by extracting the bits corresponding to each coordinate:

$$x = \sum_{i=0}^{n} (Z \pmod{2^{2i+2}}) \cdot 2^i$$

Where:

- $Z \pmod{2^{2i+2}}$ extracts the bits corresponding to the i-th coordinate.
- The result is then shifted and combined to reconstruct the original coordinate x.

The same process can be used to find the y value by de-interleaving the other bits.

Z-order Querying

Let us consider a scenario where we want to find all points within a specific range in a two-dimensional space. We define P as the set of points in the space, and we want to find all points $p \in P$ such that:

$$p.x \in [x_{min}, x_{max}] \quad \text{and} \quad p.y \in [y_{min}, y_{max}]$$

We can leverage the Z-order encoding to efficiently query this range. This is possible by calculating the order of the encoding that we want to find. By definition, we can represent the GPS coordinates of a point (x_p, y_p) into a point e_p. This point will be represented in the space of n orders, each contained in $\mathbb{Z}_4$.

Maximum Common Prefix Search

A key operation in Z-order-based spatial queries is finding the maximum common prefix between two Z-order encoded values. This operation is fundamental for determining spatial relationships between points.

Given two Z-order encoded values Z_1 and Z_2, we can find the maximum common prefix by following three steps. Firstly, we convert both values to their binary representation. Then, we compare the bits from left to right, stopping at the first position where the bits differ. Finally, we extract the common prefix up to that point. The cited algorithm shows that in the worst case, the maximum common prefix can be found in $O(n)$ time, where n is the number of bits in the Z-order encoding.

The length of the common prefix determines the size of the smallest bounding box that contains both points. This property is particularly useful for:

- Finding the smallest region containing multiple points
- Determining if points are within a certain distance of each other
- Optimizing spatial range queries

For example, consider two points with Z-order encodings:

$$Z_1 = 3310_4$$
$$Z_2 = 3312_4$$

The maximum common prefix is 331_4, indicating that these points share the same region in the first three orders of the encoded space. We used base 4 because it's the easiest way to visualize which of the four quadrants the point belongs to, as each digit represents a quadrant in a two-dimensional space.

This prefix-based approach can be extended to handle a range of queries by:

1. Encoding the query range boundaries
2. Finding the maximum common prefix of the range
3. Generating all possible Z-order values that share this prefix

The efficiency of this approach comes from the fact that we can perform these operations using simple bitwise operations, making it suitable for real-time applications.

2.5 Privacy Preserving Techniques

Privacy-preserving techniques ensure the protection of sensitive information while allowing necessary computations and data processing. These methods are crucial in maintaining confidentiality in distributed systems and data analysis. For example, in the context of location-based services, privacy-preserving techniques allow for the sharing of location data without revealing exact coordinates, thus protecting user privacy.

2.5.1 Homomorphic Encryption

Homomorphic Encryption(HE) is a form of encryption that allows specific types of computations to be performed on cipher text, producing an encrypted result that, when decrypted, matches the result of operations performed on the plain text.

Let us denote ξ_k the encryption function, ξ_k^{-1} the decryption function, and f a function that can be computed on plain texts. Homomorphic Encryption satisfies the property:

$$\xi_k(f(x)) = f(\xi_k(x))$$

In recent years, several Homomorphic Encryption schemes have been proposed, each with different properties and capabilities. The scheme used in this thesis is the Brakerski-Gentry-Vaikuntanathan (BGV, 2011) scheme[BGV12], which is a leveled fully homomorphic encryption scheme that supports both addition and multiplication operations on encrypted data. The BGV scheme is particularly notable for its efficiency and ability to handle large integers, making it suitable for practical applications in privacy-preserving computations. This scheme was based on the security of **(Ring) Learning With Errors** (RLWE) (see Section 2.5.1) problem, which is a hard problem in lattice-based cryptography. The need for such a scheme arises from the increasing demand for secure computations in various fields, including cloud computing, data analysis, and machine learning. This type of new technology is designed to resist quantum computers and cryptanalysis.

Security foundation: (Ring) Learning With Errors (RLWE)

The security of lattice-based FHE schemes, especially BGV, rests on the hardness of the Ring Learning With Errors (RLWE) problem, a ring-based variant of the Learning With Errors (LWE) problem introduced by Lyubashevsky, Peikert, and Regev in 2010[LPR10]. In RLWE, one works over a polynomial ring modulo both a prime q and an irreducible polynomial $a(x)$:

$$a(x) = a_0 + a_1 x + ... + a_{n-1} x^{n-1}, \text{where } a_i \in \mathbb{Z}_q$$

Samples are of the form $(a(x),\ b(x) = a(x)s(x) + e(x))$, where $e(x)$ is a small "error" polynomial. Recovering $s(x)$ given many such samples is presumed hard, based on reductions to the Shortest Vector Problem (SVP) in ideal lattices.

2.5.2 Homomorphic Encryption Types

Fully Homomorphic Encryption (FHE) allows the evaluation of arbitrary circuits of additions and multiplications over encrypted data, without decryption. However, earlier FHE schemes suffered from inefficiencies, particularly due to the large growth of noise. The BGV scheme answered these challenges by avoiding Gentry's

"bootstrapping"[BGV12] step via *leveled* evaluation, and controlling noise through *modulus switching* and *relinearization* techniques.

Partially Homomorphic Encryption (PHE) supports only one type of operation—either addition (e.g., Paillier) or multiplication (e.g., RSA variants). These are faster than FHE but limited in expressiveness, making them suitable for simpler tasks where only one operation type is required.

2.5.3 HE Translation Key

Originally introduced in 1998, Blaze, Bleumer, and Strauss (BBS)[BBS98] proposed an application called atomic proxy re-encryption, the mechanism enables ciphertexts encrypted under one public key to be transformed into ciphertexts decryptable under another key—without exposing the underlying plaintexts or private keys. It addresses the challenge of heterogeneity in multi-actor systems, eliminating the need for a central trusted broker to perform decryption and re-encryption. Instead, each data owner can generate a *Translation Key* that authorizes on-the-fly ciphertext conversion by other parties.

Consider a privacy-preserving workflow involving three participants—Alice, Bob, and Charlie—each with their respective keypairs (k_A^+, k_A^-), (k_B^+, k_B^-), and (k_C^+, k_C^-). Suppose Alice wants to store encrypted data on Bob's untrusted server. She encrypts her data under k_A^+ and uploads the ciphertext. Since Bob cannot decrypt (he lacks k_A^-), Charlie would also be unable to access the data directly. However, if Alice generates and distributes a Translation Key $k_{A\to C}$, Charlie can convert the ciphertext into one encrypted under k_C^+ and then decrypt it with k_C^-, without learning Alice's private key or Bob's data.

The Translation Key creation requires Alice's Private key and Charlie's Public key, ensuring that only authorized parties can perform the conversion.

$$k_{A\to C} = f(k_A^-, k_C^+)$$

By allowing secure key-conditional ciphertext conversion, the Translation Key mechanism preserves confidentiality across distinct encryption domains. This is particularly valuable when multiple data owners—with different keys—need to perform joint encrypted computations on a shared ciphertext under a common key.

2.6 Usage of HE for Matching

Homomorphic Encryption can be used to securely match location. To archive this, we can use different techniques such as:

- **Distance Calculation**: It is archived by tweaking the distance calculation algorithms (mentioned before in Section 2.3.1 e.g., Euclidean distance, Cosine similarity, Manhattan Distance) to work with encrypted coordinates. The main limit comes with the constraint that the operations must be compatible with the encryption scheme used. For instance, calculating the square Euclidean distance between two points (x_1, y_1) and (x_2, y_2) can be expressed as:
$$d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$$
That can be computed homomorphically and then decrypted before computing the square root to get the actual distance. We will further discuss this in the Testing Section 4.2.
- **Encoding Coordinates**: By encoding coordinates into a grid-like structure, we can reduce the problem into finding two different points sharing the same area-encoding. This approach comes with some limitations, mainly related to the precision of the approximation and the size of the grid. Still, it allows for efficient matching of points within a specific area without revealing their exact coordinates. This is done by applying a space-filling curve, such as Z-order (Section 2.4.1), to the coordinates.

Privacy-Preserving Z-order Queries

When implementing privacy-preserving location queries using Z-order encoding, we can employ homomorphic encryption to protect sensitive location data[ZLW20]. Let us consider a scenario where a user A wants to share their location with user B while maintaining privacy:

- Let QK_A be the QuadKey representation of A's location
- Let M_B be the bit mask specified by A for user B
- The service provider (SP) performs homomorphic multiplication: $QK_A \otimes M_B$

To ensure unambiguous results, we increment each value in $qk_i \in QK_A$ by one before encryption, such that $qk_i \in \{1, 2, 3, 4\}$. In this scheme[ZLW20]:

- A bitmask value of 1 preserves the location data
- A bitmask value of 0 masks the location data

After decryption on the client device, we:

1. Remove any zero values
2. Convert the elements back to $\mathbb{Z}_4$
3. Generate a masked QuadKey string that produces a bounding box with the desired level of detail

This approach is computationally efficient as it requires only one round of homomorphic multiplication. The resulting bounding box effectively hides both precise locations and movement patterns, providing privacy even against colluding users.

For example, given a precise GPS coordinate $(43.084451, -77.680069)$, the system can generate bounding boxes of varying sizes based on the privacy preferences (where the distance d represents the level of detail). This ensures that location data is shared at an appropriate granularity while maintaining user privacy.

2.6.1 PHE and FHE Implications

The choice between Partially and Fully Homomorphic Encryption has significant implications for system performance, security, and functionality. As previously discussed, the two methods used for proximity checks in a location-based scenario require careful consideration of the encryption scheme.

If the requirements only involve checking whether two positions share a common area, leveraging the speed and simplicity of the Z-order approach is recommended. This method allows the usage of PHE to perform fast proximity checks without significant overhead. As we have seen in Section 2.6, this approach can efficiently determine if two points are within the same area by subtracting their Z-order encodings.

Conversely, if the goal is to compute a precise floating-point distance value, a homomorphically encrypted distance function (e.g., Euclidean distance) may be necessary (see Section 4.2). However, it is important to note that in this case, the final result may be affected by computational noise inherent to FHE operations. Additionally, not all FHE schemes support floating-point operations directly, which can complicate the implementation because of the need to convert between integer and floating-point representations.

To conclude, the choice between PHE and FHE depends on the specific requirements of the application. One of the main advantages of PHE is its speed and simplicity, even though it is limited to a single operation type. On the other hand, FHE provides greater flexibility and expressiveness, allowing for complex computations

on encrypted data. However, it comes with increased computational overhead and potential noise issues, that cannot be ignored in a production-ready system.

Chapter 3

System Architecture

In this chapter, we present the system architecture for the proposed privacy-preserving location-matching protocol. The main idea is to apply the principles of homomorphic encryption to location-based services, allowing clients to securely publish and perform queries using their positions without revealing sensitive information.

3.1 Use Case

To clearly understand the system operations, let's first analyze the use case of a client who wants to find the closest parking spot available. The client needs to compute the distances between its position and the parking spots, and finally decide whether to park or not and where.

Even though it seems a straightforward mechanism, if we want to move the operations to the server, in order to preserve computational resources on the client side, a lot of challenges arise. The first one is to securely share the position of the client with the server, without exposing it to the workers. The second one is not to leak sensible information about the clients after the computation is finished.

In this scenario, it's not enough to simply encrypt the positions of the clients, as the workers need to compute the distances between the positions of the clients and the parking spots, which are also encrypted with a different key. To resolve this problem, we could just leave the parking position in clear text on the database, but this would expose the parking spots to the workers, who could then leak the information for profit. Thus, by selling the parking spots to third parties, they could be compromising the privacy of the system. In order to avoid this, and also achieve a full privacy-oriented protocol, we need to use a re-encryption mechanism, that

allows the workers to compute the distances between the positions of the clients and the parking spots, without revealing the positions of the clients to the workers.

The distance problem is avoided by using the Z-order encoding, which allows us to reduce the problem of finding matching points to finding the maximum prefix of two-bit strings. In our use case, the simulated server will perform a simple encrypted subtraction between the positions of the clients and the parking spots. At last, only the client will be able to decrypt the result, which is a simple integer value that represents the distance between the two points (If the value is 0 is a perfect match). The client can then use this value to decide whether to park or not, and where.

3.2 System Overview

The protocol is designed to combine a standard request-response protocol with a publish-subscribe mechanism.

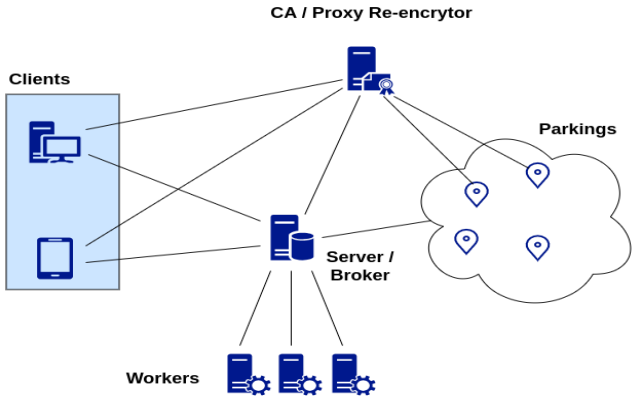

Figure 3.1: Visualization of the system architecture for the proposed privacy-preserving location matching protocol.

Homomorphic encryption incurs high computational costs, so the system optimizes performance by splitting the workload: the server handles only essential data processing, while workers compute the distances between client positions. Since HE allows computations on encrypted data, this approach preserves the privacy of clients' locations without compromising efficiency.

3.3 Actors

3.3.1 Location Data Source

The Location Data Sources (LDS), commonly referred to as sensors or parkings, are responsible for collecting and publishing location data. In our system, we assume they independently publish their data to the server, based on the event they register. For instance, if a sensor detects a free parking spot, it publishes the information to the broker, which will start the encryption protocol. This happens also when a parking spot is occupied, allowing the system to keep track of the available parking spots in real time.

Note that by maintaining the availability of the parking spots, independent from the clients, we can ensure that the system can provide privacy-preserving updates to the server. Conversely, if the clients were also responsible for reserving the parking spots, they would need to expose their positions to the server, which would compromise the privacy of the system.

3.3.2 Mobile Clients

MCs or Mobile Clients are the users of the system. Their main interest is to receive updates on the closest available parking spots, based on their current position.

In our system, the client's responsibilities are very limited, as they do not trust the server and all the other actors involved in the protocol. They only need to publish their encrypted position to the server, wait for the computation to finish, and then receive the results.

3.3.3 Server

The Backend Server (commonly called Information Broker) is the central component of the system, responsible for managing the communication between clients and sensors. It also acts like a database, handling and storing the encrypted parking spot data.

This centralized component is crucial in the architecture, as it allows all sorts of mobile actors (including clients and LDs) to have a persistent connection with the system, although it's not enough if we want to achieve a full zero-trust protocol.

Thanks to HE, the server can perform computation without knowing the clear text positions of the other actors.

The second name "Information Broker" comes from the fact that the server acts like an MQTT broker, allowing clients to subscribe to topics and receive updates on the

parking spots Figure 3.1. In order to create consistent and reliable communication between these two entities, without flooding the network with TCP/IP packets, the server uses a pub/sub mechanism, where the clients subscribe to a topic and the server publishes updates to that topic.

3.3.4 Certification Authority / Proxy

The certification authority (CA) is a trusted entity responsible for storing the public keys of the clients and managing the public/private keys of the parking spots. Because he is a trusted entity, the CA knows the private keys of the parking spots. In addition, it can generate the symmetric translation keys (see Section 2.5.3) required for secure computation of the client distances without revealing clear text positions.

In our system, the CA also acts like a proxy re-encryptor[P+17], as it is responsible for managing the right keys for the homomorphic operation made on the encrypted data.

3.3.5 Worker

The name Worker refers to a generic computational unit of the system, that can handle tasks from the server rewarded with a digital currency. In our case, the workers are responsible for computing the distances between the positions of the clients and the parking spots, using the homomorphic encryption scheme.

The workers' digital reward can be described as a token that allows them to request tasks from the server. In this way, each client can act as a worker, contributing to the overall computation of the system.

3.4 Network Protocol and Communication

3.4.1 Usage of different network protocols

The MQTT protocol is used for the communication between the clients and the server. It is necessary to use a pub/sub mechanism in order to keep alive the connection between the two entities. This is achieved by having the client subscribe to a topic, while the proxy publishes updates to that topic.

Moreover, the server is responsible for managing the connection with the LDs, allowing the clients to receive updates on the parking spots. In this way, the client MC_i **publishes** its position $Area_j$ to the topic ID_{MC_i}. The server can then apply the homomorphic operations on the data, using the keys provided by the CA.

When the computation is finished, the broker publishes the results to the topic ID_{MC_i}, allowing the client to receive updates on the parking spots. The client can then **subscribe** to the topic ID_{MC_i} to receive updates on the parking spots.

The HTTP protocol is used for the communication between the CA and the other entities of the system. Because the CA performs atomic actions, such as key generation and translation, it is necessary to use a request-response protocol to ensure that the operations are performed in a secure and reliable way.

3.5 Protocol Overview

The system is designed to allow the following operations:

API	**Subject**	**Protocol**	**Receiver**	**Payload**
Position Encoding Parameters	Clients / LDs	HTTP GET	Server	-
Key Obtain	LDs	HTTP GET	CA / Proxy	$\{id : ID_{park_j}\}$
Key Share	Client	HTTP POST	CA / Proxy	$\{id : ID_{mc}$, pub. key: $k^+_{mc}\}$
Position Publish	Client	MQTT PUB	Server	$\{position : \xi_{mc}(P_i)\}$
Topic Subscription	Client	MQTT SUB	Server	$\{topics : [position, distance]\}$
Location Publish	LDs	HTTP POST	Server	$\{\xi_{park_j}(P_j)$, $id : ID_{park_j}$, status $\in$ {free, occ.}$\}$
Translate Locations	Server	HTTP GET	CA / Proxy	$\{positions : [\xi_{park_j}(P_j)]\}$
Work Request	Worker	HTTP GET	Server	-
Task Finished	Worker	HTTP POST	Server	$\{worker : w_i$, $task : t_j$, $result : \xi_{park_j \to mc_i}(R)\}$
Distance Publish	Server	MQTT PUB	Client	$\{distances : [\forall j \in P,\ \xi_{mc_i}(R)]\}$

Table 3.1: System Operations Aligned with Architecture

3.6 Distance Preference

As we mentioned in Table 3.1, the first operation that both the MCs and LDs need to perform is to obtain the position encoding parameters from the server. In this section, we will analyze why this is necessary and how it works.

The necessity of translating a GPS reference into a grid position encoding comes with some limitations. First of all, the grid-like structure requires a fixed size for both the whole area and the single grid cell. This operation is performed by the server, in order to ensure consistency across the whole system. This operation could be performed also by single actors, but only if they are aware of a standardized measurement unit. Moreover, the subscribers could be interested in creating a custom grid, like a system some orders of magnitude larger than the others, that should still maintain the original atomic cell size.

For our study case, we will assume that the centralized server is responsible for managing the parameters. In this way, he can generate a grid that contains the geographical area of Bologna, Italy, where the simulated parking spots are located.

3.7 Scenario

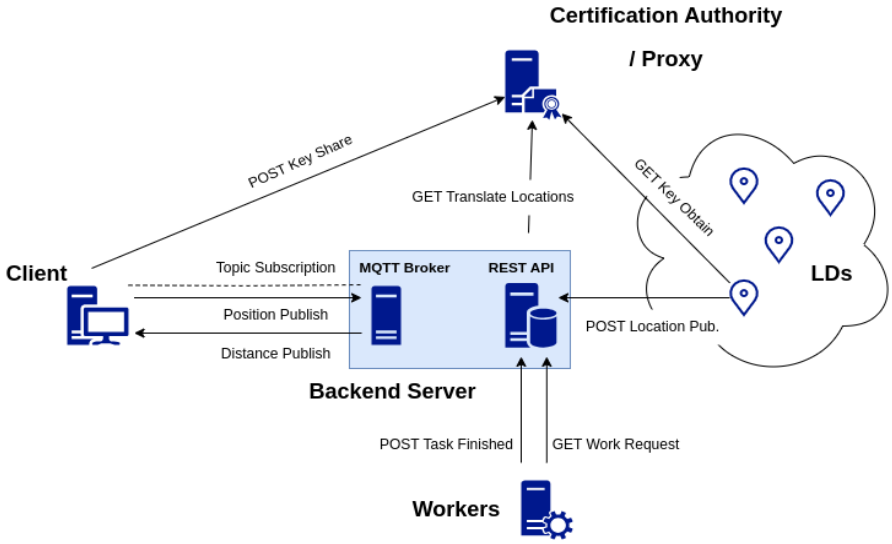

Figure 3.2: Visualization of the protocol flow

3.7.1 Protocol Flow

We can summarize the protocol flow in the following steps as shown in Figure 3.2:

1. The clients and the parking spots register to the system, after that they receive the general position encoding parameter from the server using the **Position Encoding Parameters** API via HTTP GET request. The parameters are used to encode the positions of the clients and the parking spots. *(API:* `Position Encoding Parameters`*, Payload: {None}* $\rightarrow$ $\{encoding\_params : [z\text{-}order, precision, working\ area, grid\ cell\ size]\}$*)*

2. The mobile client MC_i generates a new public/private key pair $(k^+_{mc_i}, k^-_{mc_i})$ and sends the public key to the certification authority (CA) using the **Key Share** API via an HTTP POST request. *(API:* `Key Share`*, Payload:* $\{id : ID_{mc_i},\ k^+_{mc_i}\} \rightarrow None$*)*

3. After the CA verifies the client, it generates a symmetric key $k_{mc_i,park_j}$ for each parking spot $park_j$. The operation of key generation is performed by $Key\ Translation(pp, k^+_{mc_i}, k^-_{park_j})$, where pp is the public parameters of the system, $k^+_{mc_i}$ is the public key of the client, and $k^-_{park_j}$ is the private key of the parking spot. Note that the translation operation must be done each time a new parking spot is added to the system.

4. The MC_i establishes an MQTT connection with the server and subscribes to the topic ID_{MC_i}, which is used to receive updates on the parking spots. This is done using the **Topic Subscription** API via MQTT. *(API:* `Topic Subscription`*, Payload:* $\{topics : [position, distance]\}$*)*

5. The client will then send the encrypted position $\xi_{mc}(P_i)$ to the server, where P_i is the encoded position of the client. The encryption is done using the public key $k^+_{mc_i}$, ensuring that only the client can decrypt the position. This is done using the **Position Publish** *(API:* `Position Publish`*, Payload:* $\{position : \xi_{mc}(P_i)\}$*)*. Moreover, this operation allows the client to also subscribe to the topic ID_{MC_i}, which is used to receive updates on the parking spots.

6. The server receives the location, is notified about the MQTT subscription, and starts the translation process, by invoking the **Translate Locations** API via HTTP GET request to the CA. The CA then re-encrypts the position for each parking spot using the associated key $k_{park_j \rightarrow mc_i}$, creating the re-encrypted blob $[\forall j \in \mathrm{P},\ \xi_{mc_i}(P_j)]$. *(API:* `Translate Locations`*, Payload:* $\{positions : [\xi_{park_j}(P_j)]\}$*)*

7. The server will then spread the re-encrypted positions to the workers, by invoking the **Work Request** API via HTTP GET request. The workers will then compute the distances between the client position and the parking spots, using the re-encrypted positions. *(API:* `Work Request`*,* $Payload : \{worker : w_i\} \rightarrow \{task : t_j\}$*)*

8. Once the workers compute the distances (or other metrics) between the client and parking spots, they send the result back using the **Task Finished** API via HTTP POST request. The result is a re-encrypted blob containing the distances between the client position and the parking spots. *(API:* `Task Finished`*, Payload:* $\{worker : w_i,\ task : t_j,\ result : \xi_{park_j \to mc_i}(R)\}$*)*

9. The server then publishes the aggregated results to the client via MQTT using the **Distance Publish** API. The payload contains the distances between the client's position and the parking spots, re-encrypted with the client's public key. *(API:* `Distance Publish`*, Payload:* $\{distances : [\forall j \in P,\ \xi_{mc_i}(R)]\}$*)*

10. Finally, the client receives the re-encrypted results from the topic subscription, decrypts them using its private key $k^{-}_{mc_i}$, and checks whether the location satisfies the area matching condition.

3.8 Security Considerations

The main goal of the study is to provide a privacy-preserving location-matching protocol, that allows clients to securely publish and query their positions without revealing sensitive information. In order to fulfill this goal, we need to ensure that the system is attack-resistant and that the privacy of the clients is preserved.

In the earlier approach[Gen24], the position-matching protocol used Euclidean distance calculations between client locations and parking spots provided by a trusted CA. While the server could not directly access this data, the mechanism inadvertently exposed the positions of mobile clients (MCs). The vulnerability stemmed from the lack of client authentication, enabling malicious actors to impersonate legitimate clients and intercept traffic. An attacker could exploit this by conducting a binary search on the client's location, sending iterative queries based on subscribed topics. By progressively refining the search area, the attacker could pinpoint the client's exact position, violating location privacy.

This showcased that HE is secure if and only if the encryption and decryption operation are performed correctly, preferably by the client itself.

As we also mentioned in the previous attack, the system is vulnerable to a lack of authentication of the clients. Moreover, if an attacker was able to read the network traffic, he could also manipulate the communication in order to interfere with the pub/sub mechanism.

In my implementation, I addressed those issues by introducing different techniques from the state of the art, such as using a proxy re-encryptor and a position encoding mechanism based on Z-order. The choice of leaving the decryption operation to the

client ensures that no one is able to read the sensitive data, despite an increase of computational resources required to re-encrypt each position for each parking spot.

3.8.1 Threat Model

In this section, we will analyze the threat model of the system, identifying the potential attackers and their capabilities.

3.8.2 Mitigation Strategies Malicious Actors

Adversary	**Goal / Attack**	**Mitigation Strategy**
Malicious Client	Impersonate another client or send fake position updates.	Use digital signatures: at the time of authentication with the CA, the client signs a digital contract that ensures that he is the one trying to access the system.
Malicious Worker	Manipulate computation results or leak client-sensitive data.	Because the user has no way of finding out if the information has been manipulated during the homomorphic phase, the server needs to implement a mechanism of *fake workload* to test workers' reliability.
Malicious Server	Access or manipulate sensitive client data or computational outcomes.	Because the server could only drop the communication or modify packets, but not leak sensitive information. It's suggested to have multiple backup servers independent of one from the other.
Network Attacker	Intercept and read client-sensitive data from network traffic.	All the traffic is HE encrypted.

Table 3.2: Adversarial Threats and Mitigation Strategies

In the Table 3.2, we summarize the potential adversaries and their goals, along with the mitigation strategies that can be employed to counteract their attacks. It's also important to notice that one of the assumptions of the system was that the CA is a trusted entity, moreover if an attacker is able to compromise the CA, he can

also compromise the whole system. Thus, it's not worth to consider the CA as an adversary.

Chapter 4

Testing

4.1 Testing Methodology

For the testing of the protocol, we implemented a proof-of-concept in Python, using the OpenFHE library[o+]. The testing methodology consists of the following steps. Firstly, we generate a set of random positions for the parking spots. After that, we generate a random position for the client, compute the z-order encoding for both the client and the parking spots, and finally, compute the distances between the client position and the parking spots using the homomorphic encryption scheme.

To ensure a fair comparison, we test different scenarios with different sets of hyperparameters, such as the number of parking spots, the number of clients running simultaneously, a range of network delays, and the edge size in meters of a grid cell (used to encode the GPS coordinates). The goal is to measure the performance of the homomorphic encryption scheme in terms of encryption and decryption time, as well as the time taken to compute the distances between the client position and the parking spots.

At the same time, we also computed the distance calculation using a clear computation (trusted backend) with asymmetric encrypted (RSA) traffic. This allows us to compare the performance of the homomorphic encryption scheme against a traditional asymmetric encryption scheme.

The testing environment consists of a virtual machine instance on a server, with the following specifications:

- CPU: Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz 16 cores
- RAM: 64 GB

- OS: Debian 11
- Python version: 3.8
- OpenFHE version: 1.3.0.0.20.4
- numpy version: 1.24.0

4.2 Performance Evaluation

The first test that we perform is to measure the time taken to encrypt and decrypt the client position using the homomorphic encryption scheme. The results are shown in Figure 4.1. The time taken to encrypt around 50 parking with the same key is around 0.5 seconds, while the time taken to decrypt the client position is around 0.2 seconds. This results that the homomorphic encryption scheme is efficient for this use case, but it still grows linearly with the number of data points.

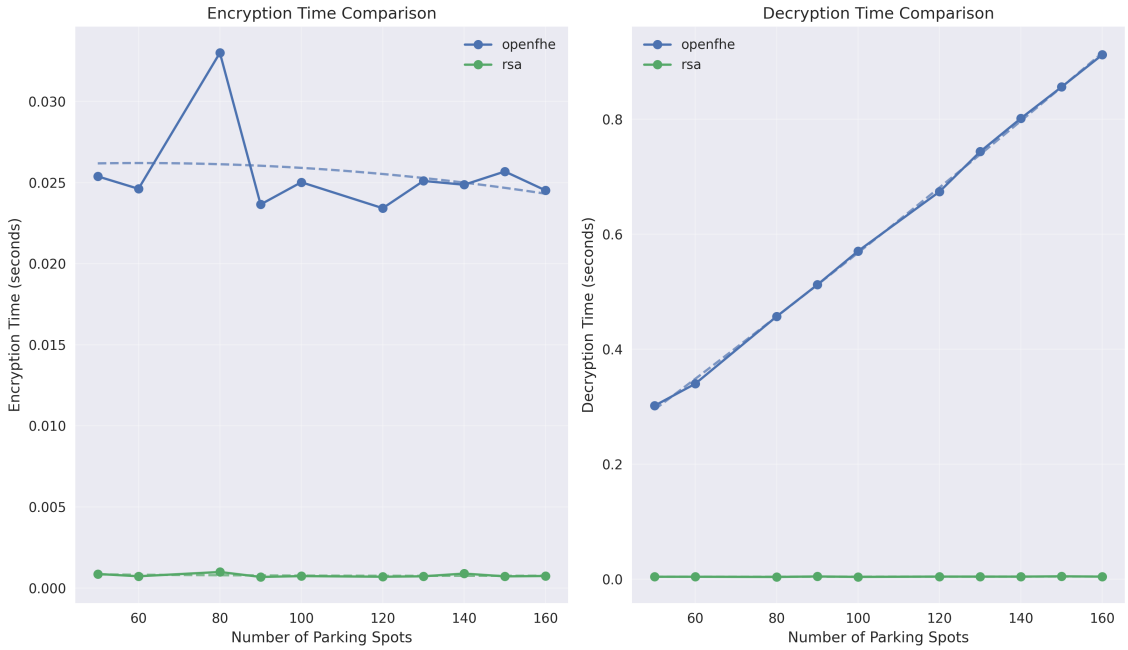

Figure 4.1: Visualization of the Encryption and Decryption time using HE

The test begins with the generation of the parameters for the homomorphic encryption scheme, followed by the key generation. The code snippet in Listing 4.1 shows the code used to encrypt and decrypt the client position.

Listing 4.1: Encryption and Decryption of the client position

```
parameters = CCParamsBGVRNS()
parameters.SetPlaintextModulus(65537)
parameters.SetMultiplicativeDepth(2)

crypto_context = GenCryptoContext(parameters)
crypto_context.Enable(PKESchemeFeature.PKE)
crypto_context.Enable(PKESchemeFeature.KEYSWITCH)
crypto_context.Enable(PKESchemeFeature.LEVELEDSHE)

keypair = crypto_context.KeyGen()
crypto_context.EvalMultKeyGen(keypair.secretKey)
```

The next step is to compute the distances between the client's position and the parking spots. The code snippet in Listing 4.2 shows the code used to compute the distances using the homomorphic encryption scheme.

Listing 4.2: Computing distances using Homomorphic Encryption

```
# Encrypt the client's position
query_plaintext = crypto_context.MakePackedPlaintext([query_encoded])
query_ciphertext = crypto_context.Encrypt(keypair.publicKey,
    query_plaintext)

# Process each spot
spots = parking_system.get_spots()

for spot_id, spot in spots.items():
    # Create plaintext for the spot's encoded position
    spot_plaintext =
        crypto_context.MakePackedPlaintext([spot['encoded_pos']])

    # Homomorphic subtraction to check if positions match
    diff_ciphertext = crypto_context.EvalSub(query_ciphertext,
        spot_plaintext)

    diff_plaintext = crypto_context.Decrypt(keypair.secretKey,
        diff_ciphertext)
```

These two sections from the original codebase showcase how the encryption distance calculation works. The full code contains additional logic for handling metrics and logging.

In order to have a fair comparison with other common encryption standards, I made the test run in parallel by multi-processing on the same machine. This refinement made the test more realistic, as in a real-world scenario, multiple clients would be sending requests to the server at the same time with multiple parking spots (subscribers).

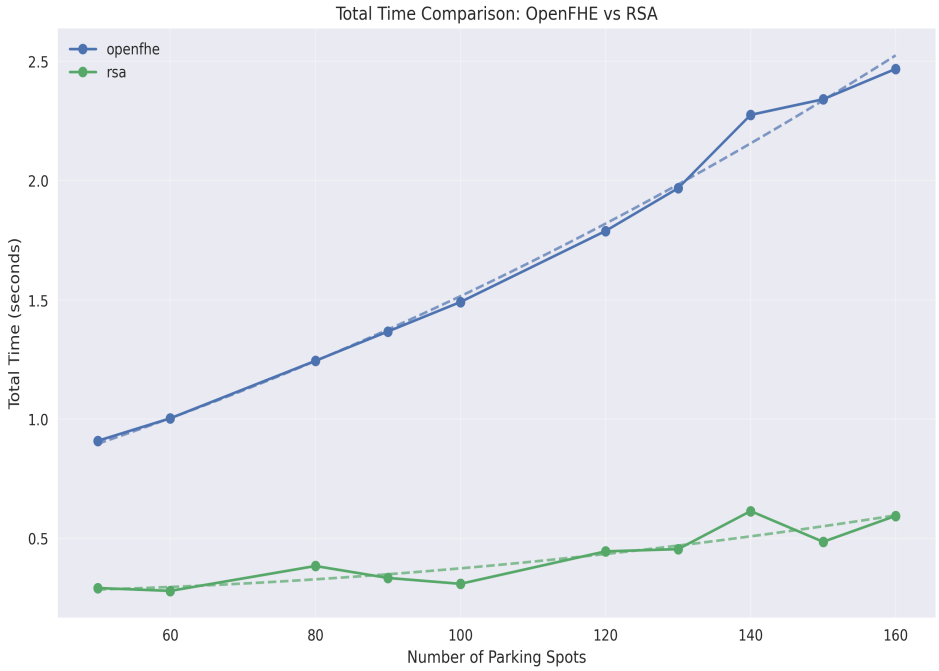

Figure 4.2: Comparison between RSA and HE for encryption and decryption

The overall time performance of the protocol is shown in Figure 4.2 confirms that the HE standard has some overhead compared to the RSA standard, but it is still acceptable for a real-world scenario. The main difference between the two encryption standards is that, with symmetric encryption, the server can compute distances much faster after decrypting the client's position, often in just a few instructions.

Although the homomorphic encryption needs to perform a simple subtraction, it is still slower, as values need to be packed to match HE standards as done in Listing 4.2.

It is also worth to mention the way the distances are computed in our protocol. Firstly, we need to encode the GPS coordinates into a grid-like structure. This operation is essential to ensure that we are considering small areas instead of points Listing 4.3. Then we apply a unique encoding to the grid coordinates, transforming them from a two-dimensional grid into a one-dimensional z-order curve. By having a single integer representation we can perform easily the distance.

Listing 4.3: Distance computation using z-order encoding

```
def calculate_grid_sizes_for_radius(radius_meters, max_lat, min_lat,
    max_lon, min_lon):
    radius_km = radius_meters / 1000.0
    lat_span = max_lat - min_lat
    lon_span = max_lon - min_lon

    # Latitude: fixed ~111.32 km/degree
    lat_cells = int(lat_span / (radius_km / 111.32))

    # Longitude: depends on latitude
    mean_lat = math.radians((max_lat + min_lat) / 2)
    lon_cells = int(lon_span / (radius_km / (111.32 *
        math.cos(mean_lat))))

    return lat_cells, lon_cells # Return separate sizes

def normalize_gps(lat, lon, lat_cells=None, lon_cells=None,
    edge_meters=500, max_lat=44.499194, min_lat=44.492307,
    max_lon=11.363250, min_lon=11.325319):

    if lat_cells is None or lon_cells is None:
        lat_cells, lon_cells =
            calculate_grid_sizes_for_radius(edge_meters, max_lat, min_lat,
            max_lon, min_lon)

    grid_x = int((lat - min_lat) / (max_lat - min_lat) * lat_cells)
    grid_y = int((lon - min_lon) / (max_lon - min_lon) * lon_cells)

    return grid_x, grid_y
```

To compute the z-order encoding, we use a straightforward approach that interleaves the bits of the x and y coordinates. The code snippet in Listing 4.4 shows how we compute the z-order encoding for the grid coordinates.

Listing 4.4: Z-order encoding for grid coordinates

```
def interleave_bits(x, y):
    """
    Interleave the bits of x and y to create a Morton code.
    x and y should be non-negative integers, each limited to 16 bits.
    """
    # Ensure inputs are within the 16-bit range
```

```
    x = min(x, 0xFFFF)
    y = min(y, 0xFFFF)

    # Convert to binary and pad with zeros to 16 bits
    x_bin = format(x, '016b')
    y_bin = format(y, '016b')

    # Interleave the bits
    result = ''
    for i in range(16):
        result += x_bin[i] + y_bin[i]

    return int(result, 2)
```

There are also other approaches to compute distances between two encrypted points[Iba22], as mentioned for the methods in Section 2.3.1:

- **Cosine similarity:** This can be resolved by normalizing the vectors, i.e., $\mathbf{x}' = \mathbf{x}/\|\mathbf{x}\|$ and $\mathbf{y}' = \mathbf{y}/\|\mathbf{y}\|$, encrypting $\mathbf{x}'$ and $\mathbf{y}'$, and then performing a simple scalar product: $\sum_i x'_i y'_i$.
- **Euclidean Distance:** would require a square root operation.
  Instead we use the **Squared Euclidean Distance** instead: $\mathrm{SED}(\mathbf{x}, \mathbf{y}) = \sum_i (x_i - y_i)^2$
- **Manhattan Distance:** in this case it also requires computing the absolute value.
  If you encrypt only binary values (i.e., $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ such that $\hat{x}_i, \hat{y}_i \in \{0, 1\}$ for all i), you can reformulate: $\mathrm{MD}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = \sum_i (\hat{x}_i - \hat{y}_i)^2 = \mathrm{HD}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ which is the **Hamming Distance** . For non-binary vectors, you can at least compute the **Squared Manhattan Distance** $\mathrm{SMD}(\mathbf{x}, \mathbf{y}) = \sum_i (x_i - y_i)^2$ but this is missing a square root to recover the standard Manhattan distance.

Rounding off, all of these methods have the problem of relying on calculations on floating point numbers, which is not ideal for homomorphic encryption. Another way could be to normalize the coordinates to integers, with a fixed order of magnitude, but it is not worth the effort, as the z-order encoding is already a good solution for this problem.

Chapter 5

Conclusion

5.1 Summary

In this thesis, I presented a protocol for finding nearby parking spots on the path of a zero-trust scenario. The protocol is based on the use of homomorphic encryption to protect the client's position and the parking spots, and it uses a combination of publish/subscribe with request/response protocols to allow clients to find nearby parking spots without revealing their position to the server. The protocol is designed to be scalable and to work in a case, where the server cannot be trusted to handle sensitive data.

5.1.1 Results

From the testing phase, we can conclude that the protocol is feasible and functional. The performance of the homomorphic encryption scheme is acceptable for a real-world scenario, although it is still computationally more expensive than a traditional RSA solution. However, it can be applied in a server zero-trust scenario, for instance where the server cannot be trusted to handle sensitive data. The introduction of PRE (Proxy Re-Encryption) allows us to offload the computation to a trusted third party, which can be used to compute the distances between the client position and the parking spots without revealing the client position to the server. Moreover, the presented approach resolves an issue present in the previous alternative solution that uses HE of the server possible leak of client position, under threat model (See Section 3.8.1).

5.1.2 Protocol Scalability

One additional factor to consider is the scalability of the protocol. The MCs and LDs can be scaled horizontally, meaning that we can add more servers to handle more clients and parking spots. The same applies to the server and workers; for instance, we can have multiple servers handling different areas of a metropolis, each with its own set of parking spots. However, the CA (Certificate Authority) cannot be scaled linearly. Indeed, the CA could have multiple instances, but they would need to be synchronized to issue compatible certificates for the same client.

5.1.3 Protocol Limitations and Security Analysis

In conclusion, the protocol presented in this thesis is a feasible solution for the problem of finding nearby parking spots in a nearly zero-trust scenario. This was possible thanks to the usage of network protocols, such as the publish/subscribe and request/response, combined with homomorphic encryption to create encrypted blobs of malleable data. During my work, I concluded that even though the publish/subscribe protocol is preferred for this kind of application, it struggles to match the requirements of a zero-trust scenario. Moreover, working with homomorphic encryption adds the requirement that the server is not allowed to know the plain text of the clients. This is a crucial point, as it does not allow us to infer information from the client data. So, the request/response protocol works better in this sense: the client requests anonymously a service, the server responds by applying a list of operations to the encrypted data, and then the client can decrypt the result. In the opposite case, the server would need to know *when* to respond to the client, which is most of the time a consequence of a side-effect to the request. Let us consider the case of the LA-MQTT protocol (see Section 2.2.2): the client subscribes to a topic, and the server publishes the result based on the client's location. In this case, the server needs to know the client's position to publish the result. Indeed, if this were the case, we would not be able to apply the homomorphic encryption scheme.

If we were to design a truly zero-trust protocol, no dependency on a central Certificate Authority (CA) would be necessary. This is a common practice today, especially in blockchain-based systems, where the CA is replaced by a distributed ledger. However, such a change would require a complete redesign of the protocol, as the CA currently handles certificate issuance for both clients and parking spots. It would also necessitate a different approach to client and parking spot authentication and authorization. Furthermore, this change would significantly increase the computational burden on each sensor. Each sensor would first need to verify whether a client is authorized to access its data, and then encrypt the data using the client's public key. In our case study, this is not feasible, as Location Detectors (LDs) are resource-constrained devices that cannot perform such complex and frequent cryptographic

operations.

5.2 Future Works

The testing phase revealed several areas for future work. One of them is the possible integration of IoT-specific libraries, such as the *SEAL Embedded* [ND21] library, which is designed for resource-constrained devices. This would allow us to run the protocol on edge devices, such as sensors or microcontrollers more efficiently. However, this change would require adapting the entire system to use the Microsoft SEAL [SEA23] library across all components.

Another significant area of potential optimization involves adjacency calculation using z-order encoding and Fully Homomorphic Encryption (FHE). In [ZLW20], the authors propose a standard for computing meaningful queries over homomorphically encrypted, position-related data. Although their context is a social media platform tracking user locations to suggest nearby friends, the methods are highly relevant here. Their approach uses z-order encoding not only to encode spatial data but also to support privacy-preserving preference levels. For example, a user may wish to share their location with a friend, but not reveal the exact position. By applying a preference bitmask, the user can selectively disclose only a coarser granularity of the z-order encoding. The paper [ZLW20] also demonstrates how to compute the exact distance between two points by constructing the smallest encoded bounding box that contains them—entirely under encryption. Integrating such concepts into our protocol could allow us not only to identify parking spots in the same area but also to estimate distances securely. Our encoding strategy is based on converting GPS coordinates into z-order curves, offering an approximate but compact representation of spatial areas. While this encoding introduces a certain error margin, the trade-off may be acceptable—or even beneficial—as it contributes to obfuscating the precise location of the user.

References

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Lecture Notes in Computer Science*, Lecture notes in computer science, pages 127–144. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, New York, NY, USA, January 2012. ACM.

[BNT19] Meriem Bettayeb, Qassim Nasir, and Manar Abu Talib. Firmware update attacks and security for IoT devices. In *Proceedings of the ArabWIC 6th Annual International Conference Research Track*, pages 1–6, New York, NY, USA, March 2019. ACM.

[Gen24] Luca Genova. Privacy nei servizi location-based service: Utilizzo della crittografia omomorfica con la-mqtt, 2024.

[Iba22] Alberto Ibarrondo. Pyfhel: Distance coordinates distance calculation tricks. `"https://github.com/ibarrond/Pyfhel/issues/155"`, 2022.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology – EUROCRYPT 2010*, Lecture notes in computer science, pages 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[M+22] Federico Montori et al. La-mqtt: Location-aware publish-subscribe communications for the internet of things. *ACM Transactions on Internet of Things*, page 28, 2022.

[M+23] Muhammad Mazhar et al. Security challenges in iot devices: A comprehensive review. *Internet of Things*, 22:100779, 2023.

[ND21] Deepika Natarajan and Wei Dai. SEAL-embedded: A homomorphic encryption library for the internet of things. *IACR Transactions on Cryp-*

tographic Hardware and Embedded Systems, (3):756–779, July 2021. https://tches.iacr.org/index.php/TCHES/article/view/8991.

[o+] OpenFHE org et al. openfhe-python.

[P+17] POLYAKOV et al. Fast proxy re-encryption for publish/subscribe systems. *ACM Transactions on Privacy and Security*, 20(4), 2017.

[PDVRC21] Muhammad Talha Paracha, Daniel J. Dubois, Narseo Vallina-Rodriguez, and David Choffnes. Iotls: understanding tls usage in consumer iot devices. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 165–178, New York, NY, USA, 2021. Association for Computing Machinery.

[SEA23] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

[SHB14] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[ZLW20] Yifan Zhang, Yingjiu Li, and Robert H Wang. Privacy-preserving location queries with homomorphic encryption. *Computers & Security*, 99:102064, 2020.

Ringraziamenti

Credo che una singola sezione di ringraziamenti a fine tesi non sia sufficiente per riassumere tutte le persone che mi hanno seguito, accompagnato, sopportato e supportato durante questo percorso. Sono sicuro che anche chi non verrà nominato personalmente saprà che possiede un posto speciale nella mia vita e non solo su questo pezzo di carta.

Vorrei iniziare ringraziando i miei genitori Massimo e Roberta per avermi concesso questa occasione unica. Ricordo i tempi di inizio superiori in cui, per quanto abbastanza portato, non prevedevo questo futuro accademico. Con il senno di poi penso che questa esperienza possa essere inclusa nelle scelte che rifarei senza pensarci due volte. Non posso immaginare come sarebbe potuto essere questo percorso senza i preziosissimi aiuti dell'esperienza di mia madre e dall'infinita pazienza e apprensione di mio padre. Solamente quando ti rendi conto che sei alla fine ogni singola parola spesa per te assume un significato diverso da quello colto originariamente: più reale, più sentito, più tuo. Successivamente un grazie anche a Pietro che, nonostante i continui attriti, le discussioni e le diversità, so che mi augura sempre il meglio e mi sprona a modo suo; sono più che convinto che anche lui eccellerà nel suo percorso universitario e di vita.

Un infinito grazie ai quattro nonni, che ho la fortuna di avere accanto anche alla fine di questo percorso. Speravo di potermi presentare alla cerimonia con una Fellali, anche se con gli anni ho appreso che la preferisco solo come ricordo. Un grazie anche a tutti gli altri parenti che mi hanno ascoltato durante le mie lamentele di un diciotto e condiviso i momenti di gioia di un trenta (voti di materie di cui a stento capivano il nome).

Ovviamente non si possono dimenticare professori e insegnanti, passati e futuri, una dedica speciale va soprattutto a loro, che ogni giorno, lezione dopo lezione, ora dopo ora riescono a trasmettere il valore della conoscenza. Mi auguro che questo possa essere sempre trasmesso e che sia motore di ricerca della verità scientifica per gli studenti futuri.

Vorrei anche ringraziare tutti gli amici e conoscenti: quelli che conosco da una vita, quelli che ho incontrato da poco, quelli con cui ho perso i rapporti e quelli con cui li ho riallacciati. Penso che ognuno di loro mi abbia fatto in parte cambiare e crescere a modo suo, permettendo tutto questo. Spesso si sente dire che questi saranno gli anni migliori della nostra vita, forse è ancora troppo presto per smentire o confermare ciò ma, grazie a voi, so di aver condiviso delle esperienze fenomenali e affrontato i minimi più bassi. Non smetterò mai di portarvi con me e, anche se le prospettive future attuali prevedono due anni lontano, in queste storie non c'è ancora scritta la parola fine.

Per finire, un piccolo grazie anche a me stesso, per ricordarmi di cosa sono capace dopo tutto quello a cui sono andato in contro ed ho affrontato a testa alta. Chiudo così questi ultimi tre anni di vita, che siano la rampa di lancio per qualcosa di incredibile.