CryptPath

Data-oblivious Shortest Path Discovery in Outsourced Settings

IN5000: Master Thesis

Andrei Geadau

Delft University of Technology

TU Delft

CryptPath

Data-oblivious Shortest Path Discovery in Outsourced Settings

by

Andrei Geadau

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday, July 18, 2023, at 10:30 PM.

Student number:	4850076
Project Duration:	November 14, 2022 - July 18, 2023
Thesis committee:	Asst. Prof. dr. Z. Erkin
	Asst. Prof. dr. S. Proksch

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Cover image created with the assistance of DALL·E 2 (Modified)
Thesis contents enhanced with the assistance of ChatGPT

TU Delft

Preface

This thesis describes my solution to securely outsourcing the shortest path problem in the context of privacy-preserving navigation systems. This works culminates an intense and enriching nine months journey in the field of secure outsource computation. It all started after taking Privacy Enhancing Technologies course by Asst. Prof. dr. Z. Erkin - who later become my supervisor.

Undertaking this endeavour would not have been possible without his guidance and support. There has never been someone to inspire me more to give the best I can, both as a researcher and as a person. I also express my heartfelt appreciation to my daily supervisor, F. Dekker, whose expertise and patience were instrumental in shaping this work.

I cannot forget about other faculty members of the Cybersecurity group, whose commitment to academic excellence has created a pleasant environment to be part of and share great ideas.

I would like to extend my gratitude to my friends for the memorable uni experience. Special thanks to Dan Andreescu, Jorrit van Assen, Ion Babalau, Bob Brockbernd, Marin Duroyon, Ivo Kroskinski, Dan Plamadeala, Konrad Ponichtera, Mariana Samardzic, Ioana Savu, Bogdan Simion, Natalia Struharova, Wessel Thomas and Bram Verboom.

Finally, I would like to thank my parents for their unconditional support throughout these five years of studying abroad.

Andrei Geadau
Delft, July 2023

Summary

Cloud computing and storage solutions have been credited with increasing competitiveness through cost savings, greater flexibility, elasticity and optimal resource allocation. However, the data outsourced to the cloud may contain private information that must be protected from misuse. In such cases, the data can only be outsourced after encryption. Graphs are widely used to model and represent data in various applications, including geographic information systems. Yet performing the shortest path algorithm over such encrypted outsourced graphs represents a challenge. Current approaches rely on trusted third parties, employ weaker security measures, or treat the cloud as a storage medium rather than addressing the algorithmic complexities directly. This makes the adoption of such protocols difficult in practice.

In this work, we develop two privacy-preserving navigation protocols to compute the shortest path over encrypted outsourced graphs, under different security and efficiency guarantees. The first protocol enables one to retrieve the shortest path in an oblivious manner, that is, without the cloud learning any information about the outsourced graph or the returned path. The second protocol assumes that the topology of the outsourced graph is public knowledge, and obtains retrieval times orders of magnitude faster. The evaluation results on a proof-of-concept implementation indicate that the condition number of the node-incidence matrix of the graph being outsourced serves as a determining factor for the number of iterations. Dense graphs exhibit a low condition number. In such cases, the high number of iterations required for convergence, quadratic time complexity with respect to the number of edges in the graph, and the inherently slow homomorphic operations results in impractical retrieval times. Conversely, sparse graphs allow for trivial resolution of shortest paths within a single iteration.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
APGD	Accelerated Projected Gradient Descent
BFV	Brakerski/Fan-Vercauteren cryptosystem
BGV	Brakerski-Gentry-Vaikuntanathan cryptosystem
CCPA	California Consumer Privacy Act
CKKS	Cheon-Kim-Kim-Song cryptosystem
DCR	Decisional Composite Residuosity problem
FHE	Fully Homomorphic Encryption
GD	Gradient Descent
GDPR	General Data Protection Regulation
GPS	Global Positioning System
HE	Homomorphic Encryption
LBS	Location Based Service
LP	Linear Programming
LWE	Learning With Errors
OPE	Order Preserving Encryption
PET	Privacy Enhancing Technologies
PGD	Projected Graient Descent
PHE	Partially Homomorphic Encryption
PIR	Private Information Retrieval
PPT	Probabilistic Polynomial Time
PSP	Privacy-Preserving Shortest Path
PSPEG	Privacy-Preserving Shortest Path over Encrypted Graphs
QP	Quadratic Programming
RLWE	Ring Learning With Errors

List of Algorithms

List of Tables

List of Figures

1 Introduction

From ride-sharing services to efficient logistics that optimize resource allocation, location data enables us to navigate the complexities of our modern society. Yet we live in an era of widespread surveillance, where big-tech companies have political and economic interests in our location data. It becomes imperative to safeguard this sensitive information. Nothing can motivate the need for location privacy more than the Family Locator vulnerability, where the real-time location of over 230.000 users was publicly accessible [1]. Governments are beginning to acknowledge the dangers posed by data misuse, leading to the emergence of regulations. However, governments themselves are also guilty of invading privacy for surveillance purposes [2].

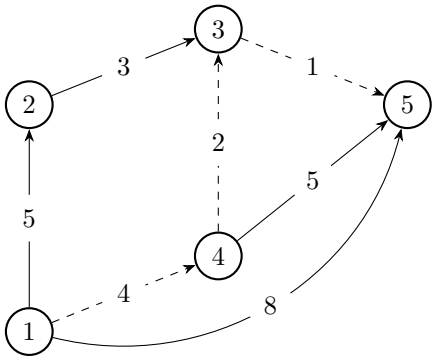
One potential solution to address both issues lies in embracing privacy-preserving navigation systems, which allow individuals to find the shortest path to their destination while ensuring the confidentiality of their location. It is important to recognize that privacy-preserving navigation systems cannot be applied to all scenarios. Every use case has different requirements, for example, in terms of security, reliability, performance, and usability, that must be addressed in order to make the adoption of these schemes more compelling and feasible in practice.

1.1. Navigation Systems

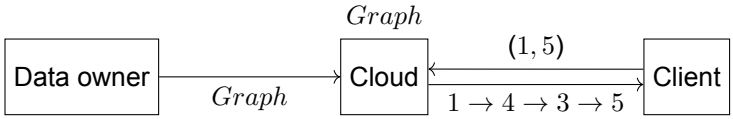
The ubiquitous presence of GPS, combined with the widespread adoption of smartphones, has fostered a rapidly growing market for location-based services (LBSs). Google Maps, Waze, and Apple Maps are utilised by more than 90% of internet users on a regular basis, as reported by a recent study [3]. Users of these innovative solutions can harness the power of their mobile devices to obtain directional assistance to their desired destinations and discover nearby amenities such as healthcare facilities, pharmacies, and law enforcement offices.

In this work, we focus on road transportation. A road network can be represented as a graph, where intersections (often referred to as ”nodes” in technical terms) are connected by streets (often called ”edges”). Each edge carries a weight, symbolizing factors like distance travelled, drive duration, or toll charges linked to the respective segment of the road. Suppose you find yourself positioned at a certain node and want to get to another one. The goal is to find the quickest, shortest, or cheapest route. For instance, consider a simple scenario with five intersections connected by seven streets as in Figure 1.1a. To get from node 1 to node 5, the shortest path between them is $1 \to 4 \to 3 \to 5$ with total cost (i.e., total length) $4 + 2 + 1 = 7$.

Navigation systems are tools that find the shortest route within a road network, usually by computing the shortest path between a specified start and end nodes. Generally, the data pertaining to the road networks is in possession of, and is updated by, a government or transportation authority, known as the *data owner*. LBSs, like Google Maps, acquire this data from the data owner and store it on cloud computers. Then, they utilize this data to calculate and provide the shortest possible routes to *clients* as requested. Figure 1.1b showcases this scenario.



(a) Dashed lines represent the shortest path between node 1 and 5.



(b) Outsourcing architecture.

Navigation systems are not exactly a novel concept. As far back as the times of the Roman Empire, citizens utilized various navigation systems to traverse the extensive network of roads and waterways that spanned across their dominion. The Romans relied on 'itinerarium', detailed road maps that provided directions, landmarks, and estimated distances between towns and settlements. This allowed merchants, soldiers and pastors to travel efficiently and securely across vast distances, connecting distant provinces and ensuring the smooth flow of goods, information, and people throughout the empire. Since the days of the Roman Empire, navigation systems have undergone a remarkable evolution, driven by advancements in technology and scientific understanding. The development of more precise and sophisticated instruments, along with the advent of digital technologies, has revolutionized the way we navigate and explore the world. Today, navigation systems have become an integral part of our daily lives. GPS technology has been integrated into smartphones, cars, aircrafts, and even wearable devices, providing us with real-time directions or traffic updates. Currently, we possess the ability to calculate the distance between any two cities within seconds, considering variables like traffic conditions, public transportation schedules, as well as any other factors, such as road closures or constructions.

Navigation systems can be used to design innovative products and services. Navigation systems are the cornerstone of ride-sharing platforms like Uber or Lyft. The navigation system not only helps drivers find the quickest route to the passenger's destination, but also aids in the dynamic pricing model by understanding traffic patterns. This ensures that drivers can pick up and drop off multiple passengers efficiently, maximizing their earnings and minimizing passenger wait times. Companies involved in courier services, food delivery, and logistic operations also heavily rely on navigation systems to optimize their delivery routes, improving efficiency, and reducing delivery times.

Another application of navigation systems is child safety. Parents can utilize navigation systems to track their children's location and ensure their safety. With GPS tracking enabled on their children's mobile devices, parents can monitor their kids' location in real-time, ensuring they safely reach school or are on their way back home. For example, FamilyLocator is an app that provides such services, allowing parents to create a digital "safe zone" and alerting them if their child leaves this designated area.

1.2. Privacy Concerns

When used responsibly and ethically, navigation systems have the potential to positively impact individuals, businesses, and society alike. But navigation systems can easily be abused. In the following section, we provide instances that underscore the necessity of safeguarding location data.

Navigation systems like Google Maps, Waze, or Apple Maps provide immense value by offering accurate, real-time directions, points of interest, and traffic information. However, their use does raise

critical privacy concerns. These concerns stem from the fact that such services inherently require access to location data, which is recognized as sensitive personal information under GDPR [4]. This data, when accumulated and analyzed over time, can reveal an extensive array of details about your life, including where you live, where you work, the places you frequently visit, your daily routines, and even your personal relationships. In addition, advertisers and businesses are highly interested in this kind of data as it can be used for targeted advertising, personalized promotions, and market research. If location data is shared or sold without consent, this constitutes a violation of privacy rights. Even anonymized and aggregated data could potentially violate GDPR if the data is not handled properly. If the data can be de-anonymized – that is, linked back to specific individuals – then the sale of that data could indeed infringe upon GDPR. It is also important to note that consent must be freely given, specific, informed, and unambiguous for data processing under GDPR. Yet until 2019, Google Maps, the most popular navigation system, did not provide sufficient information to users about the purposes and processing of their data, nor did it obtain valid consent in a transparent manner. As a result of these practices, Google was fined $57 million in 2019 [5].

A ride-sharing company holds a substantial amount of personal data of its users. This information usually includes names, email addresses, phone numbers, and sometimes physical addresses. Furthermore, they also store ride history data, which includes details about where and when a user has traveled. In the wrong hands, this information can be used for various malicious activities such as identity theft, credit card fraud, and even stalking or harassment. In 2016, Uber experienced a massive data breach that exposed the personal data of 57 million users and drivers [6]. Although the American company has significantly enhanced its security protocols in response to these events, a recent hack in September 2022 [7] raises doubts about the effectiveness of its measures.

Family locator apps are also targeted for their valuable data. While the intention behind such tracking may be to ensure safety, it can easily be misused by malicious individuals or organizations. Real-time location data can be hacked, exposing children to potential kidnappings or other forms of harm. Moreover, constant monitoring may infringe upon their right to privacy and hinder their development of independence and autonomy. In 2019, it was reported that a popular app named ”Family Locator” exposed the real-time location data of 238.000 users’ for weeks due to a security flaw [1].

1.3. Privacy by Design

Historically, the concept of protecting location data as private information has gained momentum alongside the development and spread of information technology. Initially, privacy legislation focused largely on personally identifiable information, such as names, addresses, and social security numbers. However, as the proliferation of GPS-enabled devices and services grew, the potential misuse of location data began to raise concerns. The European Union’s General Data Protection Regulation (GDPR) [4], implemented in 2018, is a notable advance in this respect. It recognizes geolocation as personal data, thus requiring explicit consent for its collection and use. In the United States, several state-level laws, including the California Consumer Privacy Act (CCPA) [8] of 2018, have addressed location data privacy.

Regulations, while crucial, often struggle to keep up with the fast-paced evolution of digital technologies and methods of data collection. They are also limited by jurisdictional boundaries and face enforcement challenges. This is why concepts like privacy by design, which integrate data protection measures into the architecture of technologies and systems from the ground up, are essential. Privacy by design ensures that privacy considerations are not an afterthought but an integral part of the design process, reducing the chances of data misuse and offering protection that is not solely reliant on regulations. This methodology, initially presented in a 1995 report on privacy-enhancing technologies (PETs) [9], involves integrating provably secure mathematical protocols into existing applications.

Homomorphic Encryption (HE) is a PET for achieving privacy by design, enabling computations to be performed directly on encrypted data without needing to decrypt it first, thus maintaining data confidentiality even during processing. This has remarkable potential in fields such as cloud computing, where sensitive data can be stored and processed securely without exposing it to third-party service providers. With HE, data can remain private even during analysis or processing, which has important implications for sensitive domains like finance, healthcare, and data analytics [10]. In the context of location data, a navigation service could compute a route for a user without ever having to know their exact location, as all computations would be performed on the encrypted data.

However, despite its significant benefits, HE comes with some trade-offs. As with any privacy-enhancing-technology, HE is notorious for hindering the ability of law enforcement agencies to investigate crimes and prevent illegal activities [11]. On a similar note, there are issues related to accountability and transparency. If an error occurs or if there is a need to audit the operations performed on the data, the opaque nature of homomorphic encryption might make it difficult to identify the source of the problem or provide a clear audit trail. Finally, HE operations are computationally expensive, with operations on homomorphically encrypted data taking orders of magnitude longer than on unencrypted data [12]. This may be impractical for many real-world applications.

1.4. Adopting Privacy

Privacy for location data is crucial, but creating effective privacy-preserving navigation systems can be difficult. The nature of these systems poses significant challenges in developing privacy-preserving protocols that can protect location data while still ensuring efficiency. There is a diverse range of research that deals with distinct privacy issues related to processing road network data.

One research trajectory focuses on Privacy-Preserving Shortest Path (PSP) problem: the client queries a cloud provider who has knowledge of the outsourced graph. This is addressed in multiple studies, such as [13], [14], [15], [16]. These studies, however, expose the volume pattern of the user's query, meaning the number of elements in the resulting shortest path. This challenge has been addressed in the work of Mouratidis et al. [17], who recognize that prior works may reveal information about whether the path is short or long.

An extension of this focuses on protecting both the client's query and the graph data when outsourced to an external cloud provider. This is known as Privacy-Preserving Shortest Path discovery over Encrypted Graph (PSPEG) problem. The graph owners can benefit from cloud storage systems by outsourcing the large-scale graph data to third-party cloud providers, thus reducing maintenance and management costs. However, this could potentially lead to data leaks, threatening user privacy. In response, graph owners may alter the graph so the cloud provider cannot access its exact information, while still retaining key topological features. The challenge here relies on the fact that encrypting the graph data results in a loss of graph querying capabilities. Notable works are Blanton et al. [18], Samanthula et al. [19] and Bramm et al. [20]. Despite extensive research on the PSPEG problem, every study contains minor issues that make the adoption of such protocols difficult in practice. For example, Bramm et al. [20] leak the volume pattern, which is the number of edges (and by implication nodes) between the starting and terminal nodes of the client. The size of the result can inadvertently expose information about whether the path is short or long, which may divulge details about the client's daily schedule. If the client is travelling along a path with the same length as a previously taken one, it may indicate they are heading to a location they visit regularly, such as work or school. Conversely, if the path is unusually long, it might be a sign that the client is embarking on a vacation journey. Blanton et al. [18] suggest approaches using linear secret sharing, which require a trusted third party to perform multiplication. But trusted third parties may have access to sensitive information, which if mishandled, lost, or stolen, can lead to privacy breaches. Users must trust that these parties will handle their data responsibly, which is not always the case. Finally, in the work of Samanthula et al. [19], the client applies Dijkstra's algorithm on the subgraphs retrieved from the cloud. This raises the question as to why the data owner does not share this data directly with the client.

A different research stream is concerned with validating the accuracy and correctness of the results returned by the cloud provider. There have been proposed database authentication schemes for verifying shortest path in road networks [21]. However, these approaches are distinct from our problem, where we assume the cloud provider is curious, but not malicious – interested in the clients' queries, but not intending to distort or interfere with their results.

There have also been studies considering graph problems in a semi-trusted model, where different parties hold separate pieces of information and collaboratively respond to a query without revealing their segment of the data to each other. For example, [22] propose a protocol that enables two entities holding different portions of a graph to calculate the all-pair shortest distances in the combined graph. In [23] an entity holds the weights of the edges, while another entity holds a heuristic for finding the shortest path from a source to a destination.

1.5. Research Goals

The main goal of this thesis is to create a protocol that tackles the PSPEG problem without requiring any trusted third parties, without requiring the client to perform Dijkstra's algorithm, and without leaking the volume pattern. Our proposed protocol involves the participation of three parties: the client, an honest-but-curious cloud service provider, and the data owner who transfers the encrypted graph to the cloud. Our work operates within the semi-honest model, which stipulates that the parties must adhere to the protocol exactly, ensuring they cannot alter their inputs or outputs. This means that each participant's private data should be both confidential after it has been submitted, and the cloud is responsible for returning the correct shortest path. Only the client, who possesses the private key, can decrypt the result of a query. This thesis specifically concentrates on the shortest path due to its extensive application in location-based services like Google Maps, Waze, and Apple Maps. Our research question is the following:

How can we design a protocol that solves the PSPEG problem while not leaking the volume pattern, not requiring the client to perform a shortest path algorithm, and not requiring assistance from trusted third parties?

1.6. Contributions

In this thesis, we propose two novel solutions to the PSPEG problem under two different settings. The first protocol encrypts the entire graph before outsourcing, whereas the second one assumes that the topology is public knowledge, and only encrypts the costs of traversing the edges. The proposed two protocols provide a trade-off between efficiency and security (see Chapters 5 and 6 for more details). More specifically, given a cloud provider who is in possession of a homomorphically encrypted weighted graph modelling a road network, the cloud computes the shortest path by applying a projected gradient descent algorithm for solving the underlying constrained optimization problem. The resulting constant-length path remains private to any other party except the issuing client. To the best of our knowledge, our protocol is the first to ensure confidentiality of the private values in the presence of an honest-but-curious cloud without relying on other trusted parties, as is the case of [18]. Moreover, our proposed protocol only requires the assistance of the client to perform comparisons, instead of performing Dijkstra's algorithm, as compared to [19]. Finally, we provide theoretical evaluations of the security and performance of our protocols as well as a practical analysis of their performance with a proof-of-concept implementation.

Of independent importance is the Linear Programming (LP) solver over homomorphic data presented in our protocol. We model the shortest path problem as an LP formulation, which can be solved in an oblivious manner by the cloud provider. In other words, the cloud performs the same sequence of operations regardless of the input data and independent of memory accesses. A similar idea can be found in [18] but our solution is more flexible, allowing any algorithm with an LP formulation to be solved in an oblivious manner. Overall, solving LP problems over homomorphically encrypted data represents a novel concept in the literature, and provides a foundation for further exploration in the field of secure outsourcing.

1.7. Outline

This thesis is organised as follows: in Chapter 2 we treat the underlying mathematical concepts of our protocol. In Chapter 3 we provide an overview of related work. In Chapter 4 we present our protocol. In Chapter 5 we prove the security of our protocol, and evaluate its complexity. In Chapter 6 we explain an extension of our protocol that does not protect the topology of the outsourced graph, but gains significant speedup. In Chapter 7 we empirically evaluate the convergence on various instances of graphs. Finally, in Chapter 8 we discuss our results and provide some closing remarks.

2

Preliminaries

This chapter covers the fundamental mathematical concepts that are necessary for understanding the upcoming chapters. Topics including graph theory, linear programming, optimization techniques and homomorphic encryption are briefly introduced.

2.1. Basic Notation

All the mathematical notations necessary to follow this chapter are presented in Table 2.1. All the log operations are performed in base 2 unless otherwise specified. The size of identity matrix $\mathbb{I}$ is generally skipped for brevity but can be inferred from the context.

Table 2.1: Notation adopted throughout this chapter.

Symbol	Definition
$\lvert\cdot\rvert$	cardinality of a set
$\lVert\cdot\rVert$	norm of a vector
$\mathbb{I}$	identity matrix
A^T	transpose of a matrix
A^{-1}	inverse of a matrix
∇f	gradient of function f
$\mathbb{C}$	the set of complex numbers
$\mathbb{R}_+$	the set of positive real numbers
$\mathbb{H} = \{z \in \mathbb{C}^N \mid z_j = \overline{z_{-j}}\}$	the set of complex numbers whose conjugates are equal
$\mathbb{Z}_q[X]/(X^N+1)$	polynomial ring of coefficient modulus q and ring dimension N
$p.q$	polynomial product
$[\cdot]_q$	implicit modulo q operation
$\lfloor\cdot\rfloor$	floor operation
$\lfloor\cdot\rceil$	round to nearest integer operations
$lcm(x,y)$	the least common multiple
$gcd(x,y)$	the greatest common divisor
$\xleftarrow{\mathcal{U}}$	random uniform distribution

2.2. Graph Theory

A graph is a data structure that consists of a set of objects known as vertices (V) and another set of objects known as edges (E). Each edge connects a pair of vertices, establishing a relationship between them. A graph is a common way to depict a road network, where intersections are the vertices and streets are the edges. Each street is assigned a weight, which can be based on factors such as distance, speed limit, or traffic conditions. We denote $|V| = n$ and $|E| = m$.

Similar to street design, the edges of a graph can be either directed or undirected. In an undirected graph, the edges have no direction and can be traversed in either direction. In a directed graph, each edge has a direction and can only be traversed in that direction. The edges can also be weighted or unweighted. A weighted edge has a numerical value associated with it, which represents the cost or distance of traversing the edge. We denote the weight of an edge e as $w(e)$. A graph is called a non-negative weighted graph if the weight of each edge in the graph is positive. Figure 2.1 illustrates a directed weighted graph with 4 vertices and 5 edges.

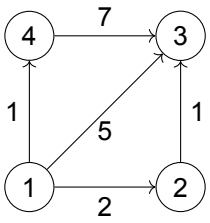


Figure 2.1: Instance of a directed weighted graph that consists of $|V| = 4$ and $|E| = 5$.

A path in graph theory is a sequence of vertices where each adjacent pair is connected by an edge. Essentially, it represents a way to travel from one node (or vertex) to another within a network. For instance, if we consider a road map where cities are represented by vertices and roads between cities by edges, a path could be a series of connected roads from one city to another. A shortest path, on the other hand, is the path with the minimal sum of weights between two vertices. For instance, in our road map example, the shortest path between two cities would be the one that minimizes the total travel distance or time. There are several algorithms used to solve shortest path problems. Dijkstra's algorithm [24], for example, is a well-known approach that uses a priority queue to iteratively select the vertex with the smallest tentative distance from a set of unvisited vertices, ultimately finding the shortest paths from a single source vertex to all other vertices in a weighted graph. Another example is the Bellman-Ford algorithm, which solves the single-source problem for a graph with negative edge weights, producing a shortest path tree. The Floyd-Warshall algorithm, on the other hand, finds shortest paths between all pairs of vertices in a graph, thus solving the all-pairs shortest path problem.

Node-arc incidence matrix

For a directed graph with n vertices and m edges, the incidence matrix is an $n \times m$ matrix A in which the rows correspond to verties and the columns correspond to edges. Each entry a_{ij} is

$$a_{ij} = \begin{cases} +1 & \text{if arc } e_j \text{ leaves vertex i} \\ -1 & \text{if arc } e_j \text{ enters vertex i} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

For example, the node-arc incidence matrix for the graph shown in Figure 2.1 is

$$A = \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} \begin{matrix} \begin{matrix} (1,4) & (4,3) & (1,3) & (1,2) & (2,3) \end{matrix} \\ \begin{bmatrix} +1 & 0 & +1 & +1 & 0 \\ 0 & 0 & 0 & -1 & +1 \\ 0 & -1 & -1 & 0 & -1 \\ -1 & +1 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

Theorem 1. *Given a node-incidence matrix A of a directed graph, the determinant of $A \times A^T$ is 0.*

Proof. The columns/rows of $A \times A^T$ are linearly dependent. This can be observed because the sum over all the columns/rows is 0. □

2.3. Linear Programming

Linear programming is a mathematical optimization technique that is used to find the best possible solution to a problem where a linear objective function is subject to linear constraints. The decision variables $x \in \mathbb{R}_+^n$ are defined. The objective function is a linear equation that expresses the objective of the problem: $c^\intercal x$, where $c \in \mathbb{R}^n$. The objective can be to maximize or minimize the objective function. The constraints of the problem are expressed as linear equations or inequalities.

The normal form of a LP problem is a mathematical representation where the problem is described by a set of linear equalities, collectively forming a polyhedron. Every linear programming problem can be transformed into standard normal form, which only includes equality constraints expressed as $Ax = b$, where$A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^{1 \times m}$. Equation 2.2 highlights the normal standard form of an LP problem. Once the objective function and constraints are defined, the problem can be solved using various LP solvers such as Gurobi or CPLEX. The optimal solution represents the best possible values of the decision variables that satisfy all the constraints and minimize the objective function. The optimal solution is usually denoted as x^*.

$$\begin{array}{ll} \text{min} & c^\intercal x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \tag{2.2}$$

2.4. Shortest Path Linear Programming Formulation

This section provides an LP model for the shortest-path problem. The model is general in the sense that it can be used to find the shortest path between any two vertices in the network. In this regard, it is equivalent to Dijkstra's algorithm. Given a directed graph G with $|V| = n$ and $|E| = m$ with non-negative weights on each edge, and a specified source vertex s and terminal vertex t, the objective is to find a path from s to t of minimum weight.

Decision variables: $x \in \mathbb{R}_+^m$

Objective:

$$\text{min} \sum_{j=1}^{m} c_j * x_j \tag{2.3}$$

Constraints:

$$\sum_{\substack{j=1 \wedge \\ e_j \text{ leaves } s}}^{m} x_j \; - \sum_{\substack{j=1 \wedge \\ e_j \text{ enters } s}}^{m} x_j = 1 \tag{2.4}$$

$$\sum_{\substack{j=1 \wedge \\ e_j \text{ enters } s}}^{m} x_j - \sum_{\substack{j=1 \wedge \\ e_j \text{ leaves } s}}^{m} x_j = 1 \tag{2.5}$$

$$\sum_{\substack{j=1 \wedge \\ e_j \text{ leaves } i}}^{m} x_j - \sum_{\substack{j=1 \wedge \\ e_j \text{ enters } i}}^{m} x_j = 0 \; \forall i \in V \setminus \{s, t\} \tag{2.6}$$

A careful reader may notice that the constraint matrix A is the node-arc incidence matrix of the graph G, while b is a vector of 0s that contains $+1$ and -1 in the positions of the source and terminal vertex, respectively. As an example, the LP formulation of the shortest path between vertices 1 and 3 from Figure 2.1 is shown in Figure 2.2. Solving this problem yields the optimum value $x^* = [0, 0, 0, 1, 1]$, with the objective value 3. The reader is encouraged to check the correctness of this result.

$$\begin{aligned} \text{minimize} \quad & 1x_1 + 7x_2 + 5x_3 + 2x_4 + 1x_5 \\ \text{subject to} \quad & \\ & \begin{aligned} 1x_1 + 0x_2 + 1x_3 + 1x_4 + 0x_5 &= 1 \\ 0x_1 + 0x_2 + 0x_3 - 1x_4 + 1x_5 &= 0 \\ 0x_1 - 1x_2 - 1x_3 + 0x_4 - 1x_5 &= -1 \\ -x_1 + 1x_2 + 0x_3 + 0x_4 + 0x_5 &= 0 \end{aligned} \\ & \\ & x_i \geq 0 \;\; \forall i \in 0 \dots 5 \end{aligned} \tag{2.7}$$

Figure 2.2: LP formulation of the shortest path from vertex 1 to 3 in the graph from Figure 2.1.

2.5. Projected Gradient Descent and Acceleration

There are multiple optimization algorithms that can be applied to solve an LP problem in standard normal form to derive the optimal solution. This includes the Simplex method [25]. There are also several iterative algorithms for solving LP problems. With each iteration, the algorithm converges towards the optimum solution. Projected gradient descent (PGD), for example, is a variant of the gradient descent algorithm (GD) used in optimization problems where the solution is confined within a particular closed compact convex set. The algorithm works similarly to GD, where an iterative process is followed to find the minimum of a function by moving in the direction of steepest descent. However, in PGD, after each gradient descent step, the intermediate solution is projected back onto the constraint set. This projection operation Π_C ensures that the solution remains within the specified constraints C at every iteration. An overview of the PGD algorithm for non-negative constraints $(x \geq 0)$ is presented in Algorithm 1.

Algorithm 1 Projected Gradient Descent for Non-Negative Decision Variables (PGD).

Input:
$f(x)$: convex function
x_0: initial point
Π_C: projection operator onto the feasible set C
α_k: step size
Output: x_k: A solution for $\min_{x \geq 0} f(x)\ s.t.\ x \in C$

$k \leftarrow 0$
while *stopping criteria not met* **do**
    $x_{k+1} \leftarrow x_k - \alpha_k \nabla f(x_k)$ ▷ Gradient Descent Step
    $x_{k+1} \leftarrow \Pi_C(x_{k+1})$ ▷ Projection Step 1
    $x_{k+1} \leftarrow max\{0, x_{k+1}\}$ ▷ Projection Step 2
    $k \leftarrow k + 1$
end while
return x_k

The projection operation is needed to ensure that the solution at each iteration of the Projected Gradient Descent algorithm remains feasible, i.e., satisfies the constraints. For a given vector $x \in \mathbb{R}^n$, the projection onto a feasible set C is the nearest point in C to x, as measured by the Euclidean distance. Mathematically, the projection operator Π_C is defined as $\Pi_C(x) = \text{argmin}_{y \in C} \|y - x\|_2^2$. Projected gradient descent is only efficient if the feasible set represents a simple convex set. When C is a polytope defined by linear equalities, such as $C = \{x \mid Ax = b\}$ for some matrix A and vector b, the projection operator Π can be computed as shown by [26] in Algorithm 2. It is important to notice that P and Q are independent of x.

Algorithm 2 Projection Operator for LP, from [26].

Input:
1: A, b: LP constraints
2: x: point to be projected
Output: projection of x onto the feasable region $Ax = b$
3: $P = \mathbb{I} - A^T(AA^T)^{-1}A$
4: $Q = A^T(AA^T)^{-1}b$
5: **return** $Px + Q$

Theorem 2. *If f is a convex function, ∇f is Lipschitz continuous with constant $L > 0$, PGD with constant step size $\alpha \leq 1/L$ satisfies*

$$f(x_k) - f^* \leq \frac{\|x_0 - x^*\|_2^2}{2\alpha k}$$

where $f^* = f(x^*)$ is the optimal cost value, x^* is the optimal value, α is the constant stepsize, k is the number of iterations performed. Hence the convergence rate of PGD is $\mathcal{O}(1/k)$.

Projected gradient descent can be accelerated in order to achieve the optimal $\mathcal{O}(1/k^2)$ convergence rate using the method introduced by Yurii Nesterov [27]. The central idea behind Accelerated Projected Gradient Descent (APGD) is that when the gradient is calculated, it uses a ”look ahead” value rather than the current position in the parameter space. This gives APGD a form of momentum, helping it to make larger steps when the gradient is consistently in one direction, and to dampen oscillations when the gradient is frequently changing direction. Algorithm 3 shows the accelerated projected gradient descent method for $x \geq 0$. Note that setting β equal to 0 results in Projected Gradient Descent presented above.

Algorithm 3 Accelerated Projected Gradient Descent for Non-Negative Decision Variables (APGD).

Input:
1: $f(x)$: convex function
2: x_0: initial point
3: Π_C: projection operator onto the feasible set C
4: α_k: step size
5: β_k: momentum
Output: x_k: A solution for $\min_{x \geq 0} f(x)\ s.t.\ x \in C$
6: $x_{-1} \leftarrow x_0$
7: $k \leftarrow 0$
8: **while** *stopping criteria not met* **do**
9: $\quad y = x_k + \beta_k(x_k - x_{k-1})$ ▷ Extrapolation Step
10: $\quad x_{k+1} \leftarrow y - \alpha_k \nabla f(y)$ ▷ Gradient Descent Step
11: $\quad x_{k+1} \leftarrow \Pi_C(x_{k+1})$ ▷ Projection Step 1
12: $\quad x_{k+1} \leftarrow max\{0, x_{k+1}\}$ ▷ Projection Step 2
13: $\quad k \leftarrow k + 1$
14: **end while**
15: **return** x

Theorem 3. *If f is a convex function, ∇f is Lipschitz continuous with constant $L > 0$, APGD with $\alpha_0 \in (0, 1)$, $\alpha_k = \frac{1}{2}(\sqrt{\alpha_{k-1}^4 + 4\alpha_{k-1}^2} - \alpha_{k-1}^2)$, $\beta_k = \frac{\alpha_{k-1}(1-\alpha_{k-1})}{\alpha_{k-1}^2 + \alpha_k}$ satisfies*

$$f(x_k) - f^* \leq \frac{2\|x_0 - x^*\|_2^2}{\alpha(k+1)^2}$$

where $f^* = f(x^*)$ is the optimal cost value, x^* is the optimal value, α is the constant stepsize, k is the number of iterations performed. Hence the convergence rate of APGD is $\mathcal{O}(1/k^2)$.

The parameters α and β may be difficult to compute. A possible optimization is to use Paul Tseng's $\beta = \frac{k-1}{k+2}$, as indicated in [28].

2.6. Homomorphic Encryption

Encryption provides data protection during transmission, as it ensures that only authorized parties can access the data. However, once the encrypted data is received, it needs to be decrypted before it can be used for any operations. Homomorphic encryption is a form of encryption that allows computations to be performed on encrypted data without first decrypting it. Essentially, it allows data privacy and data utility to coexist, overcoming the traditional compromise between privacy and functionality. This is particularly useful for scenarios like cloud computing, where a cloud provider might need to process data without viewing it.

Looking at an example, consider the case of Emma, who operates a financial consultancy firm that leverages cloud technology for storing all of her client's financial records. The advantage of using the cloud solution is that Emma does not need to oversee the base infrastructure, as it is managed by a specialized cloud service provider. Emma's main challenge is to maintain the confidentiality of her clients' financial information. To ensure this, she resorts to encrypting the data while it's stored in the database. This encryption helps deter potential external threats from accessing the sensitive information directly and restricts cloud service employees from viewing the data. However, whenever there is a requirement to update a client's financial record, Emma faces a dilemma. She either has to transfer the encrypted record back to her trusted environment or decrypt the record in the cloud, amend it, then re-encrypt before storing it again. This process presents a risk as the data has to be decrypted at some stage before it can be updated. The use of homomorphic encryption, which allows for processing of data while it remains encrypted, can help Emma solve this issue. She can perform the necessary modifications on the encrypted client data, and send it to the cloud for updating the records. Throughout this process, the data never exists in a decrypted or plain text form. This way, Emma's financial consultancy firm can uphold the security of their clients' data while enjoying the conveniences offered by cloud technology.

Let Enc be an encryption function, Dec be a decryption function, let $+$ and $*$ denote addition and multiplication operations over the plaintext domain, and $\oplus$ and $\otimes$ the corresponding additive and multiplicative operators over the ciphertext domain.

For an additive homomorphism, we have:

$$Enc(x + y) = Enc(x) \oplus Enc(y) \tag{2.8}$$

and

$$Dec(Enc(x) \oplus Enc(y)) = x + y \tag{2.9}$$

For a multiplicative homomorphism, we have:

$$Enc(x * y) = Enc(x) \otimes Enc(y) \tag{2.10}$$

and

$$Dec(Enc(x) \otimes Enc(y)) = x * y \tag{2.11}$$

There are multiple types of homomorphic encryption. Partial Homomorphic Encryption (PHE) allows the execution of only a single operation on the encrypted text: either addition or multiplication. For example, the RSA cryptosystem [29] is an example of PHE that supports multiplication. The ElGamal cryptosystem [30] is another example that supports multiplication. The Paillier cryptosystem, on the other hand, supports addition [31]. However, many cloud services require more functionality than just addition or multiplication. This is where Fully Homomrphic Encryption (FHE), also known as the "holy grail" of cryptography, can start to play an important role in securing cloud services. FHE allows arbitrary operations (though currently limited to addition and multiplication) over encrypted data. The concept of FHE was first introduced by Rivest, Adleman, and Dertouzos in 1978 [29], but it wasn't until 2009 when Craig Gentry proposed the first practical (though inefficient) construction of a fully homomorphic encryption scheme [12].

FHE schemes can be further categorized into the following: Bit-wise schemes, word-wise schemes and approximate schemes. Bit-wise schemes encrypt their input bit-wise meaning that each bit of the input is encrypted into a different ciphertext. From there, the operations are carried over each bit separately. The addition of bits corresponds to a XOR and the multiplication of bits to an AND. Examples of such schemes include FHEW [32] and TFHE [33].

Word-wise schemes represent inputs with larger integers modulo p for some $p > 2$, and compose through a series of additions and multiplications to form an arithmetic circuit. (Note that boolean arithmetic corresponds to $p = 2$). They allow to pack multiple values into one ciphertext and perform computations on these values in a single instruction. Although homomorphic operations in these schemes are less efficient than for bit-wise encryption schemes, their running time per packed slot can be lower compared to that of the binary schemes above. Schemes with these features include BGV [34] and BFV [35].

Approximate schemes allow one to perform homomorphic operations over approximated floating point values. The main idea is to add noise to encrypted values for security but considered to be a part of error occurring during computations. This noise is reduced along with plaintext by rescaling. As a result, our decryption result is an approximate value of plaintext with a predetermined precision. CKKS [36] scheme is similar to the second category in the sense that one can pack several numbers and compute on them in one instruction. The CKKS scheme does not have the algebraic constraints that lower the packing capacity of BGV and BFV. Hence, it is usually possible to pack more elements in CKKS ciphertext, thus resulting in the best-amortized cost. Unlike previous schemes, CKKS encodes complex, and thus real, numbers.

2.6.1. Paillier

Paillier cryptosystem [31] is a public-key encryption scheme that was introduced in 1999 by Pascal Paillier. The cryptosystem's additive homomorphism allows for the multiplication of two encrypted numbers to yield the encryption of their sum, or the multiplication of an encrypted number by an unencrypted scalar, resulting in the encryption of the product. The cryptosystem as shown below provide semantic security against chosen-plaintext attacks (IND-CPA). In this model, an attacker can choose a set of plaintexts and obtain their corresponding ciphertexts, and the encryption scheme is considered secure if the attacker cannot distinguish between the encryptions of any two chosen plaintexts. Paillier cryptosystem is based on the difficulty of computing discrete logarithms and the decisional composite residuosity assumption.

Key Generation

The encryption process begins with generating two large prime numbers p and q. The product of p and q, denoted as $n = p \times q$, is used as the modulus for the public and secret keys. The public key consists of the modulus n and a random integer g, which is a generator of the multiplicative group of integers modulo n^2. We then compute $\lambda = lcm(p-1, q-1)$ and $\mu = (L(g^\lambda \mod n^2))^{-1}$, where $L(x) = (x-1)/n$. The secret key consists of (λ, μ).

Encryption & Decryption

To encrypt a message m, a random integer r is first chosen such that $1 \leq r < n$ and $gcd(r, n) = 1$. The ciphertext c is then computed as $c = g^m * r^n \mod n^2$. To decrypt the ciphertext c, the secret key is used. The decryption process involves computing $m' = L(c^\lambda \mod n^2) * \mu \mod n$.

Homomorphic Operations

The homomorphic properties of Paillier cryptosystem enable computations to be performed on encrypted data. For example, given two ciphertexts $c_1 = g_1^m * r_1^n \mod n^2$ and $c_2 = g_2^m * r_2^n \mod n^2$, it is possible to compute the ciphertext of the sum of the plaintexts without revealing the underlying plaintexts. The ciphertext of the sum is computed as $c1 * c2 \mod n^2 = g^{(m_1+m_2)} * (r_1 r_2)^n \mod n^2$.

2.6.2. BFV

The Brakerski-Fan-Vercauteren (BFV) cryptosystem [35], is a public-key fully homomorphic encryption (FHE) scheme. The security of BFV relies on the difficulty of the Ring Learning With Error (RLWE) problem, which was first presented in "On Ideal Lattices and Learning with Errors Over Rings" [37]. BFV is IND-CPA secure assuming the hardness of RLWE problem under optimal parameters. Finding these optimal parameters is outside the scope of this paper, but the reader is encouraged to refer to the original paper for more details [35].

Parameters

Let q denote the modulus, d the degree of the polynomial ring R and let σ^2 denote the variance of the probability distribution $\mathcal{X}$. t is an integer with $1 < t < q$ and is the plaintext modulus coefficient.

Given a d-degree irreducible polynomial $f(x) \in Z[x]$ is a d-degree irreducible polynomial, usually taking $f(x) = x^d + 1$, where $d = 2^n$. Let $\Delta = \lfloor q/t \rfloor$ and denote with $rt(q) = q \mod t$.

Encoding & Decoding

In BFV, the plaintext space is the polynomial ring R_t. This means that messages need to be converted to polynomials before encryption. Given an integer message with a binary representation $m = a_{n-1} \cdots a_1 a_0$, its encoding is $M = a_{n-1}x^{n-1} + \cdots a_1 x + a_0$. Since n is large in practice, unused bits are set to 0.

Key Generation

The secret key SK is sampled from $\mathcal{X}$. Having generated the secret key, the public key PK is a pair of polynomials (PK_1, PK_2) as shown below:

$$PK_2 \leftarrow A \xleftarrow{\mathcal{U}} R_q \quad PK_1 \leftarrow [-A.SK + e]_q \tag{2.12}$$

Encryption & Decryption

After encoding, encryption represents a relatively straightforward step. The public key is used to encrypt message M. Three random polynomials are generated: $u, e_1, e_2 \in \mathcal{X}$, where $\mathcal{X}$ is the error distribution defined as a discrete Gaussian distribution. The ciphertext represents $C = (C_1, C_2) \in R_q^2$ as follows:

$$C_1 \leftarrow [PK_1 * u + e_1 + \Delta M]_q \quad C_2 \leftarrow [PK_2 * u + e_2]_q \tag{2.13}$$

To decrypt a ciphertext $C = (C_1, C_2)$, the party in possession of the private key performs:

$$M \leftarrow [\lfloor t * [C_1 + C_2.SK]_q / q \rceil]_t \tag{2.14}$$

Homomorphic Operations

Operations can be done homomorphically, such as addition and multiplication. The result is an increased noise level in the output ciphertext. Addition is an inexpensive process, both in terms of execution time and the amount of noise generated, increasing the noise's bit count by, at most, one [38]. Multiplication, on the other hand, involves taking two ciphertexts and outputting another ciphertext that encrypts the product of the initial plaintexts. The output ciphertext is produced by tensoring the inputs, which results in a three-element tuple. However, this can be simplified back to a two-element tuple through the use of key switching. BFV also has the capability to multiply an encrypted text by a plaintext, which yields a standard two-element tuple. Despite its versatility, multiplication is considerably more costly than addition, particularly when it comes to the growth of noise. When two ciphertexts are multiplied, the noise's bit count is increased by a parameter that is dependent on the ring.

2.6.3. CKKS

The Cheon-Kim-Kim-Song (CKKS) cryptosystem, as introduced in [36], is a public-key FHE scheme that is quoted as being the most efficient method to perform approximate HE computations over real and complex numbers [39]. CKKS also presents the best amortized cost mostly because it supports large packing capacity. Furthermore, CKKS integrates an efficient rescaling operator following each multiplication, unlike BGV and BFV which necessitate bootstrapping for such rescaling. However, the scheme does not come without its limitations. It employs approximate arithmetic instead of exact arithmetic, which means that the actual result may differ slightly from the expected result once the decryption is performed. CKKS can be transformed into its FHE equivalent with the help of bootstrapping. However, the bootstrapping algorithm of CKKS refreshes ciphertexts only partially and introduces additional loss of output precision. As such, CKKS bootstrapping is usually avoided in practice.

The security guarantees of CKKS are similar to BFV: CKKS is IND-CPA secure assuming the hardness of RLWE problem under optimal parameters. Finding these optimal parameters is outside the scope of this paper, but the reader is encouraged to refer to the original paper for more details [36]. Recent research [40] presents an efficient passive attack against CKKS that exploits linearity in the decryption function and the fact that approximate decryption results can provide clues about the RLWE errors. Despite this, as long as the decryption results are not shared with untrusted parties, then CKKS can be used without concerns about its security.

Table 2.2: Arithmetic operations noise, from [41].

Operation	Normalized error
$C \bigoplus C$	2ϕ
$C \bigoplus C \bigoplus C$	3ϕ
$C \bigotimes C$	$\phi^2 + 2\mu\phi$
$C \bigotimes C \bigotimes C$	$\phi^3 + 3\mu\phi^2 + 3\mu^2\phi$

Encoding & Decoding

CKKS encryption and decryption, and other operations over on polynomial rings. Therefore it is necessary to have a way to transform vectors of complex values into polynomials. Another contribution of CKKS is to propose a method that maps a vector of complex numbers into a single plaintext object and vice versa. Given an input of $N/2$ complex numbers $z \in \mathbb{C}^{N/2}$, the CKKS encoder step returns a single plaintext in the form of a polynomial $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ by first expanding it to $\pi^{-1}(z) \in \mathbb{H}$ and multiplying it by Δ for precision. The encoding and decoding algorithms are highlighted in Equation 2.15. The map π is the complex canonical embedding which is a variant of the Fourier transform. Scaling is done by multiplying by Δ and removing the least significant fractional parts via rounding. If coefficient modulus is denoted by $q > 1$, then Z_q denotes the set of integers $(-q/2, q/2]$.

$$Encode(z, \Delta) = \lfloor \Delta.\pi^{-1}(z) \rfloor \quad Decode(a, \Delta) = \pi(\frac{1}{\Delta}a) \tag{2.15}$$

Key Generation

The key generation procedure is similar to the one found in BFV scheme. We sample the secret key SK an element from R_2^N i.e a polynomial of degree N with coefficients in $\{-1, 0, 1\}$. The public key is a pair of polynomials (PK_1, PK_2) according to Equation 2.16.

$$PK_2 \leftarrow A \xleftarrow{\mathcal{U}} R_q \quad PK_1 \leftarrow [-A.SK + e]_q \tag{2.16}$$

Encryption & Decryption

Once we have encoded our vector, encryption represents a relatively straightforward step. The public key is used to encrypt message M. Three random polynomials are generated: $u \in R_2$, e_1, $e_2 \in \mathcal{X}$, where $\mathcal{X}$ is the error distribution defined as a discrete Gaussian distribution. The ciphertext represents $C = (C_1, C_2) \in R_q^2$ as follows:

$$C_1 \leftarrow [PK_1 * u + e_1 + M]_q \quad C_2 \leftarrow [PK_2 * u + e_2 + M]_q \tag{2.17}$$

The only difference between encryption in CKKS from that in BFV is that we do not scale M by a scalar. Decryption is performed by evaluating the input ciphertext on the secret key to generate an approximate plaintext message, as shown in Equation 2.18.

$$\tilde{M} \leftarrow [C_1 + C_2.SK]_q \tag{2.18}$$

Homomorphic Operations

Operations can be done homomorphically, such as addition and multiplication. The result is an increased approximation error in the output ciphertext. Consider two ciphertexts $C = (C_1, C_2)$ and $C' = (C'_1, C'_2)$ encrypted under the same public key. We have C_{add} = $(C_1 + C'_1, C_2 + C'_2)$. When decrypting C_{add} using SK we get $C_{add,1} + C_{add,2}.SK = C_1 + C'_1 + (C_2 + C'_2).SK = C_1 + C_2.SK + C'_1 + C'_2.SK = M + M' + 2e \approx M + M'$ given that e is negligible. Multiplication of the two ciphertexts C_1 and C_2 results in higher noise level, $C_1 \bigotimes C_2 = \Delta^2\mu_1\mu_2 + \Delta(\mu_1 e_2 + \mu_2 e_1) + e_1 e_2$. This is almost what we want except that the product payload is scaled by Δ^2. To maintain the product at the same scale as the input, we can use an operation called rescaling in CKKS terminology, to scale down the product by Δ and reduce the magnitude of multiplication noise. Bertolace et al. [41] illustrate the noise growth of the addition and multiplication of 2 or 3 ciphertexts, with a normalized error, $\phi = e/\Delta$, as summarized in Table 2.2.

2.7. Quantization

In the previous section, we introduced the Paillier and BFV encryption systems which allow homomorphic operations over positive integers. These operations include the addition of two encrypted integers, multiplication of a ciphertext integer with a plaintext integer, and, exclusive to BFV, multiplication of two ciphertexts integers. In practice, many applications of such schemes require an extension of the underlying scheme beyond integers to handle floating-point numbers. This is predominantly seen in machine learning algorithms where almost all widely-used methods necessitate floating-point numbers. A similar necessity is observed in the field of mathematical optimization.

For this reason, we must employ an additional encoding and decoding step to support real numbers. The operations of addition and multiplication must be preserved under this encoding. Namely:

$$Decode(Encode(a) + Encode(b)) = a + b \tag{2.19}$$

and

$$Decode(Encode(a) * Encode(b)) = a * b \tag{2.20}$$

for any real numbers a, b.

The representation of signed integers takes advantage of the modular arithmetic characteristics intrinsic to the Paillier scheme. Positive integers are allowed within the range of $(0, max_{int})$, where max_{int} is $\lfloor n/3 \rfloor$. Negative integers are allowed within the range (max_{int}, n). We allocate the value span between max_{int} and $n - max_{int}$ for identifying overflows. This method of encoding adheres to the two properties mentioned in Equations 2.19 and 2.20.

The representation of floating point values in integer form can be challenging. The most straightforward and often the most effective method is to scale all fractional numbers to integers and alter computations to function with these adjusted integers. This process is referred to as fixed precision encoding. In this encoding method, each floating point number is multiplied by a large value b^{Δ}, where b denotes the base and Δ is referred to as precision. The result is rounded down to the nearest integer. The decoding process involves dividing by the same large value b^{Δ}.

For example, encoding 8.765 with precision $\Delta = 4$ over base $b = 2$ results in $\lfloor 8.765 * 2^4 \rfloor = 140$. Decoding 140 results in $140/2^4 = 8.75$, which resembles the original number.

There are two downsides of fixed precision arithmetic. First, it does not fulfil the property presented in Equation 2.20: after every multiplication, one needs to divide by the large value. Under partially homomorphic schemes such as Paillier, this division is impossible to perform without decryption. This constraint may be acceptable or circumventable for certain tasks, but for others, it is an insurmountable hurdle. After decryption and decoding the result needs to be scaled down by an appropriate amount. While efficient, this technique requires always keeping track of how each plaintext has been scaled.

The second disadvantage is that the precision Δ has to be large in order to encode a real number. In the example above, a precision of 4 is clearly not enough. And yet there is a limit on how large the precision can be made, as it may cause decoding to fail. If any of the encoded numbers wrap around the plaintext modulus n the result after decoding is likely to be incorrect (recall Section 2.6.1). The same applies to BFV if one of the coefficients of the underlying plaintext polynomials wraps around the plaintext modulus t (recall Section 2.6.2).

The addition of quantized numbers does not represent a challenge: $x * b^{\Delta} + y * b^{\Delta} = (x + y) * b^{\Delta}$. However, the multiplication of two quantized numbers doubles the precision $x * b^{\Delta} * y * b^{\Delta} = (x * y) * b^{2\Delta}$. This means that, after each multiplication between an encrypted number and an encoded one, one must perform multiplication with $b^{-\Delta}$ in order to reduce the precision to the original form.

3

Related Work

This chapter is split into three main sections. First, Section 3.1 considers different works on privacy-preserving navigation systems that aim to solve PSP and PSPEG problems. Secondly, Section 3.2 examines works for securely outsourcing the computation of linear and quadratic programming models, highlighting the lack of secure protocols for the former. Last but not least, Section 3.3 covers existing security initially proposed for Searchable Encryption, but that can be generalized for outsourcing of arbitrary computation, as is the case in the PSPEG problem.

3.1. Private Shortest Path

In recent years, a series of works have been proposed to solve the private shortest path problem and its extension, the private path problem over encrypted graphs. This chapter details a selection of these works. We investigate the relevant protocols, separated into five main branches: obsfurcation-based, transformation-based, secure multi-party computation, secure cryptographic protocols, and secure information retrieval.

Lee et. al (2009) [13] propose a framework for PSP problem. It involves obfuscating the path query by injecting fake source and destination vertices. Their work assumes the existence of a trusted obfuscator which sits in between the client and cloud. The obsfurcator modifies each request from the client: instead of containing one single source v_s and one terminal vertex v_t, the obfuscator appends multiple fake vertices, turning them into sets S and T. The obfuscator forwards them to the cloud, which calculates the shortest path from all vertices in S to all vertices in T. Upon receiving the result, the Obsfurcator filters the result, and forwards to the client only the shortest path from v_s to v_t. Their method is visually summarized in Figure 3.1.

The cardinality of the sets S and T adjusts the level of security for this scheme. Increasing the number of fake vertices provides greater certainty to the client that her location remains unknown. However, if the goal is to completely hide the user's location, the scheme's performance is compromised as a large number of fake vertices need to be added. To address this issue, the authors propose an optimization to minimize overhead by distributing processing costs across multiple path queries. Their method involves introducing fake vertices near v_s and v_t, but it still allows the cloud to obtain an approximate location as pointed out by Mouratidis et al. [17]. This means that while the cloud may not be able to pinpoint the client's exact street, it can accurately locate her neighbourhood. Therefore, this approach (as well as any other obfuscation-based methods) is not suitable for designing navigation systems because of its weak privacy guarantees.

Khoshgozaran et al. (2007) [14] propose another approach based on the idea of spatial transformations. Here, the data owner, who is different from the cloud provider, maps the data from the original Euclidean space into a transformed space using a keyed function. A querying client in possession of the secret key converts her location into the transformed space and forwards it to the cloud. The latter,

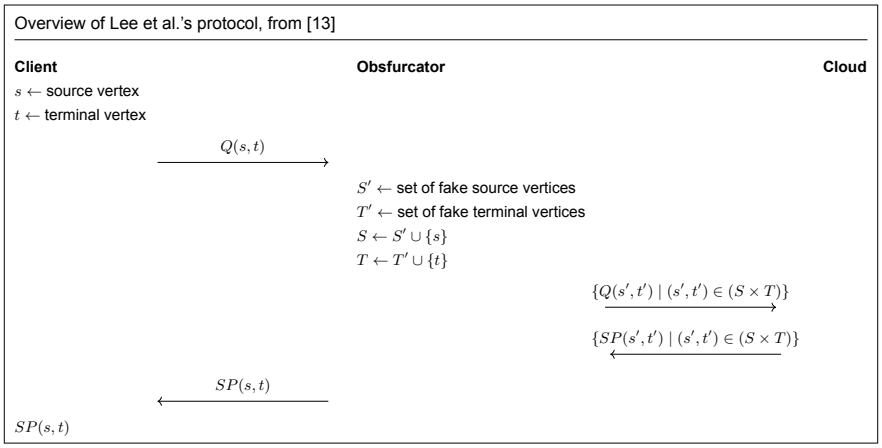


Figure 3.1: Lee et al.'s work [13] assumes the existence of a trusted obfuscator that modifies each query.

although unaware of the secret key and thus unable to map the data and query back to the original space, is still able to compute the query result. The client then uses her secret key to map back the received route in order to reveal the shortest path from v_s to v_t. [15] and [16] employ similar ideas.

As noted in [17], there is an inherent limitation in utilizing spatial transformations, which is the leakage of access pattern. This means that if the cloud has enough contextual information, it can establish a partial mapping between the two spaces. To exemplify, when numerous clients search for a specific destination and utilize deterministic encryption, and the cloud is aware of the existence of a large event, it can deduce that the clients are also attending the same event. Consequently, their suggested approach is inadequate for a privacy-conscious navigation system that would scale for a large number of clients. Transformation-based methods are an improvement over obfuscation-based methods, but they fall short of the desired level of privacy we aim for as long as they are deterministic. Their proposed scheme's practicality is further impeded by the presence of a trusted entity responsible for creating and updating encoded indexes and user identities.

Brickell et al. (2005) [22] consider more intricate designs that rely on multi-party computation. In their work, they consider the problem of jointly computing the shortest path in a scenario in which the data is vertically partitioned: two mutually distrustful parties are each in possession of a graph. The two parties jointly compute some algorithm on their combined graph but do not wish to reveal anything about their private graphs beyond that which is revealed by the output of the algorithm in question. They calculate the all-pair shortest paths in the combined graph and prove that no adversary can extract more information from the protocol transcript than what is revealed by the party's private input and the graph algorithm's result, under the assumption that both parties correctly follow the protocol. Another interesting construction comes from Faila [23]. Here, one party holds the costs (weights) associated with the edges of the graph, and the other knows a heuristic for finding the shortest path. Both parties compute the shortest path. Both schemes are proven to be secure against an honest-but-curious adversary.

Bramm et al. (2022) [20] propose a graph encryption scheme that enables exact shortest distance queries on outsourced encrypted graphs. They encode the graph as an adjacency matrix, before encrypting it with an order preserving encryption (OPE) scheme, which supports order preservation and homomorphic operation simultaneously, as described in [42]. They motivate their design choices of using OPE is used instead of FHE for efficiency reasons: they prioritize higher efficiency at the cost of more interactivity with the client. The security of their work is IND−O2CPA, as detailed in [42], which represents a weaker notion of indistinguishability compared to IND-CPA. While their protocol is secure in the (L_{Enc}, L_{Query})−CQA2 model, due to the order-preserving encryption, the leakage L_{Query} gives the volume pattern (number of edges between the source and the terminal node), a 2D order of all nodes along a path, and the edge pattern, which allows an adversary to link edges to specific nodes.

Samanthula et al. (2015) [19] propose two solutions to the PSPEG problem under two different settings. The first protocol is under a single-cloud setting whereas the second protocol utilizes a federated cloud environment. In both cases, the graph is encoded as adjacency lists, before being encrypted and sent to the cloud provider. In the two proposed protocols, the graph modelling the road network is split into $\alpha \times \alpha$ district regions in the form of a grid, with m being the maximum number of neighbours (single hop) of each vertex. The client iteratively retrieves specific sub-graphs, which allows her to compute the shortest path locally using Dijkstra's algorithm. This method has a significant computation overhead on the client side: at each iteration, the client runs Dijkstra's algorithm, hoping to find the shortest path from v_s to v_t. If that is not the case, the client runs another iteration until $SP(v_s, v_t)$ is found. Due to the per-iteration complexity of $\mathcal{O}(\alpha^2 + m)$, this method introduces a non-negligible overhead on the client side.

Blanton et al. (2013) [18] introduce a data oblivious algorithm to solve the single-pair shortest path problem, which assumes that the graph data is represented as an adjacency matrix. The computation time of the server in their method is $O(n^2)$, which is the optimal value for asymptotic complexity and works best for dense graphs. However, their method may not be efficient for sparse graphs, which is the case for most road networks. Nonetheless, their approach incorporates some remarkable features, such as the ability to identify situations where it is not possible to compute the shortest path. Furthermore, their design protects the shortest path's length by directing the algorithm to keep appending nodes to the path as long as the path is shorter than n vertices. Blanton's work guarantees the protection of users' data privacy when conducting secure computations in an outsourced environment to a cloud server, as long as memory accesses remain data-independent or oblivious. Blanton's approach utilizes Oblivious RAM techniques, which can make a non-oblivious algorithm oblivious, albeit with a polylogarithmic amortized cost per access that is proportional to the size of the data. Their solution requires multiple cloud service providers, at least three clouds for the linear secret sharing setting they propose.

Mouratidis et al. (2012) [17] stress the importance of protecting the clients' data access pattern: *'the result size itself may reveal information about whether the path is short or long'*. They solve the PSP problem with the help of hardware-aided PIR schemes, that utilize a tamper-resistant secure coprocessor installed at the server and is fully trusted by the clients. introducing a considerable overhead in terms of communication and computation on the server. As such, their work illustrates trade-offs between security and functionality: they ensure that all queries in a secure scheme perform the same number of page retrievals in order to prevent any type of leakage. It is worth mentioning that the outsourced graph is stored in plaintext by the cloud provider.

In their work, the client iteratively retrieves sub-graphs that are guaranteed to contain the shortest path between the desired source and terminal vertex. To achieve this, the space is partitioned into distinct regions. Firstly, the client receives two regions that contain the source and terminal vertices, respectively. Using this, the client iteratively and privately retrieves the sub-graph that is guaranteed to contain the shortest path. The client then applies Dijkstra's algorithms locally in order to find that path. Their solution suffers from the same limitation as [19]: in the worst case, all regions are retrieved, leaving us with a sub-optimal computation time on the client side.

3.2. Privacy-Preserving Convex Optimization

In this section, we explore different solutions for secure outsourcing of two major classes of convex optimization problems: linear and quadratic problems. While the shortest path can be formulated as a linear problem (see Section 2.4) instead of a quadratic problem, the literature on privacy-preserving quadratic optimization is more comprehensive. These solvers have been proposed as a response to the need for privacy in numerous applications of quadratic programming, including in control theory [43] [44] [45] or transportation systems [46]. For this reason, there are numerous privacy-preserving quadratic programming protocols, while the literature on privacy-preserving linear programming is abundant in protocols with either unclear [47] security guarentees, or even incorrect protocols [48].

Chen et al. (2014) [49] and Wang et al. (2015) [50] are representative works that propose privacy-preserving protocols for outsourcing linear programming problems. The idea is to 'encrypt' the original problem $\Phi(A, b, c)$ by applying an affine transformation. The resulting problem $\Phi_K(A', b', c')$ is then sent to the server. The server solves Φ_K using any LP solver and obtains the result y, together with the proof of optimality Γ. These are then forwarded to the client, which applies another linear operation in order to retrieve x^*, the solution to the original problem Φ. The client can also verify the optimality conditions using Γ. See Figure 3.2 for an overview of their construction. Xu et al. (2015) [45] and Salinas et al. (2016) [51] propose similar protocols for securely outsourcing of quadratic programs (QP).

Laud et al. [47] challenge some assumptions made by authors about transformation-based approaches. They have shown that several protocols can be broken by analyzing the structure of the encrypted problem. These schemes, which employ affine mappings, fail to modify the structure of the encrypted problem in a significant way. The authors argue that the lack of formal security definitions is a major contributor to the development and subsequent breakdown of such schemes. In Chen et al.'s paper [49], the statement *"protect (A,b,c) quite well"* without additional formal requirements is insufficient to ensure the scheme's security. On top of that, Bednarz et al. [48] found that the optimality condition does not hold for some constructions [52] [53].

$$\begin{aligned} \min \quad & c'^{\intercal} x \\ \text{s.t.} \quad & A'x = b' \\ & B'y \geq 0 \end{aligned} \tag{3.1}$$

$$\begin{cases} A' = QAM \\ B' = (B - \lambda QA)M \\ b' = Q(b + Ar) \\ c' = \gamma M^{\intercal} c \end{cases} \tag{3.2}$$

$$y = M^{-1}(x + r) \tag{3.3}$$

(a) Scheme proposed by Chen et al. [49]

$$\begin{aligned} \min \quad & c'^{\intercal} y \\ \text{s.t.} \quad & A'y = b \\ & y \geq 0 \end{aligned} \tag{3.4}$$

$$\begin{cases} A' = QAM \\ \\ b' = Qb \\ c' = \gamma M^{\intercal} c \end{cases} \tag{3.5}$$

$$y = Mx \tag{3.6}$$

(b) Scheme proposed by Wang et al. [50]

Figure 3.2: Overview of protocols for securely outsourcing LP problems, as presented in [49] and [50]. x is an $n \times 1$ vector, A is an $m \times n$ matrix, c is an $n \times 1$ column vector, and b is an $n \times 1$ column vector. Q is an $m \times m$ random non-singular matrix, M is an $n \times n$ random non-singular matrix, r is an $n \times 1$ random vector, $\gamma > 0$ is a positive scaling constant, λ is a random $n \times m$ matrix satisfying $\mid B - \lambda A \mid \neq 0$ and $\lambda b = 0$.

Shoukry et al. (2016) [54], Alexandru et al. (2017) [55], and Bertolace et al. (2022) [41] propose provably-secure methods for outsourcing the computation of quadratic programming problems. They utilize the power of homomorphic encryption, which, unlike transformation-based techniques, is based on computational hardness assumptions, making it computationally secure. A common thread among their solvers is the use of a modified projected gradient descent method to solve the quadratic programming problem. Projected gradient descent is suitable for homomorphic operations since it only contains operations that can be natively performed under homomorphism: addition and multiplication. Therefore, by encrypting the sensitive elements, the algorithm can be executed homomorphically, with only those in possession of the private key capable of decrypting the computation's outcome. To the best of our knowledge, there is currently no homomorphic encryption linear programming solver available.

3.3. Seachable Encryption Security Definitions

In recent years, privacy-preserving outsourcing methods have been presented in the literature to alleviate growing privacy concerns. Song et al. [56] first proposed the idea of Searchable Encryption (SE),

which enables searching on encrypted data stored in untrusted servers. Graph encryption is nothing but a generalization of Searchable Encryption, so a discussion about security definitions is important. Initially, the security of Searchable Encryption schemes was based on existing definitions taken from cryptography. For instance, Song et al. [56] uses indistinguishability against chosen plaintext attacks (IND-CPA). Yet these notions of security are not suitable for SE schemes, as the main leakage does not come from encrypting the ciphertexts, but rather from the user's queries and search tokens.

Goh (2003) [57] introduces the first security definition sepcifically tailored for SE schemes. He defines the notion of indistinguishability against adapting chosen keyword attacks (IND1-CKA). In this model, the adversary $\mathcal{A}$ cannot deduce anything about the document content from the search index. An IND1-CKA secure scheme generates indexes that appear to contain the same number of words *for equal-size documents*. This means that given two encrypted documents of equal size and an index, $\mathcal{A}$ cannot decide which document is encoded in the index.

Chang and Mitzenmacher (2005) [58] suggested that the IND1-CKA definition could be easily adapted for documents of different sizes, which is a more common scenario. To address this, they introduced the IND-CKA and noted that the document indexes must have the same number of keywords to meet this requirement. They also aimed to enhance the security of the trapdoors in their proposal. However, Curmola et al. (2006) identified an error in their formulation, which allowed insecure SSE schemes to meet the IND-CKA property.

Curtmola et al. (2006) [59] identified a flaw in the existing literature, which treated the security of indexes and the security of trapdoors as independent of each other. To address this, they introduced two new adversarial models for searchable encryption: IND-CKA1 and IND-CKA2. The former allows the attacker to submit two queries to an oracle: a keyword query and a random query. The oracle encrypts the index structure and returns the encrypted results of the requested query to the attacker, who must then guess which query was encrypted based on the received ciphertexts. A scheme that satisfies IND-CKA1 security ensures that an attacker cannot obtain any useful information about the encrypted index structure by observing the encrypted results of chosen keyword queries.

IND-CKA2 is a stronger security definition that extends IND-CKA1 by protecting against adaptive attacks, where the attacker can modify their queries based on the responses received from the oracle. In the IND-CKA2 game, the attacker can make a polynomial number of keyword queries and can adaptively choose their next query based on the results of the previous queries. A scheme that satisfies IND-CKA2 security ensures that an attacker cannot learn any useful information about the encrypted index structure, even when they can modify their queries adaptively based on the responses received from the oracle.

The first security definition for searchable encryption in the public-key setting was introduced by Boneh et al. (2007) [60]. This definition, called IND-CKA, requires that the adversary cannot obtain any information about the keywords without first obtaining the corresponding trapdoor. This security definition is similar to eavesdropper's security, where the adversary selects two keywords (w_0, w_1) of their choosing, which are encrypted by the challenger and returned to the adversary. By analyzing the keyword ciphertext, the adversary cannot determine whether it is the encryption of w_0 or w_1 unless the corresponding secret key is available.

Shen et al. (2009) [61] introduced the notion of a Fully Secure (FS) scheme, which hides both the data being searched and the search queries made on the encrypted data. This definition is stronger than IND-CKA2. Informally, a Fully Secure SE scheme allows nothing to be leaked, except for the access pattern. At present, FS is the most rigorous security definition that can be attained without affecting real-world practicality. This is because, in order to hide the access pattern, the result set needs to be the same length as the total number of documents stored by the third party. As such, this would imply an excessive overhead in terms of communication and space complexity.

4

A Privacy-Preserving Navigation System

In this chapter, we introduce a privacy-preserving navigation system that solves the PSPEG problem. We refer to it as PSPEG$_1$. Our protocol guarantees the privacy of the data owner's graph and the client's query. Our protocol resembles those discussed in Chapter 3, namely [18] and [19], where there are three distinct entities: the data owner, the client and the cloud. We first introduce the notation adopted throughout this work in Table 4.1, then we present the system model and assumptions, and then we proceed to describe the algorithms in more detail.

Table 4.1: Notation adopted throughout this chapter.

Symbol	Definition
$[[x]]$	encryption of value x
$[[X]]$	encryption of each individual value of matrix X
$inv(A)$	Moore–Penrose pseudo-inverse of the matrix A
$max(x, y)$	element-wise maximum between two lists of same size
$\mathbb{I}$	identity matrix

4.1. System Model

The system consists of three entities: the data owner, a client referred to as Alice, and a cloud provider. There are no constraints on how these three are formed, and a single entity can be involved by taking on one or more of the above roles. Alice first executes the $Setup$ algorithm and announces her public key. The data owner uses Alice's public key to homomorphically encrypt the necessary information required to solve the underlying optimization problem, before sending it to Alice and the cloud. The data owner maintains an updated view of the cost of traversing each edge, denoted by the cost vector c. This vector is continuously synced between the data owner and the cloud, also encrypted using Alice's public key. When Alice wants to obtain the shortest path between a starting vertex v_s and a terminal vertex v_t from A, she creates a search token by executing the $TokenGen$ algorithm. Following this, she sends the token to the cloud to query the shortest path between these two vertices in the graph. The cloud executes the $Search$ algorithm. At last, the cloud finds an encrypted shortest path between v_s and v_t and forwards it to Alice. When receiving Alice decrypts it and retrieves the query result. Section 4.4 provides more details about the protocol.

4.2. Threat Model

The adversary in our model is the cloud. We assume that it knows the identities of the data owner and of Alice. The adversary is honest-but-curious, i.e., it wishes to gain information about the Alice's queries, yet it executes all algorithms correctly, and would not falsify the data in any way. This assumption makes sense in a business-driven setting, in which a malicious hosting provider would be faced with negative publicity and a loss in consumer trust if its malicious behaviour were discovered. Furthermore, we assume that no collisions occur between the three involved parties.

The projection operators, P, Q, cost vector c and token b can be accessed in encrypted form by the cloud. The adversary is aware of the processing protocol being used and has computational power that is polynomially bounded, which is a typical assumption allowing the use of cryptographic primitives. Our work is done in the standard model, assuming that the RLWE problem is intractable. We also assume that a secure communication channel exists between all parties involved. We assume that the data owner and the cloud have enough storage capacity to store P and Q.

4.3. Design Goals

Our objective is to develop a practical protocol for solving the private shortest path, in a context where the cloud *does not deduce any information about the client's queries*. These requirements can be summarized as follows:

- **Graph protection**: The cloud should not be able to infer any kind of sensitive information (e.g. vertex identity, distance and cost values between two vertices) by analyzing the encrypted outsourced data.
- **Query protection**: The cloud should not be able to infer any kind of sensitive information (e.g. vertex identity, distance and cost values between two vertices) by analyzing the user's historical queries, or intermediate data.
- **Correctness**: The query result is the shortest path between Alice's starting and terminal vertices.

These requirements are similar to related works mentioned in Section 3, namely Samanthula et al. [19]. One observation is needed: a trivial solution to our problem involves sending c to Alice and allowing her to compute the shortest path locally using Dijkstra's algorithm [24]. Our assumption is that the cloud has reservations about transmitting its cost vector to Alice due to potential financial implications. However, our protocol does not prevent Alice from acquiring knowledge of the cost vector if she has access to a sufficient number of queries. It is incumbent upon the cloud to restrict the number of queries in order to prevent Alice from gaining knowledge of c.

4.4. Algorithms

Our protocol consists of three algorithms: $Setup$, $TokenGen$ and $Serach$. Alice can submit search tokens at arbitrary moments after the initialization procedure is complete. The queries can occur asynchronously and can be scheduled freely by the cloud, which calls the method $Search$ for each token it receives. We give an overview of the algorithms below and describe them, together with some design choices, in subsequent sections.

- $Setup$: Alice generates the FHE public-secret keypair, and announces the public key. The secret key is not shared with anyone. The data owner sends A in plaintext to Alice. The data owner sends the encrypted projection operators $[[P]], [[Q']]$ and cost vector $[[c]]$ to the cloud.
- $TokenGen$: Alice uses A in order to create a search token b. She then encrypts it using her public key before submitting it to the cloud.
- $Search$: Once in possession of the token, the cloud computes the projection operator $[[Q]]$ necessary to perform PGD over encrypted data in order to solve the underlying LP problem. It returns the encrypted optimal solution to Alice over a secure channel. Only Alice can decrypt the message and retrieve the shortest path.

4.4.1. Setup

The $Setup$ algorithm works as follows. Alice generates the FHE public-private keypair and publicly announces the public key pk. The secret key is not shared with anyone. The data owner generates A, a node-incidence matrix of the underlying graph G as described in 2.2, with the addition of a list of human-readable names containing the name of each node from G. The data owner sends A in plaintext to Alice. Furthermore, the data owner computes P and Q' as shown in Equation 4.1 and 4.2, respectively. The data owner encrypts each element of P and Q' and sends the encrypted $[[P]]$, $[[Q']]$ and $[[c]]$ to the cloud. Figure 4.1 illustrates a detailed communication diagram.

$$P = \mathbb{I} - A^T \times inv(A \times A^T) \times A \tag{4.1}$$

4.1: Initialization of projection operator component P

$$Q' = A^T \times inv(A \times A^T) \tag{4.2}$$

4.2: Initialization of partial projection operator component Q'

4.4.2. TokenGen

The TokenGen algorithm assumes the following scenario: Alice wants to get the shortest path query result between vertices v_s and v_t. First, she retrieves the indices of the respective vertices, v_{s_i} and v_{t_i}, using her topology A of the graph. She generates a vector $b \in \mathbb{R}^m$ containing $+1$ at index v_{s_i} and -1 at v_{t_i} and 0 everywhere else. Subsequently, Alice encrypts each element of the vector b using her public key pk and sends $[[b]]$ to the cloud. Figure 4.2 illustrates a detailed communication diagram.

4.4.3. Search

The Search algorithm works as follows. Upon receiving the client's token $[[b]]$, the cloud computes $[[Q]]$ using the multiplicative homomorphism of the FHE scheme: $[[Q]] = [[Q']] \bigotimes [[b]] = [[Q' \times b]]$. The cloud now has all the information in order to find the shortest path. It solves the LP problem using a modified projected gradient descent algorithm similar to Algorihtm from 1 or 3 from Chapter 2, with the notable exception that all the inputs and operations are converted into their homomorphic equivalent. Similar ideas have been implemented in the works of Shoukry et al. [54], Alexandru et al. [55], and Bertolace et al. [41], as noted in Section 3.2. Figure 4.3 illustrates a detailed communication diagram.

The algorithm begins with an initial guess x_0, which can be provided to the cloud by Alice or the cloud. Usual initialization values are 0.5. The algorithm then iteratively updates the solution by taking steps in the direction of the negative gradient of the objective function, just like in the standard Gradient Descent. However, in PGD, each update is followed by a projection step to ensure that the updated solution remains within the feasible region. We employ the projection operation $Px_k + Q$ as illustrated in Section 2.5.

The comparison with 0 ensures that the decision variables do not violate the non-negativity constraints, as illustrated in Section 2.4. There are two methods to perform secure comparisons over encrypted data. An interactive protocol, as presented in Alexandru et al.'s work [55], involves the cloud transmitting the encrypted decision variables vector $[[x_k]]$ to Alice, who then decrypts it locally and performs the comparison over plaintext. Alice then encrypts the results before she sends them back to the cloud, which resumes the algorithm. This method incurs significant overhead in terms of communication and client-side runtime computation: $\mathcal{O}(km)$ in total. Furthermore, this approach also requires $\mathcal{O}(km)$ encryptions and $\mathcal{O}(km)$ decryptions.

Non-interactive protocols also exist, but introduce a considerable computation overhead on the cloud provider. For instance, the secure comparison protocols proposed by Cheon [62] [63] use a polynomial approximation of the less-than function over positive real numbers. In particular [62] is optimal in the homomorphic setting, i.e. regarding the multiplicative depth and the number of multiplications. However, they require positive real numbers, which is not the case in our protocol. Chillotti et al. [64] show that one could calculate the maximum between two n-bit integers by evaluating a deterministic weighted automata made of $5n$ CMux gates. Using the TFHE scheme, evaluating a CMux gate takes

around 34 microseconds, meaning one can homomorphically compare two n-bit numbers in around $170n$ microseconds. These estimations correspond to the fastest (leveled) version of TFHE which avoids bootstrapping, or around $182n$ milliseconds in the case of the bootstrapped version. Since n is, in our case, 64 bits, this solution is not scalable. Iliashenko et al. [65] propose a comparison protocol suitable for arithmetic circuits (e.g. BGV or BFV) which has similar performance as FHE schemes for non-arithmetic circuits (TFHE) in basic comparison tasks such as less-than, maximum and minimum operations. Still, the authors claim to perform a secure comparison between two 20 bits encrypted numbers requires, on average, 8.66 seconds.

We decided to implement the interactive approach for three reasons: i) to reduce the complexity of the Search method for secure comparison protocols. The cloud then transmits the (non-) optimum solution $[[x_{k+1}]]$ to Alice. Alice decrypts each element of $[[x_{k+1}]]$ and performs the maximum operation locally. ii) interaction is required for the stopping criteria, where Alice can stop the algorithm when she detects convergence: $\|x_k - x_{k-1}\| \leq 10^{-5}$. iii) the extrapolation step of APGD requires two homomorphic operations: one for multiplying the decision variables with the quantized $k - 1/k + 2$ and another one in order to remove the double precision (see Section 2.7). Alice can perform this operation locally without any additional overhead in terms of time complexity.

When convergence is detected, Alice cannot expect the results of x_k to only contain 0s and 1s. As such, she needs to round each element to the closest integer. This is because of the approximate inverse in Equation 6.1 and 6.2. Values less than 0.5 will be rounded off to 0, which means they will not be included in the shortest path. On the other hand, values greater than 0.5 will be rounded to 1, signifying their inclusion in the shortest path. Finally, using the topology A, Alice can map the path to the graph and use it to navigate to her desired destination.

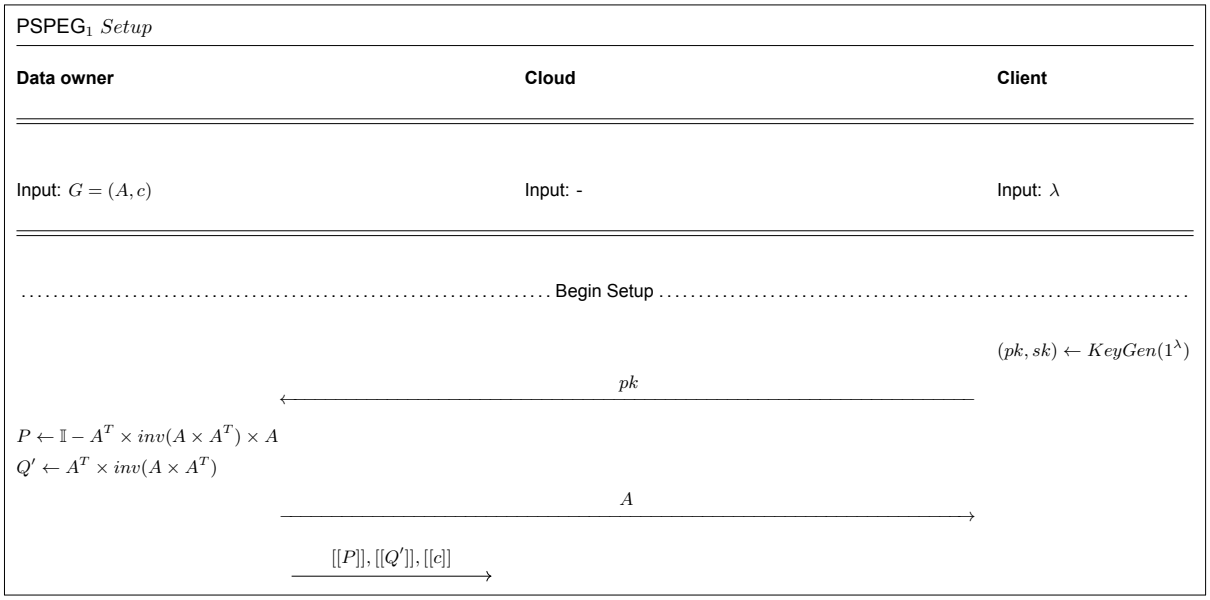


Figure 4.1: Communication diagram for PSPEG$_1$ Setup.

PSPEG$_1$ $TokenGen$

Cloud	**Client**
Input: -	Input: A, v_s, v_t
..... Begin TokenGen	
	$v_{s_i} \leftarrow getIndex(v_s, A)$ $v_{t_i} \leftarrow getIndex(v_t, A)$ $b \leftarrow createToken(v_{s_i}, v_{t_i})$
$[[b]]$ ⟵	

Figure 4.2: Communication diagram for PSPEG$_1$ TokenGen.

PSPEG$_1$ $Search$

Cloud	**Client**
Input: $x_0, [[Q']], [[P]], [[c]], [[b]], \alpha$	Input: $-$
..... Begin Search	
$[[Q]] \leftarrow [[Q']] \bigotimes [[b]]$ $k \leftarrow 0$ **while** $True$ **do**	
$[[x_{k+1}]] \leftarrow [[x_k]] \bigoplus -\alpha * [[c]]$	
$[[x_{k+1}]] \leftarrow [[P]] \bigotimes [[x_{k+1}]] \bigoplus [[Q]]$ ⟶ $[[x_{k+1}]]$	Decrypt $[[x_{k+1}]]$;

	$x_{k+1} \leftarrow max(0, x_{k+1})$
	if $\|x_{k+1} - x_k\| \leq 10^{-5}$ **do**
	  return x_{k+1}
	else
	  $\left[x_{k+1} \leftarrow x_{k+1} + \frac{k-1}{k+2}(x_{k+1} - x_k)\right]$

continue ⟵ $[[x_{k+1}]]$	

Figure 4.3: Communication diagram for PSPEG$_1$ Search.

5 Analysis

In this chapter, we analyse the security and complexity of PSPEG$_1$ introduced in Chapter 4. Recall that the three participating parties can take any arbitrary roles. We analyze two separate scenarios: the first being when the data owner and the cloud are the same entity, and the second being when they are not. This distinction holds significance because, in the first scenario, the cloud possesses knowledge of the graph and can execute queries. In the former case, we demonstrate the CQA2 security [59] of PSPEG$_1$ under the RLWE hardness assumption. In the latter, we argue that our protocol is data-oblivious [18]. Furthermore, we analyse the runtime and space complexity of the three participating entities. We compare the complexity and leakages with those from related works.

5.1. Security Proofs

We first prove that PSPEG$_1$ is secure under the CQA2-Security model, as defined in [59], assuming that the data owner and the cloud provider represent the same entity. Intuitively, this means that even if the attacker can make a polynomial number of queries and can adaptively choose their next query based on the results of the previous query, it cannot learn any useful information about the client's query.

Lemma 1. *The ring learning with errors problem in a polynomial ring $R_q = \mathbb{Z}_q[X]/f(x)$ of prime order q and a random polynomial $w \in R_q$, it is computationally hard to distinguish the uniform distribution over $R_q \times R_q$ from ordered pairs of the form $(a_i, a_i w + e_i)$, where a_i are uniformly distributed in R_q and e_i are polynomials in R whose coefficients are independently distributed Gaussians.*

Lemma 2. *Assuming that RLWE is hard in the random oracle model, BFV is IND-CPA secure.*

Lemma 3. *Assuming that RLWE is hard in the random oracle model, CKKS is IND-CPA secure.*

We establish two leakage functions $\mathfrak{L}_{enc}$ and $\mathfrak{L}_{query}$. The function $\mathfrak{L}_{enc}$ describes what is leaked to an adversary about the encrypted outsourced graph. This includes the number of vertices in topology A (denoted as n), and the number of edges in the graph (denoted as m). The $\mathfrak{L}_{query}$ leakage function tracks what is leaked as a result of issuing queries. In our proposed protocol, this is the total number of edges in the graph: m.

Adaptive Chosen Query Attack (CQA2) security, as introduced in Curtmola et al. [59], is defined using the real/ideal simulation methodology. The idea is to prove that a protocol is secure if the adversary $\mathcal{A}$ can learn approximately the same amount of information if it receives real queries (real setting) or receives random values (ideal setting), even if the attacker can make a polynomial number of queries and can adaptively choose their next query based on the results of the previous query. As such, we can formally prove that PSPEG$_1$ is secure against CQA2 by defining two games for an adversary $\mathcal{A}$ that interacts either with a real system, called the real game, or a simulator $\mathcal{S}$ that has access to leakage functions $\mathfrak{L}_{Enc}$ and $\mathfrak{L}_{Query}$. The interaction with the simulator is called the ideal game. We denote the real game for a protocol Π and adversary $\mathcal{A}$ as $Real_A^{\Pi}(\lambda)$, where λ is the security parameter. We similarly denote the ideal game as $Ideal_{A,S}^{\Pi}(\lambda)$. The real game runs algorithms in the real system and

the ideal game runs algorithms based on a simulator of the system. Our goal is to show that the adversary cannot distinguish from which games the search results are obtained. Given a security parameter λ, a challenger $\mathcal{C}$, a semi-honest adversary $\mathcal{A}$, and a simulator $\mathcal{S}$, we can conduct the following two probabilistic experiments:

$Real^{\Pi}_{\mathcal{A}}(\lambda)$:

1. Adversary $\mathcal{A}$ chooses a graph G.
2. $\mathcal{C}$ runs $Setup(1^{\lambda})$ and gives $\mathcal{A}$ the public key. $\mathcal{A}$ encrypts each value in c, P, Q' and returns $[[c]], [[P]], [[Q']]$.
3. Adversary $\mathcal{A}$ generates a polynomial number of queries $\mathcal{H}$ and passes it to $\mathcal{C}$.
4. Query Phase (repeated a polynomial, in the security parameter, number of times)
   a. Adversary $\mathcal{A}$ requests a token for query q.
   b. $\mathcal{C}$ returns to $\mathcal{A}$ the result of $TokenGen(q)$.
   c. Adversary $\mathcal{A}$ may now execute a search.
5. Adversary $\mathcal{A}$ returns, as output, a bit.

$Ideal^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$:

1. Adversary $\mathcal{A}$ chooses a graph G.
2. $\mathcal{C}$ runs $Setup(1^{\lambda})$ and gives $\mathcal{A}$ the public key. $\mathcal{A}$ encrypts each value in c, P, Q' and returns $[[c]], [[P]], [[Q']]$.
3. Adversary $\mathcal{A}$ adaptively generates a polynomial number of queries $\mathcal{H}$ and passes it to $\mathcal{C}$.
4. Query Phase (repeated a polynomial number of times)
   a. Adversary $\mathcal{A}$ requests a token for query q.
   b. $\mathcal{C}$ gives $\mathfrak{L}_{Query}(q)$ to $\mathcal{S}$. $\mathcal{S}$ simulates the corresponding query token forwards it to $\mathcal{A}$.
   c. Adversary $\mathcal{A}$ may now execute a search.
5. Adversary $\mathcal{A}$ returns, as output, a bit.

We say that Π is $(\mathfrak{L}_{Enc}, \mathfrak{L}_{Query})$-secure against adaptive attacks if for all probabilistic polynomial-time (PPT) adversaries A, there exists a simulator $\mathcal{S}$ such that:

$$|(Pr[Real^{\Pi}_{A}(\lambda) = 1] - Pr[Ideal^{\Pi}_{A,S}(\lambda) = 1]| < negl(\lambda) \tag{5.1}$$

Equivalently, a system is secure if there does not exist a PPT distinguisher that can distinguish the distribution generated by $Real^{\Pi}_{A}(\lambda)$ from $Ideal^{\Pi}_{A,S}(\lambda)$.

Theorem 4. *PSPEG$_1$'s $(Setup, TokenGen, Search)$ is $(\mathfrak{L}_{Query})$-Secure against the adaptive chosen-query attack (CQA2) assuming that the hardness of RLWE problem.*

Proof. The basis of this proof is to construct a simulator $\mathcal{S}$. Given the leakage function $\mathfrak{L}_{Query}$, $\mathcal{S}$ simulates dummy list of serach tokens $\mathcal{H}$. For all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, if they cannot distinguish between the two experiments $Real$ and $Ideal$, we say that our protocol is $(\mathfrak{L}_{Enc}, \mathfrak{L}_{Query})$-Secure against the adaptive chosen-query attack.

Simulating $\mathcal{H}$: Assume the query sequence made by adversary A is $\mathcal{H} = \{q_1, \ldots, q_n\}$. Given $\mathfrak{L}_{Query}$, $\mathcal{S}$ simulates the dummy query token q_i as follows: $\mathcal{S}$ selects a list of m (given by $\mathcal{L}_{Query}$) random elements in range $(0, q)$, where q represents the coefficient modulus, as described in Section 2.6.3 and 2.6.2. It then sends the list of random values to $\mathcal{C}$, which forwards it to the adversary $\mathcal{A}$. $\mathcal{S}$ also simulates the intermediate values $[[x_{k+1}]]$ by selecting random elements in range $(0, q)$. $\mathcal{H}$ is simulated in polynomial time because the query sequence is polynomial.

But, if RLWE is intractable, the encrypted dummy list of tokens $\mathcal{H}$ is computationally indistinguishable from the encrypted real one because the input and output are computationally indistinguishable from randomness (encryption is IND-CPA secure, as per Lemma 3 and 2). Therefore, the distinction between $Real$ and $Ideal$ experiments is not possible for all PPT adversaries A. Thus, we have

$$|(Pr[Real^{\Pi}_{\mathcal{A}}(\lambda) = 1] - Pr[Ideal^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| < negl(\lambda)$$

where $negl(\lambda)$ is a negligible function. This concludes our proof. □

If the data owner does not coincide with the cloud provider, it can be trivially shown that the attacker does not learn any information except for n and m. Figures 4.1, 4.2, 4.3 show that, with the exception of the public key, all communication comming and leaving the cloud is encrypted with an IND-CPA secure scheme (see Section 2.6). Furthermore, the cloud only performs homomorphic operations over encrypted data. As such, the cloud becomes oblivious: the cloud performs the same sequence of operations regardless of the input data and data-independent memory accesses, similar to Blanton et al. [18]. The formal proof is skipped for brevity.

Comparison with Related Protocols

We follow the same idea introduced by Blanton et al. [18], where the cloud performs the same sequence of instructions and access patterns for two different queries, thus becoming data-oblivious. Similar to our approach, their protocol protects the volume pattern by producing a path of length $|V|$. To achieve this, they instruct the algorithm to continue adding nodes to the path if the source s is reached, but the path is shorter than $|V|$ vertices long. As such, we can infer that our $\mathcal{L}_{Query}$ and $\mathcal{L}_{Enc}$ are equivalent.

Bramm et al. [20] attempt to solve the PSPEG problem using an order-preserving homomorphic encryption scheme. As such, $\mathcal{L}_{Query}$ includes the volume pattern, the order of all nodes along a path, as well as the edge pattern, which means that an attacker can identify which edges belong to specific nodes. Their $\mathcal{L}_{Enc}$ includes the total number of nodes and edges, just as in our construction, but also the order of all edges by length. This is the case in our construction, since we rely on a cryptographic scheme with stronger security requirements: IND-CPA compared to IND-O-CPA [42].

Samanthula et al. [19] do not offer a security analysis within their conference paper, and the comprehensive report has been removed from the official site. Nonetheless, we can deduce that their work does not leak the volume pattern, given that the cloud generates a single list of $3m + 1$ encrypted elements as long as the client cannot locally find the shortest path using Dijkstra's algorithm. The graph along with the client's queries are encrypted employing the Paillier cryptosystem, as delineated in Chapter 2, rendering them secure under the conditions of an adaptive chosen plaintext attack. Their proposal does not disclose the total count of nodes or edges from the outsourced graph, as they suggest incorporating dummy nodes when the count is not a multiple of $3m + 1$.

5.2. Complexity Analysis

The important factors which affect the efficiency of PSPEG$_1$ are the time and space complexity of algorithms $Setup$, $TokenGen$ and $Search$. Hence, we first present their concrete analysis, followed by analyzing the communication complexity between the client and the cloud provider. Please note that in this analysis we have considered the size of an encrypted value as constant. However, in practice, each complexity should be multiplied by the encryption key size (in bits) and by the degree of each polynomial in the packed ciphertexts (see Section 2.6).

In the $Setup$ algorithm, the client first announces the public key, which can be considered as a constant time operation. The client receives the topology from the data owner, which requires to $\mathcal{O}(mn)$ space. The cloud receives the two encrypted projection operators, requiring $\mathcal{O}(m^2)$ space and $\mathcal{O}(m^2n + mn^2 + n^3)$ time considering that that the Moore-Penrose pseudo-inverse takes $\mathcal{O}(n^3)$. Assuming that $m > n$ (there are more streets than intersections), the overall time complexity of $Setup$ is $\mathcal{O}(m^2n)$. These are sent to the cloud in encrypted form, requiring $\mathcal{O}(m^2)$ space.

In the $TokenGen$ algorithm, the client generates an n-dimensional search token. This operation takes $\mathcal{O}(n)$ time and space. The cloud, who stores the ecrypted token, requires $\mathcal{O}(n)$ space.

The $Search$ algorithm is comprised of a $\mathcal{O}(mn)$ operation for calculating $[[Q]]$, followed by k iterations, each requiring $\mathcal{O}(m^2)$ time for the cloud and $\mathcal{O}(m)$ for the client. In terms of storage, the demand is $\mathcal{O}(m^2)$ on the cloud and $\mathcal{O}(m)$ on the client.

Next, we discuss the communication complexity. In a $Setup$, the data owner sends the $\mathcal{O}(mn)$ topology to the client and $\mathcal{O}(m^2)$ projection operators to the cloud. In $TokenGen$, the client sends the cloud a list of n encrypted values. In $Search$, returns the encrypted result of $O(m)$ values. However, the protocol requires k rounds of communication. So the total communication complexity is $\mathcal{O}(km)$ for both parties.

Table 5.1: Time, space and communication complexity for the three parties.

	Setup			TokenGen			Search		
	Time	Space	Comm.	Time	Space	Comm.	Time	Space	Comm.
Data Owner	$\mathcal{O}(m^2n)$	$\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$	-	-	-	-	-	-
Cloud	-	$\mathcal{O}(m^2)$	-	-	$\mathcal{O}(n)$	-	$\mathcal{O}(km^2)$	$\mathcal{O}(m^2)$	$\mathcal{O}(km)$
Client	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(km)$	$\mathcal{O}(m)$	$\mathcal{O}(km)$

Comparison with Related Protocols

Our proposed protocol has the highest computational complexity on the cloud provider assuming that $m > n$. In Blanton et al. [18], the Search algorithm can be implemented in $\mathcal{O}(n^2)$ time. In Samanthula et al., the cloud performs $\mathcal{O}(\alpha^2 * m)$ modular exponentiations, where m represents maximum number of (1-hop) neighbours a vertex can have, and α represents the number of grids in which the graph G is split in.

As for the computation on the client side, in Mouratidis's [17] and Samanthula [19], the client needs to perform Dijkstra's algorithm locally after retrieving the subgraphs that may contain the shortest path. In the worst case, this is $\mathcal{O}((n+m)log(n))$ [24]. It remains to be seen through empirical evaluation how our $\mathcal{O}(km)$ compares to it.

6

Extension: Reduced Time at the Expense of Revealing Graph Topology

In this chapter, we put forward an extension of the protocol discussed in Chapter 4, called PSPEG$_2$. By revealing the topology A, the cloud provider can perform the optimization process solely through additive homomorphic operations, which is known to be orders of magnitude faster than fully homomorphic encryption. We analyse the security and complexity of this scheme.

6.1. Algorithms

The extension consists of the same three algorithms: $Setup$, $TokenGen$ and $Serach$. The system and adversarial model remain unchanged compared PSPEG$_1$, with the obvious difference that the graph protection requirement is dropped since the topology is sent in plaintext to the cloud provider.

6.1.1. Setup

The $Setup$ algorithm works as follows. Alice generates the Paillier public-private keypair and publicly announces the public key pk. The secret key is not shared with anyone. The data owner computes P and Q' as shown in Equation 6.1 and 6.2, respectively, before sending them to the cloud in plaintext. The data owner also sends $[[c]]$ to the cloud, and A to the client. Figure 6.1 illustrates a detailed communication diagram.

$$P = \mathbb{I} - A^T \times inv(A \times A^T) \times A \tag{6.1}$$

6.1: Initialization of projection operator component P

$$Q' = A^T \times inv(A \times A^T) \tag{6.2}$$

6.2: Initialization of partial projection operator component Q'

6.1.2. TokenGen

The TokenGen algorithm works as follows. Given that Alice wants to get the shortest path query result between vertices v_s and v_t, she retrieves the indices of the respective vertices, v_{s_i} and v_{t_i}, using her topology A of the graph. She generates a vector $b \in \mathbb{R}^m$ containing $+1$ at index v_{s_i} and -1 at v_{t_i} and 0 everywhere else. Alice encrypts b using her public key pk and sends $[[b]]$ to the cloud. Figure 6.2 illustrates a detailed communication diagram.

6.1.3. Search

The Search algorithm works as follows. Upon receiving the client's token $[[b]]$, the cloud provide first computes $[[Q]] = Q' \bigotimes [[b]]$. It solves the LP problem using a modified projected gradient descent algorithm similar to Algorihtm from 1 from Chapter 2, with the notable exception that all the inputs and operations are converted into their homomorphic equivalent. Similar to $SPSEG_1$, the algorithm is interactive: at each iteration, the decision variables are sent to the client, which performs the comparison, checks convergence criteria and performs the extrapolation step of APGD locally after decryption. Figure 6.3 illustrates a detailed communication diagram.

6.2. Complexity Analysis

The only difference from the protocol detailed in Chapter 4 is the transmission of P and Q' in plaintext. Nonetheless, as the size of the encryption was previously assumed to be a constant, this does not result in any asymptotic variations compared to the earlier protocol. In practice, each complexity should be multiplied by the encryption key size (in bits). See Section 2.6 for more details.

6.3. Security Proof

Lemma 4. *In the Decisional Composite Residuosity (DCR) problem, given composite integer n and an integer z, it is hard to decide whether z is an n-residue modulo n^2.*

Lemma 5. *Assuming that DCR is hard, Paillier is IND-CPA secure.*

Again, we establish two leakage functions $\mathfrak{L}_{enc}$ and $\mathfrak{L}_{query}$. The function $\mathfrak{L}_{enc}$, gives the number of vertices in topology A (noted as n), and the number of edges in the graph (noted as m). The $\mathfrak{L}_{query}$ gives the total number of edges in the graph. We define $\mathcal{H} = \{q_1, \ldots, q_m\}$ be a non-empty sequence of queries, and for each $q_i \in \mathcal{H}$, the corresponding tuple is noted as (s_i, t_i). For simplicity, we assume here that the data owner and the client represent the same entity. Two games are defined for an adversary $\mathcal{A}$ interacting with either a real system, called the real game, or a simulator $\mathcal{S}$ that has access to leakage functions $\mathfrak{L}_{Enc}$ and $\mathfrak{L}_{Query}$. Given a security parameter λ, an (semi-honest) adversary $\mathcal{A}$, and a simulator $\mathcal{S}$, we can conduct the following two probabilistic experiments:

$Real^{\Pi}_{\mathcal{A}}(\lambda)$:

1. Adversary $\mathcal{A}$ chooses a graph G.
2. $\mathcal{C}$ runs $Setup(1^\lambda)$ and gives $\mathcal{A}$ the public key. $\mathcal{A}$ encrypts each value in c and returns $[[c]], P, Q'$.
3. Adversary $\mathcal{A}$ generates a polynomial number of queries $\mathcal{H}$.
4. Query Phase (repeated a polynomial number of times)
    a. Adversary $\mathcal{A}$ requests the token for query q.
    b. $\mathcal{C}$ returns to $\mathcal{A}$ the result of $TokenGen(q)$.
    c. Adversary $\mathcal{A}$ may now execute a search.
5. Adversary $\mathcal{A}$ returns, as output, a bit.

$Ideal^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda)$:

1. Adversary $\mathcal{A}$ chooses a graph G.
2. $\mathcal{C}$ runs $Setup(1^\lambda)$ and gives $\mathcal{A}$ the public key. $\mathcal{A}$ encrypts each value in c and returns $[[c]], P, Q'$.
3. Adversary $\mathcal{A}$ adaptively generates a polynomial number of queries $\mathcal{H}$ and passes it to $\mathcal{C}$.
4. Query Phase (repeated a polynomial number of times)
    a. Adversary $\mathcal{A}$ requests a token for query q.
    b. $\mathcal{C}$ gives $\mathfrak{L}_{Query}(q)$ to $\mathcal{S}$. $\mathcal{S}$ simulates the corresponding query token forwards it to $\mathcal{A}$.
    c. Adversary $\mathcal{A}$ may now execute a search.
5. Adversary $\mathcal{A}$ returns, as output, a bit.

We say that Π is $(\mathfrak{L}_{Enc}, \mathfrak{L}_{Query})$-secure against adaptive attacks if for all probabilistic polynomial-time (PPT) adversaries A, there exists a simulator algorithm $\mathcal{S}$ such that:

$$|(Pr[Real^{\Pi}_{A}(\lambda) = 1] - Pr[Ideal^{\Pi}_{A,S}(\lambda) = 1]| < negl(\lambda) \tag{6.3}$$

Theorem 5. *Our proposed proposed protocol* $(Setup, TokenGen, Search)$ *is* $(\mathfrak{L}_{Enc}, \mathfrak{L}_{Query})$*-Secure against the adaptive chosen-query attack (CQA2) assuming that the hardness of DCR problem.*

Proof. The basis of this proof is to construct a simulator $\mathcal{S}$. Given the leakage function $\mathfrak{L}_{Query}$, $\mathcal{S}$ simulates a dummy list of serach tokens $\mathcal{H}$. For all Probabilistic Polynomial Time (PPT) adversaries $\mathcal{A}$, if they cannot distinguish between the two experiments $Real$ and $Ideal$, we say that our protocol is $(\mathfrak{L}_{Enc}, \mathfrak{L}_{Query})$-Secure against the adaptive chosen-query attack.

Simulating $\mathcal{H}$: Assume the query sequence made by adversary A is $\mathcal{H} = \{q_1, \ldots, q_n\}$. Given $\mathfrak{L}_{Query}$, $\mathcal{S}$ simulates the dummy query token q_i as follows: $\mathcal{S}$ selects a list of m (given by $\mathcal{L}_{Query}$) random elements in range $(0, q)$, where q represents the coefficient modulus, as described in Section 2.6.3 and 2.6.2. It then sends the list of random values to $\mathcal{C}$, which forwards it to the adversary $\mathcal{A}$. $\mathcal{S}$ also simulates the intermediate values $[[x_k]]$ by selecting random elements in range $(0, q)$. $\mathcal{H}$ is simulated in polynomial time because the query sequence is polynomial.

But, if DCR is intractable, the encrypted dummy list of tokens $\mathcal{H}$ is computationally indistinguishable from the encrypted real one because the input is computationally indistinguishable from randomness (encryption is IND-CPA secure, as per Lemma 5). Therefore, the distinction between $Real$ and $Ideal$ experiments is not possible for all PPT adversaries A. Thus, we have

$$|(Pr[Real^{\Pi}_{\mathcal{A}}(\lambda) = 1] - Pr[Ideal^{\Pi}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| < negl(\lambda)$$

where $negl(\lambda)$ is a negligible function. This concludes our proof. □

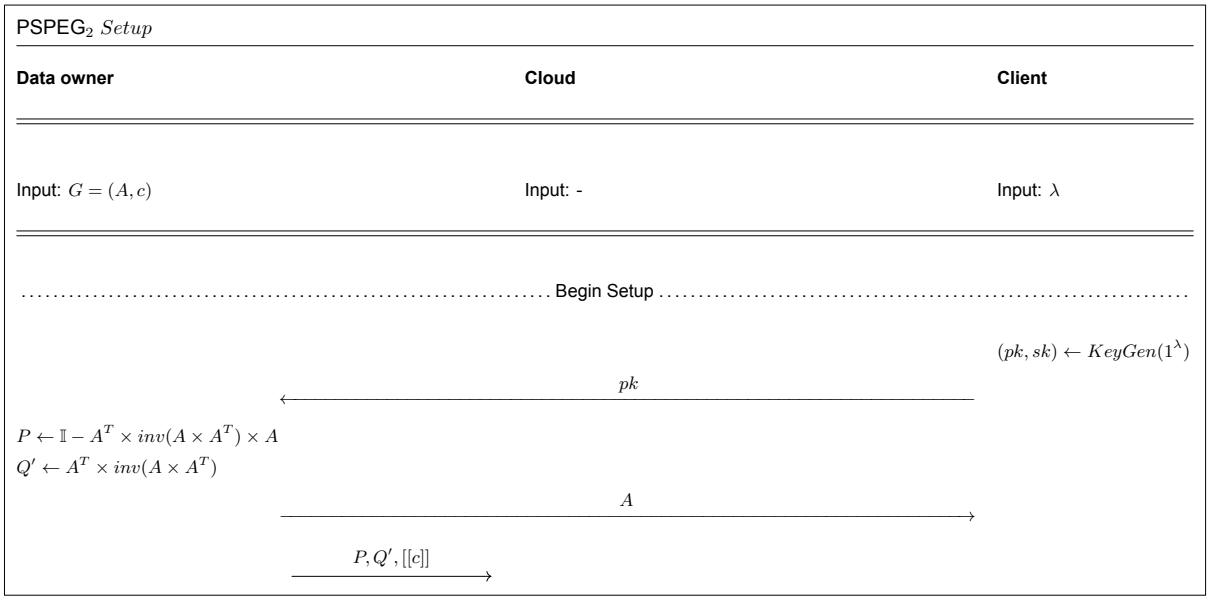


Figure 6.1: Communication diagram for PSPEG$_2$ Setup.

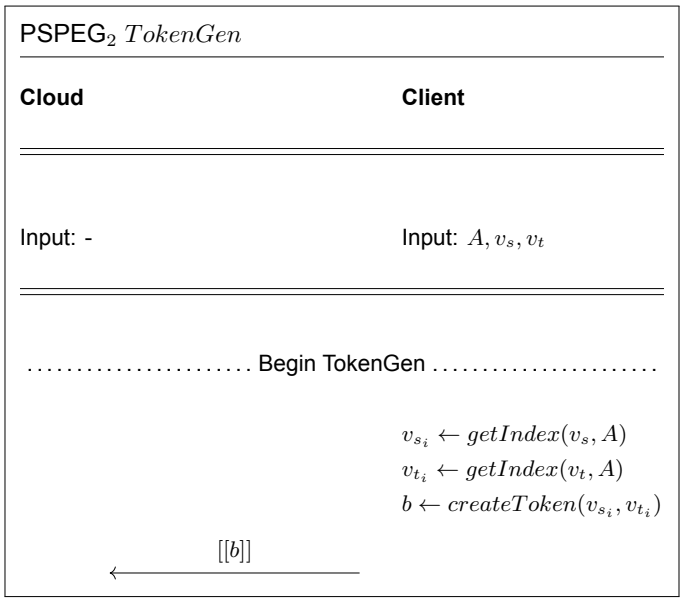


Figure 6.2: Communication diagram for PSPEG$_2$ TokenGen.

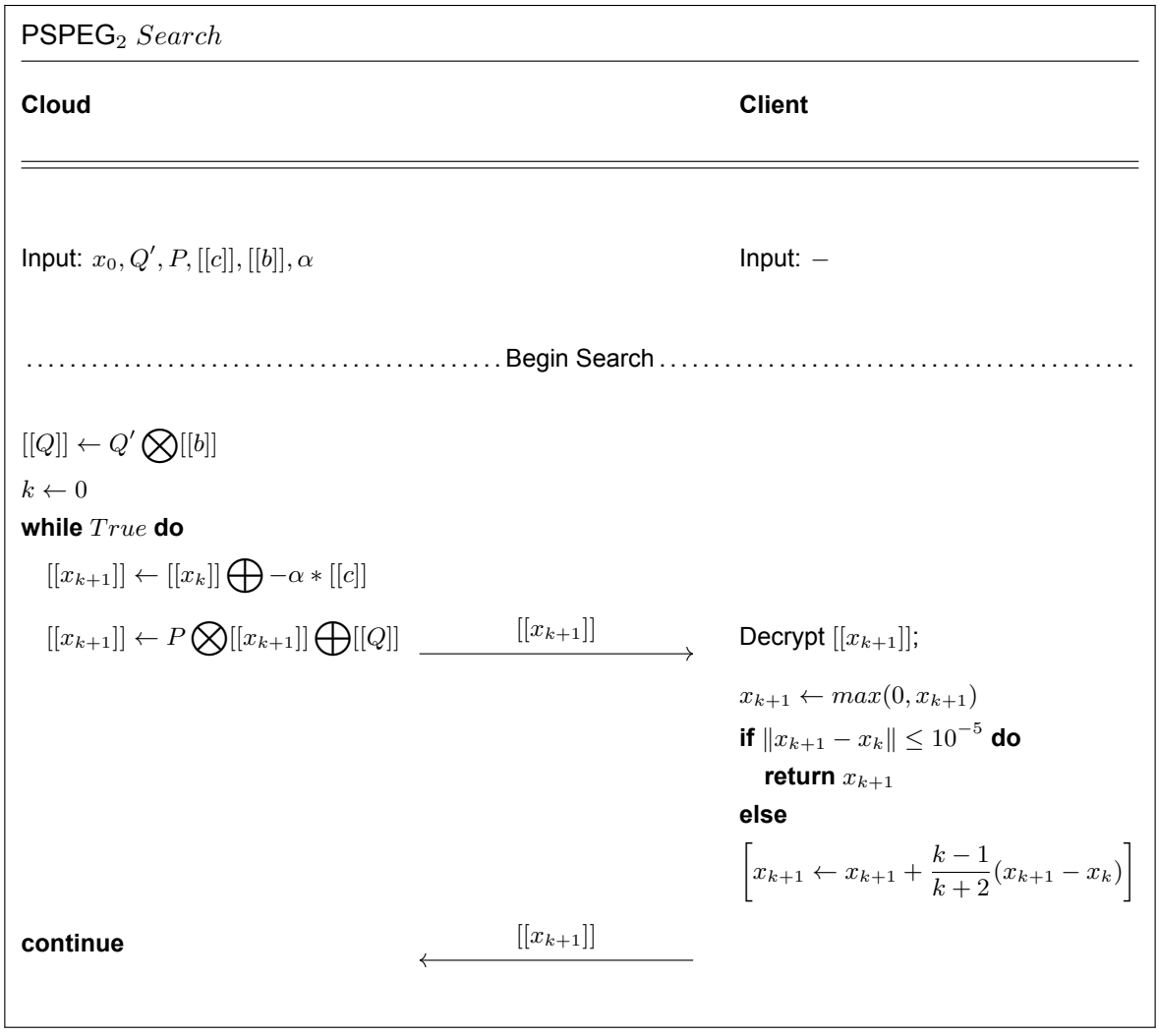


Figure 6.3: Communication diagram for PSPEG$_2$ Search.

7 Evaluation

In the previous sections, we determined through theoretical modelling the security and complexity of our proposed protocols. Various parameters heavily influence the convergence of the $Search$ algorithm. Our goal is to estimate the number of iterations required for convergence, and implicitly, the time it takes for our protocol to converge or to output the correct shortest path for various graph sizes. By doing so, we can assess the practicality of applying the protocol in real-world scenarios.

7.1. Experimental Setup

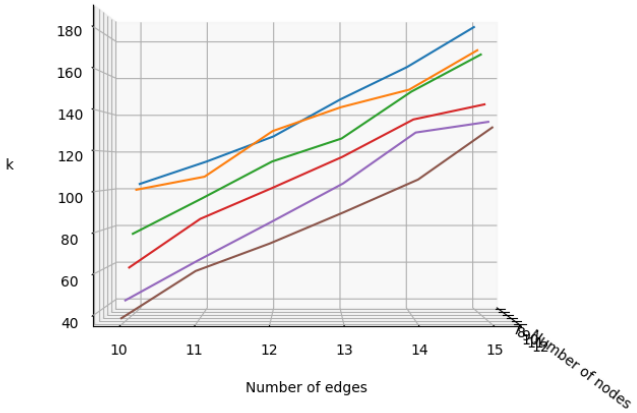
We first aim to determine the worst-case scenario, which allows us to predict the upper bounds of the algorithm's number of iterations. For this, we run PGD over plaintext, and compute the shortest path for a number of small instances. Figure 7.1 a) shows that the number of iterations (denoted by k) appears to increase linearly with respect to the number of edges. An interesting observation comes from Figure 7.1 b), which shows that the number of edges is inversely proportional with respect to the number of iterations. As such, our graph performs better on sparse graphs. The justification for this can be attributed to the condition number of the matrix A, which increases as the number of nodes increases, given a fixed number of edges, as shown in Table 7.1. As the condition number of A increases, the projection operator Q becomes an approximate solution of the system.

Table 7.1: PGD requires fewer iterations for larger condition numbers of the matrix A
Experiment performed with $\alpha = 0.001$.

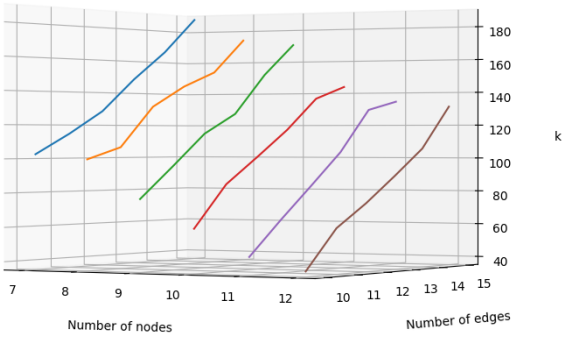
$\|V\| \times \|E\|$	5 x 20	10 x 20	15 x 20	20 x 20	50 x 20
k	429	274	142	120	49
$cond(A)$	4.1×10^{15}	4.7×10^{15}	1.3×10^{17}	2.7×10^{17}	∞

Therefore, the setup for the experiments is as follows. We generate random undirected dense graphs, which are converted to their directed counterpart by replacing each edge with a forward and a backwards edge. We then determine the longest path between any two given vertices in the graph. Unlike the polynomial-time solvable shortest path problem, finding the longest path in a weighted graph is NP-hard. To tackle this challenge, we compute the shortest path between every pair of vertices in the graph and choose the longest one. Considering the cost involved in conducting homomorphic operations [12], we are we conduct our experiments on small graph instances. We summarize in Table 7.2 the number of edges for the selected instances. Figure 7.2 a) shows an example of a generated dense graph with 5 nodes. Its longest path is highlighted in Figure 7.2 b). Each experiment, unless otherwise indicated, is repeated 50 times.

Reproducibility and integrity represent core principles of science. We ensure that all experiments are deterministic, meaning that the outputs obtained from two independent runs using the same input and hyper-parameters are identical. To achieve this, we initialize the Numpy randomness to a specific value of 420. All experiments are run on a computer equipped with i7-8750H CPU, with 32 GB of RAM.

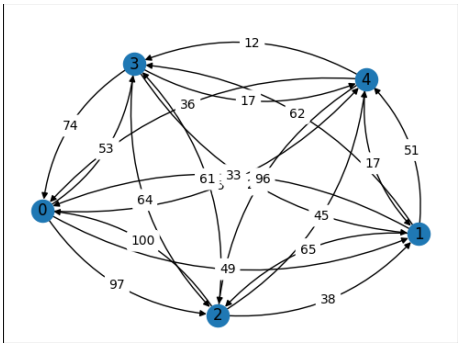


(a) Number of iterations for convergence is proportional to the number of edges.

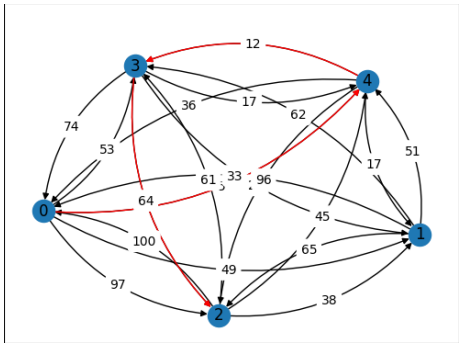


(b) Number of iterations for convergence is inversely proportional to the number of nodes.

Figure 7.1: Our protocol demonstrates worst performance on densely interconnected graphs.



(a) Generated dense graph with 5 nodes.



(b) Longest path is highlighted in red.

Figure 7.2: Generated dense graph and its corresponding longest path.

Table 7.2: Dataset consists of dense graphs consisting of n nodes m edges.

n	5	8	10	16	20
m	20	56	90	210	380

No use of the GPU for speeding up the FHE computations has been made. We encourage readers to test the implementation themselves, as it is openly available for use and can be found at the following Github repository [1].

7.2. Metrics

Three metrics of importance for our protocol are the number of iterations required for convergence, correctness and elapsed time. We say that a solution is correct either if by rounding it to the nearest integer, it is equal to an optimum solution or has an objective value at most 10^{-2} from the optimum value (in order to allow for equivalent shortest paths). Such an optimum solution is found by solving the underlying LP problem in plaintext using the SciPy optimize package. Finally, we measure the execution time, without taking into account any communication delays, in order to measure the trade-offs between usability and efficiency.

7.3. Optimization Algorithms

We are evaluating the two distinct optimization algorithms presented in Section 2.5. The first one is a PGD with non-negative constraints, detailed in Algorithm 1. The following one is the APGD algorithm, outlined in Algorithm 3, which incorporates Paul Tseng's $\beta = (k-1)/(k+2)$ that increases over time. We use this instead of the optimum parameters β because of the non-linear operations that are required to calculate it at each iteration (see Section 2.5 for more details).

7.4. Results

Convergence over Plaintext

The first experiment aims to determine the average number of iterations, denoted by k, required for the varying instances to converge. We sample a uniform distribution over the range of 0 to C, where $C = 100$ to assign weights to each edge. This number represents a common range for streets or highways, in kilometres. For any larger value, the data owner can scale it by lowering the step size α. Recall from Section 2.5 that the iterative algorithm converges if $\alpha \leq 1/L$, where $L = \|c\|_2$, the largest singular value of the cost vector c. As such, we vary the step size α from values between 0.001, 0.0001 and 0.00001. We initialize the vector x_0 with values of 0.5 for all its m elements, as discussed in Chapter 4. We run our algorithms instance until convergence is reached: $\|x_{k+1} - x_k\| < 10^{-5}$. Once convergence is reached, and we assert correctness according to the metric presented in Section 7.2.

Table 7.3: Average number of iterations required for convergence.
Experiments repeated 50 times with $C = 100$ and $\alpha = 0.001$.

$\|V\| \times \|E\|$	5 x 20		8 x 56		10 x 90		16 x 210		20 x 380	
algorithm	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)
PGD	313	80	541	10	581	4	875	0	1133	0
APGD	158	80	302	10	335	4	525	0	578	0

The number of iterations required to achieve convergence for PGD and APGD can be observed in Tables 7.3, 7.4 and 7.5. A few observations arise. First, APGD, with its optimal convergence rate of $\mathcal{O}(1/k^2)$ (refer to Section 2.5 for more details), requires fewer iterations to achieve convergence compared to PGD, regardless of the step size α. Figures 7.6 and 7.5 illustrates this by displaying the convergence rate of PGD compared to APGD for a 10x90 instance, using step size $\alpha = 0.00001$ and $\alpha = 0.0001$, respectively. Correctness is heavily influenced by the choice of step size α. The

[1] https://github.com/JayDew/master-thesis

Table 7.4: Average number of iterations required for convergence.
Experiments repeated 50 times with $C = 100$ and $\alpha = 0.0001$.

$\|V\| \times \|E\|$	5 x 20		8 x 56		10 x 90		16 x 210		20 x 380	
algorithm	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)
PGD	3717	100	5928	100	9741	100	13673	100	20197	84
APGD	322	100	574	100	777	100	1812	100	2126	84

Table 7.5: Average number of iterations required for convergence.
Experiments repeated 50 times with $C = 100$ and $\alpha = 0.00001$.

$\|V\| \times \|E\|$	5 x 20		8 x 56		10 x 90		16 x 210		20 x 380	
algorithm	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)	k (avg)	corr(%)
PGD	28292	100	38434	100	49875	100	80517	100	80874	100
APGD	706	100	1066	100	1500	100	2734	100	3297	100

results illustrate that smaller values of α increase the correctness rate for larger graph instances. This behaviour can be explained by analyzing small examples, where we see that the probabilities of some edges converge to 0.495, 0.498 and 0.49995 as we decrease the step size. The correctness score increases because the objective value is within the 10^{-2} threshold only for the latter.

The explanation in the previous paragraph contains an unexpected behaviour. The inclusion or exclusion of an edge should be marked by 0s and 1s, as opposed to 0.5. A further investigation reveals that in the case of two equal-length shortest paths, the algorithm selects both by assigning 0.5s to each of the edges included in the two paths. Similarly, in the case of three equal-length shortest paths, the algorithm assigns 0.333 for all edges that are part of the three paths. This is allowed as it does not violate the $Ax = b$ criteria presented in Section 2.2. The justification for this behaviour is that projected gradient descent, compared to other solvers such as Simplex [25], does not search the solution space by navigating the edges of the polyhedra, but ascends through the middle of it. As such, it includes all optimal solutions in the result set. While this byproduct of our algorithm can be regarded as a limitation at first sight, it enables an interesting behaviour: it returns all equivalent shortest paths to the client, which can select whichever she might prefer, based, for example, on road quality, points of interest or familiarity.

The results suggest that our protocol would benefit from a variable step size α: the cloud can start the iterative algorithm using larger values of the step size and gradually reduce it over time. There are different strategies for decay, such as linear decay, exponential decay, or step decay, where the learning rate is reduced after a fixed number of iterations or when a certain condition is met. As we have seen with the extrapolation step of APGD in Section 4, division requires two homomorphic operations, one for the multiplication of the quantized divisor and another for reducing the double scaling that results from the multiplication. Due to time constraints, we decided to leave out this optimization and perform the iterative algorithm with a fixed step size.

Adding encryption

Having analyzed the performance of the two distinct algorithms over plaintext, the focus shifts to introducing the encryption layer. We employ the Pyfhel implementation of the BFV and CKKS schemes [66]. For BFV, we use a low polynomial modulus degree of 14 because we do not pack ciphertexts. However, we mandate a large plaintext modulus bit size of 60 as each value is quantized by 2^{16} before encryption (see Section 2.7). For CKKS, we use the lowest polynomial modulus degree of 15, again, because we do not pack any ciphertexts. We use the same quantization of 2^{16}. For the Paillier implementation, we generate a 1024 bit key-size before encrypting. We use the same scaling factor of 2^{16} for the quantization. Due to time constraints, we run the experiments over 5 instances that have one single shortest path for the 5x20 graphs using APGD algorithm with $\alpha = 0.001$ step size.

The results are presented in Table 7.6. Two key observations can be drawn. Firstly, the time it takes to reach convergence increases substantially as we move from plaintext, to PHE schemes and to FHE schemes. This highlights the traditional security-utility trade-off in privacy. In the case of plaintext,

Table 7.6: Average number of iterations required for convergence. Experiments repeated 5 times with $\alpha = 0.001$ and $K_{max} = 200$ for 5x20 graphs.

$\|V\| \times \|E\|$	5 x 20			
Encryption	k (avg)	conv (%)	corr (%)	time (s)
None	91.8	100	100	0.006
$PSPEG_2$ Paillier	91.8	100	100	11.34
$PSPEG_1$ CKKS	200	0	100	437
$PSPEG_1$ BFV	91.8	100	100	2660

it takes an average of 0.0006 seconds to solve the 5 instances. However, once Paillier encryption is added, the time escalates to several seconds. The most drastic change is noted with the introduction of FHE, as the average convergence time under the BFV encryption scheme soars to the order of hours. The difference in elapsed time between BFV and CKKS can be attributed to the latter not requiring quantization, as the encoding ensures that floating point numbers are converted into polynomials (see Section 2.6.3 for more details). Secondly, CKKS does not converge within the maximum 200 iterations. The client is, however, able to retrieve the correct shortest path by rounding to the nearest integer.

To understand this behaviour, we have run BFV and CKKS on smaller 2x2 instance. Figure 7.3 highlights the relative error of BFV and CKKS as a function on the number of iterations. It it becomes clear that the approximations introduced by CKKS affect the convergence rate of the descent algorithm. The same instance shows that CKKS introduces large enough errors into x_{k+1} such that the stopping criteria $\|x_{k+1} - x_k\| < 10^{-5}$ is never fulfilled. This behaviour is visually depicted in Figure 7.4.

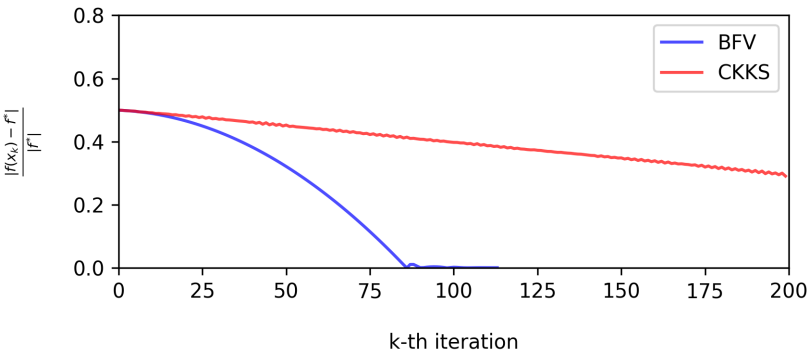


Figure 7.3: Convergence rate of APGD using BFV and CKKS for a small 2x2 instance using $\alpha = 0.001$.

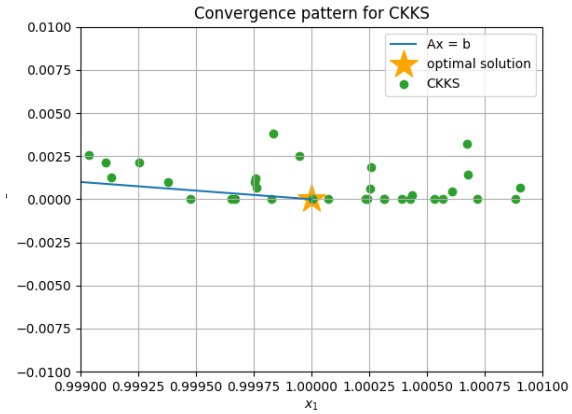


Figure 7.4: CKKS does not converge due to approximation errors.

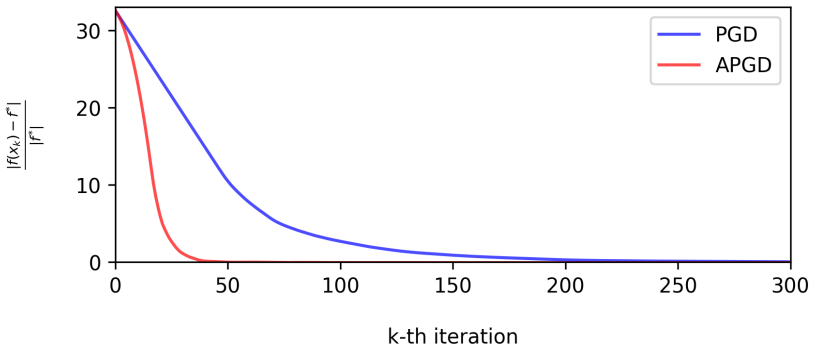


Figure 7.5: Convergence rate of PGD and APGD for a 10x90 instance using $\alpha = 0.0001$.

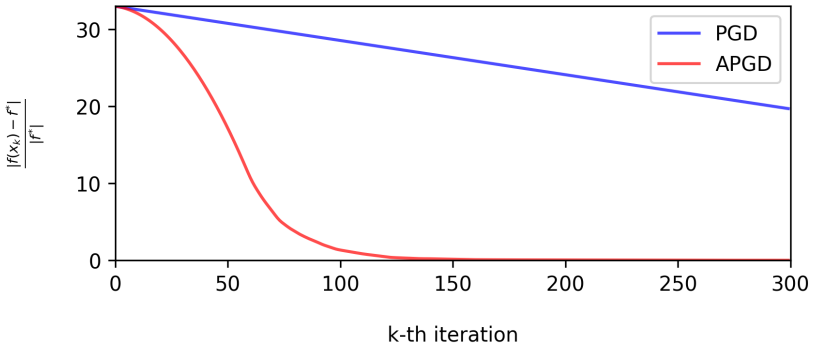


Figure 7.6: Convergence rate of PGD and APGD for a 10x90 instance using $\alpha = 0.00001$.

8

Discussion and Future Work

Privacy is a fundamental human right that is essential to individual autonomy, freedom, and dignity. It enables individuals to control their personal information and prevent abuses of power, exploitation, and discrimination. Privacy is also vital to the functioning of a democratic society and the protection of human rights. It is crucial that we continue to recognize and protect privacy as a core value. Design philosophies such as privacy by design aim to guarantee privacy by setting out clear guidelines for those working with privacy-sensitive data, requiring all involved personnel to consider privacy from the very start. Amongst others, the use of privacy-enhancing technologies (PETs) is a crucial component. Homomorphic Encryption (HE) protocols are one type of PETs that aim to provide strict privacy guarantees for computations that otherwise would be performed over plaintext data. Research into integrating HE into existing protocols is thus an important step in the process of maintaining the right to privacy.

Incorporating privacy-enhancing technologies (PETs) into existing applications can be a challenging but worthwhile endeavour. There are several steps that can be taken to successfully integrate PETs into an application. First, identify the privacy risks associated with the application and assess the potential impact of those risks on users. Then, select the appropriate PETs that can mitigate those risks, such as encryption, anonymization, or differential privacy. Always consider the trade-offs between privacy and usability. As we have seen in Chapter 3, some protocols use different techniques to solve the PSPEG problem: linear secret sharing, partially homomorphic encryption, or order-preserving homomorphic encryption. Some PSPEG protocols can perform the entire algorithm on the cloud, without requiring assistance from the client, but come at the cost of high leakage (volume pattern, order of edges etc.), as is the case with construction presented by Bramm et al [20]. Others require assistance from trusted third parties, as is the case in the linear secret-sharing method proposed by Blanton et al. [18]. Other protocols enable clients to iteratively retrieve subgraphs and perform Dijkstra's algorithm locally, as is the case of Samanthula et al. [19]. As long as there is a lack of suitable PSPEG protocols, applications will continue to use the numerous existing PSP protocols ([13], [14], [15], [16], [17]), which do not account for the privacy of the outsourced graph. We asked whether it is possible to create a PSPEG protocol that calculates the shortest path without the use of trusted third parties, without leaking the volume pattern and without the client performing any shortest path algorithm locally. In this chapter we discuss our results, consider some areas of future work, and conclude with an outlook on the future.

8.1. Discussion

In this thesis, we have seen for the first time the feasibility of homomorphically encrypted projected gradient descent algorithms for solving constrained (LP) optimization problems. The problem we consider is the following: a data owner models the graph as a node-incidence matrix before encrypting and outsourcing it to a cloud provider. An authorized client can perform shortest-path queries. The protocol employed in this study is interactive for three reasons: i) to stop the algorithm once convergence is reached, indicating that the cloud has identified a shortest path, ii) to mitigate the additional overhead caused by a secure comparison protocol, and iii) to remove two homomorphic multiplications that would otherwise be performed by the cloud provider for combining previous solutions in the extrapolation step of APGD, without changing the time complexity.

We have seen several important observations. Firstly, the PGD algorithm presented in Algorithm 1 is suitable for sparse graphs, as the large condition number of A allows the cloud to find a solution in one iteration (see Chapter 7 for more details). This is intriguing because it means that our work is complementary to that of Blanton et al. [18], which favours dense graphs. Secondly, the optimal solution retrieved by the algorithms includes all edges that are part of equivalent shortest paths. The client is faced with the responsibility of determining how to reconstruct the multiple returned routes. While this can be seen as a disadvantage, it enables the client to choose from a range of optimal paths. Thirdly, adding word-wise encryption schemes (BFV for PSPEG_1 and Paillier for PSPEG_2) may not influence the result of the optimization algorithm. However, approximate schemes (such as CKKS for PSPEG_1) result in non-convergence.

Compared to existing works on PSPEG problem, our proposed solution stands out in four regards. First, its security is equivalent to those of Samanthula et al. [19] and Blanton et al. [18], leaking no sensitive information about the outsourced graph and the client's query. This is, under the assumption that the number of nodes and edges is not considered sensitive. If that is not the case, the data owner would need to introduce dummy nodes and edges with infinite weights prior to graph encryption, in order to ensure statistical security. Compared to Bramm et al. [20], our protocols do not leak the volume pattern, nor the ordering of the edges. Secondly, our solution requires fewer assumptions compared to Blanton et al. [18], who proposes a linear secret sharing scheme as a possible instantiation of their protocol. But this requires a trusted third party for multiplications, which is not desirable for three reasons: i) trusted third parties typically act as centralized entities. If these entities are compromised, the entire system may be compromised. This risk extends to both technical failures (e.g., system outages, hacking) and human failures ii) trusted third parties often have access to a large amount of personal information, raising concerns about data privacy. The only way one can guarantee that the trusted third party does not act malicious is iii) if you offer high fees for their services. Thirdly, our protocols do not require the client to perform the shortest path algorithm locally, as is the case in Samanthula et al. [19], as the shortest path is obtained through the optimization process on the cloud. The reason this is important is that in the worst case, the client retrieves the entirety of the graph. Last but not least, our solution represents the only one that can return multiple shortest paths, giving the client the freedom to choose whichever is more convenient.

However, our protocol does not represent a one-size-fits-all solution to all forms of privacy-preserving navigation systems. First, our protocol imposes high computational requirements on the client and cloud: $O(km)$ time for the client and $O(km^2)$ for the cloud in the Search algorithm. Communication cost is $O(km)$ for both the client and the cloud. As illustrated in Chapter 5, these numbers are higher compared to both Samanthula et al. [19] and Blanton et al. [18], given the values for k depicted in Tables 7.3 - 7.5. Secondly, we motivated the need for time-varying momentum or learning rates based on the results presented in Tables 7.3 - 7.5, as it has the potential to reduce the required number of iterations for convergence. While our protocol supports linear decay, exponential decay is expensive as multiplication with an arbitrary constant requires two homomorphic multiplications. Thirdly, performing an update once graph G is outsourced is expensive. Updating the cost vector $[[c]]$ takes $O(m)$ time for the data owner. A road closure takes $O(m)$ time as well, as we need to set its respective weight to infinity. However, opening a new road requires re-initializing the entire system: $O(m^2n)$ for the data owner.

Based on these results, we conclude that our protocol does not sufficiently cater to the requirements of a general-purpose navigation system. Higher time complexity compared to existing works, together with the inherently slow HE scheme, results in impractical retrieval times for the shortest path, as demonstrated in Table 7.6. It lacks reliability: correctness may not be reached as we increase the problem instances unless a small step size is used, as seen in Table 7.3. Its novelty remains purely theoretical, despite the flexibility and the provable security it offers: our protocol does not leak the client's query volume pattern while requiring fewer assumptions compared to existing works and not performing any shortest path algorithm on the client side. Our protocol only excels under very particular circumstances, specifically when the graphs are sparse. In this case, the condition number of A ap-

proaches infinity and an optimal path is given by projection operator Q, which can trivially be obtained by the cloud provider, as shown in Table 7.1.

8.2. Future Work

Our protocol provides well-defined security guarantees but does not scale well because of its high computational complexity. As such, more use cases would benefit from an adjusted protocol. We envision three different ways in which our protocol could be extended or generalised.

Use Multi-Key FHE scheme

A direction of future work would be to outsource one single encrypted graph to the cloud provider and allow arbitrary clients to perform queries. This can be achieved using Multi-Key FHE paradigm, as described in [67]. The challenge in implementing this represents the incipient research in the field and the fact that the cloud computation scales linearly with respect to the number of users involved. As such, this method would benefit cloud storage at the cost of runtime.

Reduce the instance problem

Our proposed protocol does not scale well because of its quadratic complexity with respect to the number of edges in the outsourced graph. In order to reduce the instance of the problem, it is advisable to restrict the search scope to sub-graphs that are relevant to the user. The difficulty with implementing this extension lies in making a compromise on security (by acquiring information about the client's country, city or even neighbourhood) and usability.

Generalize for Other Algorithms

Another potential future direction involves exploring alternative graph algorithms that can be securely outsourced using the same protocol. Figure 8.1 displays an overview of our proposed protocol, where each of the 3 layers can be independently altered. As such, swapping the shortest path algorithm with another one that has an LP formulation (i.e. Maximum flow, Minimum Spanning Tree etc.) yields a completely new use case for our proposed solution.

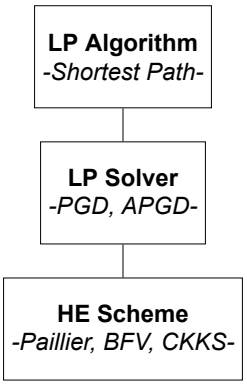


Figure 8.1: Our proposed protocol comprises 3 independent layers.

8.3. Concluding Remarks

Existing works on the PSPEG problem either ignore the volume pattern, rely on unreasonable assumptions or require the client to perform some expensive shortest-path algorithm locally. Our proposed protocol is based on Samanthula et al. [19]'s architecture that separates the data owner, client and cloud provider. The latter should not get any sensitive information regarding the underlying graph or the client's queries. Our two proposed protocols leverage homomorphic encryption to hide the structural information of graph data and guarantee that the cloud provider does not learn any information about the client's queries, including the length of the resulting paths, while allowing the cloud provider to solve the underlying linear programming (LP) problem using a projected gradient descent method. Security analysis shows that our protocol achieves CQA-2 security with less or equivalent leakage

compared with the existing schemes. The evaluation results indicate that the condition number of the node-incidence matrix of the graph being outsourced serves as a determining factor for the number of iterations. Dense graphs exhibit a low condition number, resulting in our protocol demanding an impractical number of iterations to calculate the shortest path. Conversely, sparse graphs allow for trivial resolution of shortest paths within a single iteration.

Our protocols hope to the attainability of deploying privacy-preserving navigation systems in the real world by addressing the need for extended security guarantees. In turn, this contributes to the maintenance of dignity, integrity and restoration of balance between individuals and powerful entities. But we are not quite there. Our protocols suffer from reliability and scalability issues. Attempts to identify faster or more robust ways of employing HE for arbitrary graph algorithms represent an interesting research direction.

Bibliography

[1] Laura Bednar. *App for Family Tracking Exposes Data of Thousands of Users*. Mar. 29, 2019. URL: https://www.securedata.com/blog/family-tracking-app-exposes-data (visited on 04/07/2023).

[2] Stephen Henderson. “Real-time and historic location surveillance after United States v. Jones: An administrable, mildly mosaic approach”. In: *Journal of Criminal Law and Criminology* (2013).

[3] Verto Analytics. *Most popular mapping apps in the United States as of April 2018*. Jan. 18, 2022. URL: https://www.statista.com/statistics/865419/most-popular-us-mapping-apps-ranked-by-reach/ (visited on 04/07/2023).

[4] European Parliament and Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council*. May 4, 2016. URL: https://data.europa.eu/eli/reg/2016/679/oj (visited on 07/04/2023).

[5] European Data Protection Board. *The CNIL’s restricted committee imposes a financial penalty of 50 Million euros against GOOGLE LLC*. Jan. 21, 2019. URL: https://edpb.europa.eu/news/national-news/2019/cnils-restricted-committee-imposes-financial-penalty-50-million-euros_en (visited on 04/07/2023).

[6] Steve Symanovich. *Uber Data Breach Affects 57 Million Rider and Driver Accounts*. Feb. 4, 2021. URL: https://lifelock.norton.com/learn/data-breaches/uber-data-breach-affects-57-million-rider-and-driver-accounts (visited on 04/07/2023).

[7] Edward Kost. *What Caused the Uber Data Breach in 2022?* Mar. 2, 2023. URL: https://www.upguard.com/blog/what-caused-the-uber-data-breach (visited on 04/07/2023).

[8] Stuart Pardau. “The California consumer privacy act: Towards a European-style privacy regime in the United States”. In: *Journal of Technology Law & Policy* 23 (2018), p. 68.

[9] Ronald Hes and John Borking. “Privacy-Enhancing Technologies: The Path to Anonymity”. In: *Information and Privacy Commissioner/Ontario* (1995).

[10] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. “Can homomorphic encryption be practical?” In: *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. 2011.

[11] Robert Graham. “How terrorists use encryption”. In: *CTC Sentinel* 9 (2016).

[12] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford University, 2009.

[13] Ken Lee, Wang-Chien Lee, Hong Va Leong, and Baihua Zheng. “Navigational path privacy protection: navigational path privacy protection”. In: *Proceedings of the 18th ACM conference on Information and knowledge management*. 2009.

[14] Ali Khoshgozaran and Cyrus Shahabi. “Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy”. In: *Advances in Spatial and Temporal Databases: 10th International Symposium, SSTD 2007, Boston, MA, USA, July 16-18, 2007. Proceedings 10*. 2007.

[15] Man Lung Yiu, Gabriel Ghinita, Christian S Jensen, and Panos Kalnis. “Enabling search services on outsourced private spatial data”. In: *The VLDB Journal* 19 (2010), pp. 363–384.

[16] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. “Secure KNN Computation on Encrypted Databases”. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2009.

[17] Kyriakos Mouratidis and Man Lung Yiu. “Shortest path computation with no information leakage”. In: *Proc. VLDB Endow.* (2012).

[18] Marina Blanton, Aaron Steele, and Mehrdad Alisagari. “Data-oblivious graph algorithms for secure computation and outsourcing”. In: *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 2013.

[19] Bharath Samanthula, Fang-Yu Rao, Elisa Bertino, and Xun Yi. “Privacy-Preserving Protocols for Shortest Path Discovery over Outsourced Encrypted Graph Data”. In: *2015 IEEE International Conference on Information Reuse and Integration*. 2015.

[20] Georg Bramm and Julian Schütte. “cipherPath: Efficient Traversals over Homomorphically Encrypted Paths.” In: *ICETE (2)*. 2020.

[21] Man Lung Yiu, Yimin Lin, and Kyriakos Mouratidis. “Efficient verification of shortest path search via authenticated hints”. In: *2010 IEEE 26th International Conference on Data Engineering*. 2010.

[22] Justin Brickell and Vitaly Shmatikov. “Privacy-preserving graph algorithms in the semi-honest model”. In: *Advances in Cryptology-ASIACRYPT 2005: 11th International Conference on the Theory and Application of Cryptology and Information Security*. 2005.

[23] Pierluigi Failla. “Heuristic search in encrypted graphs”. In: *2010 Fourth International Conference on Emerging Security Information, Systems and Technologies*. 2010.

[24] Edsger Dijkstra. “A note on two problems in connexion with graphs”. In: *Edsger Wybe Dijkstra: His Life, Work, and Legacy*. 2022.

[25] John Nelder and Roger Mead. “A Simplex Method for Function Minimization”. In: *The Computer Journal* (1965).

[26] Christian Boehm. “Short Course on Constrained Nonlinear Optimization II”. In: ETH Zurich.

[27] Yurii Evgen’evich Nesterov. “A method of solving a convex programming problem with convergence rate $O\left(\frac{1}{k^2}\right)$”. In: *Doklady Akademii Nauk*. Vol. 269. 3. 1983, pp. 543–547.

[28] Andersen Ang. *Nonnegative Least Squares - PGD, accelerated PGD and with restarts*. Apr. 8, 2017. URL: `https://angms.science/doc/NMF/nnls_pgd.pdf` (visited on 11/07/2023).

[29] Ronald Rivest, Adi Shamir, and Leonard Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* (1978).

[30] Taher ElGamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In: *IEEE transactions on information theory* (1985).

[31] Pascal Paillier. “Public-key cryptosystems based on composite degree residuosity classes”. In: *Advances in Cryptology—EUROCRYPT’99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. 1999.

[32] Léo Ducas and Daniele Micciancio. “FHEW: bootstrapping homomorphic encryption in less than a second”. In: *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34*. 2015.

[33] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. “TFHE: fast fully homomorphic encryption over the torus”. In: *Journal of Cryptology* (2020).

[34] Zvika Brakerski and Vinod Vaikuntanathan. “Efficient fully homomorphic encryption from (standard) LWE”. In: *SIAM Journal on computing* (2014).

[35] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. 2012.

[36] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. “Homomorphic encryption for arithmetic of approximate numbers”. In: *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. 2017.

[37] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. “On ideal lattices and learning with errors over rings”. In: *Journal of the ACM (JACM)* (2013).

[38] Robin Geelen and Frederik Vercauteren. “Bootstrapping for BGV and BFV Revisited”. In: *Journal of Cryptology* (2023).

[39] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. “Approximate Homomorphic Encryption with Reduced Approximation Error”. In: *Cryptology ePrint Archive, Paper 2020/1118* (2020). URL: https://eprint.iacr.org/2020/1118.

[40] Baiyu Li and Daniele Micciancio. “On the Security of Homomorphic Encryption on Approximate Numbers”. In: *Cryptology ePrint Archive, Paper 2020/1533* (2020). URL: https://eprint.iacr.org/2020/1533.

[41] André Bertolace, Konstantinos Gatsis, and Kostas Margellos. *Fully-homomorphically encrypted gradient descent algorithms for quadratic programming*. 2022.

[42] Yanguo Peng, Hui Li, Jiangtao Cui, Junwei Zhang, Jianfeng Ma, and Changgen Peng. “hOPE: improved order preserving encryption with the power to homomorphic operations of ciphertexts”. In: *Science China Information Sciences* (2017).

[43] Andre Teixeira, Kin Cheong Sou, Henrik Sandberg, and Karl Henrik Johansson. “Secure control systems: A quantitative risk management approach”. In: *IEEE Control Systems Magazine* (2015).

[44] Farhad Farokhi, Iman Shames, and Nathan Batterham. “Secure and private control using semi-homomorphic encryption”. In: *Control Engineering Practice* (2017). URL: https://www.sciencedirect.com/science/article/pii/S0967066117301557.

[45] Zhiheng Xu and Quanyan Zhu. “Secure and Resilient Control Design for Cloud Enabled Networked Control Systems”. In: 2015. DOI: 10.1145/2808705.2808708.

[46] Francois Belletti and Alexandre Bayen. “Privacy-preserving MaaS fleet management”. In: *Transportation research procedia* (2017).

[47] Peeter Laud and Alisa Pankova. “New attacks against transformation-based privacy-preserving linear programming”. In: *Security and Trust Management: 9th International Workshop, STM 2013, Egham, UK, September 12-13, 2013. Proceedings 9*. 2013.

[48] Alice Bednarz, Nigel Bean, and Matthew Roughan. “Hiccups on the road to privacy-preserving linear programming”. In: *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*. 2009.

[49] Fei Chen, Tao Xiang, and Yuanyuan Yang. “Privacy-preserving and verifiable protocols for scientific computation outsourcing to the cloud”. In: *Journal of Parallel and Distributed Computing* (2014).

[50] Yulong Wang and Yi Li. “An efficient and tunable matrix-disguising method toward privacy-preserving computation”. In: *Security and Communication Networks* (2015).

[51] Sergio Salinas, Changqing Luo, Weixian Liao, and Pan Li. “Efficient Secure Outsourcing of Large-scale Quadratic Programs”. In: 2016. DOI: 10.1145/2897845.2897862.

[52] Wenliang Du. “A study of several specific secure two-party computation problems”. PhD thesis. Purdue University, 2001.

[53] Jaideep Vaidya. “A secure revised simplex algorithm for privacy-preserving linear programming”. In: *2009 International Conference on Advanced Information Networking and Applications*. 2009.

[54] Yasser Shoukry, Konstantinos Gatsis, Amr Alanwar, George Pappas, Sanjit Seshia, Mani Srivastava, and Paulo Tabuada. “Privacy-aware quadratic optimization using partially homomorphic encryption”. In: Institute of Electrical and Electronics Engineers, 2016.

[55] Andreea Alexandru, Manfred Morari, and George Pappas. “Cloud-Based MPC with Encrypted Data”. In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018. DOI: 10.1109/CDC.2018.8619835.

[56] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. “Practical techniques for searches on encrypted data”. In: *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*. 2000.

[57] Eu-Jin Goh. *Secure Indexes*. Cryptology ePrint Archive, Paper 2003/216. 2003. URL: https://eprint.iacr.org/2003/216.

[58] Yan-Cheng Chang and Michael Mitzenmacher. “Privacy preserving keyword searches on remote encrypted data”. In: *ACNS*. Vol. 5. 3531. 2005.

[59] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. “Searchable symmetric encryption: improved definitions and efficient constructions”. In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006.

[60] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E Skeith. “Public key encryption that allows PIR queries”. In: *Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27*. 2007.

[61] Emily Shen, Elaine Shi, and Brent Waters. “Predicate privacy in encryption systems”. In: *Theory of Cryptography: 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings 6*. 2009.

[62] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. “Efficient homomorphic comparison methods with optimal complexity”. In: *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*. 2020.

[63] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. “Numerical method for comparison on homomorphically encrypted numbers”. In: *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part II*. 2019.

[64] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. *Improving TFHE: faster packed homomorphic operations and efficient circuit bootstrapping*. 2017.

[65] Ilia Iliashenko and Vincent Zucca. “Faster homomorphic comparison operations for BGV and BFV”. In: *Proceedings on Privacy Enhancing Technologies* (2021).

[66] Alberto Ibarrondo and Alexander Viand. “Pyfhel: Python for homomorphic encryption libraries”. In: *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 2021.

[67] Taechan Kim, Hyesun Kwak, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. “Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition”. In: *Cryptology ePrint Archive* (2022).