Preserving Secrecy in Online Social Networks: Data Outsourcing, Access Control, and Secrecy Schemes

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

Von

Gabriela Suntaxi

aus Quito

Tag der mündlichen Prüfung:	20. November 2020
Erster Gutachter:	Prof. Dr.-Ing Klemens Böhm
Zweiter Gutachter:	Prof. Dr. Jörn Müller-Quade

Acknowledgements

I want to express my gratitude to everyone who has helped through this journey. First, thanks to God, without his blessings, this achievement would not have been possible.

I also want to thank my supervisor Prof. Dr.-Ing. Klemens Böhm for giving me the opportunity to be part of his research group. His time, advices, helpful comments, and patience during these years have helped me to improve and finish this work. This opportunity has opened my eyes to the world of academia. I sincerely thank Prof. Dr. Jörn Müller-Quade, for being my second supervisor and for taking the time to discuss my work with his research group.

I would also like to thank Dr. Achraf El Ghazi for his continuous support and motivation. Thanks for the productive discussions, his motivation, and his patience. I would also extend my thanks to my colleagues. In particular, I would like to thank Saeed Taghizadeh. His kindness, his warm hospitality, and exciting discussions alleviate this hard journey. Thank you for being not only my colleague but also my friend.

I would like to thank also to my family and friends. We did it. I am grateful to my parents for their love, support, and motivation. I have climbed up one stair more thanks to them. Thanks to my sister for hearing me and motivating me to continue. I just wanted to set a good example for you. Finally, thanks to my husband, who stayed with me always. Together we learn, and together, we achieve. Thank you for standing by my side when times got hard and for making me laugh when I even did not want to smile.

Abstract

In the last years, Online Social Networks (OSNs) like Facebook and Foursquare have become a popular way to communicate and share information between users. OSNs are virtual communities, which contain information about the users and the relationships existing between them, e.g., friends. In addition to allowing interaction between users, OSNs usually offer different kinds of services to their users, e.g., querying friends within a given distance. To be able to access these services, users may be required to store in the OSN systems a variety of information such as their physical positions. Since most of the information stored in OSNs about their users is private, it is essential to protect the information from unauthorized access to avoid secrecy issues. For this purpose, OSNs use access control systems. These systems have three main components, namely, access control policies, access control model, and authorization mechanism. The access policies allow users to specify who can access their resources. The access control model provides the syntax and semantics to formalize access control policies. The formal representation of access control policies in an access control model are called authorizations. The authorization mechanism, which is managed by the OSN providers, enforce the authorizations.

Although different access control systems have been proposed in the literature, there are two main open issues concerning these systems that can influence the spread of OSNs. The first issue is related to the flexibility of access control models. One of the major challenges of OSNs is to promote information sharing among their users. Users usually tend to share information only with users who fulfill certain conditions; otherwise, they do not. To this end, access control models should provide flexibility to the policy specifiers by allowing them to express conditions on access to their data. When users decide who may access their resources, the access conditions depend on social factors and human behavior. Studies in fields like psychology and sociology have revealed that, although humans are self-interested, they often deviate from this attitude reciprocally. Reciprocity means that, in response to friendly actions, people are more cooperative. Hence, reciprocity is a powerful determinant of human behavior. However, existing access policies do not capture this reciprocity phenomenon, which may restrain users from sharing information. The second issue is that users have to trust the OSN providers to protect

their data by enforcing the authorizations. However, recent privacy breaches have put into question the trustworthiness of the service providers. The increasing economic profit obtained from selling private data increases the temptation of the providers to commit fraud.

In this thesis, we develop techniques and models to address these two issues. Our work is divided into three parts.

Our first contribution deals with the flexibility issue of access control models. Here, we propose and define the syntax and semantics of a new type of authorization, called *mutual*, which allows to model reciprocal behavior. Reciprocity comes into play with access control when persons grant access to their resources to users that allow them the same. We use location-based services as an example to deploy *mutual* authorizations. To this end, we propose two approaches to integrate *mutual* authorizations into these services. Moreover, we prove the soundness of both approaches and determine, by means of complexity analyses, under which conditions each approach performs better than the other.

Our second and third contributions address the distrust of users toward the service providers from two different angles. Our second contribution considers the scenario in which the user, i.e., the entity who wants to perform queries over the data, is also the data owner. However, due to resource constraints, the user does not want to maintain the data by himself. He wants to outsource it to a service provider and to be able to retrieve, given a query, a portion of the data that satisfies the query computation. In this case, since there is a single user, who owns the data, there is no need of access policies. Therefore, in this scenario, the problem of trust in service providers can be reduced to secretizing the outsourced data. In addition, since the query is computed at the user-side, using the received data portion, and because there is a negative correlation between secrecy and performance, it is important for the user to be able to adjust between secrecy and performance. This kind of scenario often applies to start-ups due to the economic and organizational advantages of database-as-a-service. Especially in this sector, many data exhibit a graph structure, e.g., protein networks, road networks and power-grid networks. Here, we propose a bucketization approach for secure outsourcing of graph-structured data that offer provable secrecy guarantees. Our approach allows users to adjust between the levels of secrecy and performance. In addition, to facilitate query planning, we develop a model that can predict the behavior of our algorithm.

Our third contribution considers the case in which the user is not allowed to access the data needed to compute queries. However, users are allowed to access query results over the data. In this scenario, typically, there are multiple users, each

user owns a different portion of the data, and every user can access to query results over data owned by others based on the specified access policies. Then, the OSN provider has to perform the core computation needed, and the user gets access only to the result of the services provided by the OSN. For this scenario, we develop two methods, which combine existing encryption schemes, to allow users of OSNs to query friends within a given distance. Both approaches include a revocation feature and provide secrecy guarantees under collusion assumption, i.e., an adversary can collude with the service provider. Besides, to evaluate and compare the performance of our approaches, we provide complexity analyses of them. Our analyses tell us which approach performs better at each entity involved in the system.

This dissertation includes an extensive experimental analysis of all our approaches based on synthetic and real-world datasets, which confirm the effectiveness of our methods.

Deutsche Zusammenfassung

In den vergangenen Jahren haben sich Online Social Networks (OSNs) wie Facebook und Foursquare zu einer beliebten Möglichkeit der Kommunikation und des Teilens von Informationen unter Nutzern entwickelt. OSNs sind virtuelle Communitys, die Informationen über die Nutzer und die zwischen ihnen bestehenden Beziehungen, wie z. B. Freundschaften, enthalten. Zusätzlich dazu, dass eine Interaktion der Nutzer untereinander ermöglicht wird, bieten OSNs ihren Nutzern normalerweise verschiedene Arten von Dienstleistungen an, wie z. B. die Abfrage nach Freunden innerhalb einer bestimmten Entfernung. Um auf diese Dienstleistungen zugreifen zu können, kann es sein, dass Nutzer darum gebeten werden, in den OSN-Systemen eine Reihe von Informationen, wie z. B. ihre physische Position, zu speichern. Da die meisten der in OSNs gespeicherten Informationen zu deren Nutzern privater Natur sind, ist es von wesentlicher Bedeutung, die Informationen vor unbefugtem Zugriff zu schützen, um Geheimhaltungsprobleme zu vermeiden. Zu diesem Zweck verwenden OSNs Zugriffskontrollsysteme. Diese Systeme haben drei Hauptkomponenten, nämlich die Zugriffskontrollrichtlinien, das Zugriffskontrollmodell und den Autorisierungsmechanismus. Die Zugriffskontrollrichtlinien ermöglichen es Nutzern zu spezifizieren, wer auf deren Ressourcen zugreifen darf. Das Zugriffskontrollmodell bietet die Syntax und Semantik, um die Zugriffskontrollrichtlinien zu formalisieren. Die formale Repräsentation der Zugriffskontrollrichtlinien in einem Zugriffskontrollmodell wird als Autorisierung bezeichnet. Der Autorisierungsmechanismus, welcher von den OSN-Anbietern verwaltet wird, setzt die Autorisierungen durch.

Obwohl in der Literatur verschiedene Zugriffskontrollsysteme vorgeschlagen wurden, gibt es zwei Hauptprobleme in Bezug auf diese Systeme, die sich auf die Verbreitung von OSNs auswirken können. Das erste Problem bezieht sich auf die Flexibilität von Zugriffskontrollmodellen. Eine der größten Herausforderungen von OSNs besteht darin, das Teilen von Informationen unter ihren Nutzern zu fördern. Nutzer neigen normalerweise dazu, Informationen nur mit Nutzern zu teilen, die bestimmte Bedingungen erfüllen; andernfalls tun sie es nicht. Zu diesem Zweck sollten Zugriffskontrollsysteme den Spezifizierern der Richtlinien Flexibilität bieten, damit diese die Bedingungen bezüglich des Zugriffs auf ihre Daten ausdrücken können. Wenn Nutzer entscheiden, wer auf ihre Ressourcen

zugreifen darf, hängen die Zugriffsbedingungen von sozialen Faktoren und menschlichem Verhalten ab. Studien in Fachgebieten wie der Psychologie und der Soziologie haben nachgewiesen, dass Menschen zwar ein Selbstinteresse haben, oftmals jedoch gegenseitig von dieser Haltung abweichen. Gegenseitigkeit bedeutet, dass Menschen als Antwort auf freundliche Handlungen kooperativer werden. Daher ist Gegenseitigkeit eine starke Determinante in Bezug auf menschliches Verhalten. Bestehende Zugriffsrichtlinien erfassen dieses Phänomen der Gegenseitigkeit jedoch nicht, was dazu führen kann, dass Nutzer davon abgehalten werden, Informationen zu teilen. Das zweite Problem besteht darin, dass Nutzer OSN-Anbietern dahingehend vertrauen müssen, dass sie ihre Daten schützen, wenn sie die Autorisierungen durchsetzen. Aktuelle Datenschutzverletzungen haben die Vertrauenswürdigkeit der Dienstleistungsanbieter in Frage gestellt. Scheinbar steigert der zunehmende wirtschaftliche Gewinn, der aus dem Verkauf personenbezogener Daten erzielt wird, die Versuchung der Anbieter, Betrug zu begehen.

In dieser Dissertation werden Techniken und Modelle entwickelt, um auf diese zwei Probleme einzugehen. Die Arbeit ist in drei Abschnitte aufgeteilt.

Der erster Beitrag behandelt das Flexibilitätsproblem von Zugriffskontrollmodellen. Hier schlagen wir die Syntax und Semantik einer neuen Art von Autorisierung vor, die als gegenseitig bezeichnet wird und es ermöglicht, wechselseitiges Verhalten zu modellieren. Gegenseitigkeit kommt im Rahmen der Zugriffskontrolle zum Zuge, wenn Personen jenen Nutzern den Zugriff auf ihre Ressourcen gewähren, die ihnen erlauben, das Gleiche zu tun. Wir verwenden standortbasierte Dienstleistungen als Beispiel für den Einsatz gegenseitiger Autorisierungen. Zu diesem Zweck schlagen wir zwei Ansätze vor, um gegenseitige Autorisierungen in diese Dienstleistungen zu integrieren. Darüber hinaus weisen wir die Stimmigkeit beider Ansätze nach und bestimmen auf dem Wege von Komplexitätsanalysen, unter welchen Bedingungen jeder Ansatz jeweils leistungsfähiger ist als der andere.

Unsere zweiten und dritten Beiträge gehen aus zwei verschiedenen Blickwinkeln auf das Misstrauen von Nutzern bezüglich der Dienstleistungsanbieter ein. Unser zweiter Beitrag erörtert das Szenario, in welchem der Nutzer, d. h. die Einheit, welche Abfragen von Daten durchführen möchte, auch Eigentümer der Daten ist. Aufgrund von Ressourcenbeschränkungen möchte der Nutzer die Daten jedoch nicht allein verwalten. Er möchte dies an einen Dienstleistungsanbieter auslagern, um bei einer Abfrage einen Teil der Daten abrufen zu können, welche der Durchführung der Abfrage Genüge leisten. In diesem Fall besteht kein Bedarf an Zugriffsrichtlinien, da es einen einzelnen Nutzer gibt, der Eigentümer der Daten ist. Daher kann in diesem Szenario das Vertrauensproblem bezüglich Dienstleistungsanbietern auf die Geheimhaltung ausgelagerter Daten reduziert werden.

Außerdem ist es für den Nutzer wichtig, in der Lage zu sein, eine Anpassung zwischen Geheimhaltung und Leistung vorzunehmen, da die Abfrage nutzerseitig, unter Verwendung des erhaltenen Datenabschnitts, berechnet wird und weil eine negative Korrelation zwischen Geheimhaltung und Leistung besteht. Diese Art von Szenario findet aufgrund der wirtschaftlichen und organisatorischen Vorteile von „Database-as-a-Service“ oft bei Startup-Unternehmen Anwendung. Insbesondere in diesem Bereich weisen viele Daten eine Graphstruktur auf, z. B. Protein-Netzwerke, Straßen-Netzwerke und Stromnetz-Netzwerke. Hier schlagen wir einen Gruppierungsansatz für die sichere Auslagerung von Daten mit Graphstrukturen vor, wobei nachweisbare Geheimhaltungsgarantien geboten werden. Unser Ansatz ermöglicht es Nutzern, Anpassungen zwischen Ebenen von Geheimhaltung und Leistung vorzunehmen. Zusätzlich entwickeln wir zur Erleichterung der Planung von Abfragen ein Modell, welches das Verhalten unseres Algorithmus vorhersagen kann.

Unser dritter Beitrag berücksichtigt den Fall, in dem es einem Nutzer nicht ermöglicht wird, auf Daten zuzugreifen, die zur Durchführung von Abfragen nötig sind. Die Nutzer haben jedoch Zugriff auf die Ergebnisse der Abfrage bezüglich der Daten. In diesem Szenario gibt es typischerweise mehrere Nutzer, wobei jeder einen anderen Teil der Daten besitzt, und jeder Nutzer auf Basis von spezifizierten Zugriffsrichtlinien auf Abfrageergebnisse bezüglich der Daten zugreifen kann, die anderen gehören. Dann muss der OSN-Anbieter die erforderliche Kernberechnung durchführen, und der Nutzer kann nur auf das Ergebnis von Dienstleistungen zugreifen, die vom OSN geboten werden. Für dieses Szenario entwickeln wir zwei Methoden, welche bestehende Verschlüsselungsschemata kombinieren, um es Nutzern von OSNs zu ermöglichen, Abfragen bezüglich Freunden in einer bestimmten Entfernung durchzuführen. Beide Ansätze beinhalten eine Aufhebungsfunktion und bieten Geheimhaltungsgarantien unter der Annahme geheimer Absprachen, d. h. ein Gegenspieler kann mit dem Dienstleistungsanbieter zusammenspielen. Daneben bieten wir Komplexitätsanalysen unserer Ansätze, um diese bewerten und vergleichen zu können. Unsere Analysen teilen uns mit, welcher Ansatz in jeder Einheit, die in dem System involviert ist, leistungsfähiger ist.

Diese Dissertation beinhaltet eine umfassende experimentelle Analyse all unserer Ansätze auf Basis von synthetischen und realen Datensätzen, welche die Wirksamkeit unserer Methoden bestätigen.

Contents

1 Introduction

With an estimation of 2.77 billion social media users around the globe 2019[1], Online Social Networks (OSNs) have become an important platform for interconnecting users and sharing information [BF18]. Besides allowing interaction between users, OSNs offer users different services such as location-based services or dating services. To be able to access these services, users have to store personal information such as physical positions, age, and more, in the OSN system. Consequently, OSNs store a vast amount of information about their users. This amount of information and the widespread use of OSNs open up opportunities for the development of new applications and services. However, they also increase the secrecy and privacy issues associated with these networks. Therefore, it is essential to protect the information stored in OSNs from unauthorized access. We call this kind of data protection *data secrecy*. Enforcing data secrecy is particularly crucial in OSNs. The reason is that the unauthorized access to personal information about the users can expose the information to a large number of users, which may cause unpredictable damage. In this regard also, the current EU Data Protection legal framework [fFRoE18] and the latest data secrecy debates of the Federal Trade Commission [Com19] have shown a strong need for implementing measures to guarantee the right to personal data protection.

Access control systems have become one of the central components of OSNs. They are used to prevent unauthorized access to the information of OSNs users [HAJ12]. Access control systems have three main components, namely, access control policies, access control model, and authorization mechanism [Fer10]. In what follows, we briefly introduce these components.

First, *access control policies* are high-level rules, i.e., plain-language rules, that serve to regulate access control, e.g., *"Medical records can be accessed only by medical staff"*. These policies depend on several factors such as legislation, e.g., EU data protection legislation, the domain in which the data will be managed, e.g., education or healthcare, or specific requirements of the resource owners. Second, *access control models* are used to formalize access control policies. Access control models provide the syntax and semantics needed for this formalization. The for-

[1] `https://www.statista.com/`

mal representation of access control policies in an access control model is called authorizations. Intuitively, authorizations state who can perform which actions, e.g., write or read, on which resources under which conditions. We formalize this notion in Section 2.2. This formal representation of access control policies allows proving the security properties meet by access control systems [Fer10]. Third, the *authorization mechanism* is the software module that enforces the authorizations. The enforcement is carried out by technical security mechanisms (software and hardware), such as firewalls or cryptographic techniques like encryption and digital signatures. The authorization mechanism implements the access controls imposed by the access control policies and formally defined in the access control model. Figure 1.1 illustrates the main components of an access control system.

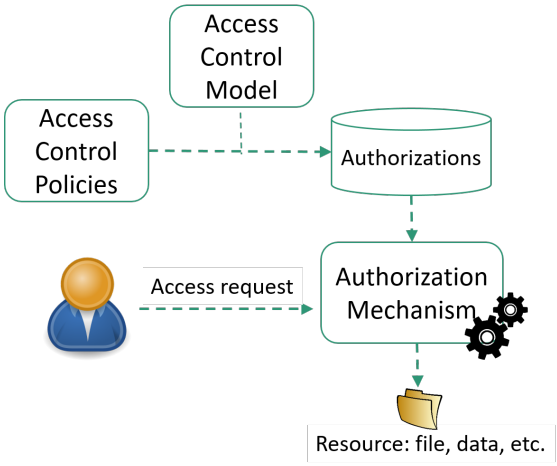


Figure 1.1: Access Control System - Main Components

Although different access control systems exist in the literature, two main open issues concerning these systems can influence the spread of OSNs. The first issue involves the access control model, and the second one the authorization mechanism. In what follows, we explain these two issues.

The first issue is related to the flexibility offered by existing access control models. One of the major challenges of OSNs is to promote information sharing among their users. Users usually tend to share information only with users who fulfill certain conditions; otherwise, they do not. To this end, access control models should provide flexibility to the policy specifiers by allowing them to express conditions on access to their data. When users decide who may access their resources, the access conditions depend on social factors and human behavior. Studies in fields like psychology and sociology have revealed that, although humans are self-interested, they often deviate from this attitude reciprocally. Reciprocity means that, in

response to friendly actions, people are more cooperative. Hence, reciprocity is a powerful determinant of human behavior. However, existing access control models do not capture this reciprocity phenomenon, which may restrain users from sharing information.

As mentioned above, the second issue concerns the authorization mechanism. In OSN systems, besides hosting the data of the users, the service providers host the authorizations and run the authorization mechanism. That is, users have to trust the OSN provider to control access to their data correctly. However, recent privacy breaches have put into question the trustworthiness of the service providers. The recent Facebook-Cambridge Analytica scandal [Com18, Con18] is one of the most illustrative examples of such privacy breaches in which personal data of millions of users of Facebook were harvested without their consent. Apparently, the increasing economic profit obtained from selling private data increases the temptation of the providers to commit fraud.

In the following, we give an overview of the challenges involved in this research area and our contributions, Section 1.1, and present the structure of the thesis, Section 1.2.

1.1 Challenges and Contributions

In this thesis, we focus on open issues concerning the components of access control systems. Here, we propose solutions to tackle two critical issues: (1) the flexibility problem of existing access control models, and (2) the distrust of users toward the service providers, i.e., the entities responsible for hosting the data and running the authorization mechanism. In the following subsections, we describe the challenges and our major contributions to each of these two problems.

1.1.1 Access Control Model - Flexibility Problem

A large body of evidence in fields like psychology, economics, and sociology indicates that reciprocity is a powerful determinant of human behavior. However, existing access control models do not capture this reciprocity phenomenon. Reciprocity comes into play with access control when persons grant access to their resources to users that allow them the same. In other words, the basic idea is that I let you profit from my resource if and only if I can benefit from yours, which implies an exchange of resources between two parties. Such an exchange should be fair. Fairness concretely implies two principles. First, users must have a guaranteed right to profit from users who have benefited from their resources. Second,

the payoff that a pair of users receive from exchanging resources between them has to be equal, to some adaptable extent.

All existing access control models, such as [SCFY96, OP03, TLT15], consider only two kinds of grants, allow and deny. These two grants are not enough to model the reciprocity principle. Therefore, a new type of grant is needed. We call this new grant *mutual* and authorizations that make use of it *mutual* authorizations. Integrating reciprocity into access control models is subject to several challenges.

First, access control models have to provide the syntax and semantics to capture the reciprocity principle. For *mutual* authorizations, in particular, the semantics are not trivial. Contrary to authorizations with only the grants *allow* and *deny*, the semantics of *mutual* authorizations have to consider not only the authorization that a user has received but also the ones that the user has assigned. Besides, the semantics have to be suitable for integration in existing access control models.

Second, it is not obvious how to guarantee a fair exchange of resources when using *mutual* authorizations. Such fairness is naturally guaranteed when resources have the same degree of sensitivity, .e.g., physical positions. But it is not the case for complex scenarios where resources have different degrees of sensitivity, e.g., health records. One can guarantee such fairness by specifying that users can exchange only resources with the same degree of sensitivity. The problem here lies in defining the sensitivity of each resource. If we let the system or a trusted authority assign the degree of sensitivity, the owners of the resources may not agree with the assigned sensitivity. On the other hand, if the owners assign the degree of sensitivity to their resources individually, the assignment will be subjective. To illustrate this, consider two users Alice and Bob, who want to exchange their health records. Whereas Alice has a sexual disease, Bob is in perfect health. If both users assign the same degree of sensitivity to their resources, an exchange of resources between them could happen. However, Alice may not find fair opening her health record to Bob.

Another challenge is to incorporate this new kind of authorizations with existing services, such as location-based services (LBS). This integration has to reuse existing implementations, be efficient, and guarantee data secrecy, i.e., only authorized users can access the services. In the following, we discuss our contributions to overcome these challenges.

Contributions: We introduce a new kind of grant, called *mutual*. It allows to model reciprocal behavior. We extend Attribute-based access control model (ABAC), with our *mutual* grant, and define the syntax and semantics of *mutual* authorizations. We selected ABAC because it is one of the most general access

control models existing in the literature. To focus on the main idea of *mutual* authorizations, we consider at first scenarios in which resources have the same degree of sensitivity. Then, we extend our basic model, i.e., the syntax and semantics of *mutual* authorizations, to support a general setting in which resources have different degrees of sensitivity.

Our extension aims to guarantee a fair exchange of resources when dealing with resources with different degrees of sensitivity. However, deciding whether two resources have the same degree of sensitivity is a general problem, and it is far from being solved in an automated manner. Therefore, our extended model of *mutual* authorizations only alleviates this problem. It lets the owners assign the degrees of sensitivity to their resources, and their peers can evaluate such an assignment. We use these evaluations to compute a trust value for each user, and we let users specify the minimum level of trust that their peers should have to exchange resources with them. We call this extension, trust-based authorizations.

Next, we show how to integrate and evaluate *mutual* authorizations into existing services; to this end, we use LBS. Specifically, we focus on k-nearest neighbor queries and range queries. To this end, we propose two approaches, called *filtering-querying* and *querying-filtering*. We prove the correctness and completeness of both approaches. We also conduct complexity analyses of them and evaluate both approaches experimentally. Besides, we conduct experiments to evaluate how *mutual* authorizations affect the performance when deciding whether an access request is allowed or denied.

1.1.2 Authorization Mechanism - Distrust Problem

Data secrecy can be addressed straightforwardly by trusting the entity that manages the authorization mechanism—the service provider. That is, users have to trust that the service provider will enforce access control on their data based on their authorizations. However, as explained before, the increasing privacy breaches in OSNs lead us to believe that fully trusting the service providers is not a realistic assumption in OSN scenarios. We tackle the distrust of users towards service providers from two different angles, as we explain in what follows.

Case 1 - Data outsourcing: Here, we consider the scenario in which the user, i.e., the entity who wants to perform queries over the data, is also the data owner. However, due to resource constraints, the user does not want to maintain the data by himself. He wants to outsource it to a service provider and to be able to retrieve, given a query, a portion of the data that satisfies the query computation. This kind of scenario belongs to the Database as a Service (DBaaS) paradigm, which is one of the emerging services of cloud computing. The DBaaS model has many

benefits, including economic and organizational advantages. For these reasons, it has attracted the interested of many enterprises, especially start-ups. Particularly in this sector, many data exhibit a graph structure, e.g., protein networks, road networks, and power-grid networks.

In the DBaaS model, data secrecy involves two aspects [CFB05]: data secrecy with respect to the users and data secrecy with respect to the service providers. These aspects aim to protect data against unauthorized access by end-users, and by the service providers, respectively. Since in this scenario, we consider a single user, who owns the data and wants to perform queries over it, data secrecy with respect to the end-users is nonessential. That is, access control policies are not needed. Therefore, in this scenario, the problem of trust in the service providers can be reduced to secretizing the outsourced data and guaranteeing data secrecy with respect to the service providers.

One of the most challenging issues in this scenario is how to trade-off between the obtained data secrecy guarantees and the efficiency of the envisioned solution. Since the query is computed at the user-side using the retrieved data portion from the service provider, the accuracy of this portion affects the query performance. Further, data secrecy correlates negatively with performance. Therefore, the difficulty lies in retrieving an accurate data portion that satisfies the query conditions while guaranteeing data secrecy.

Another challenge is related to the type of data structure considered in this scenario, i.e., graphs. Graph-structured data store information about the nodes and their relationships (edges). Especially in this kind of data, these relationships are always present, which is not the case for other data structures. The information stored in a node and its relationships both can be used to identify an individual in a graph. Therefore, approaches for secretizing graph-structured data should guarantee that they do not leak this kind of information.

Next, the data secrecy guarantees have to be provable, which requires a rigid definition of secrecy, i.e., to define the type of adversaries one is dealing with, the desired secrecy guarantees, and the information that one accepts to leak. In the following, we describe our major contributions in this data outsourcing scenario.

Contributions: First, we define a secrecy notion for graph-structured data based on the concepts of indistinguishability and searchable encryption. Next, we propose a bucketization approach for secure outsourcing of graph-structured data that offers provable secrecy guarantees and allows users to adjust between the levels of secrecy and performance. While our approach works for all types of graph queries in principle, we focus on neighbor and adjacency queries. These queries are essen-

tial information needs regarding graphs [MP10]. We describe the specifics for these queries, such as division of work between client and server. Besides, to facilitate query planning, we develop a performance model that can predict the behavior of our algorithm depending on its parameter values and properties of the input graph. Finally, we demonstrate with a set of experiments the accuracy of our performance model and the efficiency of our approach for query processing.

Case 2 - Enabling access to services: Here, we consider the scenario in which the user is not allowed to access the data needed to compute queries. However, users are allowed to access query results over the data. In this scenario, typically, there are multiple users, each user owns a different portion of the data, and every user can access to query results over data owned by others based on the specified access policies. This kind of scenario is a typical one of the Web 2.0 social services like OSNs. In the last years, LBS have become a popular feature of these networks due to the increasing diffusion of mobile devices. OSNs that embraces LBS, such as querying friends within a given distance, are known as mobile social networks (MSNs). We focus on this kind of social network.

Similar to the previous case, Case 1, in this scenario, the data is outsourced to an untrusted service provider, i.e., the social network provider. However, since this scenario considers multiple users and each user owns a portion of the data, users have to specify access control policies on their data. Therefore, data secrecy has to be guaranteed with respect to both the users and the service provider. That is, the service provider performs the core computation needed, the user gets access only to the result of the services provided by the MSN, and the data is kept secret from both unauthorized users and the service provider. In addition, because the service provider is untrusted, an important problem in MSNs is to guarantee data secrecy under the assumption of collusion attacks where adversaries, including malicious users, collude with the service provider to gain unauthorized access to information. Preserving secrecy in MSNs is challenging.

First, the limited computation power of mobile devices requires mobile users to outsource any heavy computation task. This not only restricts the choice of encryption schemes that can be used but also the design alternatives for a solution.

Second, although encryption guarantees that only holders of the right key can decrypt a given ciphertext, because of our collusion assumption, this guarantee does not hold anymore. To illustrate this, consider two entities A and B, a message m, and its corresponding ciphertext c. Assume that A is not authorized to access m, whereas B is authorized and has the key to decrypt c. If A colludes with B, A can either get the key from B to decrypt c or send c to B to decrypt it and get back m. To solve this problem, one can add further entities in the

system architecture and use multi-layer encryption. Then, one needs to assign the ciphertext and the keys of each encryption layer to the entities so that in case of collusion, unauthorized entities cannot gain access to the information. That is, the unauthorized entities cannot access either the ciphertext or all the keys needed to decrypt all encryption layers because a non-colluding entity holds the ciphertext or at least one of the keys.

Third, the envisioned solution has to consider that in a multi-user setting, such as our MSN scenario, several owners want to share information with authorized users. To this end, one can use different cryptographic techniques. However, by comparing the known advantages and disadvantages of each cryptographic technique, it is not obvious which one performs better in our scenario. In what follows, we discuss our major contributions to overcome these challenges.

Contributions: For this scenario, we develop two methods, which combine existing encryption schemes and use multi-layer encryption with an adequate key distribution among the entities of the system, to allow users of MSNs to query friends within a given distance. The main difference between the first and the second approach is that they use, among other encryption schemes, symmetric and attribute-based encryption, respectively. Both approaches include a revocation feature and provide secrecy guarantees under collusion assumption, i.e., an adversary can collude with the service provider. Specifically, we guarantee that any adversary, including the service provider, is not able to learn: the physical position of the users, the distance between his position and that of the users, and whether two users are allowed to learn the distance between them. Besides, to evaluate and compare the performance of our approaches, we provide complexity analyses of them. Our analyses tell us which approach is better at the user side and the service provider side. We conduct experiments to validate the result of our complexity analyses and finally determine which approach performs better in practice.

1.2 Dissertation Outline

The remaining chapters of this dissertation are organized as follows:

Chapters 2 and 3 introduce access control systems and cryptography, respectively. These two research areas are relevant to the work described in this dissertation. We formally define the basic notions related to these areas that are needed to build our approaches.

Chapter 4 presents the first part of our contribution that focuses on the Access Control Model and its flexibility problem to capture. We define the syntax and semantics of a new type of authorization — called *mutual authorizations*. *Mutual authorizations* allow us to model the reciprocity phenomenon, which is a fundamental aspect that shapes human behavior. Moreover, we use location-based services to show how to integrate *mutual authorizations* with existing services. We conduct experiments to evaluate our approaches.

Chapter 5 presents the second part of our contribution that focuses on the Access Control Mechanism. In this chapter, we study the data outsourcing scenario and tackle the distrust of users towards the service providers. Here, we consider a single user who owns a graph-structured data and wants to perform queries over it while keeping the information secret. We start by formally defining a secrecy notion for graph-structured data. Next, we present our bucketization approach for secure outsourcing of graph-structured data, which combine encryption and bucketization techniques and offer provable secrecy guarantees. We show that finding an optimal bucketization tailored to graph-structured data is NP-hard. We therefore come up with a heuristic, which guarantees that the worst bucketization solution will be off by a factor of $\frac{11}{9}$ of the optimal one. We also introduce a performance model to evaluate the behavior of our algorithm and facilitate query planning. Finally, we run experiments to evaluate our approach and validate our performance model.

Chapter 6 also focuses on the Access Control Mechanism but from a different perspective compared to Chapter 5. Here, we consider a mobile social network scenario, where multiple users own a portion of the data and want to access location-based services while keeping their information secret from the service provider and unauthorized users. To solve this problem, we come up with two approaches, which combine existing encryption schemes and use multilayer encryption and provide secrecy guarantees under collusion assumption, i.e., an adversary can collude with the service provider. Next, we present complexity analyses of our approaches, which allow us to compare them, and conduct experiments to validate such analyses.

Lastly, Chapter 7 summarizes the major contributions of this dissertation and gives an outlook of open research questions.

2 Access Control Systems

Two main research areas are relevant to the work presented in this dissertation: Access Control Systems and Cryptography. This chapter gives an overview of the first research area, and the next Chapter, Chapter 3, covers the second one.

Access control systems are one of the most relevant components not only of OSNs but also of any data management system. The main goal of these systems is to protect resources from unauthorized operations based on the access conditions determined by the policy specifiers. Policy specifiers are the organizations or individuals responsible for managing the resources—usually the resource owners. As illustrated in Figure 1.1, access control systems have three components, namely, access control policies, access control model and authorization mechanism. In what follows, we describe these three components.

2.1 Access Control Policies

Access control policies are high-level rules according to which access control must be enforced. These policies must capture all real-world access conditions that need to be imposed in a system, such as the ones established by law, practices, organizational regulations, or specific requirements of the resource owners. Defining access control policies is a challenging process because it requires to identify all access conditions that apply to the specific scenario in which the access control system will be implemented.

2.2 Access Control Models

Access control models provide the syntax and semantics to formalize access control policies into authorizations. These models have to be flexible and expressive enough to capture and formalize all the access conditions expressed in the access control policies.

In what follows, we introduce the elements of an authorization. We then provide a set of quality criteria to evaluate the expressive power of access control models.

Finally, we give an overview of one of the most general access control models—Attribute-based Access Control (ABAC).

2.2.1 Authorization

Authorizations are the basic building block of access control systems. They specify the operations that a subject can perform an operation on a resource. An authorization has four essential elements: subjects, resources, operations and grants.

- *Subjects:* Subjects specify the entities that can request to perform operations on resources. Examples of subjects are: human-beings, system processes or devices.
- *Resources:* Resources are the entities that require protection from unauthorized access. Example of resources are: files, applications or devices.
- *Operations:* Operations are the types of actions that subjects can request to perform on resources. Examples of operations are: update, read or write.
- *Grants:* Grants are the rights to execute operations. Existing access control models support two grants: *allow* and *deny*. That is, the set of grants is fixed, which is not the case for the sets of subjects, resources, and operations; the access control scenario defines these sets.

  We call the authorizations that use the grants *allow* and *deny*, positive and negative authorizations, respectively. Access control models can use positive and negative authorizations in a mutually exclusive way or a combination. In the first case, mutually exclusive way, the access control model specifies explicitly one type of authorization, positive or negative, and implicitly the other type. Specifying explicitly positive authorizations indicates that an access request is granted if there is a positive authorization for it; otherwise, it is denied. The opposite happens for negative authorizations specified explicitly and positive ones specified implicitly. In the second case, the access control model combines positive and negative authorizations. Such a combination facilitates to capture access exceptions, see Example 2.1.

  Example 2.1. Consider that we want to allow all members of a group to read a file, except to one specific member Alice. If positive and negative can be combined, we can specify this requirement by assigning a positive authorization to the group and a negative authorization to Alice. It is more difficult to capture this access requirement if one has to specify explicitly positive authorizations. In that case, we would

have to specify a positive authorization for each member of the group except Alice.

Although specifying explicitly negative authorizations will solve the problem of Example 2.1, the opposite example of denying all but one will arise the same difficulty when using negative authorizations explicitly.

2.2.2 Quality Criteria for Access Control Models

Various access control models have been proposed in the literature [SCFY96, OP03, TLT15, HFK+13]. All of them have different features and methods to provide access control, and there is no model that suits all access control scenarios. In the following, we present a set of quality criteria to evaluate the expressive power of these models. While the criteria described below are not exhaustive, they do provide the most important metrics to measure the quality of access control models.

- *Access conditions flexibility:* Access control policies can constraint access depending on the characteristics of the subjects, the resources, and the environment. Environment conditions are characteristics relevant to the access requests that are independent of the subjects and resources such as current time, day of the week, or temperature. For instance, a project manager may want to allow his team members access to the project's documents only on working days. Access control models should provide the language to capture different access conditions specified in terms of the subjects, the resources, and the environment.

- *Groups support:* Access control models should support authorizations specified for groups of subjects, e.g., doctors, and resources, e.g., file directories. Supporting groups simplifies the management of authorizations because an authorization that specifies a group involves all the members of such a group. Supporting groups is an essential feature in complex environments with a large number of resources and subjects.

- *Handling authorization conflicts:* Authorization conflicts happen when a subject receives more than one authorization with different grants for the same operation on the same resource, see Example 2.1. Access control models should support conflict resolution strategies, i.e., strategies to decide which authorization prevails over the others when in conflict. Several conflict resolution strategies have been proposed in the literature such as and *deny-take -precedence* and *recency-overrides*. According to *deny-take -precedence*, neg-

ative authorizations take precedence over positive ones, whereas according to *recency-overrides*, authorizations specified later cancel earlier ones.

- *Administrative authorizations support:* Administrative authorizations define the entities that can create, modify, or delete authorizations to control access to the resources. Examples of these entities are central authorities or resource owners. In the first case, only the central authority can specify the authorizations. In the second one, each resource is associated with an owner, and only the owner can specify the authorization for their resources. Access control models should provide the language to define administrative authorizations.

2.2.3 Attribute-based Access Control

One of the most general access control models is ABAC. In recent years, ABAC has gained considerable attention from business, academia and standard bodies [BSK16, HFK+13]. ABAC specifies the authorizations in terms of defined attributes of the subjects, defined attributes of the resources and defined environment conditions. Attributes are characteristics of the subjects, resources, and environment conditions, e.g., gender, name. They are not atomic, especially, their value is a non-empty set of atomic values. The subject attributes, resource attributes, and the environment conditions have to be: defined in the system, assigned a set of allowable values, and managed under an authority. Such an authority has to guarantee that unauthorized entities cannot tamper with the attributes to gain access to the resources. Without this information, specifying authorizations in ABAC will fail. Next, we describe how to specify the essential elements of an authorization in ABAC.

- *Subject specification:* $Attr_{subject} = \{a_{subject_1}, a_{subject_2}, \cdots, a_{subject_n}\}$ stands for the set of n subject attributes. So-called subject constraints specify the subjects of an authorization. A subject constraint $Cons_{subject}$ has the syntax:

$$Cons_{subject} = Cons_{subject} \wedge Cons_{subject} \mid a_{subject_i} = value \mid a_{subject_i} \geq value \mid a_{subject_i} \leq value$$

  where $1 \leq i \leq n$ and *value* refers to an atomic value. Since the attribute $a_{subject_i}$ is not atomic, for a given subject s, the expression $a_{subject_i}(=|\geq|\leq) value$ resolves to $\exists x \in s.a_{subject_i}, x\ (=|\geq|\leq)\ value$, where $s.a_{subject_i}$ denotes the set of atomic values of the attribute $a_{subject_i}$ for s. We use S to denote the set of all subjects. Given a subject constraint $Cons_{Subject}$, the set of subjects induced by the constraint contains all subjects $s \in S$ that fulfill

$Cons_{subject}$. Then, a subject specification is the set of all subjects that meet a given subject constraint. For instance, the set of subjects induced by the constraint $gender = female \wedge income < 60k$ contains all subjects $s \in S$ whose gender is female and their income is less than 60k.

- *Resource specification:* $Attr_{resource} = \{a_{resource_1}, a_{resource_2}, \cdots, a_{resource_m}\}$ stands for the set of m resource attributes. So-called resource constraints specify the resources of an authorization. A resource constraint $Cons_{resource}$ has the syntax:

$$Cons_{resource} = Cons_{resource} \wedge Cons_{resource} \,|\, a_{resource_j} = value \,|\, a_{resource_j} \geq value \,|\, a_{resource_j} \leq value$$

  where $1 \leq j \leq m$ and *value* refers to an atomic value. Since the attribute $a_{resource_j}$ is not atomic, for a given resource res, the expression $a_{resource_j}$ $(=|\geq |\leq)$ *value* resolves to $\exists x \in res.a_{resource_j}, x$ $(=|\geq|\leq)$ *value*, where $res.a_{resource_j}$ denotes the set of atomic values of the attribute $a_{resource_j}$ for res. A resource specification is the set of all resources that meet a given resource constraint.

- *Operation specification:* Contrary to subjects and resources, operations do not have attributes. Operations are fixed set of values, e.g., read or write. The scenario, where the access control system will be deployed, determines the set of operations needed. Op denotes the set of all operations.

- *Grant specification:* Similar to operations, grants are fixed set of values. Gr denotes the set of all grants. In ABAC, the set of grants contains two elements, $Gr = \{allow, deny\}$.

Besides the four elements of an authorization specified above, which are supported by any traditional access control model, ABAC allows specifying, if needed, the environment conditions that need to be evaluated during an access request. For instance, the environment constraint $access\text{-}time = 8:00AM \wedge access\text{-}day = Monday$ specifies that the time and day of access (8:00AM, Monday) have to be evaluated during an access request.

Environment specification: $Attr_{environment} = \{a_{environment_1}, a_{environment_2}, \cdots, a_{environment_l}\}$ stands for the set of l environment attributes. So-called environment constraints specify the environment conditions relevant to the access requests. An environment constraint $Cons_{environment}$ has the syntax:

$$Cons_{environment} = Cons_{environment} \wedge Cons_{environment} \,|\, a_{environment_k} = value \,|\, a_{environment_k} \geq value \,|\, a_{environment_k} \leq value$$

where $1 \leq k \leq l$ and *value* refers to an atomic value. Since the attribute $a_{environment_k}$ is not atomic, for a given environment condition *env*, the expression $a_{environment_k}$ $(=|\geq|\leq)$ *value* resolves to $\exists x \in env.a_{environment_k}, x$ $(=|\geq|\leq)$ *value*, where $env.a_{environment_k}$ denotes the set of atomic values of the attribute $a_{environment_k}$ for *env*. An environment specification is the set of all environment conditions that meet a given environment constraint. For instance, the environment constraint $access\text{-}time = 8:00AM \wedge access\text{-}day = Monday$ specifies that the time and day of access (8:00AM, Monday) have to be evaluated during an access request.

Using the specifications of subjects, resources, operation, grants and environment conditions in ABAC, we define an authorization as follows:

Definition 2.1: Authorization

Let a subject constraint $Cons_{subject}$, a resource constraint $Cons_{resource}$, an operation $op \in Op$, a grant $gr \in Gr$, and an environment constraint $Cons_{environment}$ be given. An **authorization** A is a 5-element tuple $\langle\, Cons_{subject}, Cons_{resource}, op, gr, Cons_{environment} \,\rangle$. The authorization A indicates that the subjects specified by $Cons_{subject}$ has been assigned the grant gr to invoke the operation *op* on the resources specified by $Cons_{resource}$ under the environment conditions specified by $Cons_{environment}$.

Example 2.2 illustrates how to express an authorization in ABAC, Definition 2.1.

Example 2.2. Consider a user Bob who wants to *allow* all persons with the role of *Cashier* to read his file *File1* on Mondays. Let us call this authorization A_{Bob}. In ABAC, A_{Bob} can be expressed as follows: $A_{Bob} = \langle\ role = Cashier, name = File1, read, allow, access\text{-}day = Monday \,\rangle$, where *role* is a subject attribute in $Attr_{subject}$, *name* is a resource attribute in $Attr_{resource}$, *read* is an operation in *Op*, *allow* is a grant in *Gr*, and *access-day* is an environment attribute in $Attr_{environment}$.

2.3 Authorization Mechanism

The authorization mechanism, also known as the reference monitor, is the software module that decides whether an access request is authorized or denied and enforces the decision. To perform these tasks, the authorization mechanism should have access to information about the resources being protected, the subjects who request access, and the authorizations governing access to the resources. The authorization mechanism can use different security techniques to enforce access control such as

physical security devices, firewalls or encryption techniques. Besides, it should have two characteristics [SdV00]: It has to be tamper-proof, i.e., any modification done by an unauthorized entity should be detectable, and non-bypassable, i.e., access requests cannot be circumvented.

3 Cryptography

Data secrecy refers to the protection of information from unauthorized access. One way to ensure data secrecy is to apply encryption techniques on the data before storing it on a storage server. This chapter provides an overview of existing encryption schemes, Section 3.1. Next, it introduces notions that will help us to determine whether a given cryptographic scheme is secure, Section 3.2. Finally, it describes standard secrecy definitions for encryption schemes, Section 3.3.

3.1 Encryption Schemes

There are two main types of encryption techniques: symmetric encryption schemes and asymmetric encryption schemes. Next, we present these encryption schemes.

3.1.1 Symmetric Encryption

Symmetric encryption schemes, also known as private-key encryption schemes, rely on a secret key shared—in advance—by two parties who want to communicate with each other secretly. Intuitively, symmetric encryption schemes work as follows: First, two parties share in advance a key k. Second, to communicate with each other, one party (the sender) encrypts a plaintext message m using the key k and sends the resulting ciphertext c to the receiver. Finally, the receiver takes the ciphertext c, decrypts it using the key k and obtains back the message m. The encryption and decryption processes use the same key k. Therefore, the sender must trust that the receiver (and vice versa) will not reveal their shared key to any entity. Formally:

Definition 3.1: Symmetric Encryption Scheme

A **symmetric encryption scheme** $\mathcal{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms: (1) A key generation algorithm KGen that returns a key k. (2) An encryption algorithm Enc that can be probabilistic or deterministic and takes as input the key k and a message m to return a ciphertext c. (3) A

deterministic decryption algorithm Dec that takes as input the key k and a ciphertext c to return a message m such that $\mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}, m)) = m$.

We write $\mathsf{Enc}(\mathsf{k}, m)$ and $\mathsf{Dec}(\mathsf{k}, c)$ for the operations of encrypting m under key k and decrypting c under key k, respectively.

The encryption algorithm Enc can be probabilistic or deterministic. A probabilistic encryption algorithm involves randomness: Running the encryption algorithm with the same inputs, i.e., a key k and a message m, several times outputs different ciphertexts. Contrary to probabilistic encryption algorithms, deterministic ones involve no randomness: Running the encryption algorithm with the same inputs several times outputs the same ciphertext. This property of deterministic encryption algorithms allows searching on the encrypted data straightforwardly. To illustrate, consider a relational table where each value of the table is encrypted using a deterministic encryption algorithm and a key k. A user, who knows k, can look-up a data m in the encrypted table by encrypting m under key k and using the generated ciphertext to find a match in the encrypted table. Because of the deterministic encryption property, the ciphertext generated during the search process is the same as the one generated when the table was encrypted.

However, this property of the deterministic encryption algorithms allows adversaries to identify whether two or more ciphertexts encrypt the same message. If an adversary finds that a ciphertext occurs more than once, the adversary knows that the underlying encrypted message must also be the same. We give more details about secrecy notions in the next Section, Section 3.2.

With probabilistic encryption algorithms, an adversary cannot identify whether two or more ciphertexts encrypt the same message. The reason is that encrypting the same message under the same key several times yields different ciphertexts. However, the randomness of the encryption process has a disadvantage: searching on the encrypted data requires additional techniques such as the use of indexes. We will deepen these searching techniques in Chapters 5 and 6.

3.1.2 Asymmetric Encryption

Contrary to symmetric encryption schemes, where two parties share a key secretly, using asymmetric encryption schemes, two parties, who want to communicate with each other, do not need to share any key secretly, as we will explain. Asymmetric encryption schemes, also known as public-key encryption, use a pair of keys—one key to encrypt and the other one to decrypt. The encryption key is called the

public key, and it is publicly available to all parties who want to send messages. The decryption key is called the secret key, and only the receiver knows this key.

Intuitively, asymmetric encryption schemes work as follows: First, the receiver generates a pair of keys $(\mathsf{pk}, \mathsf{sk})$, where pk is the public key and sk is the secret key. Second, to communicate with the receiver, the sender encrypts a plaintext message m using the public key pk and sends the resulting ciphertext c to the receiver. Finally, the receiver takes the ciphertext c, decrypts it using the secret key sk, and obtains back the message m. The receiver can make his public key available by publishing it on his webpage, storing it in an open directory, or sending it in the clear to the sender. The secrecy of these encryption schemes relies only on the secret key sk, not in the public key pk. Therefore, the receiver is the only entity responsible for keeping the key sk secret from any entity, including the sender. Formally:

Definition 3.2: Asymmetric Encryption Scheme

An **asymmetric encryption scheme** $\mathcal{AE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms: (1) A key generation algorithm KGen that returns a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$. (2) A encryption algorithm Enc that can be probabilistic or deterministic and takes as input the public key pk and a message m to return a ciphertext c. (3) A deterministic decryption algorithm Dec that takes as input the secret key sk and a ciphertext c to return a message m, such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.

Asymmetric encryption schemes have two main advantages compared to symmetric encryption schemes. First, they address the key-distribution problem of symmetric encryption schemes. That is, two parties who want to communicate with each other do not need to share in advance a secret key. Second, using asymmetric encryption schemes, a user who wants to receive messages from multiple senders needs to store only one single secret key. When using symmetric encryption schemes, a user has to store one shared key for each sender.

The main disadvantage of asymmetric encryption schemes compared to symmetric ones is that they are less efficient—approximately 2 to 3 orders of magnitude slower than symmetric encryption schemes[KL07].

Next, we will present two variants of asymmetric encryption schemes: homomorphic encryption schemes and attribute-based encryption schemes. We will use these schemes later in Chapter 6 to build our proposed approaches for MSNs, which offer secrecy guarantees to the users.

3.1.3 Homomorphic Encryption

Homomorphic encryption schemes are asymmetric encryption schemes that allow performing specific types of operations on the encrypted data. Given an operation $\cdot$ and two ciphertexts, c_1 and c_2, generated using homomorphic encryption, a homomorphic encryption scheme provides a homormorphic operation $\otimes$ such that when applied $\otimes$ on c_1 and c_2, the result of such operation generates a new ciphertext c'. The decryption of the ciphertext c' results in a plaintext that matches the output of applying the given operation $\cdot$ on the decrypted ciphertexts c_1 and c_2.

These schemes support various types of operations on the encrypted data such as addition and multiplication. We can classify homomorphic encryption schemes by the number and types of operations that they support. There are three main groups: partially homomorphic encryption, somewhat homomorphic encryption, and fully homomorphic encryption.

Partially homomorphic encryption schemes (PHE) support only one type of operation that can be applied any number of times. That is, given as input two ciphertexts, c_1 and c_2, a PHE scheme computes the supported operation on c_1 and c_2, and outputs a ciphertext c_3. One can use c_3, together with another ciphertext, as input to the PHE scheme to compute the supported operation again. One can compute the supported operation any number of times.

Somewhat homomorphic encryption schemes support more than one operation but a limited number of times. The reason is that the resulting ciphertext of a homomorphic operation contains random noise. This noise grows with the number of the subsequent homomorphic operations applied to the resulting ciphertext. One can decrypt a ciphertext correctly only if its noise is lower than a threshold value determined by the scheme used.

Fully homomorphic encryption schemes support any number of multiplication and addition operations, and thus they can handle any function that can be represented by addition and multiplication.

In this thesis, we will use somewhat homomorphic encryption schemes. As we will explain in Chapter 6, in our scenario, we need to perform a limited number of addition and multiplication operations on the encrypted data. Furthermore, these schemes are more efficient than fully homomorphic schemes [AAUC18]. Formally:

Definition 3.3: Somewhat Homomorphic Encyption Scheme

A **somewhat homomorphic encryption scheme** SHE is an asymmetric encryption scheme in which the message space is a ring $(R, +, \cdot)$ and the ci-

phertext space is also a ring $(R, \oplus, \otimes)$ such that for all messages $m1, m2 \in R$, and all pair of keys $(\mathsf{pk}, \mathsf{sk})$, $m1 + m2 = \mathsf{Dec}\,(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m1) \oplus \mathsf{Enc}(\mathsf{pk}, m2))$, and $m1 \cdot m2 = \mathsf{Dec}\,(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m1) \otimes \mathsf{Enc}(\mathsf{pk}, m2))$. A SHE can compute the supported operations a limited number of times.

3.1.4 Attribute-based Encryption

Attribute-based encryption schemes allow a user to encrypt data and generate a single ciphertext that can be later decrypted by multiple receivers without needing to share a secret key among them. That is, each receiver can decrypt a single ciphertext using his secret key. A key authority, which is a trusted entity, generates the secret keys of the users. Traditional symmetric and asymmetric encryption schemes do not support this feature of attribute-based encryption. With traditional symmetric encryption schemes, to be able to decrypt a given ciphertext, the senders and each of the receivers have to share—in advance—the encryption/decryption key. With traditional asymmetric encryption schemes, each user has a secret key; however, a user can decrypt only ciphertexts generated using his public key. Then, if a user wants to send an encrypted message to multiple receivers, the user has to generate multiple ciphertexts using for each ciphertext, the public key of each receiver.

Attribute-based encryption is a type of asymmetric encryption in which the secret key of a user and the ciphertext are generated based on a set of tuples of the form $(attribute = value)$, where *attribute* is a user attribute, e.g., role, and *value* refers to a specific value of such an attribute, e.g., student. We call a tuple of the form $(attribute = value)$, attribute-value pair. A user can decrypt a ciphertext c only if the set of attribute-value pairs used to generate his secret key matches the set of attribute-value pairs used to generate c. In what follows, we explain in depth the encryption/decryption process.

There are two types of attribute-based encryption schemes: ciphertext-policy attribute-based encryption scheme (CP-ABE) and key-policy attribute-based encryption scheme (KP-ABE). The encryption/decryption process is different for both schemes.

Ciphertext-Policy Attribute-based Encryption

CP-ABE schemes use an access policy, which consists of constraints on attribute-value pairs, to encrypt a given message. To generate the secret key of a user, these schemes use a set of attribute-value pairs. A user can decrypt a ciphertext

c, if and only if the set of attribute-value pairs used to generate his secret key satisfies the access policy used to generate c. Figure 3.1 illustrates a CP-ABE scheme. The encrypted file is generated based on the access policy defined by the file owner: (role=Student Council President) $\vee$ (role=Undergraduate Student) $\wedge$ (major= Computer Science). The key authority generates the secret key of each user based on a set of attribute-value pairs associated with the user. Only "User 1" can decrypt the encrypted version of the file because the set of attribute-value pairs {(role = Student Council President) , (major=Statistics)} used to generated his secret key satisfies the access policy (role = Student Council President) $\vee$ (role = Undergraduate Student) $\wedge$ (major = Computer Science).

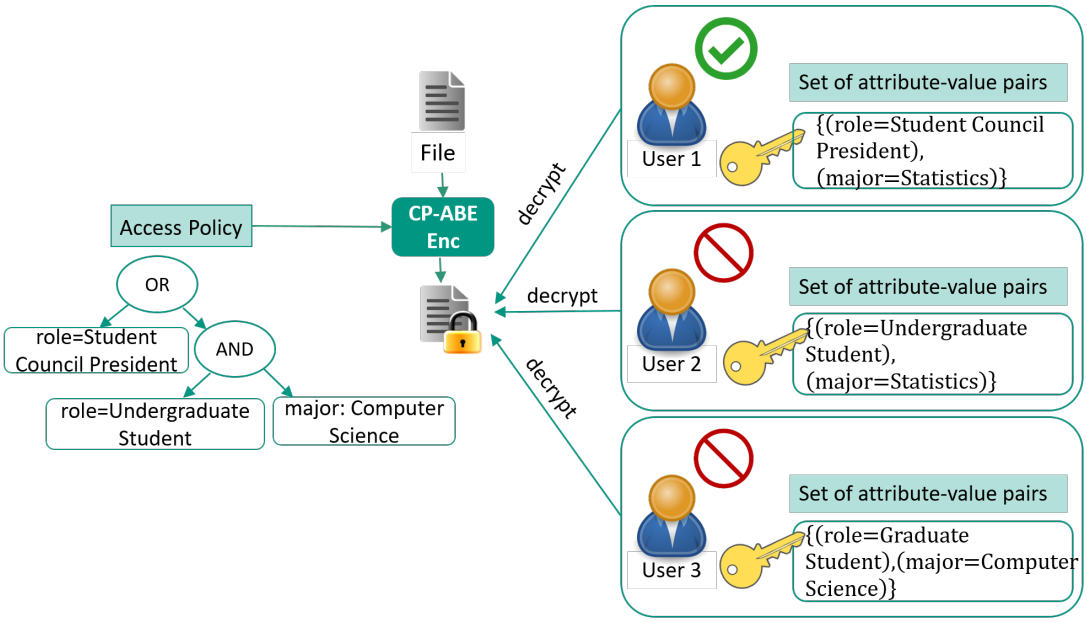


Figure 3.1: Ciphertext-policy attribute-based encryption scheme (CP-ABE)

Key-Policy Attribute-based Encryption

KP-ABE is the counterpart of CP-ABE in the sense that: KP-ABE schemes use an access policy to generate the secret key of a user, and a set of attribute-value pairs to generate the ciphertext of a given message. A user can decrypt a ciphertext c if the set of attribute-value pairs used to generate c satisfies the access policy used to generated his secret key. Then, the main difference between KP-ABE and CP-ABE is that using CP-ABE, a user who encrypts a message can specify who can decrypt it through the access policy used to generate the ciphertext [BSW07].

With KP-ABE, a user who encrypts a message does not have control to decide who can decrypt it, except by using a set of attribute-value pairs to generate the ciphertext. He has to trust the key authority to issue the secret keys to the users based on their attributes adequately to allow or deny them access to the ciphertext. Figure 3.2 illustrates a KP-ABE scheme. The secret key of the user is generated by the key authority based on the access policy (role = Student Council President) $\vee$ (role = Undergraduate Student) $\wedge$ (major = Computer Science). Each ciphertext is generated based on a set of attribute-value pairs defined by the owner of the file. The user can decrypt the encrypted version of "File 1" because the set of attribute-value pairs used to generated the corresponding ciphertext {(role=Student Council President), (major=Statistics)} satisfies the access policy (role=Student Council President) $\vee$ (role=Undergraduate Student) $\wedge$ (major = Computer Science). However, he cannot decrypt the encrypted "File 2" and "File 3".

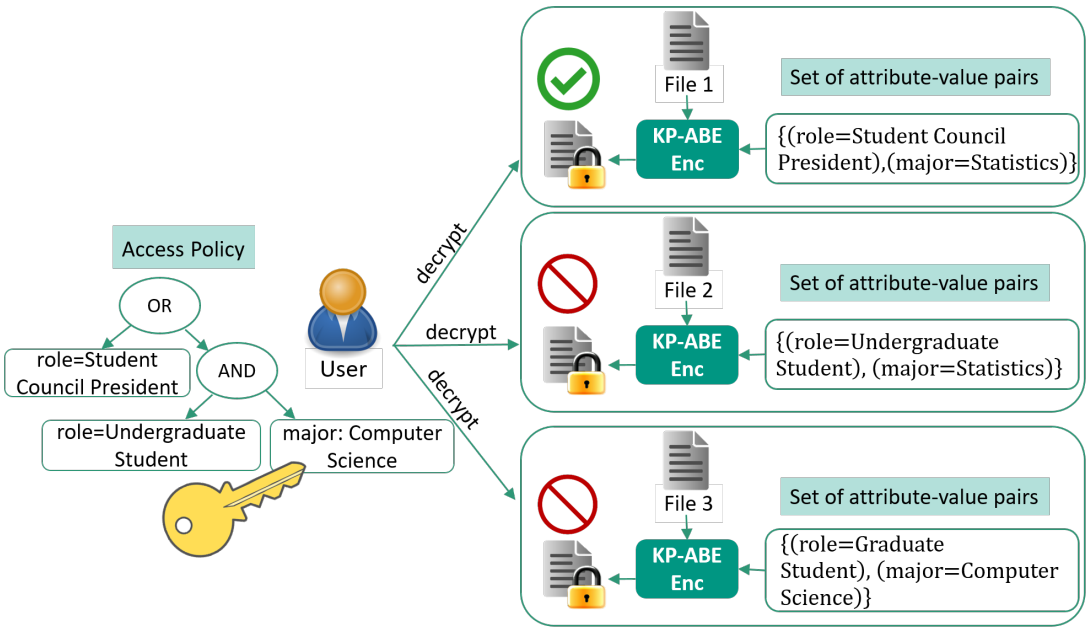


Figure 3.2: Key-policy attribute-based encryption scheme (KP-ABE)

In our thesis, we will use CP-ABE in Chapter 6. Next, we formalize this encryption scheme. Let γ and c_γ be an access policy and a ciphertext generated based on γ, respectively. Further, let ω_u and sk_{ω_u} be a set of attribute-value pairs associated to a user u and the secret key of user u generated based on ω_u, respectively. Next, $\mathsf{Enc}(\mathsf{pk}, m_\gamma)$ denotes the operation of encrypting m under the access policy γ and

public key pk, and $\mathsf{Dec}((\mathsf{pk}, \mathsf{sk}_{\omega_u}), c_\gamma)$ denotes the operation of decrypting c_γ under the pair of public and secret keys $(\mathsf{pk}, \mathsf{sk}_{\omega_u})$.

Definition 3.4: Ciphertext-Policy Attribute-Based Encryption Scheme

A **CP-ABE scheme** consists of four algorithms $(\mathtt{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$: (1) A setup algorithm, $\mathtt{Setup}$, that selects two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ and use them to generate a public key pk and a master key $\mathtt{mk}$. The setup algorithm outputs the pair of keys $(\mathsf{pk}, \mathtt{mk})$. (2) A key generation algorithm KGen that takes as input a set of attribute-value pairs ω_u associated with a user u and the master key $\mathtt{mk}$, to return a secret key sk_{ω_u}. (3) A probabilistic encryption algorithm Enc that takes as input a message m, an access policy γ and the public key pk to return a ciphertext c_γ. (4) A deterministic decryption algorithm Dec that takes as input the pair of public and secret keys $(\mathsf{pk}, \mathsf{sk}_{\omega_u})$ and a ciphertext c_γ to return a message m or an error message $\perp$. If the set of attribute-value pairs ω_u satisfies the access policy γ, $\mathsf{Dec}((\mathsf{pk}, \mathsf{sk}_{\omega_u}), \mathsf{Enc}(\mathsf{pk}, m_\gamma)) = m$; otherwise $\mathsf{Dec}((\mathsf{pk}, \mathsf{sk}_{\omega_u}), \mathsf{Enc}(\mathsf{pk}, m_\gamma)) = \perp$.

3.2 Secure Cryptographic Schemes

Several cryptographic schemes that aim to provide secrecy have been proposed in the literature. In this Section, we address the question: "What is a secure cryptographic scheme?" In this dissertation, we use the term "secure" to indicate that a scheme fulfills a given secrecy definition, as we will explain in what follows.

Deeming a cryptographic scheme as *secure* requires a secrecy proof, which itself requires a formal secrecy definition. Next, we explain what do these concepts, secrecy proof and secrecy definition, mean.

3.2.1 Formal secrecy definitions

Formal secrecy definitions define what is meant with secrecy. If a cryptographic scheme fulfills a given secrecy definition, one can say that the scheme is secure with respect to the given definition. A formal secrecy definition has two components [KL07]: secrecy guarantees and adversary model.

Secrecy Guarantees

The secrecy guarantees define the threats that a cryptographic scheme must protect against if it fulfills the given secrecy definition. That is, a secure cryptographic scheme has to prevent adversaries from performing the actions established by the secrecy guarantees. From the point of an adversary, the secrecy guarantees establish what a successful attack on the scheme is.

Adversary Model

The adversary model describes the abilities and limitations that an adversary is assumed to have. The notion of a secure cryptographic scheme is meaningful only with respect to a particular adversary model.

Typically an adversary model includes the following components:

- **The adversarial behavior strategy:** It defines the actions that the adversary might take regarding the protocol specification. With protocol specification, we mean the process that describes how a given cryptographic scheme should be used to try to achieve certain secrecy guarantees. There are two main adversarial behavior strategies: semi-honest and malicious. Adversaries that follows the first and the second strategy are known as semi-honest adversaries (or honest-but-curious adversaries), and malicious adversaries, respectively. Semi-honest adversaries follow the protocol specification, but they try to learn information from the messages (or ciphertexts) that they receive during the execution of the protocol. Malicious adversaries may deviate arbitrarily from the protocol specification.
- **Computational strategy:** It defines the computational complexity adversaries are assumed to have. In cryptography, adversaries commonly have polynomial capabilities, i.e., they run probabilistic polynomial-time algorithms, or unbound capabilities, i.e., they do not have computational limits.
- **Protocol execution strategy:** It specifies the number of times adversaries are allowed to execute the protocol. Common protocol execution strategies are [Can01]: stand-alone and concurrent-composition. In the stand-alone strategy, the adversary is allowed to execute the protocol a single time. In the concurrent-composition strategy, the adversary can execute the protocol several times.
- **Collusion strategy:** The collusion strategy is relevant only in multiparty protocols. It defines when or how the entities participating in a protocol specification come under the control of the adversary. One has to specify

the entities that may behave adversarially and the entities that may collude with each other during the attack. There exist two main collusion strategies: static and adaptive [HL10]. In the static strategy, the adversary is given a set of entities to collude with, and the honest entities remain honest during the protocol execution. In the adaptive strategy, the adversary can collude with any entity in the system during the protocol execution.

- **Further assumptions:** They state any other assumption, which has not been covered by the above components. One can specify assumptions regarding the environment and resources that adversaries might use. For instance, one can assume that the secret keys are sent through a secure communication channel. One can also specify the assumed knowledge of the adversaries, also known as accepted information leakage. For instance, a cryptographic scheme may accept to leak certain information such as the number of tuples of a relational table or the number of users registered in a system.

Note that an adversary model does not specify the strategies of an attack because it is impossible, in real scenarios, to predict them.

3.2.2 Secrecy Proofs

To prove that a given cryptographic scheme is secure, one needs a formal secrecy definition, which includes the secrecy guarantees and the ability that the adversary is assumed to have. Cryptographic schemes are constructed based on a number of well-known cryptographic problems, which themselves rely on open problems in complexity theory like $\mathsf{P} \neq \mathsf{NP}$. A secrecy proof is a reduction that shows that if an adversary, with the abilities established in the adversary model, breaks the defined secrecy guarantees of a given scheme, which implies that the adversary can solve the underlying complexity problem of the scheme.

In the next subsection, Subsection 3.3, we describe standard secrecy definitions for encryption schemes. In Chapters 5 and 6, we will use these notions to build and prove the secrecy of our proposed schemes.

3.3 Standard Secrecy Definitions for Encryption Schemes

In this section, we give an overview of standard secrecy guarantees, Section 3.3.1, and adversary models for encryption schemes, Section 3.3.2.

3.3.1 Standard Secrecy Guarantees for Encryption Schemes

The standard well-known secrecy guarantees for encryption schemes are semantic secrecy, indistinguishability, and non-malleability. Next, we explain these notions.

- **Semantic Secrecy (SS):** Semantic secrecy states that given a ciphertext, an adversary cannot learn any information about the underlying plaintext.
- **Indistinguishability (IND):** The indistinguishability notion states that given a ciphertext c corresponding to one of two plaintexts known by an adversary, the adversary cannot distinguish which of the plaintext was encrypted by observing c.
- **Non-malleability (NM):** The non-malleability notions states that given a ciphertext c, an adversary is unable to output another ciphertext c' such that the underlying plaintext of c' is meaningful related to the underlying plaintext of c.

Note that in the three secrecy guarantees explained above, the adversary is given a ciphertext c. We refer to c as the "challenge ciphertext".

The next subsection presents standard adversary models of encryption schemes, which specify the abilities of the adversaries.

3.3.2 Standard Adversary Models for Encryption Schemes

Standard adversary models, sorted by increasing power of the adversary, used to prove that a given encryption scheme is secure are:

- **Ciphertext-only attacks (COA):** In this attack, the adversary can observe ciphertexts but does not know the associated plaintext.
- **Known-plaintext attacks (KPA):** In this attack, the adversary can observe pairs of plaintexts and their corresponding ciphertexts.
- **Chosen-plaintext attacks (CPA):** In this attack, the adversary has access to an encryption oracle. The adversary can use the encryption oracle as long as the adversary has not received the challenge ciphertext. That is, the adversary can select some plaintexts and obtain their corresponding ciphertexts. Once the adversary receives the challenge ciphertext, she cannot encrypt any other plaintext.
- **Chosen-ciphertext attacks (CCA):** In this attack, the adversary has access to an encryption oracle and a decryption oracle. The adversary can

use both oracles as long as the adversary has not received the challenge ciphertext. That is, the adversary can select some plaintexts and obtain their corresponding ciphertexts, and some ciphertexts and obtain their corresponding plaintexts. Once the adversary receives the challenge ciphertext, she cannot encrypt any other plaintext or decrypt any other ciphertext.

These standard adversary models consider adversaries who run probabilistic polynomial-time (PPT) algorithms and who can break the secrecy guarantees of a scheme with some negligible probability. Intuitively, adversaries are required to run in a "reasonable" amount of time, and their probability of breaking the secrecy guarantees of a scheme are "very small" as to be zero for all practical purposes. We will formalize these two notions, probabilistic polynomial-time and negligible probabilities, in the next subsection, Subsection 3.3.3. We use the term "success probability" to refer to the probability that an adversary breaks the secrecy guarantees of a scheme.

To construct a secrecy definition, one can combine one of the secrecy guarantees presented in Subsection 3.3.1 with one of the adversary models. Such a combination allows building several secrecy definitions such as SS-CPA, which combines semantic secrecy with chosen-plaintext attacks, or IND-CPA, which combines the indistinguishability notion with chose-plaintext attacks. The authors in [BDPR98] and [WSI03] have studied and proven the relationships existing between the different secrecy definitions. For instance, in the CPA model, semantic secrecy is equivalent to the indistinguishability notion [BDPR98]. In this thesis, we will adopt the indistinguishability notion.

As mentioned above, adversaries are assumed to run PPT algorithms and we allow them for some small success probability. A secrecy definition that considers adversaries with unbounded computational power and allow adversaries for a success probability of zero is known as *perfect secrecy*. To achieve perfect secrecy, a given encryption scheme has to use an encryption key as long as the plaintext, and each encryption key can be used only once [KL07], which make perfect secrecy impractical. Therefore, relaxing the computational power of the adversaries and allowing them for some small success probability allow building encryption schemes for all practical purposes [KMVOV96].

Next, we select IND-CPA to showcase how these secrecy definitions look like. In the next subsection, we formally define IND-CPA.

3.3.3 Indistinguishability under chosen-plaintext attacks

As mentioned in the previous subsection, we consider adversaries who run probabilistic-polynomial time (PPT) algorithms and can break the secrecy guarantees of a given encryption scheme with some negligible probability. We use the term PPT adversaries to denote adversaries that run probabilistic polynomial-time algorithms. Before defining the indistinguishability secrecy notion under chosen-plaintext attacks, we formalize the notions of PPT algorithms and negligible probability.

Probabilistic-Polynomial Time Algorithms and Negligible Probabilities

We first introduce the notion of *security parameter* [Dam88]. Both the computational power of the adversaries and their success probability are expressed in terms of the security parameter.

Definition 3.5: Security Parameter

Given an encryption scheme Π, the **security parameter** of Π, denoted as n, is an integer variable that measures the input size of the computational problem on which Π is based.

The security parameter allows honest parties to adjust the security level of a given encryption scheme. Increasing the security parameter implies higher secrecy, but it affects the efficiency of the scheme negatively. That is, there is a trade-off between secrecy and performance.

Definition 3.6: Probabilistic Polynomial-Time Algorithm

A **probabilistic polynomial-time algorithm**, PPT algorithm, is an algorithm running in polynomial time such that for some polynomial poly and some input $n \in \{0,1\}^*$, the algorithm always halts after $|\mathsf{poly}(n)|$ steps independently of the outcome of its internal coin tosses.

Definition 3.7: Negligible Function

A **function f of type $\mathbb{N} \to \mathbb{R}_0^+$ is negligible** iff $\forall c \in \mathbb{N} : \exists n_0 \in \mathbb{N}$ such that for all integers $n \geq n_0, f(n) < n^{-c}$. We use negl to denote an arbitrary negligible function.

Definition 3.8: Negligible Probability

Given a negligible function negl, a **negligible probability** is a probability with density function behaving as negl.

IND-CPA Secrecy Notion

The concept of indistinguishability is defined based on an indistinguishability experiment between a PPT adversary $\mathcal{A}$ and a challenger. The indistinguishability experiment under chosen-plaintext attacks (IND-CPA) is the following: The challenger generates a random key k. The challenger gives access to $\mathcal{A}$ to an encryption oracle. That is, the adversary can send plaintexts and obtain their corresponding ciphertexts generated using the key k. At the end of the experiment, $\mathcal{A}$ sends to the challenger two messages m_0 and m_1, where m_0 and m_1 are different to the messages sent to the encryption oracle. The challenger selects uniformly at random one of these two messages and encrypts it using the key k. The challenger sends the encrypted message, the challenger ciphertext, to $\mathcal{A}$. $\mathcal{A}$ outputs a *guess* to indicate which of the two messages corresponds to the challenger ciphertext. The indistinguishability concept states that an encryption scheme is secure if no PPT adversary $\mathcal{A}$ can successfully guess which message was encrypted with a probability significantly better than $\frac{1}{2}$, i.e., random guessing.

Next, we formalize the indistinguishability notion. Given a message m, let $|m|$ be the length of message m. We use $\leftarrow_{\$}$ to denote a randomize procedure.

Definition 3.9: The Adversarial Indistinguishability Experiment IND-CPA

Given a symmetric encryption scheme $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, an adversary $\mathcal{A}$ and a security parameter n, the **adversarial indistinguishability experiment** under chosen-plaintexts attacks $\text{IND-CPA}_{\mathcal{A},\Pi}(n)$ is:

$\text{IND-CPA}_{\mathcal{A},\Pi}(n)$

1 : $\mathsf{k} \leftarrow \mathsf{KGen}(1^n)$

..... Setup Completed.....

2 : **for** $i = 1, \ldots, q$
3 : $\quad m_i \leftarrow \mathcal{A}(1^n)$
4 : $\quad c_i = \mathsf{Enc}(\mathsf{k}, m_i)$
5 : $(m_0, m_1) \leftarrow_{\$} \mathcal{A}(1^n)$
6 : $b \leftarrow_{\$} \{0,1\}$
7 : $c_b = \mathsf{Enc}(\mathsf{k}, m_b)$
8 : $\overline{b} \leftarrow_{\$} \mathcal{A}(1^n, c_b)$
9 : **return** 1 **if** $b = \overline{b}$ **else** 0

Figure 3.3 illustrates the adversarial indistinguishability experiment IND-CPA.

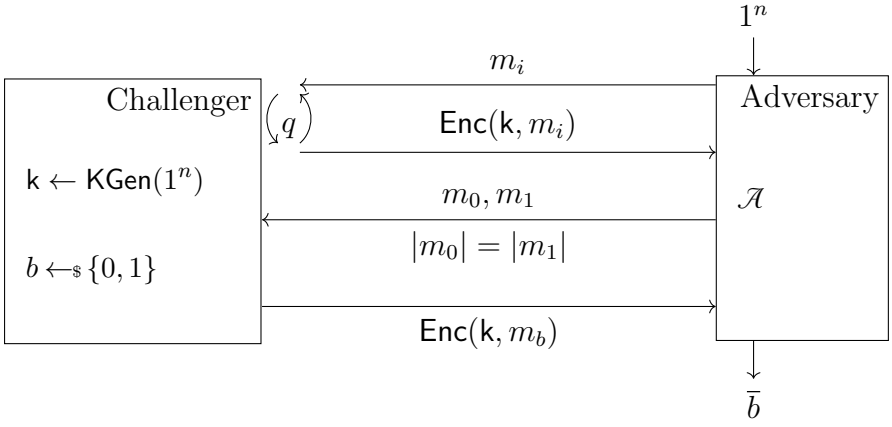


Figure 3.3: Indistinguishability experiment $\text{IND-CPA}_{\mathcal{A},\Pi}(n)$

Given a PPT adversary $\mathcal{A}$ and a symmetric encryption scheme Π, let $\mathsf{Adv}_{\mathcal{A},\Pi}(n)$ denote the advantage of the adversary, which measures the success probability of the adversary.

Definition 3.10: An IND-CPA Secure Symmetric Encryption Scheme

A symmetric encryption scheme $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is **indistinguishability secure** under chosen-plaintexts attacks, IND-CPA secure, if for all

PPT adversaries $\mathcal{A}$:

$$\mathsf{Adv}_{\mathcal{A},\Pi}(n) = |\Pr[\text{IND-CPA}_{\mathcal{A},\Pi}(n) = 1] - \Pr[\text{IND-CPA}_{\mathcal{A},\Pi}(n) = 0]|$$

is negligible.

4 Integrating Reciprocity into Access Control Models

A crucial requirement of information systems, including OSNs, is to guarantee data secrecy by protecting the information stored in these systems from unauthorized access. Access control systems play an important role to guarantee data secrecy. In this chapter, we focus on access control models, which are one of the components of access control systems.

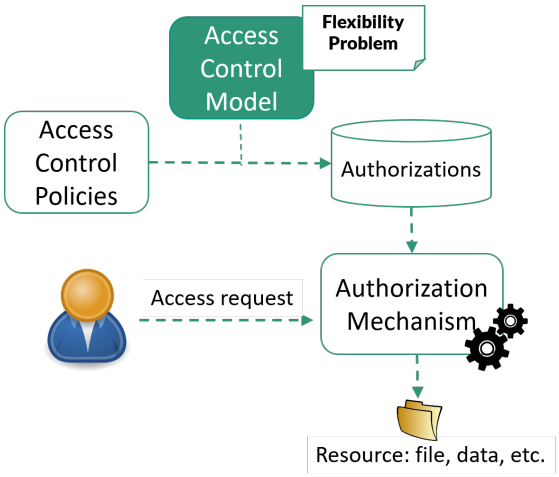


Figure 4.1: Access Control Model - Flexibility Problem

Specifically, we increase the flexibility of existing access control models by defining the syntax and semantics of a new type of authorization called *mutual*[2]. *Mutual* authorizations allow modeling reciprocity. Studies in different areas, such as psychology [HMS98], economics [FF06, SBC09], and sociology [FFG02], have revealed that reciprocity is a fundamental aspect that shapes human behavior. Reciprocity indicates that, people tend to be kind to someone that has been kind to them. Integrating reciprocity into access control means letting persons grant access to their resources to users that allow them the same.

[2]Parts of this chapter have been published in [SEGB19b] and in an extended version in [SEGB19c]

Example 4.1. Think of a car sharing system. Alice, a user of the system, is more like to allow others using her car if she gets something in return. That is, Alice wants to allow a user Bob using her car, if Bob allows her using his.

Various access control models have been proposed [SCFY96, OP03, TLT15]. However, all existing models consider only two type of grants: allow and deny. These two grants are not enough to support reciprocity explicitly. Consider Example 4.1, to support such reciprocal sharing, with traditional access control models, Alice can verify whether Bob has allowed her using his car and then set her access privileges for him accordingly. Such a solution, however has secrecy issues and it is not scalable.

Example 4.2. Think again of a car sharing system and consider that 10000 car owners, including Alice, use the system. Alice would like to allow any other user of the system to use her car if and only if the user allows her to use his car. With traditional access control models, Alice would have to verify for each user, whether the user has allowed her using his car and then set her access privileges for him accordingly. That is, she needs to repeat this process 9999 times. Moreover, this process requires privileges to be public, which is unrealistic because they are sensitive information. Alice would also have to beware of changes in the access policies of others continuously, e.g., Alice would have to revoke the access to any user u as soon as u revokes Alice access to his car. With the reciprocity feature envisioned in turn, all that Alice has to do is to specify one authorization.

To capture this reciprocity phenomenon, we need a new kind of grant. We call it *mutual* and authorizations that make use of it *mutual authorizations*. Contrary to positive authorizations, i.e., authorizations that use the grant *allow*, *mutual* authorizations involve an exchange of resources between two users.

In this chapter, we define the syntax and semantics of *mutual* authorizations (Section 4.1). One can add these new authorizations to any existing access control model. To study and illustrate how this addition can look like conceptually, we select ABAC, one of the most general access control models. To focus on the idea of reciprocity itself, we first assume that all resources have the same sensitivity degree, e.g., physical positions.

Next, in Section 4.2, we generalize our model to support settings where resources have different sensitivity degrees, e.g., health records. A patient who has a stigmatizing disease may consider that his health record has a higher sensitivity degree than the health record of a patient who is in perfect health. However, whether a

disease is stigmatizing or not is a subjective issue. It is not obvious how to guarantee a fair exchange of resources when using *mutual* authorizations, especially in such general settings where resources have different sensitivity degrees. Our general model deals with these problems.

As a use case for the deployment of *mutual* authorizations, we use location-based services (LBS). This integration of *mutual* authorizations into existing services— LBS in our example— has to reuse existing implementations, be efficient, and guarantee that only authorized users can access the services. Section 4.3 presents two approaches to show how one can achieve this integration.

We conduct experiments to evaluate the performance of an access decision when *mutual* authorizations are involved and that of our proposed approaches, which integrate *mutual* authorizations into LBS. Section 4.4 presents the results of such experiments. Section 4.5 gives an overview of related work in the area.

4.1 Mutual Authorizations: Syntax and Semantics

In this section, we introduce first the syntax of *mutual* authorizations and then their semantics. As mentioned in the introduction of this chapter, to focus on the idea of reciprocity itself, in this section, we assume that all resources have the same sensitivity degree.

4.1.1 Syntax

An authorization, as specified in Definition 2.1, consists of five elements: a set of subjects, a set of resources, an operation, a grant, and a set of environment conditions.

Given an authorization A, we use the following notation to refer to each of the elements of an authorization: $subject(A)$, $res(A)$, $op(A)$, and $grant(A)$ denote, respectively, the subjects induced by $Cons_{subject}$ of A, the resources induced by $Cons_{resource}$ of A, the operation op of A, and the grant gr of A. For simplicity, we do not specify the environment constraints involved in an authorization. That is, the set of environment attributes is an empty set denoted as $\emptyset$.

We assume that resources always have an attribute *owner*, i.e., $owner \in Res_{Att}$. The resource owners are the responsible for creating authorizations to control the operations that a set of subjects is allowed to perform over their resources. For brevity, we omit the attribute *owner* from the set of resource attributes, and given an authorization A assigned by a user u, we do not explicitly write the resource

constraint $owner = u$, but we assume it to be present. Next, we use $user(A)$ and U to denote the user who has specified A and the set of all users, respectively.

Note that from the point of view of an authorization, we differentiate two types of persons in the system: users and subjects. Given an authorization A, we use the terms **user** and **subject** to refer, respectively, to the person who has assigned A, and the person who has received A.

Table 4.1 summarizes the notation used to refer to the elements of a given authorization A.

Notation	Description
$user(A)$	The user who specified A
$subject(A)$	The set of subjects that meet the subject constraint of A
$res(A)$	The set of resources that meet the resource constraint of A
$op(A)$	The operation op of A, where $op \in Op$
$grant(A)$	The grant gr of A, where $gr \in Gr$

Table 4.1: Elements of an Authorization A

Existing access control models are based on the grants $Gr = \{deny, allow\}$. We extend Gr with a new kind of grant, which we call *mutual*. Mutual grants capture the reciprocity phenomenon by means of *mutual authorizations*. Given an authorization A, A is a *mutual* authorization if $grant(A) = mutual$.

4.1.2 Semantics

The semantics of all authorization can be reduced to questions of the form: "Can a person s perform the operation op on the resource res_u owned by u?". We call this an *access request*.

Definition 4.1: Access Request

An **access request** $Req = \langle s,\ op, res_u \rangle$ is a tuple consisting of a person s, an operation $op \in Op$ and a resource owned by a person u, res_u. An access request indicates that s requests to perform the operation op on the resource res_u owned by u.

In the context of positive and negative authorizations, answering an access request is straightforward. A positive authorization A uses the grant *allow* and states that

$user(A)$ authorizes $subjects(A)$ to invoke $op(A)$ on $res(A)$. A negative authorization uses *deny* and states that $user(A)$ forbids $subjects(A)$ to invoke $op(A)$ on $res(A)$. However, in the context of *mutual* authorizations, to answer an access request, one must consider the authorizations that u has assigned to s and the ones that u has received from s.

Given two authorizations A and B, we use $res(A) = res(B)$ to indicate that the resources in both authorizations are of the same type.

Definition 4.2: Mutual Authorizations: Semantics

Given a *mutual* authorization A, A states that $user(A)$ allows invoking $op(A)$ on $res(A)$ to the subjects in $subjects(A)$ who have issued an authorization B to $user(A)$, if the following boolean expression evaluates to true: $(res(B) = res(A)) \wedge ((grant(B) = allow) \vee (grant(B) = mutual)) \wedge (op(B) = op(A))$.

For simplicity and to ease the presentation, in the remainder of this chapter, we restrict the elements of an authorization, Definition 2.1, as follows: (1) The set of attribute for persons is $Attr_{Person} = \{role, name\}$. Given a subject s, $r(s)$ is the set of roles of s. The set of subjects that receive a given authorization A is $subjects(A) = \{s \in S \mid name{=}s \vee (\exists r : role{=}r \wedge r \in r(s))\}$. (2) The set of operations is $Op = \{read\}$. (3) The set of grants is $Gr = \{allow, mutual, deny\}$. (4) We do not specify the environment conditions when expressing an authorization, i.e, we omit the environment specification from Definition 2.1. Furthermore, as mentioned at the beginning of this section, we assume that all resources have the same sensitivity degree, unless stated otherwise. Specifically, we consider a setting with physical positions as the only type of resource. That is, the resources specified in the authorizations are the physical positions of the users. We assume that each user $u \in U$ has one physical position, p_u.

So far our semantics of *mutual* authorizations, Definition 4.2, assumes that there exists a single authorizations that involves a given user u and a given subject s. However, in real scenarios, a subject could receive more than one authorization from the same user over the same resource. In such cases, authorization conflicts may arise. In the next subsection, Subsection 4.1.3, we formally define authorization conflicts and explain our strategy to solve them.

4.1.3 Authorization Conflicts Resolution

Let $\mathcal{A}$ denote the set of all authorizations.

Definition 4.3: Authorization Conflicts

Given a set of authorizations $A_C \subseteq \mathcal{A}$, a subject $s \in S$ and a user $u \in U$, an **authorization conflict** exists with respect to u and s if s has received more than one authorization on the same resource with different grants assigned by u. Formally, an authorization conflict exists with respect to u and s if $\exists A, B \in A_C : (user(A) = user(B) = u) \wedge (s \in subjects(A) \cap subjects(B)) \wedge (res(A) = res(B)) \wedge (grant(A) \neq grant(B))$.

Example 4.3. Consider a user Alice who has two roles $r(Alice) = \{r1, r2\}$. Assume now that a user Bob with physical position p_{Bob} has assigned the following two authorizations: $A = \langle role{=}r1, p_{Bob}, read, mutual\rangle$ and $B = \langle role = r2, p_{Bob}, read, deny\rangle$. Authorizations A and B are in conflict with respect to Alice and Bob. Namely, A assigns a *mutual* grant to Alice while B assigns her a *deny* grant for reading the physical position of Bob, p_{Bob}.

Solving authorization conflicts requires deciding which authorization prevails over the others when in conflict. There exists several conflict resolutions strategies in the literature that consider positive and negative authorizations. For instance, the *recency-overrides* strategy states that authorizations specified later take precedence over earlier ones. We select a conflict resolution strategy in which authorizations are assigned a precedence based on their grants. We call it *deny-mutual* precedence strategy.

Definition 4.4: Deny-Mutual Precedence Strategy

A *deny-mutual precedence strategy* is a prioritization of the grants in Gr. It states that an authorization with the grant *deny* precedes an authorization with the grant *mutual* and an authorization with the grant *mutual* precedes an authorization with the grant *allow*. We refer to this precedence as $deny \gg mutual \gg allow$.

We select the precedence $deny \gg mutual \gg allow$ because assigning a higher precedence to a negative authorization eliminates the risk of possible leakage [HA11]. Given two authorizations A and B, we write $A \gg B$ to denote that $grant(A) \gg grant(B)$. Next, we interpret an operation not granted explicitly as denied.

Given a set of authorizations $\mathcal{B}$ and a subject s, intuitively, our conflict resolution process is as follows:

1. Select all authorizations $A \in \mathcal{B}$ where $s \in subjects(A)$.

2. Group the authorizations selected in the previous step by the user who has assigned them. Each group contains authorizations where the user who has assigned them is the same. There are no two or more sets containing authorizations assigned by the same user. We call the function that does this grouping *authorization-grouping function*.

3. For each group of authorizations, select the authorization with the highest precedence based on the *deny-mutual* precedence strategy.

Definitions 4.5 and 4.6 formalizes the conflict resolution process.

Definition 4.5: Authorization-Grouping Function

An **authorization-grouping function** $group : \mathcal{P}(\mathcal{A}) \times S \rightarrow \mathcal{P}(\mathcal{P}(\mathcal{A}))$ takes as input a set of authorizations $\mathcal{B} \subseteq \mathcal{A}$ and a subject $s \in S$ and outputs a set $\mathcal{C}$ of sets of authorizations such that:

1. $\bigcup_{\mathcal{D} \in \mathcal{C}} \mathcal{D} = \{A \in \mathcal{B} \mid s \in subjects(A)\}$.
2. $\forall \mathcal{D}_1, \mathcal{D}_2 \in \mathcal{C} : \mathcal{D}_1 \neq \mathcal{D}_2 \Rightarrow \mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$.
3. $\forall \mathcal{D} \in \mathcal{C}, \forall A, B \in \mathcal{D} : user(A) = user(B) \wedge s \in subjects(A) \cap subjects(B)$.
4. $\forall \mathcal{D}_1, \mathcal{D}_2 \in \mathcal{C}, \forall A \in \mathcal{D}_1, \forall B \in \mathcal{D}_2 : \mathcal{D}_1 \neq \mathcal{D}_2 \Rightarrow user(A) \neq user(B)$.

Example 4.4. Consider a user s with roles $r(s) = \{r_1, r_2\}$ the following set of authorizations:

$$\mathcal{B} = \left\{ \begin{array}{l} A = \langle role{=}r1, p_{Bob}, read, mutual \rangle \\ B = \langle role = r2, p_{Bob}, read, deny \rangle \\ C = \langle name = Alice, p_{Carol}, read, allow \rangle \\ D = \langle name = Carol, p_{Bob}, read, mutual \rangle \\ E = \langle role = r1, p_{Carol}, read, mutual \rangle \end{array} \right\}$$

Given as input the set $\mathcal{B}$ and subject Alice, the output of the authorization-grouping function $group(\mathcal{B}, Alice)$ is the set $\mathcal{C} = \{\{A, B\}, \{C, E\}\}$. $\mathcal{C}$ contains all authorizations with the subject Alice, and each subset in $\mathcal{C}$ contains authorizations assigned by the same user.

Let a *user-grant* tuple be a tuple of the form $\langle t_{user}, t_{grant} \rangle$, where t_{user} is a user in U and t_{grant} is a grant in Gr. We note that our resolve-conflicts function does not

consider the resources involved in the authorizations in conflict because we restrict our study to a specific resource, the physical positions of users.

Definition 4.6: Resolve-Conflicts Function

A **resolve-conflicts function** — $resC : \mathcal{P}(\mathcal{A}) \times S \rightarrow \mathcal{P}(U \times Gr)$ is a function that takes as input a set of authorizations $\mathcal{B} \subseteq \mathcal{A}$ and a subject $s \in S$ and outputs a set $\mathcal{C}$ of *user-grant* tuples. For each set of authorizations $\mathcal{B}_1 \subseteq \mathcal{B}$ with respect to subject s that are in conflict, $\mathcal{C}$ contains a user-grant tuple $\langle t_{user}, t_{grant} \rangle$, where t_{user} is the user who has assigned the authorizations in $\mathcal{B}_1$, and t_{grant} is the grant with the highest precedence in $\mathcal{B}_1$ that t_{user} has given to s. Formally,

$$resC(\mathcal{B}, s) = \begin{cases} \{\langle user(C), grant(C) \rangle \mid \\ C \in \mathcal{B}, \forall A \in \mathcal{B} : C \gg A\} & \text{if } \forall A, B \in \mathcal{B} : \\ & (user(A) = user(B)) \wedge \\ & (s \in subject(A) \cap subject(B)) \\ \bigcup_{\mathcal{B}_1 \in group(\mathcal{B}, s)} resC(\mathcal{B}_1, s) & \text{otherwise} \end{cases}$$

Given a tuple $t \in resC(\mathcal{B}, s)$, we use t_{user} and t_{grant} to refer to the first and second element of t, respectively.

Example 4.5. Consider the set of authorizations $\mathcal{B}$ from Example 4.3. The resolve-conflicts function $resC(\mathcal{B}, Alice)$ works as follows: First, it groups the authorizations that Alice has received by the user who has assigned them. That is, it calls the authorization-grouping function $group(\mathcal{B}, Alice)$. This function outputs the set $\mathcal{C} = \{\{A, B\}, \{C, E\}\}$. Second, for each subset in $\mathcal{C}$, it applies the deny-mutual precedence strategy and outputs a user-grant tuple, which contains the user and the grant corresponding to the authorization with the highest precedence. Then, for the subset $\{A, B\}$, since $B \gg A$, the output user-grant tuple is $\langle Bob, deny \rangle$. For the subset $\{C, E\}$, since $E \gg C$, the output user-grant tuple is $\langle Carol, mutual \rangle$. Finally, the resolve-conflicts function $resC(\mathcal{B}, Alice)$ outputs the set $\{\langle Bob, deny \rangle, \langle Carol, mutual \rangle\}$.

4.1.4 Authorized Access Request

Answering an access request, Definition 4.1, requires solving first any authorization conflict. An access request $\langle s, op, res_u \rangle$ is authorized if, after resolving conflicts with respect to s, there exists (1) a tuple with the grant *allow* or (2) a tuple with

the grant *mutual*, and after resolving conflicts with respect to u there is a tuple either with the grant *allow* or *mutual*. Formally:

Definition 4.7: Authorized access request

Given the set of authorizations $\mathcal{A}$, an **access request** $\langle s, op, res_u \rangle$ is **authorized** if one of the following conditions is met:

1. $\exists t \in resC(\mathcal{A}, s) : t_{user} = u \wedge t_{grant} = allow$
2. $\exists t \in resC(\mathcal{A}, s), \exists e \in resC(\mathcal{A}, u) : t_{user} = u \wedge t_{grant} = mutual \wedge e_{user} = s \wedge (e_{grant} = allow \vee e_{grant} = mutual)$.

We refer to the process of deciding whether an access request is authorized or not as *access decision process*.

When using *mutual* authorizations, depending on the scenario, different issues can arise. In the next section, we discuss issues that can influence the implementation of *mutual* authorizations and present alternatives to solve them.

4.2 Extending Mutual Authorizations

The idea of *mutual* authorizations is that I let you profit from my resource if and only if I can profit from yours. This implies that the exchange of resources must be fair. The reference standard to determine what is a fair or unfair exchange is an equitable share of payoffs.[FF06] In the *mutual* authorization context, fairness implies two principles:

- **Profit Guarantees:** It means to guarantee the right of a user to profit from an exchange.
- **Equal Payoffs:** It means to equal, to some extent, the payoffs that a pair of users receive from exchanging resources with each other.

Two issues can affect these two fairness principles. The first problem, named *revocation fraud*, is associated with authorizations updates, and it can affect the first principle. The second problem, named *sensitivity problem*, is related to resources with different sensitivity, and it can affect the second principle. Subsections 4.2.1 and 4.2.2 describe these problems and show how one can solve them. Based on these solutions, we extend the semantics and syntax of *mutual* authorizations in Subsection 4.2.3.

For the sake of simplicity, we assume that each user owns one resource. This actually is the case with various real settings, such as health records or physical

positions of users. We expect the techniques used to solve the two problems mentioned to be applicable in scenarios where users own more than one resource as well.

4.2.1 Revocation Fraud Problem

Problem Description

Users may relax their authorizations for a moment, merely to spy out the others, i.e., change a negative authorization to *mutual*, and right after accessing the interesting resources return to negative. We call this problem *revocation fraud*.

Solution

Any solution to this problem has to guarantee that users will always profit from *mutual* authorizations. That is, for instance, if Alice accesses a resource owned by Bob using a *mutual* authorization, Bob should be able to access the resource owned by Alice regardless of any future authorization updates.

To this end, we propose to add a mechanism to the access control model. We call this mechanism, access control list. Intuitively, an access control list registers information about the users who access resources using *mutual* authorizations. We then integrate the registered information into the access decision process. Specifically, an access control list, dubbed *acl*, is a set of subjects authorized to perform an operation on a specific resource. Next, we formalize the syntax and semantics of an access control lists.

Definition 4.8: Access control entry

An access control entry is a tuple t of the form $\langle s, op \rangle$ where $s \in S$ and $op \in Op$. We use $subject(t)$ and $op(t)$ to denote, respectively, the subject and operation of t.

Definition 4.9: Access control list

Given a resource res, the access control list of res, acl_{res}, is a list of access control entries.

Definition 4.10: Access control list: Semantics

Given an access control list of a resource res, acl_{res}, a tuple $t \in acl_{res}$ indicates that the $subject(t)$ is allowed to perform the operation $op(t)$ on the resource res.

To overcome the revocation fraud problem, we apply the following changes to the access control model:

- **Register:** If Alice accesses the resource of Bob using a *mutual* authorization, Bob is added to the access control list of the resource owned by Alice.
- **Verify:** When Bob wants to access to the resource of Alice, the access decision process verifies first if Bob is in the access control list of such a resource. If so, Bob is granted access to the resource of Alice regardless of the authorizations in $\mathcal{A}$.
- **Delete:** Bob is deleted from the access control list of the resource of Alice after he has accessed her resource.

Users cannot modify the access control lists of their resources. The access control authority has to guarantee the integrity of these lists.

4.2.2 Sensitivity Problem

Problem Description

To introduce the concept of *mutual* authorization, we have considered, so far, one type of resource with the same degree of sensitivity – our example has been physical positions. In more complex scenarios, resources can have different degrees of sensitivity, e.g., health records. In such scenarios, although *mutual* authorizations state that only resources with the same sensitivity may be exchanged, if the users assign a sensitivity level to their resources, unfair exchanges can happen. We call this problem *sensitivity problem*.

Example 4.6. Think of a health record system. Consider two users Alice and Bob, each one owning a health record, and assume that they have assigned the same sensitivity to their health records. Alice may not find it fair to open her health record if she has a stigmatizing disease in exchange for looking at the record of Bob, who is in perfect health. However, whether a disease is stigmatizing or not is a subjective issue.

Solution

To deal with the sensitivity problem, at first sight, one might let the system or a trusted authority assign the degree of sensitivity of all resources. Such a solution could be easily implemented by extending the semantics of *mutual* authorizations to specify that users can only exchange resources with the same sensitivity. However, the owners of the resources may not agree with the assigned sensitivity.

Example 4.7. Think of a library management system and of two users Alice and Bob who want to exchange their thesis. Suppose that a system determines the degree of sensitivity of all resources, e.g., books, theses, etc. Since Alice and Bob have the same type of resource, i.e., thesis, the system may assign the same degree of sensitivity to them. However, after the exchange, Alice may find it unfair to have exchanged her thesis with Bob because, from her point of view, the content of her thesis is more sensitive than the one of Bob.

Our solution to alleviate the sensitivity problem consists of three parts:

1. The owners specify the degree of sensitivity of their resources subjectively.
2. When using *mutual* authorizations, users specify the minimum level of trust that subjects involved in such authorizations must have to access their resources.
3. The trust of a user u is computed using the ratings provided by the users who have accessed the resource of u, res_u. These ratings depend on how users perceive the sensitivity of res_u compared to the one assigned by u.

To implement the first part of our solution, one needs to define a sensitivity scale. The second part requires to extend the syntax and semantics of *mutual* authorization. The next subsection, Subsection 4.2.3, features this extension. Finally, the third part can be implemented using a reputation system. In principle, one can use any reputation system that fulfills the following requirements:

- *Score:* It has to provide a trust (reputation) score for each user, a real number.
- *Rating:* It must support positive and negative ratings, within a numerical scale. For example, if Alice accesses a resource owned by Bob, with sensitivity 10, Alice may rate the transaction as negative if she deems the sensitivity of the resource accessed lower.

- *Scope:* It must compute a global trust score for each user u, considering the opinions of all the peers who have interacted with u.
- *Integrity:* The system has to guarantee that users will not be able to tamper with their reputation.

Similarly to reputation systems, our solution, at first sight, is prone to whitewashing attacks. In this attacks, users with poor reputation change their identities to start over again. One can solve this problem by using existing solutions for reputation systems. For instance, one can restrict the number of identifiers that each user can obtain. Such a restriction can be done by binding user identifiers with IP addresses or by requiring entry fees[SL12].

4.2.3 Trust-based authorizations

In this subsection, we extend the syntax and semantics of *mutual* authorizations, Section 4.1, by incorporating our solutions to the revocation fraud and sensitivity problems discussed in Subsections 4.2.1 and 4.2.2.

We extend the concept of *mutual* authorizations to allow users to specify the minimum level of trust of subjects to perform an operation on their resources. We call our extension *trust-based authorizations*. In the following, we define their syntax and semantics.

Syntax

As explained in Subsection 4.2.2, we use a reputation system to determine the trust level of each user. In such a system, each user has a trust value which is computed based on the evaluation received by the other users. We use $trust_w$ to denote the trust value of user w, i.e., the trust that the system places on w.

Definition 4.11: Trust-based authorization: Syntax

Let a person constraint $Cons_{Person}$, a resource constraint $Cons_{Resource}$, an operation $op \in Op$, a grant $gr \in Gr$, and a trust value $tr \in \mathbb{R}_{>0}$ be given. A **trust-based authorization** A_t is a 5-element tuple $\langle\, Cons_{Person}, Cons_{Resource}, op, gr, tr \,\rangle$. We use $trust(A_t)$ to denote the trust value specified in A_t. $\mathcal{A}_T$ denotes the set of all trust-based authorizations.

Definition 4.11 includes *positive*, *mutual*, and *negative* authorizations. Since the sensitivity problem is only related to *mutual* authorizations, in the case of *positive* and *negative* authorizations, the trust value has to be ignored.

Semantics

Next, we define the semantics of trust-based *mutual* authorizations. The semantics of *positive* and *negative* authorizations remain as defined before while ignoring the trust conditions. Given a resource res, let $sensitivity(res)$ denote the degree of sensitivity of res, i.e., the value of *sensitivity* corresponding to res.

Definition 4.12: Trust-based *mutual* authorization: Semantics

Let a trust-based *mutual* authorization A_t, where $grant(A_t) = mutual$, be given. A_t states that $user(A_t)$ allows invoking $op(A_t)$ on $res(A_t)$ to all subjects $s \in subjects(A)$ who have issued an authorization B to $user(A_t)$ if the following expression evaluates to true: $(sensitivity(\ res(B_t)) = sensitivity(res(A_t))) \wedge (trust_s \geq trust(A_t)) \wedge (trust_{user(A_t)} \geq trust(B_t)) \wedge ((grant(B_t) = allow) \vee (grant(B_t) = mutual)) \wedge (op(B_t) = op(A_t))$.

Similarly to authorizations without trust conditions, authorization conflicts may arise when using trust-based authorizations. In the next subsection, we discuss these conflicts and explain how to handle them.

Trust-based Conflict Resolution

Trust-based authorizations have two sources of conflicts. First, similarly to authorizations without trust, a conflict exists when a subject receives from the same user trust-based authorizations with different grants. Second, a conflict exists when a subject receives from the same user trust-based authorizations with the same grants but with different trust values.

Example 4.8. Let res_{Bob} be a resource owned by Bob. Consider a user Alice with roles $r(Alice) = \{r1, r2\}$ and the trust-based authorizations $A_t = \langle role{=}r1, res_{Bob}, read, mutual, 0.8\rangle$ and $B_t = \langle role{=}r2, res_{Bob}, read, mutual, 0.6\rangle$. The trust-based authorizations A_t and B_t are in conflict with respect to Bob and Alice. Namely, A_t and B_t assign a *mutual* grant to s, but the trust values of both authorizations are different.

To solve the first source of conflict, we use the *deny-mutual* precedence strategy ($deny \gg mutual \gg allow$). To solve the second source of conflict, we prioritize trust-based authorizations with the highest trust values. We make this choice for the same reasons that we assign the highest precedence to a *negative* authorization, i.e., to eliminate the risk of possible leakages. For sake of simplicity, we split our resolve conflicts function in two parts, Definitions 4.13 and 4.14. The first part

and second part resolve conflicts by taking into account the grants and the trust values, respectively.

Given two trust-based authorizations A_t and B_t, we write $A_t \gg_{grant} B_t$ and $A_t \gg_{trust} B_t$ to denote, respectively, $grant(A_t) \gg grant(B_t)$ and $trust(A_t) \geq trust(B_t)$.

Definition 4.13: Grant Resolve-Conflicts Function

A **grant resolve-conflicts function** — $resC_{gr} : \mathcal{P}(\mathcal{A}_T) \times S \to \mathcal{P}(\mathcal{A}_T)$ is a function that takes as input a set of authorizations $\mathcal{B} \subseteq \mathcal{A}_T$ and a subject $s \in S$ and outputs a set $\mathcal{C}$ of trust-based authorizations. For each set of authorizations $\mathcal{B}_1 \subseteq \mathcal{B}$ with respect to subject s that are in conflict, $\mathcal{C}$ contains a trust-based authorization C_t such that C_t is the authorization with highest grant precedence in $\mathcal{B}_1$. Formally,

$$resC_{gr}(\mathcal{B}, s) = \begin{cases} \{C_t \mid C_t \in \mathcal{B}, \\ \forall A_t \in \mathcal{B} : C_t \gg_{grant} A_t\} & \text{if } \forall A_t, B_t \in \mathcal{B} : \\ & user(A_t) = user(B_t) \wedge \\ & s \in subjects(A_t) \cap \\ & subjects(B_t) \\ \bigcup_{\mathcal{B}_1 \in group(\mathcal{B}, s)} resC_{gr}(\mathcal{B}_1, s) & \text{otherwise} \end{cases}$$

Definition 4.14: Grant-Trust Resolve-Conflicts Function

A **grant-trust resolve-conflicts function** — $resC^{tr}_{gr} : \mathcal{P}(\mathcal{A}_T) \times S \to \mathcal{P}(\mathcal{A}_T)$ is a function that takes as input a set of authorizations $\mathcal{B} \subseteq \mathcal{A}_T$ and a subject $s \in S$ and outputs a set $\mathcal{C}$ of trust-based authorizations. For each set of authorizations $\mathcal{B}_1 \subseteq \mathcal{B}$ with respect to subject s that are in conflict, $\mathcal{C}$ contains a trust-based authorization C_t such that $grant(C_t)$ is the grant with the highest precedence in $\mathcal{B}_1$, and $trust(C_t)$ is the highest trust value in $\mathcal{B}_1$. Formally,

$$resC^{tr}_{gr}(\mathcal{B}, s) = \begin{cases} \{C_t \mid C_t \in \mathcal{B}, \\ \forall A_t \in \mathcal{B} : C_t \gg_{trust} A_t\} & \text{if } \forall A_t, B_t \in \mathcal{B} : \\ & user(A_t) = user(B_t) \wedge \\ & s \in subjects(A_t) \cap \\ & subjects(B_t) \wedge \\ & grant(A_t) = grant(B_t) \\ \bigcup_{\mathcal{B}_1 \in group(resC_{gr}(\mathcal{B}, s), s)} resC^{tr}_{gr}(\mathcal{B}_1, s) & \text{otherwise} \end{cases}$$

Figure 4.2 illustrates the steps that the grant-trust resolve-conflict function takes to solve authorization conflicts and Algorithm 1 implements the function.

Figure 4.2: Grant-trust Resolve-conflict Function $resC^{tr}_{gr}(\mathcal{B}, Alice)$

Algorithm 1: grantTrustResolveConflicts

Input : Authorization Set $\mathcal{B}_T \subseteq \mathcal{A}_T$, subject $s \in S$
Output: Authorization Set $\mathcal{C}_T \subseteq \mathcal{B}_T$

Initialize: $\mathcal{C} \leftarrow \emptyset$, $\mathcal{D} \leftarrow \emptyset$;
/* Select the authorizations in $\mathcal{B}_T$ assigned to s and group them by the users who have assigned them. Output a set containing these groups of authorizations */
$\mathcal{D}_T \leftarrow group(\mathcal{B}, s)$;
foreach *set* $\mathcal{D}_1$ *in* $\mathcal{D}_T$ **do**
  Create $B_t \leftarrow \langle\ \rangle$; // Create an empty authorization
  foreach *authorization* A_t *in* $\mathcal{D}_1$ **do**
    if $grant(A_t) \gg grant(B_t)$ **then** // Solve grants conflicts
      $B_t \leftarrow A_t$; // Replace B_t with A_t
    if $grant(A_t) == grant(B_t)$ **then**
      if $trust(A_t) \gg trust(B_t)$ **then** // Solve trust values conflicts
        $B_t \leftarrow A_t$; // Replace B_t with A_t
  Add B_t to $\mathcal{C}_T$;
return $\mathcal{C}_T$

Trust-based Authorized Access Request

Next, we define how to evaluate whether an access request is authorized or not when using trust-based authorizations. Such a definition has to include our solutions to the revocation fraud and sensitivity problems.

Given a set of trust-based authorizations $\mathcal{A}_T$, Figure 4.3 illustrates the process to decide whether an access request is authorized or not, i.e., the steps needed to

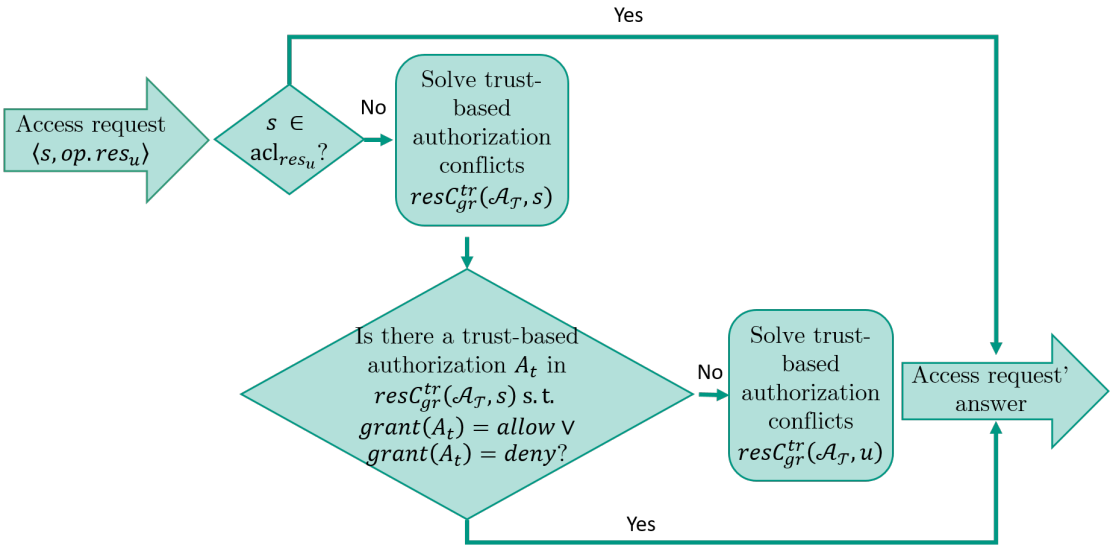


Figure 4.3: Trust-based authorizations Access Request Process

answer of a given access request. Definition 4.15 formalizes an authorized access request.

Definition 4.15: Authorized access request with acl

Given the set of trust-based authorizations $\mathcal{A}_\mathcal{T}$, an **access request** $\langle s, op, res_u \rangle$ is **authorized** if one of the following conditions is met:

1. $\exists t_{acl} \in acl_{res_u} : subject(t_{acl}) = s$
2. $\exists A_t \in resC^{tr}_{gr}(\mathcal{A}_\mathcal{T}, s) : (user(A_t) = u) \wedge (grant(A_t) = allow)$
3. $\exists A_t \in resC^{tr}_{gr}(\mathcal{A}_\mathcal{T}, s), \exists B_t \in resC^{tr}_{gr}(\mathcal{A}_\mathcal{T}, u) : (user(A_t) = u) \wedge (grant(A_t) = mutual) \wedge (trust(A_t) \leq trust_s) \wedge (user(B_t) = s) \wedge \big((grant(B_t) = allow) \vee ((grant(B_t) = mutual) \wedge (trust(B_t) \leq trust_u)) \big) \wedge (sensitivity(res(A_t)) = sensitivity(res(B_t)))$

4.3 Integrating Mutual Authorizations into LBS

In this section, we use location-based services to showcase how one can integrate *mutual* authorizations into existing services. Such integration has to reuse existing implementations and guarantee that only authorized users can access the services, i.e., data confidentiality. To achieve data confidentiality, one has to verify the authorizations and to determine whether a given user should get access to the resources required for the service.

Example 4.9. Think of a LBS provider (LBSP). A user has issued a location-dependent query that the LBSP now executes, knowing the position of the user. How can the LBSP take in authorizations to answer such queries? The result of such a query is the set of persons who fulfill the query constraint, and whose physical positions the querying user may see. Different designs of this integration are conceivable. In particular, one may (1) execute the query first and then filter the result based on the authorizations, or (2) first compute the set of positions that the querying user may see, called *view of the user*, and then execute the query on this view. Deciding which alternative is better is not trivial.

In the remainder of this chapter, since the type of resources that users are willing to exchange are the same, i.e., physical positions, we consider they have the same sensitivity. Therefore, we do not specify the degree of sensitivity of the resources and use the authorizations without trust conditions.

Before showing how to integrate LBS with *mutual* authorizations, it is important to define the soundness principle that an algorithm in the context of LBS and *mutual* authorizations should fulfill. In the next subsection, Subsection 4.3.1, we define this principle.

4.3.1 Soundness Criteria

An algorithm is sound if it is both *correct* and *complete* [SZFJ09]. Intuitively, in our integration of *mutual* authorization into LBS, an algorithm is complete if all persons who satisfy the query constraints and whose information the querying user may see are part of the result. It is correct if the users in the result satisfy the query constraints, and the querying user is allowed to see their information. Before formalizing the correctness and completeness properties in the context of LBS and *mutual* authorizations, we introduce two constraints, namely location and authorization constraints. We will use these constraints to define the properties mentioned above.

Definition 4.16: Location Constraint

A **location constraint**, *LCons*, is a predicate on physical positions.

We use $dist(p_u, p_v)$ to denote the distance between the physical positions of persons u and v.

Example 4.10. Consider a distance d and the physical positions of two persons u and v, p_u and p_v, respectively. The constraint $dist(p_u, p_s) \leq d$ is a location constraint.

Definition 4.17: Authorization Constraint

Given two persons u and s, an **authorization constraint** $ACons$ is a predicate on a set of authorizations M that involve persons u and s.

Example 4.11. Let a set of authorizations M and two persons u and s be given. The access request $\langle s, read, p_u \rangle$ is an authorization constraint. If $\langle s, read, p_u \rangle$ is authorized, the predicate evaluates to *true*; otherwise it evaluates to *false*.

Definition 4.18: Query

Given a set of persons $\mathcal{P}$, a **query** $Q(\mathcal{C})$ is a set of location and authorization constraints $\mathcal{C}$. Its output is the elements of $\mathcal{P}$ that fulfill $\mathcal{C}$. $Ans_{\mathcal{P}}(Q(\mathcal{C}))$ denotes the output of $Q(\mathcal{C})$.

Given as input a set of constraints, a user algorithm is an algorithm that outputs a set of users who fulfill the given set of constraints.

Definition 4.19: User algorithm

A **user algorithm** $\Pi : \mathcal{P}(U) \times \mathcal{P}(Cons) \rightarrow \mathcal{P}(U)$ is an algorithm that takes as input a set of users $U_1 \in \mathcal{P}(U)$ and a set of constraints $\mathcal{C}$, and outputs a set of users U_2.

In the context of LBS and *mutual* authorizations, a user algorithm Π computes a location and an authorization constraint on the physical positions of a given set of persons P. Based on these two constraints, we define the correctness and completeness of algorithm Π.

Definition 4.20: Correctness in the context of LBS and *mutual* authorizations

Let a user algorithm Π, a set $U_1 \in \mathcal{P}(U)$ and a set of constraints $\mathcal{C} = \{LCons, ACons\}$ be given. The user algorithm Π is **correct** with respect to

U_1 and $\mathcal{C}$ if for all $u \in \Pi(U_1, \{LCons,\ ACons\})$, $u \in Ans_{U_1}(Q(LCons)) \wedge u \in Ans_{Ans_{U_1}(Q(LCons))}(Q(ACons))$.

In other words, a given algorithm Π is correct if for all users u in the output of Π the following two conditions are met:

1. u is a person in $U1$ that fulfills $LCons$.
2. u is a person in $Ans_{U_1}(Q(LCons))$ that fulfills $ACons$.

Definition 4.21: Completeness in the context of LBSs and *mutual* authorizations

Let a user algorithm Π, a set $U_1 \in \mathcal{P}(U)$, and the set of constraints $\mathcal{C} = \{LCons, ACons\}$ be given. Π is **complete** with respect to U_1 and $\mathcal{C}$ if for all persons u with $u \in Ans_{U_1}(Q(LCons)) \wedge u \in Ans_{Ans_{U_1}(Q(LCons))}(Q(ACons))$, $u \in \Pi(U_1, \{LCons,ACons\})$.

To illustrate the completeness property, think of a user algorithm Π that always outputs an empty set. Then Π fulfills the correctness principle. However, Π is not useful.

Definition 4.22: Soundness in the context of LBS and *mutual* authorizations

Let a user algorithm Π, a set $U_1 \in \mathcal{P}(U)$ and the set of constraints $\mathcal{C} = \{LCons, ACons\}$ be given. Π is **sound** with respect to U_1 and $\mathcal{C}$ if: Π is correct with respect to U_1 and $\mathcal{C}$, and Π is complete with respect to U_1 and $\mathcal{C}$.

Before proceeding to describe our set of primitives and algorithms for integrating LBS with *mutual* authorizations, in the next subsection, we present our algorithm that implements our resolve conflicts function, Definition 4.6.

4.3.2 Resolve Conflicts Algorithm

Authorization conflicts can be solved at design time, i.e., during the insertion of authorizations in a system, or at query time, i.e., when access to a resource is required. Solving authorization conflicts at design time could require to modify the structure of an organization, see Example 4.12. For this reason, we solve authorization conflicts at query time. Algorithm 2 implements our resolve conflicts function, Definition 4.6. The algorithm outputs a map *autMap*, which stores pairs

of keys and values. The key is a pair consisting of a user $u \in \mathcal{U}$ and a subject $s \in \mathcal{S}$, and the value of the map is the grant of the authorization $A \in \mathcal{B}$ with the lower precedence with respect to u and s. We use the left arrow "$\leftarrow$" to indicate that the value on the right-hand side is assigned to the term on the left-hand side.

Example 4.12. Let us consider Example 4.3. Assume now that the authorization A has been inserted first in the system, and now Bob wants to insert authorization B. Based on the *deny-mutual* precedence strategy, authorization B has precedence over A. Then, during the insertion process, with respect to Alice, authorization B should be inserted, and A should be deleted. However, this change will affect all users who have either $role1$ or $role2$. In this case, it may be necessary to modify the organizational structure of the business in such a way that this authorization update will not affect other users.

Next, given as input to Algorithm 2 the set of authorizations $\mathcal{B}$, the set of users $\mathcal{U}$, and the set of subjects $\mathcal{S}$, we prove its correctness with respect to Definition 4.6. That is, we show that Algorithm 2 solve all authorization conflicts in the set $\mathcal{B}$ with respect to each subject $s \in \mathcal{S}$ and the set of users $\mathcal{U}$.

Lemma 4.1. *Given an authorization set $\mathcal{B} \subseteq \mathcal{A}$, a set of users $\mathcal{U}$, a set of subjects $\mathcal{S}$, and the subset of authorizations $N = \{A \in \mathcal{B} \mid user(A) \in \mathcal{U}, subjects(A) \cap \mathcal{S} \neq \emptyset\}$, for each $s \in \mathcal{S}$ and for each tuple $\langle t_{user}, t_{grant} \rangle \in resC(N, s)$ exists an entry $e = ((t_{user}, s), t_{grant})$ in resolveConflicts$(\mathcal{B}, \mathcal{U}, \mathcal{S})$.*

Proof. First, we show that each entry in the map $autMap$, output by Algorithm 2, corresponds to one authorization in the set $N = \{A \in \mathcal{B} \mid user(A) \in \mathcal{U}, subjects(A) \cap \mathcal{S} \neq \emptyset\}$. Algorithm 2 considers all authorizations $A \in \mathcal{B}$ (Line 2), and for each authorization, the algorithm evaluates if A satisfies the constraint $user(A) \in \mathcal{U}, subjects(A) \subseteq \mathcal{S}$ (Line 3). The map output by the algorithm contains only authorizations that satisfy such a constraint (Lines 7 and 9). Second, we show that for each entry $e = ((u, s), grant)$ in $autMap$, where $u \in \mathcal{U}$ and $s \in \mathcal{S}$, the grant $authMap.get(u, s)$ is the one with the highest precedence with respect to u and s in the set $\mathcal{B}$. For each pair of elements (u, s), Algorithm 2 verifies if there is an entry with key (u, s) in $autMap$ (Line 5). If such an entry exists, the algorithm updates the value of the entry, i.e., the grant, only if the grant has lower precedence than the grant of the authorization that is being evaluated, $grant(A)$ (Lines 6-7). If an entry with key (u, s) does not exist in the map, Algorithm 2 inserts in the map the entry with key (u, s) and value $grant(A)$ (Line 9). Once Algorithm 2 has evaluated all authorizations in $\mathcal{B}$, $autMap$ will contain, for each pair of elements (u, s), the grant with the highest precedence in the set $\mathcal{B}$ with

Algorithm 2: resolveConflicts

Input : Authorization Set $\mathcal{B}$, user Set $\mathcal{U}$, subject Set $\mathcal{S}$
Output: Map *autMap*

```
/* Initialize an empty map autMap to store pairs of keys and
   values. The key is a pair (u, s), where u is a user and s is a
   subject, and the value is a grant of an authorization.        */
```

1 Initialize: $autMap\langle(user, subject), grant\rangle \leftarrow$ *empty map*;
2 **foreach** *authorization* A *in* $\mathcal{B}$ **do**

```
    /* Verify if user(A) is inside the set U and one or more
       subjects in subjects(A) are inside the set S                */
```

3 **if** $user(A) \in \mathcal{U} \wedge subjects(A) \cap \mathcal{S} \neq \emptyset$ **then**
4 **foreach** $s \in subjects(A) \cap \mathcal{S}$ **do**

```
        /* Verify if autMap contains an entry with key
           (user(A), s)                                            */
```

5 **if** $autMap.containsKey((user(A), s))$ **then**

```
            /* Verify if the value of the entry with key
               (user(A), s) has lower precedence than the grant
               grant(A)                                            */
```

6 **if** $autMap.get((user(A), s)) \ll grant(A)$ **then**

```
                /* Replace the value of the entry with key
                   (user(A), s) with the grant grant(A).           */
```

7 $autMap.put((user(A), s), grant(A))$;
8 **else**

```
                // Add the entry ⟨(user(A), s), grant(A)⟩ to AutMap.
```

9 $autMap.put((user(A), s), grant(A))$;
10 **return** *autMap*;

respect to u and s. Then, each entry $e = ((u, s), grant)$ corresponds to a tuple in $\langle t_{user} = u, t_{grant} = grant\rangle$ in $resC(N, s)$. □

4.3.3 Primitives and Algorithms for Mutual Authorizations

Depending on the services offered by a system, one may need different primitives. A primitive is a basic unit that performs a specific functionality, and that can be combined with other primitives. In the case of LBS, to answer queries, we need to know which persons a given person has allowed reading his physical position. The two primitives *Primitive-Request* and *Primitive-View* are sufficient to this end, as we will show in Subsection 4.3.5.

- *Primitive-Request:* Given two persons u and s, may s read the physical position of u?
- *Primitive-View:* Given a person s, whose physical positions is s allowed to read? We call this set $View_s$, the view of s.

In the following, we present our algorithms, *Pr-Request* algorithm and *Pr-View* algorithm to implement the primitives *Primitive-Request* and *Primitive-View*, respectively. Since authorization conflicts can exist, both algorithms make use of the *resolveConflicts* algorithm, Algorithm 2.

Given two persons u and s, the *Pr-Request* algorithm, Algorithm 3, determines if person s can can read the physical position of person u. Given a person s, the *Pr-View* algorithm, Algorithm 4, outputs the view of s.

Next, we prove that given the authorization set $\mathcal{A}$ and a person s, the output of the *Pr-View* algorithm, Algorithm 4, contains all persons that s is allowed to read their physical positions and not more. We only present the proof of Algorithm 4. The proof of Algorithm 3 can be done similarly as the proof of Algorithm 4.

Lemma 4.2. *Given an authorization set $\mathcal{A}$ and a person s,*

1. *For all persons $u \in$ Pr-View$(\mathcal{A}, s)$, Algorithm 4, s is authorized to read the physical position of u with respect to Definition 4.7.*
2. *If $u \notin$ Pr-View$(\mathcal{A}, s)$, then s is not authorized to read the physical position of u with respect to Definition 4.7.*

Proof. We will prove that for each user $u \in$ *Pr-View*$(\mathcal{A}, s)$, $\langle s, read, p_u \rangle$ is authorized, Definition 4.7, if one of the following conditions is met:

(1) $\exists t \in resC(\mathcal{A}, s) : t_{user} = u \wedge t_{grant} = allow.$

(2) $\exists t \in resC(\mathcal{A}, s), \exists l \in resC(\mathcal{A}, u) : t_{user} = u \wedge t_{grant} = mutual \wedge l_{user} = s \wedge (l_{grant} = allow \vee l_{grant} = mutual)$

Lemma 4.1 proves that for each tuple $\langle\, t_{user},\, t_{grant} \,\rangle \in resC(\mathcal{A}, s)$ exists an entry $e = ((t_{user}, s), t_{grant})$ in the output *resolveConflicts*$(\mathcal{A}, \{U\}, \{s\})$. We replace, in conditions (1) and (2), a tuple $t = \langle t_{user}, t_{grant} \rangle \in resC(\mathcal{A}, s)$ with an entry $e = ((t_{user}, s), t_{grant}) \in$ *resolveConflicts*$(\mathcal{A}, \{U\}, \{s\})$. Next, Algorithm 4 adds a person u to its output set, if $t_{user} = u \wedge t_{grant} = allow$ (Lines 3-4). That is, this step evaluates condition (1). Next, if $t_{grant} = mutual$, Algorithm 4 adds u to the set $MutualRA$ (Lines 5-6). Then, for each $u \in MutualRA$, the algorithm adds u to its output set, if there exists an entry f in *resolveConflicts*$(\mathcal{A}, \{s\}, MutualRA)$ such

Algorithm 3: Pr-Request

Input : Authorization Set $\mathcal{A}$, Access request $\langle s, read, p_u \rangle$
Output: Boolean resp

/* The set Set_{pp} contains the person whose physical position is requested. The set Set_{req} contains the person who request the access. */

1 Initialize: $Set_{pp} \leftarrow \{u\}$, $Set_{req} \leftarrow \{s\}$,
$ReceiveAut_s\langle(user, subject), grant\rangle \leftarrow$ *empty map*,
$ReceiveAut_u\langle(user, subject), grant\rangle \leftarrow$ *empty map*;

/* Invoke the *resolveConflicts* Algorithm. $ReceiveAut_s$ contains only one entry e with key (u, s) and the value corresponds to the grant of the authorization with the highest precedence that u has assigned to s. */

2 $ReceiveAut_s \leftarrow resolveConflicts(\mathcal{A}, Set_{pp}, Set_{req})$;
3 **foreach** *entry* e *in* $ReceiveAut_s$ **do**
4 **if** $e.getValue() = allow$ **then**
/* Output true if the grant is "allow". s can read p_u */
5 **return** true;
6 **if** $e.getValue() = mutual$ **then**
/* If the grant is "mutual", invoke the *resolveConflicts* Algorithm. $ReceiveAut_s$ contains only one entry t with key (s, u) and the value corresponds to the grant of the authorization with the highest precedence that s has assigned to u. */
7 $ReceiveAut_u \leftarrow resolveConflicts(\mathcal{A}, Set_{req}, Set_{pp})$;
8 **foreach** *entry* t *in* $ReceiveAut_u$ **do**
9 **if** $t.getValue() = allow \vee t.getValue() = mutual$ **then**
10 **return** true ; // Output true if the grant is "allow"
11 **return** false ; // s cannot read p_u.

that $f = ((s, u), allow)$ or $f = ((s, u), mutual)$ (Lines 9-10). That is, this step evaluates condition (2). □

4.3.4 System Architecture

Our system architecture consists of a location-based service provider, *LBSP*, and users who want to access LBS. The *LBSP* has: (1) a user database DB_U, which stores the position of each user u, p_u, and (2) an authorization database, DB_A, which stores the set of authorizations $\mathcal{A}$. See Figure 4.4. We assume a database

Algorithm 4: Pr-View

Input : Authorization Set $\mathcal{A}$, person s
Output: Set $View_s$

```
/* Initialize the set U, which contains all users U, the set S,
   which contains the person given as input, s, the empty set
   MutualRA, and the empty map AA_s                              */
```
1 Initialize: $\mathcal{U} \leftarrow U, \mathcal{S} \leftarrow \{s\}, View_s \leftarrow \emptyset$, $AA_s\langle(user, subject), grant\rangle \leftarrow$ *empty map*, $MutualRA \leftarrow \emptyset$;

```
/* Invoke the resolveConflicts Algorithm.                        */
```
2 $ReceiveAut_s\langle(user, subject), grant\rangle \leftarrow resolveConflicts(\mathcal{A}, \mathcal{U}, \mathcal{S})$; **foreach** *entry e in* $ReceiveAut_s$ **do**
3 | **if** $e.getValue() = allow$ **then**
| | /* If the value of the entry $einReceiveAut_s$ is "allow", add the user, which is part of the key of entry e, to the set $View_s$ */
4 | | add $e.getKey.User()$ to $View_s$;
5 | **if** $e.getValue() = mutual$ **then**
| | /* If the value of the entry $einReceiveAut_s$ is "mutual", add the user, which is part of the key of entry e, to the set $MutualRA$ */
6 | | add $e.getKey.User()$ to $MutualRA$;
7 **if** $MutualRA.size() \neq 0$ **then**
| /* Invoke the $resolveconflicts$ algorithm */
8 | $AuthMap_s \leftarrow resolveConflicts(\mathcal{A}, \mathcal{S}, MutualRA)$;
9 | **foreach** *u in* $MutualRA$ **do**
| | /* Verify if the map $AuthMap_s$ has an entry with key (s, u) and value "mutual" or "allow" */
10 | | **if** $AuthMap_s.get((s, u)) = mutual \vee AuthMap_s.get((s, u)) = allow$ **then**
11 | | | add u to $View_s$;
12 **return** $View_s$;

management system featuring R-tree indexing for spatial query processing on DB_U and B-tree indexing for authorizations queries on DB_A. In our prototype, we have implemented the LBS ourselves in Java, as well as access control, in the form of the primitives described in Subsection 4.3.3. Note that our focus is on the conceptual level; studying design alternatives regarding the architecture is not part of this dissertation.

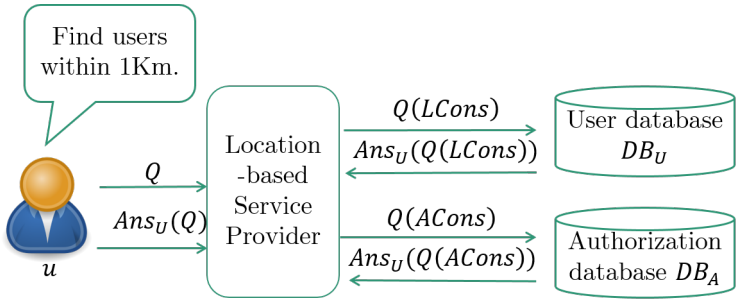


Figure 4.4: System Architecture

The LBSP supports location-dependent queries. There are different types of such queries [IMI10], and we focus on two of them here: k-nearest neighbor queries and range queries.

Definition 4.23: Location-dependent Query

Given a set of persons $\mathcal{P}$, a **location-dependent query** $Q(LCons)$ takes a location constraint $LCons$ and outputs the persons who fulfill it.

Definition 4.24: k-Nearest Neighbor Query

Given an integer k and a person s, a **k-nearest neighbor (kNN) query**, $knn(k, s)$, is a location-dependent query where the location constraint $knn_{k,s}(p_u)$ is: $\forall M \subseteq U, \big(\forall x \in M, dis(p_x, p_s) < dis(p_u, p_s)\big) \Rightarrow |M| \leq k$, where U is the set of all users. In words, if the previous predicate evaluates to true for a physical position p_u, then the corresponding person u is in the result of the $knn(k, s)$ query; otherwise not.

Definition 4.25: Range Query

Given a distance d and a person s, a **range query** $range(d, s)$ is a location-dependent query where the location constraint $range_{d,p_s}(p_x)$ is: $dist(p_x, p_s) \leq d$.

Definition 4.24 is a higher order logic. However, to facilitate proofs that our proposed approaches are sound, we will use the following recursive definition, Definition 4.26.

Definition 4.26: k-Nearest Neighbor Query Recursive Definition

A **k-nearest neighbor query** kNN is a location- dependent query with location constraint (k, s), where k is an integer number and s is the persons who issues the query. The result $Ans(knn)$ of such a query is the set of users $u \in U$ such that $|Ans(knn)| = k \wedge \forall u \in Ans(knn), \forall v \in U \setminus Ans(knn) : dist(p_s, p_u) \leq dist(p_s, p_v)$.

Definition 4.27: Bounded Result-Size Query

Given a location-dependent query $Q(LCons)$, Q is a **bounded result-size query** if the location constraint $(LCons)$ contains an explicit restriction on the number of elements of $Ans(Q)$. Otherwise, Q is a unbounded result-size query.

kNN and range queries are examples of bounded result-size and unbounded result-size queries, respectively. If Q is a bounded result-size query, after executing Q and filtering $Ans(Q)$ with the users whose position s is authorized to see, the filtered query result may not fulfill the original constraint $LCons$ anymore. See Examples 4.13 and 4.14.

Definition 4.28: Authorizations Received

Given the set of authorizations $\mathcal{A}$, the **authorizations that s has received** are all authorizations $A \in \mathcal{A}$ such that $s \in subjects(A)$.

Example 4.13. Consider a kNN query with constraint $knn = (2, Alice)$. Suppose that (1) the neighbors of Alice are Bob, Carol, and Dan, and their distances to Alice are 1, 2 and 3 km, respectively, and (2) Alice has received two authorizations A, B where $user(A) = Carol$, $grant(A) = allow$, $user(B) = Dan$ and $grant(B) = allow$. The LBSP evaluates the kNN query and outputs $Ans(kNN) = \{Bob, Carol\}$. After filtering $Ans(kNN)$ based on the authorizations that Alice has received, the result contains only $\{Carol\}$. This result does not meet the constraint $knn = (2, Alice)$. One needs to set the value of the parameter k equal to 3 to obtain, after the filtering step, $\{Carol, Dan\}$.

Example 4.14. Continuing with Example 4.13, suppose that *Alice* wants to find all persons within 2 km, i.e., $range(2km, Alice)$. The LBSP outputs

$Ans(range) = \{Bob, Carol\}$. After filtering $Ans(range)$ with respect to the authorizations that Alice has received, the filtered result is $\{Carol\}$, similar to Example 4.13. This result fulfills the constraint $range(2km, s)$.

4.3.5 Design Alternatives for Integrating LBS with Mutual Authorizations

We see two design alternatives to integrate *mutual* authorizations in our system architecture: *Querying-Filtering (QF)* and *Filtering -Querying (FQ)*. The advantage of QF is that it can leverage existing LBS implementations. However, it may need to restart the query processing, which affects performance, as we will discuss. FQ does not need to restart the query processing.

This subsection is organized as follows. First, we present our QF and FQ approaches for kNN queries and ranges queries. Then, we discuss their advantages and disadvantages when comparing them to each other. Next, we prove that our algorithms are sound. Finally, we analyze the complexity of our both approaches and compare them to determine which one performs better than the other one.

Querying-Filtering Approach (QF)

Given a location-dependent query $Q(LCons)$, the QF approach works as follows: First, the LBSP executes $Q(LCons)$ on the user database DB_U and returns $Ans(Q)$. Second, it filters $Ans(Q)$ for the persons whose position the querying user may read. For the filtering process, there are two options:

(a) Verify for each person $u \in Ans(Q)$ if $\langle s, read, p_u \rangle$ is authorized, i.e., execute *Primitive-Request*, Algorithm 3. If so, then u is added to the final answer.

(b) Compute *Primitive-View*, $View_s$, Algorithm 4. The final answer is the intersection of $View_s$ and $Ans(Q)$.

With Option (a), to solve authorization conflicts, one has to read all authorizations in $\mathcal{A}$ for each person in $Ans(Q)$. With Option (b), although it is still needed to solve authorization conflicts, the elements of $\mathcal{A}$ will be read at most two times. The first time, the algorithm obtains the authorizations assigned to the querying person. The second one, it verifies, for each *mutual* authorization, whether the access request is authorized. In the following, we focus on QF only in combination with (b).

Algorithms 5 and 6 show the details for implementing QF for kNN and range queries, respectively. Both algorithms assume that the LBSP uses index structures to answer location-dependent queries, like B-tree or R-tree. We call the services used by LBSP to compute kNN and range queries, *computeKNN*(knn) and *computeRange*($range$), respectively, where knn and $range$ are the location constraints of the queries.

Algorithm 5: Querying-Filtering kNN

Input : Authorization Set $\mathcal{A}$, int k, person s
Output: Set Ans

1 Initialize: $View_s \leftarrow \emptyset$, $Ans \leftarrow \emptyset$, $temp_{all} \leftarrow [\,]$, $temp_{visible} \leftarrow [\,]$, $notEnough \leftarrow true$, $k_{all} \leftarrow 0$, $k_{old} \leftarrow 0$;
2 $View_s \leftarrow Pr\text{-}View(\mathcal{A}, s)$; // Compute the view of s
3 **while** $notEnough$ **do**
  /* Estimate an integer value $k_{all} \geq k \wedge k_{all} \geq k_{old}$ such that after computing the k_{all}-nearest neighbors of s and filtering the result based on $View_s$, the filtered result fulfills the constraint (k, s) */
4 $k_{all} \leftarrow estimateK(k, k_{old}, s)$;
  /* Compute the k_{all}-nearest neighbors of s ordered by distance in ascending order */
5 $temp_{all} \leftarrow computeKNN(k_{all}, s)$;
  /* Select all users in $temp_{all}$ who are also in $View_s$. Add such users to $temp_{visible}$, while keeping the order by distance */
6 $temp_{visible} \leftarrow filter(temp_{all}, View_s)$;
7 **if** $temp_{visible}.size() \geq k$ **then**
    /* If $temp_{visible}$ contains at least k elements, output the first k */
8    $Ans \leftarrow$ select $topK(k, temp_{visible})$;
9    $notEnough \leftarrow false$;
10 **else**
11    $k_{old} \leftarrow k_{all}$;
    /* Restart the process of querying and filtering */
12 return Ans;

Filtering-Querying Approach (FQ)

Given a location-dependent query $Q(LCons)$, the FQ approach works as follows: First, the LBSP invokes the *Pr-View* algorithm to determine the view of s, Algorithm 4. Second, the LBSP executes $Q(LCons)$ over the view of s and outputs a

Algorithm 6: Querying-Filtering Range

Input : Authorization Set $\mathcal{A}$, double $dist$, person s
Output: Set Ans

1 Initialize: $View_s \leftarrow \emptyset$, $Ans \leftarrow \emptyset$, $temp_{all} \leftarrow \emptyset$;
2 $View_s \leftarrow Pr\text{-}View(\mathcal{A}, s)$; `// Compute the view of` s
`/* Compute the users located within distance` $dist$ `from` s `*/`
3 $temp_{all} \leftarrow computeRange(dist, s)$;
`/* Select the users in` $temp_{all}$ `who are also in` $View_s$ `*/`
4 $Ans \leftarrow temp_{all} \cap View_s$ return Ans;

final result. Contrary to QF, since the evaluation of the location-dependent queries must take place on the filtered result, the LBSP cannot use the pre-computed materializations, i.e., $computeKNN(knn)$ and $computeRange(range)$. That is, the LBSP needs new primitives to execute the supported queries. We have identified two primitives:

- $computeD(p_s, p_u)$: It computes the distance between the physical positions of two given persons s and u.
- $sortByD(M)$: It sorts the elements of the given list M in ascending order by with respect to their distance value. M is a list of 2-element tuples, where the elements of each tuple are a person u and a distance d.

We use the well-known Haversine distance [Rob57] to implement the first primitive, and the merge sort algorithm to implement the second one. Algorithms 7 and 8 show the details for implementing FQ for kNN and range queries, respectively.

Algorithm 7: Filtering-Querying kNN

Input : Authorization Set $\mathcal{A}$, int k, person s
Output: Set Ans

1 Initialize: $View_s \leftarrow \emptyset$, $Dist \leftarrow \emptyset$, $View_{order} \leftarrow [\,]$, $Ans \leftarrow \emptyset$;
2 $View_s \leftarrow Pr\text{-}View(\mathcal{A}, s)$; `// Compute the view of` s
3 **foreach** *person* u *in* $View_s$ **do**
4 double $d \leftarrow computeD(p_s, p_u)$; `// Compute the distance between` s `and` u
5 add tuple $\langle u, d \rangle$ to $Dist$
6 $View_{order} \leftarrow sortByD(Dist)$; `// Order` $Dist$ `in ascending order`
7 $Ans \leftarrow$ select $topK(k, View_{order})$; `// Select the first` k `elements`
8 return Ans

Algorithm 8: Filtering-Querying Range

Input : Authorization Set $\mathcal{A}$, double $dist$, person s
Output: Set Ans

```
1  Initialize: View_s ← ∅, Dist ← ∅, neighbors, Ans ← ∅;
2  View_s ← Pr-View(A, s) ;                    // Compute the view of s
3  foreach person u in View_s do
       /* Compute the distance between s and u                          */
4      double d ← computeD(p_s, p_u);
5      add tuple ⟨u, d⟩ to Dist
6  foreach tuple t in Dis do
7      if t_distance ≤ dist then
           /* If the distance value stored in tuple t is greater or
              equal than dist, add the person stored in t to Ans        */
8          Ans ← t_pers;
9  return Ans
```

In the next subsection, we discuss the advantages and disadvantages of both approaches, QF and FQ.

Advantages and Disadvantages of QF and FQ

With QF, the LBSP can make use of the available index structures in the user database. However, in the case of bounded result-size queries, the LBSP may need to restart the query if the filtered result does not satisfy the initial constraints, cf. Example 4.13. With FQ, bounded result-size queries do not require restarts, Example 4.14. However, the evaluation of location-dependent queries must take place on the filtered result. Therefore, the LBSP cannot use the indexes structures of the user database to execute queries efficiently. Finally, with both approaches, the costs of updates, i.e., positions of persons and authorizations updates, only depend on the scalability and costs of updating the index structures used. The analysis of the impact of updates is beyond the scope of this dissertation. We refer the reader to [MS18] for information about the analysis and costs of updates.

QF and FQ are Sound

We assume that the algorithms used to evaluate a given location-dependent query are correct and complete with respect to the location constraint $LCons$. It remains to prove that the integration of these algorithms into the context of *mutual* authorizations is correct and complete. We only present the proofs for the algorithms

that support kNN queries, Lemmas 4.3 and 4.4, because the proofs for range queries can be done in the same manner following the proofs of Lemmas 4.3 and 4.4.

Lemma 4.3. *Let a set of authorizations $\mathcal{A}$ and a location constraint (k, s) of a kNN query be given, where k is an integer, and s is the query issuer. Algorithm 5, QF for kNN queries, is sound.*

Proof. If an algorithm is sound, it means that it is correct and complete, Definition 4.22. Let *Ans* be the result output by Algorithm 5. We first prove that the algorithm is correct with respect to Definition 4.20. We assume that the service used by Algorithm 5 to compute a given kNN query, $computeKNN(k_{all}, s)$, where $k_{all} \geq k$ (Line 4), is correct with respect to the location constraint (k_{all}, s). If a person u is in *Ans*, u is in $temp_{visible}$ (Line 8). If u is in $temp_{visible}$, u is $temp_{all}$ and u is in $View_s$ (Line 6). $View_s$ is the output of $Pr\text{-}View(\mathcal{A}, s)$, so s is authorized to read the physical position of u, Lemma 4.2. Next, $u \in rst(ACons_{\mathcal{A},s}, U)$, where $ACons_{\mathcal{A},s}$ is an authorization constraint. Since u is in $temp_{all}$, u is in $computeKNN(k_{all}, s)$. $computeKNN(k_{all}, s)$ is correct. Therefore, u satisfies the location constraint (k_{all}, s) with respect to U. The size of $temp_{visible}$ is greater or equal than k. The function *topK* selects the k first elements from $temp_{visible}$. Then, $u \in rst((k, s), rst(ACons_{\mathcal{A},s}, U))$. Consequently, Algorithm 5 is correct. Now, we prove that Algorithm 5 is complete with respect to Definition 4.21. Consider a person u with $u \in rst(ACons_{\mathcal{A},s}, U) \wedge u \in rst((k, s), rst(\ ACons_{\mathcal{A},s}, U))$. Since u satisfies the authorization constraint with respect to U, u is in $View_s$. $View_s = rst(ACons_{\mathcal{A},s},\ U)$, Definition 4.17. Then u is in $rest((k, s), View_s)$ and $rest((k, s), View_s) \subseteq rest((\ k_{all},\ s\ ), U)$. Therefore, u is in $rest((k_{all}, s), U)$. Next, we know that $computeKNN(k_{all}, s)$ is complete. Then u is in $temp_{all}$ and u is in $temp_{visible}$. Because $u \in rst\,((k, s), View_s)$, u is in *topK* and u is *Ans*. That is, Algorithm 5 is complete; consequently, it is sound. □

Lemma 4.4. *Let a set of authorizations $\mathcal{A}$ and a location constraint (k, s) of a kNN query be given, where k is an integer, and s is the query issuer. Algorithm 7, FQ for kNN queries, is sound.*

Proof. If an algorithm is sound, it means that it is correct and complete, Definition 4.22. Let *Ans* be the output of Algorithm 7. We first prove that Algorithm 7 is correct with respect to Definition 4.20. If a person u is in *Ans*, u is in the list $View_{order}$ (Line 6). Then, there is a 2-element tuple $\langle u, d\rangle$ in *Dist*. That is, u is in $View_s$. $View_s$ is the output of $Pr\text{-}View(\mathcal{A}, s)$. So, s is authorized to read the position of u, Lemma 4.2. Then, $u \in rst(ACons_{\mathcal{A},s}, U)$, where $ACons_{\mathcal{A},s}$ is an authorization constraint. Algorithm 7 uses the primitives *computeD*, *sortByD* and *topK* to compute the k-nearest neighbors of a given person s. We assume

that the combination of these primitives to compute a given kNN query is correct with respect to the location constraint (k, s). Since these primitives compute the result using as input the set $View_s$ (Line 3), $u \in rst((k, s), View_s)$, and $View_s = rst(ACons_{\mathcal{A},s}, U)$. That is, Algorithm 7 is correct. We now prove that Algorithm 7 is complete with respect to Definition 4.21. Consider a person u with $u \in rst(ACons_{\mathcal{A},s}, U) \wedge u \in rst((k, s), rst(ACons_{\mathcal{A},s}, U))$. Since u satisfies the authorization constraint with respect to U, $u \in View_s$. Therefore, there is a 2-element tuple $\langle u, d\rangle$ in $Dist$. Then, u is in $View_{order}$, Line 7. Since u satisfies the location constraint (k, s) with respect to the set $View_s$, u is in $topK(k, View_{order})$. Then u is in Ans. That is, Algorithm 7 is complete; consequently, it is is sound. □

Time Complexity analysis

A complexity analysis is helpful to predict the behavior of FQ and QF, and to facilitate meaningful comparisons. An average complexity analysis depends on the internal behavior of the database, which is specific to the product and is not openly available. Furthermore, if there are changes in the system settings, the average analysis is void. Therefore, our complexity analysis targets at the worst case, which offers stronger guarantees.

In our time complexity analysis, we focus on our FQ and QF approaches for kNN queries. The analysis of the approaches for range queries can be done in the same way, as we will explain.

Recall that to fulfill a given location constraint (k, s) of a kNN query, Algorithm 5 uses an estimation function *estimateK*. This function estimates a value $k_{all} \geq k$ for a given k, such that after computing the k_{all}-nearest neighbors of s and filtering the result based on $View_s$, the filtered result satisfies the original constraint (k, s). Let k_{real} be the value of k_{all} used by Algorithm 5 to compute the final output, i.e., k_{real} is equal to the value k_{all} of the last run of Algorithm 5. Let further be $\delta = k_{real} - k$.

For the complexity analysis of Algorithm 5, we assume that *estimateK* computes the value k_{real} in the first run, i.e., no restarts are needed. We discuss this assumption in the next subsection, where we compare both approaches.

Lemma 4.5. *Let the number of persons n, a kNN query knn=(k,s), the view size of the query issuer, s, $|View_s|$, and a set of authorizations $\mathcal{A}$, be given. The time complexity of QF with no restarts is*

$$T_C = \mathcal{O}(n + (k + \delta) \cdot |View_s|) + \mathcal{O}(\mathcal{A}) \tag{4.1}$$

Proof. The following steps are required to compute a given kNN query with the *querying-filtering* approach, with no restarts:

(1) Compute the view $View_s$ of the query issuer s. We use $\mathcal{O}(\mathcal{A})$ to denote the complexity of this step.

(2) Search the $(k+\delta)$-nearest neighbors in the user database. The complexity of a kNN query using R-tree indexes is $\mathcal{O}(n)$ [MC15]. We validated through initial experiments that this complexity applies to the praxis.

(3) Filter the result by checking for each person returned in the second step if the person is in the view of s, $View_s$. The complexity of this step is $\mathcal{O}((k+\delta) \cdot |View_s|)$.

Consequently, the time complexity of executing a kNN query with the *querying-filtering* approach is $T_C = \mathcal{O}(n + (k+\delta) \cdot |View_s|) + \mathcal{O}(\mathcal{A})$. □

Lemma 4.6. *Let the number of persons n, a kNN query $knn = (k, s)$, the size of the view of the query issuer s, $|View_s|$, and the set of authorizations $\mathcal{A}$ be given. The time complexity of FQ is*

$$T_C = \mathcal{O}\big(|View_s| \cdot \log(n) + |View_s| + |View_s| \cdot \log(|View_s|) + k\big) + \mathcal{O}(\mathcal{A}) \quad (4.2)$$

Proof. The following steps are required to compute a given kNN query with the *filtering-querying* approach:

(1) Compute the view of the query issuer s, $View_s$. $\mathcal{O}(\mathcal{A})$ denotes the complexity of this step.

(2) Look up in the user database to obtain the physical position of each person in the view $View_s$. This has a complexity of $\mathcal{O}(|View_s| \cdot \log(n))$.

(3) Compute the distance between the querying user s and each of the persons in the view $View_s$. The complexity of this step is $\mathcal{O}(|View_s|)$.

(4) Order the persons in the view $View_s$ by distance to the querying user s in ascending order. The order is done using the merge sort algorithm. The complexity of this step is $\mathcal{O}(|View_s| \cdot \log(|View_s|))$.

(5) Select the k first persons. This has a complexity of $\mathcal{O}(k)$.

Consequently, the time complexity of executing a kNN query with the *filtering-querying* approach is T_C=$\mathcal{O}(|View_s| \cdot \log(n) + |View_s| + |View_s| \cdot \log(|View_s|) + k) + \mathcal{O}(\mathcal{A})$. □

We note that, since $\forall x > 0, n > 0 : x > x \cdot log(n)$, Equation 4.2 can be further simplified to $T_C = \mathcal{O}\big(|View_s| + k\big) + \mathcal{O}(\mathcal{A})$. However, to allow a more accurate comparison of both approaches, in the next subsection, we do not simplify it.

Comparison of the QF and FQ Approaches

One needs to compare the complexity of QF and FQ approach, and find their intersection points to decide which approach is better to answer a given kNN query:

$$\mathcal{O}(n + (k+\delta) \cdot |View_s|) + \cancel{\mathcal{O}(\mathcal{A})} = \mathcal{O}\big(|View_s| \cdot \log(n) + |View_s| + |View_s| \cdot \log(|View_s|) + k\big) + \cancel{\mathcal{O}(\mathcal{A})} \quad (4.3)$$

In order to solve Equation (4.3) for $|View_s|$, we first omit the big-O notation:

$$n + (k+\delta) \cdot |View_s| = |View_s| \cdot \log(n) + |View_s| + |View_s| \cdot \log(|View_s|) + k \quad (4.4)$$

Solving Equation (4.4) for $|View_s|$ yields Equation (4.5). For given values of n, k and δ, Equation (4.5) is the size of the view so that the time complexity in the worst case is equal. We refer to this size of the view as $View_{equal}$. $\mathcal{W}$ in Equation (4.5) is the Lambert-W function [SZFJ09].

$$View_{equal}(n, k, \delta) = \frac{n \cdot \ln(2) - k \cdot \ln(2)}{\mathcal{W}(2^{1-k-\delta} \cdot n \cdot (n-k) \cdot \ln(2))} \quad (4.5)$$

We now analyze Equation (4.5) with the best case scenario for QF. The best scenario is the one in which the nearest neighbors of s are the persons whose positions s is allowed to read, i.e., δ=0. Equation 4.5 depends on the parameters: n, k and δ. To further simplify it, similarly to other approaches [YPBV14, HAK00], we set the parameters k of the kNN query to 20, and δ=0. Then $View_{equal}$ only depends on the number of persons n.

$$View_{equal}(n, 20, 0) = \frac{n \cdot \ln(2) - k \cdot \ln(2)}{\mathcal{W}(2^{-19} \cdot n \cdot (n-20) \cdot \ln(2))} \quad (4.6)$$

Figure 4.5 plots the QF and FQ approaches, for $k = 20$ and $\delta = 0$. The x-axis is the number of persons n, the y-axis the size of the view $|View_s|$ of the query

issuer s and the z-axis the time complexity T_C. It shows the intersection points of both approaches. Given an intersection point and its corresponding number of persons n, Equation (4.6) yields the size of its view. We conclude that, for a given n, if $|View_s| < View_{equal}$, the time complexity of FQ is smaller than that of QF, and vice versa. In Table 4.2, using Equation (4.6), we list the intersection points of QF and FQ, for different numbers of persons n. For instance, if $n = 2000$, $View_{equal} \approx 1014.31$. Then, for $n = 2000$, if the size of the view of the query issuer is smaller than approximately 1014.31, FQ performs better than QF.

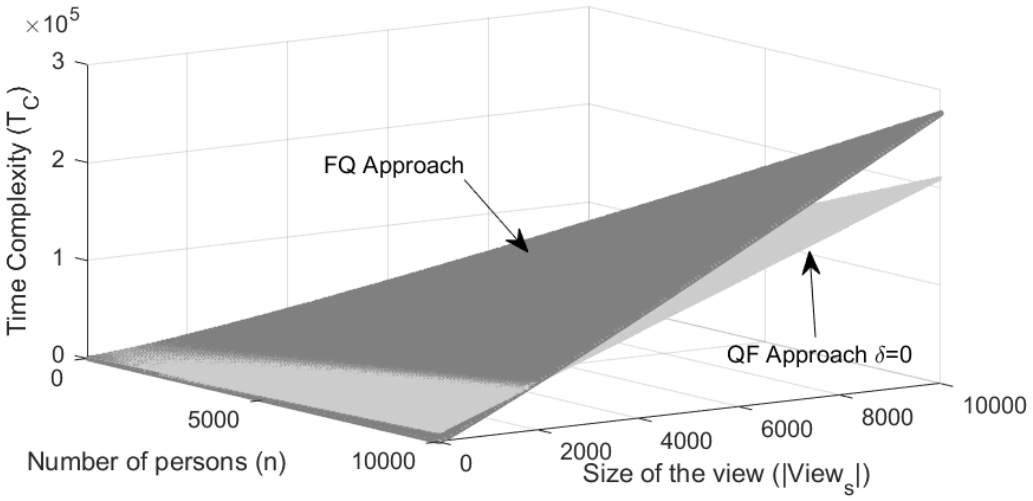


Figure 4.5: Complexity of the QF and FQ Approaches– knn Query ($k = 20, \delta = 0$)

n	$View_{equal} \approx$
2000	1014.3096
4000	1231.4594
10000	1920.9585
20000	2935.2272
40000	4709.5727
100000	9267.9800
317080	23032.341
3000000	1520464307

Table 4.2: Intersection points of QF and FQ — knn Query ($k = 20, \delta = 0$)

Analysis of the Comparison of the QF and FQ Approaches in Real Scenarios

So far, the plot in Figure 4.5 and the values in Table 4.2 correspond to the best case scenario for QF, i.e., $\delta = 0$. We now explain why a focus on this case is sufficient.

Let us consider real scenarios such as online social networks like Orkut and LiveJournal. The number of connections that a person s has in these networks is the number of persons that have declared to have a relationship with s, e.g., friends, colleagues. This translates to our authorization model as the size of the view of s. In [LS15], the authors found that considering about 3 million nodes, the average number of connections of a person in Orkut and LiveJournal is 223.99 and 520.04, respectively. In DBLP with 317080 nodes, the average number of connections is 64.98 [LS15]. This suggests that the size of the view of a given person increases monotonically with the number of persons. Analogously, Table 4.2 reveals that $View_{equal}$ grows monotonically with the number of persons n. We can also observe that, if $n = 2000$, $View_{equal}$ is already greater than the average number of connections for 3 million persons in real scenarios. For n equal to 3 million, $View_{equal}$=152046. This indicates that the size of the view of a given person in real scenarios is smaller than $View_{equal}$ for a given n. This implies that FQ performs better than QF even in the best case scenario of QF. So we do not dwell into the behavior of QF with restarts.

The analysis of the approaches for range queries can be done in the same way. Range queries are unbounded result-size queries. QF for range queries does not need any restart. In the analysis of QF for kNN queries, we did not consider restarts because we assumed *estimateK* to compute k_{real} in the first run. Therefore, the skeleton structure of the complexity analysis is the same for both type of queries. For these reasons, we omit this part.

4.4 Experiments

This section presents an experimental analysis of the impact of *mutual* authorizations on the performance when answering an access request, and an experimental validation of the complexity analysis of QF and FQ.

4.4.1 Impact of Mutual Authorizations

This subsection compares the performance of *mutual* authorizations with that of *positive* and *negative* authorizations. Specifically, we study the performance when

answering an access request, i.e., deciding whether an access request is authorized or not.

Evaluation Scenarios

In Sections 4.1 and 4.2, we have shown two different implementations of *mutual* authorizations: (1) a basic implementation, which can be used in scenarios with resources with the same sensitivity degree, e.g., location-based services, and (2) an extended implementation, trust-based authorizations, which can be used in scenarios with resources with different degrees of sensitivity, e.g., health records. Both implementations behave differently regarding performance. The reason is that the extended implementation has to evaluate the sensitivity of the resources and the trust value of the users to answer a given access request. Hence, our experiments consider both scenarios, i.e., scenarios with resources with the same and with different sensitivity.

Next, given an access request $Req = \langle s,\ \ op, res_u \rangle$, deciding whether Req is authorized or not involves only the authorizations assigned by u to s and the ones assigned by s to u. That is, this decision is independent of all other authorizations in the system. Therefore, to evaluate the performance when deciding whether Req is authorized or not, we alternate the grants of the authorizations assigned by u to s, and the ones assigned by s to u. As a result of this alteration, and considering the possible scenarios regarding resources, we identify and consider the following cases in our experimental analysis:

- *Scenario 1 - Resources with the same degree of sensitivity:* Table 4.3 shows all possible grants of the authorizations that two users u and s can assign to each other. Based on the assigned authorizations, this scenario has two cases: *S1-Case 1* $\langle Allow/Deny, * \rangle$ and *S1-Case 2* $\langle Mutual, * \rangle$. Each case refers to answering the access request Req given the authorizations with the grants indicated in the column headers and row headers of Table 4.3. For instance, *S1-Case 1* $\langle Allow/Deny, * \rangle$ refers to answering the access request Req given that u has assigned a *positive* or *negative* authorization to s. s assigns an authorization with any grant to u.

- *Scenario 2 - Resources with different sensitivity degree:* Table 4.4 shows all possible trust-based authorizations that two users u and s can assign to each other. Based on the assigned trust-based authorizations, this scenario considers three cases: *S2-Case 1* $\langle$ *Trust Allow/Deny*, $* \rangle$, *S2-Case 2.1* $\langle Trust\ Mutual, Allow/Deny \rangle$, and *S2-Case 2.2* $\langle$ *Trust Mutual*, $Mutual \rangle$. Similar to the previous scenario, each case refers to answering the access

request *Req* given the trust-based authorizations indicated in the column headers and row headers of Table 4.4.

u to s / s to u	Allow	Deny	Mutual
Allow	S1-Case 1 $\langle Allow/Deny, * \rangle$		S2-Case 2 $\langle Mutual, * \rangle$
Deny			
Mutual			

Table 4.3: Experiment Cases - Resources with the same Degree of Sensitivity

u to s / s to u	Trust Allow	Trust Deny	Trust Mutual
Trust Allow	S1-Case 1 $\langle Trust\ Allow/Deny, * \rangle$		S2-Case 2.1 $\langle Trust\ Mutual, Allow/Deny \rangle$
Trust Deny			
Trust Mutual			S2-Case 2.2 $\langle Trust\ Mutual, Mutual \rangle$

Table 4.4: Experiment Cases – Resources with Different Degrees of Sensitivity

In scenarios where resources have different degrees of sensitivity (Scenario 2), given an access request $Req = \langle s,\ op, res_u \rangle$, if u has assigned a trust-based *mutual* authorization to s, one has to differentiate between the cases S2-Case 2.1 $\langle Trust\ Mutual, Allow/Deny \rangle$ and S2-Case 2.2 $\langle Trust\ Mutual, Mutual \rangle$. The reason is that in the scenario S2-Case 2.2 $\langle Trust\ Mutual, Mutual \rangle$, answering the access request *Req* also requires to evaluate the sensitivity of the resources and the trust values of the users involved in the authorizations. This is not needed if s has assigned a trust-based *positive* or *negative* authorization to u. Regarding *mutual* authorizations in scenario 1, i.e., S1-Case 2 $\langle Mutual, * \rangle$, this differentiation is not needed either because scenario 1 considers resources with the same sensitivity. Our implementation of *mutual* authorizations does not compute the sensitivity of resources and the trust values of users.

Experiment Setup

We use the dataset Epinion, a network dataset of users connected with directed edges. The network has 131828 users and 841372 edges. The network has 30.9% reciprocal edges, i.e., the proportion of edges for which an edge in the opposite

direction exists. We use the network dataset to build a labeled directed graph G as follows:

1. The users represent the vertices of G, and the edges of the network represent the labeled directed edges of G. An edge (u, s) indicates that user u has assigned an authorization to subject s.

2. We label the non-reciprocal edges with labels *allow*, *deny* , or *mutual*. The labels are selected at random. The label of an edge (u, s) represents the grant of the authorization that u has assigned to s.

3. For each pair of reciprocal edges of the form (u, s) and (s, u), we assign to one of the edges a label *mutual*, and to the remaining one a label *allow* or *mutual*, selected at random.

4. We assume that two vertices u, s that are not connected through an edge (u, s) indicates that u has assigned a *deny* authorizations to s. This is because we adhere to the *default-deny* principle, i.e., any access not explicitly assigned a *positive* or *mutual* authorization is denied.

5. To evaluate scenario 2, i.e., resources with different sensitivity degree, we assign at random to each labeled edge a trust value from the set $T = \{0.1, 0.2, \cdots 1\}$. We also assign at random to each of the 131828 users a trust value from T. Additionally, we assume that each user owns a resource, and we assign a sensitivity value (from 1 to 5) to each resource at random.

Query Sample

For each of the cases in scenarios 1 and 2, we select a sample of 1000 access requests for the evaluation. This is, for *S1-Case 1* $\langle Allow/Deny, * \rangle$, we select 1000 *allow* or *deny* edges. Then for each selected edge (u, s), we measure the time needed to decide whether the access request $\langle s,\ op, res_u \rangle$ is authorized or not. Next, for *S1-Case 2* $\langle\ Mutual, *\ \rangle$, we select 1000 *mutual* edges and create and evaluate the access request as in the previous case. We repeat the procedure for each case in scenario 2. In total, we run our experiments for 5000 access requests.

Experimental Results

Figures 4.6(a) and (b) show the average time required to answer an access request, for the cases in scenarios 1 and 2, respectively. The x-axis represents the different cases in each scenario, and the y-axis indicates the average time to the decision.

Discussion - Scenario 1: For scenario 1, we observe in Figure 4.6(a) that answering an access request in *S1-Case 1* $\langle Allow/Deny, * \rangle$ is faster (on average 1.6 times faster) than in *S1-Case 2* $\langle Mutual, * \rangle$. The reason is that in S1-Case 2, given an access request $\langle s, \ op, res_u \rangle$, one has to consider not only the authorizations assigned by u to s but also the ones assigned by s to u.

Discussion - Scenario 2: Regarding scenario 2, we observe in Figure 4.6(b) that answering an access request in scenario *S2-Case 1* $\langle Trust\ Allow/Deny, * \rangle$ is faster than in scenarios *S2-Case 2.1* $\langle Trust\ Mutual, Deny \rangle$ and *S2-Case 2.2* $\langle\ Trust\ Mutual, Allow/Mutual\ \rangle$. Since we omit the trust evaluation for trust-based *positive* and *negative* authorizations, scenario *S2-Case 1* $\langle Trust\ Allow/Deny, * \rangle$ is the same as the first one of scenario 1, i.e., *S1-Case 1* $\langle Allow/Deny, * \rangle$. From the *mutual* authorizations cases, i.e., *S2-Case 2.1* and *S2-Case 2.2*, scenario *S2-Case 2.1* requires less time to answer an access request (1.4 faster than in S2-Case 2.2). The reason is that scenario S2-Case 2.1 considers that a *positive* and a *negative* authorization has been assigned from s to u. That is, the trust and sensitivity evaluations are not needed. Finally, comparing the performance of both scenarios, we can observe that scenario 1 performs better than scenario 2. The difference in the performance of both scenarios is expected since, with scenario 1, it is not necessary to evaluate the sensitivity of the resources and the trust values of the users. We conclude that the decrease in performance when using *mutual* authorizations is within an acceptable range while offering users more flexibility when controlling access to their resources.

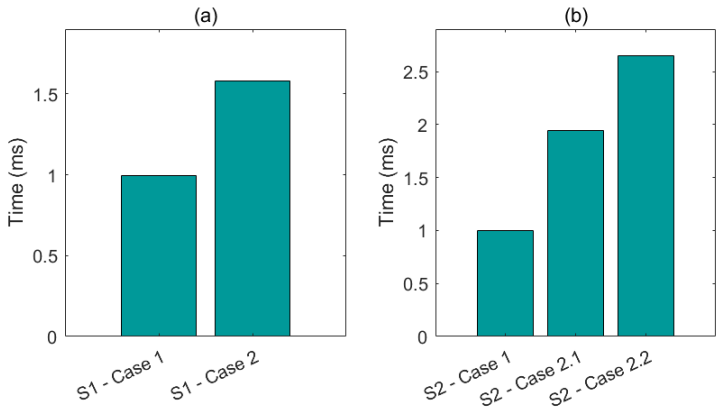


Figure 4.6: Access Request Decision – Performance Evaluation

4.4.2 Experimental Validation of the Complexity Analysis of QF and FQ

Our complexity analysis of our QF and FQ approaches, Subsection 4.3.5, already allows us to compare both approaches. However, since that analysis covers the worst case, experimental results are needed to validate and to determine how far the worst case deviates from the concrete behavior of individual queries. To implement the QF approach, we use the R-tree index structure from Oracle, and the remaining implementation was done in Java.

Experiment Setup

In our complextiy analysis, we found that the parameters that affect the performance of FQ and QF are: the number of persons n, the size of the view of a given person s, i.e, $|View_s|$, the parameter k of the kNN query, and the value $\delta = k_{real} - k$. Similarly to the complexity analysis, for simplicity, we set δ to 0 and k to 20. We set the remaining parameters, n and $|View_s|$, as follows:

Number of persons n: We create a dataset with 317080 persons. This number is the size of the DBLP dataset. To assign a position to each person, we choose a random physical position from the Tokyo dataset [YZZY15], which contains 573703 real check-ins, i.e., positions.

Size of the view of a person s, $|View_s|$, and query sample: We chose 1500 persons at random from the 317080 persons. We assigned the authorizations so that we have 15 classes of different sized views (from 50 to 40000). For each class, we have 100 persons with the respective view size, i.e., 1500 queries in 15 different classes in total.

Experiment Results

Figure 4.7 shows a comparison of the query-processing times for kNN queries with QF and FQ. We grouped the persons of our query sample by the size of their view and plotted the query-processing time. We excluded the database connection time and network communication time from the run time reported. The dotted line in Figure 4.7(a) represents the average size of the view in DBLP with 317080 nodes, i.e., 64.98 [LS15]. The dashed line in Figure 4.7(c) is the size of the view for which the performance of both approaches is equal, i.e., $View_{equal} \approx 23032,3$.

Discussion. For real scenarios, i.e., scenarios in which the size of the view is equal to 64.98, FQ performs better than QF for all queries. These results are in line with our complexity analysis, and one may interpret them as an indication that

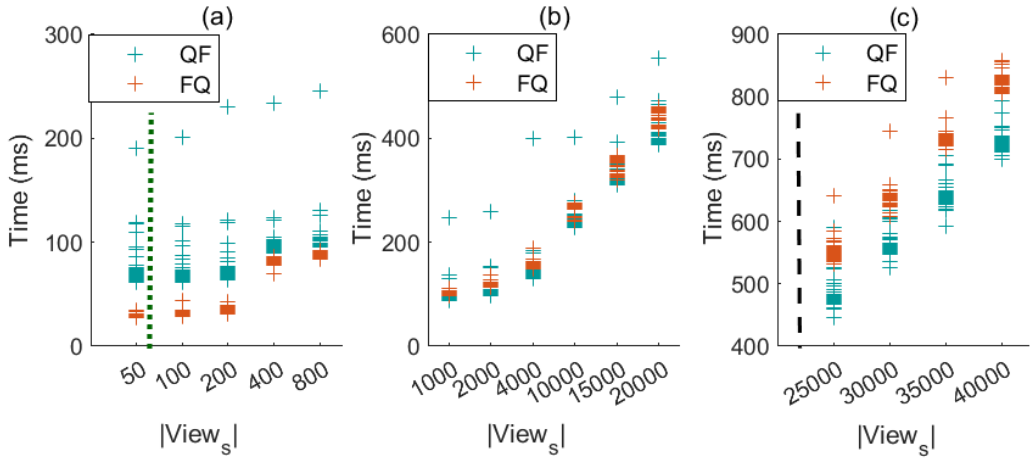


Figure 4.7: Comparison of the QF and FQ Approaches for kNN Queries

our analysis also holds for the average case. These findings remain correct for a size of the view of up to 800, which is higher than the highest average size of the view in real scenarios, i.e., 520. However, as Figure 4.7(b) shows, with a view size between 1000 up to 20000, the processing times of most of the queries with QF are lower than that of the ones with FQ, in contrast to our complexity analysis. This can be expected since our analysis has focused on the worst case. In Figure 4.7(c), we observe that the processing times of most of the queries with QF, or all the queries in the case of the last two groups, i.e., 35000 and 40000, are lower than that of the ones with FQ. These results indicate that for a view with size greater than the value $View_{equal}$ (dashed line) our complexity analysis holds even for the average case.

4.5 Related Work

A considerable amount of literature has been published on access control models and several models exist such as role-based access control model (RBAC) [SCFY96], Task RBAC [OP03], ABAC [YT05], and Relation-based access control model [GZC08]. The difference to ours is that they only consider the grants *deny* and *allow*. These two grants are not enough to capture *mutual* authorizations.

The concept of reciprocity in general, however, is not new, and it has played an important role in different fields like psychology, economics, sociology and game theory [SBC09, FF06, FFG02, Rab93]. To the best of our knowledge, we are the first to incorporate the reciprocity concept into access control models.

Besides access control models, encryption techniques have also been studied to achieve data confidentiality[VFJ+10, BSW07]. The main idea is to encrypt the resources and to enforce access control with the decryption keys assigned to the users. The approach in [VFJ+10] encrypts the data with different keys, depending on the authorizations to be enforced. After encryption, the decryption keys are given to users based on their access privileges. In the approach proposed in [BSW07], the authors encrypt the data together with an access structure. The access structure represents a set of attributes along with values that the users must fulfill to access the data. The decryption key that the authorized users received is generated based on their attribute values. Users can decrypt a ciphertext c if their decryption key matches the attribute values of the access structure associated with c. This work does not consider mutual access to resources because the decryption keys are generated and distributed without considering reciprocity.

There is another set of works that focuses on formalizing and verifying the authorization constraints in RBAC and its extensions. These proposals use Colored Petri-Nets [SMJG05] or the Unified Modeling Language UML [RLFK04]. They focus on (1) introducing formal techniques to verify the design and consistency of authorizations on RBAC models, i.e., model checking, and on (2) providing a graphical representation of the authorizations, as visualizations. These works are confined to *positive* and *negative* authorizations as well. That is because these works are based on access control models existing at that time.

4.6 Summary

Reciprocity is a powerful determinant of human behavior. However, none of the existing access control models explicitly supports it. In this chapter, we have proposed a new type of authorization, called *mutual*. *Mutual* authorizations allow users to grant access to their resources to users that allow them the same. We have extended the attribute-based access control model to incorporate *mutual* authorizations and have formally defined their syntax and semantics. At first, to focus on reciprocity, we have considered only resources with the same sensitivity. Next, we have extended our model of *mutual* authorizations to support a more general case, i.e., resources with different sensitivity values. Our generalization lets owners assign the degree of sensitivity of their resources themselves, but their peers can evaluate such an assignment a posteriori. Based on the evaluation by others, each user receives a trust value. When using *mutual* authorizations, users can then express the minimum level of trust their peers should have to exchange resources with them. We call this extension trust-based authorizations. Next, since the result of a given service in the presence of *mutual* authorizations is not obvious,

we have studied this as well. To this end, we have selected location-based services as a use case for the deployment of *mutual* authorizations, and we have proposed two approaches. A complexity analysis tells us when each approach is better. We have conducted experiments to evaluate the impact of *mutual* authorizations on the performance of answering an access request, and to validate our complexity analysis.

5 Secure Outsourcing of Graph-Structured Data

In this chapter and the next one (Chapter 6), we focus on the authorization mechanism—one of the components of access control systems, which carries out the enforcement of the authorizations. Specifically, in this chapter, we tackle the distrust of users towards the service providers from a data outsourcing perspective.

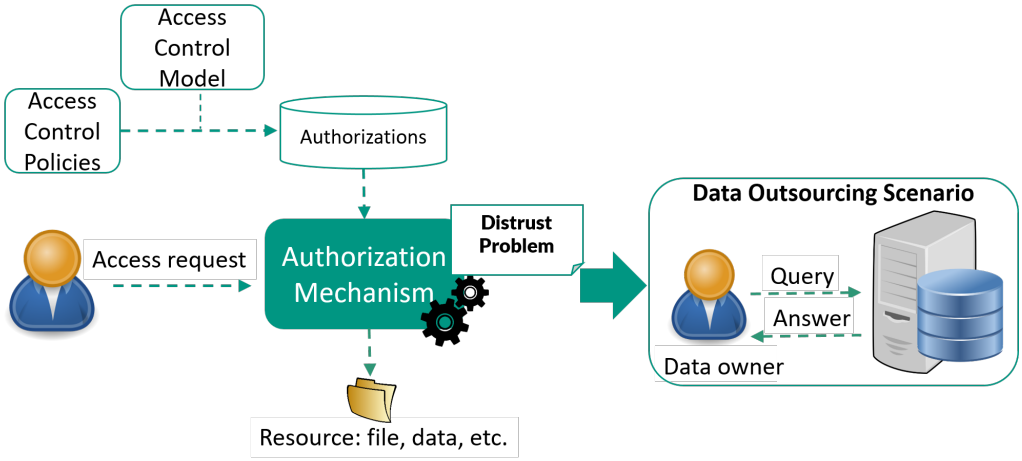


Figure 5.1: Access Control Mechanism - Distrust Problem

Outsourcing databases to a third-party service provider has become ubiquitous. While economic and organizational advantages are obvious, database outsourcing is challenging concerning data secrecy. Databases contain sensitive information that needs to be protected against adversaries, including the service provider. Even if an adversary, including the service provider, accesses the data, the adversary should not be able to learn any information from the accessed data. In this chapter, we address this problem for graph-structured data[3].

Another significant trend is that a broad range of real-world datasets exhibits a graph structure. Furthermore, many real graphs, such as the email network or the

[3]An earlier version of this chapter was published in [SEGB20].

Web, follow a scale-free power-law distribution [BA99]. At the same time, these graphs often contain sensitive information. In this chapter, we focus on graphs with this characteristic.

Graph-structured data store information about the nodes and their relationships (edges). Especially in this kind of data, these relationships are always present, which is not the case for other data structures. An adversary can use the information stored in a node and its relationships to identify an individual in a graph. Therefore, approaches for secure storage of graph-structured data should protect against leaking this kind of information. Next, there have to be provable secrecy guarantees, which requires a rigid definition of secrecy, i.e., to define the type of adversaries one is dealing with, the desired secrecy guarantees, and the information that one accepts to leak. At the same time, the approaches should not do away with the advantages of database outsourcing. In particular, query processing should take place on the server as much as possible.

We are not aware of any previous work on secure storage featuring a cost model for query processing. However, a cost model is needed to have a good understanding of the expected performance of query processing, to facilitate comparisons between alternatives, assuming that the alternatives also have a cost model, and to predict the impact of parameter changes. Furthermore, any query optimizer, which is an integral component of a modern database-management system, depends on cost models to come up with good query-executions plans [MPS99, MVW98, GTK01].

To summarize, there are two requirements that a secure storage scheme for graph-structured data should fulfill:

(R1) An adversary, including the service provider, must not be able to learn any useful information from the outsourced graph database, except for some predefined information. These secrecy guarantees offered by the scheme must be provable (i.e., secrecy).

(R2) The scheme should support a broad range of queries. It should do so efficiently, with controlled effort, and most of the work should be done at the server side. To quantify this, a performance model is needed.

The first requirement calls for a rigid definition of secrecy. Here, we consider adversaries who have access to the graph stored a the service provider and can observe query executions over time, i.e., the encrypted queries issued and their encrypted results. Our secrecy notion explicitly states all possible leakages which could result from our specified adversary. Since existing secrecy notions consider different secrecy guarantees, as we will discuss in Section 5.5, and deal with dif-

ferent types of adversaries, we propose a new one, i.e., formalize the notion just sketched. Section 5.1 formally introduces our secrecy notion.

We propose, in Section 5.2, a bucketization approach for secure storage of graph-structured data that meets our requirements. It has turned out that subtle design decisions have a significant impact. For example, it makes a big difference regarding secrecy, whether we partition nodes into buckets instead of edges. This is because partitioning nodes could leak information on the graph structure, as we will explain. Our approach works as follows: First, it encrypts the labels of the nodes and edges to protect them against deterministic chosen-plaintext attacks, i.e., an adversary cannot learn any useful information from the encrypted nodes and edges. Second, it uses a bucketization technique to protect against frequency attacks, i.e., an adversary cannot learn secret information based on the frequency of the ciphertexts. In our scenario, the frequency represents the degree of a node. While our approach works for all types of graph queries in principle, we focus on neighbor and adjacency queries. These queries are essential information needs regarding graphs [MP10]. We then describe the specifics for these queries, such as division of work between client and server.

Section 5.3 presents a performance model for query processing on scale-free graphs. The performance model consists of a number-of-buckets model, which estimates the number of buckets obtained when applying our bucketization approach, and a query-cost model.

In Section 5.4, we conduct systematic experiments both on synthetic and on real datasets. Our experiments validate the accuracy of our estimation model and demonstrate the efficiency of the proposed bucketization technique.

Secure database storage has been widely studied. However, existing techniques [HMT04, ABG+05, HIM05] either cannot be applied to graph-structured data or do not cover both requirements R1 and R2. Finally, in Section 5.5, we review existing approaches in the area and explain more in-depth why one cannot apply these approaches in our scenario.

5.1 Our Secrecy Notion

We start by introducing, in Subsection 5.1.1, the require notation for this Chapter. Subsection 5.1.2 presents our secrecy notion for graph-structured data.

5.1.1 Notation

Let $G = (V, E)$ be a graph, where V is a set of nodes and $E \subseteq V \times V$ is a relation between nodes. $|V|$ and $|E|$ are the number of nodes and edges, respectively. $\mathcal{G}$ is the set of all graphs. Without loss of generality, we assume that the relationships between the nodes are directed. An undirected edge can be represented by two directed edges. The size of a given graph G, dubbed $size(G)$, is a tuple $(|V|, |E|)$ that contains the number of nodes and the number of edges of G. The degree of a node $u \in V$, $deg(u)$, is the number of outgoing edges of u. The multiset of degrees of a given graph G, $Deg(G)$, is the multiset that contains the degree of each node $u \in V$. A neighbor query $Q_{Neighbor}(G, u)$ takes as input a graph G and a node $u \in V$, and returns the set of all nodes adjacent to u in G:

$$Q_{Neigbhor}(G, u) = \{v \in V \mid (u, v) \in E\}$$

An adjacency query $Q_{Adjacency}(G, u, v)$ takes as input a graph G and a pair of nodes $u, v \in V$, and verifies whether node u is adjacent to node v:

$$Q_{Adjacency}(G, u, v) = true \text{ iff } (u, v) \in E$$

A query history of a graph G, qH_G, is a list of n queries $qH_G = [q_1, ..., q_n]$ over G, where q_1 is the earliest query in the list, and $q_1, ..., q_n$ either are either neighbor or adjacency queries. Given a value x and a multiset of values $\mathcal{V}$, the frequency of x in $\mathcal{V}$ is the number of occurrences of x in $\mathcal{V}$.

5.1.2 Our Secrecy Notion for Graph-structured Data

Here, we describe the secrecy notion we target. The goal is to build a secrecy notion with the following characteristic: If an algorithm used to secure a given graph fulfills this notion, it is guaranteed that it only leaks the information formally stated, as we will explain. Allowing some leakage is standard with state-of-the-art secrecy notions, especially in the area of searchable encryption [CGKO11, LWW+10, BBHJ11, CYW+11, WRD+17, MKNK15].

Let a data structure, dubbed ds, be any structure that can be implemented in a database. This definition – naturally – is somewhat vague. For instance, something can be two separate data structures, or one could count them as one. However, this definition is enough for our purposes; we just need it to bring some rigidity to the explanations that follow. We will provide concrete instantiations, later on, doing away with this vagueness.

Definition 5.1: Graph-secretization Algorithm

Given a graph G, a **graph-secretization algorithm** τ is an algorithm that takes as input G and transforms it to a list of d data structures, $[ds_1, ..., ds_d]$, so that some information from G is kept secret. We call the result of applying τ to G, the **transformed graph** $transformed_G$.

If an algorithm τ complies with a secrecy definition, such secrecy definition defines which information τ must keep secret and how it should do so.

Definition 5.2: Adversary

An **adversary** $\mathcal{A}$ is a malicious user who has access to the transformed graph $transformed_G$ and can observe query executions over it.

We now define the information leakage that we are willing to accept. We consider four leakages. The first two, called access and search patterns, are related to the execution of queries, Definitions 5.5 and 5.6 respectively, and the second two, called the size of G and multiset of degrees (Subsection 5.1.1), are related to the graph itself.

For a given graph G, and a query history qH_G with n queries, the access pattern is a list of n elements that contains information about G. In concrete, if the i-th query is a neighbor query, the i-th element in the list is a lower, x, and upper bound, y, on the degree of the queried node. If the i-th query is an adjacency query, then the i-th element in the list is a Boolean, stating whether the queried edge exists in G or not.

Definition 5.3: Neighbor-access Pattern

Given a graph G and a neighbor query $Q_{Neighbor}(G, u)$, a **neighbor-access pattern** $\alpha(Q_{Neighbor}(G, u))$ of $Q_{Neighbor}(G, u)$ is a tuple (x, y) such that $x \leq deg(u) \leq y$.

Definition 5.4: Adjacency-access Pattern

Given a graph G and an adjacency query $Q_{Adjacency}(G, u, v)$, an **adjacency-access pattern** $\alpha(Q_{Adjacency}(G, u, v))$ of $Q_{Adjacency}(G, u, v)$ is a Boolean which takes the value $true$ iff $(u, v) \in E$; otherwise it is $false$.

Definition 5.5: Access Pattern

Given a graph G and a query history qH_G, the **access pattern** $\alpha(qH_G)$ **induced by** qH_G is a list $[\alpha(q_1), ..., \alpha(q_n)]$ such that for all $i \in \{1, ..., n\}$:

- if q_i is a neighbor query, then $\alpha(q_i) = \alpha(Q_{Neighbor}(G, u))$
- if q_i is an adjacency query, then $\alpha(q_i) = \alpha(Q_{Adjacency}(G, u, v))$.

Definition 5.6: Search Pattern

Given a graph G and a query history qH_G, the **search pattern** $\sigma(qH_G)$ induced by qH_G is a $n \times n$ binary symmetric matrix with the following entries: for $1 \leq i, j \leq n$, $\sigma[i][j] = 1$ if query $q_i = q_j$, and 0 otherwise.

Since we want to prevent attackers from learning the exact degree of a queried node, we limit the neighbor-access pattern that we are willing to leak. To this end, we introduce the notions of degree uncertainty and z-access pattern.

Definition 5.7: Degree Uncertainty

Given a a graph G, its transformed graph $transformed_G$, and (3) an access pattern $\alpha(qH_G)$, the **degree uncertainty** z of $transformed_G$ is an integer so that for all neighbor-access patterns in $\alpha(qH_G)$ it holds that $|x - y| \geq z$.

Example 5.1. Think of a graph G, its corresponding $transformed_G$, and the degree uncertainty $z = 5$. In this case, one can be sure that independent from the queries executed over $transformed_G$, the absolute difference between the lower and upper bounds of the neighbor-access pattern of any node in G that an adversary can learn is always greater than or equal to 5.

Definition 5.8: Z-access Pattern

Given a degree uncertainty $z \in \mathbb{Z}$, a **z-access pattern** $\alpha_z(qH_G)$ is an access pattern in which all neighbor-access patterns fulfill z.

Definition 5.9: Accepted Information Leakage

Given a transformed graph $transformed_G$ and a degree uncertainty $z \in \mathbb{Z}$ with $z \geq 1$, the **accepted information leakage**, with the *Ind-Graph* secrecy notion to be defined, is:
(L1) The z-access pattern $\alpha_z(qH_G)$

(L2) The search pattern $\sigma(qH_G)$,
(L3) The size of the original graph $size(G)$
(L4) The multiset of degrees $Deg(G)$

Although leakage L1 could lead to attacks such as the ones featured in [KKNO16, NKW15], L1 and L2 are in line with the work described in [CGKO11, BBHJ11]. L3 is similar to the leakage accepted by [WRD+17, MKNK15]. We use leakage L4 only to evaluate the trade-off between secrecy and performance, and for specific graph-secretization algorithms that only leak part of the multiset of degrees. L4 can be relaxed further, as we will prove in Subsection 5.2.6. Proposing a graph-secretization algorithm that guarantees secrecy against the information leakage L1-L4 is out of the scope of this dissertation.

To evaluate the secrecy guarantees offered by a graph-secretization algorithm, one needs a secrecy notion, i.e., given an adversary with certain knowledge, when does a secrecy breach indeed occur.

We propose a secrecy notion for graph-structured data called Graph Indistinguishability, *Ind-Graph*. Our secrecy notion is based on the concepts of indistinguishability presented by [KL07] and the notion of searchable encryption presented by [CGKO11]. We use indistinguishability as our secrecy notion for the same reason as the one featured in [KL07]. That is, given an algorithm, it is easier to show that it fulfills the indistinguishability concept than the one of semantic secrecy. However, the secrecy guarantees are the same. As explained in Subsection 3.3.3, the concept of indistinguishability is defined based on an indistinguishability experiment between an adversary and a challenger. Before describing such an experiment in our graph setting, we define first the trapdoor term for a given query.

Definition 5.10: Query Trapdoor

Given a query q over graph G and a key K, the **trapdoor** of query q, t_q, is the output of a deterministic algorithm $T(K, q)$ that allows to execute q over the transformed graph $transformed_G$.

Observing the execution of a list of queries is equivalent to having their trapdoors. In Subsection 5.2.6, we specify the trapdoors for our graph-secretization algorithm.

In general, the idea behind the indistinguishability experiment is that an adversary $\mathcal{A}$ is allowed to feed two inputs in the experiment. The challenger randomly chooses one of the inputs and uses an algorithm to secure the selected input. $\mathcal{A}$ receives the output of the experiment, but she does not know which one has been the chosen input. The output of the experiment should represent all the

information that $\mathcal{A}$ can observe, and its inputs should represent all the information needed to produce the output mentioned. The selection of the inputs can, however, be restricted based on the accepted information leakage, Definition 5.9. At the end of the experiment, $\mathcal{A}$ has to "guess" the input chosen by the challenger. The final output of the experiment is defined to be 1 if $\mathcal{A}$ "guesses" correctly and 0 otherwise. If the final output is 1, we say that $\mathcal{A}$ has succeeded.

Definition 5.11: Graph-indistinguishability Experiment

Let $\mathcal{A}$ be an adversary, τ a graph-secretization algorithm and n a security parameter. The **graph-indistinguishability experiment *Ind-Graph*** is defined as follows:

$\mathit{Ind\text{-}Graph}_{\mathcal{A},\tau}(n)$

$\mathcal{A}$ *chooses* $(G_0, qH_{G_0}, G_1, qH_{G_1})$
$b \leftarrow_{\$} \{0,1\}$
for $1 \leq i \leq n$
  $t_{b,i} \leftarrow t_k(q_i)$
let $T_b = [t_{b,1}, ..., t_{b,n}]$
$\overline{b} \leftarrow \mathcal{A}(\tau(G_b), T_b)$
return 1 *if* $b = \overline{b}$ *else* 0

with the restrictions that $\alpha_z(qH_{G_0}) = \alpha_z(qH_{G_1}), \sigma(qH_{G_0}) = \sigma(qH_{G_1})$, $size(G_0) = size(G_1)$ and $Deg(G_0) = Deg(G_1)$.

Definition 5.12: An *Ind-Graph* Secure Algorithm

Given an adversary $\mathcal{A}$, a graph-secretization algorithm τ and a security parameter n, a graph-secretization algorithm τ is called ***Ind-Graph* secure** if the function

$$\mathsf{Adv}_{\mathcal{A},\tau}(n) := \left| \Pr\left[\mathit{Ind\text{-}Graph}_{\mathcal{A},\tau}(n) = 1\right] - \frac{1}{2} \right|$$

is negligible for any adversary $\mathcal{A}$ whose computational effort is bounded to polynomial time.

5.2 Our Secrecy Approach

We now describe our graph-secretization algorithm. We call it *bucketization algorithm*. We first give an overview and describe the underlying system architecture. Then we describe the challenges, formalize the problem, and present our approach.

5.2.1 System Architecture

We consider a database-as-a-service setting where a third-party service provider stores data owned by the clients. For confidentiality reasons, i.e., data secrecy, clients apply techniques to secure the data before passing it to the service provider.

We first introduce some bucketization notions, Subsection 5.2.1, and then we present our system architecture model and query processing, Subsection 5.2.1.

Bucektization Notions

Definition 5.13: Bucket

Given a graph G, a **bucket** b is a finite set of edges of G. Each bucket has an identifier denoted by $bucketID(b)$. All buckets have the same capacity denoted by $maxEdges$, i.e., a bucket can store at most $maxEdges$ edges. The *frequency of bucket* b, *freq*(b), is the number of edges that bucket b stores. The set of buckets of G that stores all edges of G is denoted by $\mathcal{B}_G$

Definition 5.14: Index Information

Given a graph G and a corresponding set of buckets $\mathcal{B}_G$, the **index information**, dubbed $index_G$, is a map $index_G : V \rightarrow \mathcal{B}_G$ such that for each $u \in V$, $index_G$ contains the set of identifiers of buckets that store at least one outgoing edge of u.

Definition 5.15: Bucketizatization Structure

A **bucketization structure** of a given graph G, denoted BS_G, is a representation of G consisting of two parts: a set of buckets $\mathcal{B}_G$ and the index information $index_G$. We use $\mathcal{BS}_G$ to denote the set of all possible bucketization structures of graph G.

From now on, given a graph G, we use the terms bucketization structure $\mathcal{BS}_G$ and transformed graph $transformed_G$ to refer to the output of our bucketization scheme and the output of any graph-secretization algorithm, respectively.

Figure 5.2 illustrates a bucketization structure. We use parentheses to denote tuples and curly brackets to denote sets. We write $\mathsf{Enc}_d(k, m)$ and $\mathsf{Enc}_p(k, m)$ to denote the operations of encrypting m under key k using a deterministic encryption algorithm and encrypting m under key k using a probabilistic encryption algorithm.

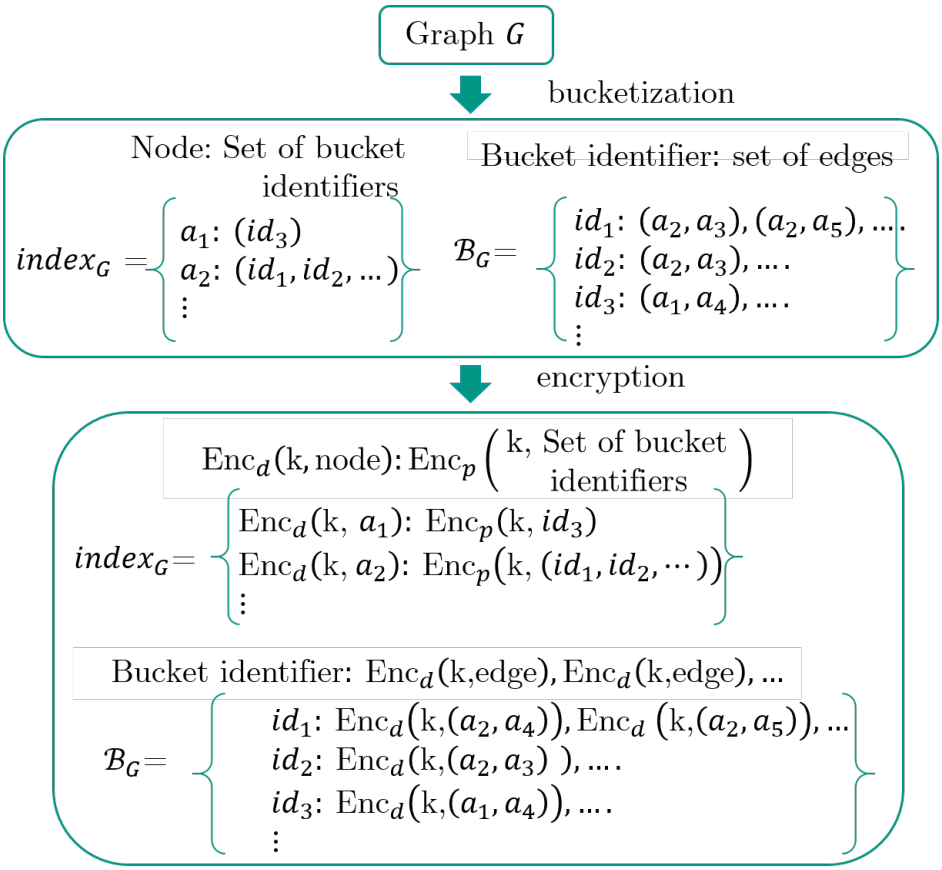


Figure 5.2: Bucketization and Encryption of Graph G

Definition 5.16: Bucketization Function

A **bucketization function** $buck : \mathcal{G} \to \mathcal{BS}_{\mathcal{G}}$ is a function that generates a bucketization structure BS_G for a graph $G \in \mathcal{G}$.

Definition 5.17: Encryption Function

Given a bucketization structure of a graph G, BS_G, an **encryption function** $\mathsf{Enc} : \mathcal{BS}_G \rightarrow \mathcal{BS}_G$ performs an encryption of B as follows: First, in the index information, each label of a node is encrypted deterministically, and the bucket identifiers are encrypted probabilistically. Second, in the set of buckets, each edge is encrypted deterministically.

We use encryption techniques, deterministic and probabilistic, to protect against deterministic chosen-plaintext attacks, Definition 5.18. In addition to that, we use bucketization techniques to protect against frequency attacks. Regarding bucketization, we aim for an optimal bucketization concerning query performance. We explain the specifics of our approach in Subsection 5.2.4.

Definition 5.18: Deterministic Chosen-plaintext Attack [BS08]

A **deterministic chosen-plaintext attack** is a relaxed notion of chosen-plaintext attachs, Subsection 3.3.2, in which the adversary never sees the same plaintexts encrypted with the same key more than once. The adversary can choose several plaintexts to be encrypted and obtain their corresponding ciphertexts. The adversary sends two different plaintexts m_0 and m_1 to the challenger, with the restriction that the plaintexts are distinct from the messages sent previously. The adversary receives the ciphertext of one of them. The goal of the adversary is to distinguish if she has received the ciphertext of m_0 or m_1.

A deterministic encryption scheme is not secure against chosen-plaintext attacks (CPA). The encryption scheme must be probabilistic to offer secrecy guarantees against CPA [KL07]. However, if a deterministic encryption scheme does not encrypt the same plaintext more than once, i.e., the plaintext messages to be encrypted are unique, the scheme is secure against deterministic chosen-plaintext attacks [BS08, BBO07]. A probabilistic encryption scheme offers secrecy guarantees against both types of attacks. Secrecy proofs are in Subsection 5.2.6.

System Architecture Model and Query Processing

To achieve data secrecy, before outsourcing a graph G, the client applies a bucketization function on G, and after encryption, the client outsources the bucketization structure to the service provider. Figure 5.3 illustrates the system architecture model of the database outsourcing scenario. It consists of a trusted client and an untrusted server. Since the server that stores the data is untrusted, the client

should have some computational capabilities to process queries and results between the users and the server. We assume that the client has two components for query processing, namely the query translator and the query post-processor. Query processing is as follows:

1. A user sends a query to the client.
2. The query translator translates the query into a list of queries, called server-query list. This list contains one or several server-side queries and one filtering client-side query. The server-side queries in the server-query list, apart from the first one, are in general not concrete queries, i.e., they require additional information to be executed.
3. The query translator sends the server-query list to the query post-processor.
4. The query post-processor sends to the server the next server-side query in the server-query list.
5. The server executes the server-side query.
6. The server sends the encrypted results to the client.
7. The query post-processor decrypts the results. If there are more server-side queries in the server-query list, the query post-processor uses the decrypted results to instantiate the next server-side query. Steps 4, 5, 6 and 7 start again until the server has processed all server-side queries.
8. Finally, the query post-processor gets the final encrypted results, decrypts them, executes the filtering client-side query and sends the result to the user.

5.2.2 Bucketization Challenges

The encryption function encrypts the label of the nodes with deterministic encryption. Deterministic encryption is secure under deterministic chosen-plaintext attacks [BS08, BBO07]. However, frequency attacks are still feasible. To secure against frequency attacks, we use a bucketization technique tailored to graph-structured data. Finding a bucketization that guarantees both secrecy as well as good query performance is challenging. The reason is that it is not obvious how to assign edges to buckets, see Examples 5.2 and 5.3. The bucketization structure may expose the frequency of buckets.

Example 5.2. Think of an email network with nodes $V = \{$ *Alice, Bob, Carol, Dan, Eva* $\}$ and edges $E = \{$ (*Alice*, *Bob*), (*Alice, Dan*),

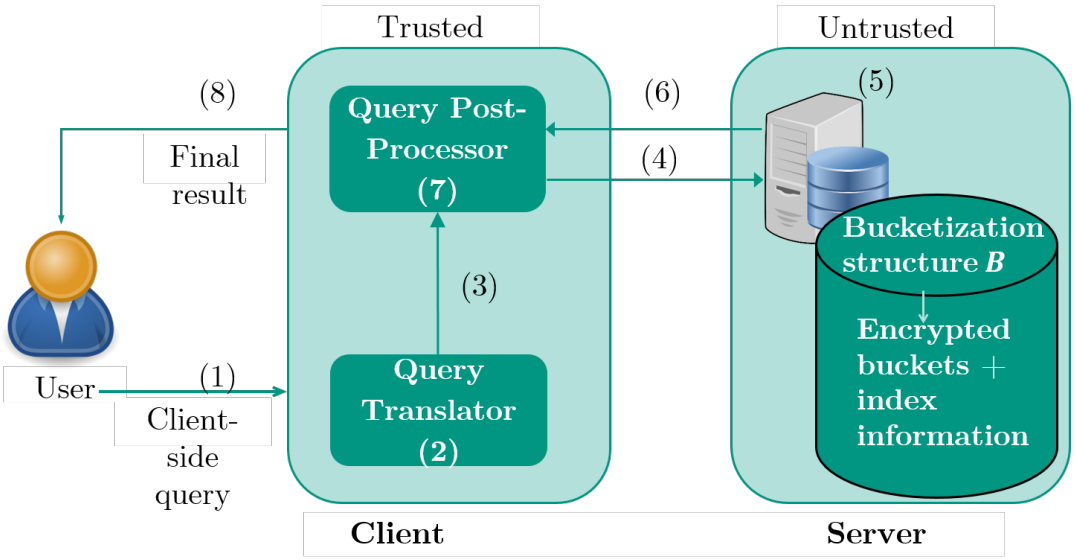


Figure 5.3: System Architecture and Query Processing

(*Alice, Carol*), (*Alice, Eva*), (*Bob, Dan*), (*Carol, Eva*), (*Carol, Alice*), (*Dan, Carol*), (*Eva, Bob*) }. Assume that we apply a bucketization algorithm that assigns edges randomly and stores two edges per bucket. In the worst case, the algorithm assigns the four edges of Alice to four different buckets. That is, during query processing, it is necessary to access four buckets to retrieve the edges of Alice. Then the overall query processing effort, i.e., client and server workload, is rather large, because the server has to access more buckets, and the client has to filter more data.

Example 5.3. Consider Example 5.2. If each bucket stores all the edges belonging to only one node and no other edges, the frequency of each bucket reveals the node degree. An adversary who knows the degree of each node in the network, i.e., the number of emails that each user has sent, can conclude that the bucket with four edges corresponds to Alice and the one with two edges to Carol.

Definition 5.19: Link between Bucket and Degree

Given a graph G, a set of buckets of G, $\mathcal{B}_G$, a node $u \in V$, and a bucket $b \in \mathcal{B}_G$, **a link between bucket** b **and the degree of node** u exists if

$\forall b' \in \mathcal{B}_G \setminus \{b\}, \forall v \in V \setminus \{u\} : freq(b) \neq freq(b') \wedge deg(u) \neq deg(v) \wedge freq(b) = deg(u)$.

Assigning edges to buckets at random will likely bog down query performance, cf. Example 5.2. Then, one should store the edges of a node in as few buckets as possible. However, storing all edges of a node in one bucket creates a link between the degree of nodes and their corresponding buckets, which might affect secrecy. To avoid information leakage, one should create indistinguishable buckets. Specifically, we aim for an equal frequency of buckets, i.e., all buckets should reach the maximal capacity of $maxEdges$. Since an assignment may not always yield full buckets, it is promising to merge them a posteriori, add dummy edges or both; our approach will feature both. Preliminary experiments of ours have shown that dummy edges do increase the overall query-processing time significantly both at the client and the server. Therefore, the total number of dummy edges should be as small as possible.

5.2.3 The Optimal Bucketization Problem

The optimal bucketization problem is as follows: Given a graph $G = (V, E)$ as input, we search for a bucketization structure of graph G, BS_G, that meets the constraints $c_1 - c_4$.

(c_1) Each edge $(u, v) \in E$ is assigned to one bucket.

(c_2) Each bucket stores at most $maxEdges$ edges, where $maxEdges$ is a given parameter.

(c_3) Edges adjacent to the same node are placed in as few buckets as possible. Formally, let the function $ind : V \times \mathcal{BS}_G \rightarrow \mathbb{N}$ be as follows: $ind(u, S) := |\{b \in BS_G \mid \exists x \in V : (u, x) \in b\}|$. Given a node u and a bucketization structure of a graph G, BS_G, the function ind returns the number of buckets that store the edges of node u. Then $\forall BS'_G \in \mathcal{BS}_G, \forall u \in V : ind(u, BS_G) \leq ind(u, BS'_G)$.

(c_4) The total number of buckets should be as small as possible, while prioritizing Constraint c_3.

We prioritize Constraint c_3 over c_4 so that query performance is not affected, see Example 5.2. We call a bucketization that meets Constraints c_1 to c_4, an *optimal bucketization*.

Our optimal bucketization problem is NP-hard. To prove this, we reduce the Bin-packing problem (BP problem) [Joh73] to our problem. The authors in [Joh73] and [GJ78] have proven that the BP problem is NP-complete and strongly NP-complete, respectively. The hardness result, together with the proofs, are in Appendix A.

5.2.4 Our Bucketization Approach

Finding an optimal bucketization is NP-hard. Therefore, we propose a heuristic that aims to meet the constraints established in the previous subsection, Subection 5.2.3. Our algorithm has two phases: an initialization phase and a merging phase. See Figure . Due to the complexity of the optimal bucketization problem, we use heuristics in the merging phase.

Definition 5.20: Initial Bucketization Structure

Given a graph G, the **initial bucketization structure** of graph G, BS_G^0 is the result of the initialization phase of the bucketization algorithm applied to graph G.

Definition 5.21: Final Bucketization Structure

Given a graph G, a **final bucketization structure** of graph G, BS_G^f, is a bucketization resulting from the initialization and bucket merging phases applied to G.

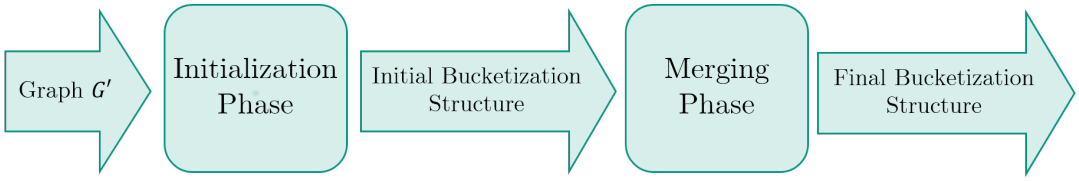


Figure 5.4: Phases of our Bucketization Approach

The Initialization Phase

Algorithm 9 describes the initialization phase of our bucketization approach for a graph G. Since the length of the ciphertext could reveal the length of the plaintext, we pad the label of the nodes to avoid this leakage (Line 1). Formal secrecy proofs are presented in Section 5.2.6.

Algorithm 9: Initialization

Input : Graph: $G = (V, E)$, int: $maxEdges$
Output: initial bucketization: BS_G^i

```
1 labelOfNodes.pad(V); ;                    // Pad the label of the nodes
2 foreach v in V do
3 |   create (ceil(1, v.numberOfEdges()/maxEdges)) buckets;
4 |   assign randomly up to maxEdges edges of v to each bucket;
5 |   generate the corresponding index information;
```

Example 5.4 and Example 5.5 illustrate how the assignment of edges works, and explain the need for randomness with this assignment, respectively.

Example 5.4. Let us set $maxEdges = 10$. Given a node v that has 27 edges, the initialization algorithm creates three buckets. Then, it chooses 10 random edges from the 27 edges and assigns them to the first bucket. Next, it chooses another 10 random edges from the remaining ones for the second bucket, and the 7 remaining edges go to the third bucket.

Example 5.5. For the sake of an easy example, consider a setting where emails can be revoked without difficulty. For this example, we consider the buckets in Fig.5.5. Assume that the bucketization algorithm does not assign edges randomly; instead, it assigns them alphabetically. Assume further that Alice has one email to each of the following persons: Bob, Carol, Dan and Eve. If an adversary knows who has received emails from Alice, although the edges are encrypted, the adversary will learn that the bucket $b2$ stores the last edge, i.e., the email sent to Eve. Gaining this knowledge is information leakage. This leakage can lead to an attack with an extended adversary model where the adversary has access to the log history. In the case where the bucket $b2$ disappears, the adversary knows that one email was revoked, and she will learn that the revoked email was the last one, i.e., the email sent to Eve. A random assignment of edges reduces the chance that the adversary learns extra information.

After the initialization phase of the bucketization process, the initialization algorithm has placed all edges into their corresponding buckets. At this point, some buckets may not have reached the maximal capacity, $maxEdges$. Even if we encrypt the buckets at this stage, the initial bucketization is not secure. If the degree of a node is less than or equal to $maxEdges$, its edges have been placed in one

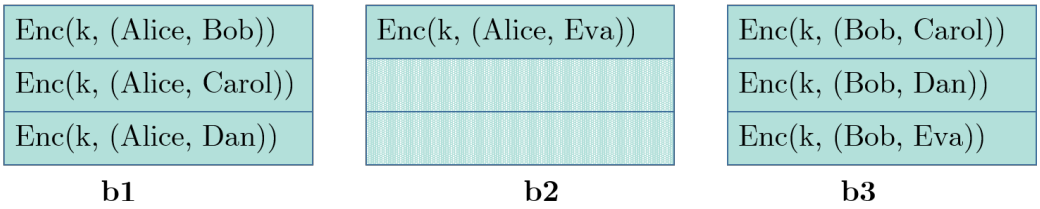


Figure 5.5: Illustration of Example 5.5

bucket exclusively. That is, nodes with degree less than or equal to *maxEdges* have a link between their degree and the frequency of their corresponding buckets. By using such a link, an adversary will be able to identify the buckets of these nodes, see Example 5.3.

One could consider inserting dummy edges at this stage to avoid leaking information. Although adding dummy edges solves secrecy problems, the query performance of this solution is affected. The overall query processing time without the merging phase increases because the number of buckets at the initialization phase is higher than the number of buckets after the merging phase, and this slows down the server.

The Merging Phase

Definition 5.22: Bucket Merging

Bucket merging is an operation that puts the content of two buckets in a new one and then deletes the two emptied ones.

In the merging phase, the algorithm merges buckets to fulfill Constraint $c4$. Algorithm 10 identifies pairs of buckets that can be merged to obtain buckets with the same frequency. Different heuristics are conceivable at this stage. We choose a First Fit Decreasing approach (FFD) proposed in [GJ79]. We will justify this decision after having explained the algorithm.

Notice that two nodes could have the same set of bucket identifiers. Encrypting the set of bucket identifiers with deterministic encryption could lead to frequency attacks. The use of probabilistic encryption prevents these attacks. Formal secrecy proofs are in Section 5.2.6.

Lemma 5.1. *The worst case solution of our bucketization algorithm with the FFD approach is off by a factor of* $\frac{11}{9}$ *from the optimal one.*

Algorithm 10: Merge Buckets

Input : initial bucketization BS_G^i, int: $maxEdges$
Output: final bucketization: BS_G^f

```
/* Set B' store buckets with frequency less than maxEdges. Set
   SB_G^f store full buckets. Set B_m store the buckets resulting
   from a merge.                                                  */
```

Initialize: $B' := \{b \in BS_G^i | b.numberOfEdges() < maxEdges\}$; $BS_G^f := BS_G^i \setminus B'$; $B_m := \{\}$
Order B' by number of edges in decreasing order;
foreach $b_i \in B'$ **do**
  foreach $b_j \in B_m$ **do**

```
    /* Search for the first bucket in B' such that
       |b_i| + |b_j| ≤ maxEdges                                   */
```

    if b_i *fits in* b_j **then**

```
      /* Put the content of buckets b_i, b_j into bucket b       */
```

      $b \leftarrow merge(b_i, b_j)$;
      delete b_i, b_j;
      if $b.numberOfEdges() = maxEdges$ **then**

```
        /* If b is full, add b to set of full buckets           */
```

        add b to BS_G^f

```
      /* If bucket b is not full, add b to the set B_m          */
```

      else
        add b to B_m
      break;

```
  /* If a merge was not possible, place b_i in B_m.             */
```

  if $(b_i \in B')$ **then** move b_i to B_m;
foreach $b \in B_m$ **do**
  b.addDummyEdges() ; `// Add dummy edges to non-full buckets`
  add b to BS_G^f

```
/* Encrypt the edges of each bucket individually using
   deterministic encryption. In the index information, encrypt
   the labels of the nodes using deterministic encryption, and
   the set of bucket identifiers using probabilistic encryption.
   Before encryption, pad the set of bucket identifiers such that
   all sets have the same length.                                */
```

Encrypt BS_G^f;
return BS_G^f;

Proof. Garey and Johnson [GJ79] have proven that the worst case solution for the bin packing problem with the FFD approach is off by a factor of $\frac{11}{9}$ from the optimal one. Our algorithm uses the FFD approach in the merging phase. □ □

Other heuristics for the merging phase, such as Best Fit and Next Fit, have a worse approximation ratio, $\frac{17}{10}$ and 2, respectively, [GJ79].

5.2.5 Query Transformation

Unlike other approaches such as the one presented in [HVS+10], our bucketization approach does not lose any information regarding the original graph. Consequently, there is no limitation regarding the kind of query one can process in principle. However, query processing can affect the computation at the client-side because the client might have to do most or all of the query processing work. Our focus has been on reducing the client workload for neighbor and adjacency queries. For neighbor queries, the client workload, with our approach, consists of a query transformation process and a decryption and filtering process; the server performs the rest of the query execution. For adjacency queries, the client workload, with our approach, consists only of a query transformation process; the server performs the rest of the query execution. In the following, we describe the processing of neighbor and adjacency queries.

Client-side queries are transformed into server-side queries. We use the conventions *server.m* and *client.m* to indicate that method m runs at the server-side and the client-side, respectively. Algorithm 11 shows the transformation process for neighbor queries. Algorithm 12 shows the procedure for adjacency queries.

5.2.6 Secrecy Proofs

In this subsection, we first prove that our bucketization approach fulfills the secrecy notion *Ind-Graph* defined in Section 5.1. Then, we prove that our algorithm fulfills our secrecy notion even with a more relaxed Leakage L4, Definition 5.9.

Our Bucketization Approach is Ind-Graph Secure

In our algorithm, the parameter $maxEdges$ can be set freely. In Definition 5.5, we have defined the z-access pattern based on a degree uncertainty z. Then, for a given z, to guarantee that our bucketization algorithm fulfills our secrecy notion, the parameter $maxEdges$ must be set accordingly, i.e., $maxEdges > z$. We set $maxEdges = z + 1$.

Algorithm 11: Neighbor Query Processing over the Bucketization Structure of Graph G, BS_G.

Input : $Q_{Neighbor}(G, u)$, key k, Bucketization Structure BS_G
Output: $Edges := \{\}$

Initialize: EncBucketIDs:={}, BucketIDs:={}, EncEdges:={}, EdgesTemp :={};
/* The client encrypts node u and generates the server-side query. */
encNode ← client.$enc_d(k, u)$;
if *server.indexInformation.contain(encNode)* **then**
  /* Retrieve from the index information the set of bucket identifiers of the encrypted node encNode */
  EncBucketIDs ← indexInformation(encNode);
  /* Decrypt the set $EncBucketIDs$ */
  BucketIDs ← client.$\mathsf{Dec}_p(k, EncBucketIDs)$ **foreach** b *in* $BucketIDs$ **do**
    /* Retrieve the edges stored in bucket b */
    e ← server.SetOfBuckets.ValueOf(b);
    add e to EncEdges;
  foreach e *in* $EncEdges$ **do**
    EdgesTemp ← client.add.$dec_d(k, e)$; ; // Decrypt the edges
  foreach e *in* $EdgesTemp$ **do**
    /* Filter false positives and dummy edges */
    if $!e.isFalsePositive \vee !e.isDummy$ **then**
      add e to $Edges$
return $Edges$;

Figure 5.6 shows the setup of the indistinguishability experiment from Definition 5.12.

Figure 5.7 illustrates an abstract output of the bucketization algorithm, where $|C_{(node)}|$ is the length of the ciphertext representing an encrypted node, $|N_{(enc)}|$ is the number of encrypted nodes, $|C_{(bucketIDs)}|$ is the length of the ciphertext representing the set of bucket identifiers, $|C_{(edge)}|$ is the length of the ciphertext representing an encrypted edge, $bucketID$ are random identifiers of the buckets, and $|\mathcal{B}_G|$ is the number of buckets. We use these notations in our secrecy proofs.

Lemmas 5.2-5.4 tell us that our bucketization algorithm guarantees that: Given as input two graphs G_0 and G_1, and two query histories qH_{G_0}, qH_{G_1}, which comply with the restrictions of the experiment, an adversary $\mathcal{A}$, who has access to the

Algorithm 12: Adjacency Query Processing over the Bucketization Structure of graph G, BS_G.

Input : $Q_{Adjacency}(G, u, v)$, key k, Bucketization Structure BS_G
Output: Boolean isEdge

```
1 Initialize: isEdge =false;
  /* Encrypt using deterministic encryption the two nodes given as
     input as part of the query and generatethe server-side query.
     */
2 encEdge ← client.enc_d^K(u, v)
  /* Search in the set of buckets for the encrypted edge encEdge  */
3 foreach bucket in server.SetOfBuckets() do
4 |   if server.setOfBuckets.ValueOf(bucket) contains encEdge then
5 |   |   isEdge=true ;                          // The nodes are adjacent
6 return isEdge;
```

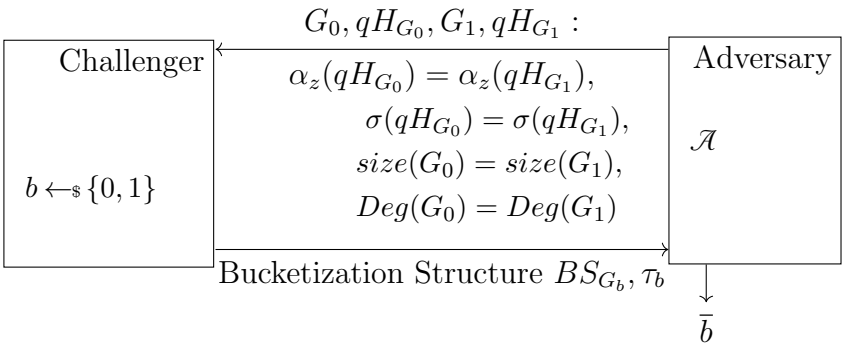


Figure 5.6: Indistinguishability experiment *Ind-Graph*

bucketization structure of one the graphs and the corresponding trapdoors, cannot say which graph the indistinguishability experiment selected as input. We organize the proofs of Lemmas 5.2-5.4 as follows: We demonstrate that for both inputs given by the adversary $\mathcal{A}$, the properties of the index information, the set of buckets and the trapdoors either are the same, or their differences do not allow the adversary $\mathcal{A}$ to decide which graph has been selected as input.

To facilitate the proof, we first prove that our bucketization algorithm is *Ind-Graph* with respect to the set of buckets output, Lemma 5.2, with respect to the index information output, Lemma 5.3, and with respect to the trapdoors, Lemma 5.4.

Lemma 5.2. *Let the following be given:*

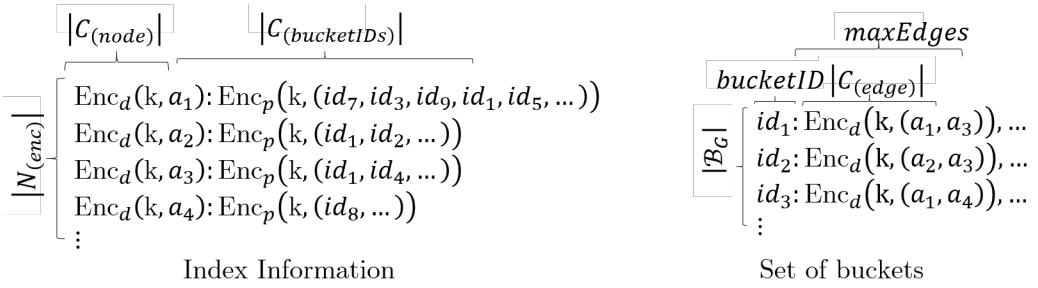


Figure 5.7: Abstract output of the bucketization algorithm

1. *An adversary $\mathcal{A}$ who chooses two graphs G_0, G_1 and two query histories qH_{G_0}, qH_{G_1} in line with Definition 5.12.*
2. *The set of buckets the bucketization algorithm has generated by selecting randomly G_0, qH_{G_0} or G_1, qH_{G_1}.*

The adversary $\mathcal{A}$ cannot decide which tuple, (G_0, qH_{G_0}) or (G_1, qH_{G_1}), has been the input of the bucketization algorithm for $maxEdges$ values equal to or greater than the degree uncertainty z of the z-access pattern.

Proof. The properties of the set of buckets that do not change for both inputs given by the adversary $\mathcal{A}$ are the size of the buckets and the number of buckets $|\mathcal{B}_G|$. The properties that differ are the bucket identifiers and the encrypted edges. First, all buckets have the same size equal to $maxEdges$. Since the bucketization algorithm uses the same value for the parameter $maxEdges$, the size of all buckets for either G_0 or G_1 is the same. Second, $|\mathcal{B}_G|$ depends on the number of edges of the nodes. After the initialization phase, our bucketization algorithm uses the FFD approach to merge the buckets. Since $Deg(G_0) = Deg(G_1)$, $|\mathcal{B}_G|$ is the same for both graphs. Third, the bucketization algorithm generates the bucket identifiers, $bucketIDs$, randomly. The bucketization algorithm will generate random bucket identifiers for both graphs. So, $\mathcal{A}$ cannot identify whether the $bucketIDs$ correspond to Graph G_0 or G_1. Fourth, the edges in a graph are unique, and they are encrypted deterministically. So the edges are secure against deterministic chosen-plaintext attacks, i.e., an adversary cannot learn any useful information about the original edges, including their frequencies. Therefore, $\mathcal{A}$ cannot recognize whether the encrypted edges in the set of buckets correspond to G_0 or G_1. Consequently, the adversary $\mathcal{A}$ cannot distinguish whether the tuple (G_0, qH_{G_0}) or the tuple (G_1, qH_{G_1}) has been the input of the bucketization algorithm. □

Lemma 5.3. *Let the following be given:*

1. *An adversary $\mathcal{A}$ who chooses two graphs G_0 and G_1 and two query histories qH_{G_0} and qH_{G_1} in line with Definition 5.12.*
2. *The index information the bucketization algorithm has generated by selecting randomly G_0, qH_{G_0} or G_1, qH_{G_1}.*

The adversary $\mathcal{A}$ cannot distinguish whether the tuple (G_0, qH_{G_0}) or the tuple (G_1, qH_{G_1}) has been the input of the bucketization algorithm for maxEdges values equal to or greater than the degree uncertainty z of the z-access pattern.

Proof. The properties of the index information that do not change for both inputs given by the adversary $\mathcal{A}$ are: the number of encrypted nodes $|N_{(enc)}|$, the length of the ciphertext representing an encrypted edge $|C_{(node)}|$, and the length of the ciphertext representing the set of bucket identifiers $|C_{(bucketIDs)}|$. The properties that can be different are the encrypted nodes and the encrypted sets of bucket identifiers. First, $|N_{(enc)}|$ is the same for both graphs because $|V_0| = |V_1|$. Second, the bucketization algorithm pads the labels of the nodes before encryption. Then, $|C_{(node)}|$ is the same for all the nodes in both graphs. Third, the sets of bucket identifiers are padded and then encrypted. Then, $|C_{(bucketIDs)}|$ is the same for all sets of bucket identifiers in both graphs. Fourth, the nodes are unique. The bucketization algorithm encrypts them deterministically, so they are secure against deterministic chosen-plaintext attacks. Therefore, the adversary $\mathcal{A}$ cannot distinguish whether the encrypted nodes correspond to G_0 or G_1. Fifth, the bucketization algorithm uses probabilistic encryption to encrypt the set bucket identifiers. So, they are secure against chosen-plaintext attacks, i.e., an adversary cannot learn any useful information from the set of bucket identifiers, including their frequencies. That is, $\mathcal{A}$ cannot recognize whether the encrypted sets of bucket identifiers correspond to G_0 or G_1. Consequently, the adversary $\mathcal{A}$ cannot distinguish whether the tuple (G_0, qH_{G_0}) or the tuple (G_1, qH_{G_1}) has been the input of the bucketization algorithm. □

Lemma 5.4. *Let the following be given:*

1. *An adversary $\mathcal{A}$ who chooses two graphs G_0, G_1 and two query histories qH_{G_0}, qH_{G_1} in line with Definition 5.12.*
2. *The list of trapdoors the bucketization algorithm has generated by selecting randomly G_0, qH_{G_0} or G_1, qH_{G_1}.*

The adversary $\mathcal{A}$ cannot distinguish whether the tuple (G_0, qH_{G_0}) or the tuple (G_1, qH_{G_1}) has been the input of the bucketization algorithm for maxEdges values equal to or greater than the degree uncertainty z of the z-access pattern.

Proof. In our setting, a query in a query history qH_G can be either a neighbor or an adjacency query. Given a query history qH_G with n queries, the list of trapdoors T of qH_G consists of n items— one for each query. The content of the trapdoors depends on the type of query. If the i-th query $q_i \in qH_G$ is a neighbor query, the i-th trapdoor t_i consists of an encrypted node, which corresponds to the query q_i, and a set of bucket identifiers, which correspond to the buckets that store the encrypted edges of that node. If the i-th query $q_i \in qH_G$ is an adjacency query, the i-th trapdoor t_i consists of an encrypted edge, which corresponds to the query q_i. Let T_0 and T_1 be the lists of trapdoors of qH_{G_0} and qH_{G_1}, respectively. First, because $\alpha_z(qH_{G_0}) = \alpha_z(qH_{G_1})$ and $\sigma(qH_{G_0}) = \sigma(qH_{G_1})$ for all $i \in \{1, ..., n\}$, the following holds:

- If the i-th query q_i is a neighbor query, the i-th trapdoor t_i in T_0 has one encrypted node with the same length $|C_{(node)}|$ and the same number of $bucketIDs$ as the i-th trapdoor t_i in T_1.
- If the i-th query q_i is an adjacency query, the i-th trapdoor t_i in T_0 has one encrypted edge, and the i-th trapdoor t_i in T_1 has one encrypted edge as well.

Then, T_0 and T_1 have the same structure in both cases. Second, T_0 and T_1 could have different content, i.e., the encrypted nodes and the bucket identifiers for neighbor queries, and the encrypted edges for adjacency queries in T_0 can be different from the ones in T_1. However, as demonstrated in Lemmas 5.2 and 5.3, the nodes, and edges are secure against deterministic chosen-plaintext attacks, and the bucket identifiers are generated randomly. Then, based on the content of the trapdoors, $\mathcal{A}$ cannot distinguish whether the list of trapdoors corresponds to qH_{G_0} or qH_{G_1}. Consequently, an adversary $\mathcal{A}$ cannot distinguish whether the tuple (G_0, qH_{G_0}) or the tuple (G_1, qH_{G_1}) has been the input of the bucketization algorithm. □

We have proven in Lemmas 5.2 - 5.4 that the index information, the set of buckets and the list of trapdoors, generated by our algorithm, do not leak any information that can help an adversary to distinguish the input selected by the indistinguishability experiment – when looked at in isolation. It remains to demonstrate that the entire output does not leak such information. If there is not a link between the data structures of the output, it is enough to show that each data structure, when looked at in isolation, does not leak information. However, if there are links between them, it is necessary to show that $\mathcal{A}$ cannot use this information to discern the input selected by the indistinguishability experiment.

Example 5.6 shows that the output data structures together could leak information even though each data structure separately does not.

Example 5.6. Consider the graphs G_0 and G_1 of Figure 5.8, and two query histories $qH_{G_0} = [Q_{Adjacency}\ (G_0, A, C)]$ and $qH_{G_1} = [Q_{Adjacency}\ (G_1, A,\ \ C)]$. Each data structure in the bucketization structure BS_{G_0} of graph G_0 has the same structure as its corresponding data structure in the bucketization structure BS_{G_1} of graph G_1. Next, the lists of trapdoors T_{G_0} and T_{G_1} of the bucketization structures BS_{G_0} and BS_{G_1}, respectively, have the same structure. Additionally, because of the encryption used and the uniqueness of the encrypted values, the content of each data structure cannot be used to distinguish the input given to the algorithm. That is, they are indistinguishable from each other. However, one can use patterns, which can only be recognized in the entire structure of the output, to distinguish the input given to the algorithm. In this current example, the trapdoor in T_{G_0} occurs in the set of buckets, contrary to the trapdoor in T_{G_1}. Using this information, an adversary can recognize the input, given the output of the algorithm, by checking if the ciphertext of the trapdoor occurs in the set of buckets, i.e, in the encrypted edges. If this occurs, the input is G_0; otherwise, the input is G_1.

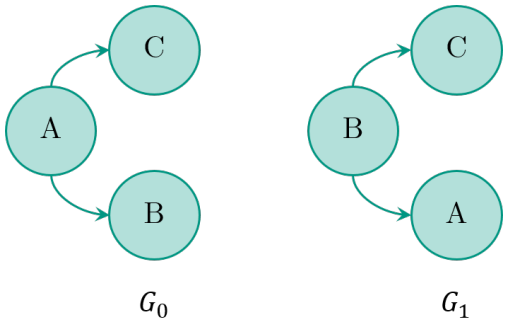


Figure 5.8: Graphs G_0, G_1 - Example 5.6

Lemma 5.5. *Let the following be given:*

1. *An adversary $\mathcal{A}$ who chooses two graphs G_0, G_1 and two query histories qH_{G_0}, qH_{G_1} in line with Definition 5.12.*
2. *The index information, the set of buckets and the list of trapdoors the bucketization algorithm has generated by selecting G_0, qH_{G_0} or G_1, qH_{G_1} randomly.*

An adversary $\mathcal{A}$ cannot distinguish based on links between the output structures whether the tuple (G_0, qH_{G_0}) or the tuple (G_1, qH_{G_1}) has been the input of the

bucketization algorithm for maxEdges values equal to or greater than the degree uncertainty z of the z-access pattern.

Proof. The possible links between the output structures are a link between the trapdoor of a neighbor query and the index information and a link between the trapdoor of an adjacency query and the set of buckets. However, because $\alpha_z(qH_{G_0}) = \alpha_z(qH_{G_1})$ and $\sigma(qH_{G_0}) = \sigma(qH_{G_1})$, these links are the same in both inputs. Consequently, an adversary $\mathcal{A}$ cannot distinguish based on the links between the output structures whether the tuple (G_0, qH_{G_0}) or the tuple (G_1, qH_{G_1}) has been the input of the bucketization algorithm. □

Theorem 5.6. *Our bucketization algorithm fulfills the secrecy notion Ind-Graph, Definition 5.12, for maxEdges values equal to or greater than the degree uncertainty z of the z-access pattern.*

Proof. With Lemmas 5.2 - 5.5, we have shown that the index information, the set of buckets, the list of trapdoors or a combination of them do not violate Definition 5.12. It remains to prove that the access pattern of our bucketization algorithm does not leak more than the degree uncertainty z. With our bucketization approach, the neighbor access pattern for any neighbor query is the tuple $((\#Buckets - 1) \cdot maxEdges + 1, \#Buckets \cdot maxEdges)$, where $\#Buckets$ is the number of buckets retrieve during the execution of a given neighbor query, i.e., $\#Buckets = \lceil \frac{deg(u)}{maxEdges} \rceil$. Then, the degree uncertainty is $|-maxEdges+1|$. Since we set $maxEdges = z+1$, the access pattern of our bucketization algorithm does not leak more than the degree uncertainty of the z-access pattern. Thus, our algorithm is Ind-Graph secure. □

A Relaxed Leakage L4

In this subsection, we show that our algorithm fulfills our secrecy notion even with a more relaxed Leakage L4, Definition 5.9. We proceed to define the notion needed, Definition 5.23, to relax L4.

Definition 5.23: Size of the Transformed Graph

Given a transformed graph $transformed_G$, the **size of the transformed graph** is a list $sizeT_G = [|ds_1|, ..., |ds_d|]$ which contains the size of all data structures in $transformed_G$.

Given a graph G, the size of its corresponding transformed graph $sizeT_G$ is the size of the bucketization structure BS_G. Our bucketization structure has two data

structures: the index information and the set of buckets. The size of the index information is the number of nodes, and the size of the set of buckets are the number of buckets. Then, $sizeBS_G = [|V|, |B_G|]$, where $sizeBS_G$ denotes the size of the bucketization structure BS_G.

Example 5.7 shows two graphs G_0 and G_1 that do not fulfill the restriction $Deg(G_0) = Deg(G_1)$. But the size of their bucketization structures is the same, i.e., $sizeBS_{G_0} = sizeBS_{G_1}$.

Example 5.7. Let us consider two graphs G_0 and G_1, each graph has 4 nodes and 8 edges, and their multisets of degrees are $Deg(G_0) = \{2, 2, 2, 2\}$ and $Deg(G_1) = \{3, 2, 2, 1\}$, respectively. Assume that $maxEdge = 4$. Even though $Deg(G_0) \neq Deg(G_1)$, the bucketization algorithm will output 2 buckets for both graphs. Then the size of the bucketization structure for both graphs is the same, i.e., $sizeBS_{G_0} = sizeBS_{G_1} = [4, 2]$.

Theorem 5.7. *Our bucketization algorithm fulfills the secrecy notion Ind-Graph, Definition 5.12, even by replacing leakage L4 with the more relaxed leakage of the size of the transformed graph $sizeT_G$, Definition 5.23, for $maxEdges$ values equal to or greater than the degree uncertainty z of the z-access pattern.*

Proof. With Lemmas 5.2 - 5.5, we have proven that our bucketization algorithm is *Ind-Graph* secure. We have used the restriction imposed by Leakage L4, i.e., $Deg(G_0) = Deg(G_1)$, only in the proof of Lemma 5.2. We used it to demonstrate that given two graphs, which are input by the adversary, the number of buckets $|B_G|$ is the same for both graphs. The same is also guaranteed with the restriction $sizeT_{G_0} = sizeT_{G_1}$, which in our bucketization means that $sizeBS_{G_0} = sizeBS_{G_1}$. Then, our bucketization algorithm fulfills the secrecy notion *Ind-Graph* even when replacing Leakage L4 with the more relaxed leakage regarding the size of the transformed graph $sizeT_G$. □

5.3 Performance Model

A performance model allows predicting the behavior of an algorithm and facilitates meaningful comparisons or evaluations of algorithms. Query optimizers, which are essential features of any modern database system, require such performance models to estimate the costs of various execution plans accurately and to find the most efficient one [MPS99, MVW98, GTK01]. If such performance models are not available, query optimizers will resort to coarse estimates, which may be grossly off, with disastrous consequences when it comes to system performance. The

difference between the cost of the best execution plan and a random choice could be in orders of magnitude [RH05]. With our bucketization algorithm, the number of buckets output by the algorithm is a crucial parameter for query performance, as explained in Section 5.2.2. Estimating the number of buckets is cumbersome, and we estimate a range. But even to determine this range, it is necessary to have a model that describes relevant properties of the given graph. Due to the importance of scale-free networks, we review some of their properties and use them to derive the so-called number-of-buckets model and query-cost model.

5.3.1 Scale-Free Networks

Real-world networks have two essential features:

- *Growth*: Real-world networks often are the result of a continuous growth process.
- *Preferential Attachment*: Nodes with a higher degree will have a higher probability to be connected to a new node. This property causes that most nodes in the network will have only a few edges, and a few nodes gradually turn into hubs, i.e., their degree greatly exceeds the average.

These two features are responsible for the power-law distribution of scale-free networks. Many real-world networks, such as genetic networks or the actor network, follow a power-law distribution [BA99].

Barabasi and Albert [BA99] introduced a model capturing the properties of scale-free networks— the Barabasi-Albert Model (BA). The properties of the BA model that we use to build our performance model are:

- *Degree Exponent*, γ: It is the exponent of the power-law distribution of scale-free networks. It plays an essential role in predicting many properties of these networks, e.g., the highest node degree. The degree exponent of many real networks is between 2 and 3 [BA99]
- *Growth Parameter*, m: At each time step, a new node joins the network with m edges that connect it to m existing nodes.
- *Probability of a Node with Degree k*, ρ_k: Given the growth parameter m, the probability that a randomly chosen node has degree of k is given by:

$$\rho_k = \frac{2m(m+1)}{k(k+1)(k+2)}$$

- *Number of Edges*, $|E|$: In the BA model $|E|$ is given by:

$$|E| = m \cdot |V|$$

- *Largest Node Degree*, k_{max}: The expected value of the largest node degree in the BA is:

$$k_{max} \sim |V|^{\frac{1}{\gamma-1}}$$

- *Lowest Node Degree*, k_{min}: It is the minimum degree in the network. For k_{min} there is no characterization, each graph can have different values of k_{min}.

5.3.2 The Number-of-Buckets Model

Table 5.1 summarizes the notation that we will use to introduce our number-of -buckets model.

Notation	Description
$Bucket^{ini}_{Full}$	Number of full buckets, i.e., bucket with $maxEdges$ edges, after the initialization phase
$Bucket^{ini}_{Full_u}$	Number of full buckets generated for a given node u after the initialization phase
$Edges_{NFB}$	Number of edges placed in non-full buckets after the initialization phase
$Buckets_{Exp}$	Expected number of buckets
$Dummy_{Exp}$	Expected number of dummy edges
ρ_k	Probability of a node with degree k
k_{min}	Value of the smallest node degree of a given graph G
k_{max}	Value of the largest node degree of a given graph G
$deg(u)$	Degree of a given node u

Table 5.1: Number-of-Buckets Model - Notation

Recall that after the initialization phase of the bucketization algorithm, some buckets are full, and some are not. Lemma 5.8 captures the number of buckets that have reached their maximal capacity after the initialization phase of the algorithm.

Lemma 5.8. *The number of full buckets after the initialization phase of the algorithm is:*

$$Bucket_{Full}^{ini} = \sum_{k=k_{min}}^{k_{max}} \left(|V| \cdot \rho_k \cdot \left\lfloor \frac{k}{maxEdges} \right\rfloor \right)$$

Proof. Given a graph $G = (V, E)$ and a node $u \in V$ with degree $deg(u)$, the number of full buckets generated for u after the initialization phase is $Bucket_{Full_u}^{ini} = \left\lfloor \frac{deg(u)}{maxEdges} \right\rfloor$. $Bucket_{Full_u}^{ini}$ is calculated regardless of the other nodes in G. Next, it is required to calculate $Bucket_{Full_u}^{ini}$ for all nodes $u \in V$. According to the properties of the BA model, the probability that a randomly chosen node has a degree of k is given by ρ_k. Then, the total number of nodes with degree k is $|V| \cdot \rho_k$. For all the nodes with degree k, the total number of buckets is $|V| \cdot \rho_k \cdot \left\lfloor \frac{deg(u)}{maxEdges} \right\rfloor$. Finally, to estimate the total number of buckets after the initialization phase, we have to consider the degree of all the nodes. The degree value of the nodes is between k_{min} and k_{max}. □

If we know the number of full buckets, we know the number of edges that the bucketization algorithm has stored in full buckets. Then we can calculate the number of edges stored in non-full buckets, see Lemma 5.9.

Lemma 5.9. *The number of edges that have been assigned to buckets that are not full is:*

$$Edges_{NFB} = |E| - Bucket_{Full}^{ini} \cdot maxEdges$$

Proof. The number of edges already stored in full buckets after the initialization phase is $Bucket_{Full}^{ini} \cdot maxEdges$. We subtract this number from the total number of edges $|E|$ to obtain $Edges_{NFB}$. □

Using the previous two lemmas, Lemmas 5.8 and 5.9, we introduce the range of the number-of-buckets Model, see Theorem 5.10.

Theorem 5.10. *Given a graph $G = (V, E)$ that follows the BA Model, the expected number of buckets $Bucket_{Exp}$ is in the range:*

$$\begin{aligned} Bucket_{Full}^{ini} + \left\lceil \frac{Edges_{NFB}}{maxEdges} \right\rceil &\leq Bucket_{Exp} \\ &\leq Bucket_{Full}^{ini} + \frac{11}{9} \cdot \left\lceil \frac{Edges_{NFB}}{maxEdges} \right\rceil \end{aligned} \tag{5.1}$$

Proof. The lowest value of the range is the number of buckets obtained with the optimal bucketization. With an optimal bucketization, the non-full buckets are merged so that their edges, $Edges_{NFB}$, fill exactly $\left\lceil \frac{Edges_{NFB}}{maxEdges} \right\rceil$ buckets. For the upper bound of the model, the worst performance ratio of the FFD approach used in our algorithm is $\frac{11}{9}$ of the optimal solution. Consequently, the upper bound is the sum of the number of full buckets after the initialization phase, $Bucket_{Full}^{ini}$, and the number of buckets after the merging in the worst case, i.e., $\frac{11}{9} \cdot \left\lceil \frac{Edges_{NFB}}{maxEdges} \right\rceil$. □

Corollary 5.11 provides a range of the expected number of dummy edges.

Corollary 5.11. *Given a graph $G = (V, E)$ that follows the BA Model, the expected number of dummy edges, $Dummy_{Exp}$, is in the range*

$$\left(Bucket_{Full}^{ini} + \left\lceil \frac{Edges_{NFB}}{maxEdges} \right\rceil \right) \cdot maxEdges - |E| \leq Dummy_{Exp} \leq \left(Bucket_{Full}^{ini} + \frac{11}{9} \cdot \left\lceil \frac{Edges_{NFB}}{maxEdges} \right\rceil \right) \cdot maxEdges - |E| \tag{5.2}$$

Proof. The lower bound of the expected number of buckets from Theorem 5.10 multiplied with $maxEdges$ yields the total number of edges stored in the buckets. Subtracting from this number the real number of edges yields the lower bound of expected dummy edges. The analogous argument applies to the upper bound. □

Discussion: The number-of-buckets model helps us to predict query performance. Depending on the type of queries, our bucketization algorithm distributes the query workload between the client and the server, e.g., with neighbor queries, the client has to filter possible false positives. Lemma 5.8 gives the number of buckets that do not generate false positives because they are full and store edges belonging to the same node. This number depends not only on the characteristics of the given graph, e.g., distribution of the number of edges per node, but also on the parameter $maxEdges$. We obtain the percentage of buckets that have false positives by comparing $Bucket_{Full}^{ini}$ to the expected number of buckets from Theorem 5.10. Buckets that contain false positives result in more work at the client. That is, a low percentage of full buckets increases the average query processing effort at the client. Concerning adjacency queries, the server performs all the query processing. The number of dummy edges affects query performance at the server. Preliminary experiments of ours show that more dummy edges increase the query execution time at the server proportionally. Another parameter that affects query performance at the server is the number of buckets. The reason is that the server has to

perform a lookup in the set of buckets to answer queries. Then a large number of buckets also affects the query performance at the server.

All the properties about our bucketization algorithm that affect query performance, like the ones discussed in the previous paragraph, are summarized in our query-cost model, Subsection 5.3.3.

5.3.3 Query-Cost Model

In our query-cost model, we assume that queries are executed without using index structures on both a given graph and its corresponding bucketization structure.

Let a graph G, its corresponding bucketization structure BS_G and a query Q be given. We use $RTserver_Q^G$, $RTclient_Q^G$, $RTserver_Q^{BS_G}$ and $RTclient_Q^{BS_G}$ to denote the runtime complexity of executing query Q over graph G at the server, the runtime complexity of executing query Q over graph G at the client, the runtime complexity of executing query Q over the bucketization structure BS_G at the server, and the runtime complexity of executing query Q over the bucketization structure BS_G at the client, respectively.

Definition 5.24: Query Performance Ratio

The **query performance ratio** of executing a given query Q over a given graph G and its corresponding bucketization structure BS_G at the server side is

$$\mathcal{R}server_Q = \frac{RTserver_Q^{BS_G}}{RTserver_Q^G}$$

and the query performance ratio at the client side is

$$\mathcal{R}client_Q = \frac{RTclient_Q^{BS_G}}{RTclient_Q^G}$$

We start by analyzing the processing of neighbor queries, followed by adjacency queries. We assume that a given graph and its corresponding bucketization structure do no use index structures. Then a single lookup of an edge in a given graph G has a complexity of $\mathcal{O}(|E|)$. In the bucketization structure, a single lookup of a node in the index information has complexity of $\mathcal{O}(|V|)$, and a single lookup of a bucket in the set of buckets has a complexity of $\mathcal{O}(Buckets_{Exp})$, where $Buckets_{Exp}$ is the expected number of buckets, Theorem 5.10.

The encryption and decryption complexity depends on the security parameter, Definition 3.5 of the underlying encryption scheme [PU09]. For instance, in the case of AES, the encryption and decryption complexity depends on the length of the message m to be encrypted, $|m|$, [OEHB11]. Then, the complexity of the encryption/decryption process is $\mathcal{O}(|m|)$. In our approach, we use two types of encryption schemes, deterministic and probabilistic. Each encryption scheme has a secret key with a constant size. Then we consider two different security parameters— one for each type of encryption. We use ds and ps to denote the security parameter of the deterministic encryption and the probabilistic encryption, respectively. The complexity of deterministic encryption and decryption is $\mathcal{O}(ds)$, and the complexity of the probabilistic one is $\mathcal{O}(ps)$.

Table 5.2 summarizes the notation that we will use to introduce our query-cost model.

Notation	**Description**
$\mathcal{R}server_Q$	Query performance ratio of executing at the server a given query Q
$\mathcal{R}client_Q$	Query performance ratio of executing at the client a given query Q
$RTserver_Q^G$	Runtime of executing a given query Q over graph G at the server
$RTserver_Q^{BS_G}$	Runtime of executing a given query Q over the bucketization structure BS_G at the server
$RTclient_Q^G$	Runtime of executing a given query Q over graph G at the client
$RTclient_Q^{BS_G}$	Runtime of executing a given query Q over the bucketization structure BS_G at the client
$\|B_{G_u}\|$	Number of buckets that store the edges of a given node u
$\|B_G\|$	Number of buckets
$\|Dummy\|$	Number of dummy edges
$\mathcal{O}(ds)$	Time complexity of deterministic encryption and decryption
$\mathcal{O}(ps)$	Time complexity of probabilistic encryption and decryption

Table 5.2: Query-Cost Model - Notation

Query-Cost Model for Neighbor Queries

Lemma 5.12. *Let a Graph $G = (V, E)$, its bucketization structure BS_G and a neighbor query $Q_{Neigbhor}$ (G, u) be given. The server-side and the client-side performance ratio of executing the given query are:*

$$\mathcal{R}server_{Q_{Neighbor}(G,u)} = \frac{\mathcal{O}\left(\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot |B_G|\right)}{\mathcal{O}(|E|)}$$

$$\mathcal{R}client_{Q_{Neigbhor}(G,u)} = \mathcal{O}\left(\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot maxEdges\right)$$

Proof. In the given graph G, we need to access the edges E stored at the server to retrieve all edges that belong to node u. Then, the effort of executing a neighbor query on the server side is $RT^{G}_{Q_{Neighbor}(G,u)} = \mathcal{O}(|E|)$. At the client, no work is necessary. With our bucketization in turn, the following steps are required:

1. Encrypt Node u for querying. The effort is $\mathcal{O}(ds)$.
2. Retrieve the set of bucket identifiers of node u from the index information. This step has a complexity of $\mathcal{O}(|V|)$.
3. Decrypt the set of bucket identifiers. The effort is $\mathcal{O}(ps)$.
4. For each bucket identifier, one lookup in the set of buckets $\mathcal{B}_G$ is required. The number of buckets that store the edges of node u is $|B_{G_u}| = \left\lceil \frac{deg(u)}{maxEdges} \right\rceil$. The complexity of this step is $\mathcal{O}\left(\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot (|B_G|)\right)$.
5. Decrypt and filter the $\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot maxEdges$ edges. The decryption and filtering has a complexity of $\mathcal{O}\left(\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot maxEdges\right)$.

The server performs Steps 2 and 4, the client Steps 1, 3 and 5. The step with the highest complexity at the client is Step 5, and at the server it is Step 4. Consequently, the effort for executing a neighbor query at the server and at the client is:

$$RT^{BS_G}_{Q_{Neighbor}(G,u)} = \mathcal{O}\left(\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot (|B_G|)\right)$$
$$RT^{BS_G}_{Q_{Neighbor}(G,u)} = \mathcal{O}\left(\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot maxEdges\right)$$

Finally,

$$\mathcal{R}server_{Q_{Neighbor}(G,u)} = \frac{\mathcal{O}\left(\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot (|B_G|)\right)}{\mathcal{O}\left(|E|\right)}$$
$$\mathcal{R}client_{Q_{Neighbor}(G,u)} = \mathcal{O}\left(\left\lceil \frac{deg(u)}{maxEdges} \right\rceil \cdot maxEdges\right)$$

□

Query-Cost Model for Adjacency Queries

Lemma 5.13. *Let a Graph $G = (V, E)$, its bucketization structure BS_G and an adjacency query $Q_{Adjacency}(G, u, v)$ be given. The server-side and client-side performance ratio of executing the given query are:*

$$\mathcal{R}server_{Q_{Adjacency}(G,u,v)} = \frac{\mathcal{O}\left(|E| + |Dummy|\right)}{\mathcal{O}\left(|E|\right)}$$

$$\mathcal{R}client_{Q_{Adjacency}(G,u,v)} == \mathcal{O}(ds)$$

Proof. In the given graph G, in order to check whether a queried edge (u, v) exists in E, one needs to execute one lookup on the edges E. Then, the effort of executing an adjacency query at the server is $RTserver^{G}_{Q_{Adjacency}(G,u,v)} = \mathcal{O}(|E|)$. At the client, no work is necessary. With our bucketization in turn, the following steps are required:

1. Encrypt the queried edge (u, v) for querying. The effort is $\mathcal{O}(ds)$.
2. Execute one lookup in the encrypted edges that are stored in the set of buckets $\mathcal{B}_G$. The complexity of this step is $\mathcal{O}(|E| + |Dummy|)$.

Step 1 takes place at the client, it is an encryption operation with complexity $\mathcal{O}(ds)$. So the ratio of executing the given query at the client is $\mathcal{R}client_{Q_{Adjacency}(G,u,v)} = \mathcal{O}(ds)$.

At the server, the effort of executing the given query over the bucketization structure BS_G is $RT^{BS_G}_{Q_{Adjacency}(G,u,v)} = \mathcal{O}\left(|E| + |Dummy|\right)$. Then, the effort of executing the given query at the server is $\mathcal{R}server_{Q_{Adjacency}(G,u,v)} = \frac{\mathcal{O}(|E|+|Dummy|)}{\mathcal{O}(|E|)}$. □

Discussion Query-Cost Model

From the Query-Cost Model, we can learn that for adjacency and neighbor queries the parameter $maxEdges$ plays an important role regarding the query-execution effort at client and server. If the parameter $maxEdges$ increases, the number of dummy edges increases, and the server must take more effort to answer queries. At the client, to answer neighbor queries, if the parameter $maxEdges$ increases, the workload at the client increases as well. That is because the client has to filter more false positives. Therefore, we suggest for scale-free networks that the parameter $maxEdges$ should take smaller values, exactly m. In the next section, we demonstrate through experiments how this parameter affects the performance of our bucketization algorithm.

5.4 Experiments

In this section, we present experiments to evaluate the accuracy of our number-of-buckets model and the performance of our bucketization approach. Notice that the query-cost model is derived from the number-of-buckets model. Consequently, its accuracy mainly depends on the accuracy of the number-of-buckets model.

5.4.1 Experiment Setup

Input Datasets

In our experiments, we use synthetic and real datasets.

Synthetic Datasets: We used Networkx [SS08] to generate eight different undirected graphs that follow the BA Model. Table 5.4 shows the characteristics of these graphs, where $|V|$ is the number of nodes, m the growth parameter, and $|E|$ the number of edges. Related experimental studies on secrecy preserving on graph-structured data, [YFY14, ZP08], have considered synthetic graphs with nodes between 3000 and 25000. We vary the number of nodes from 5000 to 150000. For graphs with 5000 and 10000 nodes, we set the growth parameter m equal to 6 and 8. For graphs with 40000 and 150000 nodes, we set the growth parameter m equal to 8 and 10. The number of edges of each graph depends on the number of nodes and the growth parameter m.

Real Datasets: Our bucketization approach can work with any graph. Table 5.3 the real datasets that we used in our experiments. Barabasi and Albert in [BA99] proved that the Actor and the Web-network are scale-free. In the Actor-network, the nodes represent actors and movies. An edge connects a movie with an actor

Synthetic Data	$\|V\|$	m	$\|E\|$
$G1$	5000	6	29964
$G2$	5000	8	39936
$G3$	10000	6	59964
$G4$	10000	8	79936
$G5$	40000	8	319936
$G6$	40000	10	399900
$G7$	150000	8	1199936
$G8$	150000	10	1499900

Table 5.3: Synthetic Graph-structured Datasets

Datasest	$\|V\|$	$\|E\|$	**Type of Network**	m
Actor-network [Kun13]	1048575	1137725	Scale free Network	4
Web-network [LLDM09]	2381903	2312497	Scale-free Network	5
Citation-network [LK14]	27770	352807	Non-scale-free Network	NA

Table 5.4: Real Graph-structured Datasets

who has played in it. The Actor-network exhibits the preferential attachment feature. Namely, if an actor has played in more movies, a casting director is more familiar with his or her skills. The nodes in the Web network are web pages, and the edges represent hyperlinks between them. The nodes in the Citation network are articles, an edge is created between articles a and b if article a cites b.

Queries

Based on initial experiments and the Query-Cost Model, Section 5.3.3, we observe that node degree plays an essential role in the query performance evaluation. Therefore, one has to select carefully the object that will be part of an experiment sample, i.e., the queries, to have a representative sample. In all experiments that follow, the experiment sample consists of actual nodes from the graph that is being queried. In the following, we present the experiment results for these type of queries. In the context of neighbor queries, there are two kinds of nodes with very different query performance: hubs, and non-hubs. To have equally represented hubs and non-hubs in our query sample, we divide neighbor queries into two groups: $RandomQ_{Neighbor}(G, u)$ and $HubQ_{Neighbor}(G, u)$. For the group of queries $RandomQ_{Neighbor}(G, u)$, we select the input node u randomly from the set

of nodes V without considering the hubs in the graph. For $HubQ_{Neighbor}(G, u)$, we identify the hubs in the graph and use them as input. For adjacency queries, $Q_{Adjacency}(G, u, v)$, we select at random the nodes u, v from the set of nodes V. The execution time of adjacency queries depends on the total number of edges, including dummy edges (Section 5.3.3). Then, a distinct consideration of hubs is not necessary in this case.

Evaluation Measures

We use seven metrics which let us evaluate the accuracy of the number-of-buckets model (NBM) and the performance of the bucketization approach.

The NBM metrics are:

- $|Bucket_G|$**:** This metric quantifies the number of buckets obtained when applying our bucketization algorithm to a given graph G.
- $Dummy_G$**:** This metric quantifies the percentage of dummy edges when applying our bucketization algorithm to Graph G.
- $Bucket^{ini}_{Full}$**:** This metric quantifies the percentage of buckets that are full after the initialization of the bucketization algorithm on the given graph G.

The bucketization performance metrics are:

- $RTserver^{BS_G}_Q$**:** This metric quantifies the server query-processing time when using our bucketization structure of graph G, i.e., the time required by the server in order to answer a query sent by the client.
- $RTclient^{BS_G}_Q$**:** This metric quantifies the client query processing time when using our bucketization structure of graph G, i.e., the time required by the client to decrypt the results returned from the server and filter false positives.
- $TotalRT^{BS_G}_Q$**:** This metric quantifies the total query processing time using our bucketization structure, i.e., it adds up the processing time at the client and at the server.
- $Total\mathcal{R}_Q$**:** This is the ratio of the total query processing time using our bucketization structure to that using the given graph G.

5.4.2 Results

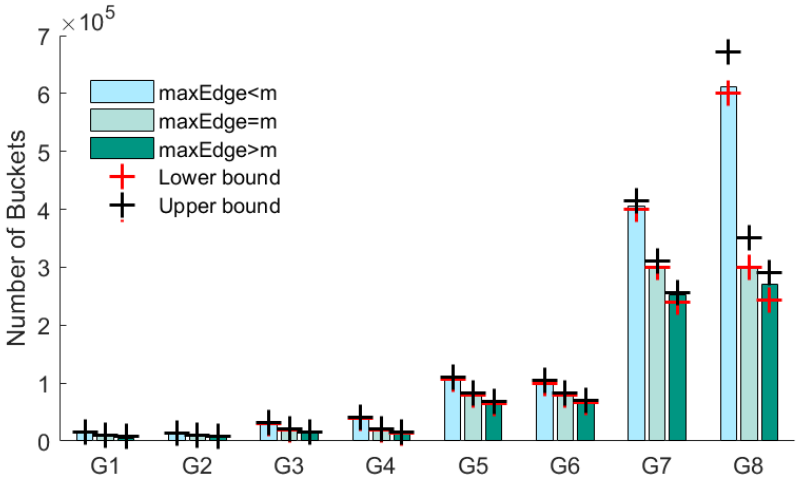
We now present the results of the experiments. We start by discussing the evaluation of the number-of-buckets model, and then the performance. Our experiments

evaluate the trade-off between secrecy and performance and study the effect of each of the metrics defined in the previous subsection. We develop our number-of-buckets model based on the characteristics of scale-free networks. Therefore, for the evaluation of this model, we use only the scale-free datasets, namely all synthetic datasets, $G1, \ldots, G8$, and the two scale-free real datasets, the Actor and Web networks. For the performance evaluation, we use all the real datasets. In the experiments, we have varied the parameter $maxEdges$.

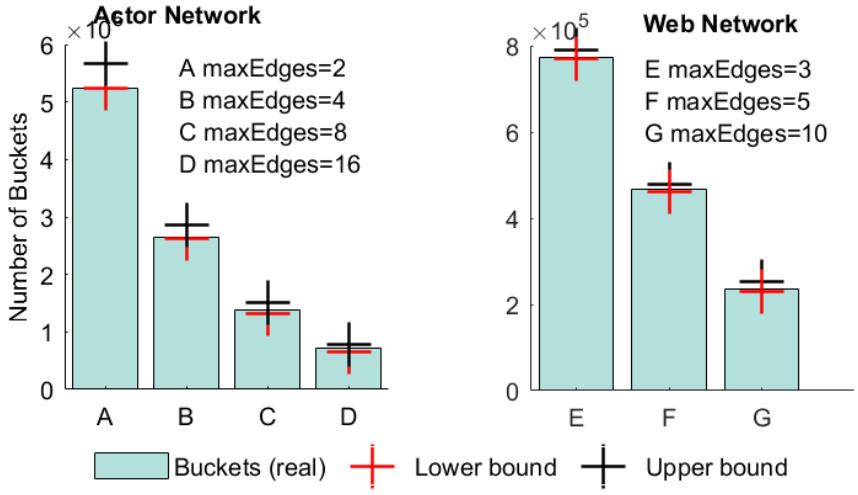
Number-of-Buckets Model Evaluation

$|Bucket_G|$: Figure 5.9 shows the numbers of buckets obtained with the synthetic and real datasets. For both types of datasets, we have used different values for the parameter $maxEdge$. The markers on each plot are the lower and upper bounds calculated with our number-of-buckets model. In all our experiments, the number of buckets obtained is always inside the range calculated with Theorem 5.10. Moreover, if the parameter $maxEdges$ is lower than the growth parameter m, the number of buckets obtained is between the lower bound and the middle of the range given by the number-of-buckets model. Contrary, if $maxEdges > m$, the number of buckets gets closer to the upper bound of the number-of-buckets model. We explain this effect as follows: In scale-free networks, most of the nodes in the graph have degree equal to m. If we set the parameter $maxEdge$ equal to m, most buckets will reach their maximal capacity after the initialization phase, and fewer buckets will remain for the merging phase. Then the total number of buckets gets closer to the optimal solution of our algorithm, which is the lower bound of our estimation. If we set the parameter $maxEdges$ with values greater than m, after the initialization phase, most of the buckets will not have reached their maximal capacity, and our bucketization algorithm will consider them for the merging phase. In the merging phase, because of the heuristic used, it is not always possible to reach an optimal solution. Therefore, the number of buckets obtained gets closer to the upper bound of the number-of-buckets model.

$Dummy_G$: We calculate the percentage of dummy edges in comparison with the size of the given graph for the synthetic data and the real datasets. Table 5.5 shows the average percentage of dummy edges for the synthetic datasets. Table 5.6 shows the exact percentage of dummy edges for the real datasets. We can observe in both tables that the number of dummy edges needed increases as the parameter $maxEdges$ takes values greater than m. More dummy edges mean a larger database, which is likely to affect the efficiency of the querying process on the server-side. We will examine the query performance in the next subsection.



(a) Synthetic Datasets



(b) Real Datasets

Figure 5.9: $|Bucket_G|$ obtained for the synthetic and real datasets

$Bucket^{ini}_{Full}$**:** Table 5.7 shows the average percentage of full buckets after the initialization phase of our bucketization algorithm for the synthetic datasets. Table 5.8 shows the exact percentage of full buckets after the initialization phase. The number of full buckets decreases as the parameter $maxEdges$ takes values higher than m. If a bucket stores edges that belong to different nodes, dummy edges or both, the client will require more query processing effort. Full buckets after the initialization phase contain edges that belong to a single node. More full buckets right after initialization let our algorithm to come closer to the optimal solution.

$maxEdges$	$Dummy_G$
$1 < maxEdges < m$	1.217%
$maxEdges = m$	0.889%
$maxEdges > m$	26.513%

Table 5.5: Percentage of Dummy Edges for the Synthetic Datasets

Dataset	$maxEdges$	$Dummy_G$
Actor Network	2	0.428%
	4	0.748%
	16	7.629%
Web Network	3	0.223%
	5	1.112%
	10	6.349%

Table 5.6: Percentage of Dummy Edges for the Real Datasets

An optimal solution implies fewer buckets for the merging process, fewer dummy edges, and fewer false positives when querying.

$maxEdges$	$Bucket^{ini}_{Full}$
$1 < maxEdges < m$	88.79%
$maxEdges = m$	86.81%
$maxEdges > m$	46.65%

Table 5.7: Average percentage of Full Buckets after the initialization phase - Synthetic Datasets

Performance Evaluation

As in the previous section, we conducted our experiments with both synthetic and real datasets, and their results are very much the same. In what follows, due to its applicability to real-world scenarios, we present the results on the real datasets.

$RTserver^{BS_G}_Q$**:** Figure 5.10(a) shows the average query-processing time of the server for random neighbor queries, hub neighbor queries, and adjacency queries. For random and hub neighbor queries, $RandomQ_{Neighbor}(G, u)$ and $HubQ_{Neighbor}(G, u)$,

Dataset	$maxEdges$	$Bucket_{Full}^{ini}$
Actor Network	2	59.15%
	4	55.78%
	16	14.49%
Web Network	3	81.38%
	5	80.18%
	10	47.96%

Table 5.8: Percentage of Full Buckets after the initialization phase - Real Datasets

the query-processing time of the server increases as the parameter $maxEdges$ decreases. The reason is that when the parameter $maxEdge$ decreases, the number of buckets increases, which means that during query execution, the server has to retrieve more buckets. In contrast, the query-processing time of the server for adjacency queries, $Q_{Adjacency}(G, u, v)$, increases as the parameter$maxEdges$ takes higher values. The increase, in this case, is due to the number of dummy edges inserted. Our experiments in the previous subsection showed that the number of dummy edges needed grows, as the parameter $maxEdges$ increases.

$RTclient_Q^{BS_G}$**:** For this part of the evaluation, we only consider two random and hub neighbor queries, $RandomQ_{Neighbor}(G, u)$ and $HubQ_{Neighbor}(G, u)$. We omit adjacency queries because they do not require any post-processing on the client. Figure 5.10(b) shows the average query-processing time of the client. In the scale-free networks, i.e., the Actor and Web networks, the query-processing time at the client increases as the parameter $maxEdges$ takes larger values. In the Citation-network, with $maxEdges = 12$, i.e., the average degree of the network, the client query-processing time decreases compared to the time obtained with different values of the parameter $maxEdges$.

$TotalRT_Q^{BS_G}$**:** Figure 5.11 shows the total average query-processing time for random and hub neighbor queries and adjacency queries with the real datasets. For the queries $RandomQ_{Neighbor}(G, u)$ and $HubQ_{Neighbor}(G, u)$, the total execution time increases as the parameter $maxEdges$ decreases. This increment is related to the increase of the query-processing time of the server, which we have explained with the metric $P_{SQprocessing}$. For adjacency queries, the query-processing time of the server and the total query-processing time are the same. The reason is that adjacency queries do not require any post-processing on the client, i.e., the server performs all the query-processing work.

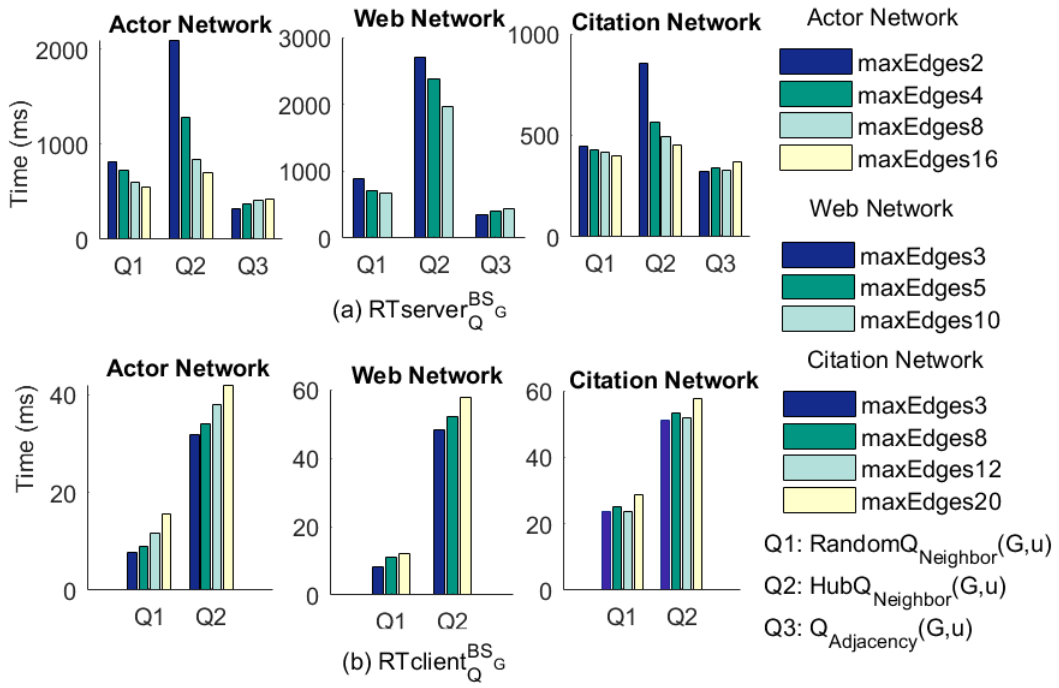


Figure 5.10: Server Query-processing Time, $RTserver_Q^{BS_G}$, and Client Query-processing Time, $RTclient_Q^{BS_G}$ -Real Datasets

We can see from the analysis of the query-processing time at the server, at the client, and the total query-processing that the best value to set the parameter $maxEdges$ in scale-free networks is the growth parameter m. In this kind of networks, most nodes have a degree equal to m so that most buckets will be full after the initialization phase. For non-scale-free networks, we observe that the best value to set $maxEdges$ is the average degree of the network. To evaluate the last metric, $Total\mathcal{R}_Q$, we set $maxEdges$ to the best option, i.e., $maxEdges = m$ for the scale-free networks and $maxEdges = 12$, average degree, for the non-scale-free network.

$Total\mathcal{R}_Q$**:** Figure 5.12 shows a comparison of the total query processing time for random and hub neighbor queries and adjacency queries using our bucketization structures and the original graphs of the three real datasets. For each diagram in Figure 5.12, each pair of points on the x-axis, i.e., 1 and 2, 3 and 4, 5 and 6, corresponds to one type of query. The first pair corresponds to random neighbor queries, $RandomQ_{Neighbor}(G, u)$, the second one to hub neighbor queries, $HubQ_{Neighbor}(G, u)$, and the third one to adjacency queries, $Q_{Adjacency}(G, u, v)$. In each pair, the first point represents the total query-processing time using our bucketization structure, and the second one is the total time using the original graph.

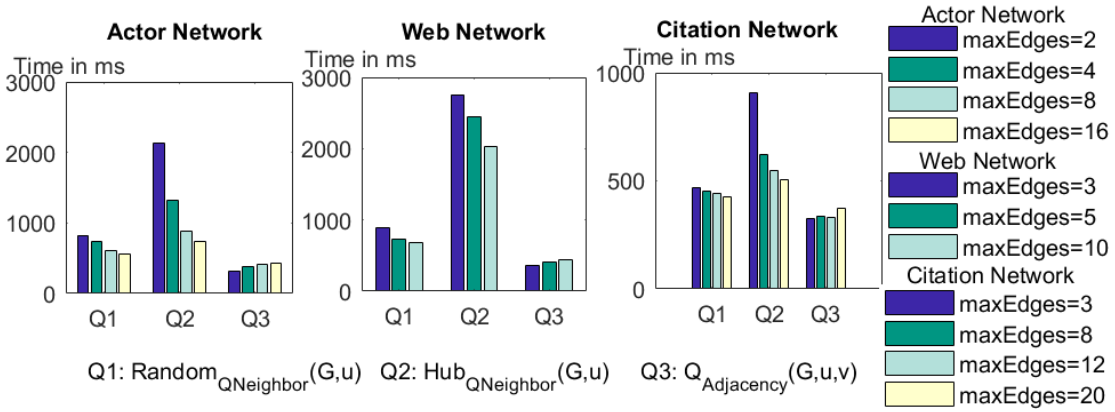


Figure 5.11: Total Average Query-processing Time - Real Datasets

We deem the total execution time on the original graph the optimum. So we evaluate our approach depending on how much the query-processing time increases in comparison with the original graph. In the Actor-network, the execution of random neighbor queries, $RandomQ_{Neighbor}(G, u)$, with our bucketization approach is on average 3.44 times slower than with the original graph, the execution of hub neighbor queries, $HubQ_{Neighbor}(G, u)$, is 5.12 times slower and the execution of adjacency queries, $Q_{Adjacency}(G, u, v)$, is 2.88 times slower. In the Web-network, the execution of random neighbor queries, $RandomQ_{Neighbor}(G, u)$, with our bucketization approach is on average 2.90 times slower than with the original graph, the execution of hub neighbor queries, $HubQ_{Neighbor}(G, u)$, is 10.15 times slower, and the execution of adjacency queries, $Q_{Adjacency}(G, u, v)$, is 4.76 times slower. In the Citation-network, the execution of random neighbor queries, $RandomQ_{Neighbor}(G, u)$, with our bucketization approach is on average 4.51 times slower than with the original graph, the execution of hub neighbor queries, $HubQ_{Neighbor}(G, u)$, is 4.78 times slower, and the execution of adjacency queries, $Q_{Adjacency}(G, u, v)$, is 4.93 times slower.

To summarize, in the scale-free networks, i.e., the Actor and Web-networks, except for hub neighbor queries, $HubQ_{Neighbor}(G, u)$, with our approach, the query execution time is 3.5 times slower than with the original graph. In the non-scale-free network, i.e., the Citation-network, with our approach, the query execution time is approximately 5 times slower than with the original graph. In our opinion, these are reasonable prices for secrecy guarantees. So our bucketization approach is practical and feasible for secrecy for graph-structured data.

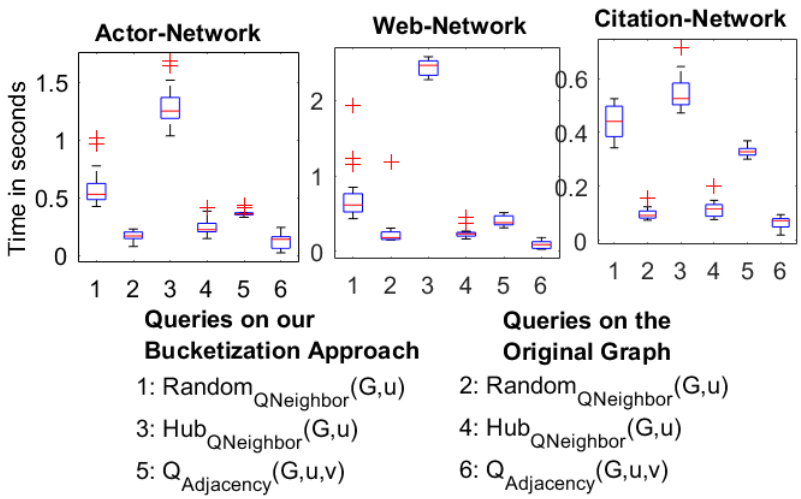


Figure 5.12: Total query processing time - Real datasets

5.5 Related Work

In this section, we first review existing secrecy notions. Then we analyze work on bucketization for relational databases and on secure storage of graph-structured data.

5.5.1 Secrecy Notions - Related Work

Different secrecy notions have been proposed in the literature. Each secrecy notion considers different secrecy guarantees and deals with different adversary models. Notions such as the ones presented by Fan et al. [FCC+15] and Zhang et al. [ZSW+14] offer guarantees against chosen-plaintext attacks and known-plaintext attacks, respectively. In our graph setting scenario, these guarantees are not enough because the edges of the graph can also reveal information. Wang et al. [WL06] define a secrecy notion for XML documents, which is based on the definition of perfect secrecy. Their secrecy notion considers adversaries who have access to the secretized XML documents and some metadata needed to perform queries. We additionally assume that the adversary can observe query executions over time. Goh et al. introduce a secrecy notion known as searchable encryption, which was later extended in [CGKO11, VLSD+10, KPR12]. Searchable encryption is a technique that allows performing keyword searches in encrypted documents. An index is generated based on the plaintext data to increase the performance of the search process. The index consists of two data structures: an array that stores, for each keyword w, the encrypted set of identifiers of all documents containing

w, and a look-up table which includes, for each w, information that allows to locate and decrypt the elements from the array. A trapdoor, which is a deterministic algorithm run by the client, allows testing for the occurrence of a keyword in a document. Their secrecy notion considers two adversarial models, namely nonadaptive chosen keyword attacks (IND-CKA1) and adaptive chosen keyword attacks (IND-CKA2). These secrecy notions define secrecy for indexes to ensure that an adversary will not be able to learn the content of an encrypted document from its index. Formally, the secrecy notions IND-CKA1/2 comprise secrecy for trapdoors and guarantee that the trapdoors do not leak information about the keywords apart from the outcome and access pattern of a query. Similarly to the secrecy notion proposed in this chapter, IND-CKA1/2 use the concept of indistinguishability. However, our secrecy notion is applied to graphs, i.e., the challenger takes as input two graphs given by the adversary.

5.5.2 Bucketization on Relational Databases - Related Work

Secure database storage has been widely studied. However, existing techniques such as the ones presented in [HMT04, ABG+05] either cannot be applied to graph-structured data, or they do not cover our requirements R1 and R2. Approaches for graph-structured data, [SNS10] do not keep the information of the entire graph. Next, they cannot answer certain queries, such as neighbor and adjacency queries, which are essential needs when working with graphs [MP10]. Other approaches like the ones presented in [HMT04, ABG+05, HIM05] could exhibit unwanted behavior when being adapted to graph-structured data, e.g., leak information. Several approaches are based on bucketization. In this context, bucketization encrypts each tuple in an original relation as one string and assigns an index to each encrypted tuple. The approach generates indexes in such a way that more than one encrypted tuple could have the same index value. Encrypted tuples with the same index value are called partitions. Each index value is related to a partition of the domain of an original attribute. The server stores the secretized relation and index information. In what follows, we sketch two adaptations of these approaches to graphs and show that these alternatives are not appropriate to solve our problem.

With both adaptations, we represent the edges in a two-attribute relation, T_{Edges}, where each attribute stores one node of the edge. Borrowing from bucketization schemes for relational databases, two alternatives come to mind: one-dimensional bucketization and multidimensional bucketization.

One dimensional bucketization. Here, the domains of the two-attributes in T_{Edges} are considered as one domain and then divided into partitions. This solution

cannot be considered secure because it could exhibit some of the original graph structure.

Example 5.8. Consider the graph from Figure 5.13. If the bucketization algorithm assigns nodes A, B and C to different buckets, the connections between the buckets will share the same structure as the original graph. Table 5.9 shows the secretized relation. This solution represents a partition of nodes into buckets. The partitions are $[b1, \{A\}]$, $[b2, \{B\}]$, $[b3, \{C\}]$, $[b4, \{D, E\}]$. The relationships between the index values $(b1, b2)$, $(b2, b3)$ and $(b3, b1)$ share the same structure as the original edges E.

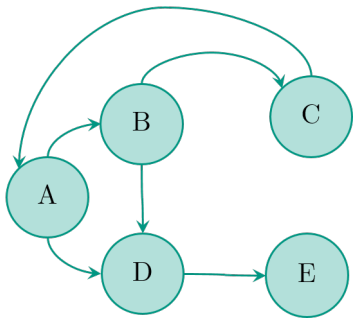


Figure 5.13: Graph - Example 5.8

e-tuple	**Node 1**	**Node 2**
$enc(A, B)$	$b1$	$b2$
$enc(A, D)$	$b1$	$b4$
$enc(B, C)$	$b2$	$b3$
$enc(B, D)$	$b2$	$b4$
$enc(C, A)$	$b3$	$b1$
$enc(D, E)$	$b4$	$b4$

Table 5.9: Secretized Relation of Example 5.8

Multidimensional bucketization. With this option, the domain of each attribute is partitioned individually. Given an optimal multidimensional bucketization, this bucketization can be secure. However, finding an optimal multidimensional bucketization concerning query performance is NP-hard [LDR06]. This NP--hard problem can be solved with heuristics such as in [LDR06] and [WD08]. But these solutions do not consider certain graphs characteristics such as grouping edges

of a node in the same partition to answer relevant graph queries, e.g., neighbor queries, efficiently. So these approaches do not solve our problem.

5.5.3 Secure Storage for Grap-Structured Data - Related Work

Secure storage for graph-structured data: Syalim et al. [SNS10] have proposed an approach that guarantees secrecy for the labels of nodes of a graph. They design their secrecy model to protect provenance metadata. Provenance metadata is information that allows tracing who has contributed to the creation of a document. The authors represent the provenance metadata as a directed acyclic graph. The labels of nodes store the provenance metadata of a specific document version, and the edges represent the relationships between a version of a document and its successors. Their approach use multiple layer encryption to guarantee that only authorized users, who have the corresponding decryption keys, can access the provenance metadata. Since the labels of the nodes are encrypted, an adversary, who does not possess the encryption keys, is unable to learn the original plaintexts. However, in general, an adversary can identify a node by its degree and determine the relationships between nodes without knowing the content of the labels, i.e., edge leakage. Our approach guarantees that neither nodes nor edges leak information. Regarding query processing, the authors consider two types of queries: access to the label of a node and access to the label of the parent nodes of a node. These two types of queries may be enough for provenance metadata. In general, however, different graph--specific types of queries should be supported. The authors in [HVS+10] present an approach for finding the shortest path between two nodes in a directed graph. The approach perturbs the edges to guarantee edge privacy. The perturbation modifies the structure of the graph to some extent. Therefore, query results only are approximate. As XML documents are a specific kind of graphs, we briefly turn to this research direction as well. Want et al. [WL06] proposed an encryption scheme for XML documents which considers different levels of granularity for encryption. Their approach divides the XML document into blocks of different sizes and encrypts each block as a whole. A block can contain subtrees of the XML document at any depth, e.g., parent and child elements, or just the content of chosen elements. The authors in [CRK+13] proposed a solution for evaluation of tree pattern queries in encrypted XML documents. The authors take as starting point that XML documents have a domain hierarchy, i.e., parent and child hierarchy. Each element of the XML document is given a position based on the domain hierarchy. The authors use the position of the elements to create a vector for each XML document. They encrypt the vectors to ensure secrecy. Similarly, a tree pattern query is transformed into a vector. The evaluation of a tree pattern query requires measuring the distance between the encrypted vector that represents the

XML document and the encrypted vector that represents the query. These two approaches [CRK+13, WL06] require the existence of a domain hierarchy such as parent-child, to create blocks or vectors, respectively. In graph-structured data, such a hierarchy typically does not exist.

To summarize, none of the related approaches we are aware of does address Requirements R1 and R2 nor features a model of the costs of query processing that considers relevant characteristics of the graph.

5.6 Summary

A core challenge when outsourcing a database is to preserve data confidentiality. In this chapter, we have studied this problem for graph-structured data, and we have proposed a secrecy model for this kind of data based on the concept of indistinguishability. Existing proposals, such as the ones presented in [FCC+15, ZSW+14, WL06], deal with different types of adversaries and different secrecy guarantees. Our secrecy definition guarantees that, given a secretized graph, an adversary cannot learn any information about the original graph beyond the information leakage specified. While a bucketization of the edges gives way to the secrecy envisioned here, as we have shown, finding an optimal bucketization is NP-hard. We have proposed a heuristic that guarantees that the worst bucketization solution will be off by a factor of $\frac{11}{9}$ of the optimal one. Next, to facilitate query planning, we predict the behavior of our algorithm, taking into account its parameters and properties of the input graph. That is, we propose a performance model that allows estimating the number of buckets and the query-processing complexity. Our experiments with both real and synthetic datasets confirm the accuracy of our model and the effectiveness of our approach.

6 Providing Secure Services to Users of Online Social Networks

In this chapter, we focus on the authorization mechanism. Here, we tackle the distrust of users towards the service providers to enable access to services to users of online social networks while keeping their information secret from unauthorized access. Specifically, we consider the scenario in which each user of the online social network system owns a portion of the data, and they are not allowed to access the data of others. However, they are allowed to access the result of a given query over the data owned by others based on the access policies specified in the OSN system. OSNs offer users different kinds of services. Here, we focus on location-based services (LBS) due to their increasing demand. OSNs that embraces LBS are known as mobile social networks (MSNs)[4].

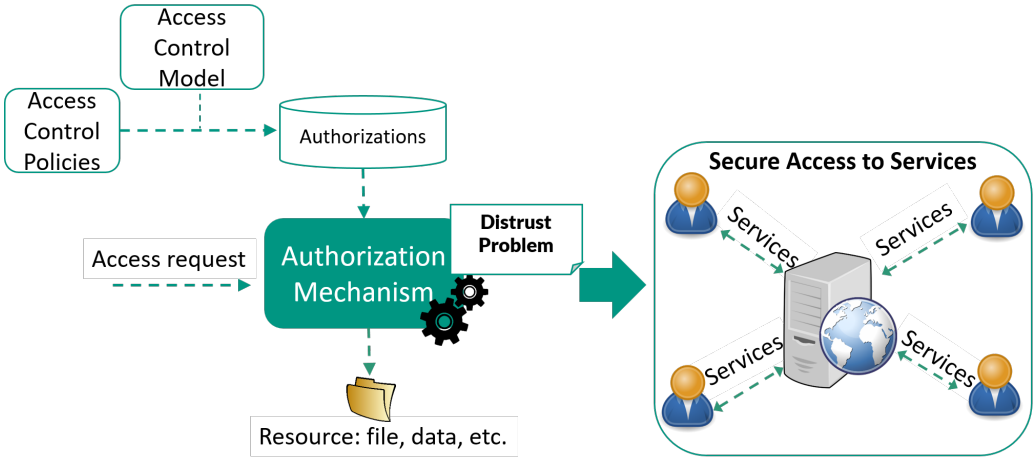


Figure 6.1: Access Control Mechanism - Distrust Problem

Mobile Social Networks have become popular in the last years. In these kinds of networks, each user specifies who is authorized to learn information about his

[4] An earlier version of this chapter was published in [SEGB] and in an extended version in [SEGB19a]

physical position, i.e., users establish authorization relationships with others. Due to the dynamic relationships between users that are inherent to human behavior, revocation of such privileges is a fundamental feature of this kind of network.

A typical architecture of MSNs consists of three major components [Fal11]: service providers, mobile users, and network infrastructure. Depending on the features supported by MSNs, the service providers can be dedicated servers, e.g., location server or access control server, that give services to the users through the network infrastructure [Fal11]. Mobile users, e.g., mobile phones or wearable devices, receive data or service results from the service providers. The network infrastructure transfers data from a source, e.g., service providers, to a destination, e.g., mobile users. A typical MSN architecture, which is enough to provide standard LBS and has been considered by existing work in the area [LLC+14, SYY+16, LYL+17], consists of users and two service providers: the LBS provider and the access control server (ACS). The LBS provider stores the positions of the users, and the ACS stores the authorization relationships.

The information stored in MSNs, i.e., physical positions of the users and authorization relationships, are sensitive information that should be kept secret from any adversary, including the service providers. This information is particularly sensitive because one can use it to infer further personal information. For instance, one can use the position of users to infer their state of health or personal preferences [ASV08]. The number of users of these services may indicate a lack of secrecy concerns. However, studies [SMSB14, MTR+09] have shown that mobile users are not aware of various important characteristics of these systems: the data they collect, who is using the data, and how they use it. That is, issues regarding mobile secrecy remain poorly understood by end-users.

Moreover, many users use these services without explicitly criticizing secrecy not because they are careless about their secrecy, but because they do not have a choice. Once the secrecy problems with LBS are solved, users may then switch to more secure systems when they become available. Consequently, the information stored by these systems has to be kept secret from unauthorized users, i.e., confidentiality.

One can achieve confidentiality by trusting the entity in the system that manages the access policies, i.e., the ACS. However, recent privacy breaches on existing social networks, such as the Facebook-Cambridge Analytica scandal [Com18, Con18], have put this into question. In such attacks, the ACS has allowed unauthorized entities, intentionally or unintentionally, to access private information. Collusion attacks where adversaries, including malicious users, collude with the service provider to gain unauthorized access to information, are an important problem in MSNs.

In this chapter, we study how to facilitate LBS in MSNs while providing secrecy guarantees to the users under collusion assumption. We focus on one specific service — querying friends within a given distance. As motivated earlier, we also cover revocation. Regarding collusion, we study the case of pairwise collusion in which a user tries to access unauthorized information by colluding with either the LBS provider or the ACS. We define our adversary model in Section 6.1.2. Regarding secrecy guarantees, we provide users with the following guarantees:

- $G_{position}$: Given a user u, only authorized entities can learn the physical position of u.
- $G_{distance}$: Given a user u, only authorized entities can learn the distance between their physical position and the one of u.
- $G_{authorization}$: Given two users u and v, an adversary will not be able to learn whether u or v are allowed to learn the distance between them.

Existing works in the area [LLC+14, LYL+17, WXL12], either does not consider collusion attacks or their system architectures assume trusted entities, which shifts the problem to the trusted entity. Besides, these approaches do not provide a rigorous specification of the collusion strategy as part of their adversary model, and they consider weaker adversaries, see Section 6.1.2. Next, none of the existing work we are aware of does protect $G_{authorization}$ against both the LBS provider and ACS. Section 6.5 contains a more detailed description of related works in the area.

Here, we propose two approaches, which combine existing encryption schemes, to allow users of MSNs to query friends within a given distance. Both approaches include a revocation feature and provide users with the secrecy guarantees $G_{position}$, $G_{distance}$ and $G_{authorization}$ under the collusion assumption. Section 6.1 describes our problem and defines our adversary model rigorously.

Section 6.2 presents our approaches to allow MSNs users to query friends within a given distance while keeping their information secret from unauthorized users. We name our approaches two-layer symmetric encryption (*2lSE*) and two-layer attribute-based encryption (*2lABE*). The main difference between them is that they use, among other encryption schemes, symmetric encryption and attribute-based encryption, respectively. We prove that both approaches fulfill our secrecy guarantees and analyze their time complexity to evaluate their performance. Our analyses tell us which approach is better at each entity involved in the system.

Finally, we conduct experiments to validate the results of our complexity analyses and to determine which approach performs better in practice. Section 6.4 presents the results of such experiments. Next to other insights, despite the advantages of the *2lABE* approach compared to the *2lSE*—a more straightforward key-manage-

ment and the storage of a single encrypted copy of each message at the service provider—, we found that the *2lSE* approach is on average twice as efficient in our scenario. Therefore, we propose to consider the *2lSE* approach, which not only solves the secrecy problem existing in MSNs but also is more performant than the *2lABE* approach.

6.1 Problem Definition

6.1.1 System Architecture

To provide LBS, MSNs require for each user u registered in the system the following information:

- The physical position of u denoted by $p_u = (x_u, y_u)$.
- The set of users who have allowed u to learn the distance between their position and the one of u. We call this set, the set of grantors of u, $Grantor_u$.
- The set of users to whom u has allowed to learn the distance between his position, p_u, and their positions. We call this set, the set of grantees of u, $Grantee_u$.

We assume that an off-the-shelf positioning technology, e.g., GPS, GSM, or BLE, inputs the physical position of each user in the system, and we consider a periodic positioning update policy [TSBV05]. Our system takes the physical positions as delivered by the positioning technology used. We assume further that the physical positions input to the system are real, i.e., users have honestly disclosed their location and have not manipulated it. We deem this assumption realistic – one can facilitate it by using technology such as hardware trusted sensors [SW10], which provide a signature to verify the validity of the position, or deep learning techniques, which detect abnormal traffic patterns to some extent [OKM19].

Regarding the system architecture, similarly to existing approaches such as the ones presented in [LLC+14, SYY+16], we consider a MSN system consisting of users and two service providers: the LBS provider and the ACS. The LBS stores the physical position of each user. The ACS stores the sets of grantors and grantees of each user. In addition to these entities, we consider a key-authority, which is only responsible for key issuing. Figure 6.2 illustrates our system architecture. We have omitted from the architecture illustration components related to the physical and network layers, such as physical transmission medium and routers, to focus on the essence of our work.

One can integrate our system into other kinds of architectures like IoT by placing the physical devices of our system (positioning and smart devices) at the perception layer, the network technology at the communication layer, the cloud platform at the middleware layer, and the LBS application at the application layer.

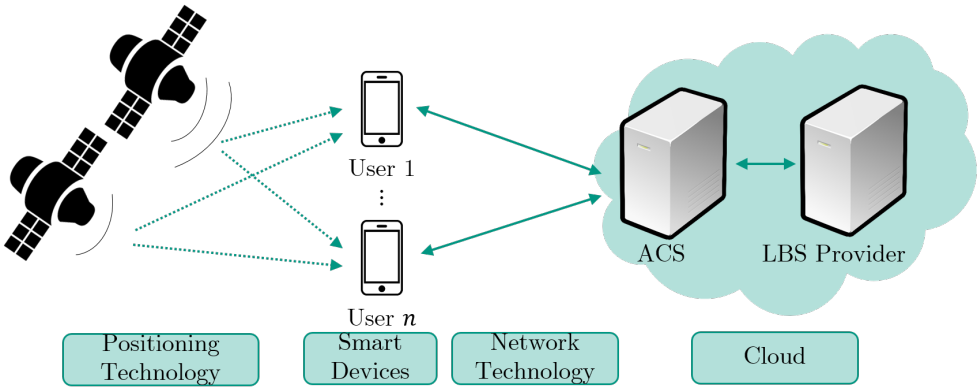


Figure 6.2: MSN System Architecture

Users can update their position and allow or revoke access from users at any time. Similarly as in Chapter 4, users send access requests, Definition 4.1. Since users want to establish who can learn the distance between their physical positions, the operation of an access request is "read," and the resources involved in the access request are the physical position of the users. For instance, the access request $\langle s, \ read, p_u \rangle$ indicates that user s request to read the physical position of user u. We write $\langle s, \ read, p_u \rangle = true$ to indicate that the access request is authorized. Otherwise, $\langle s, \ read, p_u \rangle = false$. We call a user who authorizes an access request and a user who receives an authorized access request, grantor and grantee, respectively.

Users must encrypt their information before outsourcing to the service providers to avoid them from learning information from the outsourced data. Users send their queries to the ACS to obtain LBS. The ACS and the LBS provider interact with each other to compute the query result, which the ACS sends to the user. Section 6.2 contains the details of our approaches. LBS support different types of queries. Here, we focus on range queries, one of the most important queries in MSNs [XZTS08].

Definition 6.1: Range Query in MSNs

Given a user u and a distance d, a **range query**, $Range(u, d)$, is a query that returns the users who are located within a distance d from u and who have authorized u to learn the distance between their physical positions and the

one of u. Formally, $Range(u, d) = \{v \in U \mid v \in Grantor_u \wedge dist(p_u, p_v) \leq d\}$, where U denotes the set of all users.

6.1.2 Adversary Model

To formally define an adversary model, as mentioned in Subsection 3.2.1, one needs to specify the following components: the collusion strategy, the computational strategy, the adversarial behavior strategy, the protocol execution strategy, and any further assumption regarding the adversary, which has not been covered by the previous components.

Next, we describe the leading alternatives for each of these components briefly and specify ours. We define our model based on a *just-strong-enough* principle, in which the selected alternatives for each of the mentioned components are just strong enough to model real-world adversaries under a collusion assumption with revocation capability, as we explain in the remaining of this subsection.

Collusion Strategy

The main collusion strategies are static and adaptive [HL10]. In the static strategy, the adversary receives a set of entities to collude with, and the honest entities remain honest during the protocol execution. In the adaptive strategy, the adversary can collude with any entity in the system during the protocol execution. Here, we consider the static strategy. Although the static strategy is weaker than the adaptive one, developing highly efficient schemes that are secure under the static strategy serves as an important step for constructing secure systems under the adaptive one [HL10]. We refer to any entity that participates in a collusion as an adversary.

Computational Strategy

The main computational strategies are polynomial and unbound. In the polynomial strategy, adversaries run in polynomial time, while in the unbound strategy, they do not have computational limits. Similar to existing works [HL10, LLC+14], we consider the polynomial strategy.

Adversarial Behavior Strategy

The main adversarial behavior strategies are semi-honest, covert, and malicious strategy [HL10]. In the semi-honest strategy, every colluding entity follows the protocol specification. That is, each entity performs the tasks assigned to it correctly. Adversaries can access to the state of all colluding parties to try learning information from it. In the covert strategy, adversaries may deviate from the protocol specification if honest entities fail to detect them. This strategy represents many real-world scenarios like financial or political settings, where the involved entities, i.e., companies or individuals, cannot afford the embarrassment, loss of reputation, and law punishment associated with being caught cheating. In the malicious strategy, the colluding entities can deviate arbitrarily from the protocol specification, according to the instructions of the adversary.

Since the security guarantees provided by schemes in the semi-honest strategy are weak, and approaches that offer security guarantees under the malicious strategy are inefficient to be implemented and used in practice [HL10], we opt for the covert strategy, which represents real-world adversaries. However, we restrain the covert strategy further to cover realistic scenarios where unauthorized entities try to gain access to the information. Here, we limit adversaries to deviate from the protocol only to advantage themselves, but they will not disadvantage any entity in the system. Advantaging and disadvantaging an entity means to give the entity unauthorized access to the information and to prevent the entity from accessing authorized information, respectively.

Protocol Execution Strategy

The main protocol execution strategies are stand-alone and concurrent-composition [Can01]. In the stand-alone strategy, the adversary can execute the protocol a single time, while in the concurrent-composition one, he can run the protocol several times. Considering adversaries in the concurrent-composition strategy is a harder problem to solve. However, having a scheme for the stand-alone strategy can be used to design approaches for the concurrent-composition one [HL10].

Following our *just-strong-enough* principle for specifying the adversary model, we consider the stand-alone strategy, and we extend it to fulfill our needs. Here, we allow the execution of the protocol twice, which lets us evaluate the secrecy guarantees met after a revocation takes place. Before describing the extension, we explain the need for it.

In a single protocol execution, the adversary can collude with the entities of the system based on the defined collusion strategy, send a set of queries, and get

their answers based on the information stored at each entity of the system at the moment of the protocol execution. In our scenario, the entities of the system store, among other information, the encrypted sets of grantors and grantees of the users. This information stays unmodified during the protocol execution. To evaluate the secrecy guarantees of a scheme under revocation, one needs to change the authorizations, i.e., adjust the set of grantors and grantees of the users involved in the revocation. Therefore, after updating the information, we need to allow the adversary to execute the protocol a second time. In the second protocol execution, the adversary can send queries and get their answers, but it is forbidden to collude with any entity. The purpose of the second execution is to evaluate whether a revoked user can gain unauthorized access to information that he could access on the first execution. We call this extension, the twofold-composition strategy.

In the remaining of this chapter, when we refer to adversaries, we imply adversaries with the power specified in our adversary model. Note that neither a user is an adversary of himself nor are the entities that a user has allowed them to access his data. Having specified our adversary model, it only remains to define the setting of the static strategy, i.e., determine the possible collusion scenarios.

Static Strategy Setting

Each execution of the protocol with our approaches involves four entities: the key-authority, a user, the ACS, and the LBS provider. First, the key-authority is an honest entity. It is only responsible for issuing keys during the registration of users in the system; it does not participate in any other phase of the protocol specification; it does not store any information. Details of our protocol are in Section 6.2. Thus, considering the key-authority as an honest entity does not shift the collusion problem to it. Second, the LBS provider, ACS, and the user can be adversaries. However, we limit ourselves to the case of pairwise collusion, which is in line with the selected adversary behavior strategy, i.e., the covert strategy. Since three entities participate in the critical phases of the protocol execution, i.e., access request, query, and revocation phases, at least one honest entity is needed to detect the deviation from the protocol of the adversaries.

Further Assumptions

In line with existing approaches [LYL+17, YPBV14, WXL12], we assume: the LBS provider and ACS do not collude with each other and they cannot identify users by observing their IP addresses in the connections. Regarding the first assumption, we find it reasonable because we assume that different companies run the ACS, the LBS provider, and the internet provider. Regarding our second assumption, one

can solve it by using hidden IP techniques that guarantee network intractability such as Virtual Private Network services [LK+07], TOR encryption (The Onion Router) [DMS04] or proxy servers [WWY+12].

Figure 6.3 summarizes the main strategies of each component of our adversary model. The strategies highlighted in dark color are the ones that we selected to define our adversary model, as explained in this subsection.

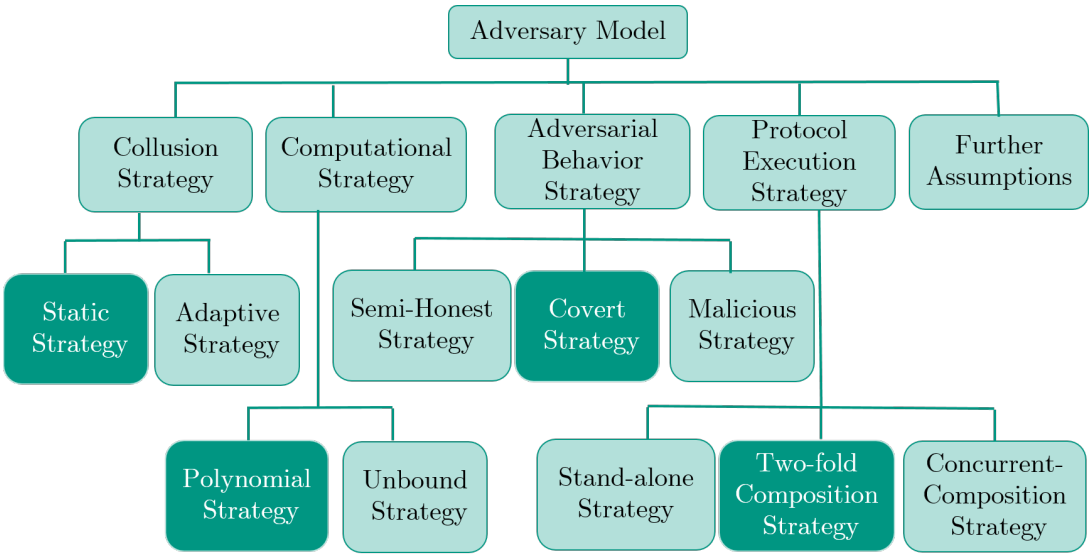


Figure 6.3: *Adversary Model Strategies*

6.1.3 Secrecy Guarantees

Based on our adversary model, we aim to offer the secrecy guarantees $G_{position}$, $G_{distance}$ and $G_{authorization}$ stated in the introduction of this chapter.

We note that given a resource r, an adversary is any entity that is not authorized to access r. Given a user u, authorized users, i.e., the grantors of u, can learn the distance between their position and the one of u, and the physical position of u.

6.1.4 Preliminaries and Notation

Before presenting our approaches, in the next section, Section 6.2, we introduce some notions and notation that we will use in this chapter.

Our approaches use different cryptographic techniques for encryption and key distribution. Regarding encryption, as we will explain in this section, our approaches

use symmetric encryption, asymmetric encryption, somewhat homomorphic encryption, and ciphertext-policy attribute-based encryption. See Section 3.1 for formal definitions of the previous mentioned encryption schemes.

We note that general CP-ABE schemes do not guarantee the security of the access policy used to encrypt a given ciphertext c, i.e., an entity who has access to c can learn who is authorized to decrypt c. Learning this information is against the secrecy guarantee $G_{authorization}$ that we aim to offer. However, we consider stronger CP-ABE schemes with *hidden policy* as proposed in [LGR+12].

Regarding key distribution, we use, in Section 6.2.1, the Diffie-Hellman key exchange protocol [DH76].

Definition 6.2: Diffie-Hellman key exchange

The **Diffie-Hellman key exchange** (DH) is a protocol that allows two parties, A and B, that have no prior knowledge of each other to establish a shared secret key jointly. The protocol is as follows:

1. A trusted party chooses and publishes two integers p and g, where p is large, e.g., 512 bits, and g is a primitive root modulo p.4A primitive root modulo p is an integer g such that $g \pmod p$ has multiplicative order $p-1$ [Rib12].
2. The parties A and B choose the secret integers, a and b, respectively.
3. A computes $Z_A \equiv g^a \pmod p$ and sends Z_A to B. B computes $Z_B \equiv g^b \pmod p$ and sends Z_B to A.
4. A computes the shared key $k_{ba} \equiv {Z_B}^a \pmod p$. B computes the shared key $k_{ab} \equiv {Z_A}^b \pmod p$. The shared key value is:

$$k_{ba} \equiv {Z_B}^a \pmod p \equiv (g^b)^a \pmod p \equiv g^{ab} \pmod p$$
$$\equiv (g^a)^b \pmod p \equiv {Z_A}^b \pmod p \equiv k_{ab}$$

Table 6.1 summarizes the encryption schemes and the encryption/decryption keys that we use to build our approaches.

6.2 Our Approaches

Users have to encrypt the information before outsourcing it to the ACS and the LBS provider to keep their information secret. There are different possibilities to encrypt the information. Due to the decryption and revocation overhead at the users-side, we disregard naive solutions such as the one of Example 6.1. The illustrated solution has the following shortcomings: it affects the storage capacity

Encryption scheme	Enc/Dec Keys	Description
Symmetric encryption ($\mathcal{SE}$)	k_u	Key of user u
	k_{uv}	Shared key between users u and v
Asymmetric encryption ($\mathcal{AE}$)	$\mathsf{pk}_{ACS}, \mathsf{sk}_{ACS}$	Public and secret keys of the ACS
	$\mathsf{pk}_{LBS}, \mathsf{sk}_{LBS}$	Public and secret keys of the LBS provider
Somewhat homomorphic encryption (SHE)	$\mathsf{pk}_H, \mathsf{sk}_H$	Public and secret keys for SHE
Ciphertext-policy attribute-based encryption (CP-ABE)	$\mathtt{mk}_{ABE}, \mathsf{pk}_{ABE}$	Master and public keys for CP-ABE
	sk_{ω_u}	Secret key for CP-ABE of user u, where ω_u is the set of attributes of u,

Table 6.1: Summary of Encryption Schemes and the Corresponding Keys Used

and limited processing resources of mobile devices, and revocation is not only inefficient for data owners, but it also requires authorized users to be online. Given a set S, let $|S|$ denote the cardinality of S.

Example 6.1. Assume that each user u encrypts his name with a key k_u, stores his encrypted name at the LBS provider, and distributes k_u to all authorized users. Such a solution has several problems. First, u has to store as many keys as grantors. Second, during query processing, u receives, as a result, a set of encrypted names corresponding to users who fulfill the query condition. Since u stores one key for each of his grantors and u ignores which key to use to decrypt each ciphertext, the decryption process has a worst-case complexity of $\mathcal{O}(|Grantor_u|^2)$. Third, if u wants to revoke access from a user, u has to generate a new key $k_{u'}$, encrypt his name with $k_{u'}$, replace his encrypted name at the LBS provider with the new ciphertext and distribute $k_{u'}$ to all still authorized users.

In the following, we show how to use and combine existing cryptographic techniques to implement a scheme under our secrecy guarantees. We come up with two approaches. To ease the explanation of them, we first start by describing two

basic schemes, called basic two-layer symmetric encryption, *basic 2lSE*, and basic two-layer attribute-based encryption, *basic 2lABE*. Our basic approaches meet our secrecy guarantees under a weaker adversary model than the one defined in Section 6.1.2. With it, we weaken the protocol execution strategy by considering the stand-alone strategy instead of the twofold-composition strategy, i.e., the adversary executes the protocol only once. We then show how to extend the basic schemes to meet our secrecy guarantees under our actual adversary model, as defined in Section 6.1.2.

The main difference between our basic schemes lies in the cryptographic techniques used to encrypt the names which are sent as query answers. The basic schemes consist of four phases:

- *Initialization Phase:* In this phase, the key-authority generates and distributes keys. The initialization of the system happens once.
- *Registration Phase:* In this phase, new users register in the system.
- *Access Request Phase:* In this phase, a user u sends the access request $\langle u,\ read, p_v\rangle$ to user v.
- *Query Phase:* In this phase, users send range queries and get back query answers.

In Sections 6.2.1 and 6.2.2, we explain these phases for the *basic 2lSE* and the *basic 2lABE* schemes, respectively.

6.2.1 Basic Two-layer Symmetric Encryption (basic 2lSE)

Initialization Phase

The entities involved in this phase are the key-authority, the ACS, and the LBS provider. The key-authority generates three pairs of keys $(\mathsf{pk}_{LBS}, \mathsf{sk}_{LBS}), (\mathsf{pk}_{ACS}, \mathsf{sk}_{ACS})$, and $(\mathsf{pk}_H, \mathsf{sk}_H)$. The users and the service providers, ACS and LBS provider, use these keys during the registration, access request, and query phases. The key-authority sends the secret keys sk_H and sk_{ACS} to the ACS and the secret key sk_{LBS} to the LBS provider. It also chooses the integers p and g of the DH protocol, Definition 6.2, which our approach use to generate and share secret keys between a pair of users in the access request phase.

Registration phase

This phase involves four entities: the key-authority, the ACS, the LBS provider, and a user u. Figure 6.4 shows the steps of this phase.

First, the key-authority sends to u the following information: two identifiers, id_u^{ACS} and id_u^{LBS}, a secret key k_u, the public keys pk_{LBS} and pk_{ACS}, and the integers p and g. Second, u selects an integer number η_u and compute the value $Z_u \equiv g^{\eta_u} \pmod{p}$. We call η_u and Z_u the secret and public numbers of u, respectively. These two numbers are used as part of the DH protocol in the access request and query phases, as we will explain in the next two subsections. Next, u stores at the LBS provider his identifier id_u^{LBS}, his encrypted position $\mathsf{Enc}(\mathsf{pk}_H, p_u)$,[5] and his encrypted public number $\mathsf{Enc}(\mathsf{pk}_{ACS}, Z_u)$. The use of SHE allows the LBS provider to compute the encrypted square distance between the encrypted positions of two users. The LBS provider cannot decrypt any of the ciphertexts because it does not have the secret keys. Finally, u stores at the ACS his identifier id_u^{ACS} and two empty sets $Grantor_u$ and $Grantee_u$. Information is added to these two sets in the access request phase. We note that neither the LBS provider nor the ACS knows the link between users and their identifiers.

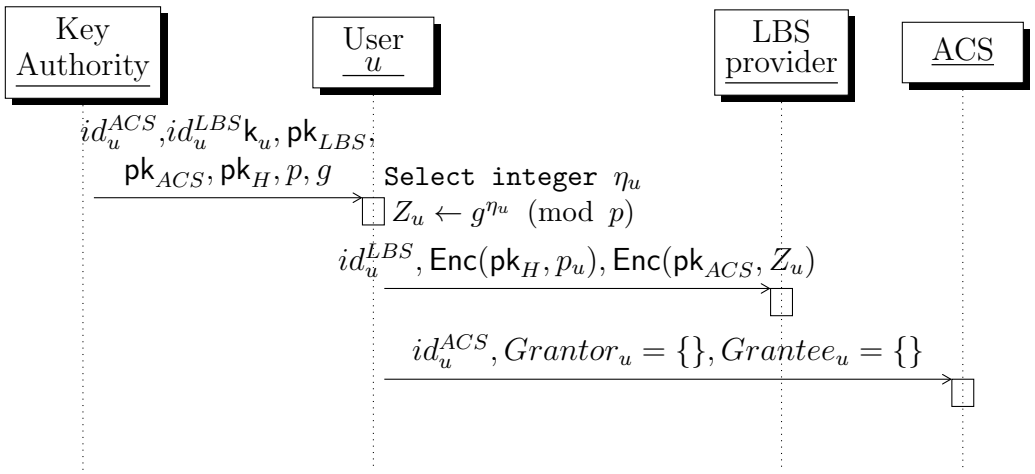


Figure 6.4: Registration Phase - *basic 2lSE*

Access Request Phase

In this phase, a user u sends the access request $\langle u,\ read, p_v \rangle$ to user v. If user v authorizes the access request, i.e., $accessReq(u, v) = true$, v stores encrypted information at the ACS and the LBS provider, as we will explain in this subsection.

[5] In reality, we encrypt a given position p_u as $\mathsf{Enc}(\mathsf{pk}_H, x_u)$ and $\mathsf{Enc}(\mathsf{pk}_H, y_u)$.

The providers use this information to process queries sent by u. Before explaining the steps of this phase, let us analyze how query processing works to understand the information that v has to store. Example 6.2 illustrates two design alternatives, *querying-filtering* and *filtering-querying*, to answer a given range query.

Example 6.2. Think of an LBS provider and an ACS. Assume that for each user u, the LBS provider stores his physical position after encrypting it with somewhat homomorphic encryption, and the ACS stores the set of grantors $Grantor_u$. Different design alternatives are conceivable to answer a given range query. In particular, one can consider the two designs alternatives discussed in Subsection 4.3.5: *querying-filtering* and *filtering-querying*. With the first alternative, the LBS provider executes, first, the query and then sends the result to the ACS to filter it based on the set $Grantor_u$. With the second alternative, the ACS sends, first, the set $Grantor_u$ to the LBS provider, and then the LBS provider executes the query using only the physical positions of users in $Grantor_u$. With both alternatives, since the LBS provider stores encrypted positions, it cannot use indexing, like B-tree or R-tree, for spatial query processing. Then in terms of performance, the *querying-filtering* alternative is not a suitable option because the LBS provider would need to compute the encrypted square distances between the encrypted position of u and the ones of all the users in the system.

Due to performance reasons, as explained in Example 6.2, we opt for the *filtering-querying* approach. Then the set of grantors of each user u has to contain information that allows the LBS provider to reduce the computation cost during query execution. Specifically, if v authorizes the access request, v has to add, among other information, his encrypted identifier id_v^{LBS} in the set $Grantor_u$, and to store his encrypted name at the LBS provider. The encrypted names of the users are sent as query answers, as we will explain in the query phase.

The access request phase involves the following entities: the users that are part of the access request, u and v, the LBS provider, and the ACS. Figure 6.5 illustrates the steps of this phase. We denote the concatenation of strings a and b by $a\|b$.

First, user u sends the $\langle u,\ read, p_v\rangle$ to user v together with his identifier id_u^{ACS} and his public number Z_u. If v authorizes the access request, v computes the shared key $\mathsf{k}_{uv} \equiv {Z_u}^{\eta_v} \pmod p$ and selects two random numbers $r_{u_v}^{ACS}, r_{u_v}^{LBS} \in \mathbb{Z}$. Next, v encrypts his name using two layers of encryption. v uses the shared key k_{uv} for the inner layer of encryption and the public key pk_{ACS} for the outer layer of encryption. v stores at the LBS provider the resulting ciphertext, $\mathsf{Enc}(\mathsf{pk}_{ACS}, \mathsf{Enc}(\mathsf{k}_{uv}, v))$, together with the random number $r_{u_v}^{LBS}$. The LBS provider cannot decrypt the

ciphertext, even if it colludes with any of the users, because none of them has the key to decrypt the outer layer of encryption. Secrecy proofs are in Section 6.2.4.

Next, v sends to the ACS the identifier of u, id_u^{ACS}, together with a tuple t which consists of two elements: the random number $r_{u_v}^{ACS}$ and the ciphertext $\mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS} \| r_{u_v}^{LBS})$ The ACS adds t to the set $Grantor_u$. The ACS uses the ciphertext that is part of tuple t during query processing, as we will explain in the query phase. Note that only v knows his identifier id_v^{LBS}, then he is the only one who can add his encrypted id to the set of grantors of other users. Finally, to revoke access, v stores in his set of grantees, $Grantee_v$, at the ACS, the ciphertext $c = \mathsf{Enc}(\mathsf{k}_v, u \| id_u^{ACS} \| r_{u_v}^{ACS} \| r_{u_v}^{LBS})$. c contains the name of u and index information that allows v to revoke access from u, i.e., v can delete both the tuple t added in the set $Grantor_u$ and the encrypted name stored at the LBS provider. Only v knows the key to decrypt c.

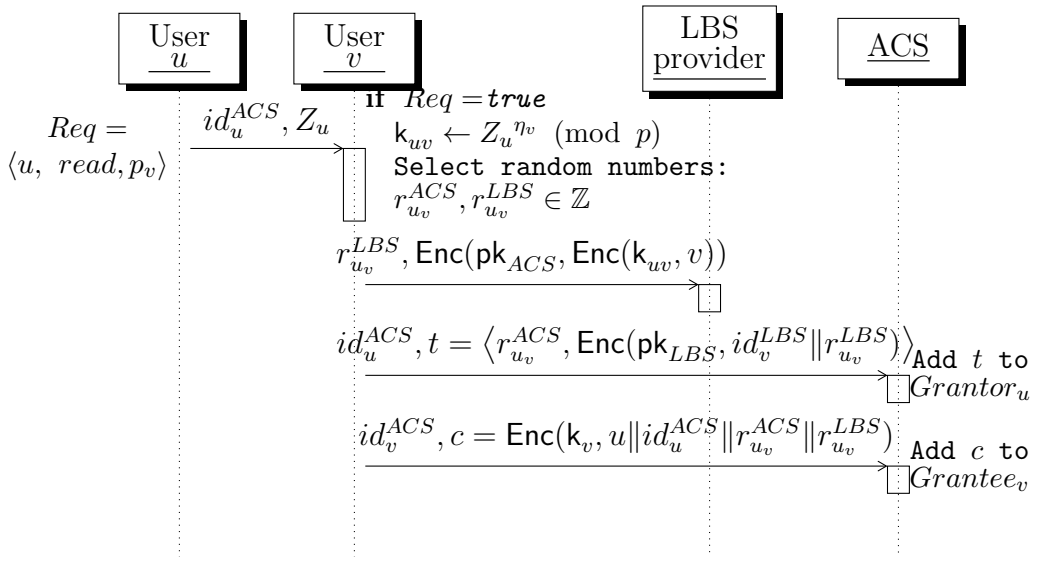


Figure 6.5: Access Request Phase - *basic 2lSE*

Query Phase

This phase involves three entities: the querying user u, the ACS, and the LBS provider. We use the following notation: Given n strings, $st_1, \cdots, st_n$, and the string $st = st_1 \| st_2 \| \cdots \| st_n$, the function $get(st, i)$, where $1 \leq i \leq n$, returns the i-th string in st. Given a tuple t which consist of n elements, we use t_i, where $1 \leq i \leq n$, to denote the i-th element of tuple t. Figure 6.6 illustrates the steps of this phase.

First, the querying user u sends his identifier id_u^{ACS}, the constraint d of the range query and his encrypted position $\mathsf{Enc}(\mathsf{pk}_{LBS}, (\mathsf{Enc}(\mathsf{pk}_H, p_u))$. u encrypts his position using two layers of encryption because the encrypted position is sent thought the ACS to the LBS provider, and the ACS knows the decryption key sk_H. The outer layer of encryption prevents the ACS from learning the position of u.

Second, using the id id_u^{ACS}, the ACS retrieves the set of grantors of u, $Grantor_u$. Recall that each tuple t in $Grantor_u$ consists of two elements, a random number, and a ciphertext. The ACS constructs a set C that contains the ciphertext of each tuple t in $Grantor_u$, i.e., t_2, and sends C and the encrypted position of u to the LBS provider.

Third, the LBS provider decrypts each ciphertext c in C and obtains the plaintext st, which consists of an identifier id concatenated with a random number $rand$. The LBS provider searches and retrieves the encrypted position and the encrypted public number corresponding to the identifier id. We use p_{id} and Z_{id} to denote the retrieved encrypted position and the encrypted public number, respectively. Using p_{id} and the encrypted position of u, the LBS provider computes the encrypted square distance between them, $\mathsf{Enc}(\mathsf{pk}_H, dist(p_{id}, p_u)^2)$. It also searches and retrieves the encrypted name, corresponding to the number $rand$. We denote this ciphertext by $name_{rand}$. Then, it creates a tuple e that contains three elements: $name_{rand}, Z_{id}$, and $\mathsf{Enc}(\mathsf{pk}_H, dist(p_{id}, p_u)^2)$. It adds e to the result set Res and sends Res to the ACS.

Next, for each tuple $e \in Res$, the ACS decrypts the element e_3, i.e., the encrypted square distance. If the decrypted distance is less or equal than d^2, the ACS (1) decrypts e_1 and e_2, i.e., the outer layer of the encrypted name and the encrypted public number, (2) creates a tuple l containing the decrypted information, and (3) adds l to the set of answers Ans. The ACS sends Ans to u.

Finally, for each tuple l in Ans, u uses l_2, i.e., the public number, and his secret number η_u to compute the shared key $\mathsf{k}_{shared} \equiv {l_2}^{\eta_u} \pmod{p}$, and decrypts l_1, i.e., the usernames , using k_{shared}. The decrypted names correspond to users that fulfill the query condition.

Next, we provide a working example to show how our *basic 2lSE* approach works step by step.

Example 6.3. Consider two users Alice and Bob. Suppose that Alice wants to query the set of users within a distance d. Assume further that Bob has allowed Alice to query the distance between their physical positions. Figure 6.7 illustrates the query processing steps (numbers within circles). It

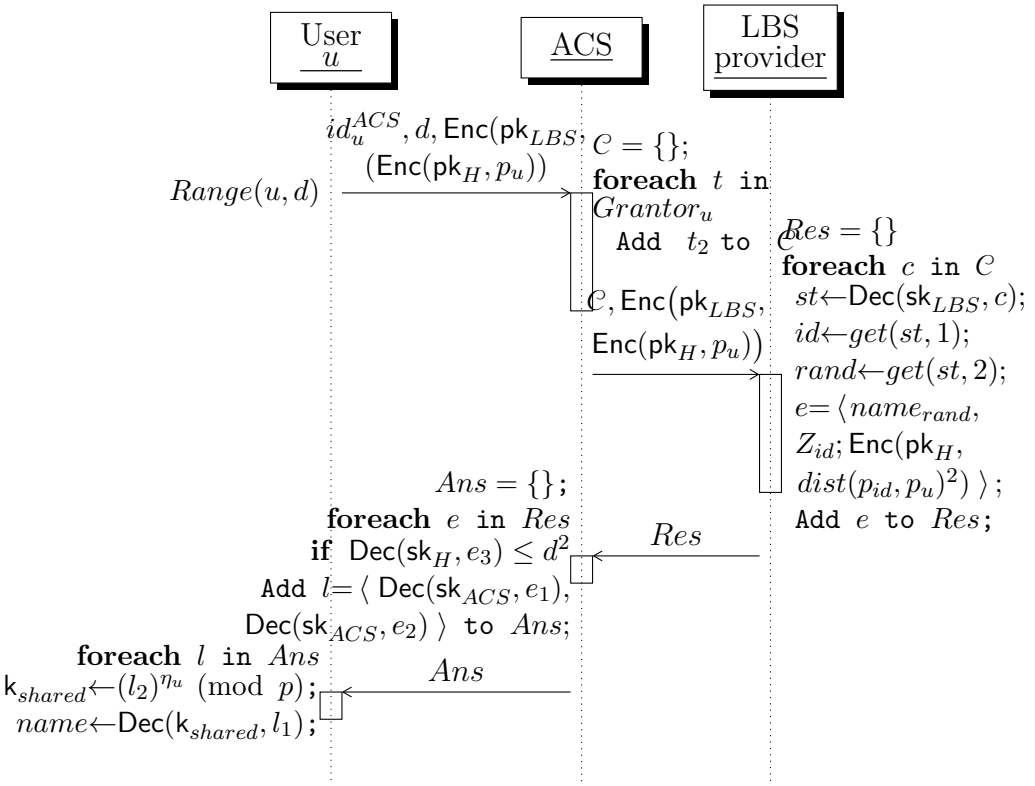


Figure 6.6: Query Phase - *basic 2lSE*

also contains the information which each entity in the system stores. The data stored in the tables at the LBS provider and ACS is information about Alice and Bob.

6.2.2 Basic Two-layer Attribute-based Encryption (basic 2lABE)

before describing the *basic 2lABE* approach, we specify the differences between our two approaches to facilitate the understanding. With the *basic 2lSE*, the LBS provider stores, for each user u, $|Grantee_u|$ copies of the encrypted name of u. Each ciphertext is generated using a shared key between u and each v in $Grantee_u$. Users use the DH protocol to generate shared keys between them. In contrast, with the *basic 2lABE* there is no need to store multiple copies of the encrypted name or to share keys. Instead, each user encrypts his name using CP-ABE, Definition 3.4. Because of the properties of CP-ABE, each user receives a secret key based on his set of attributes, and only users who fulfill the access policy

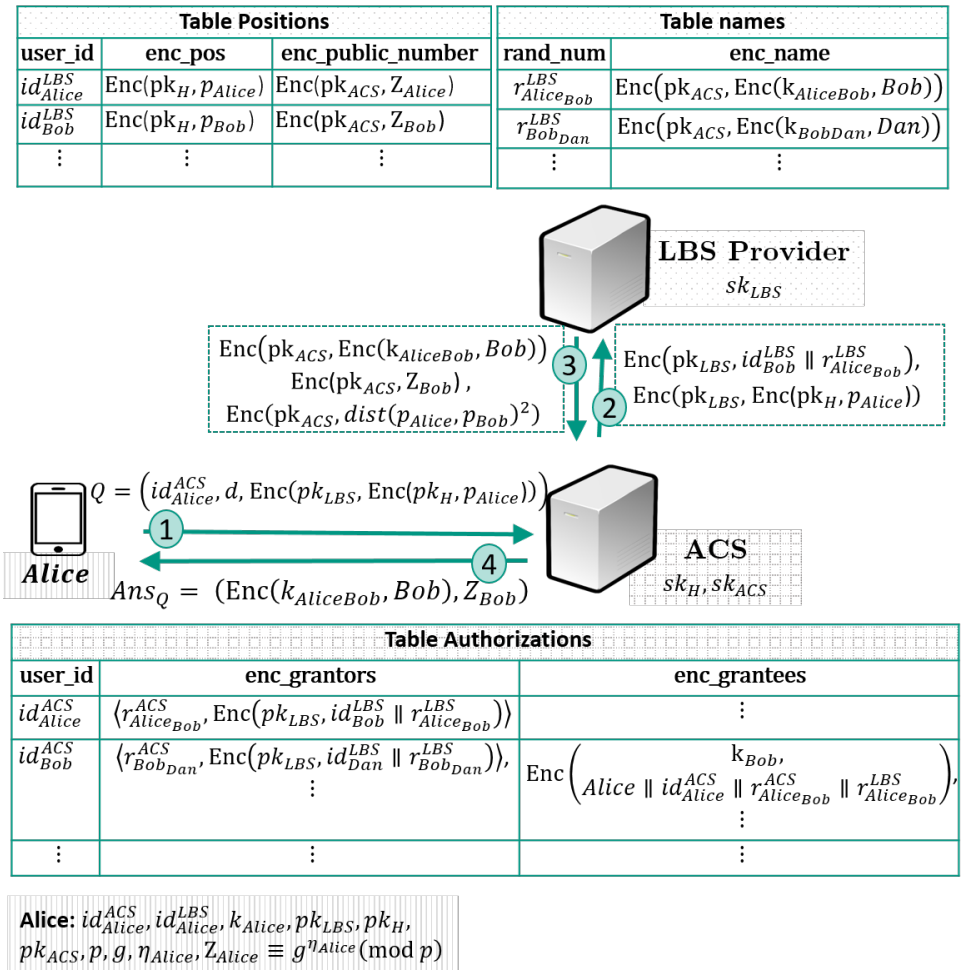


Figure 6.7: *2lSE* Approach - Example

can decrypt a given ciphertext. In Sections 6.2.2 and 6.2.2, we specify the access policy and the set of attributes used for the encryption and decryption process.

Next, we describe the initialization, registration, access request, and query phases of the *basic 2lABE*. For brevity, we only describe the differences between the *basic 2lABE* and *basic 2lSE* schemes.

Initialization Phase

This phase differs from that of the *basic 2lSE*, Section 6.2.1, only in the selection of the parameters p and g of the DH protocol. Since the *basic 2lABE* uses CP-ABE, it does not need the DH protocol for generating and sharing keys. The

key-authority generates a public key pk_{ABE} and a master key $\mathtt{mk}_{ABE}$. It uses these keys during the registration phase (Section 6.2.2). The key-authority keeps $\mathtt{mk}_{ABE}$ secret from all entities.

Registration Phase

Before explaining the steps of this phase, let us analyze how CP-ABE works in our scenario. The key-authority has to generate a secret key for each user u using attributes associated with u. Since the access policy used in this work considers the names of the users to establish who can decrypt a given ciphertext, the set of attributes associated with the users is $Attr = \{name\}$. If one needs to use other attributes, one has to include them in the set of attributes associated with the users. Given a user u, the access policy defined by u, γ_u, is a disjunction of terms of the form (*name=value*), where *name* is a user attribute, and *value* refers to an atomic value.

Example 6.4. Consider a user u, the set $Grantee_u = \{v, w\}$, and a message m. User u wants to encrypt m and allow the users in $Grantee_u$ to decrypt it. u can use the usernames to identify the users in $Grantee_u$, and to generate the access policy γ_u needed to encrypt m. u can specify the access policy $\gamma_u = (name = v) \vee (name = w)$ and generate the ciphertext $c = \mathsf{Enc}(\mathsf{pk}_{ABE}, m_{\gamma_u})$. γ_u indicates that c can be decrypted by users with name v or w. The secret keys of users v and w are generated based on their attributes, e.g., the secret key of v, sk_{ω_v}, is generated based on the set of attributes $\omega_v = \{name = v\}$. Only users whose secret key fulfill the access policy γ_u can decrypt c.

This phase differs from that of the *basic 2lSE*, Section 6.2.1, in the following: First, the key-authority uses the master key $\mathtt{mk}_{ABE}$ and the set of attributes of the registering user u, $\omega_u = \{name = u\}$ to generate the secret key sk_{ω_u}. Second, the key-authority, instead of sending to u the parameters p and g, sends the keys pk_{ABE} and sk_{ω_u}. Third, u does not follow the DH protocol. Instead, u stores at the LBS provider, apart from his identifier id_u^{LBS} and his encrypted position, his encrypted name $\mathsf{Enc}(\mathsf{pk}_{ACS}, \mathsf{Enc}(\mathsf{pk}_{ABE}, u_{\gamma_u}))$, with access policy $\gamma_u = \big((name = u)\big)$. Note that, since u has not authorized any access request, yet, γ_u specifies that only u can decrypt his encrypted name.

Access Request Phase

This phase differs from that of the *basic 2lSE*, Section 6.2.1, in the following: First, since the *basic 2lABE* does not use the DH protocol, users u and v do not compute the shared key k_{uv}. Instead, v has to update his access policy γ_v by adding to the disjunction the term "$(name = u)$". Second, v does not generate the random number $r_{u_v}^{LBS}$. That is because, the LBS provider stores, together with the identifier id_v^{LBS}, a single ciphertext containing the encrypted name of v, which can be decrypted by all his grantors. Then, v selects only one random number, $r_{u_v}^{ACS} \in \mathbb{Z}$, instead of two. As a consequence, the tuple t added to the set $Grantor_u$ is $t = \langle r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS}) \rangle$, instead of $t = \langle r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS} \| r_{u_v}^{LBS}) \rangle$, and the ciphertext added to the set $Grantee_v$ is $c = \mathsf{Enc}(\mathsf{k}_v, u \| id_u^{ACS} \| r_{u_v}^{ACS})$, instead of $c = \mathsf{Enc}(\mathsf{k}_v, u \| id_u^{ACS} \| r_{u_v}^{ACS} \| r_{u_v}^{LBS})$. Third, v encrypts and updates at the LBS provider his name using, for the inner layer of encryption, the public key pk_{ABE}, and the access policy γ_v, instead of using the shared key k_{uv}.

Query Phase

This phase differs from that of the *basic 2lSE*, Section 6.2.1, in the following: First, the set of ciphertexts C that the LBS provider receives from the ACS contains only encrypted identifiers, instead of encrypted identifiers concatenated with random numbers. Second, the LBS provider neither retrieves public numbers nor uses the decrypted random numbers to retrieve encrypted names. Instead, it only uses the decrypted identifiers to retrieve the encrypted positions and encrypted names. Third, the set of tuples that the ACS receives from the LBS provider does not contain encrypted public numbers. Fourth, the querying user u does not need to compute shared keys to decrypt the ciphertexts received as query answer. Instead, u uses his secret key sk_{ω_u}.

6.2.3 Extending the Basic Schemes

Having the basic schemes, *basic 2lSE* and *basic 2lABE*, we now show how to extend them to meet our secrecy guarantees $G_{position}$, $G_{distance}$ and $G_{authorization}$ under our adversary model defined in Section 6.1.2. That is, we consider the twofold-composition strategy in which the protocol is executed twice. The only problem of the basic schemes under this strategy relies on the revocation process. With both approaches, a revoked user, in case of collusion with the LBS provider, could gain access to unauthorized information, as explained in Example 6.5. In the case of collusion with the ACS, there is no information leakage, as we prove in Section 6.2.4.

Example 6.5. Consider Example 6.3. Assume that Alice colludes with the LBS provider and sends a range query. The ACS sends to the LBS provider, during query processing, a message containing, among others, the identifier id_{Bob}^{LBS}. If the system does not receive other queries while it computes the query of Alice, the LBS provider and Alice can learn that id_{Bob}^{LBS} is the identifier of Bob. Assume now that Bob revokes access to Alice. Alice can regain this access by following herself the steps of the access request phase. Specifically, with the *basic 2lSE*, Alice can follow all the steps of the access request phase that Bob, i.e., the grantor, should execute. With the *basic 2lABE*, Alice can follow the steps of the access request phase and store information at the ACS. But she cannot update the ciphertext c containing the name of Bob, stored at the LBS provider, because she does not know the access policy γ_{Bob} that Bob used to generate c. Nevertheless, during querying processing, with the information added at the ACS, Alice will receive the encrypted name of Bob, which she cannot decrypt, but she knows it corresponds to Bob.

Extended Two-layer Symmetric Encryption (2lSE)

If a revoked user colludes with the LBS provider, we need the ACS to detect the attack and raise an alarm. To do so, we extend the *basic 2lSE* as follows: First, in the initialization phase, in addition to the steps of the *basic 2lSE*, the key-authority generates a pair of keys $(\mathsf{pk}_{ACS'}, \mathsf{sk}_{ACS'})$, and sends $\mathsf{sk}_{ACS'}$ to the ACS. The key-authority keeps for itself the key $\mathsf{pk}_{ACS'}$. Second, in the registration phase, the key-authority sends to each registering user v, in addition to the information sent with the *basic 2lSE*, a ciphertext containing his identifier encrypted using two layers of encryption. The key for the inner layer is pk_{LBS} and the one for the outer layer is $\mathsf{pk}_{ACS'}$. That is, each user v receives, additionally, the ciphertext $\mathsf{Enc}(\mathsf{pk}_{ACS'}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS}))$.

Third, in the access request phase, the tuple t that the grantor v sends to the ACS changes from $t = \langle r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS} \| r_{u_v}^{LBS}) \rangle$ to $t = \langle\, r_{u_v}^{ACS}, \mathsf{Enc}(\mathsf{pk}_{ACS'}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS})), \mathsf{Enc}(\mathsf{pk}_{LBS}, r_{u_v}^{LBS}) \,\rangle$. Next, the ACS uses the key $\mathsf{sk}_{ACS'}$ to decrypt the element t_2, replaces t_2 with the decrypted information and adds t to the set $Grantor_u$. Since v is the only user who knows the ciphertext $\mathsf{Enc}(\mathsf{pk}_{ACS'}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS}))$, the ACS knows that the access request has been registered by the actual grantor and not by a revoked user who has colluded with the LBS provider. In other words, even if a user u learns the identifier of one of his grantors v, in case of revocation, u cannot regain access because only v knows his cipher-

text $c = \mathsf{Enc}(\mathsf{pk}_{ACS'}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS}))$ and only the key-authority knows the key $\mathsf{pk}_{ACS'}$ to generate c.

We assume that the ACS uses a decryption algorithm Dec that indicates successful decryption. That is, the decryption $\mathsf{Dec}(\mathsf{sk}_{ACS'}, \mathsf{Enc}(\mathsf{k}', c))$, where c is a ciphertext, is called successful if $\mathsf{sk}_{ACS'} = \mathsf{k}'$. One can implement a successful decryption algorithm by concatenating the hash value $\mathsf{H}(c)$ to the encryption $\mathsf{Enc}(\mathsf{k}', c \| \mathsf{H}(c))$ and checking this relation in the decryption algorithm [HLK19].

Fourth, in the query phase, the set of ciphertexts C that the ACS sends to the LBS provider is different from that of the *basic 2lSE*. Here, C is a set of tuples, where each tuple contains two ciphertexts. Specifically, for each tuple t in the set $Grantor_u$, the ACS adds to C a tuple consisting of the elements t_2 and t_3, i.e., the encrypted identifier and the encrypted random number. The LBS provider decrypts both ciphertexts and processes the query as with the *basic 2lSE*.

Figures 6.8, 6.9, and 6.10 illustrate the steps of the registration, access request and query phases, respectively. The changes compared to the basic scheme are highlighted in red.

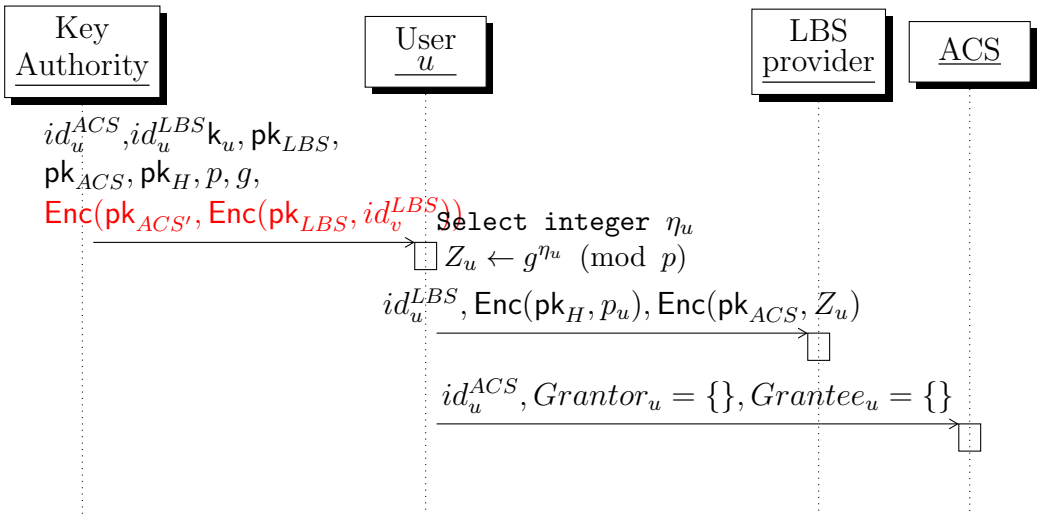


Figure 6.8: Registration Phase - *basic 2lSE*

Extended Two-layer Attribute-based Encryption (2lABE)

To solve the existing problems when a revoked user colludes with the LBS provider, we use the same technique as that of the *2lSE*, presented in the previous subsection. That is, the ACS, during the access request phase, receives the following ciphertext: $\mathsf{Enc}(\mathsf{pk}_{ACS'}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_v^{LBS}))$, and verifies that the actual grantor has sent the

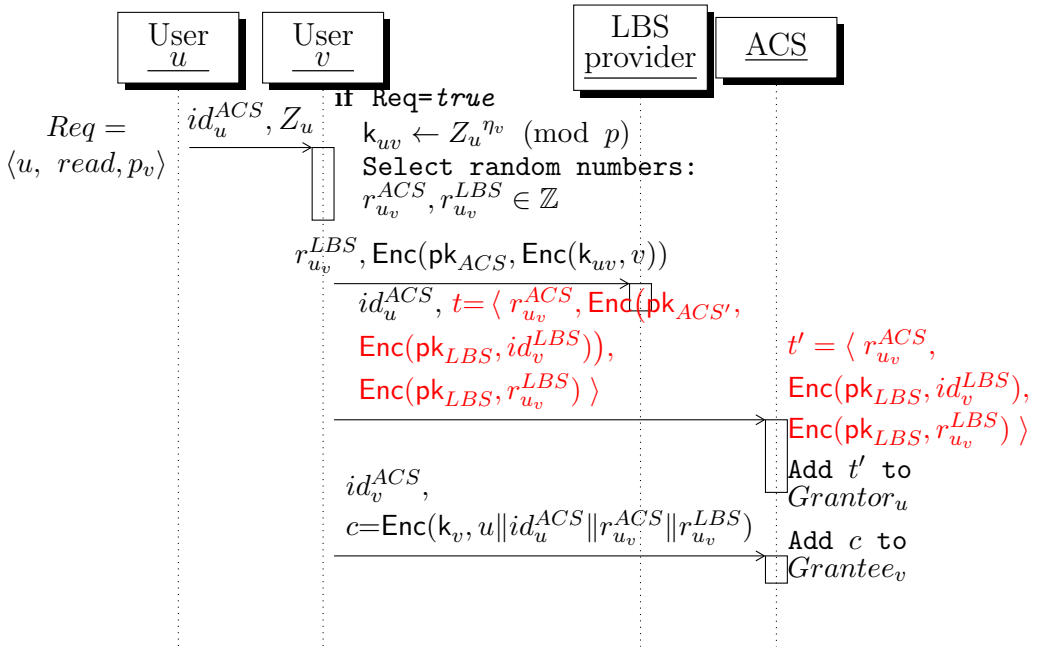


Figure 6.9: Access Request Phase - *basic 2lSE*

received ciphertext. The query processing phase is similar to that of the *basic 2lABE* approach.

6.2.4 Secrecy Proofs

This subsection provides the secrecy analysis of our proposed approaches. The proofs are organized as follows: For each approach, we first prove that, under our adversary model defined in Section 6.1.2, the approach provides the secrecy guarantee $G_{position}$, then $G_{distance}$, and finally $G_{authorization}$.

For each proof, we first consider the case that a user colludes with the LBS provider and then the case that a user colludes with the ACS, during the first execution of the protocol. During the second execution of the protocol, the user is not allowed to collude anymore (twofold-composition strategy). The case where each party individually acts as an adversary is straightforward and therefore omitted. Given two entities A and B, we study the case where entity A colludes with B, and omit the case where B colludes with A. In the case of collusion of two entities A and B, we assume that either A or B has access to all information and functionality of A and B together.

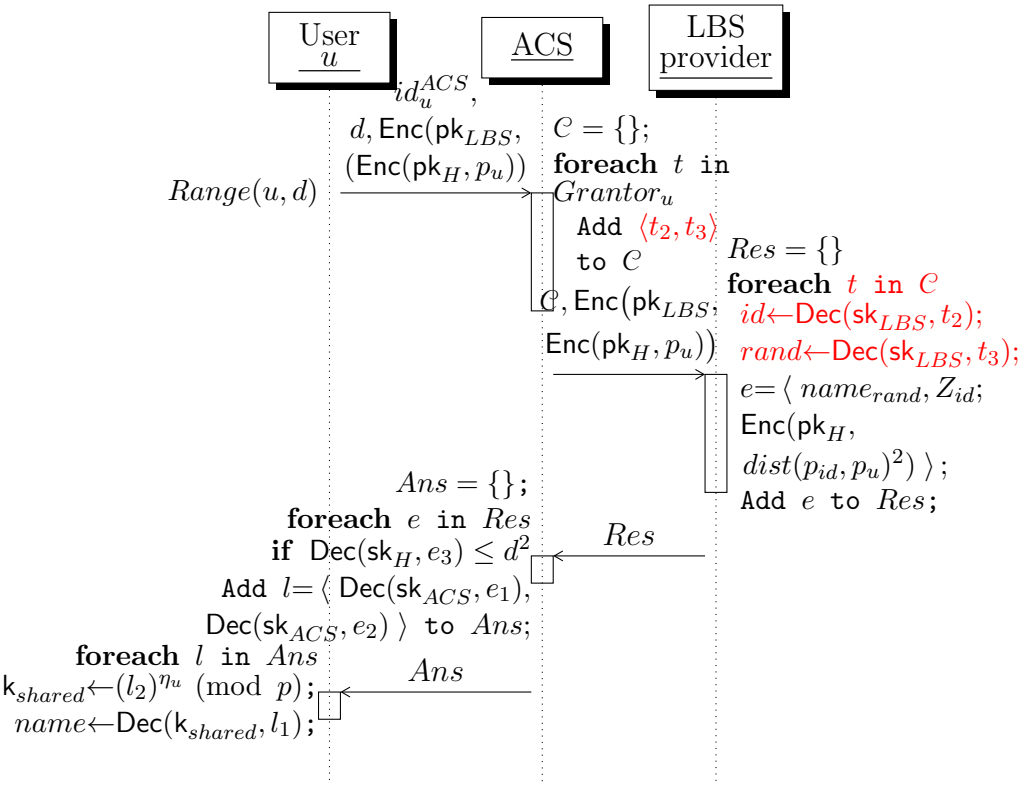


Figure 6.10: Query Phase - *basic 2lSE*

Before starting with the proofs, let us recall the information that the users, LBS provider, and ACS have. First, each user s knows: his identifiers id_s^{LBS}, id_s^{ACS}, the keys k_s, pk_H and pk_{ACS}, his encrypted identifier $\mathsf{Enc}(\mathsf{pk}_{ACS'}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_s^{LBS}))$, and the parameters p, g, η_s. Second, the LBS provider stores: the user identifiers, the encrypted positions, the encrypted public numbers, the encrypted names together with the random numbers, and the secret key sk_{LBS}. Additionally, the LBS provider knows the relations existing between the information that it stores, e.g., the encrypted position that corresponds to a given identifier. Third, the ACS stores: the encrypted sets of grantors and grantees of all the users, and the keys sk_H, sk_{ACS} and $\mathsf{sk}_{ACS'}$. Additionally, the ACS knows the relations existing between the stored information.

2lSE - Secrecy Proofs

Lemma 6.1. *Given a user u, the 2lSE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 6.1.2, only authorized entities can learn the physical position of u.*

Proof. Case 1: A user s colludes with the LBS provider. During the first protocol execution, s has access to the information that he owns and the one stored at the LBS provider. However, none of the information that s has can be used to decrypt the encrypted positions. Only the ACS knows the decryption key sk_H. According to our adversary model, s cannot collude with both entities at the same time. Furthermore, somewhat homomorphic encryption, which is used to encrypt the physical positions of the users, is IND-CPA secure [LMSV11].

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn the physical positions of users who have not authorized him to do so. During query processing, the LBS provider receives from the ACS a set containing the encrypted identifiers of the grantors of s. So, s knows that the set of identifiers used by the LBS provider to process his query corresponds to his grantors. Since the identifiers are random numbers, s cannot link the identifiers with their owners, except if $|Grantor_s| = 1$. In this last case, s can learn the identifier of his unique grantor, namely u, id_u^{LBS}, see Example 6.5. However, id_u^{LBS} does not leak any information about the position of u or cannot be used to decrypt his encrypted position. Next, during query processing, s receives a set of encrypted names and public numbers corresponding to the grantors of s that fulfill the query condition. Since the information received belongs to the grantors of s, s is allowed to learn such information. Next, assume that the LBS provider deviates from the query processing protocol, as part of the collusion. The LBS provider processes the query with different identifiers from the ones sent by the ACS. So, instead of using the information that belongs to the grantors of s, it selects different identifiers and random numbers to give s information that he is not authorized to access. However, the identifiers are random numbers, and neither s nor the LBS provider knows the relationship between identifier and users. Next, if the LBS provider processes the query with identifiers selected at random, s receives as query result a set of encrypted names and public numbers. The names are encrypted using probabilistic symmetric encryption, which is IND-CPA secure [KL07], and s does not know the key to decrypt them. Next, by using the received public numbers, and the parameters p and g, s cannot compute the decryption key because of the security offered by the DH problem, which has been proven to be computationally infeasible in [Bon98]. Then, s cannot learn, in the first protocol execution, the physical position of users that he is not authorized.

During the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, information that he has learned during the first protocol execution, and the one that he will obtain during the second protocol execution. However, during the second protocol

execution, s does not learn any new information. Then, as in the first execution, this information is not enough to decrypt and learn the physical positions of other users.

Case 2: A user s colludes with the ACS. During the first protocol execution, s has access to the information that he owns and the one stored at the ACS. Although s knows the key sk_H to decrypt the encrypted positions, s does not have the ciphertext to decrypt them because they are stored at the LBS provider. Since, according to our adversary model, s cannot collude with both entities at the same time, s cannot use the key sk_H to decrypt the physical positions of other users.

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn the physical positions of other users.

By following the query processing protocol, s receives a set of encrypted names and public numbers corresponding to the grantors of s that fulfill the query condition. Since the information received belongs to the grantors of s, s is allowed to learn that information. Next, assume that the ACS deviates from the query processing protocol, as part of the collusion. That is, the ACS includes in the set $Grantor_s$, information from the other sets of grantors to give access to s to information that he is not authorized to access. s receives as query answer, a set of encrypted names and public numbers. Based on the same arguments as the ones presented in the Case 1, s cannot compute the decryption key and cannot learn any information from the encrypted names. Then, during the first protocol execution, s cannot learn the physical position of users who have not authorized him.

During the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, information that he has learned during the first execution, and the one that he will obtain during the second protocol execution. Different from the case 1, after query processing, s cannot learn any extra information, i.e., s has access to the same information as in the first protocol execution. Using the same arguments as in the first execution, $s$cannot learn the physical positions of users that he is not authorized..

Consequently, given a user u, the *2lSE* approach guarantees that, in the presence of adversaries with the power defined in our adversary model, only authorized entities can learn the physical position of u. □

Lemma 6.2. *Given a user u, the 2lSE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 6.1.2, only entities they themselves have authorized can learn the distance between their physical position and the one of u.*

Proof. Case 1: A user s colludes with the LBS provider. During the first protocol execution, s has access to the information that he owns and the one stored at the LBS provider. Using this data, as proved in Lemma 6.1, s cannot learn any useful information from the encrypted data.

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn the distance between his position and the one of the users who have not authorized him to do so. As explained in the proof of Lemma 6.1, during query processing, if $|Grantor_s| = 1$, s can learn the identifier of his unique grantor, namely u, id_u^{LBS}. The identifier id_u^{LBS} is a random number and it does not leak any information about the physical positions or the distance between users. Additionally, as proved in Lemma 6.1, by following the query processing protocol, s receives as query answer only information that he is authorized to access.

Next, assume that the LBS provider deviates from the query processing protocol, as part of the collusion. That is, instead of using the information that belongs to the grantors of s, the LBS provider selects different identifiers and random numbers to give s information that he is not authorized to access. As proved in Lemma 6.1, since s cannot compute the encryption key and probabilistic encryption is IND-CPA secure [KL07], s cannot learn any useful information from the encrypted data received during query processing. Then, s cannot learn, in the first protocol execution, the distance between his position and the one of users that he is not authorized.

Next, during the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, information that he has learned during the first protocol execution, and the one that he will obtain during the second protocol execution. Assume that in the second protocol execution, u revokes access to s. Then s can try himself to add id_u^{LBS} to his set $Grantor_s$ following the steps of the access request phase. However, s needs to send the ciphertext $c = \mathsf{Enc}(\mathsf{pk}_{ACS'}, \mathsf{Enc}(\mathsf{pk}_{LBS}, id_u^{LBS}))$ to the ACS. Since s does not know c or the key $\mathsf{sk}_{ACS'}$ to generate c, s cannot add id_u^{LBS} in his set $Grantor_s$. Therefore, in the case of revocation, s cannot regain access.

Case 2: A user s colludes with the ACS. During the first protocol execution, s has access to the information that he owns and the one stored at the ACS. Knowing all this information, as proved in Lemma 6.1, s cannot obtain the ciphertexts corresponding to the encrypted positions of other users. So s cannot learn any useful information in this step.

In addition to the information that s has learned, s can execute queries and get their respective answers.

As proved in Lemma 6.1, by following the query processing protocol, s receives as query answer only information that he is authorized to access.

Next, assume that the ACS deviates from the query processing protocol, as part of the collusion. That is, the ACS includes in the set $Grantor_s$, information from the other sets of grantors to give access to s to information that he is not authorized to access. As proved in Lemma 6.1, and similar to Case 1, since s cannot compute the encryption key and probabilistic encryption is IND-CPA secure [KL07], s cannot learn any useful information from the encrypted data received during query processing. Then, s cannot learn, in the first protocol execution, the distance between his position and the one of users that he is not authorized.

During the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, information that he has learned during the first execution, and the one that he will obtain during the second protocol execution. Different from Case 1, after query processing, s cannot learn any extra information. Using the same arguments as in the first execution, s cannot learn the distance between his position and the one of the users who have not authorized him. So, in the case of revocation, by using the information that s knows, s cannot regain access.

Consequently, given a user u, the *2lSE* approach guarantees that, in the presence of adversaries with the power defined in our adversary model, only entities they themselves have authorized can learn the distance between their physical position and the one of u. □

We note that although users do not receive as part of the query answers the physical position of authorized users, they could submit fake positions and check the results of the query to infer the real position of a target. However, as proved in Lemma 6.2, if a user u submits a query, the query result contains only users who have authorized u to learn the distance between their physical position and the one of u. That is, only authorized users could infer the real position of a target. However, these users are authorized to access such information according to our secrecy guarantees.

Lemma 6.3. *Given two users u and v, the 2lSE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 6.1.2, an adversary will not be able to learn whether u or v are allowed to learn the distance between them.*

Proof. Case 1: A user s colludes with the LBS provider. During the first protocol execution, s has access to the information that he owns and the one stored at the LBS provider. First, the user identifiers are random numbers, and by using all the

known information, neither the LBS provider nor s can determine the link between users and identifiers.

Second, the information that s knows does not contain any link between grantees and grantors. Then, s cannot determine whether two given users u and v are allowed to learn the distance between them.

In addition to the information that s has learned, s can execute queries and get their respective answers. We now show that s cannot use this information to learn whether two given users u and v are allowed to learn the distance between them. As explained in the proof of Lemma 6.1, during query processing, if $|Grantor_s| = 1$, s can learn the identifier of his unique grantor, namely u, id_u^{LBS}. However, given a set of queries, neither s nor the LBS provider knows who are the querying users. Therefore, if the identifier id_u^{LBS} is used during query processing, s cannot determine who are the grantees of u.

Additionally, during query processing, s receives a set of encrypted names and public numbers corresponding to the grantors of s that fulfill the query condition. However, the received information does not reveal any data about the grantees and grantors of other users. Then, given two users u and v, during the first protocol execution, s is not able to learn whether u or v are allowed to learn the distance between them.

During the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, the one that he has learned during the first execution, and the one that he will obtain during the second protocol execution. Since the information that s gets in the second protocol execution is similar to the one of the first execution, this information is not enough to learn whether u or v are allowed to learn the distance between them.

Case 2: A user s colludes with the ACS. During the first protocol execution, s has access to the information that he owns and the one stored at the ACS. First, the set of grantors is encrypted using asymmetric encryption and the key pk_{LBS}. Since s does not know pk_{LBS} and asymmetric encryption is IND-CPA secure [KL07], s cannot learn any useful information from the encrypted data.

Second, the set of grantees are encrypted using probabilistic encryption and the key of its owner. Since s knows only the key to decrypt his set of grantees, and probabilistic encryption is IND-CPA secure [KL07], s cannot learn any useful information from the encrypted data.

In addition to the information that s has learned, s can execute queries and get their respective answers. Similar to the case 1, the information that s received

during query processing does not reveal any data about the grantees and grantors of other users.

Next, during the second protocol execution, since s is not allowed to collude anymore with any entity, s only has access to information that he owns, the one that he has learned during the first protocol execution, and the one that he will obtain during the second execution. Similar to the case 1, s has access to the same information as in the first protocol execution. Therefore, using the same arguments as in the first protocol execution, s cannot learn whether u or v are allowed to learn the distance between them.

Consequently, given two users u and v, the *2lSE* approach guarantees that, in the presence of adversaries with the power defined in our adversary model, an adversary will not be able to learn whether u or v are allowed to learn the distance between them. □

2lABE - Secrecy Proofs

Lemma 6.4. *Given a user u, the 2lABE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 6.1.2, only authorized entities can learn the physical position of u.*

Lemma 6.5. *Given a user u, the 2lABE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 6.1.2, only entities they themselves have authorized can learn the distance between their physical position and the one of u.*

Lemma 6.6. *Given two users u and v, the 2lABE approach guarantees that, in the presence of adversaries with the characteristics defined in our adversary model, Section 6.1.2, an adversary will not be able to learn whether u or v are allowed to learn the distance between them.*

The proofs of Lemmas 6.4 - 6.6 are analogous to the proofs of Lemmas 6.1 - 6.3, respectively. Although the *2lSE* and the *2lABE* approaches have some differences, the properties of the *2lSE* used for the proofs also hold for *2lABE*. First, both approaches differ in the encryption scheme used to encrypt the usernames. The *2lABE* uses CP-ABE. Since CP-ABE is IND-CPA secure [BSW07], adversaries cannot learn any useful information from the encrypted names. Second, different from the *2lSE*, the *2lABE* does not use the DH protocol to share keys between users. Instead, each user receives, as part of the CP-ABE, a secret key for decryption. A user can decrypt a ciphertext c only if the attributes used to generate his secret key satisfies the access policy used to generate c. That is, a user can decrypt only usernames that belong to his grantors.

6.3 Time Complexity Analysis

A complexity analysis is helpful to predict the behavior of the *2lSE* and *2lABE* approaches and to facilitate meaningful comparisons. As explained in Section 5.3 an average complexity analysis depends on the internal behavior of the database, which is specific to the product and is not openly available. Furthermore, if there are changes in the system settings, the average analysis is void. So our complexity analysis is a worst case analysis. Here, we focus on the complexity of the query phase because this is the most frequently used phase in our scenario.

6.3.1 2lSE - Time Complexity Analysis

The number of operations at the user-side depends on the number of grantors of the querying user u that are inside the query range. Then, our worst-case complexity analysis considers that all the users in the set of grantors of u are located within the query range. To perform the complexity analysis of our approaches, one needs to specify the complexity of the encryption/decryption process, which depends on the type of encryption/decryption algorithm used. We select the following well-known algorithms from the literature: For symmetric and asymmetric encryption schemes, we use AES and RSA, respectively. For somewhat homomorphic encryption, we use algorithms based on the learning with errors problem over rings, such as the ones presented in [LPR10]. For CP-ABE, we select the algorithm presented in [LGR+12].

Let A be the total number of authorized access requests, i.e., $A = \sum_{u \in U} |Grantor_u|$, $B(p_u)$ be the bit string representation of the physical position p_u, and $|B(p_u)|$ its length. Further, let TC_{user}, TC_{ACS}, and TC_{LBS} denote the time complexity at the user-side, the ACS and the LBS provider, respectively. A ciphertext generated using somewhat homomorphic encryption is represented as a matrix M. We use $|M|$ to denote the size of the matrix. Next, the initialization of the RSA algorithm requires to select at random two large primes. We use N to denote the product of the selected primes. Next, note that p and η_u are the integer and the secret number of a given user u, respectively, which are part of the DH protocol.

Lemma 6.7. *Given a range query Range $= (u, d)$, the time complexities of the 2lSE approach at the user-side, the ACS, and the LBS provider are:*

$$\begin{aligned} TC_{user} &= \mathcal{O}\big(|Grantor_u| \cdot \log(p)^2 \cdot \log(\eta_u)\big) \\ TC_{ACS} &= \mathcal{O}\big(|Grantor_u| \cdot (|B(p_u)| \cdot \log(|B(p_u)|) + \log(N)^3)\big) \\ TC_{LBS} &= \mathcal{O}\big(|Grantor_u| \cdot |M|^3\big) \end{aligned} \tag{6.1}$$

Proof. We start by analyzing the complexity of the encryption/decryption process of each of the encryption schemes used in the *2lSE* approach, i.e., symmetric, asymmetric encryption, and SHE. First, the complexity of symmetric encryption schemes, specifically AES, depends on the length of the message m to be encrypted, $|m|$, [OEHB11]. Then, the complexity of the encryption/decryption process is $\mathcal{O}(|m|)$. In our scenario, we use symmetric encryption to encrypt the usernames. We consider that the length of the string that represents a username has a constant size of 12 bytes. Therefore, the complexity of the encryption/decryption process using symmetric encryption reduces to $\mathcal{O}(1)$.

Second, the complexity of asymmetric encryption schemes, specifically RSA, is based on the complexity of modular exponentiation. Given a message m, the resulting ciphertext using RSA is $c = m^e \pmod N$, and the decryption of c is $m = c^d \pmod N$, where e is the public key and d is the secret key. Given the integer numbers B, C, N, the complexity of the modular exponentiation $B^C \pmod N$ is $\mathcal{O}(\log(N)^2 \cdot \log(C))$. We consider that the exponent e in the RSA algorithm, i.e., the public key, is fixed, as specified in FIPS-186-3 [LG09]. So the encryption complexity using RSA is $\mathcal{O}(\log(N)^2)$. Using standard RSA assumptions, the exponent d in the RSA algorithm, i.e., the secret key, has size in bits close to that of N. Then, the decryption complexity using RSA is $\mathcal{O}(\log(N)^3)$.

Third, using somewhat homomorphic encryption, SHE, schemes [LPR10], the encryption/decryption process depends on modular multiplication and addition of vectors. Given a message m, the complexity of encrypting/decrypting m using SHE is $\log(|B(m)|)$ per bit [CZZ16], assuming the use of the Montgomery multiplication, which is one of the fastest methods available for performing modular multiplication. Then, the encryption/decryption complexity of SHE schemes is $\mathcal{O}(|B(m)| \cdot \log(|B(m)|))$. The following steps are required to compute a given range query with the *2lSE* approach.

1. Encrypt using SHE the position of the querying user u. The user executes this step. The complexity of this step is $\mathcal{O}(|B(p_u)| \cdot \log(|B(p_u)|))$.

2. Use the identifier of the querying user u, id_u^{ACS} to retrieve the set of grantors $Grantor_u$. The ACS executes this step. We assume the ACS uses B-tree indexing. Then, the complexity of this step is $\mathcal{O}(\log(|U|))$.

3. Decrypt the set of encrypted identifiers and the set of encrypted random numbers sent by the ACS. Since both sets have a size of $|Grantor_u|$ and each element of these sets is encrypted using the asymmetric encryption, the LBS provider has to execute $2 \cdot |Grantor_u|$ asymmetric decryptions. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot \log(N)^3)$.

4. For each decrypted identifier in step 3, retrieve the corresponding encrypted position and encrypted public value. The LBS provider does this step. We assume the LBS provider uses B-tree indexing. Then, the complexity of this step is $\mathcal{O}(|Grantor_u| \cdot \log(|U|))$.

5. Compute the encrypted square distances between the querying user and the ones retrieved in step 4. The LBS provider executes this step. This operation requires three homomorphic additions and two multiplications. A ciphertext generated using SHE is represented as a matrix. So adding and multiplying two ciphertexts imply addition and multiplication of two matrices [SS15]. The addition and multiplication operations have a complexity of $\mathcal{O}(|M|^2)$ and $\mathcal{O}(|M|^3)$, respectively. Since $\mathcal{O}(|M|^3)$ dominates $\mathcal{O}(|M|^2)$, the complexity of computing one encrypted square distance is $\mathcal{O}(|M|^3)$. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot |M|^3)$.

6. Retrieve from the set of encrypted names stored at the LBS provider the encrypted names corresponding to each of the random numbers decrypted in step 3. The LBS provider executes this step. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot log(A))$.

7. Decrypt the information sent by the LBS provider: the encrypted square distances, the second layer of encryption of the encrypted names, and the encrypted public numbers. The ACS executes this step. It decrypt in total $3 \cdot |Grantor_u|$ ciphertexts. The square distances are encrypted using SHE, and the names and public numbers are encrypted using asymmetric encryption. Then, the ACS performs $|Grantor_u|$ SHE decryptions and $2 \cdot |Grantor_u|$ asymmetric decryptions. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot (|B(p_u)| \cdot \log(|B(p_u)|) + \log(N)^3))$.

8. Compute the shared keys and decrypt the encrypted names corresponding to users that fulfill the query condition. The user does this step. u computes in total $|Grantor_u|$ shared keys. The shared keys are generated using the DH protocol, which requires modular exponentiation. Given a public number of a user v, Z_v, and the secret number of u, η_u, the complexity of the modular exponentiation ${Z_v}^{\eta_u} \pmod p$ is $\mathcal{O}(\log(p)^2 \cdot \log(\eta_u))$, where p is the parameter of the DH protocol. The number of decryptions performed are $|Grantor_u|$, where each decryption has a complexity of $\mathcal{O}(1)$. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot \log(p)^2 \cdot \log(\eta_u))$.

By considering the step with the highest complexity that each entity of the system performs, one can easily construct the terms of Equation 6.1. □

6.3.2 2lABE - Time Complexity Analysis

We introduce further notation: Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups of the same order, where $\mathbb{G}$ and $\mathbb{G}_T$ are the groups selected during the initialization of the CP-ABE scheme. Further, let e be a bilinear map of the form $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. The encryption/decryption process using CP-ABE is based on bilenear mappings and operations in the groups $\mathbb{G}$ and $\mathbb{G}_T$. We use C_e and $C_{\mathbb{G}_T}$ to denote the complexity of computing e, and the complexity of performing an operation in the group $\mathbb{G}_T$, respectively.

Lemma 6.8. *Given a range query Range $= (u, d)$, the time complexities of the 2lABE approach at the user-side, the ACS, and the LBS provider are:*

$$\begin{aligned} TC_{user} &= \mathcal{O}\big(|Grantor_u| \cdot (C_e + C_{\mathbb{G}_T})\big) \\ TC_{ACS} &= \mathcal{O}\big(|Grantor_u| \cdot (|B(p_u)| \cdot \log(|B(p_u)|) + \log(N)^3)\big) \\ TC_{LBS} &= \mathcal{O}\big(|Grantor_u| \cdot |M|^3\big) \end{aligned} \tag{6.2}$$

Proof. We start by analyzing the complexity of the encryption/decryption process of each of the encryption schemes used in the *2lABE* approach, i.e., asymmetric encryption, SHE, and CP-ABE. As shown in the proof of Lemma 6.7, the encryption and decryption complexity of asymmetric encryption schemes are $\mathcal{O}(\log(N)^2)$ and $\mathcal{O}(\log(N)^3)$, respectively, and the encryption/decryption complexity of SHE is $\mathcal{O}(|B(m)| \cdot \log(|B(m)|))$, where m is the given message.

Next, the encryption complexity of CP-ABE schemes [LGR+12] depends on the size of the access policy and the cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$. The encryption costs using CP-ABE is $(n+3) \cdot C_{\mathbb{G}} + 2 \cdot C_{\mathbb{G}_T}$, where n is the size of the access policy and $C_{\mathbb{G}}$ is the complexity of performing an operation in the group $\mathbb{G}$ [XWL13]. In our scenario, the size of the access policy depend on the size of the set of grantees of each user. Then, the encryption complexity of CP-ABE is $\mathcal{O}(|Grantee_u| \cdot C_{\mathbb{G}} + C_{\mathbb{G}_T})$. The decryption cost using CP-ABE is $2 \cdot C_e + 2 \cdot C_{\mathbb{G}_T}$ [XWL13]. Then, the decryption complexiy of CP-ABE is $\mathcal{O}(C_e + C_{\mathbb{G}_T})$.

The following steps are required to compute a given range query with the *2lABE* approach.

(1)-(3) Similar to the step 1-3 of the *2lSE*.

(4) Retrieve the encrypted position and encrypted name corresponding to each decrypted identifier in step 3. The LBS provider executes this step. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot \log(|U|))$.

(5) Similar to the step 5 of the *2lSE*.

(6) Decrypt the encrypted square distances and the second layer of encryption of the encrypted names sent by the LBS provider. The total number of ciphertexts that the ACS has to decrypt is $2 \cdot |Grantor_u|$, where the square distances are encrypted using SHE and the names are encrypted using asymmetric encryption. This step is done by the ACS. The complexity of this step is $\mathcal{O}(|Grantor_u| \cdot (|B(p_u)| \cdot \log(|B(p_u)|) + \log(N)^3))$.

(7) Decrypt the encrypted names corresponding to users that fulfill the query condition. The user executes this step. The number of decryptions performed are $|Grantor_u|$. Then the complexity of this step is $\mathcal{O}(|Grantor_u| \cdot (C_e + C_{\mathbb{G}_T}))$.

By considering the step with the highest complexity that is performed by each entity of the system, one can easily construct the terms of Equation 6.2. □

6.3.3 Discussion

From our worst-case analysis, Lemmas 6.7 and 6.8, we observe that both approaches differ only at one entity: the user-side. The main difference relies on the decryption process, which is related to the encryption scheme used. With the *2lSE*, the encryption/decryption complexity of symmetric schemes is $\mathcal{O}(1)$. However, the querying user has to perform a total of $|Grantor_u|$ exponentiations module p to calculate the shared key. In contrast, with the *2lABE*, each user uses a single key to decrypt the received ciphertexts. However, the encryption/decryption complexity of CP-ABE schemes depends on the size of the access policy used to generate a given ciphertext and bilinear pairing operations, which are computationally expensive [XWL13, PYJ14].

In fact, both schemes have a trade-off between secrecy and efficiency, as we explain in the following. Our *2lSE* approach uses the DH protocol to build a shared secret key between two users. The secrecy of this protocol relies on the Computational DH problem (CDH) [DH76]. Our *2lABE* approach uses CP-ABE to encrypt the name of the users. CP-ABE schemes are built based on bilinear pairing functions, which themselves rely on the CDH problem. The secrecy and efficiency of both approaches rely, among others, on the hardness of the CDH problem, and the algebraic operations in the cyclic group that is being used. The CDH problem has a trade-off between secrecy and performance. A large order of a cyclic group implies higher secrecy [GKR04], i.e., less success probability of an adversary solving the CDH problem. However, the efficiency of computing algebraic operations is affected. Then one has to make a trade-off between the secrecy level needed and the efficiency of the approach.

Encryption Scheme	Library
Somewhat Homomorphic Encryption	Microsoft SEAL [SEA18]
Symmetric, Asymmetric encryption and CP-ABE	Java Pairing-Based Cryptography Library and the Cryptographic Packages namely javax.crypto and java.security

Table 6.2: Libraries Used for Implementation

Since a complexity analysis considers only the dominating operations, a worst-case analysis is insufficient to determine the performance of an algorithm in practice. For instance, for the complexity at the LBS provider, one can observe that both approaches have the same time complexity. However, one can find significant differences between both approaches in the less dominant steps. With the *2lSE* approach, the LBS provider has to perform $|Grantor_u|$ extra searches with complexity $\log(A)$, which will affect its performance. Therefore, we additionally perform experiments to validate our complexity analysis, evaluate how the differences between both approaches impact on their performance and determine at the end which approach performs better in practice.

6.4 Experiments

In this section, we present an experimental analysis of the performance of the *2lSE* and *2lABE* approaches.

6.4.1 Experiment Setup

Dataset and Query Sample

We use the Tokyo dataset [YZZY15] in our experiments, which is a dataset that contains 573703 real check-ins, i.e., positions. We choose a sample of 1000 users at random. Next, we divide the users into ten equally classes. We generate authorized access requests such that all the users in each class have the same number of grantors. Specifically, we selected at random from the dataset the following grantors sizes: 10, 25, 50, 100, 250, 500, 750, 1000, 2500, and 5000.

Encryption Algorithms

We use the following libraries for the implementation of our approaches:

Evaluation Measures

For our evaluation, we considered the access request and query phases. All other phases are executed only once, at least with respect to one of the entities of the systems.

We consider six measures: the storage size, the access request time, the query processing time at the user-side, the query processing at the LBS provider, the query processing time at the ACS, and the total query processing time. The total query processing time is the sum of the query processing time at each of the entities.

Note that positioning and query processing are two separate activities. In our experiments, we do not study the time that a positioning technology needs to provide the physical position of the users and its accuracy. The level of position accuracy and its computational time depend on the underlying technology and factors such as receiver noise and satellite geometry. For instance, under open sky conditions, GPS-based solutions have an accuracy of $3-5$ meters and a computational time of 3.6 seconds [ZB11]. In the case of GSM-based solutions, they have an accuracy of 65 - 134 meters [CSC+06] and a computation time of 7.04 milliseconds [IY11]. Improving the accuracy and computational time of the positioning technologies is not a topic of this dissertation.

6.4.2 Results

We now use the metrics specified in the previous subsection to evaluate experimentally our approahes.Note that we omit the network-communication time.

Storage-Size

Figure 6.11(a) shows the total storage size occupied by each of the approaches. The dark green and light green colors represent the storage size at the ACS and LBS providers, respectively. The *2lABE* approach requires more storage capacity. However, the difference between the storage capacity of both approaches is minimal (2 percent in our scenario).

Access Request Time

Given an authorized access request *accessReq*(u,v)=*true*, we measure the time required by the grantor v to generate and add the information needed at the ACS and the LBS provider, with each approach. Figure 6.11(b) displays the average

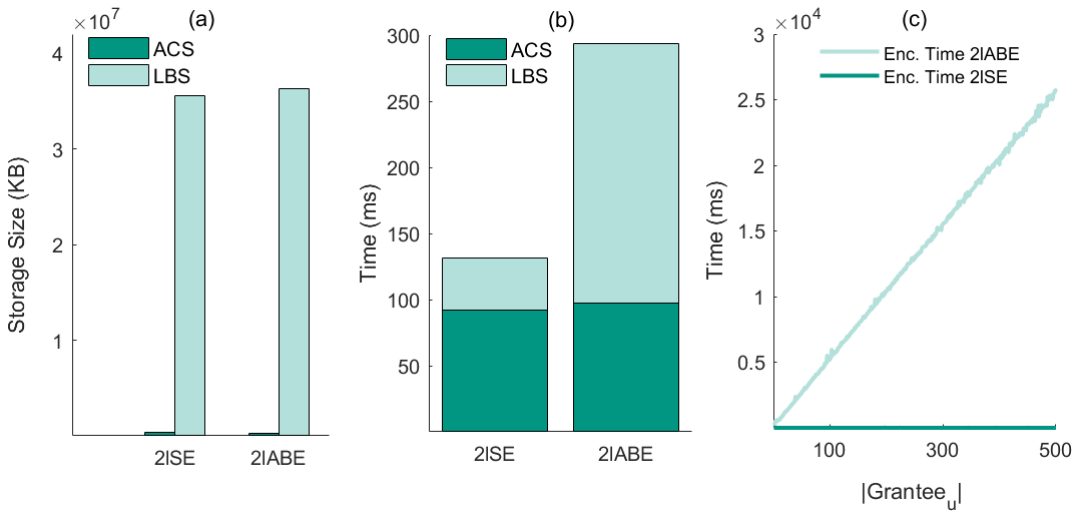


Figure 6.11: Storage size(a), Access request time (b) and Encryption time (c)

access request time with both approaches for our sample. Both approaches require the same amount of time to generate and store the corresponding information at the ACS (blue color). With the *2lABE* approach, the grantors require more time to generate and store the information at the LBS provider. Such a time increment is due to the encryption scheme used to encrypt the usernames stored at the LBS provider.

One could think that our results are specific to the number of grantees of the users in our sample. Next, we show that our results apply to any number of grantees. Lemma 6.8 shows that the encryption complexity using CP-ABE, which is used by the *2lABE* approach, depends, among others, on the number of grantees that a user has. To analyze in-depth, the effect of each encryption scheme, we select a user and increase his number of grantees, starting from 1 up to 500. Next, we measure the time required by the user to encrypt his username against the size of his set of grantees.

Figure 6.11(c) shows the encryption time with both approaches. One can observe that even for a set of grantees with cardinality one, the *2lSE* approach performs better than the *2lABE*. Furthermore, as expected from our complexity analysis, the encryption time with the *2lABE* grows linearly with the number of grantees. With the *2lSE* the encryption time is constant because each grantor generates a different ciphertext for each of his grantees. Consequently, with the *2lSE*, the encryption time of each ciphertext is independent of the number of grantees of a user. With the *2lABE*, although each grantor generates only one ciphertext,

which can be accessed by all of his grantees, the number of operations required to encrypt a ciphertext depends, among others, on the number of grantees.

Query Time

Figures 6.12(a), 6.12(b), and 6.12(c) show the average query processing time for each of the ten classes at the ACS, the LBS provider, and the user, respectively. The query processing times of the LBS provider and ACS with the *2lSE* approach are higher than that of the ones with the *2lABE*. In contrast, the query processing time of the user with the *2lSE* approach is less than the one with the *2lABE*. The reasons are as follows:

- With the *2lSE*, the LBS provider performs an additional search to recover the encrypted version of the name that can be decrypted by the querying user. The LBS provider does not perform this search with the *2lABE* because it stores one encrypted version of the name for each user.
- With the *2lSE* approach, the ACS decrypts three ciphertexts for each user v in the query answer: the second layer of the encrypted name of v, the encrypted square distance, and the encrypted public number Z_{η_v}. With the *2lABE* approach, it decrypts only the encrypted square distances and the second layer of the encrypted names.
- With the *2lSE* approach, the user has to compute the shared key and decrypt the names encrypted with symmetric encryption. With the *2lABE* the user decrypts the names encrypted with CP-ABE. The encryption/decryption process using CP-ABE is computationally more expensive than the one using symmetric encryption and even more expensive than computing the shared keys and decrypting.

To summarize, the query processing time of the *2lABE* at the LBS provider and the ACS is less than the one with the *2lSE*. However, the user-side performance of the *2lSE* is that much high that it compensates for the advantages of the *2lABE*. Figure 6.12(d) shows that the total average query processing time, i.e., the sum of the times required by the LBS provider, the ACS, and the user, with the *2lSE* approach is much less (approximately by a factor of 2) than the one with the *2lABE*.

Discussion

One crucial difference between the proposed approaches that affect query performance is the encryption scheme used to encrypt the usernames. In general, in our

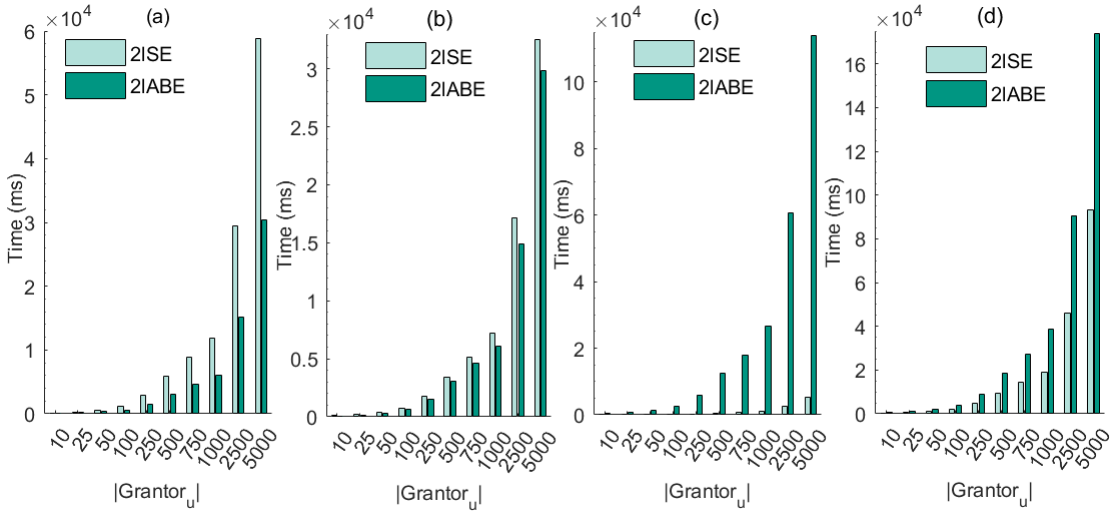


Figure 6.12: Average query processing time at the ACS (a), the LBS provider (b) and the user-side (c). Total average query processing time (d)

scenario, as our experimental results show, having multiple encrypted copies of a message (*2lSE*) is more efficient in terms of query performance.

With the *2lSE*, during the query process, the additional searching step needed doubles the query processing time at the LBS provider side compare to the *2lABE* approach. At the user-side, the opposite happens with the decryption step— with the *2lABE* the decryption time increases by more than twice compared to the decryption time of the *2lSE*. However, the time contribution to the total query processing time of the LBS provider is less in comparison to that of the user. The total query processing time reflects these time differences. The *2lSE* approach is twice more efficient than the *2lABE*, which is impressive. Therefore, one can say that the 2 percent extra storage required by the *2lSE* pays off with better query performance.

Our results not only apply to any social network with the same average number of friends as Orkut and Livejournal, including Facebook, where the average number of friends is 322 [SU15], but also to unrealistic scenarios where the size of the set of grantors is 5000. These results are in line with our complexity analysis, and one may interpret them as an indication that our analysis also holds for the average case. Consequently, the *2lSE* approach is the most feasible option in our scenario.

6.5 Related Work

We categorize exiting works aiming to preserve location secrecy in two groups: location secrecy in LBS and location secrecy in mSNs. The main difference between these two categories is that approaches in the first one do not consider access policies, i.e., users are allowed to access the location of any user in the system.

Location secrecy in LBS: Several techniques have been proposed in the literature to achieve location secrecy in LBS such as:

- Mix zones: This concept was first proposed in [BS03] and has been widely extended in [FSH09, PLL+14]. The goal of mix zones is to prevent an adversary from tracking long-term user movements [BS04]. Mix zones are spatial regions with a predetermined size. Users inside a mix zone do not report their position or communicate with the LBS provider. These approaches focus on data anonymization and replace the user identity with a pseudonym. They offer anonymity guarantees by changing the pseudonyms of users inside a mix zone such that an adversary will not be able to link users that go inside the mix zone with those leaving it. Since these approaches do not hide the position of users outside the mix zones, one can perform any query on the plaintext data. However, users inside the mix zones do not communicate with the LBS provider, and therefore they cannot get any service. This lack of communication affects functionality.

- Coordinates transformation: Approaches in this area aim to guarantee that an adversary should not be able to learn the position of the users. Coordinates transformation consists on mapping the location of users to another space coordinate and addressing the query on the transformed space. In the approach presented in [Gut06], users use a transformation function over their physical position before sending it to the LBS provider. The transformation function consists of shifts and rotations. Each user sets the parameters of the transformation function and distributes these parameters among the rest of users to allow them to recover the original physical position. This approach is subject to data reconstruction attacks [KHC16]. To avoid this kind of attacks, the authors in [LBCP08] use agents to transform the physical positions of users. Agents are trusted third parties servers that act as middleware between the users and the LBS provider. Agents periodically change their transformation functions, which prevents the adversaries from analyzing the data. These approaches allow users to query for one specific user at each time, i.e., point queries. The secrecy of this approach relies on trusted third servers.

- Cryptography: Approaches like the ones proposed in [NTL+11, ZLLH12] keep the position of users secret from any adversary, including the service provider. These approaches use encryption techniques and hash functions to compute proximity between two users privately. The authors in [ZLLH12] use, besides hash functions, location tags to prevent users from cheating their location, i.e., announcing an untruthful location. Location tags are pieces of information obtained from the network protocol like 802.11 frames in WiFi networks. In these approaches, users are allowed to query for the proximity of one specific user at each time, i.e., point queries.
- Other works in the area [CML06, MCA06, GKK+08, YPBV14] focus on executing private queries over public data. The public data is information about points of interest, e.g., museums, restaurants, which is owned by the LBS provider. The goal of these approaches is to allow users to execute queries without revealing their location to the LBS provider. In our scenario, the data stored at the LBS provider is not public data and must be kept secret from any adversary, including the LBS provider.

Location secrecy in mSNs: Approaches in this area restrict access to the physical position of users based on access control policies. Each user defines a set of users who are authorized to read his physical position. The authors in [PWS+14] introduced a scheme to guarantee location secrecy. Their approach uses encryption and coordinate transformation techniques. The LBS provider stores the transformed data. For querying purposes, users have to distribute among their friends the transformation parameters and the corresponding decryption keys. This scheme deals with point and nearest neighbor queries.

Wei et al. [WXL12] proposed an approach called Mobishare. The architecture of this approach consists of a trusted central tower, an untrusted LBS provider and an untrusted ACS. Mobishare uses dummy techniques to prevent the LBS provider and the ACS from learning the identities of the users and their physical positions. Before outsourcing the data to the LBS provider, Mobishare replaces users identities with pseudonyms and adds dummy pseudonyms together with dummy locations. Since the LBS provider stores the position of users in plaintext, it can compute any query. Before sending the final query answer to the user, the ACS filters the data based on the access policies, and the central tower replaces pseudonyms with user identities. However, an adversary who can observe query executions could be able to identify real pseudonyms from dummy ones. Using this information, an adversary could link a pseudonym with a user identity and therefore learn the position of the user. The adversary could also acquire information about access policies, i.e., the set of users that have allowed a given user to access their position. To avoid this leakage, the authors in [LLC+14] extended the

previous approach by adding dummy queries and using a private set intersection protocol. The authors in [LYL+17] extended the previous approach by introducing a new architecture with multiple LBS providers. The authors aim to prevent an adversary from identifying queries coming from the same user because an adversary could use this information to learn the identities of the users. To summarize, approaches in this area focus on point or nearest neighbor queries, or they rely on trusted central towers. Furthermore, their adversary model is weaker than ours, as explained in Section 6.1.2.

6.6 Summary

Location-based services are an essential feature provided by MSNs. However, users usually are reluctant to share their position with others due to secrecy reasons. In this chapter, we have shown how to offer LBS, in the example of range queries, in MSNs with a revocation feature while providing to the users the secrecy guarantees $G_{position}$, $G_{distance}$, and $G_{authorization}$ under collusion assumption. This chapter presented two approaches namely two-layer symmetric encryption, *2lSE*, and two-layer attribute-based encryption, *2lABE*. The main differences between them is that they use, among other encryption schemes, symmetric and attribute-based encryption, respectively. A complexity analysis of the query phase tells us that both approaches differ only at one entity—the user-side. We have further compared our approaches experimentally. Although with the *2lABE*, the key management is more straightforward than with the *2lSE*, and the LBS provider stores a single encrypted copy for each message, we found that our former solution is on average twice more efficient in our scenario.

7 Conclusion and Future Work

Most of the information stored in online social networks about their users is private. Online social networks use access control systems to protect the information from unauthorized access and avoid secrecy issues. In this dissertation, we have tackled two open issues regarding these systems: the flexibility of existing access control models and the distrusts of users towards the entity that manages the authorization mechanism, i.e., the service providers. We have developed techniques and models to address these two issues.

7.1 Summary

In Chapters 2 and 3, we first introduced two research areas relevant to our work — access control systems and cryptography. We have formally defined the basic notions and have given and overview of these areas. We have used these notions in the following chapters to build our proposed approaches.

Chapter 4 studied the first issue — the flexibility problem of existing access control models. Studies in different fields [FFG02, SBC09, FF06] have revealed that, while humans are self-interested, they often deviate from this attitude reciprocally, i.e., in response to friendly actions, people are more cooperative. None of the existing access control models capture this reciprocity phenomenon. We increased their flexibility by defining the syntax and semantics of a new type of authorization, called *mutual authorizations*. *Mutual authorizations* allow users to grant access to their resources to users that enable them to the same. To focus on the idea of reciprocity itself, we have first assumed that all resources have the same sensitivity degree, e.g., physical positions. We have also generalized our model to support settings where resources have different sensitivity degrees, e.g., health records. Deciding whether two resources have the same degree of sensitivity is a general problem that is far from being solved in an automated manner —in any scenario. Our general model of *mutual* authorization, inspired by trust models from the literature such as [KSGM03, ZH07, HLB11], aims to alleviate this problem. To show how to integrate *mutual authorizations* with existing services, we used location-based services. We presented two approaches for such integration, proved

their soundness, and conducted time complexity analyses of them. F We have evaluated the impact of *mutual* authorizations on the performance of an access decision and the performance of our approaches experimentally.

Chapter 5 and 6 studied the second issue —the distrust of users towards the service providers— from two different angles.

Chapter 5 considered the scenario in which the user, i.e., the entity who wants to perform queries over the data, is also the data owner. A broad range of real-world datasets exhibits a graph structure, and many real graphs, such as the email network or the Web, follow a scale-free power-law distribution. Here, we focused on datasets with these characteristics. We proposed a secrecy notion for graph-structured data. We then presented our bucketization approach for secure outsourcing of this kind of data. We showed that finding an optimal bucketization tailored to graph-structured data is NP-hard. We therefore came up with a heuristic, which guarantees that the worst bucketization solution will be off by a factor of $\frac{11}{9}$ of the optimal one. Our approach encrypts the label of the nodes and edges to protect them against deterministic chosen-plaintext attacks and creates buckets with the edges to protect against frequency attacks. We also came up with a performance model for query processing on scale-free graphs. We conducted systematic experiments to validate the accuracy of our estimation model and demonstrate the efficiency of our proposed bucketization technique.

Chapter 6 considered the scenario in which each user owns a portion of the data, and users are not allowed to access to the data needed to compute queries. But they are allowed to access query results over the data. Specifically, we focused on mobile social network scenarios, where multiple users want to access location-based services while keeping their information secret from both the service provider and unauthorized users. We have shown how to offer location-based services, in the example of range queries, in mobile social networks with a revocation feature while providing secrecy guarantees under collusion assumption to the users. We introduced two approaches namely two-layer symmetric encryption, *2lSE*, and two-layer attribute-based encryption, *2lABE*. We presented complexity analyses of our approaches, which allow us to compare them. Finally, we conducted experiments to validate our complexity analyses and study the performance of our approaches.

7.2 Future Work

This dissertation provides techniques and models to address secrecy issues in online social networks. Possible improvements and future research steps include the following:

Our work regarding *mutual* authorizations assumes that the service provider is trusted. It will be interesting to consider an adversary model with an untrusted service provider and study how to combine encryption techniques into our *mutual* authorization model to achieve data secrecy. Another direction in this area is to explore how to incorporate *mutual* authorizations into existing model-checking techniques, like the approaches presented in [SMJG05, RLFK04], to evaluate the correctness of access request decisions. Yet another direction is to extend *mutual* authorizations by considering different resources with more than one controller, i.e., users who can regulate access to the resource, and users owning more than one resource with different degrees of sensitivity.

Finally, in our mobile social network scenario, one could study how to offer location-based services while providing secrecy guarantees considering a different kind of access policies. For instance, one can consider integrating our *mutual authorization* model, or location constraints [BCDP05], where users restrict access to their resources based on the location of the accessing user. Another direction is to extend our approaches with position verification techniques [VdHKAS+16] to verify that the positions of the users that are input to the system are real.

Appendices

A Hardness Result

We start by introducing the Bin-packing problem problem.

Definition A.1: Bin-packing Problem

Let a set of n bins $C = \{c_1, c_2, \ldots, c_n\}$ and the same number of n items $I = \{a_1, a_2, \ldots, a_n\}$ be given. All bins have equal capacity w_c, and the weight of each item $a_i \in I, w_{a_i}$, is smaller than or equal to the capacity w_c. The **Bin-packing problem** (BP problem) is to find a mapping $BP : I \to C$ of each item in I to one bin in C such that the following Constraints $bp1$, $bp2$ and $bp3$ are met:

(bp1) An item is assigned to only one bin.

(bp2) The sum of the weights of all items assigned to a bin does not exceed the bin capacity w_c. Formally, $\forall c_j \in C : W_{c_j} \leq w_c$ where $W_{c_j} = \sum_{a_i \in \{a \in I | BP(a) = c_j\}} w_{a_i}$.

(bp3) The number m of bins used is as small as possible, i.e., minimize $m = \left|\bigcup_{a_i \in I} \{BP(a_i)\}\right|$.

For the hardness proof, we use a restricted version of the bin-packing problem. Here, we limit the weight of the items to be polynomial in n. Since the bin-packing problem is strongly NP-complete, bounding any of its numerical parameters by a polynomial in the length of the input, the resulting problem remains NP-complete [GJ78].

We introduce Lemmas A.1 and A.2, which we use in the hardness proof. These two lemmas help us to show that an instance of the bin-packing problem, called *initial BP*, can be reduced in polynomial time to an instance of the bucketization problem, called *transformed BP*, and to prove that one can transform a given solution of the $transformed\ BP$ to a solution of the *initial BP* in polynomial time. In what follows, we identify the steps required to construct the *transformed BP*.

Transformed BP construction: Given a bin-packing problem with a set of items I, the *transformed BP* is constructed as follows:

- For each item $a_i \in I$, create the set of nodes $V_i = \{a_i, a_{i1}, a_{i2}, \ldots, a_{iw_{a_i}}\}$ and the set of edges $E_i = \{(a_i, a_{i1}), (a_i, a_{i2}), \ldots, (a_i, a_{iw_{a_i}})\}$.
- The graph that represents the *transformed BP* is $G = (\cup_{i=1}^{n} V_i, \cup_{i=1}^{n} E_i)$.

Lemma A.1. *Given an initial bin-packing problem, initial BP, the transformed BP can be constructed in polynomial time.*

Proof. For each item $a_i \in I$, in order to build *transformed BP*, we need $(w_{a_i} + 1)$ nodes and w_{a_i} edges. Altogether this requires $\sum_{i=1}^{n}(w_{a_i} + 1)$ steps. Since in the restricted version of the bin-packing problem, the weight of the items are polynomial in n, the construction is still polynomial in n. Then an *initial BP* can be transformed to a *transformed BP* in polynomial time. □

Example A.1 illustrates the construction of the *transformed BP*.

Example A.1. Consider the *initial BP* with set of items $I = \{a_1, a_2, a_3, a_4\}$ with weights $w_{a_1} = 3, w_{a_2} = 1, w_{a_3} = 2, w_{a_4} = 4$ and the set C of bins with capacity $w_c = 5$. Figure. A.1 shows the corresponding *transformed BP*.

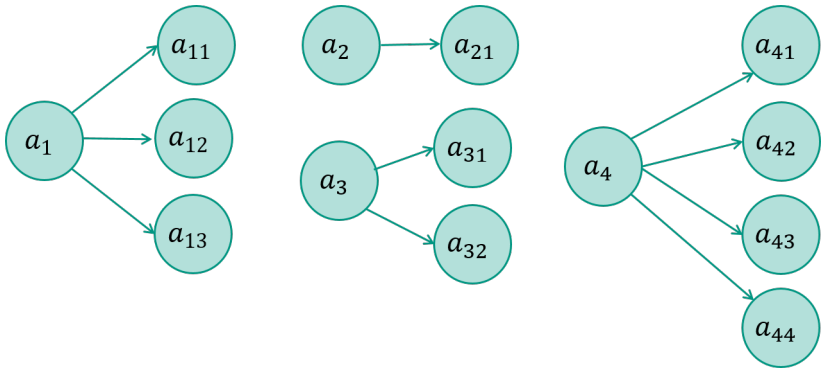


Figure A.1: *Transformed BP* of Example A.1

Once we have built *transformed BP*, we can run an algorithm that solves the bucketization problem on it, by setting *maxEdges* to w_c. The solution of the *transformed BP* is a bucketization structure BS that fulfills Constraints c1-c4. The following set of buckets $\mathcal{B}_G$, which is part of the bucketization structure BS, is a possible solution of the *transformed BP* of Example A.1:

$$\mathcal{B}_G = \left\{ \begin{array}{l} b_1 : (a_1, a_{11}), (a_1, a_{12}), (a_1, a_{13}), (a_3, a_{31}), (a_3, a_{32}) \\ b_2 : (a_2, a_{21}), (a_4, a_{41}), (a_4, a_{42}), (a_4, a_{43}), (a_4, a_{44}) \end{array} \right\}$$

where b_1 and b_2 are the bucket identifiers. Since we set $maxEdges = w_c$, it holds for all buckets $b \in \mathcal{B}_G$ that $freq(b) \leq w_c$.

The next lemma, Lemma A.2, states that a solution of the *initial BP* can be constructed in polynomial time from a solution of the *transformed BP*. Before moving to Lemma A.2, we first explain the solution construction process.

Initial BP solution construction: A solution of the *initial BP* can be constructed from a solution of the *transformed BP* as follows:

- Select the bins $c_j, \ldots, c_m$ needed to store the items in I, where $m = |\mathcal{B}_G|$.
- Map each item $a_i \in I$ to one bin c_j, where $j \in \{1, \ldots, m\}$, as follows: $\forall b_j \in \mathcal{B}_G : I_{c_j} = \{a_i \mid \exists y \in \{1, \ldots, w_{a_i}\}, (a_i, a_{iy}) \in b_j\}$. Then $\forall a_i \in I_{c_j} : a_i \rightarrow c_j$.

The following shows the solution constructed for the *initial BP* from the *transformed BP* of Example A.1:

$$m = |\mathcal{B}_G| = 2$$
$$I_{c_1} = \{a_1, a_3\},\ I_{c_2} = \{a_2, a_4\}$$
$$a_1 \rightarrow c_1$$
$$a_3 \rightarrow c_1$$
$$a_2 \rightarrow c_2$$
$$a_4 \rightarrow c_2$$

Lemma A.2. *A solution of the transformed BP can be transformed to a solution of the corresponding initial BP in polynomial time.*

Proof. Consider a bucketization of the *transformed BP* that fulfills Constraints c_1-c_4. We transform it to a bin-packing solution with the *solution construction process*. Now, we proceed to demonstrate that the transformed solution fulfills the constraints of the bin-packing problem, bp1 to bp3, with respect to the *initial BP*. We start by analyzing the constraints of the bin-packing problem and of the bucketization problem. First, Constraint bp1 is fulfilled because of Constraints c_1 and c_3 of the bucketization problem. Constraint c_1 ensures that each edge is assigned to only one bucket. Then $\forall i \neq j : c_i \cap c_j = \varnothing$. Together with the fact that for all items $a_i \in I, w_{a_i} \leq maxEdges$, Constraint $c3$ ensures that the edges belonging to the same node are placed in the same bucket.

Second, Constraint bp2 is fulfilled because of Constraint c_2 of the bucketization problem. For all bins $c_j \in C, W_{c_j} = freq(b_j)$ and $maxEdges = w_c$, then $freq(b_j) \leq$

w_c, which fulfills Constraint bp1. Third, bp3 is fulfilled because of Constraint c_4. The number of buckets is the number of bins used in the *initial BP* solution. Then minimizing the buckets is the same as minimizing the number of bins used.

Finally, a bucketization solution of a *transformed BP* can be transformed to a solution of the *initial BP* in polynomial time. The reconstruction requires one lookup in all the elements of each bucket $b_i \in \mathcal{B}_G$. Then the complexity of the reconstruction is $\mathcal{O}(m)$, where m is the total number of elements, i.e., edges, stored in $\mathcal{B}_G$. Since $\mathcal{B}_G$ stores the set of items I, $m = \sum_{i=1}^{n} w_{a_i}$.

□

Theorem A.3. *Finding an optimal bucketization that meets Constraints c_1-c_4 is NP-hard.*

Proof. With Lemma A.1 and Lemma A.2 we have shown that an instance of the bin-packing problem can be reduced to an instance of the bucketization problem in polynomial time. Since the bin-packing problem is NP-hard [Joh73], the optimal bucketization problem is NP-hard as well.

□

List of Figures

List of Tables

Bibliography

[AAUC18] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)*, 51(4):79, 2018.

[ABG+05] Gagan Aggarwal, Mayank Bawa, Prasanna Ganesan, Hector Garcia-Molina, Krishnaram Kenthapadi, Rajeev Motwani, Utkarsh Srivastava, Dilys Thomas, and Ying Xu. Two can keep a secret: A distributed architecture for secure database services. *CIDR 2005*, 2005.

[ASV08] Vijayalakshmi Atluri, Heechang Shin, and Jaideep Vaidya. Efficient security policy enforcement for the mobile environment. *Journal of Computer Security*, 16(4):439–475, 2008.

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[BBHJ11] Christoph Bösch, Richard Brinkman, Pieter H Hartel, and Willem Jonker. Conjunctive wildcard search over encrypted data. *Secure data management*, 6933:114–127, 2011.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Annual International Cryptology Conference*, pages 535–552. Springer, 2007.

[BCDP05] Elisa Bertino, Barbara Catania, Maria Luisa Damiani, and Paolo Perlasca. Geo-rbac: A spatially aware rbac. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, New York, NY, USA, 2005.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *Annual International Cryptology Conference*, pages 26–45. Springer, 1998.

[BF18] Elisa Bertino and Elena Ferrari. Big data security and privacy. In *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, pages 425–439. Springer, 2018.

[Bon98] Dan Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

[BS03] Alastair R Beresford and Frank Stajano. Location privacy in pervasive computing. *IEEE Pervasive computing*, (1):46–55, 2003.

[BS04] Alastair R Beresford and Frank Stajano. Mix zones: User privacy in location-aware services. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 127–131. IEEE, 2004.

[BS08] Dan Boneh and Victor Shoup. *A graduate course in applied cryptography*. https://crypto.stanford.edu/ dabo/cryptobook/, 2008.

[BSK16] Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan. Label-based access control: An abac model with enumerated authorization policy. In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, pages 1–12. ACM, 2016.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Security and Privacy, 2007.*, 2007.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 136–145. IEEE, 2001.

[CFB05] Barbara Carminati, Elena Ferrari, and Elisa Bertino. Securing xml data in third-party distribution systems. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 99–106. ACM, 2005.

[CGKO11] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[CML06] Chi-Yin Chow, Mohamed F Mokbel, and Xuan Liu. A peer-to-peer spatial cloaking algorithm for anonymous location-based service. In *Proceedings of the 14th annual ACM international symposium on*

Advances in geographic information systems, pages 171–178. ACM, 2006.

[Com18] MacKenzie F Common. Facebook and cambridge analytica: let this be the high-water mark for impunity. *LSE Business Review*, 2018.

[Com19] Federal Trade Commission. Data security, 2019.

[Con18] Nicholas Confessore. Cambridge analytica and facebook: The scandal and the fallout so far. *The New York Times*, 4:2018, 2018.

[CRK+13] Jianneng Cao, Fang-Yu Rao, Mehmet Kuzu, Elisa Bertino, and Murat Kantarcioglu. Efficient tree pattern queries on encrypted xml documents. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 111–120. ACM, 2013.

[CSC+06] Mike Y Chen, Timothy Sohn, Dmitri Chmelev, Dirk Haehnel, Jeffrey Hightower, Jeff Hughes, Anthony LaMarca, Fred Potter, Ian Smith, and Alex Varshavsky. Practical metropolitan-scale positioning for gsm phones. In *International Conference on Ubiquitous Computing*, pages 225–242. Springer, 2006.

[CYW+11] Ning Cao, Zhenyu Yang, Cong Wang, Kui Ren, and Wenjing Lou. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 393–402. IEEE, 2011.

[CZZ16] Qi Cheng, Jun Zhang, and Jincheng Zhuang. Lwe from non-commutative group rings. *arXiv preprint arXiv:1612.06670*, 2016.

[Dam88] Ivan Bjerre Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In *Conference on the Theory and Application of Cryptography*, pages 328–335. Springer, 1988.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[Fal11] Howard Falk. Applications, architectures, and protocol design issues for mobile social networks: A survey. *Proceedings of the IEEE*, 99(12):2125–2129, 2011.

[FCC+15] Zhe Fan, Byron Choi, Qian Chen, Jianliang Xu, Haibo Hu, and Sourav S Bhowmick. Structure-preserving subgraph query services. *IEEE Transactions on Knowledge and Data Engineering*, 27(8):2275–2290, 2015.

[Fer10] Elena Ferrari. *Access Control in Data Management Systems*. Morgan and Claypool Publishers, 2010.

[FF06] Armin Falk and Urs Fischbacher. A theory of reciprocity. *Games and economic behavior*, 54(2), 2006.

[FFG02] Ernst Fehr, Urs Fischbacher, and Simon Gächter. Strong reciprocity, human cooperation, and the enforcement of social norms. *Human Nature*, 13(1), 2002.

[fFRoE18] European Union Agency for Fundamental Rights and Council of Europe. *Handbook on European data protection law*. Publications Office of the European Union, 2018.

[FSH09] Julien Freudiger, Reza Shokri, and Jean-Pierre Hubaux. On the optimal placement of mix zones. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 216–234. Springer, 2009.

[GJ78] Michael R Garey and David S Johnson. "strong"np-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[GJ79] Michael R Garey and David S Johnson. A guide to the theory of np-completeness. *WH Freemann, New York*, 70, 1979.

[GKK+08] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. Private queries in location based services: anonymizers are not necessary. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 121–132. ACM, 2008.

[GKR04] Rosario Gennaro, Hugo Krawczyk, and Tal Rabin. Secure hashed diffie-hellman over non-ddh groups. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 361–381. Springer, 2004.

[GTK01] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *ACM SIGMOD Record*, volume 30, pages 461–472. ACM, 2001.

[Gut06] Andreas Gutscher. Coordinate transformation-a solution for the privacy problem of location based services? In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 7–pp. IEEE, 2006.

[GZC08] Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. Relbac: Relation based access control. In *Semantics, Knowledge and Grid, 2008. SKG'08. Fourth International Conference on*. IEEE, 2008.

[HA11] Hongxin Hu and Gail-Joon Ahn. Multiparty authorization framework for data sharing in online social networks. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2011.

[HAJ12] Hongxin Hu, Gail-Joon Ahn, and Jan Jorgensen. Multiparty access control for online social networks: model and mechanisms. *IEEE Transactions on Knowledge and Data Engineering*, 25(7):1614–1627, 2012.

[HAK00] Alexander Hinneburg, Charu Aggarwal, and Daniel A Keim. What is the nearest neighbor in high dimensional spaces? In *Proceeding of the 26th VLDB*, 2000.

[HFK+13] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162), 2013.

[HIM05] Hakan Hacıgümüş, Bala Iyer, and Sharad Mehrotra. Query optimization in encrypted database systems. In *International Conference on Database Systems for Advanced Applications*, pages 43–55. Springer, 2005.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer, 2010.

[HLB11] Christian Hutter, Raphael Lorch, and Klemens Bohm. Evolving cooperation through reciprocity using a centrality-based reputation system. In *2011 IEEE/WIC/ACM International Conferences on*

Web Intelligence and Intelligent Agent Technology, volume 2, pages 264–271. IEEE, 2011.

[HLK19] Florian Hahn, Nicolas Loza, and Florian Kerschbaum. Joins over encrypted data with fine granular security. In *Data Engineering, 2019. ICDE 2019. IEEE 35th International Conference on*, pages 674–685. IEEE, 2019.

[HMS98] Elizabeth Hoffman, Kevin A McCabe, and Vernon L Smith. Behavioral foundations of reciprocity: Experimental economics and evolutionary psychology. *Economic inquiry*, 36(3):335–352, 1998.

[HMT04] Bijit Hore, Sharad Mehrotra, and Gene Tsudik. A privacy-preserving index for range queries. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 720–731. VLDB Endowment, 2004.

[HVS+10] Xiaoyun He, Jaideep Vaidya, Basit Shafiq, Nabil Adam, and Xiaodong Lin. Reachability analysis in privacy-preserving perturbed graphs. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 691–694. IEEE, 2010.

[IMI10] Sergio Ilarri, Eduardo Mena, and Arantza Illarramendi. Location-dependent query processing: Where we are and where we are heading. *ACM Computing Surveys (CSUR)*, 42(3), 2010.

[IY11] Mohamed Ibrahim and Moustafa Youssef. Cellsense: An accurate energy-efficient gsm positioning system. *IEEE Transactions on Vehicular Technology*, 61(1):286–296, 2011.

[Joh73] David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[KHC16] Hyeong-Il Kim, Seungtae Hong, and Jae-Woo Chang. Hilbert curve-based cryptographic transformation scheme for spatial query processing on outsourced private data. *Data & Knowledge Engineering*, 104:32–44, 2016.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340. ACM, 2016.

[KL07] Jonathan Katz and Yehuda Lindell. Introduction to modern cryptography, 2007.

[KMVOV96] Jonathan Katz, Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM, 2012.

[KSGM03] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM, 2003.

[Kun13] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

[LBCP08] Dan Lin, Elisa Bertino, Reynold Cheng, and Sunil Prabhakar. Position transformation: a location privacy protection method for moving objects. In *Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop*, pages 62–71, 2008.

[LDR06] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Mondrian multidimensional k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 25–25. IEEE, 2006.

[LG09] G Locke and P Gallagher. Fips pub 186-3: Digital signature standard (dss). *Federal Information Processing Standards Publication*, 3:186–3, 2009.

[LGR+12] Xiaohui Li, Dawu Gu, Yanli Ren, Ning Ding, and Kan Yuan. Efficient ciphertext-policy attribute based encryption with hidden policy. In *International Conference on Internet and Distributed Computing Systems*, pages 146–159. Springer, 2012.

[LK+07] Marc Lasserre, Vach Kompella, et al. Virtual private lan service (vpls) using label distribution protocol (ldp) signaling. Technical report, RFC 4762, January, 2007.

[LK14] J Leskovec and A Krevl. Snap datasets: Stanford large network dataset collection. *URL http://snap.stanford.edu/data/cit-HepTh.html*, 2014.

[LLC+14] Jingwei Li, Jin Li, Xiaofeng Chen, Zheli Liu, and Chunfu Jia. {MobiShare}+: Security improved system for location sharing in mobile online social networks. *J. Internet Serv. Inf. Secur.*, 4(1):25–36, 2014.

[LLDM09] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[LMSV11] Jake Loftus, Alexander May, Nigel P Smart, and Frederik Vercauteren. On cca-secure somewhat homomorphic encryption. In *International Workshop on Selected Areas in Cryptography*, pages 55–72. Springer, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[LS15] Silvio Lattanzi and Yaron Singer. The power of random neighbors in social networks. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, 2015.

[LWW+10] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Fuzzy keyword search over encrypted data in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–5. IEEE, 2010.

[LYL+17] Jin Li, Hongyang Yan, Zheli Liu, Xiaofeng Chen, Xinyi Huang, and Duncan S Wong. Location-sharing systems with enhanced privacy in mobile online social networks. *IEEE Systems Journal*, 11(2):439–448, 2017.

[MC15] Rajendra Prasad Mahapatra and Partha Sarathi Chakraborty. Comparative analysis of nearest neighbor query processing techniques. *Procedia Computer Science*, 57, 2015.

[MCA06] Mohamed F Mokbel, Chi-Yin Chow, and Walid G Aref. The new casper: Query processing for location services without compromis-

ing privacy. In *Proceedings of the 32nd international conference on Very large data bases*, pages 763–774. VLDB Endowment, 2006.

[MKNK15] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. Grecs: graph encryption for approximate shortest distance queries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 504–517. ACM, 2015.

[MP10] Hossein Maserrat and Jian Pei. Neighbor query friendly compression of social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 533–542. ACM, 2010.

[MPS99] S Muthukrishnan, Viswanath Poosala, and Torsten Suel. On rectangular partitionings in two dimensions: Algorithms, complexity and applications. In *International Conference on Database Theory*, pages 236–256. Springer, 1999.

[MS18] Dinesh P Mehta and Sartaj Sahni. *Handbook of data structures and applications*. Chapman and Hall/CRC, 2nd edition edition, 2018.

[MTR+09] Clara Mancini, Keerthi Thomas, Yvonne Rogers, Blaine A Price, Lukazs Jedrzejczyk, Arosha K Bandara, Adam N Joinson, and Bashar Nuseibeh. From spaces to places: emerging contexts in mobile privacy. In *Proceedings of the 11th international conference on Ubiquitous computing*, pages 1–10. ACM, 2009.

[MVW98] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Wavelet-based histograms for selectivity estimation. In *ACM SIGMoD Record*, volume 27, pages 448–459. ACM, 1998.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[NTL+11] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, Dan Boneh, et al. Location privacy via private proximity testing. In *NDSS*, volume 11, 2011.

[OEHB11] Ghizlane Orhanou, Saïd El Hajji, and Youssef Bentaleb. Eps aes-based confidentiality and integrity algorithms: Complexity study. In *Multimedia Computing and Systems (ICMCS), 2011 International Conference on*, pages 1–4. IEEE, 2011.

[OKM19] Safa Otoum, Burak Kantarci, and Hussein T Mouftah. On the feasibility of deep learning in sensor network intrusion detection. *IEEE Networking Letters*, 1(2):68–71, 2019.

[OP03] Sejong Oh and Seog Park. Task–role-based access control model. *Information systems*, 28(6), 2003.

[PLL+14] Balaji Palanisamy, Ling Liu, Kisung Lee, Shicong Meng, Yuzhe Tang, and Yang Zhou. Anonymizing continuous queries with delay-tolerant mix-zones over road networks. *Distributed and Parallel Databases*, 32(1):91–118, 2014.

[PU09] A John Prakash and V Rhymend Uthariaraj. Multicrypt: A provably secure encryption scheme for multicast communication. In *Networks and Communications, 2009. NETCOM'09. First International Conference on*, pages 246–253. IEEE, 2009.

[PWS+14] Krishna PN Puttaswamy, Shiyuan Wang, Troy Steinbauer, Divyakant Agrawal, Amr El Abbadi, Christopher Kruegel, and Ben Y Zhao. Preserving location privacy in geosocial applications. *IEEE TMC*, 13(1):159–173, 2014.

[PYJ14] Liaojun Pang, Jie Yang, and Zhengtao Jiang. A survey of research progress and development tendency of attribute-based encryption. *The Scientific World Journal*, 2014, 2014.

[Rab93] Matthew Rabin. Incorporating fairness into game theory and economics. *The American economic review*, 1993.

[RH05] Naveen Reddy and Jayant R Haritsa. Analyzing plan diagrams of database query optimizers. In *Proceedings of the 31st international conference on Very large data bases*, pages 1228–1239. VLDB Endowment, 2005.

[Rib12] Paulo Ribenboim. *The book of prime number records*. Springer Science & Business Media, 2012.

[RLFK04] Indrakshi Ray, Na Li, Robert France, and Dae-Kyoo Kim. Using uml to visualize role-based access control constraints. In *Proceedings of the ninth ACM symposium on Access control models and technologies*. ACM, 2004.

[Rob57] C Carl Robusto. The cosine-haversine formula. *The American Mathematical Monthly*, 64(1), 1957.

[SBC09] Luca Stanca, Luigino Bruni, and Luca Corazzini. Testing theories of reciprocity. *Journal of economic behavior & organization*, 71(2), 2009.

[SCFY96] Ravi S Sandhu, Edward J Coynek, Hal L Feinsteink, and Charles E Youmank. Role-based access control models. *IEEE computer*, 29(2), 1996.

[SdV00] Pierangela Samarati and Sabrina Capitani de Vimercati. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer, 2000.

[SEA18] Microsoft SEAL (release 3.1). `https://github.com/Microsoft/SEAL`, December 2018. Microsoft Research.

[SEGB] Gabriela Suntaxi, Aboubakr Achraf El Ghazi, and Klemens Böhm. Preserving secrecy in mobile social networks. *ACM Transactions on Cyber-Physical Systems*, in press.

[SEGB19a] Gabriela Suntaxi, Aboubakr Achraf El Ghazi, and Klemens Böhm. On preserving secrecy in mobile social networks. Technical report, Karlsruhe Institute of Technology, 2019.

[SEGB19b] Gabriela Suntaxi, Achraf El Ghazi, and Klemens Böhm. Mutual authorizations: Semantics and integration issues. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, SACMAT '19, Toronto, Canada, 2019. ACM.

[SEGB19c] Gabriela Suntaxi, Achraf El Ghazi, and Klemens Böhm. On mutual authorizations: Semantics, integration issues, and performance. Technical report, Karlsruhe Institute of Technology, 2019.

[SEGB20] Gabriela Suntaxi, Aboubakr Achraf El Ghazi, and Klemens Böhm. Secrecy and performance models for query processing on outsourced graph data. *Distributed and Parallel Databases*, January 2020.

[SL12] Yan Sun and Yuhong Liu. Security of online reputation systems: The evolution of attacks and defenses. *IEEE Signal Processing Magazine*, 29(2):87–97, 2012.

[SMJG05] Basit Shafiq, Ammar Masood, James Joshi, and Arif Ghafoor. A role-based access control policy verification framework for real-time

systems. In *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005.* IEEE, 2005.

[SMSB14] Irina Shklovski, Scott D Mainwaring, Halla Hrund Skúladóttir, and Höskuldur Borgthorsson. Leakiness and creepiness in app space: Perceptions of privacy and mobile app use. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pages 2347–2356. ACM, 2014.

[SNS10] Amril Syalim, Takashi Nishide, and Kouichi Sakurai. Preserving integrity and confidentiality of a directed acyclic graph model of provenance. *Data and Applications Security and Privacy XXIV*, pages 311–318, 2010.

[SS08] Daniel A Schult and P Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, volume 2008, pages 11–16, 2008.

[SS15] Thomas Shortell and Ali Shokoufandeh. Secure signal processing using fully homomorphic encryption. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 93–104. Springer, 2015.

[SU15] Seydi Ahmet Satici and Recep Uysal. Well-being and problematic facebook use. *Computers in Human Behavior*, 49:185–190, 2015.

[SW10] Stefan Saroiu and Alec Wolman. I am a sensor, and i approve this message. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, pages 37–42. ACM, 2010.

[SYY+16] Nan Shen, Jun Yang, Ke Yuan, Chuan Fu, and Chunfu Jia. An efficient and privacy-preserving location sharing mechanism. *Computer Standards & Interfaces*, 44:102–109, 2016.

[SZFJ09] Jie Shi, Hong Zhu, Ge Fu, and Tao Jiang. On the soundness property for sql queries of fine-grained access control in dbmss. In *2009 Eigth IEEE/ACIS International Conference on Computer and Information Science*. IEEE, 2009.

[TLT15] Romuald Thion, François Lesueur, and Meriam Talbi. Tuple-based access control: a provenance-based information flow control for relational data. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015.

[TSBV05] Goce Trajcevski, Peter Scheuermann, Hervé Brönnimann, and Agnès Voisard. Dynamic topological predicates and notifications in moving objects databases. In *Proceedings of the 6th MDM International Conference*, pages 77–85. ACM, 2005.

[VdHKAS+16] Rens W Van der Heijden, Frank Kargl, Osama MF Abu-Sharkh, et al. Enhanced position verification for vanets using subjective logic. In *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, pages 1–7. IEEE, 2016.

[VFJ+10] Sabrina Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Encryption policies for regulating access to outsourced data. *ACM Transactions on Database Systems (TODS)*, 35(2), 2010.

[VLSD+10] Peter Van Liesdonk, Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. Computationally efficient searchable symmetric encryption. In *Workshop on Secure Data Management*, pages 87–100. Springer, 2010.

[WD08] Jieping Wang and Xiaoyong Du. A secure multi-dimensional partition based index in das. In *Asia-Pacific Web Conference*, pages 319–330. Springer, 2008.

[WL06] Hui Wang and Laks VS Lakshmanan. Efficient secure query evaluation over encrypted xml databases. In *Proceedings of the 32nd international conference on Very large data bases*, pages 127–138. VLDB Endowment, 2006.

[WRD+17] Qian Wang, Kui Ren, Minxin Du, Qi Li, and Aziz Mohaisen. Secgdb: Graph encryption for exact shortest distance queries with efficient updates. In *International Conference on Financial Cryptography and Data Security*, pages 79–97. Springer, 2017.

[WSI03] Yodai Watanabe, Junji Shikata, and Hideki Imai. Equivalence between semantic security and indistinguishability against chosen ciphertext attacks. In *International Workshop on Public Key Cryptography*, pages 71–84. Springer, 2003.

[WWY+12] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 109–120. ACM, 2012.

[WXL12] Wei Wei, Fengyuan Xu, and Qun Li. Mobishare: Flexible privacy-preserving location sharing in mobile online social networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 2616–2620. IEEE, 2012.

[XWL13] Runhua Xu, Yang Wang, and Bo Lang. A tree-based cp-abe scheme with hidden policy supporting secure data sharing in cloud computing. In *2013 International Conference on Advanced Cloud and Big Data*, pages 51–57. IEEE, 2013.

[XZTS08] Kefeng Xuan, Geng Zhao, David Taniar, and Bala Srinivasan. Continuous range search query processing in mobile navigation. In *Parallel and Distributed Systems ICPADS'08*, pages 361–368. IEEE, 2008.

[YFY14] Peipei Yi, Zhe Fan, and Shuxiang Yin. Privacy-preserving reachability query services for sparse graphs. In *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*, pages 32–35. IEEE, 2014.

[YPBV14] Xun Yi, Russell Paulet, Elisa Bertino, and Vijay Varadharajan. Practical k nearest neighbor queries with location privacy. In *2014 IEEE 30th ICDE*. IEEE, 2014.

[YT05] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *Web Services, 2005. ICWS 2005*. IEEE, 2005.

[YZZY15] Dingqi Yang, Daqing Zhang, Vincent W Zheng, and Zhiyong Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1), 2015.

[ZB11] Paul A Zandbergen and Sean J Barbeau. Positional accuracy of assisted gps data from high-sensitivity gps-enabled mobile phones. *The Journal of Navigation*, 64(3):381–399, 2011.

[ZH07] Runfang Zhou and Kai Hwang. Powertrust: A robust and scalable reputation system for trusted peer-to-peer computing. *IEEE Transactions on parallel and distributed systems*, 18(4):460–473, 2007.

[ZLLH12] Yao Zheng, Ming Li, Wenjing Lou, and Y Thomas Hou. Sharp: Private proximity test and secure handshake with cheat-proof location tags. In *European Symposium on Research in Computer Security*, pages 361–378. Springer, 2012.

[ZP08] Bin Zhou and Jian Pei. Preserving privacy in social networks against neighborhood attacks. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 506–515. IEEE, 2008.

[ZSW+14] Yingguang Zhang, Sen Su, Yulong Wang, Weifeng Chen, and Fangchun Yang. Privacy-assured substructure similarity query over encrypted graph-structured data in cloud. *Security and Communication Networks*, 7(11):1933–1944, 2014.