PRIVACY-PRESERVING COLLABORATION IN AN INTEGRATED SOCIAL ENVIRONMENT

A Dissertation presented to
the Faculty of the Graduate School
at the University of Missouri

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Nitish Milind Uplavikar
Dr. Wei Jiang, Dissertation Supervisor

December, 2021

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

PRIVACY-PRESERVING COLLABORATION
IN AN INTEGRATED SOCIAL ENVIRONMENT

presented by Nitish Milind Uplavikar,

a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Wei Jiang

Dr. Dan Lin

Dr. Chi-Ren Shyu

Dr. Shih-Kang Chao

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Wei Jiang, for the thoughtful comments and recommendations on this dissertation. He has been a great and constant source of inspiration, counsel, encouragement, and support. I am fortunate to have such an unparalleled, considerate advisor like him and am grateful for all that he has done for me. My words seem insufficient to express his entire contribution in this journey. Additionally, I am thankful for the time, expertise, and comments of all the members of the committee who provided useful feedback and recommendations, Dr. Lin, Dr. Shyu, and Dr. Chao. I am also thankful for the University of Missouri, the College of Engineering, and the EECS Department in particular, and all the faculty and staff for all their considerate guidance and help navigating the entire process. I would also like to thank my family and friends for all their unconditional support.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

ABSTRACT

Privacy and security of data have been a critical concern at the state, organization and individual levels since times immemorial. New and innovative methods for data storage, retrieval and analysis have given rise to greater challenges on these fronts. Online social networks (OSNs) are at the forefront of individual privacy concerns due to their ubiquity, popularity and possession of a large collection of users' personal data. These OSNs use recommender systems along with their integration partners (IPs) for offering an enriching user experience and growth. However, the recommender systems provided by these OSNs inadvertently leak private user information.

In this work, we develop solutions targeted at addressing existing, real-world privacy issues for recommender systems that are deployed across multiple OSNs. Specifically, we identify the various ways through which privacy leaks can occur in a friend recommendation system (FRS), and propose a comprehensive solution that integrates both Differential Privacy and Secure Multi-Party Computation (MPC) to provide a holistic privacy guarantee. We model a privacy-preserving similarity computation framework and library named Lucene-P^2. It includes the efficient privacy-preserving Latent Semantic Indexing (LSI) extension. OSNs can use the Lucene-P^2 framework to evaluate similarity scores for their private inputs without sharing them. Security proofs are provided under semi-honest and malicious adversary models.

We analyze the computation and communication complexities of the protocols proposed and empirically test them on real-world datasets. These solutions provide functional efficiency and data utility for practical applications to an extent.

Chapter 1

Introduction

Online Social Networks (OSNs) are ubiquitous today, and have gone far beyond their original intent and impacted human life across multiple spheres such as enabling knowledge creation [1], sociopolitical movements [2], health management [3,4], etc. Following the original OSNs that enable users to connect with old or new friends/acquaintances, there are other emerging and more specialized OSNs that cater to certain demand or enable specific services for the users, such as, Spotify (Music), Instagram (Photos), Fitbit (Fitness), and so on.

Depending on the function and the type of an OSN, personalised recommendations can be made to its users regarding new friends, groups, events, goods for purchase, etc. OSNs use friend recommender systems (FRS) to increase their user base and enrich online user interaction. In order to prvide more accurate recommendations, the existing FRS also leverages information from other OSNs by forming integration partners (IP) [5]. However, these partnerships have led to many security and privacy issues. For example, Facebook's (FB) partners namely Microsoft's Bing can see virtually all FB users' friends without consent, and Netflix and Spotify were able to read FB user's private messages [6, 7].

Consider the example of Spotify which allows its users to connect or create an account using the individual's Facebook account. The users can listen to songs and create their own contents, they can also follow their friends' collection or some artist. As a result, a small network can be formed by Spotify users. However, Spotify cannot boast of a rich, stable and robust network that Facebook has of its users, that is created over a duration of almost a decade. Reaching that stage of network structure would take some time and not allow Spotify the ability to provide more accurate recommendations, of either users or songs, to its users. One solution to this problem is that Spotify can, in collaboration with Facebook, provide diverse yet relevant recommendations to its users. Here, Spotify (the client) and Facebook (the server) would be in an integration partnership with the server providing recommendation services, based on its social network data, to the client. While this has benefits, there are numerous privacy/security risks:

1. *Leaking the server's social network data to the client*: The client can recreate the server's owned network to a certain extent by repeatedly running queries for the same users. Using a mutual friends based FRS, the client can repeatedly execute such type of query to infer one friendship at a time. Eventually, the client may be able to construct the original graph or sub-graph of friendship that is owned by the server.

2. *Leaking client's user information*: The server knows which users in its network are subscribing to the services provided by the client. Users subscribing to client's services might want to keep this membership information private from the server. On the other hand, the client may want to keep its users' information private.

3. *Leaking client's network data to the server*: The server can recreate the network on the client by analyzing the recommendation scores and patterns of users queried. It can assume that the user always picks the top one or two

recommended users and create a sub-graph of the possible friendship graph for a user that would be created on the client.

1.1 Overview

In this section, we cover the problem addressed in this work. Initially we define the problem and then describe the adversary model in which the protocol is proposed.

1.1.1 Problem Definition

In this work, we propose privacy-preserving friend recommendation (PPFR) protocols in the aforementioned integration partnership environment to protect the private network data at both the server S and the client C. In practice, the server in our application domain is a well-established social network like Facebook, and the client is a specialized social network who wants to provide friend recommendations to its own users based on the social network data from the server. The network datasets are modeled as graphs and denoted by G_S and G_C at the server and the client respectively.

Without any security guarantee, a mutual friend based FR generally works as follows: given a user u at the client C, C selects a set of users $\{v_1, \ldots, v_m\}$ who may potentially become friends with u. C issues a query in the form of m pairs of user-ids, $(u, v_1), \ldots, (u, v_m)$, to the server S. For each pair (u, v_i), S performs the necessary computation to obtain the mutual friends count $s_i \leftarrow m_{G_S}(u, v_i)$ which will be sent to C. The client can then make use of the scores to recommend some users (e.g., top k or above a predefined threshold) in $v_1, \ldots, v_m$ to u. Thus, the FR functionality is defined as follows between a server S and a client C:

$$\mathrm{FR}(\langle S, G_S\rangle, \langle C, (u, v_1), \ldots, (u, v_m)\rangle) \rightarrow \langle C, s_1, \ldots, s_m\rangle \tag{1.1}$$

1.1.2 Adversary Model and Protocol Overview

We assume the server and the client are semi-honest [8]. That is, the participants follow the prescribed steps of the protocol, and then try to learn additional information. In addition, we assume the parties are computationally bounded. For a privacy-preserving friend recommendation (PPFR) protocol in the integration partnership environment, the following information needs to be protected.

- G_S: The server's network data that needs to be protected from the client. Since information regarding G_S can be inferred from the mutual friends counts, this implies that $s_1, \ldots, s_m$ should be protected.
- G_C: Information regarding the client's network data should not be disclosed to the server. This implies that the server should not know the user ids, e.g., $(u, v_1), \ldots, (u, v_m)$ used in the recommendation.

In order to prevent the server from knowing $(u, v_1), \ldots, (u, v_m)$ and simultaneously derives $s_1, \ldots, s_m$, one natural choice is to adopt Secure Multiparty Computation (SMC) techniques [8] to implement the FR functionality given in Equation 1.1. However, SMC alone is not sufficient. As discussed previously, the similarity scores can leak information regarding the server's network data G_S.

What if we modify the FR functionality? Instead of returning the similarity scores, the server can return (1) the top k most similar users where $1 \leq k \leq m$, or (2) all the users whose similarity scores are above a given threshold. While both options seemingly leak less information about G_S, it is hard to actually quantify the exact degree of information leakage. Instead, we adopt Differential Privacy (DP) [9–11] as a formal and quantifiable model to control information leakage from the similarity scores. Specially, only differentially private similarity scores are returned to the client. In summary, the key novelty of our proposed PPFR protocol is to intelligently combine two dominant privacy-preserving tools to prevent information leaking during the friend recommendation process.

1.2 Contribution

We contribute two major solutions here. The first solution addresses the real-world privacy issues in friend recommendation within an integrated collaborative environment. In this case, an integrated environment consists of two or more organisations, e.g. OSNs, that want to provide a cross-platform service (recommendation) while leveraging their individual private inputs and resources in a privacy-preserving way.

Similarly, the second solution, named Lucene-P^2, addresses the privacy issues in distributed text-based search for documents. Lucene-P^2 allows two parties to compute secure similarity amongst their private input documents without actually sharing them. However, the application of Lucene-P^2 is not just restricted to document entities but it can be used to compute privacy-preserving similarity scores for any text-based entities. Thus, it enables privacy-preserving entity-based friend recommendation e.g. two OSNs may obtain similarity scores for their private users without sharing their user information with each other to perform friend recommendations.

Given below is a list of contributions with regards to the privacy-preserving (friend) recommendation in integrated environments and privacy-preserving (document, entity or friend) recommendation in distributed environment.

1.2.1 Privacy-Preserving Friend Recommendation based on Common Friends

Following are our contributions by way of the PPFR protocol.

1. We provide the Privacy-preserving Friend Recommendation (PPFR) protocol that integrates both Secure Multi-party Computation (SMC) and Differential

Privacy (DP) to provide a holistic privacy guarantee. More precisely, we guarantee privacy of G_C depending on the underlying homomorphic encryption used, i.e. the Paillier Cryptosystem [12], and G_S to an extent (ϵ) provided by ϵ-DP.

2. We provide the security analysis for PPFR in the semi-honest adversary model.

3. We present time and communication complexities for the PPFR protocol.

4. We empirically test the solution on the DBLP real-world dataset. Performance is reported along with utility evaluation e.g. Kendall's τ, Spearman correlation coefficient ρ, precision, recall etc.

1.2.2 Privacy-Preserving Friend Recommendation based on Similar Profiles

The filtering technique of this solution is analogous to the content-based filtering technique used in recommender systems. As mentioned earlier, the privacy-preserving entity-based recommendation could be applied for any entity e.g. documents, products, users etc. In this work, we provide Lucene-P^2, which has been developed for document entities. Lucene-P^2 can be easily modified for enabling Privacy-preserving Friend Recommendation by having the documents contain user profile information or by constructing the internal vectors from the OSN user profiles.

Our contributions are as follows:

Objective and Contribution

A system overview of Lucene-P^2 is given in Figure 1.1 where Alice, with a query document, wants to retrieve relevant information from Bob's collection.

- Input representation: Initially, both Alice and Bob's inputs will be processed by the data representation component whose outputs become the inputs to the next component.

- Secure correlation computation: This component returns the IDs of documents in Bob's collection most relevant to Alice's query. During the process, no other information regarding the private inputs of Alice and Bob is disclosed to the other party.
- Information sharing: Once Bob knows which documents are correlated to Alice's query, the actual documents that satisfy predetermined access control policies will be shared with Alice.

Apache Lucene forms a significant core of other Apache projects, such as Apache Solr, Elasticsearch, DocFetcher, Swiftype, etc. All of these are widely used in enterprise and research environments. Hence, addressing the privacy-preserving side of similarity computation in Lucene has a broad and realistic impact, e.g., Cloudera, Hortonworks, and MapR big data solutions, Human Metabolome Database interfaces, etc.

As the name implies, the framework utilizes Lucene to produce an inverted index for a document collection and represents each document within the vector space model. Although Lucene provides an approach to compute information relevancy or correlation, when used directly, Alice must disclose her private query document to Bob and vice versa. Therefore, one of the key technical contributions of our model is the introduction of secure protocols to compute information correlation without disclosing any private (and irrelevant) information.

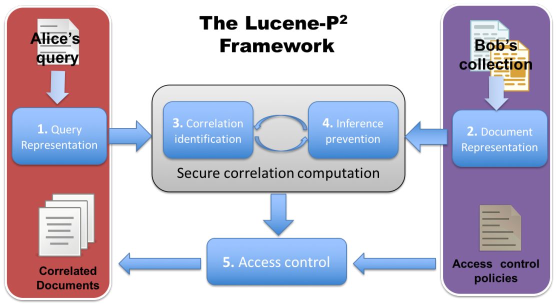

Figure 1.1: System Overview

The main objective of this work is to develop the core techniques required by the Lucene-P^2 framework to control disclosure without relying upon a trusted third party. Since un- or semi-structured information (e.g., text-based documents) are ubiquitous in many organizations and on the Internet, we focus on supporting the standard user query-document response model. The access control component of the framework can adopt the existing access control methodologies. As a result, this work only focuses on the secure correlation computation (SCC) module. A basic version of an SCC protocol can be defined as follows:

$$\text{SCC}(\langle P_a, q\rangle, \langle P_b, D\rangle) \rightarrow s_1, \ldots, s_n \tag{1.2}$$

where P_a and P_b denote Alice and Bob respectively. Each pair of angular brackets on the left of Equation 1.2 indicates the private input of the corresponding party. The final result is a set of n similarity scores $s_1, \ldots, s_n$ based on the scoring function adopted in Lucene (details are given in Section 2.2). The proposed SCC protocols are computationally secure under the semi-honest model defined in the literature of secure multi-party computation [13,14], and we will discuss its limitations and potential extensions under the malicious adversary model. In addition, our design goal is to minimize the number of servers needed and the interactions or round complexity between the client and the servers. The main contributions of this work are summarized below:

- To model and develop a privacy-preserving framework and library based on the non-secure similarity computation framework offered by Apache Lucene, a widely used Information Retrieval (IR) library.
- As part of the proposed framework, we have developed various Multi-Party Computation protocols based on either additive homomorphic cryptosystems or secret sharing schemes.

- We apply the dimension reduction technique, Latent Semantic Indexing (LSI), to the SCC protocol, called SCC-LSI, evaluate and compare the performance improvement on a real dataset.

- We provide the security and complexity analysis of these techniques, and examine the implications of the proposed protocols under the semi-honest and malicious adversary models.

Chapter 2

Related Work and Background

A large amount of interesting work has been done in the area of Recommender Systems (RSs), in general, as well as within the field of Privacy-preserving Data Analytics. Due to the sensitive nature of the data processed by these RSs, there is a high requirement for some privacy-preserving RS solutions. In this chapter, we begin by covering the background details of concepts used within this work and a few related references.

2.1 Related Work

In this section, we see a few related works in privacy-preserving friend recommendation and privacy-preserving document-similarity computation. It is interesting to note that our second work, i.e. the privacy-preserving document-similarity computation, involves secure similarity computation amongst two documents belonging to two different parties. This approach can be applied as content-based friend recommendation. This content-based friend recommendation is achievable by representing the user profiles as vectors in Euclidean space.

2.1.1 Private Friend Recommendation

As specified earlier, Differential Privacy guarantees an individual's privacy while enabling the analysis of the differentially private data. In this section, we consider other similar works in the areas of differential privacy (DP), multi-party computing (MPC) and online social networks (OSN).

Due to the promise of providing user-level privacy, DP has been widely used in Online Social Networks. Primarily, within OSNs, DP is used for publishing the network data in the form of histograms [15–19]. Our work does not use DP for publishing puposes but to enable interactive cross-OSN collaboration for recommendation purposes. The aforementioned DP histogram publication solutions cannot capture large-scale and frequent changes within the network that could be possible in our method.

Apart from using DP for publishing network histograms, considerable amount of work focusses on publishing non-histogram data which prevents node re-identification or edge disclosure. Generally, these techniques are termed as *node-DP* [19] and *edge-DP* [20], respectively. Node-DP protects the presence of a user within the dataset, whereas edge-DP protects the presence of a relationship. In Friend-Rec-P^2, we are protecting existing friendships while recommending friends. *k-edge* privacy is a general form of *node-DP* ($k \leftarrow (n-1)$) and *edge-DP* ($k \leftarrow 1$) for n nodes. [21] propose a trust-based friend recommendation system in OSNs using multi-hop trust chain based on user attributes. Their approach uses kNN for co-ordinate matching and creating a trust network. [22] computes recommender results by aggregating multi-hop trust chain utilities in a privacy-preserving way. Trust-based networks are difficult to quantify and track with time. [23] provide two algorithms based on additive homomorphic encryption scheme and anonymous message routing. The solution maintains the users' friend list private, which is similar to our approach, however, their friend recommendation is accurate and may leak more information than required. All the solutions are developed for a single OSN as against our solution for two integration partner

OSNs. Other private friend recommendation techniques involve using anonymization techniques e.g. [24] propose segmentation tree approach over hypergraph model of graph.

2.1.2 Privacy-Preserving Text-based Search

Existing approaches to similar document detection, without considering privacy protection, have been developed to identify correlated contents. The idea of detecting similar documents was introduced in [25]. Given a document, a set of (non-crypto) hashes are computed based on all possible substrings of a certain length. A subset of these hashes is then applied to represent each document as fingerprints. Two documents are deemed to be similar if the number of fingerprints in common is greater than some pre-defined threshold. There has been a variety of research into document fingerprints [26–34]. While hashing appears to have some inherent privacy-preserving properties (because the text is not revealed in the clear), in practice, any overlapping text (i.e., matching hashes that corresponding to words, n-grams, or sentences) are revealed. This results in the disclosure of an unknown amount of text. It leads to protocols that are less secure than what we introduce in this work.

There are several alternative approaches which merit attention. First, ranking methods from the information retrieval literature provide an alternative strategy to detect similar documents [35, 36]. In this setting, a vector space model is relied upon for detecting similar documents. By contrast, there are instance-based clustering techniques have been suggested [37]. Still, all of these approaches assume that information is accessible to at least the service providers. Therefore, they are not applicable to our problem.

Secure Correlation Detection Approaches

The problem, as well as solutions, to secure similar document detection (SSDD) [38–40] is closely related to our work. The SSDD problem was first introduced in [40], where a secure protocol was proposed to compute the cosine similarity between two documents that were independently owned by two parties without leaking their actual contents. The approach offered in [39] utilized a local clustering technique to reduce the computation complexity of the solution. The research in [38] adopted a different document representation model and developed a secure protocol to compute the Jaccard coefficient between two documents. Our approach is different from the set of existing techniques. First, our approach provides for a substantially more advanced similarity function to identify correlated information, which is commonly used in the existing information retrieval (IR) toolkit. Secondly, we introduce a programming library that can be combined with Lucene. In doing so, we provide users with the flexibility to implement their own privacy-preserving IR applications.

A more recent paper [41] investigated the problem of privacy-preserving plagiarism detection. Although the application is similar, the approach to the solution is very different. For a start, it requires three entities to execute - root server, querier and document submitter. It runs in two phases: in the first phase, document sources are obtained, while in second phase, the sentences are aligned with the sources, if they are similar. Although the second phase is similar in nature to our work, it is too computationally expensive as sentences are represented by tf-idf (Term Frequency in sentence/snippet-Inverse Snippet Frequency) and then dot product is computed, for every pair of snippets/sentences belonging to each similar candidate document sources obtained in the first phase, the same way as what we perform it using secure dot product techniques given in [39, 40]. However, we do it between two parties and have tf-idf at document level which can dramatically reduce the number of pair-wise comparisons. Their scoring function is not commonly adopted in IR tasks. The approach also needs to disclose certain summary information of each document to the cloud server. As a result, the solutions presented in this work are more secure and practical.

In [42], the authors proposed a new framework, based on topic disclosure model and an algorithm that suppresses the user intention for a search query. The decoy terms, using ghost queries, are added to protect user's query privacy. The paper [43] addressed the problem of search privacy in the Internet using a statistical approach. The authors model the problem of query scrambling into a set covering problem. In [44], the authors proposed a browser extension that can semantically dissociate queries in real time. It is basically a proxy between the user and search engine. For dissociation purposes, the authors use the concept of Personalized Query Classification. It is possible that such an extension can violate terms and conditions of search services and dissociated profiles can be easily identified as those would be different than average non-dissociated profiles. In summary, all these techniques are based on information obfuscation and do not achieve the same secure guarantee as our approach. Also, these techniques are not applicable for securely computing Lucene similarity metrics.

Private Keyword Search, Private Information Retrieval, and Private Statistics Evaluation

Private key word search techniques over encrypted data [45–52] focus on the situation where an entity outsources its data, in encrypted form, to a remote server. The server, in turn, performs the computation necessary to return only the encrypted data that contain some specific keywords in a user query. Under our proposed problem domain, first of all, the data are not encrypted. Secondly, keyword matching is only one way to measure similarity. When data are not encrypted, more efficient privacy-preserving protocols can be designed to compute advanced similarity measures (e.g., the one invoked in Lucene). The same arguments can be applied to order-preserving or searchable encryption schemes (e.g., [53–55]).

It should be recognized that private information retrieval (PIR) [56–63] is also related to the work addressed in this work. However, most existing techniques only protect the search (or user query) from disclosure and provide no control to the server. The information being searched is, in theory, completely open. In [63], information is protected in both ways; however, the retrieved information cannot be relied upon to compute more advanced similarity measures (e.g., cosine similarity and KL-divergence) that are necessary for IR applications to identify the relevant information. In SPIN [64], a user can query de-identified medical records. Yet we cannot adopt this system because it is specialized to the medical domain and the user queries are not protected from the system operator.

Private statistics evaluation [65] focuses on the task of securely evaluating a function on selected data points which can be reduced to secure evaluation of a multivariate polynomial. There are no straightforward ways to apply these protocols to solve the proposed problem, especially when our goal is to minimize the number of servers and the round complexity.

2.2 Background

We classify the background technical details into security and privacy related ones and information retrieval-based ones. We cover the main relevant topics below.

2.2.1 Data Privacy and Security Primitives

We provide the key technical background and threat models that are needed to develop the proposed protocols. We then discuss friend recommendation (FR) based on the criteria of number of mutual friends and provide a brief overview of differential privacy (DP) [66], its application within our work in order to enhance the privacy.

The Threat and Adversary Model

Regarding a distributed protocol, security is generally related to the amount of information leaked during the protocol execution. To maximize privacy protection, the ideal solution is to utilize a trusted third party (TTP) to perform all the required computations. However, it is not realistic to assume the existence of such a TTP. As a result, we adopt the fundamental methodologies in the literature of secure multiparty computation (SMC) [13]. Briefly speaking, the goal of SMC is to develop a secure protocol that provides the same security guarantee as the TTP model.

In order to prove the security guarantee of an SMC protocol, we need to clarify the adversary model assumed in the design of the proposed secure protocols. An adversarial model generally specifies what an adversary or attacker is allowed to do during an execution of a secure protocol. Under SMC, there are three adversary models [14, 67]: semi-honest, malicious and covert. The semi-honest model assumes that the participating parties follow the protocol, but they are allowed to compute any other information based on their own inputs, outputs and the messages received during the execution of an SMC protocol. Under the malicious model, the participating parties can diverge arbitrarily from the prescribed computation, e.g., a malicious party can distort his or her local computation. Under the covert model, malicious behavior may not be preventable during protocol execution, but can be detected later.

In this work, we initially assume the participating parties are semi-honest. Developing protocols under the semi-honest setting is an important first step towards constructing protocols with stronger security guarantees. Subsequently, we will show how to extend our protocols to satisfy the malicious adversary model. In addition, at most one party can be malicious in a two-party SMC protocol. If both servers were malicious, computations would be out of control, and there is nothing we can do about it. Thus, in SMC, two-party secure protocols always assume that there is no collusion and at least one of the parties is semi-honest. The following definition captures the properties of the secure protocol under a semi-honest adversary model [14].

Definition 1. *Let a_i be the input of party P_i, $\Pi_i(\pi)$ be P_i's execution image of the protocol π and b_i be the output for party P_i computed from π. Then, π is secure if $\Pi_i(\pi)$ can be simulated from a_i and b_i such that distribution of the simulated image is computationally indistinguishable from $\Pi_i(\pi)$.*

An execution image generally includes the inputs, the outputs and the messages communicated during protocol execution. To prove a protocol is secure under the semi-honest model, we generally need to show that the execution image of a protocol does not leak any information regarding the private inputs of participating parties [14].

Differential Privacy (DP)

Differential Privacy (DP) [66] is a privacy model that provides a formal mathematical bound on the increase in privacy risk. This privacy risk could be for any person whose data is being used within some computation. An algorithm can be considered to be differentially private if for all possible outputs, the likelihood of getting any output does not significantly vary based on the inclusion or exclusion of any single person's data. Differential Privacy i.e. ϵ-DP could be formally defined as provided below:

For $\epsilon > 0$, a mechanism (viz algorithm) $T(\cdot)$ with domain G is ϵ-differentially private if for every $x, y \in G$ such that $\|x - y\|_1 \leq 1$ and every $\Lambda \subseteq \text{Range}(T)$,

$$Pr\left[T(x) \in \Lambda\right] \leq e^{\epsilon} Pr\left[T(y) \in \Lambda\right]$$

While there are many general ways to achieve Differential Privacy, here we consider the simple Laplace Mechanism [68]. In this Laplace mechanism, a suitably scaled noise is added to the output. More specifically, the global sensitivity S_m of the function m

is computed as the maximum change in the output for any two neighboring inputs. Finally, the mechanism involves adding noise, proportional to S_m/ϵ, to the output.

Additive Homomorphic Encryption

The proposed protocols adopt an additive homomorphic encryption (HEnc) scheme as the building block. Let E_{pk} and D_{pr} be the encryption and decryption functions in an HEnc scheme with public key pk and private key pr. Without pr, no one can discover x from $E_{pk}(x)$ in polynomial time. An HEnc has the following properties:

- The encryption function is additively homomorphic:
  $E_{pu}(x_1 + x_2) = E_{pu}(x_1) \times E_{pu}(x_2)$.
- Given a constant c and $E_{pu}(x)$, $E_{pu}(c \cdot x) = E_{pu}(x)^c$.
- The encryption function is probabilistic and semantically secure [69]. In other words, two encryptions of x differ with very high probability.

Any homomorphic encryption system is applicable, however, in this work we adopt the Paillier encryption scheme [70] for its efficiency and implementation simplicity.

2.2.2 Information Retrieval Primitives

Here, we provide a few brief details of the information retrieval topics relevant to our research. These topics include the prominent filtering approaches of recommender systems, a simple friend recommendation criteria that is used in online social networks. Similarly, we cover a similarity metric used for recommendation etc., within Apache Lucene (Core), a widely-used and open-source information retrieval library.

Filtering Approaches in Recommender Systems

A wide variety of considerations have to be made while proposing a Recommender System (RS). Some of the prominent factors involve type of data, filtering algorithm, model used, techniques, scalability required, objective and quality of recommendations required, etc. [71]. Prominent amongst these is the filtering algorithm employed. Content-based filtering and collaborative filtering are the major filtering algorithms apart from the demographic and hybrid ones. A short description of these follows.

1. *Content-based Filtering:* In this method, similarity is computed between existing entities and entities to be recommended. In this approach, information about the entities should be known beforehand. For example, a user is recommended an item that is similar to previous items that the user had purchased.

2. *Collaborative Filtering:* In this method, a set of users provide entities with scores. Items ranked by users similar to the user being recommended are filtered

Our proposed solutions can be categorized into these two filtering methods.

Friend Recommendation based on Mutual Friends

The topic of FR is widely addressed [72, 73]. One of the most common features in a social network to use for FR is related to the number of common friends between two users. This feature assumes that two people are more likely to be *friends* if they have some common friends amongst themselves. The similarity score using mutual friends can be computed based on set intersection. Given two users u and v, let l_u and l_v denote the friend lists of u and v respectively. Then, their similarity is computed as:

$$\text{Similarity}(u, v) = |l_u \cap l_v| \tag{2.1}$$

The Similarity Metric in Lucene

As stated in the previous sections, the main challenge is to securely identify the correlated information. Since Lucene is a popular information retrieval tool, we will adopt

Table 2.1: Legend of Common Notations

Symbol	Definition
D	A document collection
$q, \vec{q}$	A query document and its vector form
$d, \vec{d}$	A document from D and its vector form
m	Size of the global vector space
t	A term in a document
n	$n = \|D\|$ (document collection size)
SCC	Secure Correlation Computation
tf-idf	Term and inverse document frequencies

its similarity metrics to identify correlated information. In the later sections, we will propose a simplified version that can be evaluated more efficiently without affecting the effectiveness in our problem domain. The notations adopted are presented in Table 2.1, and the similarity metric in Lucene is defined by the following equations:

$$\text{score}(q, d) = \text{coord}(q, d) \cdot \text{queryNorm}(q) \cdot \sum_{t \in q} \text{weight}(t, d) \tag{2.2}$$

$$\text{weight}(t, d) = \text{tf}(t, d) \cdot \text{idf}(t)^2 \cdot \text{getBoost}(t) \cdot \text{norm}(t, d) \tag{2.3}$$

$$\text{norm}(t, d) = \text{lengthNorm}(d) \cdot \prod_{\text{field } f \text{ in } d \text{ named } t} f.\text{boost}() \tag{2.4}$$

where q denotes a query document and d belongs to a document collection. The summaries of the sub-functions used in the above equations are given below, in an item-wise manner:

- f.boost() - get field f's, index-time specified, boost value:

  Boost real values indicate the importance of a field within a document. Highly relevant fields, with respect to indexing and thereby searching, are comparatively awarded higher values with respect to others. If there are multiple fields with the same name, say t, then a product of all of their individual boost values is computed and returned.

- lengthNorm(d) - field length norm:

  As shown in Equation 2.4, the norm is calculated based on the number of tokens in a field within a document. This ensures parity among short and long fields in a document for computing scores. The fields chosen must contain the term t.

- norm(t,d) - document normalization factor:

  Each document has a variable number of fields. Such fields are of varying sizes and user-defined. These fields have a field name (title, abstract, etc.) and associated information (i.e. boost values). In this case, the normalization factor (see equation 2.4) is computed as a product of field f's boost values and the document length norm.

- getBoost(t) - query term boost:

  Users can mark and indicate a term t's importance to the query q during query creation. This can be done by providing boost values for term t. This function, i.e. getBoost(t), returns this preset boosting value for term t. The default value is 1.

- idf(t) - inverse document frequency; expression $= 1 + \log\left[\frac{|D|}{\mathrm{df}(t)+1}\right]$:

  The idf(t) function returns the inverse document frequency of any term t in the collection. Document frequency (df) is the number of documents the term appears in. High idf() score for a term indicates that it is a greater representative of documents as compared to the terms with low score. Low idf() score terms occur in a relatively high number of documents, thereby qualifying those documents to only a small extent.

- tf(t, d) - term frequency:

  tf(t, d) provides the square root of the term frequency of the term t in document d.

- queryNorm(q) - query normalizing factor; expression $= \left[\sqrt{\sum_{t\in q}(\mathrm{idf}(t))^2}\right]^{-1}$:

  This returns a normalization factor for a query document q. This ensures that query with frequently-occurring terms is evaluated on par with the one having comparatively less frequently-occurring terms. The normalization factor is constant for all the documents d that are compared against the input query document q, as a result, it does not affect the order of recommendation ranking for these documents, if omitted.

- coord(q, d) - common terms; expression $= \frac{|q\cap d|}{|q|}$:

  Captures relatedness i.e. the ratio of q's terms present within the given document d.

Chapter 3

Privacy-Preserving Friend Recommendation based on Common Friends

Ubiquitous Online Social Networks (OSN)s play a vital role in information creation, propagation and consumption. Given the recent multiplicity of OSNs with specially accumulated knowledge, integration partnerships are formed (without regard to privacy) to provide an enriched, integrated and personalized social experience. However, given the increasing privacy concerns and threats, it is important to develop methods that can provide collaborative capabilities while preserving user privacy. In this chapter, we focus on friend recommendation systems (FRS) for such partnered OSNs. We identify the various ways through which privacy leaks can occur, and propose a comprehensive solution that integrates both Differential Privacy and Secure Multi-Party Computation to provide a holistic privacy guarantee. We analyze the security of the proposed approach and evaluate the proposed solution in terms of both utility and computational complexity by experimenting with DBLP real dataset.

3.1 The Proposed Protocol

In this section, we discuss the details of our proposed PPFR protocol. Under Secure Multiparty Computation (SMC), there are several ways to implement a secure protocol, e.g., additive homomorphic encryption (HEnc), secret sharing (SS), oblivious transfer (OT), garble circuits(GC), and fully homomorphic encryption (FHE). Approaches based on OT, GC and FHE requires the FR functionality being represented as a Boolean or an arithmetic circuit. When the dataset at the server is very large, the size of the circuit can become intractable. In addition, the social network data G_S keep changing all the time, the circuit has to be reconstructed every time a change is made to G_S. SS based approach requires at least three independent parties. HEnc provides a good balance among these factors. Therefore, the proposed protocol utilizes an HEnc scheme, such as Paillier [12]. More importantly, our protocol design provides a novel way to securely compute the similarity score based on mutual friends which can be implemented with any specific aforementioned SMC approach.

3.1.1 Protocol Initialization

The algorithm and sub-algorithms are dependent upon an encryption system (HEnc) which exhibits homomorphic additive properties. We use the Paillier Cryptosystem [12] for this purpose, with pu being the public key and pr, the private key, known by the client only. Under DF, ϵ guarantees an upper bound on the difference of the output distributions from any two input datasets that differ by a single record. The lower the value of ϵ, the more similar the two distributions would be and the more similar the two distributions are, the more difficult it would be to predict which dataset out of the two was the actual input. The parties running the PPFR protocol should agree upon this value prior to the execution. From ϵ, we can derive λ, the scaling parameter of Laplace distribution from which the noise will be generated. Note that the global sensitivity of the similarity is 1 since at most the number of common users can increase (or decrease) by 1. Therefore λ set to $1/\epsilon$ is sufficient to ensure ϵ-differential privacy.

3.1.2 The Main Protocol

The key steps of the proposed PPFR protocol are given in Algorithm 1. The private input from the server is its network data G_S, represented as an adjacency list L: user u_i's friend list is denoted by L_i. The private input from the client is a pair of users (u_a, u_b), and the protocol returns a similarity score between the two users based on G_S. To achieve the functionality given in Equation 1.1, the protocol can be run in parallel with each of m user pairs: $(u, v_1), \ldots, (u, v_m)$, and m similarity scores are returned to the client at the end. We assume that there is a mapping from a user id to a value in $\mathbb{Z}_N^*$. All notations in Algorithm 1 that represent users are in $\mathbb{Z}_N^*$. Step-by-step explanations of the protocol are given below, and the index i is in $\{1, \ldots, n\}$ where n is the number of users at the server.

- Step 1: The client encrypts the users ids. Instead of u_a and u_b, the client encrypts $-u_a$ and $-u_b$ which is equivalent to $N - u_a$ and $N - u_b$. At the end of the step, the encrypted inverses of the user ids, $E_{pu}(-u_a)$ and $E_{pu}(-u_b)$, are sent to the server.

- Step 2: Γ_i is an encryption of user id u_i, and $\hat{\Gamma}_i$ is a random permutation of Γ_i to hide the relative positions of u_a and u_b in G_S. In practice, it seems this permutation is not necessary; however, it is needed to formally prove the security of the protocol. η_i is equal to 0 if $u_i = u_a$ in the permuted user list; otherwise, it is a random value chosen from $\mathbb{Z}_N^*$. θ_i is defined similarly, except that it is equal to 0 if $u_i = u_b$. Since the computations are performed based on the encrypted u_a and u_b, the server does not know which u_i corresponds to u_a or u_b. In other words, the server does not know the similarity score is computed for which two users.

- Step 3: The client decrypts $E_{pu}(\eta_i)$ and $E_{pu}(\theta_i)$, and construct n encrypted values, each of which is denoted by $E_{pu}(k_i)$, where k_i is equal to 1 if either η_i or θ_i is 0. Alternatively, we can interpret that k_i is equal to 1 if either $u_a = u_i$ or $u_b = u_i$. This implies that there are only two k_is equal to 1, and the rest of k_i values are 0s.

- Step 4: For each friend list L_i, the server computes $E_{pu}(\omega_i)$: each user u_j in L_i is replaced with $E_{pu}(k_j)$ or the j^{th} encrypted value received from the client. Then these encrypted values are multiplied together to get $E_{pu}(\omega_i)$. Based on the k_j values, ω_i can only have three distinct values:

- $\omega_i = 0$: indicating that u_i is not a friend of u_a or u_b.
- $\omega_i = 1$: indicating that u_i is a friend of u_a or u_b, but not both.
- $\omega_i = 2$: indicating that u_i is a friend of both u_a and u_b.

Suppose that we can derive a value ω'_i such that $\omega'_i = 0$ if $\omega_i = 1$ or $\omega_i = 0$, and $\omega'_i = 1$ if $\omega_i = 2$. Then $\sum_{i=1}^{m} \omega'_i$ is the mutual friend count of u_a and u_b. To derive ω'_i, we use the following equation:

$$\omega'_i = \frac{\omega_i(\omega_i - 1)}{2} \tag{3.1}$$

The goal of Steps 5 and 6 is to derive $E_{pu}(\omega'_i)$ from $E(\omega_i)$. Since ω_i can leak information about G_S, the server randomizes it, by adding a random value, to produce $E_{pu}(\omega_i + r_i)$.

- Step 5: Decrypting $E_{pu}(\omega_i + r_i)$, the client obtains $\omega_i + r_i$. Then the client computes $(\omega_i + r_i)(\omega_i - 1 + r_i)$, and sends the encrypted result to the server.
- Step 6: Since the server knows r_i and $E_{pu}(\omega_i)$, the server can obtain

$$E_{pu}(-2r_i\omega_i + r_i - r_i^2)$$

  Multiplying it with $E_{pu}((\omega_i + r_i)(\omega_i - 1 + r_i))$, the server obtains $E_{pu}(\omega_i(\omega_i - 1))$. Because h is the multiplicative inverse of 2 in $\mathbb{Z}_N^*$, the sub-step (c) produces the encryption of ω'_i. Based on how ω'_i is derived, we know that ρ is the mutual friend count. The sub-step (f) randomizes the actual score by adding a noise generated from a Laplace distribution.
- Step 7: The client decrypts the encrypted score to get the differentially private similarity score for u_a and u_b.

Algorithm 1 PPFR($\langle \text{Server}, G_S \rangle, \langle \text{Client}, u_a, u_b \rangle) \rightarrow \langle \text{Client}, s \rangle$

```
Require: $h \leftarrow 2^{-1} \mod N$, and the index $i$ varies from 1 to $n$ where $n$ denotes the number of users at the server
1: Client:
    (a) Compute $E_{pu}(-u_a)$ and $E_{pu}(-u_b)$
    (b) Send both to the server
2: Server:
    (a) Receive $E_{pu}(-u_a)$ and $E_{pu}(-u_b)$ from the client
    (b) Compute $\Gamma_i \leftarrow E_{pu}(u_i)$
    (c) $\hat{\Gamma}_i \leftarrow \pi(\Gamma_i)$, where $\pi$ is a random permutation
    (d) $E_{pu}(\eta_i) \leftarrow \left[\hat{\Gamma}_i \cdot E_{pu}(-u_a)\right]^{r_i}$, where $r_i \in_R \mathbb{Z}_N^*$
    (e) $E_{pu}(\theta_i) \leftarrow \left[\hat{\Gamma}_i \cdot E_{pu}(-u_b)\right]^{r'_i}$, where $r'_i \in_R \mathbb{Z}_N^*$
    (f) Send $E_{pu}(\eta_i)$ and $E_{pu}(\theta_i)$ to the client
3: Client:
    (a) Receive $E_{pu}(\eta_i)$ and $E_{pu}(\theta_i)$ from the server
    (b) Decrypt $E_{pu}(\eta_i)$ and $E_{pu}(\theta_i)$ to obtain $\eta_i$ and $\theta_i$
    (c) Compute $E_{pu}(k_i)$, such that,
            $$k_i \leftarrow \begin{cases} 1 & \text{if } \eta_i = 0 \text{ or } \theta_i = 0 \\ 0 & \text{otherwise} \end{cases}$$
    (d) Send $E_{pu}(k_i)$ to the server
4: Server:
    (a) Receive $E_{pu}(k_i)$ from the client
    (b) $E_{pu}(\omega_i) \leftarrow \prod_{u_j \in L_i} E_{pu}(k_j)$
    (c) $E_{pu}(\omega_i + r_i) \leftarrow E_{pu}(\omega_i) \times E_{pu}(r_i)$, where $r_i \in_R \mathbb{Z}_N$
    (d) Send $E_{pu}(\omega_i + r_i)$ to the client
```

```
5: Client:
    (a) Receive $E_{pu}(\omega_i + r_i)$ from the server
    (b) $\omega_i + r_i \leftarrow D_{pr}(E_{pu}(\omega_i + r_i))$
    (c) Compute $E_{pu}((\omega_i + r_i)(\omega_i - 1 + r_i))$ and send them to the server
6: Server:
    (a) Receive $E_{pu}((\omega_i + r_i)(\omega_i - 1 + r_i))$ from the client
    (b) $E_{pu}(\omega_i(\omega_i - 1)) \leftarrow E_{pu}((\omega_i + r_i)(\omega_i - 1 + r_i)) \times E_{pu}(-2r_i\omega_i + r_i - r_i^2)$
    (c) $E_{pu}(\omega_i') \leftarrow [E_{pu}(\omega_i(\omega_i - 1))]^h$
    (d) $E_{pu}(\rho) \leftarrow \prod_{i=1}^{n} E_{pu}(\omega_i')$
    (e) Compute noise $\delta' \leftarrow \lceil \delta \rceil$, where $\delta \sim_R \text{Laplace}(0, \lambda)$
    (f) $E_{pu}(s) \leftarrow E_{pu}(\rho) \times E_{pu}(\delta')$
    (g) Send $E_{pu}(s)$ to the client
7: Client:
    (a) Receive $E_{pu}(s)$ from the server
    (b) $s \leftarrow D_{pr}(E_{pu}(s))$
```

3.1.3 Security Analysis

The security of PPFR can be proved using the simulation method in [8]. First, we need to build a simulator based on the private input and output for each party. Since the computations between the two parties are asymmetric, the simulators are different for the individual parties. Let Π_S and Π_C denote the real execution images for the server and the client respectively. Similarly, $\Pi_{\tilde{S}}$ and $\Pi_{\tilde{C}}$ denote the simulated execution images. Next, we show how to construct a simulator Simulator-S to produce $\Pi_{\tilde{S}}$.

Simulator-S

Algorithm 2 Simulator-$S(pu)$

```
Require: $pu$ is the public key of Paillier
1: Generate four randoms $r_1$, $r_2$, $r_3$ and $r_4$ from $Z_N$
2: Return $E_{pu}(r_1)$, $E_{pu}(r_2)$, $E_{pu}(r_3)$ and $E_{pu}(r_4)$
```

For a two-party distributed protocol, private information can be disclosed from the messages exchanged during the execution of the protocol. The server receives messages from the client at Steps 2, 4 and 6 of the PPRF protocol. Thus, the real execution image Π_S consists of the following information ($1 \leq i \leq n$):

- $E_{pu}(-u_a)$, $E_{pu}(-u_b)$, $E_{pu}(k_i)$ and $E_{pu}((\omega_i + r_i)(\omega_i - 1 + r_i))$

The key steps of Simulator-S are given in Algorithm 2, and the simulated execution image $\Pi_S^{\sim}$ consists of

- $E_{pu}(r_1)$, $E_{pu}(r_2)$, $E_{pu}(r_3)$ and $E_{pu}(r_4)$

where each encrypted value corresponds to the value from the real execution.

Claim 1. $\Pi_S^{\sim}$ *is computationally indistinguishable from* Π_S.

Proof. Suppose the claim is not true, then this implies that one of the $E_{pu}(r_i)$ values is computationally distinguishable from the corresponding value of the real execution image. Without loss of generality, assume $E_{pu}(r_1)$ is computationally distinguishable from $E_{pu}(u_a)$. However, this contradicts the fact that the Paillier encryption scheme is semantically secure or computationally indistinguishable [12]. Therefore, $\Pi_S^{\sim}$ must be computationally indistinguishable from Π_S. □

The above claim demonstrates that fact that any information that the server learned during the execution of PPFR can be derived by what the server already knows. Thus, from the client's perspective, the protocol is computationally secure. Next we need to prove the protocol is secure from the server's perspective by building a simulator to simulate the client's execution image.

Simulator-C

The client receives messages from the server at Steps 3, 5 and 7 of the PPRF protocol. Thus, the real execution image Π_C consists of the following:

- $X \sim \langle E_{pu}(\eta_1), \ldots, E_{pu}(\eta_n), E_{pu}(\theta_1), \ldots, E_{pu}(\theta_n), \eta \equiv \eta_1 \cdots \eta_n, \theta \equiv \theta_1 \cdots \theta_n \rangle$
- $Y \sim \langle E_{pu}(\omega_1 + r_1), \ldots, E_{pu}(\omega_n + r_n), \omega_i + r_i, \ldots, \omega_n + r_n \rangle$

Algorithm 3 Simulator-$C(pr, s)$

```
Require: $pr$ is the private key of Paillier, $s$ is the output from PPRF
1: For $1 \leq j \leq n$, randomly generate $r_{1j}$, $r_{2j}$, and $r_{3j}$ from $Z_N$
2: For $1 \leq j \leq n$, compute $E_{pu}(r_{1j})$, $E_{pu}(r_{2j})$ and $E_{pu}(r_{3j})$
3: Let $\eta'$ and $\theta'$ be random permutations of a sequence of $n$ values, such that only
   one of them is 0 and the rest are randomly generated from $\mathbb{Z}_N^*$
4: Return the following $(1 \leq j \leq n)$
     • $X' \sim \langle E_{pu}(r_{1j}), E_{pu}(r_{2j}), \eta', \theta' \rangle$
     • $Y' \sim \langle E_{pu}(r_{3j}), r_{3j} \rangle$
     • $Z' \sim \langle E_{pu}(s), s \rangle$
```

- $Z \sim \langle E_{pu}(s), s \rangle$

where X, Y and Z denote the random variables related to the corresponding pair of values. For each pair, the first component is the message received, and the second component is the value derived from the first component. The simulator needs to simulate these messages and the information derived from them. Algorithm 3 provides the key steps for Simulator-C.

Claim 2. *$\Pi_C^{\sim}$ is computationally indistinguishable from Π_C.*

Proof. This proof is very similar to that of the previous claim. We omit some of the technical details. However, we want to emphasize that η' is indistinguishable from η due to the fact that the user list is randomly permuted at Step 2(c) of Algorithm 1. □

The above claim demonstrates that fact that any information that the client learned during the execution of the PPFR protocol can be derived by its private input and output. Thus, from the server's perspective, the protocol is computationally secure. Combining both, we prove the security of PPFR. Since appropriately scaled Laplacian noise is added to the output, differential privacy is achieved as well.

3.1.4 Complexity Analysis

In order to theoretically measure the computation complexity of PPFR, our analyses are based on the number of the most expensive operations used in the protocol which happen to be the encryption E and decryption D operations, as well as the exponentiation of a ciphertext. We use the Paillier cryptosystem where the costs of E and D are approximately similar. We represent both costs as e, use x to represent the cost of obtaining the exponentiation of a ciphertext, and use p to represent the cost of multiplying two ciphertests. Let t be the number of bits required for representing the ciphertext. The overall protocol complexity is derived for both the parties (the client and server), and provided in table 3.1, where O and T_x represent the computation and communication complexity, respectively.

Step	Server		Client	
	O	T_x	O	T_x
1			$2e$	$2t$
2	$ne + 2nx + 2np$	$2nt$		
3			$3ne$	nt
4	$(n^2 - 2m)e + ne + np$	nt		
5			$2ne$	nt
6	$(n+1)e + (2n+1)p + x$	t		
7			e	
Total	$(n^2 + 3n - 2m + 1)e+$ $(2n+1)x + (5n+1)p$	$(3n+1)t$	$(5n+3)e$	$2(n+1)t$

Table 3.1: Computation and Communication Complexity of the PPFR algorithm

3.2 Experimental Results

In this section, we provide a detailed empirical analysis, both qualitative and quantitative, of the performance of the various algorithms proposed.

3.2.1 Experimental Setup

Dataset: In the absence of social network data, we use the real-world DBLP computer science bibliography dataset (Aug. 2018) [74] to test our system. The original dataset contains 6420665 bibliographic records. The dataset is preprocessed as follows:

From the dataset we extract 2185132 authors (represented as nodes) and their corresponding collaborator list giving us 9507042 edges that represent each collaboration relationship. One issue with the DBLP dataset is that author names are used to identify authors, and therefore, common names would be associated with all the papers attributed for that particular name and consequently all the collaborators for those papers as well e.g. one of the commonly occurring names had 2575 number of collaborators. Since the recommendation functionality is based on the collaborator (or friend) count, we removed outliers (i.e., authors with collaborator count that is more than three standard deviations above the mean). After this we still have 2156785 authors, but the maximum number of collaborators is reduced to 73.

Machine Hardware Details: The machines used for both S and C have the following specifications:

- Processor: 64-bit Intel® Xeon® CPU E2186G @ 3.80GHz 12CPU(s)
- Memory: 62GiB DIMM DDR4 2666MHz (0.4 ns)
- Hard disk: 1024GB PC400 NVMe SK hynix

Programming Languages and Libraries: The entire implementation is in C. The GNU GMP [75] library was used for efficiently computing the cryptographic primitives required for the algorithms. The public key encryption scheme, used in the algorithms, was based on the Pailler Cryptosystem [12] for the cryptosystem's additive homomorphic properties.

3.2.2 Empirical Analysis

In this section, we report on the performance of the algorithm with the strongest privacy guarantee viz. the optimum PPFR algorithm, by varying different parameters such as the dataset size, the key size, and the count of queries.

Performance Evaluation:

First, we vary the user count (n). As seen in Figure 3.1, the query response time linearly increases with the number of users in the database. A single query run for 20%, i.e. roughly 0.4 million, of the total DBLP authors takes under 685s to complete. Typically, the size

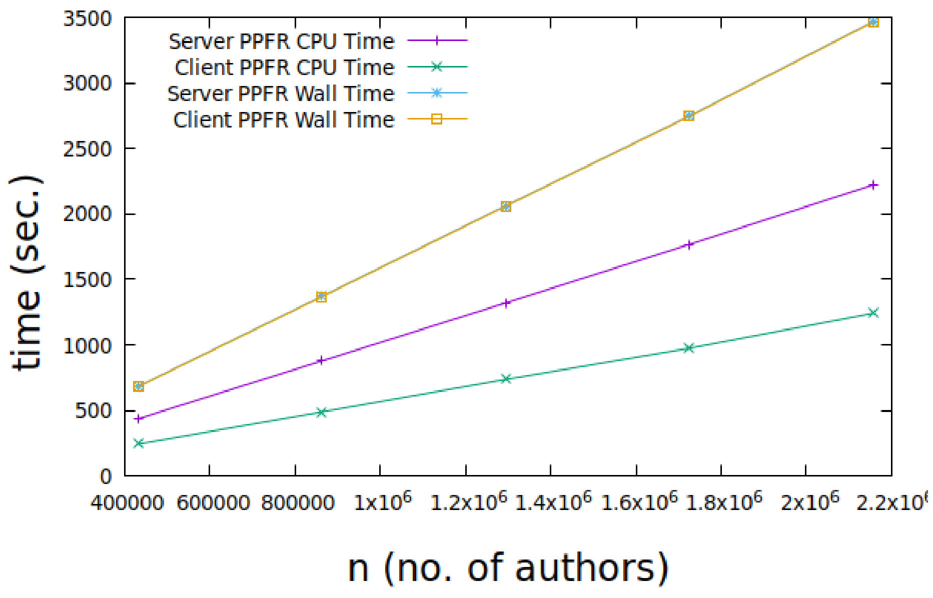

Figure 3.1: Run time vs node count

of the dataset considered by any OSN to run queries over would be lesser, by orders of magnitude than the 0.2 million users here, for any normal user. It is to be noted that certain online social services (C) restrictively operate for a given region, or a set of users only. Based on this background information, the data owner (S) can always consider a subset of its graph G_S to run queries. Access to this background information does not harm our privacy guarantee of maintaining relationships or friends private nor leak users ids in G_M nor leak potential new friendships formed in G_M by way of this application.

Next, we vary the edge count (m). The results are shown in Figure 3.2, and are similar to the case of varying author (node) count. Note however, that figure 3.2 was generated by regulating the node percentage. Therefore, it involves a combined effect of edge count and node count variation on times reported, i.e. it is not purely independent of the number of authors n.

Figure 3.3 shows the time taken by S and C when the keysize is varied. As expect, the time for execution increases exponentially with respect to the key size. Typically, the key size used is 1024. 20% of the nodes were considered in order to obtain this result.

Finally, we vary the count of queries(q) and obtain the total CPU time and the Wall clock time. As can be seen from the figure 3.4, the times are linear in q.

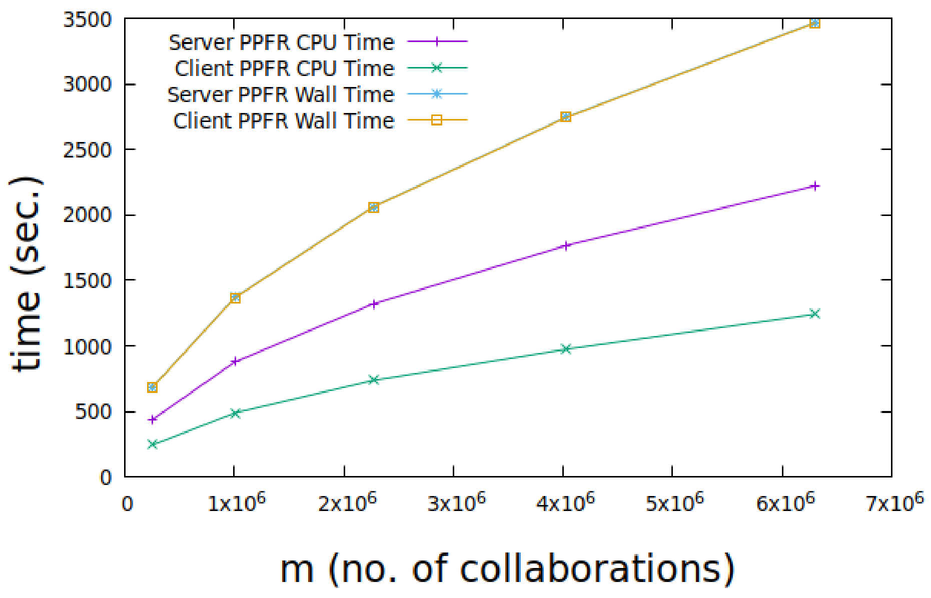

Figure 3.2: Run time vs edge count

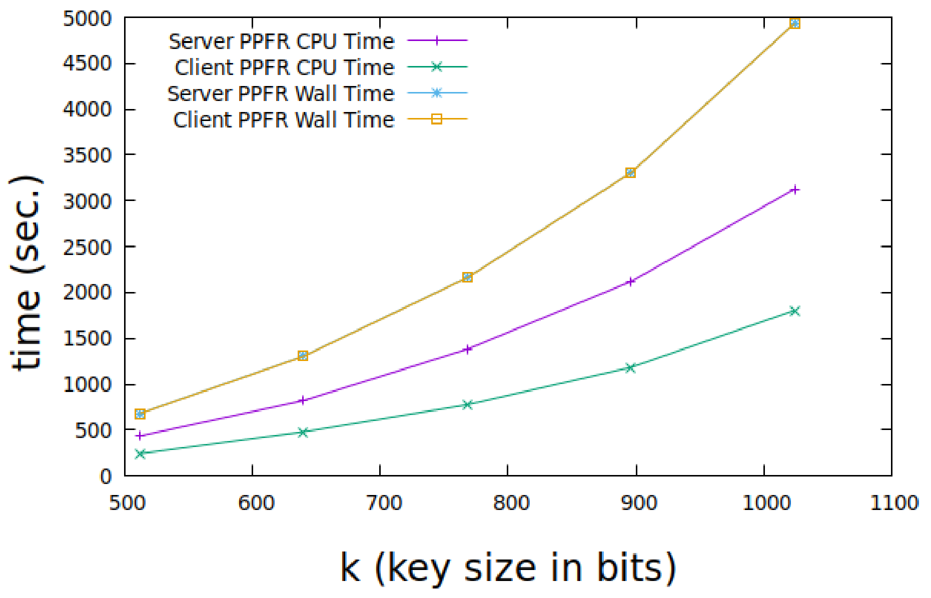

Figure 3.3: Run time vs public key size

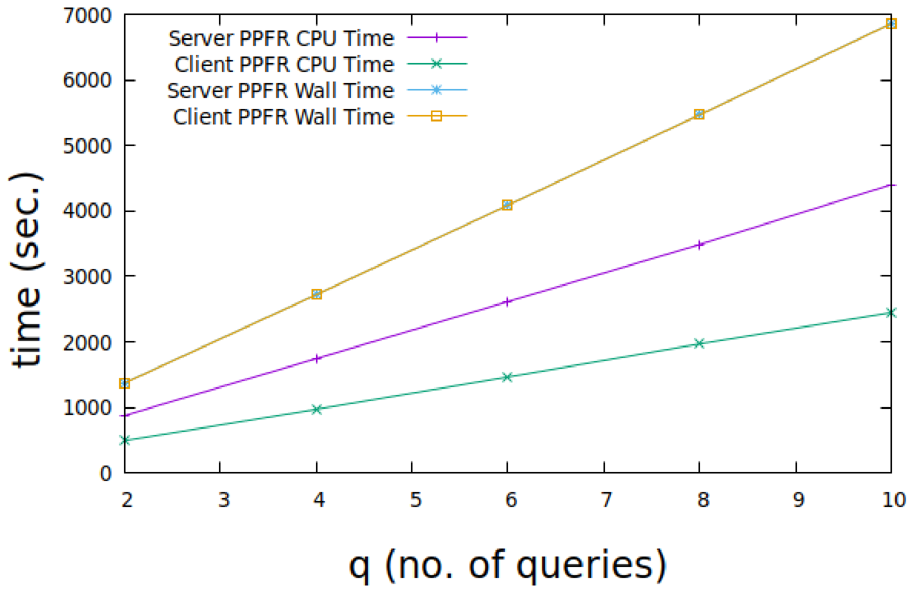

Figure 3.4: Run time vs count of queries

Utility Evaluation:

In this section, we plot Spearman's corrected ρ [76], Kendall's τ, τ_b [77] as well as the precision, recall and f-measure statistics [78] of the top-k recommended users in order to measure the utility of the proposed approach. ρ measures the magnitude and direction of the monotonic relationship amongst the variables while τ (τ_b) measures the difference in probabilities of data being in the same order and in different order [77]. We measured both Kendall's τ and Kendall's τ_b, since τ_b accounts for ties that can occur, especially, in ranked lists for non-DP scores. Although the number of ties depends on the underlying DBLP dataset, in this case, measuring τ and τ_b provides some information about the underlying dataset while highlighting the need for using pre-processing techniques that can reduce the ties and improve the overall utility of the system or vice-versa.

The utility coefficients are computed over the ranked list of size c before and after perturbation. These coefficients are averaged over 10000 times with a list generated each time for a randomly selected author. Please note that for $c = 70$ there were less number of authors to choose from as compared to other list sizes.

Figures 3.5, 3.7, and 3.9 measure the utility when the list size (c) is varied.From Figure

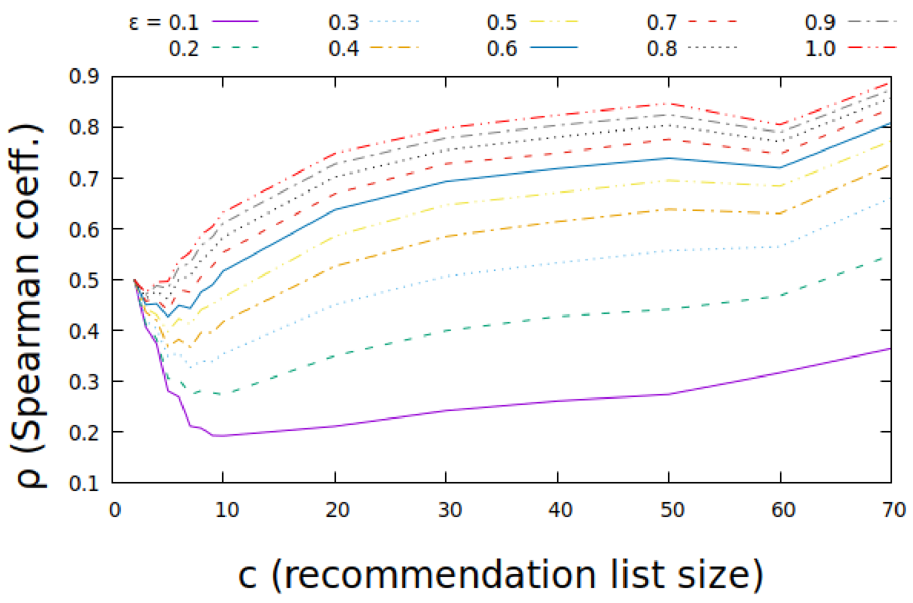

Figure 3.5: Spearman coeff. (ρ) vs list size (c)

3.5, we can see how PPFR's ρ initially decreases as the list size increases (until $c = 10$) and then subsequently rises along with c. This is uniquely captured just by the Spearman coefficient and not by τ or τ_b. The reason for this behaviour might be that for a given ϵ and sensitivity S ($S = 1$ for edge DP), for any list score, the differences in ranks before and after perturbation might be bounded to some extent while the n, i.e. c in this case, is incremented. Figures 3.6, 3.8, and 3.10 measure the utility for different values of the privacy parameter ϵ. As expected, the utility of PPFR increases as ϵ is increased. The utility also increases as the list size is increased. From figure 3.10, it can be seen that low c values with high ϵ perform better than high c value and low ϵ values.

Unlike Spearman's, Kendall's coefficients do not reach their lowest values closer to $c = 10$, see figures 3.7, 3.9, however, they do show mild fluctuations around it indicating greater uncertainty. In general, $\tau_b \geq \tau$ against both c and privacy loss parameter ϵ, which can be seen from Figures 3.7 - 3.10

We also measure precision, recall and the f-measure statistic for top-k ranks within a list of size c. Due to space restrictions we only report results for $k = 5$ and $k = 10$. From Figures (3.11), (3.12), (3.13), (3.14), (3.15), (3.16), it can be seen that the observed precision is generally higher than the recall. This effect is due to the multiple rank ties found in scores

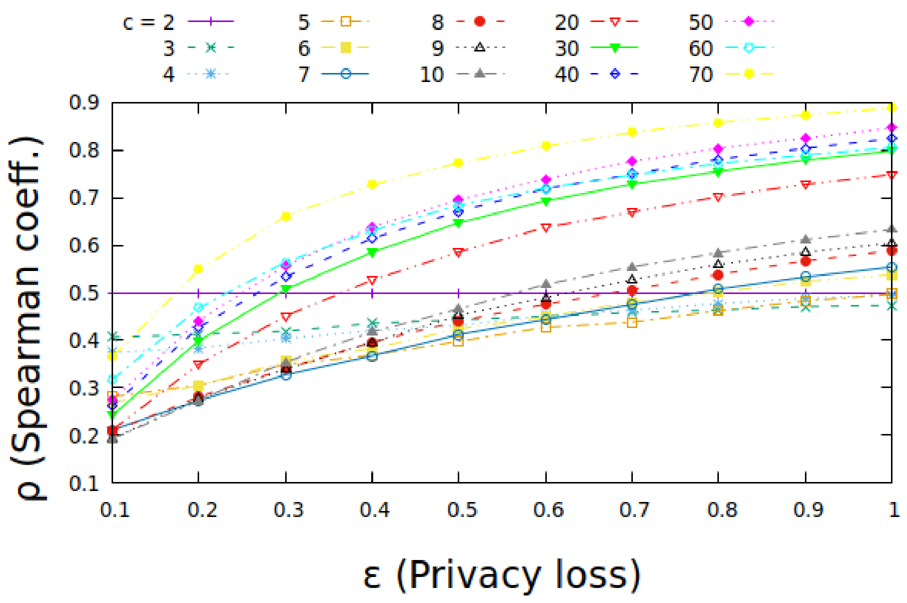

Figure 3.6: Spearman coeff. (ρ) vs Privacy loss (ϵ)

before DP perturbation.

Figures (3.11), (3.12), (3.13), (3.14), (3.15), (3.16) and (3.17) show the relation of precision, recall and f-measure with c. For both top-5 and top-10, as c rises, all the three metrics decrease and the relationship increasingly represents a decreasing logarithmic relation. It can be seen that top-10 performs better as compared to top-5 for the f-measure statistic. Furthermore utility increases with rise in ϵ, except for $c = 5$ and $c = 10$ where it is constant at 1. It can be seen that for lower values of c, the precision, recall and f-measure increase in an approximately linear way.

Figure 3.17 shows the relation of utility with respect to k. In this figure we plot precision, recall and f-measure by varying list sizes (c) for $\epsilon = 0.1$. As expected, the utility for a fixed k decreases as c is incremented. It is seen that by incrementing k, the utility score initially decreases but then rises depending significantly on the other parameters. This information is useful if top-k has to be applied on some sub-graph only as depending on the sub-graph size (c) selected, system utility may get unintentionally modified.

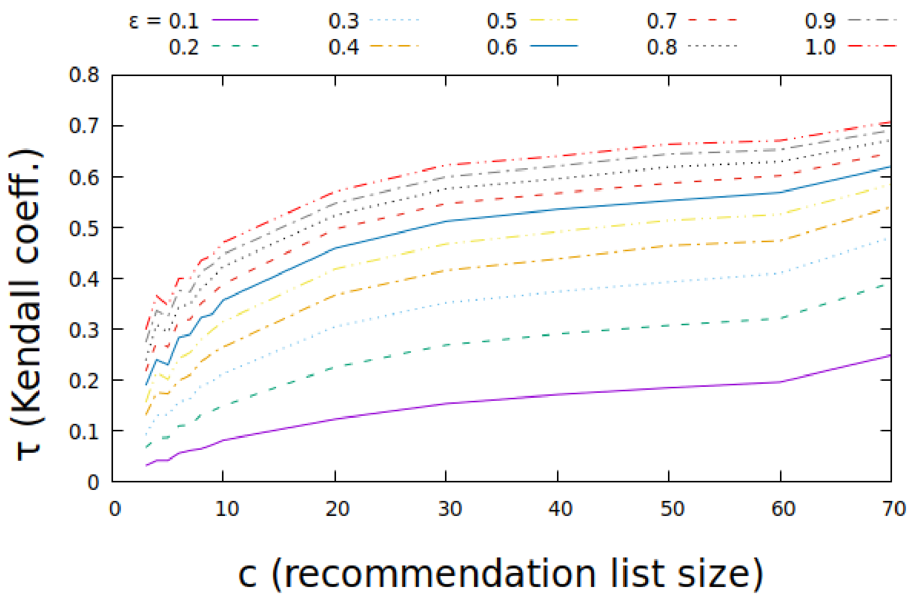

Figure 3.7: Kendall coeff. (τ) vs list size (c)

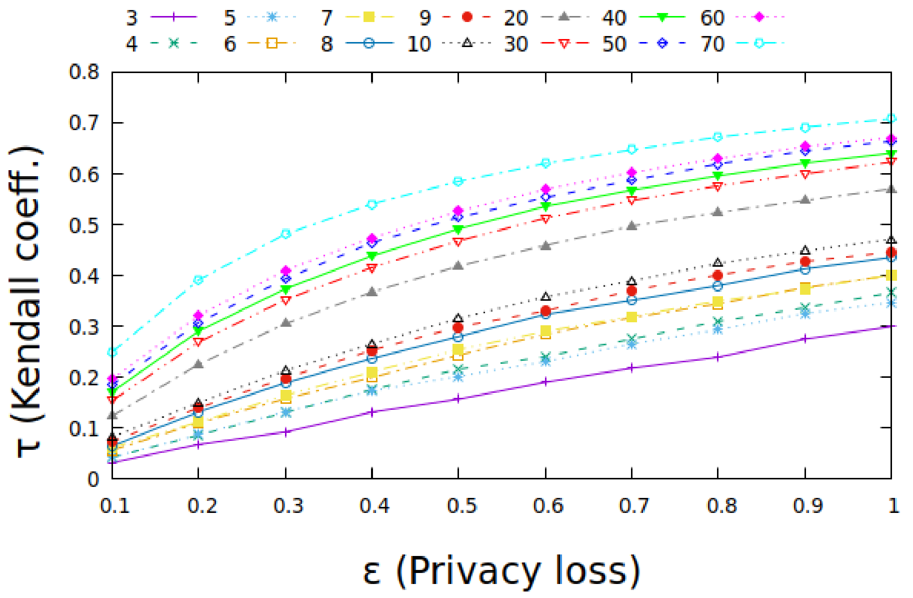

Figure 3.8: Kendall coeff. (τ) vs Privacy loss (ϵ)

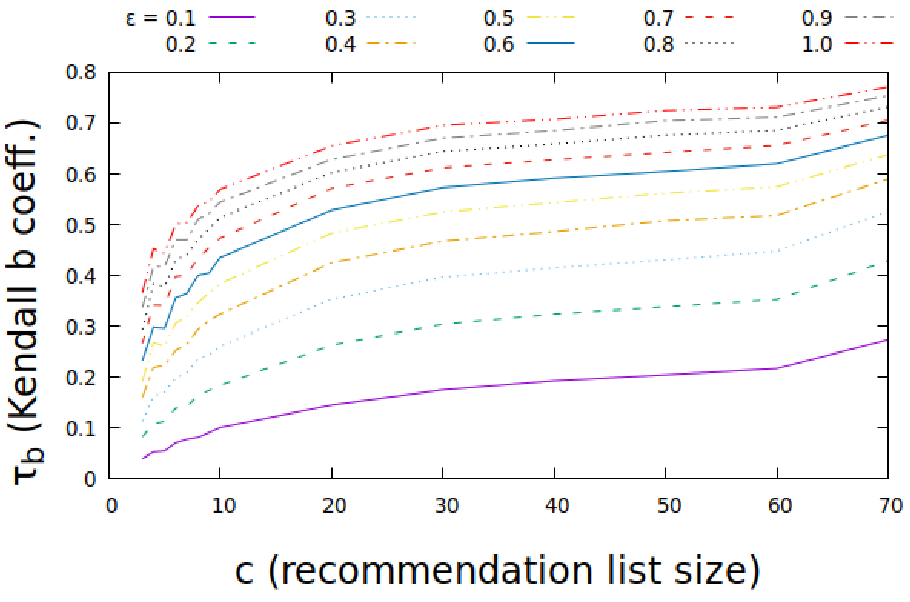

Figure 3.9: Kendall coeff. (τ_b) vs list size (c)

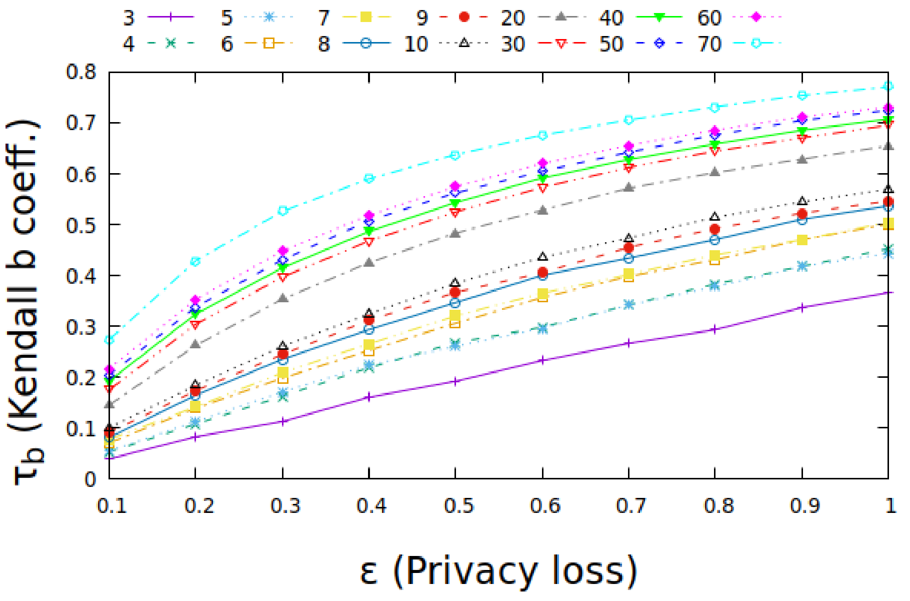

Figure 3.10: Kendall coeff. (τ_b) vs Privacy loss (ϵ)

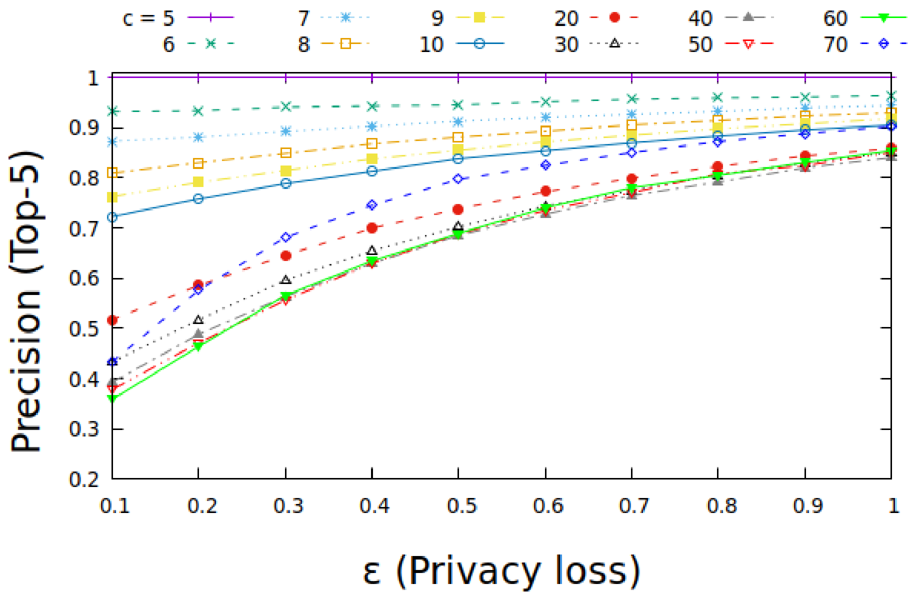

Figure 3.11: Top-5: Precision vs Privacy loss (ϵ)

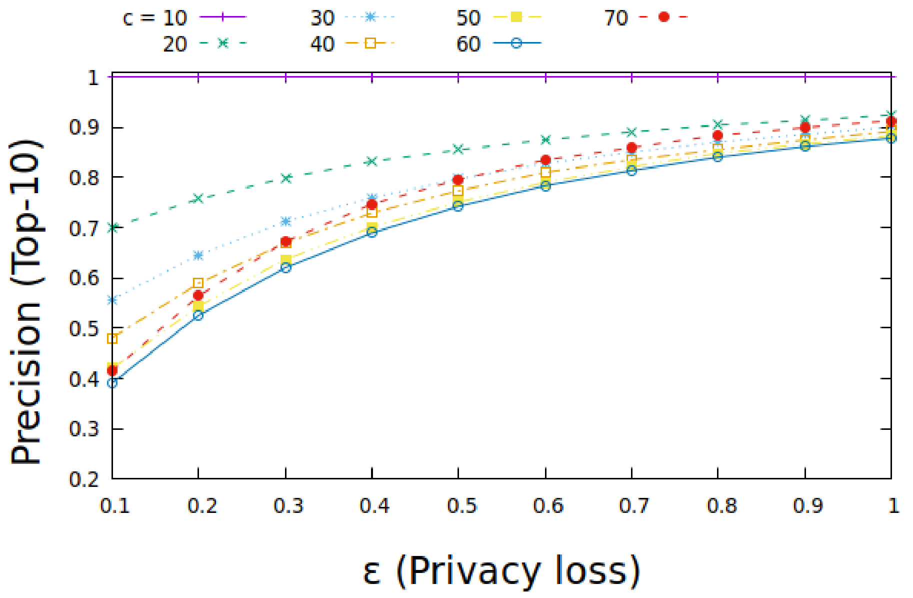

Figure 3.12: Top-10: Precision vs Privacy loss (ϵ)

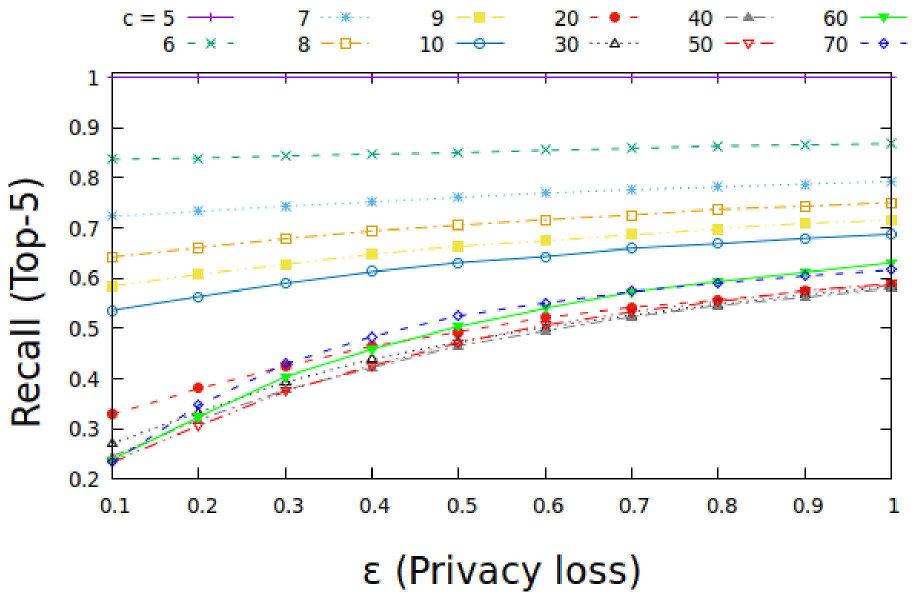

Figure 3.13: Top-5: Recall vs Privacy loss (ϵ)

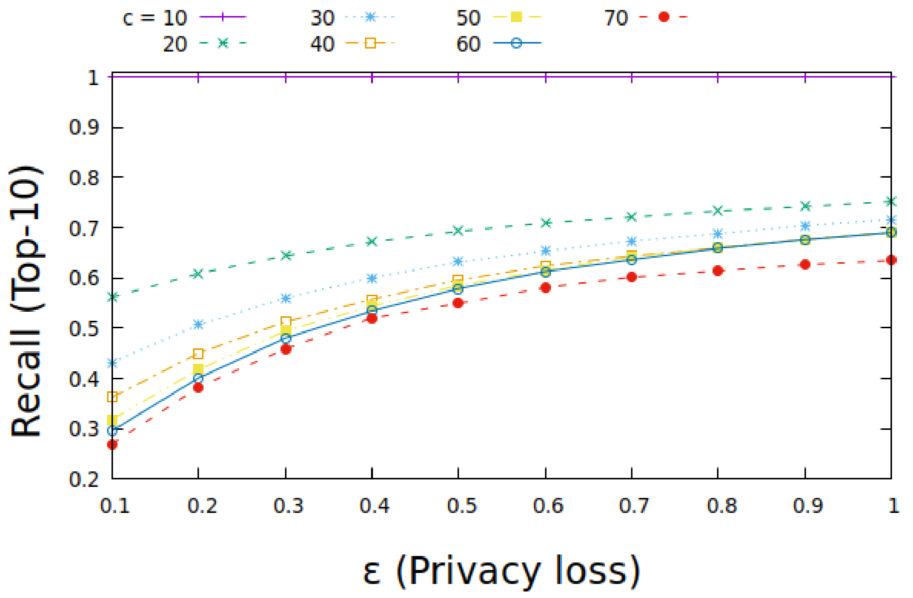

Figure 3.14: Top-10: Recall vs Privacy loss (ϵ)

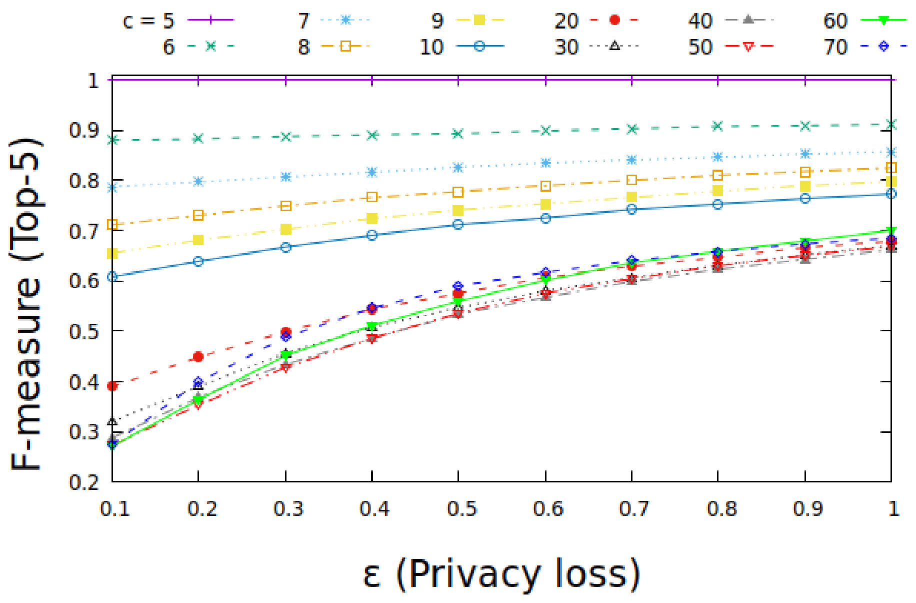

Figure 3.15: Top-5: F-measure vs Privacy loss (ϵ)

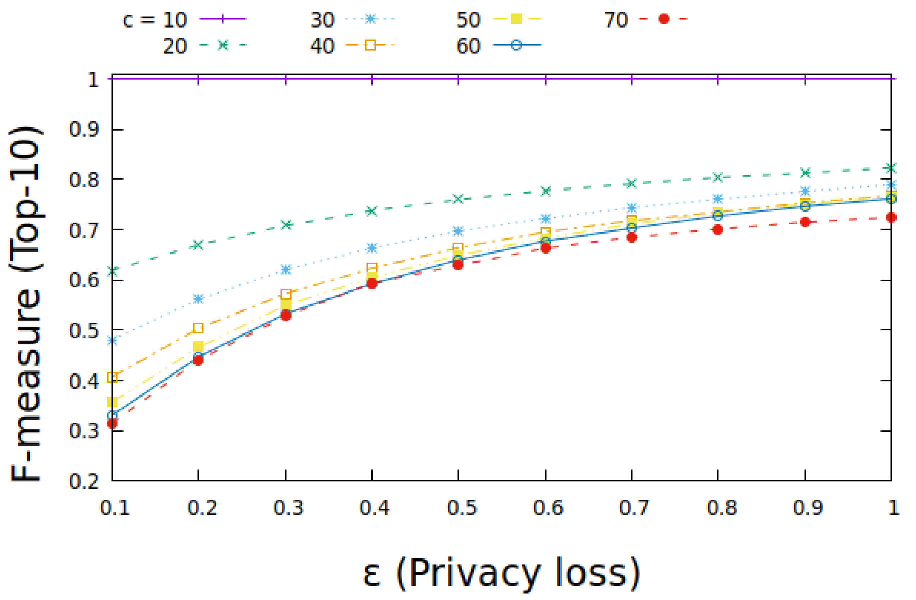

Figure 3.16: Top-10: F-measure vs Privacy loss (ϵ)

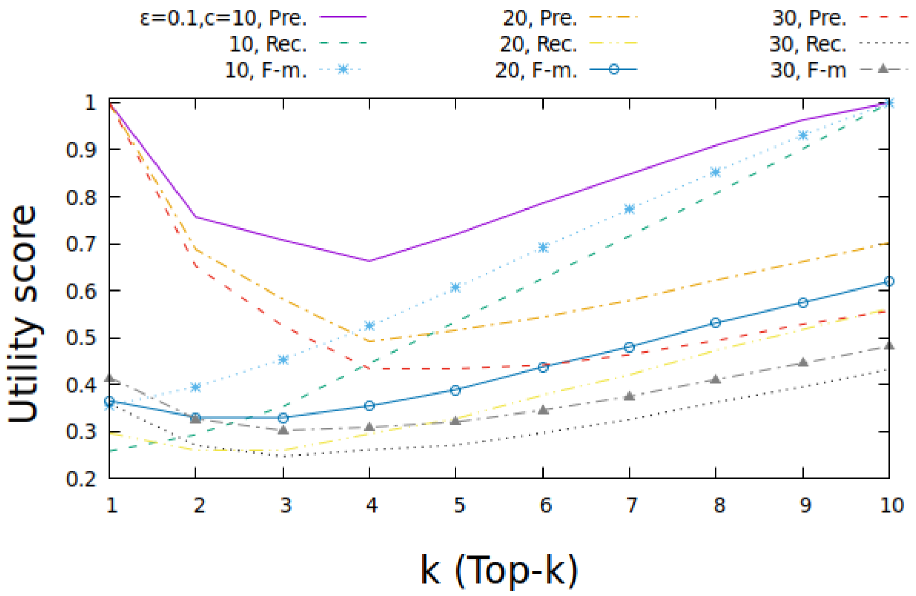

Figure 3.17: Precision, Recall, F-measure vs k for the algorithm PPFR

Chapter 4

Privacy-Preserving Friend Recommendation based on Common Friends - Using Secret Sharing Approaches

The work presented in this chapter is an extension of the work presented in the previous chapter 3. The fundamental differences between the two approaches, however, are the algorithms, number of parties required for computations, the privacy-preserving methodologies used and the speedup gained as a result of them. Both of these techniques have their benefits and can be deployed as per the underlying system requirements in which these would be applied.

In the previous approach, a Differential Private (DP) noise was added in step (f) of algorithm (1) in order to prevent the leakage of information over a bound specified by the DP parameter ϵ [79]. The same approach can be applied to the algorithm presented in this chapter. Therefore, in this chapter we will leave out the discussion of the commonalities between the two approaches referenced here.

The chief difference is in the way the mutual friend count score is computed. In the previous approach we used Additive Homomorphic Encryption (HEnc) while in this proposed solution we use the additive secret sharing technique. Both are Multi-Party Computation

(MPC) techniques.

4.1 The Proposed Protocol

In this section, we provide the details about the PPFR–SI algorithm and its pre-processing step defined by PPFR–SI-PreProc algorithm.

4.1.1 Protocol Initialization

```
**Algorithm 4** PPFR–SI-PreProc $(\langle P_1, G_S\rangle) \rightarrow (\langle P_0, [A]_Q^{P_0}\rangle, \langle P_2, [A]_Q^{P_2}\rangle)$
Require: $h \leftarrow 2^{-1} \mod N$, $n$ denotes the number of users at the server. Index $i$ s.t.
    $1 \leq i \leq n$. $Q$ is a randomly generated large prime s.t. $Q \in Z_N$. $A_{n\times n} \leftarrow G_S$'s
    adjacency matrix representation, $a_{i,j} \leftarrow$ element at row $i$ and column $j$ in $A$,
    where $1 \leq i, j \leq n$. $[x]_Q^{P_l} \leftarrow$ share of $x$ belonging to party $P_l$.
1: $P_1$:
    (a) Generate adjacency matrix $A_{n\times n}$ from $G_S$ s.t. $a_{i,j} \in A$
    (b) Randomly generate two shares $[a_{i,j}]_Q^{P_0}$ and $[a_{i,j}]_Q^{P_2}$ from $a_{i,j}$
    (c) Send $[A]_Q^{P_0}$ to $P_0$, where $[a_{i,j}]_Q^{P_0} \in [A]_Q^{P_0}$
    (d) Send $[A]_Q^{P_2}$ to $P_2$, where $[a_{i,j}]_Q^{P_2} \in [A]_Q^{P_2}$
2: $P_0$:
    (a) Receive $[A]_Q^{P_0}$ from $P_1$
3: $P_2$:
    (a) Receive $[A]_Q^{P_2}$ from $P_1$
```

4.1.2 The Main Protocol

Algorithm 5 PPFR–SI $(\langle P_0, [A]_Q^{P_0}\rangle, \langle P_1, A\rangle, \langle P_2, u_a, u_b, [A]_Q^{P_2}\rangle) \rightarrow \langle P_2, s\rangle$

```
Require: $h \leftarrow 2^{-1} \mod N$, and the index $i$ varies from 1 to $n$ where $n$ denotes the
    number of users at the server. Let $Q$ be a randomly generated large prime such
    that $Q \in Z_N$. Let matrix $A_{n\times n}$ represent $G_S$'s adjacency matrix representation.
1: $P_1$:
    (a) Generate shares $[u_i]_Q^{P_0}$ and $[u_i]_Q^{P_1}$ from $u_i$
    (b) Send $\gamma_{1,i}$, $\gamma_{2,i}$ and $[u_i]_Q^{P_0}$ to $P_0$, where $\gamma_{1,i}$, $\gamma_{2,i} \in_R \mathbb{Z}_Q$
2: $P_0$:
    (a) Receive $\gamma_{1,i}$, $\gamma_{2,i}$ and $[u_i]_Q^{P_0}$ from $P_1$
3: $P_2$:
    (a) Generate shares $[u_a]_Q^{P_0}$, $[u_a]_Q^{P_1}$ and $[u_b]_Q^{P_0}$, $[u_b]_Q^{P_1}$ from $u_a$, $u_b$, respectively
    (b) Send $[u_a]_Q^{P_l}$, $[u_b]_Q^{P_l}$ to $P_l$, where $l \in \{0,1\}$
4: $P_l$: $l \in \{0,1\}$
    (a) Receive $[u_a]_Q^{P_l}$, $[u_b]_Q^{P_l}$ from $P_2$
    (b) Compute $[\kappa_{1,i}]_Q^{P_l} \leftarrow \gamma_{1,i} \cdot \left([u_a]_Q^{P_l} - [u_i]_Q^{P_l}\right)$ and $[\kappa_{2,i}]_Q^{P_l} \leftarrow \gamma_{2,i} \cdot \left([u_b]_Q^{P_l} - [u_i]_Q^{P_l}\right)$
    (c) Send $[\kappa_{1,i}]_Q^{P_l}$ and $[\kappa_{2,i}]_Q^{P_l}$ to $P_2$
5: $P_2$:
    (a) Receive $[\kappa_{1,i}]_Q^{P_l}$ and $[\kappa_{2,i}]_Q^{P_l}$ from $P_l$, where $l \in \{0,1\}$
    (b) Construct $\kappa_{1,i}$ and $\kappa_{2,i}$ from $[\kappa_{1,i}]_Q^{P_l}$ and $[\kappa_{2,i}]_Q^{P_l}$, respectively, where $l \in \{0,1\}$
    (c) Compute query indices $\alpha$, $\beta$ from $\kappa_1$, $\kappa_2$ via eqs. (4.1), (4.2), respectively
        $$\alpha \leftarrow \begin{cases} r_{1,i} & \forall \kappa_{1,i} \neq 0 \\ i & \exists \kappa_{1,i} = 0 \end{cases} \quad (4.1) \qquad \beta \leftarrow \begin{cases} r_{2,i} & \forall \kappa_{2,i} \neq 0 \\ i & \exists \kappa_{2,i} = 0 \end{cases} \quad (4.2)$$
        where, both $r_{1,i}$, $r_{2,i} \in_R \{1, 2, \ldots, n\}$ are randomly generated
    (d) Send $\alpha$, $\beta$ to $P_0$
6: $P_0$:
    (a) Receive $\alpha$, $\beta$ from $P_2$
```

```
7: $P_1$:
    (a) Generate $n$ Beaver product triplet shares: $[x_j]_Q^{P_0}$, $[x_j]_Q^{P_2}$, $[y_j]_Q^{P_0}$, $[y_j]_Q^{P_2}$,
        $[x_j \cdot y_j]_Q^{P_0}$, $[x_j \cdot y_j]_Q^{P_2}$, where, $x_j, y_j \in_R Z_Q$ and $1 \leq j \leq n$
    (b) Send $[x_j]_Q^{P_0}$, $[y_j]_Q^{P_0}$, and $[x_j \cdot y_j]_Q^{P_0}$ to $P_0$
    (c) Similarly, send $[x_j]_Q^{P_2}$, $[y_j]_Q^{P_2}$, and $[x_j \cdot y_j]_Q^{P_2}$ to $P_2$
8: $P_1$:
    (a) Compute noise $\delta' \leftarrow \lceil \delta \rceil$, where $\delta \sim_R \text{Laplace}(0, \lambda)$
    (b) Generate random shares $[\delta']_Q^{P_0}$ and $[\delta']_Q^{P_2}$ from $\delta'$
    (c) Send $[\delta']_Q^{P_0}$ to $P_0$
    (d) Send $[\delta']_Q^{P_2}$ to $P_2$
9: $P_l$: $l \in \{0, 2\}, 1 \leq j \leq n$
    (a) Receive $[x_j]_Q^{P_l}$, $[y_j]_Q^{P_l}$, $[x_j \cdot y_j]_Q^{P_l}$ from $P_0$
        $\Big(\text{Let } [c_j]_Q^{P_l} \leftarrow [a_{\alpha,j}]_Q^{P_l} \text{ and } [d_j]_Q^{P_l} \leftarrow [a_{\beta,j}]_Q^{P_l}\Big)$
    (b) Compute $[c_j + x_j]_Q^{P_l}$, $[d_j + y_j]_Q^{P_l}$
    (c) Send $[c_j + x_j]_Q^{P_l}$, $[d_j + y_j]_Q^{P_l}$ to $P_{2-l}$
    (d) Receive $[c_j + x_j]_Q^{P_{2-l}}$, $[d_j + y_j]_Q^{P_{2-l}}$ from $P_{2-l}$
    (e) Compute $(c_j + x_j)$, $(d_j + y_j)$
    (f) Compute $[c_j \cdot d_j]_Q^{P_l} \leftarrow (c_j + x_j)\,[d_j + y_j]_Q^{P_l} - (c_j + x_j)\,[y_j]_Q^{P_l} - (d_j + y_j)\,[c_j]_Q^{P_l} +$
        $[x_j \cdot y_j]_Q^{P_l}$
    (g) Compute $[\rho]_Q^{P_l} \leftarrow \sum_{j=1}^{n} [c_j \cdot d_j]_Q^{P_l}$
    (h) Compute $[s]_Q^{P_l} \leftarrow [\rho]_Q^{P_l} + [\delta']_Q^{P_l}$
10: $P_0$:
    (a) Send $[s]_Q^{P_0}$ to $P_2$
11: $P_2$:
    (a) Receive $[s]_Q^{P_0}$ from $P_0$
    (b) Compute $s \leftarrow [s]_Q^{P_0} + [s]_Q^{P_2}$
```

4.1.3 Security Analysis

In this section we provide the security analysis of PPFR–SI and PPFR–SI-PreProc protocol The private data i.e. adjacency matrix A for P_1 and query ids u_a and u_b for P_2 are exposed as shares only.

Both the shares need to be held by the adversary in order to regenerate the secret or private data, which does not happen in this case. It is important to note that α and β, the shares of query user ids, are supposed to be known as per the protocol.

Except what is supposed to be known as per protocol specifications, all intermediate data are shared as shares. Secret sharing being inherently information theoretic in nature helps us to prove that the protocol is secure [80].

4.1.4 Complexity Analysis

In this section, we present the complexity analysis of the PPFR–SI main protocol given in algorithm (5). The main operations performed for the shares are share generation, their re-construction, addition, product of shares (not the secrets) as well as sending them over to the other party. Complexity costs can be categorized as computational and transfer costs over the network.

Computational Costs: Computational costs could be further classified as product and addition costs, as given below.

1. Product cost (p): Let the cost of product of two shares, and not their secret, be represented by 'p'. Cost p also includes the modulus of the product result by Q.

2. Addition cost (a): Share generation operation involves random number generation for one part of share and subtraction from the secret to generate the other part of share. Random number generation can be considered as a constant 'read' operation and a huge amount of these random values can be created as a pre-processing step for the algorithm. Thus, subtraction operation is involved primarily followed by a modulus operation by Q. Subtraction could be generalized as an addition operation. Each

addition operation over the shares is strictly followed by a modulus by Q. Let the joint cost of addition (subtraction) and modulus operation be denoted by 'a'. Likewise, share reconstruction requires plain addition followed by the modulus operation.

Due to the nature of the operations, product operation p is more expensive than the addition cost a of the shares.

Communication cost: In Secure Multi-Party Computation (SMC), parties have to transfer intermediate data for secure evaluation of the function. In the case of our solution, PPFR–SI, share values are transferred over the network. In order to measure the cost for communication, we can simply count the number of shares being transferred from one party to the other. Since all the shares are of constant size, we can represent the size of a share as t bytes. For example, if party P_1 sends one share across to party P_2, we can note only the transfer operation of t bytes once.

Complexity Analysis: Table (4.1) represents the computation and communication costs for the PPFR–SI algorithm (see alg. 5). n is the number of nodes within the graph G_S over which the mutual friend count score is to be computed, please see section (4.1) for relevant details. We report both the costs for all the three parties involved in the computation. For each party, computation costs are represented under column titled O, while the communication costs are represented using T_x.

It can be seen that for all the parties, the computation and communication costs are linear in terms of the number of nodes (n) within the network i.e. $O(n)$. The constant terms have been dropped as per the standard practice in reporting complexities. Considering the share product cost (p) to be higher than addition cost (a), party P_0 has the highest computational workload followed by parties P_1 and P_2.

P_1 has the most communication complexity due to sending out the shares of user ids (u_i) as well as the Beaver triplet shares in steps (1) and (7), respectively.

PPFR–SI vs PPFR

We presented protocol PPFR, see alg. (1), which computes the same function, i.e. mutual friend or neighbor count, as that of PPFR–SI (5). However, PPFR uses the additive homomorphic properties of public key cryptosystem to arrive at the solution, while using just two

Step	Third Party (P_0)		Server (P_1)		Client (P_2)	
	O	T_x	O	T_x	O	T_x
1			$2na$	$3nt$		
2						
3					$2a$	$4t$
4	$2n(a+p)$	$2nt$	$2n(a+p)$	$2nt$		
5					na	$2t$
6						
7			$2n(a+p)$	$6nt$		
8			a	$2t$		
9	$n(8a+3p)$	$2nt$			$n(8a+3p)$	$2nt$
10		t				
11						1
Total	$5n(2a+p)$	$4nt$	$2n(3a+2p)$	$11nt$	$3n(3a+p)$	$2nt$

Table 4.1: Computation and Communication Complexity of the PPFR–SI algorithm

parties P_1 and P_2. Both these protocols add Differential Private noise to avoid leaking any possible edge level information from their outputs. Both these protocols can be deployed based on the system constraints of number of parties afforded.

Complexity of PPFR is provided in section (3.1.4) and table (3.1). It can be theoretically seen that PPFR–SI outperforms PPFR significantly with the help of an additional party, namely, P_0.

Computation complexity of PPFR is quadratic ($O(n^2)$) in e and linear ($O(n)$) in x and p, where e, x and p stand for costs of encryption/decryption, exponentiation over ciphertext domain and product over ciphertext domain, respectively. These are significant costs over the linear complexities discussed in this section for PPFR–SI. Moreover, ciphertexts span thousands of bits i.e. 2048 or 4096 where as the shares could be represented within 64 bits. Thus, costs for a, and p for PPFR–SI are quite low, i.e. orders of magnitude lower, than costs for e, x and p for PPFR due to its need to compute over encrypted data.

Additionally, when comparing the communication complexities, both are linear in t, however, t of PPFR typically consists of 2048 or 4096 bits whereas for PPFR–SI it would be 64 bits only, which would lead to significant decrease in network traffic in favor of PPFR–SI.

4.2 Experimental Results

In this section, we will discuss about the experimental details such as the setup used in terms of the computing hardware and dataset. We will also provide some analysis based on the graphs, which depict the performances among the parties in PPFR–SI protocol.

4.2.1 Experimental Setup

Dataset: The dataset used for experimental purposes is the same as the one used for PPFR as described in section 3.2.1 and provided here. In the absence of social network data, we use the real-world DBLP computer science bibliography dataset (Aug. 2018) [74] to test our system. The original dataset contains 6420665 bibliographic records. The dataset is pre-processed as follows:

From the dataset we extract 2185132 authors (represented as nodes) and their corresponding collaborator list giving us 9507042 edges that represent each collaboration relationship. One issue with the DBLP dataset is that author names are used to identify authors, and therefore, common names would be associated with all the papers attributed for that particular name and consequently all the collaborators for those papers as well e.g. one of the commonly occurring names had 2575 number of collaborators. Since the recommendation functionality is based on the collaborator (or friend) count, we removed outliers (i.e., authors with collaborator count that is more than three standard deviations above the mean). After this we still have 2156785 authors, but the maximum number of collaborators is reduced to 73.

Machine Hardware Details: The machines used for both S and C have the following specifications:

- Processor: 64-bit Intel® Xeon® Gold 6240R CPU @ 2.40GHz
- Memory: 196Gib DIMM DDR4
- Hard disk: 480GB SSD SATA 3.3, 6.0 Gb/s

Programming Languages and Libraries: The entire implementation was done in C language. No external library was required except 'pthread' unlike PPFR.

4.2.2 Empirical Analysis

In this section, we provide the plots with graphs for various parties for their CPU time i.e. User time + System time for PPFR–SI. We also compare the protocol with the PPFR algorithm to display its efficiency and improvement.

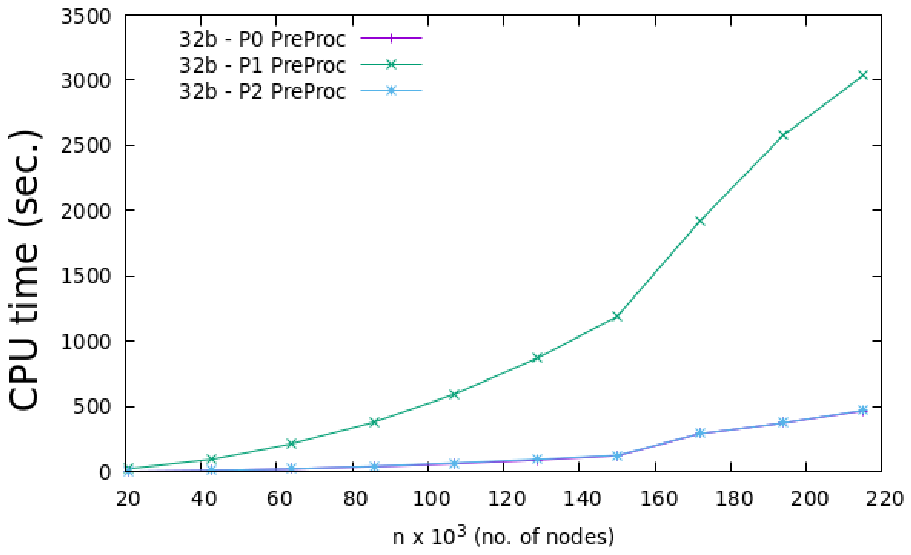

Figure 4.1: Pre-Proc CPU time vs node count

Performance Evaluation: Figures 4.1, 4.2, 4.3 and 4.4 provide an empirical comparison among the times i.e. CPU and Wall clock times versus the number of nodes in the graph.

Figure 4.1 shows the comparison in terms of the parties involved in the PPFR protocol. As can be seen, major bulk of work is done by the P_1. This work primarily consists of generating adjacency matrix A and then creating out shares for all the individual values in it. The graphs shown for PPFR–SI and its pre-processing counterpart PPFR–SI-PreProc are for 32 bit shares.

Similar pattern can be seen in figure (4.2) where bulk of work is done by P_1.

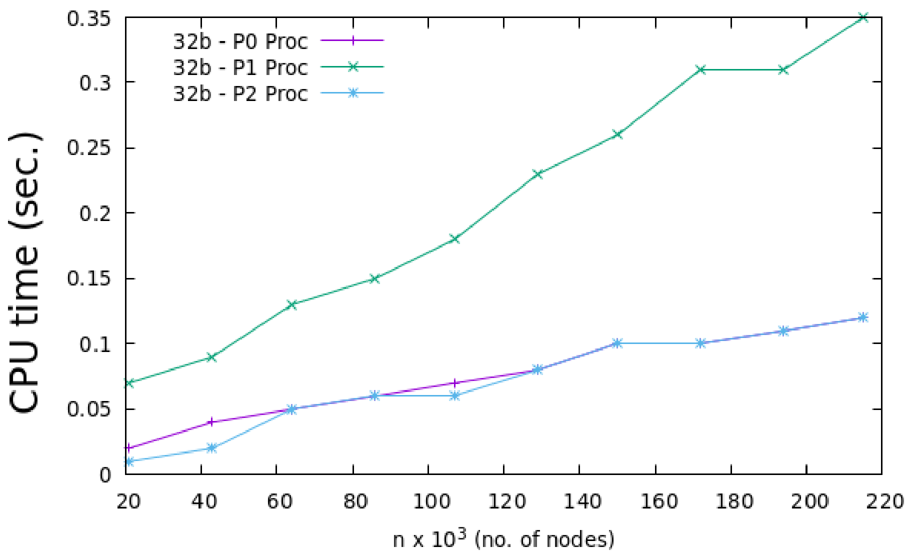

Figure 4.2: Proc. CPU time vs node count

Figure (4.3) shows the performance for algorithm PPFR–SI-PreProc (see alg. (4)). This is a one-time operation only so it does not affect most of the queries. PPFR–SI requires that this protocol is run at least once and then PPFR–SI could be run as many times.

Figure (4.4) shows wall clock time comparison between Homomorphic Encryption-based protocol PPFR and PPFR–SI. It is for a single query. Public key size (N) for PPFR was 1024, while for PPFR–SI the share size was 32 bits. As can be seen in the figure, PPFR–SI significantly outperforms PPFR as the query time is under 0.2 seconds throughout the values of n used. PPFR takes about 2861 seconds to compute over 215678 nodes.

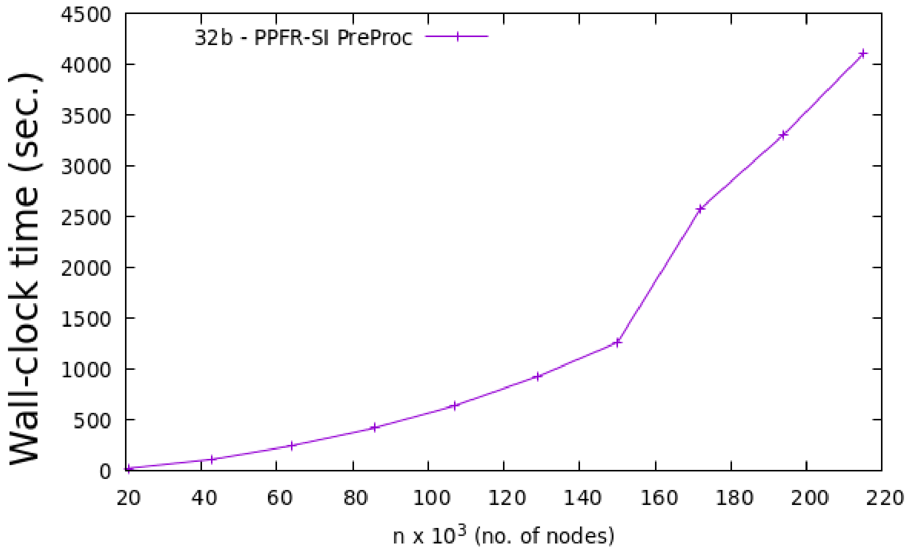

Figure 4.3: Pre-Proc Wall time vs node count

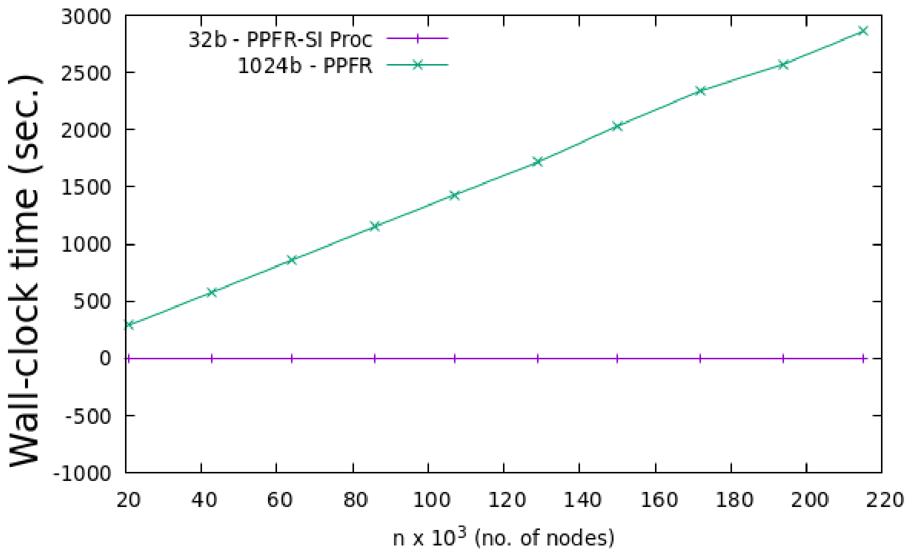

Figure 4.4: Comparing performance between PPFR–SI and PPFR

Chapter 5

PPFR–AL: Efficient PPFR over Adjacency List using Secret-Sharing

In the previous sections, we saw the protocols - PPFR, PPFR–SI. PPFR was designed using Homomorphic Encryption as the building block. It was computed over adjacency list. Then we saw the PPFR–SI algorithm, which used additive secret sharing as the building block and ran faster that PPFR. Although it was designed to compute over the adjacency matrix and required a pre-processing algorithm (PPFR–SI-PreProc), which was to be run at the start, atleast once. In this section we propose an algorithm which is linear in terms of number of nodes n as complexity and operates on the adjacency list representation.

We present a novel, privacy-preserving way to compute mutual neighbor count for two vertices in a graph over three parties. The method is realized using the secret-sharing, the secure multi-party computation technique (SMCs). The logic is based on obtaining the cardinality of a set intersection operation in a privacy-preserving way.

5.1 The Proposed Protocol

5.1.1 Problem Addressed

Let there be three parties - P_0, P_1, P_2. P_1 owns its private graph G containing vertices or nodes (V) and edges or relationships (E) among the nodes. P_2 would like to know the mutually adjacent node count (s) for any two nodes, say u_x, and u_y, both of which may or may not be in G, but computed in collaboration with P_1.

However, there are a few privacy constraints in this problem scenario, such as:

1. P_1 would like to keep its graph (G) private or atleast have an upper bound (ϵ) on the amount of information leaked from the final count score s
2. P_2 would like to keep its query nodes' ids (u_x, u_y) private
3. P_2 would like to keep the final score (s) private

5.1.2 Protocols In Brief

In this section, we propose the protocols that enable the functionality of Privacy-Preserving Mutual Neighbor Count Computation (PP-MNCC) for two given vertices in a given graph.

Privacy-Preserving Mutual Neighbor Count Computation (PP-MNCC) Protocol

PP-MNCC requires three parties for its evaluation. Here, two parties P_1 and P_2 provide their private inputs, while an additional third party (P_0) assists them to reduce the complexity of overall computation.

In order to evaluate PP-MNCC, P_1 uses its private input graph and P_2 uses its private input query consisting of a pair of vertices, which may or may not be a part of P_1's graph. At the end of the PP-MNCC protocol, P_2 is able to reconstruct the count of mutual neighbors for both of its input vertices. The third party (P_0) assists the computation but it cannot learn any information from the randomized data, that it is exposed to, during the computation.

With respect to the privacy aspect of PP-MNCC only, firstly, P_2's private input query nodes are not leaked. Secondly, P_1 cannot determine the score, s, reconstructed by P_2. However, P_2 can learn about P_1's graph data based on the output score (s). If the ideal Trusted Third Party model (TTP) is used instead of the secret-sharing approach provided here, the same information would have been leaked. Thus, the information leaked using simply the output score is not an issue for the PP-MNCC protocol. However, to overcome this shortcoming, we protect P_1's graph data to an extent provided by the noise addition mechanism of the edge-based ϵ-Differential Privacy.

Logically, PP-MNCC is based on the following simple steps:

1. Identify records corresponding to both the query vertices
2. Construct a set of adjacent nodes for each of those records
3. Compute a set-intersection for these sets
4. Report the cardinality of the resultant intersection set

5.1.3 Notations

Table 2.1 contains notations that would be used in the following sections.

5.1.4 Defining Structures and Constructs

Here we discuss about the constructs that form the basis of the algorithms proposed. They consist mainly of basic structures and their respective notations.

Let graph $G = \{V, E\}$, where, V is a set of vertices (u_i) and E is a set of edges such that edge $e_{i,j}$ ($= e_{j,i}$) indicates some relationship between entity i and entity j, such that $i \neq j$, in a non-looped and undirected graph. These entities are referred, across parties participating in SMC, using some public pseudo-ids denoted by u_i. u_i could be numeric ids, email id hashes, social media handle hashes, ORCIDs etc.

With the aid of examples from figures 5.1 and 5.2 we highlight the terminology usage undertaken henceforth in this work.

Symbol	Constraints	Description
u_i	$u_i \in \mathbb{Z}_Q$	Vertex (node) pseudo id
V	$V = \{u_i \mid u_i \in \mathbb{Z}_Q\}$	Set of vertices
$e_{i,j}$	$e_{i,j} = \{u_i, u_j\} \mid$ $u_i \neq u_j \in V$	u_i and u_j are related
E	$E = \{e_{i,j} \mid u_i, u_j \in V\}$	Set of edges
G	$G = \{V, E\}$	Network graph
V_{d+}	$V_{d+} \subseteq V, \deg(u_i) \geq d$	Vertices with minimum degree d
u_x, u_y	$u_x, u_y \overset{?}{\in} V$	Query node ids for x and y
L_i	$L_i = \{u_j \mid (e_{i,j} \in E)\}$	Neighbor node list for u_i
L	$L = \{L_i \mid \forall u_i \in V\}$	Adjacency list for graph G
$\vec{k}^x$	$k_i^x \in \{0,1\}$	Query index vector for u_x
$\vec{\omega}^x$	$\omega_i^x \in \{0,1\}$	Neighborhood vector for u_x
$\vec{\sigma}^{x,y}$	$\sigma_i^{x,y} \in \{0,1\}$	Mutual neighbors vector of u_x and u_y
$\rho^{x,y}$	$0 \leq s \leq \min(t^x, t^y)$, $s = \rho^{x,y}, t^k = \deg(u_k)$	Mutual neighbor count of u_x, u_y
P_1		Server
P_2		Client
P_0		Facilitator, third party
$[x]_Q^{P_l}$	$[x]_Q^{P_l} \in_R \mathbb{Z}_Q, l \in \{0,1,2\}$	Secret share of x for P_l
G_{P_1}		Party P_1's private graph G

Table 5.1: Notations for PPFR–AL Protocol

Adjacency List (L)

$$L_i = \{u_j | (e_{i,j} \in E \ \& \ u_i, u_j \in V \ \& \ i \neq j)\} \tag{5.1}$$

$$L = \{L_i | \forall u_i \in V\} \tag{5.2}$$

L represents the adjacency list for the graph G. L_i provides a set of entities adjacent (neighboring) to u_i in G. L could be represented as per equation 5.2, e.g. figure 5.1.

Query Vector ($\vec{k}^x$)

$$k_i^x = \begin{cases} 1 & i = x, u_i \in V \\ 0 & i \neq x, u_i \in V \end{cases} \tag{5.3}$$

Given u_x, vector $\vec{k}^x$ is a *query vector* that is represented as a binary vector. All the vertices in V have a corresponding component in $\vec{k}^x$. k_i^x represents the value of ith component (that represents u_i) in $\vec{k}^x$ and indirectly represents boolean value of $u_x \overset{?}{=} u_i$. 1 is set for the component corresponding to query id (item) u_x and 0 otherwise, see equation (5.3).

Neighborhood Vector ($\vec{\omega}^x$)

$$\omega_i^x = \begin{cases} 1 & e_{x,i} \in E, x \neq i \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Similar to *query vector*, given u_x, vector $\vec{\omega}^x$ is a *neighborhood* or *relationship vector* that is represented as a binary vector. 1 is set for the component corresponding to an item (or user id) u_y in the case where a relationship exists between u_x and u_y, $x \neq y$, or 0 is set otherwise, see equation 5.4.

Mutual Neighbor Vector ($\vec{\sigma}^{x,y}$)

$$\sigma_j^{x,y} = \omega_j^x \cdot \omega_j^y \tag{5.5}$$

Given neighbor vectors $\vec{\omega}^x$ and $\vec{\omega}^y$ for nodes u_x and u_y, respectively. $\sigma_j^{x,y} = 1$ or non-zero indicates that u_j is a neighbor of both u_x and u_y, i.e. a mutual neighbor. If $\sigma_j^{x,y} = 0$, then u_j is strictly a neighbor of one of them or none of them.

Mutual Neighbor Count ($\rho^{x,y}$ or $s^{x,y}$)

$$\rho^{x,y} = s^{x,y} = \sum_{j \leftarrow 1}^{|\vec{\sigma}^{x,y}|} \sigma_j^{x,y} \tag{5.6}$$

Given mutual neighbor vector ($\vec{\sigma}^{x,y}$), compute mutual neighbor count by adding up all the components in the given vector ($\vec{\sigma}^{x,y}$). This is also represented using the variable s.

Query List (Λ^x)

$$\Lambda_i^x = \{k_j^x | (e_{i,j} \in E \ \& \ u_i, u_j \in V \ \& \ i \neq j)\} \tag{5.7}$$

Given $\vec{k}^x$ and L, Λ^x indicates that the list structure is obtained by simply replacing u_i with k_i^x within L. It can be observed that Λ_i^x will have exactly one entry or item as 1 if u_i is a neighbor of u_x in G and provided u_x and u_i are distinct.

Secret Shared Lists and Vectors

For each of the lists and vectors specified in the earlier sections, here we provide the notations used for their secret shared representations for some generic party P_l, where $l \in \{0, 1, 2\}$.

1. Adjacency List Share $\left([L]_Q^{P_l}\right)$ -

$$[L_i]_Q^{P_l} = \left\{[u_j]_Q^{P_l} \,|(e_{i,j} \in E \;\&\; u_i, u_j \in V \;\&\; i \neq j)\right\}$$

$$[L]_Q^{P_l} = \left\{[L_i]_Q^{P_l} \,|\forall u_i \in V\right\}$$

2. Query Vector Share $\left(\left[\vec{k}^x\right]_Q^{P_l}\right)$ -

$$[k_i^x]_Q^{P_l} = \begin{cases} [1]_Q^{P_l} & i = x, u_i \in V \\ [0]_Q^{P_l} & i \neq x, u_i \in V \end{cases}$$

3. Neighborhood Vector Share $\left([\vec{\omega}^x]_Q^{P_l}\right)$ -

$$[\omega_i^x]_Q^{P_l} = \begin{cases} [1]_Q^{P_l} & e_{x,i} \in E, x \neq i \\ [0]_Q^{P_l} & \text{otherwise} \end{cases}$$

4. Query List Share $\left([\Lambda^x]_Q^{P_l}\right)$ -

$$[\Lambda_i^x]_Q^{P_l} = \left\{\left[k_j^x\right]_Q^{P_l} \,|(e_{i,j} \in E \;\&\; u_i, u_j \in V \;\&\; i \neq j)\right\}$$

$$L = \begin{cases} L_a : & \{u_b, u_c, u_d\} \\ L_b : & \{u_a\} \\ L_c : & \{u_a, u_d\} \\ L_d : & \{u_a, u_c\} \\ L_e : & \phi \end{cases} \tag{5.8}$$

Figure 5.1: Example: Adjacency List (L)

$$\vec{k}^a = \{1,0,0,0,0\}$$
$$\vec{k}^d = \{0,0,0,1,0\}$$

(a) Query Vector ($\vec{k}^x$)

$$\vec{\omega}^a = \{0,1,1,1,0\}$$
$$\vec{\omega}^d = \{1,0,1,0,0\}$$

(b) Neighborhood Vector ($\vec{\omega}^x$)

Figure 5.2: Example: Query and Neighborhood Vectors

$$\Lambda^a = \begin{cases} \Lambda^a_a : & \{0,0,0\} \\ \Lambda^a_b : & \{1\} \\ \Lambda^a_c : & \{1,0\} \\ \Lambda^a_d : & \{1,0\} \\ \Lambda^a_e : & \phi \end{cases}$$

(a) Query List (Λ^a)

$$\Lambda^d = \begin{cases} \Lambda^d_a : & \{0,0,1\} \\ \Lambda^d_b : & \{0\} \\ \Lambda^d_c : & \{0,1\} \\ \Lambda^d_d : & \{0,0\} \\ \Lambda^d_e : & \phi \end{cases}$$

(b) Query List (Λ^d)

Figure 5.3: Example: Query List (Λ^x)

$$\left[\vec{k}^a\right]^{P_l}_Q = \left\{[1]^{P_l}_Q, [0]^{P_l}_Q, [0]^{P_l}_Q, [0]^{P_l}_Q, [0]^{P_l}_Q\right\}$$
$$\left[\vec{\omega}^a\right]^{P_l}_Q = \left\{[0]^{P_l}_Q, [1]^{P_l}_Q, [1]^{P_l}_Q, [1]^{P_l}_Q, [0]^{P_l}_Q\right\}$$
$$\left[\vec{\sigma}^{a,d}\right]^{P_l}_Q = \left\{[0]^{P_l}_Q, [0]^{P_l}_Q, [1]^{P_l}_Q, [0]^{P_l}_Q, [0]^{P_l}_Q\right\}$$

(a) Vector Share Notation for Party P_l

$$\left[\Lambda^d\right]^{P_l}_Q = \begin{cases} \left[\Lambda^d_a\right]^{P_l}_Q : & \left\{[0]^{P_l}_Q, [0]^{P_l}_Q, [1]^{P_l}_Q\right\} \\ \left[\Lambda^d_b\right]^{P_l}_Q : & \left\{[0]^{P_l}_Q\right\} \\ \left[\Lambda^d_c\right]^{P_l}_Q : & \left\{[0]^{P_l}_Q, [1]^{P_l}_Q\right\} \\ \left[\Lambda^d_d\right]^{P_l}_Q : & \left\{[0]^{P_l}_Q, [0]^{P_l}_Q\right\} \\ \left[\Lambda^d_e\right]^{P_l}_Q : & \phi \end{cases}$$

(b) Query List Share Notation ($\left[\Lambda^d\right]^{P_l}_Q$) for Party P_l

Figure 5.4: Example: Vectors and Lists represented as secret shares

5.1.5 Protocol Ordering and Initialization

Algorithm PP-MNCC (alg. 9) is the main algorithm that provides the functionality of Privacy-Preserving Mutual Neighbor Count Computation. PP-MNCC is based upon three sub-algorithms that help in *Query Vector Computation* ($\vec{k}^x$, $\vec{k}^y$), *Neighborhood Vector Computation* ($\vec{\omega}^x$, $\vec{\omega}^y$) and *Beaver Triples Multiplication* ($\vec{\sigma}^{x,y}$).

The Query Vector Computation protocol has two variants based on the number of parties, which is based on the pseudo-id assumption used. Namely, those protocols are PP$_2$-QVC (alg. 6) and PP$_3$-QVC (alg. 7) for two and three parties, respectively. PPFR–AL (alg. 8) performs the neighborhood vector computation. PP-BM is the Beaver Triples-based Multiplication proposed by [81] and as applied by [82]. Given private inputs $[x]_Q^{P_0}$ and $[y]_Q^{P_0}$ owned by P_0 and private inputs $[x]_Q^{P_1}$ and $[y]_Q^{P_1}$ owned by P_1, PP-BM provides private outputs $[x \cdot y]_Q^{P_0}$ to P_0 and $[x \cdot y]_Q^{P_1}$ to P_1, respectively, without disclosing any information about x, y, and $(x \cdot y)$ to either parties. PP-BM requires a third party in order to generate random Beaver triples for performing the computation.

5.1.6 Two Party Query Vector Computation (Alg. PP$_2$-QVC)

Algorithm PP$_2$-QVC, see alg. (6), is a two party protocol that computes query vector $\left(\vec{k}^x\right)$, given u_x, in a privacy-preserving way. P_1 and P_2 are the parties involved in this protocol. G_S and u_x are the private inputs of P_1 and P_2, respectively. P_2 obtains private $\left(\vec{k}^x\right)$ after successful execution of PP$_2$-QVC.

PP$_2$-QVC using an example

Table 5.2 and 5.3 provide the state of variables for PP$_2$-QVC. The example is the same as in fig. (5.1) on which P_1's private graph G_S is based, P_2's private queries are u_a and u_d.

PP$_2$-QVC step-wise details

Following are the details of the steps provided in algorithm PP$_2$-QVC (alg. 6):

Algorithm 6 PP$_2$-QVC $(\langle P_1, G_S\rangle, \langle P_2, u_x\rangle) \rightarrow \left(\langle P_1, \phi\rangle, \langle P_2, \vec{k}^x\rangle\right)$

```
Require: Index $i$ varies from 1 to $n$ where $n = |V|$ at the $P_1$. Select $Q$ as a large
    prime. All computations are modulo $Q$ i.e. from the finite field $\mathbb{Z}_Q$.
1: $P_1$:
     (a) Generate non-zero random $r_i$, such that, $r_i \in_R \mathbb{Z}_N$
     (b) Compute $\alpha_i \leftarrow (u_i \cdot r_i)$
     (c) Send $\alpha_i$ to $P_2$
2: $P_2$:
     (a) Receive $\alpha_i$ from $P_1$
     (b) Compute $\beta_i \leftarrow \alpha_i \cdot (u_x^{-1})$
     (c) Generate non-zero random $s_i$, such that, $s_i \in_R \mathbb{Z}_N$
     (d) Compute $\delta_i \leftarrow (\beta_i \cdot s_i)$
     (e) Send $\delta_i$ to $P_1$
3: $P_1$:
     (a) Receive $\delta_i$ from $P_2$
     (b) Compute $\eta_i \leftarrow \delta_i \cdot \left(r_i^{-1}\right)$
     (c) Send $\eta_i$ to $P_2$
4: $P_2$:
     (a) Receive $\eta_i$ from $P_1$
     (b) Compute $\theta_i \leftarrow \eta_i \cdot \left(s_i^{-1}\right)$
     (c) Assign $k_i^x$ as:
         $$k_i^x = \begin{cases} 1 & \text{if } \theta_i = 1 \\ 0 & \text{otherwise} \end{cases}$$
```

V	u_i	r_i	α_i	u_x^{-1}	β_i	s_i	δ_i
a	17	4	25	29	37	31	29
b	34	18	10	–	32	30	14
c	18	42	25	–	37	27	10
d	3	13	39	–	13	41	17
e	5	6	30	–	10	21	38

Table 5.2: Alg. PP$_2$-QVC Variable States: $Q = 43$, $x = d$ (1/2)

r_i^{-1}	η_i	s_i^{-1}	θ_i	k_i^x
11	18	25	20	0
12	39	33	40	0
42	33	8	6	0
10	41	21	1	1
36	35	41	16	0

Table 5.3: Alg. PP$_2$-QVC Variable States: $Q = 43$, $x = d$ (2/2)

St 1: P_1 randomizes its nodes' pseudo-ids and stores them as α_i. After randomizing α_i is sent to P_2.

St 2: P_2 wants to keep its query node id u_x private so obtains its multiplicative inverse modulo Q from $\mathbb{Z}_Q$. P_2 computes β_i as a product of received α_i and u_x^{-1}. It then randomizes β_i using randomly generated s_i from $\mathbb{Z}_Q$. The product δ_i is then sent to P_1. For the correct index k, such that $k = x$, value of δ_k would be $(1 \cdot r_k \cdot s_k)$.

St 3: In this step, P_1 receives δ_i and removes the randomization that it introduced by obtaining a product of inverse of r_i (i.e. r_i^{-1}) and δ_i. This is called η_i which is sent to P_2.

St 4: Similar to earlier step, P_2 receives η_i and removes the randomization effect of s_i to obtain θ_i. At index location k, such that $k = x$, $\theta_i = 1$ while rest locations would be ratios of ids.

5.1.7 PP$_3$-QVC: For Three Party Query Vector Computation

Algorithm PP$_3$-QVC (alg. 7) is a three party protocol that computes the query vector $\left(\vec{k^x}\right)$ in a privacy-preserving way. u_x and G_S are the private inputs of P_2 and P_1, respectively. P_2 obtains private $\left(\vec{k^x}\right)$ after successful execution of PP$_3$-QVC.

PP$_3$-QVC using an example

Table (5.4) and (5.5) provide a brief instance of the protocol variables' values for the example shown in figure (5.1) for $Q = 43$ and $x = d$.

Algorithm 7 PP$_3$-QVC $(\langle P_0, \phi\rangle, \langle P_1, G_S\rangle, \langle P_2, u_x\rangle) \rightarrow \Big(\langle P_0, \phi\rangle, \langle P_1, \phi\rangle, \langle P_2, \vec{k}^x\rangle\Big)$

```
Require: The index $i$ varies from 1 to $|V|$. Select $N$ as a large prime. All computations are modulo $N$ i.e. from the finite field $\mathbb{Z}_N$.
1: $P_1$:
    (a) Generate shares $[u_i]_Q^{P_0}$ and $[u_i]_Q^{P_1}$ from $u_i$
    (b) Generate random numbers $r_i$ such that $r_i \in_R \mathbb{Z}_Q$
    (c) Send $r_i$ and $[u_i]_Q^{P_0}$ to $P_0$
2: $P_0$:
    (a) Receive $r_i$ and $[u_i]_Q^{P_0}$ from $P_1$
3: $P_2$:
    (a) Generate shares $[u_x]_Q^{P_0}$, $[u_x]_Q^{P_1}$ from $u_x$
    (b) Send $[u_x]_Q^{P_0}$, $[u_x]_Q^{P_1}$ to $P_0$, $P_1$, respectively
4: $P_l$: $l \in \{0, 1\}$
    (a) Receive $[u_x]_Q^{P_l}$ from $P_2$ as $c \leftarrow [u_x]_Q^{P_l}$
    (b) Compute $[\kappa_i]_Q^{P_l} \leftarrow r_i \cdot \Big(c - [u_i]_Q^{P_l}\Big)$
    (c) Send $[\kappa_i]_Q^{P_l}$ to $P_2$
5: $P_2$:
    (a) Receive $[\kappa_i]_Q^{P_0}$, $[\kappa_i]_Q^{P_1}$ from $P_0$, $P_1$, respectively
    (b) Construct $\kappa_i$ from shares $[\kappa_i]_Q^{P_0}$ and $[\kappa_i]_Q^{P_1}$
    (c) Assign $\vec{k}^x$, such that,
        $$k_i^x \leftarrow \begin{cases} 1 & \text{if } \kappa_i = 0 \\ 0 & \text{otherwise} \end{cases}$$
```

V	u_i	$[u_i]_Q^{P_0}$	$[u_i]_Q^{P_1}$	r_i
a	17	29	31	5
b	34	30	4	9
c	18	19	42	12
d	3	14	32	37
e	5	3	2	25

Table 5.4: Alg. PP$_3$-QVC Variable States: $Q = 43$, $x = d$ (1/2)

u_x	$[u_x]_Q^{P_0}$	$[u_x]_Q^{P_1}$	$[\kappa_i]_Q^{P_0}$	$[\kappa_i]_Q^{P_1}$	κ_i	k_i^x
3	8	38	24	35	16	0
–	–	–	17	5	22	0
–	–	–	40	38	35	0
–	–	–	36	7	0	1
–	–	–	9	40	6	0

Table 5.5: Alg. PP$_3$-QVC Variable States: $Q = 43$, $x = d$ (2/2)

5.1.8 The Main Protocol

Algorithm PP-MNCC, i.e. alg. (9), defines the functionality of the privacy-preserving mutual neighbor count s. P_1's adjacency list L (i.e. graph G) and P_2's query user ids u_x and u_y are their private inputs. Finally, on successful completion of the protocol only P_2 can obtain the score s. Party P_0 does not provide any of its private inputs neither can it deduce any information about L, $\langle u_x, u_y \rangle$ nor any information about s. P_0 generates random Beaver multiplication triples required for the privacy-preserving multiplication of secret shares.

Algorithm 8 PPFR–AL $\Big(\langle P_0, \phi\rangle, \langle P_1, L\rangle, \langle P_2, \vec{k}^x\rangle\Big) \rightarrow \Big(\langle P_0, [\vec{\omega}^x]_Q^{P_0}\rangle, \langle P_1, \phi\rangle, \langle P_2, [\vec{\omega}^x]_Q^{P_2}\rangle\Big)$

```
Require: $i$ s.t. $u_i \in V$, $j$ s.t. $u_j \in (V_{1+} + \Delta)$, $\Delta$ is set of randomly added fake
    isolated nodes. $V_{1+} \subseteq V$, $V_{1+}$ is a set of nodes with a degree of atleast 1. $Q$ is a
    large prime. All computations are modulo $Q$ i.e. from finite field $\mathbb{Z}_Q$.
1: $P_2$:
    (a) Generate shares $[k_i^x]_Q^{P_1}$ and $[k_i^x]_Q^{P_2}$ from $k^x$
    (b) Send $[k_i^x]_Q^{P_1}$ to $P_1$
2: $P_1$:
    (a) Receive $[k_i^x]_Q^{P_1}$ from $P_2$
    (b) Compute $\left[\Omega_j^x\right]_Q^{P_0}$ as per equation (5.9):
```

$$\left[\Omega_j^x\right]_Q^{P_0} \leftarrow \sum_{\forall u_i \in L_j} [k_i^x]_Q^{P_1} \tag{5.9}$$

```
    (c) Generate shares of 0 i.e. $[0_j]_Q^{P_0}$ and $[0_j]_Q^{P_2}$
    (d) Compute $\alpha_j \leftarrow \left[\Omega_j^x\right]_Q^{P_0} + [0_j]_Q^{P_0}$
    (e) Send $[0_j]_Q^{P_2}$ to $P_2$
    (f) Send $\alpha_j$ to $P_0$
3: $P_2$:
    1. Receive $[0_j]_Q^{P_2}$ from $P_1$
    2. Generate a random number $p$ s.t. $p \in_R \mathbb{Z}_Q$, $p \neq 0$
    3. Compute $p\,[k_i^x]_Q^{P_2} \leftarrow \left(p \cdot [k_i^x]_Q^{P_2}\right)$
    4. Send $p\,[k_i^x]_Q^{P_2}$ to $P_1$
```

```
4: $P_1$:
    1. Receive $p\,[k_i^x]_Q^{P_2}$ from $P_2$
    2. Compute $p\left[\Omega_j^x\right]_Q^{P_2}$ as per equation (5.10):
           $p\left[\Omega_j^x\right]_Q^{P_2} \leftarrow \sum_{\forall u_i \in L_j} p\,[k_i^x]_Q^{P_2}$    (5.10)
    3. Generate random numbers $r_j$ s.t. $r_j \in \mathbb{Z}_Q$, $r_j \neq 0$
    4. Compute $\beta_j \leftarrow p\left[\Omega_j^x\right]_Q^{P_2} + r_j$
    5. Send $\beta_j$ to $P_2$
    6. Send $r_j$ to $P_0$
5: $P_2$:
    1. Receive $\beta_j$ from $P_1$
    2. Compute $\left[\omega_j^x\right]_Q^{P_2} \leftarrow \beta_j + \left(p \cdot [0_j]_Q^{P_2}\right)$
    3. Send $p$ to $P_0$
6: $P_0$:
    1. Receive $\alpha_j$ and $r_j$ from $P_1$
    2. Receive $p$ from $P_2$
    3. Compute $\left[\omega_j^x\right]_Q^{P_0} \leftarrow \left(p \cdot [\alpha_j]_Q^{P_0}\right) - r_j$
```

PP-MNCC using an example

Following are the intermediate variable states for a brief example graph shown in figure (5.1). Here, $x = a$, $y = d$, and $Q = 43$.

1. P_0, P_1, P_2: Compute query vector $\vec{k}^a$ for id u_a

$$\vec{k}^a = \{1, 0, 0, 0, 0\}$$

2. P_0, P_1, P_2: Compute query vector $\vec{k}^d$ for id u_d

$$\vec{k}^d = \{0, 0, 0, 1, 0\}$$

Algorithm 9 PP-MNCC $(\langle P_0, \phi\rangle, \langle P_1, L\rangle, \langle P_2, u_x, u_y\rangle) \rightarrow \langle P_2, s\rangle$

```
Require: i s.t. $u_i \in V$, j s.t. $u_j \in (V_{1+} + \Delta)$, $\Delta$ is set of randomly added fake
    isolated nodes. $V_{1+} \subseteq V$, $V_{1+}$ is a set of nodes with a degree of atleast 1. $Q$ is a
    large prime. All computations are modulo $Q$ i.e. from finite field $\mathbb{Z}_Q$.
1: $P_0$, $P_1$, $P_2$: Compute query vector $\vec{k}^x$ for id $u_x$
       $$(\langle P_2, k_i^x\rangle) \leftarrow \text{PP}_3\text{-QVC}(\langle P_1, L\rangle, \langle P_2, u_x\rangle)$$
2: $P_0$, $P_1$, $P_2$: Compute query vector $\vec{k}^y$ for id $u_y$
       $$(\langle P_2, k_i^y\rangle) \leftarrow \text{PP}_3\text{-QVC}(\langle P_1, L\rangle, \langle P_2, u_y\rangle)$$
3: $P_0$, $P_1$, $P_2$: Compute neighbor vector $[\vec{\omega}^x]_Q^{P_l}, l \in \{0,2\}$
       $$\left(\left\langle P_0, [\vec{\omega}^x]_Q^{P_0}\right\rangle, \left\langle P_2, [\vec{\omega}^x]_Q^{P_2}\right\rangle\right) \leftarrow \text{PPFR–AL}\left(\langle P_1, L\rangle, \left\langle P_2, \vec{k}^x\right\rangle\right)$$
4: $P_0$, $P_1$, $P_2$: Compute neighbor vector $[\vec{\omega}^y]_Q^{P_l}, l \in \{0,2\}$
       $$\left(\left\langle P_0, [\vec{\omega}^y]_Q^{P_0}\right\rangle, \left\langle P_2, [\vec{\omega}^y]_Q^{P_2}\right\rangle\right) \leftarrow \text{PPFR–AL}\left(\langle P_1, L\rangle, \left\langle P_2, \vec{k}^y\right\rangle\right)$$
5: $P_0$, $P_1$, $P_2$: Compute mutual neighbor vector $[\vec{\sigma}^{x,y}]_Q^{P_l}, l \in \{0,2\}$
       $$\left(\left\langle P_0, [\vec{\sigma}^{x,y}]_Q^{P_0}\right\rangle, \left\langle P_2, [\vec{\sigma}^{x,y}]_Q^{P_2}\right\rangle\right) \leftarrow \text{PP-BM}\left(\left\langle P_0, [\vec{\omega}^x]_Q^{P_0}, [\vec{\omega}^y]_Q^{P_0}\right\rangle, \left\langle P_2, [\vec{\omega}^x]_Q^{P_2}, [\vec{\omega}^y]_Q^{P_2}\right\rangle\right)$$
6: $P_0$ and $P_2$: Compute mutual neighbor count $[\rho]_Q^{P_l}, l \in \{0,2\}$
       $$[\rho]_Q^{P_l} \leftarrow \sum_{j \leftarrow 1}^{|V_{1+}+\Delta|} \left[\sigma_j^{x,y}\right]_Q^{P_l}$$
7: $P_0$: Send $[\rho]_Q^{P_0}$ to $P_2$
8: $P_2$:
    (a) Receive $[\rho]_Q^{P_0}$ from $P_0$
    (a) Construct $\rho$ from shares $[\rho]_Q^{P_0}$, $[\rho]_Q^{P_2}$ and set $s \leftarrow \rho$
```

3. P_0, P_1, P_2: Compute neighbor vector $[\vec{\omega}^a]_Q^{P_l}, l \in \{0, 2\}$

$$[\vec{\omega}^a]_Q^{P_l} = \left\{[0]_Q^{P_l}, [1]_Q^{P_l}, [1]_Q^{P_l}, [1]_Q^{P_l}, [0]_Q^{P_l}\right\}$$

4. P_0, P_1, P_2: Compute neighbor vector $\left[\vec{\omega}^d\right]_Q^{P_l}, l \in \{0, 2\}$

$$\left[\vec{\omega}^d\right]_Q^{P_l} = \left\{[1]_Q^{P_l}, [0]_Q^{P_l}, [1]_Q^{P_l}, [0]_Q^{P_l}, [0]_Q^{P_l}\right\}$$

5. P_0, P_1, P_2: Compute mutual neighbor vector $\left[\vec{\sigma}^{a,d}\right]_Q^{P_l}, l \in \{0, 2\}$

$$\left[\vec{\sigma}^{a,d}\right]_Q^{P_l} = \left\{[0]_Q^{P_l}, [0]_Q^{P_l}, [1]_Q^{P_l}, [0]_Q^{P_l}, [0]_Q^{P_l}\right\}$$

6. P_0 and P_2: Compute mutual neighbor count $[\rho]_Q^{P_l}, l \in \{0, 2\}$

$$[\rho]_Q^{P_l} \leftarrow [1]_Q^{P_l} = \sum_{j \leftarrow 1}^{|V_{1+} + \Delta|} \left[\sigma_j^{a,d}\right]_Q^{P_l}$$

7. P_0: Send $[\rho]_Q^{P_0}$ to P_2

8. P_2:

   (a) Receive $[\rho]_Q^{P_0}$ from P_0

   (b) Construct ρ from shares $[\rho]_Q^{P_0}$, $[\rho]_Q^{P_2}$ and set $s \leftarrow 1 = \rho$

5.1.9 Complexity Analysis: The PPFR–AL Protocol

We provide the complexity analysis for the main protocol PPFR–AL in this sub-section. Protocol analysis is done in a similar way as described in section (4.1.4). The computation costs over the shares for addition, subtraction, share generation and reconstruction operations is represented by a, while p is the cost for multiplication of shares. The communication cost of sending a share over the network is represented by t, where t can be the number of bits required to represent the share. A detailed discussion about these costs has been provided earlier in the section (4.1.4) for PPFR–SI protocol.

Let l represent $|(V_{1+} + \Delta)|$ i.e. the summation of the cardinality of the set of nodes having at least one neighbor i.e. $|V_{1+}|$ and the cardinality of the set of fake nodes (Δ) added so as to hide the number of nodes with a degree of at least 1 or, in other words, the number of isolated nodes. $0 \leq |V_{1+}| \leq 2m$, where $m = |E|$. $|\Delta| \ll m$. Therefore, l has an upper bound of $3m$ as $0 \leq l \leq 3m$.

Complexity for PPFR–AL

Step	Third Party (P_0)		Server (P_1)		Client (P_2)	
	O	T_x	O	T_x	O	T_x
1					na	nt
2			$2(m+l)a$	$2lt$		
3					np	nt
4			$(2m+l)a$	$2lt$		
5					$la+lp$	1
6	$la+lp$					
Total	$l(a+p)$		$(4m+3l)a$	$4lt$	$(n+l)(a+p)$	$2nt+1$

Table 5.6: Computation and Communication Complexity of the PPFR–AL protocol

Table 5.6 provides the complexity information for PPFR–AL protocol. It is clear that the computation complexities of P_0, and P_1 are linear in terms of the number of edges m. While the computation and communication complexities for party P_2 is linear in the number of nodes i.e. n. The communication complexity for P_1 is linear in the number of edges m.

Using protocol PPFR–SI's complexity analysis, given in section (4.1.4), and the complexity analysis of PPFR–AL in this section, it can be seen that, for parties P_0 and P_1, both the protocols are linear in terms of number of nodes and edges, respectively. This is significantly better than for protocol PPFR, which is quadratic in the number of nodes (n), see section (3.1) for more details. The added benefit of PPFR–AL is that it does not require the one-time quadratic in n operation of PPFR–SI-PreProc, which is required by PPFR–SI. As compared to PPFR–SI, PPFR–AL is quite efficient especially when $|E| \ll |V|$.

Chapter 6

Privacy-Preserving Friend Recommendation based on Similar Profiles

6.1 Introduction

There is ample evidence to show that sharing information has great benefits to society across a wide range of sectors and applications. For instance, information sharing endeavors can support the scientific discovery process, enable governments to assess and refine policies, and enhance intelligence and national security. At the same time, our societal norms and legal boundaries provide for confidentiality and personal privacy rights, such that not all useful information is readily accessible. One way in which society has attempted to resolve this tension is through the adoption of laws that use a “minimum necessary” principle, which is typified in the U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA). Under this principle, (potentially sensitive) data can be shared with requesters (e.g., biomedical scientists), provided that it is restricted to minimum amount necessary to pursue an investigation. Yet, how can we engineer information management and retrieval systems to support such principles? The key to enabling and facilitating information sharing is to develop a framework under which people can identify and share only what is needed (based on information content) without disclosing what is not needed [83].

In this work, we address this issue through the integration of Lucene, a popular open source feature rich search engine library [84], with privacy-preserving functionality, the hybrid of which we refer to as Lucene-P^2. In addition to identifying the need-to-know, Lucene-P^2 can enable a large array of useful applications. For context, let us consider two driving examples. First, imagine a peer-reviewed journal (or research-oriented conference) receives a number of paper submissions. It is in the best interests of the editor to determine if any of the submissions contain plagiarized materials. In addition, the editor may want to know if any papers under reviews have been simultaneously submitted to other journals (i.e., the double submission problem). Since paper submissions are confidential, the editor cannot simply utilize the existing plagiarism detection systems to check for double submission. On the other hand, the Lucene-P^2 framework can directly be invoked to solve this problem.

Second, let us consider the medical domain, where an agency in control of drug safety aims to learn if a certain pharmaceutical that has been recently introduced into the market is inducing adverse events at a greater than expected rate, so that further investigative action can be instituted. Due to privacy and confidentiality issues, the events at disparate healthcare providers may not be shared freely [85]. This problem is exacerbated by the fact that without making any information available, the agency (or biomedical scientists more generally) will never know about the existence of such data. As a consequence, they will never ask permission to access it. We acknowledge that information sharing in the medical domain is subject to very strict guidelines, but it is possible if "Use or disclosure is sought solely to review protected health information as necessary to prepare a research protocol or for similar purposes preparatory to research" [86]. Thus, we believe that the Lucene-P^2 framework can be relied upon to justify this necessity without disclosing protected information.

6.1.1 Why Adopting Lucene?

Lucene [84] is a Java-based text indexing and search library that allows users to develop text-search based information retrieval applications. Lucene adopts the most widely used information retrieval models. Given a document collection, it can build an inverted index and produce text representations under the vector space model with Term Frequency - Inverse Document Frequency (tf-idf) information [87]. The Lucene library also implements commonly used similarity metric to measure information correlation which is more accurate than a simple cosine similarity and keyword based search model. To directly adopt Lucene in the proposed problem domain, the participating parties have to share their information one way or the other, causing privacy issues. In our proposed framework, Lucene is only used locally or independently by each party to produce the index structures, the vector space and the tf-idf frequency vectors. We also adopt Lucene's similarity metrics due to their proven effectiveness in text retrieval. Because of its widespread use, by building privacy-preserving features into the Lucene library, we believe this can facilitate user adoption of the proposed privacy enhancing technology.

6.2 Transformations for Cryptographic Processing

Here we illustrate how the normalization, scaling and floor transformations are performed on the Term-Document Matrix (TDM), where an entry TDM(i,j) represents the term-frequency-inverse-document frequency for term t_i in document d_j. Cryptographic operations

$$\begin{array}{c c}
\begin{array}{ccc} d_1 & d_2 & d_3 \end{array} & \\
\begin{bmatrix} 1.403 & 0.0741 & 0.6222 \\ 1.1636 & 0.8108 & 4.5333 \\ 3.2381 & 16.2 & 0.3333 \\ 0.7439 & 0.59 & 5.6667 \end{bmatrix} & \begin{array}{c} t_1 \\ t_2 \\ t_3 \\ t_4 \end{array}
\end{array}$$

Figure 6.1: Term-Document Matrix (TDM)

$$\begin{bmatrix} 0.3702 & 0.0046 & 0.0853 \\ 0.3071 & 0.05 & 0.6218 \\ 0.8545 & 0.9981 & 0.0457 \\ 0.1963 & 0.0363 & 0.7772 \end{bmatrix}$$

(a) Column-wise Normalization

$$\begin{bmatrix} 370.2 & 4.6 & 85.3 \\ 307.1 & 50.0 & 621.8 \\ 854.5 & 998.1 & 45.7 \\ 196.3 & 36.3 & 777.2 \end{bmatrix}$$

(b) Scaling-up, e.g., by factor 1000

$$\begin{array}{cc} \begin{array}{ccc} d_1 & d_2 & d_3 \end{array} & \\ \begin{bmatrix} 370 & 4 & 85 \\ 307 & 50 & 621 \\ 854 & 998 & 45 \\ 196 & 36 & 777 \end{bmatrix} & \begin{array}{c} t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} \end{array}$$

(c) Floor Transformation

Figure 6.2: Transformation stages:- (6.2a) NM, (6.2b) SM, (6.2c) FM

under the Paillier Cryptosystem require the plaintext to be of integer form. Figures 6.1 and 6.2 display the transformation process required in the proposed protocols. Column-wise normalization helps to eliminate the factor of document length, as shown in Figure 6.2a. The values lie in $[0, 1]$. The next step consists of scaling-up the numbers by a factor, say 1000, which depends on the number of dimensions or terms m. Factor is chosen in such a way that the final product should not cause an overflow. Scaling-up is shown in Figure 6.2b. The final step is simply obtaining the floor transformation or casting the data type from real to integer. It is important to note that since transformations are performed along document or query vectors, the relative ordering of similarity between documents would be unaffected.

Although not possible during the generation of document vectors, negative values may be produced during the singular value decomposition process. These negative values can be transformed by performing either the Translation of Axes or the Shifting of Origin method commonly used in coordinate geometry. The shifting parameter is based on the smallest (negative) number in the matrix.

6.3 Secure Correlation Computation

In this section, we develop a secure correlation computation (SCC) protocol for the Lucene-P^2 framework that computes scores according to Equation 2.2 with slight modifications:

- coord(q, d) - Since results from different queries are not compared, we can remove the normalizing factor in the common term function without affecting the rankings among the retrieved results.

- queryNorm(q) - This function returns a constant value when the input query is fixed. Thus, this function can be removed without affecting the relative ordering among the retrieved result.
- getBoost(t) - In Lucene, the getBoost function is generally set to 1 for every term. In this work, we will adopt the same output for the getBoost function.

According to the formulation given in Equation 1.2, the proposed SCC protocols are secure in the sense that:

- the query document q from P_a is not disclosed to P_b and
- P_b's document collection D is not disclosed to P_a.

The two terms in Equation 2.2, $\text{coord}(q, d)$ and $\sum_{t \in q} \text{weight}(t, d)$, have to be computed securely. Before detailing the proposed protocols, we next clarify how documents are represented.

6.3.1 Pre-Processing Stage

To be consistent with Lucene, both the query q and the document d are represented in the same global vector space G. We assume the size of G is bounded by m. The notations $\vec{q}$ and $\vec{d}$ are used to represent q and d under the vector space information retrieval model. These vectors can be binary or the tf-idf weighting. In addition, $\vec{q}[i]$ and $\vec{d}[i]$ represent the i^{th} components of $\vec{q}$ and $\vec{d}$ respectively. The following information needs to be computed beforehand:

- The global vector space G: is generated based on P_b's document collection and is shared with P_a.
- According to G, both P_a and P_b can generate the term frequency vectors $\vec{q}$, $\vec{d}$ and $\vec{d'}$

as follows:

$$\begin{aligned}\vec{q}[i] &\leftarrow \text{tf}(G[i], q) \overset{?}{>} 0 \\ \vec{d}[i] &\leftarrow \text{tf}(G[i], d) \overset{?}{>} 0 \\ \vec{d'}[i] &\leftarrow \text{weight}(G[i], d)\end{aligned}$$

where $\vec{q}$ and $\vec{d}$ are binary vectors under the vector space model; i.e., an entry value 1 means the corresponding term exists in the document and 0 otherwise. By contrast, $\vec{d'}$ contains the tf-idf related information based on Equation 2.3. These vectors are computed locally at each party, so no secure protocols are needed to perform these tasks.

6.3.2 The SCC Protocol

The design of SCC is based on the following observations:

- Although $\vec{d'}$ contains fraction values, these values can be scaled up by a constant factor to ensure that secure computations are done over the integer domain, which is required under the adopted encryption scheme. Note that since N is at least a 1,024-bit number, for most applications, the scaled values should still be bounded by N. For instance, suppose the scaled up numbers are 100-bit long (a very large number for most applications), the similarity scores are bounded by 200-bit more or less. Thus, scaling the values will extremely unlikely to cause inaccurate results due to exceeding the domain limit. Plus, we can always increase the size of N if needed.
- The common term function coord(q, d) is equivalent to $\vec{q} \bullet \vec{d}$, the dot product between $\vec{q}$ and $\vec{d}$.
- The summation part of the score function $\sum_{t \epsilon q}$ weight(t, d) is equivalent to $\vec{q} \bullet \vec{d'}$.

In the protocol, P_a has the private key pr of an HEnc scheme, while the public key pu is known to P_b. The private-public key pair can be generated by P_a.

The input for this algorithm comes from both parties P_a and P_b. P_a provides the query document q. The output consists of a set of similarity scores related to each document in

Algorithm 10 SCC$(\langle P_a, q\rangle, \langle P_b, D\rangle) \rightarrow s_1, \ldots, s_n$

```
Require: The global vector space $G$ and the size of the document collection $D$ are
    known to both parties; $q$ is $P_a$'s private input, and $D$ is $P_b$'s private input
1: $P_b$
        Compute $\vec{d_j}$ and $\vec{d'_j}$ based on $G$, where $n = |D|$, $m = |G|$, $1 \leq j \leq n$
2: $P_a$
    1. Generate a private-public key pair $(pr, pu)$ of an HEnc scheme and send $pu$
       to $P_b$
    2. Generate $\vec{q}$ based on $G$
    3. Compute $E_{pu}(\vec{q}[1]), \ldots, E_{pu}(\vec{q}[m])$ and send to $P_b$
3: $P_b$ $(j = 1$ to $n)$
    1. Compute $s_{j_1} \leftarrow \prod_{i=1 \wedge \vec{d_j}[i] \neq 0}^{m} E_{pu}(\vec{q}[i])$
    2. Compute $s_{j_2} \leftarrow \prod_{i=1}^{m} E_{pu}(\vec{q}[i])^{\vec{d'_j}[i]}$
4: $P_a$ and $P_b$ $(j = 1$ to $n)$
        $\langle P_a, s_j\rangle \leftarrow$ MP$(\langle P_a, pr\rangle, \langle P_b, pu, s_{j_1}, s_{j_2}\rangle)$, where $s_j = D_{pr}(s_{j_1}) \cdot D_{pr}(s_{j_2})$
5: $P_a$
        Send $s_1, \ldots, s_n$ to $P_b$
```

P_b's collection. The SCC protocol is a secure two-party protocol, i.e., it is run by both the parties P_a and P_b. Some steps of the protocol are run by a single party and some by both parties. Labels P_a and P_b, appearing next to the protocol step number, indicate which party is executing that step. The final step is a two-party protocol in itself and hence, is represented by the P_a and P_b label. Next, we step through the details of each individual sub-step presented in Algorithm 10.

- **Public input** - P_b, using Lucene, produces the global vector space G. We have removed the stop words and have used the Porter stemming algorithm [88] provided by Lucene. Once G is generated, it is sent to the querying party P_a when it requests to start the Lucene-P^2 system.

- **Step 1** - Based on G, P_b creates two vectors $\vec{d}$ and $\vec{d'}$, for every document in D, by relying upon a one-to-one mapping of vector dimensions and specifically ordered terms in the global vector space G. $\vec{d}$ and $\vec{d'}$ are the binary and the tf-idf document

Algorithm 11 $\text{MP}(\langle P_a, pr\rangle, \langle P_b, E_{pu}(x), E_{pu}(y)\rangle) \rightarrow \langle P_a, x \cdot y\rangle$

```
Require: $P_a$: private key $pr$; $P_b$: $E_{pu}(x)$ and $E_{pu}(y)$, $pu$ is the public key of an HEnc
1: $P_b$
      1. Randomly generate $r_1$ and $r_2$ from $Z_N$
      2. Compute $E_{pu}(x+r_1)$ and $E_{pu}(y+r_2)$ and send them to $P_a$
2: $P_a$
      1. Decrypt $E_{pu}(x+r_1)$ and $E_{pu}(y+r_2)$ to get $x+r_1$ and $y+r_2$
      2. Compute $E_{pu}(xy+xr_2+yr_1+r_1r_2)$ and send it to $P_b$
3: $P_b$
      1. Compute $E_{pu}(xr_2)$, $E_{pu}(yr_1)$ and $E_{pu}(r_1r_2)$
      2. Derive $E_{pu}(xy)$ and send it to $P_a$
4: $P_a$
      Decrypt $E_{pu}(xy)$ to get $xy$
```

vectors, respectively, as discussed in Section 6.3.1.

- **Step 2** - To execute the SCC protocol, P_a generates a private-public key pair (pr, pu) of an HEnc scheme, such as Paillier. At this point, P_a shares the public key pu with P_b. Once P_a receives the global vector space G from P_b, it constructs a binary document vector $\vec{q}$ for the query document q as described in Section 6.3.1. At the end of the step, P_a encrypts $\vec{q}$ component-wise using its public key pu and sends the encrypted query to P_b.

- **Step 3** - P_b generates $E_{pu}\left(\vec{q} \bullet \vec{d_j}\right)$ and $E_{pu}\left(\vec{q} \bullet \vec{d'_j}\right)$ denoted by s_{j_1} and s_{j_2} respectively.

- **Step 4** - Both parties utilize a well-known secure multiplication (MP) protocol (see Algorithm 11) to produce $\left(\vec{q} \bullet \vec{d_j}\right) \cdot \left(\vec{q} \bullet \vec{d'_j}\right)$ which is only known to P_a.

For completeness, the key steps of MP are provided in Algorithm 11. Given two ciphertexts $E_{pu}(x)$ and $E_{pu}(y)$ from P_b, the MP protocol computes and returns the plaintext $x \cdot y$ to P_a.

- **Step 1** - P_b generates two random numbers r_1 and r_2 from Z_N to randomize $E_{pu}(x)$ and $E_{pu}(y)$, respectively. The randomizations $E_{pu}(x+r_1)$ and $E_{pu}(y+r_2)$ are obtained by using the additive homomorphic property. $E_{pu}(x + r_1)$ and $E_{pu}(y + r_2)$ are then sent to P_a.

- **Step 2(1)** - P_a knows the private key pr. Thus, given a ciphertext $c = E_{pu}(m)$, where pu is the public key, P_a can compute its plaintext m by computing $m = E_{pr}(c)$. After obtaining $E_{pu}(x+r_1)$ and $E_{pu}(y+r_2)$, P_a decrypts them to get plaintexts $x+r_1$ and $y + r_2$.

- **Step 2(2)** - Once $x+r_1$ and $y+r_2$ have been derived, P_a multiplies them and encrypts the result using pu. $E_{pu}((x+r_1)\cdot(y+r_2)) = E_{pu}(xy+xr_2+yr_1+r_1r_2)$. The encrypted result is then sent to P_b.

- **Step 3(1)** - P_b knows $E_{pu}(x)$, $E_{pu}(y)$, r_1 and r_2. Using the additive homomorphic property, P_b computes:

$$\begin{aligned} E_{pu}(r_2 \cdot x) &= E_{pu}(x)^{r_2} \\ E_{pu}(r_1 \cdot y) &= E_{pu}(y)^{r_1} \\ E_{pu}(r_1 \cdot r_2) &= E_{pu}(r_1 \cdot r_2) \end{aligned}$$

- **Step 3(2)** - Again using the additive homomorphic property, P_b computes:

$$\begin{aligned} E_{pu}(-r_2 \cdot x) &= E_{pu}(r_2 \cdot x)^{N-1} \\ E_{pu}(-r_1 \cdot y) &= E_{pu}(r_1 \cdot y)^{N-1} \\ E_{pu}(-r_1 \cdot r_2) &= E_{pu}(r_1 \cdot r_2)^{N-1} \end{aligned}$$

  These values are then multiplied with $E_{pu}(xy + xr_2 + yr_1 + r_1r_2)$, obtained from P_a, to realize the encrypted form of $x \cdot y$. Then P_b sends $E_{pu}(xy)$ to P_a for decryption.

- **Step 4** - P_a decrypts the obtained ciphertext $E_{pu}(x \cdot y)$ using its private key pr to derive $x \cdot y$.

6.3.3 Security Analysis

In addition to the similarity scores, the proposed SCC protocol discloses the global vector space G and the size of P_b's document collection under the semi-honest model. The security of SCC can be proved using the simulation method in [14] according to Definition 1. First, we need to build a simulator based on the private input and output for each party. Since the computations between the two parties are asymmetric, the simulators are different for the individual parties. According to the SCC protocol, let Π_{P_a} and Π_{P_b} denote the real execution images for parties P_a and P_b respectively. Similarly, $\Pi^S_{P_a}$ and $\Pi^S_{P_b}$ denote the simulated execution images. Next, we show how to construct a simulator Simulator-$\Pi^S_{P_a}$ to produce $\Pi^S_{P_a}$.

Simulator-$\Pi^S_{P_a}$

Algorithm 12 Simulator-$\Pi^S_{P_a}(pu, s_1, \dots, s_n)$

```
Require: $pu$ is the public key of Paillier, $s_1, \dots, s_n$ are the output from SCC
1: For $1 \le j \le n$, randomly generate $\alpha_j$ and $\beta_j$ from $Z_N$
2: For $1 \le j \le n$, compute $E_{pu}(\alpha_j)$, $E_{pu}(\beta_j)$ and $E_{pu}(s_j)$
3: Return the following $(1 \le j \le n)$
     • $X' \sim \langle E_{pu}(\alpha_j), \alpha_j \rangle$
     • $Y' \sim \langle E_{pu}(\beta_j), \beta_j \rangle$
     • $Z' \sim \langle E_{pu}(s_j), s_j \rangle$
```

For a two-party distributed protocol, private information can be disclosed from the messages exchanged during the execution of the protocol. Party P_a receives messages from P_b at step 4 of the SCC protocol which translates into the steps 1(2) and 3(2) of the MP protocol. More specifically, P_a receives the following messages: $E_{pu}(x_j + r_{j_1})$, $E_{pu}(y_j + r_{j_2})$ and $E_{pu}(x_j y_j)$, where x_j and y_j represent the underlying values of s_{j_1} and s_{j_2} computed at step 3 of Algorithm 10 and the r values are randomly generated for each pair of x_j and y_i. Thus, the real execution image Π_{P_a} consists of the following information $(1 \le j \le n)$:

- $X \sim \langle E_{pu}(x_j + r_{j_1}), x_j + r_{j_1} \rangle$

- $Y \sim \langle E_{pu}(y_j + r_{j_2}), y_j + r_{j_2} \rangle$
- $Z \sim \langle E_{pu}(x_j y_j), x_j y_j \rangle$

where X, Y and Z denote the random variables related to the corresponding pair of values. For each pair, the first component is the message received, and the second component is the value derived from the first component. The simulator needs to simulate these messages and the information derived from them. Algorithm 12 provides the key steps for Simulator-$\Pi_{P_a}^S$.

Claim 3. *$\Pi_{P_a}^S$ generated from Simulator-$\Pi_{P_a}^S$ is computationally indistinguishable from Π_{P_a}.*

Proof. Suppose the claim is not true, then $\Pi_{P_a}^S$ is computationally distinguishable from Π_{P_a}. This implies three possibilities: X' is computationally distinguishable from X, Y' is computationally distinguishable from Y, or Z' is computationally distinguishable from Z. If X' is computationally distinguishable from X, then $\langle E_{pu}(\alpha_j), \alpha_j \rangle$ is computationally distinguishable from $\langle E_{pu}(x_j + r_{j_1}), x_j + r_{j_1} \rangle$. This implies that either $E_{pu}(\alpha_j)$ is distinguishable from $E_{pu}(x_j + r_{j_1})$ or α_j is distinguishable from $x_j + r_{j_1}$. Since both α_j and $x_j + r_{j_1}$ are uniformly random in Z_N and indistinguishable, it must be that case that $E_{pu}(\alpha_j)$ is computationally distinguishable from $E_{pu}(x_j + r_{j_1})$. However, this contradicts the fact that the Paillier encryption scheme is semantically secure or computationally indistinguishable [70]. Therefore, X' must be computationally indistinguishable from X. Adopting the same argument, we can prove that Y' must be computationally indistinguishable from Y. Regarding Z and Z', since $s_j = x_j y_j$, in order for both to be distinguishable, it must be that case that $E_{pu}(s_j)$ is distinguishable from $E_{pu}(x_j y_j)$. Again, this contradicts the security guarantee of Paillier. In conclusion, $\Pi_{P_a}^S$ is computationally indistinguishable from Π_{P_a}. $\square$

The above claim demonstrates that fact that any information that P_a learned during the execution of the SCC protocol can be derived by its private input and output. Thus, from P_b's perspective, the protocol is computationally secure. Next we need to prove the protocol is secure from P_a's perspective by building a simulator to simulate P_b's execution image.

Simulator-$\Pi^S_{P_b}$

Algorithm 13 Simulator-$\Pi^S_{P_b}(D, pu)$

```
Require: $D$ is the document collection and $pu$ is the public key of Paillier
1: Randomly generate $Q_1, \ldots, Q_m$ from $Z_{N^2}$
2: For $1 \le j \le n$, randomly generate $R_j$ from $Z_{N^2}$
3: Return the following $(1 \le j \le n)$
       • $X' \sim Q_1, \ldots, Q_m$
       • $Y' \sim R_j$
```

Party P_b receives messages from P_a at steps 2(3) and 4 of the SCC protocol. The step 4 further translates into the step 2(2) of the MP protocol. Thus, the real execution image Π_{P_b} consists of the following information $(1 \le j \le n)$:

- $X \sim E_{pu}(\vec{q}[1]), \ldots, E_{pu}(\vec{q}[m])$
- $Y \sim E_{pu}(x_j y_j + x_j r_{j_2} + y_j r_{j_1} + r_{j_1} r_{j_2})$

where X and Y denote the random variables related to the corresponding encrypted values. There is a main difference of the real execution images of P_a and P_b where Π_{P_a} additionally includes the information derived from its received messages since decryptions are performed on the message. On the other hand, P_b never decrypts any values, so there is no explicit information derived from its received messages. Algorithm 13 provides the key steps for Simulator-$\Pi^S_{P_b}$.

Claim 4. *$\Pi^S_{P_b}$ generated from Simulator-$\Pi^S_{P_b}$ is computationally indistinguishable from Π_{P_b}.*

Proof. Suppose the claim is not true, then this implies two possibilities: X' is computationally distinguishable from X, or Y' is computationally distinguishable from Y. If X' is computationally distinguishable from X, then Q_i is computationally distinguishable from $E_{pu}(\vec{q}[i])$. However, this contradicts the fact that the Paillier encryption scheme is semantically secure or computationally indistinguishable [70]. Therefore, X' must be computationally indistinguishable from X. Applying the same argument, we can prove that Y' must be computationally indistinguishable from Y. As a consequence, $\Pi^S_{P_b}$ is computationally indistinguishable from Π_{P_b}. □

The above claim demonstrates that fact that any information that P_b learned during the execution of the SCC protocol can be derived by its private input. Thus, from P_a's perspective, the protocol is computationally secure. Combining both claims, we can conclude that the SCC protocol is computationally secure as long as the parties follow the prescribed steps of the protocol.

6.3.4 Complexity Analysis

To analyze the computational cost of an SMC protocol, it is common to use the number of encryptions as a basis since the encryption/decryption operation is generally the most expensive basic operation. We will also include exponentiation of a ciphertext as a basic operation. Since the computations are asymmetric between P_a and P_b, we estimate the costs according to each individual party.

For the MP protocol, the cost at step 1(2) is equivalent to two encryptions. Because the encryption and decryption costs are close for the Paillier encryption scheme adopted in this work, we treat the cost of a decryption operation the same as that of an encryption operation. Therefore, the cost at step 2 or step 3 is about three encryptions, and the cost at step 4 is about one encryption. Overall, the computational cost of MP for P_b is 5 encryption operations, and the computational cost for P_a is 4 encryption operations. Let t denote the number of bits needed to represent each encrypted value. Then for the MP protocol, P_b needs to send $3t$ bits and P_a needs to send t bits.

At step 2 of the SCC protocol, P_a needs to perform m encryption operations. Based on the previous analysis, at step 4, P_a needs to perform $4n$ encryption operations. Thus, the computational cost for P_a is about $m + 4n$ encryptions, and the computational cost for P_b is about $5n$ encryptions and mn encryption exponentiations. The communication complexity is bounded by $(m + n)t$ bits and $3nt$ bits for P_a and P_b respectively. We count a pair of send and receive as one round of communication. The SCC protocol requires two rounds of communication between P_a and P_b.

6.4 The SCC Protocol with LSI

As our empirical results illustrate in Section 6.5, the proposed SCC protocol may not be sufficiently efficient for a real-time application. Thus, in this section, we investigate strategies to improve the computational efficiency with an acceptable level of privacy guarantee. Since the global vector space G can be very large, one potential opportunity to improve the computational efficiency of SCC is to reduce the number of dimensions in G without affecting the accuracy of the retrieved results. In this section, we discuss a commonly used dimension reduction technique and how it can be securely adopted by SCC.

6.4.1 Dimension Reduction with LSI

Latent Semantic Indexing (LSI) is based on various linear algebraic concepts such as matrix decomposition, Singular Value Decomposition (SVD), and low-rank matrix approximations [87]. It has been widely used in information retrieval to improve result accuracy, so it is applicable for all text-based information retrieval applications. A document collection can be represented using a matrix structure, where the rows correspond to the terms occurring in the collection and the columns corresponding to the individual document. An entry at index location (i, j) can be the tf-idf or frequency information about term i in document j. This representation is generally called a term by document matrix.

After LSI transformation, each document can be represented by a vector with an order of 50 to 150 values/dimensions. Thus, LSI offers an economical way of representing documents. The main step in LSI is SVD to produce a reduced term-document matrix which is used to reduce the dimensions of the documents in D and the user query. We refer to this matrix as the SVD matrix. According to Equation 2.2, we need two SVD matrices, $C_{[m,k]}$ and $W_{[m,k]}$, to transform the vectors related to $\text{coord}(t, d)$ and $\text{weight}(t, d)$ respectively.

Based on the document collection D, P_b is responsible for producing $C_{[m,k]}$ and $W_{[m,k]}$. In addition to $\vec{q}[i]$, $\vec{d}[i]$ and $\vec{d'}[i]$, we need another vector computed based on P_a's query:

$$\vec{q'}[i] \leftarrow \text{weight}(G[i], q)$$

To reduce the dimensions of the vectors, we treat each as a 1 by m matrix. Then the binary vectors are reduced by multiplying with the matrix $C_{[m,k]}$, while the term-frequency related vectors are reduced by multiplying with the matrix $W_{[m,k]}$. Let $\vec{q}_C$, $\vec{q}'_W$, $\vec{d}_C$ and $\vec{d}'_W$ denote the dimensionally-reduced vectors of $\vec{q}$, $\vec{q}'$, $\vec{d}$ and $\vec{d}'$ respectively. The similarity between q and d is then calculated with the following equation:

$$\text{score}(q, d) = \left(\vec{q}_C \bullet \vec{d}_C\right) \cdot \left(\vec{q}'_W \bullet \vec{d}'_W\right) \tag{6.1}$$

where $\bullet$ denotes the dot product of two vectors. Since P_b generates $C_{[m,k]}$ and $W_{[m,k]}$, $\vec{d}'_C$ and $\vec{d}'_W$ can be easily computed by P_b. Since q is P_a's private input query, P_a cannot send q to P_b to produce $\vec{q}'_C$ and $\vec{q}'_W$. A secure way is required to reduce query vectors from P_a.

6.4.2 The SCC-LSI Protocol

The SCC-LSI protocol performs the same task as the SCC protocol: find the similarity scores between one party's query q and the other party's document collection D securely. The main steps of the SCC-LSI protocol are provided in Algorithm 14. The key difference between SCC and SCC-LSI is that the SCC-LSI protocol requires additional steps to produce $C_{[m,k]}$, $W_{[m,k]}$, $\vec{d}_C$ and $\vec{d}'_W$, and secure steps to produce $\vec{q}_C$ and $\vec{q}'_W$. It should be noted that the way to compute the similarities also differs between the two protocols.

- **Public input** - In addition to G and $|D|$, the size of the reduced vector k is known to both parties.
- **Step 1** - For each document, P_b produces $\vec{d}$ and $\vec{d}'$ based on G. More importantly, P_b also needs to produce the SVD matrices $C_{[m,k]}$ and $W_{[m,k]}$ which are subsequently used to generate the dimensionally-reduced vectors $\vec{d}_C$ and $\vec{d'}_W$ for each document.
- **Step 2** - This step is identical to that of the SCC protocol. Briefly, P_a encrypts his query vector component-wise, and sends the encrypted values to P_b.
- **Step 3** - Since P_a has no access to $\vec{G}$, $C_{[m,k]}$ and $W_{[m,k]}$, it is impossible for P_a to produce $\vec{q}'$, $\vec{q}_C$ and $\vec{q}'_W$. However, the encrypted forms of these vectors can be

generated by P_b. More specifically, the sub-steps (1), (2) and (3) produce $E_{pu}(\vec{q}')$, $E_{pu}(\vec{q}_C)$ and $E_{pu}(\vec{q}'_W)$ respectively. At the last sub-step, P_b, for each document in D, generates $E_{pu}\left(\vec{q}_C \bullet \vec{d}_{jC}\right)$ and $E_{pu}\left(\vec{q}'_W \bullet \vec{d}'_{jW}\right)$ denoted by s_{j_1} and s_{j_2} which are the private inputs from P_b to execute the MP protocol.

- **Step 4** - For each document $d_j \in D$, the two parties utilize the MP protocol to produce $\left(\vec{q}_C \bullet \vec{d}_{jC}\right) \cdot \left(\vec{q}'_W \bullet \vec{d}'_{jW}\right)$ which is only known to P_a.

Security Analysis

The security guarantee of the SCC-LSI protocol is almost the same as that of the SCC protocol. The only information disclosed is the global vector space G, the size of the document collection and the size of the reduced vectors. The same analyses given in Section 6.3.3 can be used here to prove the security of the SCC-LSI protocol.

Complexity Analysis

The complexity for the MP protocol does not change, so we directly adopt the results from Section 6.3.4. Overall, the computational cost for P_b is about 5 encryption operations, and the computational cost for P_a is about 4 encryption operations. In addition, P_b needs to send $3t$ bits and P_a needs to send t bits.

At step 2 of the SCC protocol, P_a needs to perform $2m$ encryption operations. At step 4, P_a needs to perform $4n$ encryption operations. Thus, the computational cost for P_a is about $2m + 4n$ encryptions. At step 3, P_b needs to perform $2km$ and $2kn$ encryption exponentiations at sub-steps (1) and (2). In addition, step 4 requires P_b to perform $5n$ encryptions. As a result, the computational cost for P_b is about $5n$ encryptions and $2k(m + n)$ encryption exponentiations. The communication complexity is the same as the SCC protocol, bounded by $(m + n)t$ bits and $3nt$ bits for P_a and P_b respectively. SCC-LSI also requires two rounds of communication between P_a and P_b.

6.4.3 The SCC-LSI* Protocol

In theory, if n is much smaller than m, the SCC-LSI protocol should be more efficient than the SCC protocol on P_b's side. However, the computational cost at P_a is doubled. We may ask if it is possible to reduce the costs for both parties. Here we present the SCC-LSI* protocol, a simplified version of SCC-LSI, to further improve the computational efficiency.

The SVD matrices $C_{[m,k]}$ and $W_{[m,k]}$ have reduced ranks comparing to the original term document matrices, and the SVD method is not reversible. In other words, given $C_{[m,k]}$ and $W_{[m,k]}$ there does not exist a method to derive the original matrices. However, we cannot formally quantify the amount of information leaked from the SVD matrices which is still an open problem. Here, we assume that the information disclosed from $C_{[m,k]}$ and $W_{[m,k]}$ regarding P_b's document collection is negligible. The SCC-LSI* protocol is constructed based on this assumption. The key steps are given in Algorithm 15. The differences between SCC-LSI and SCC-LSI* are highlighted in the following steps:

- **Step 1** - P_b shares $C_{[m,k]}$ and $W_{[m,k]}$ with P_a.
- **Step 2** - Since P_a has $\vec{G}$, $C_{[m,k]}$ and $W_{[m,k]}$, P_a can produce $\vec{q}_C$ and $\vec{q}'_W$ locally, encrypt them component-wise, and send the encrypted values to P_b.
- **Step 3** - Unlike the SCC-LSI protocol, P_b only needs to compute $E_{pu}\left(\vec{q}_C \bullet \vec{d}_{jC}\right)$ and $E_{pu}\left(\vec{q}'_W \bullet \vec{d}'_{jW}\right)$.

Security Analysis

For the SCC-LSI* protocol, we cannot adopt the same security proof strategies as the previous two protocols due to the fact that the SVD matrices are disclosed to P_a. However, in the following, we analyze if any information is disclosed from these matrices and point out a randomization technique to fully secure the shared matrix.

Our LSI implementation is based on the LSI theory proposed in [89,90]. Lucene similarity score consists of products of two dot products - tf-idf vectors ($\vec{q}_C$, $\vec{d}_j$) and binary vectors ($\vec{q}'_W$, $\vec{d}'_j$). Here, we concentrate on one of the pairs as similar argument applies to the other pair. P_a obtains a truncated vector $\vec{q}_C$ from m dimensions to k, $k << m$ with at least a difference of 2 to 3 orders depending on the size of G from the shared $C_{[m,k]}$ (see Equation 18.21 in [90] or Section 4.2.4 in [89]). Equation (6.2) represents the query truncation operation, '$'$' indicates the transpose operation, $\vec{q}_{[m,1]}$ is the query before truncation.

$$\vec{q}_C = \left[\vec{q}_{[m,1]}\right]' \cdot C_{[m,k]} \tag{6.2}$$

On the server P_b side, who performs the critical Singular Value Decomposition (SVD) before the above computation takes place, the computation looks like [89, 90]:

$$A_{[m,n]} = C_{[m,r]}\Sigma_{[r,r]}\left[E_{[n,r]}\right]' = \left[C_{[m,r]}\Sigma^{1/2}_{[r,r]}\right]\left[E_{[n,r]}\Sigma^{1/2}_{[r,r]}\right]' \tag{6.3}$$

where m and n are the number of terms and documents respectively, A is term-by-document matrix (tf-idf or binary values) and C indicates term-term fuzzy relatedness with respect to number of documents. The product of rows i and j of $\left[C_{[m,r]}\Sigma^{1/2}_{[r,r]}\right]$ gives this relatedness in terms of document counts for terms indexed by i and j. It is important to note that Σ is never shared in our protocol. Similarly, E indicates document-document incidence with respect to terms. Σ is a diagonal matrix consisting of eigenvalues of A, and $r \leq \min(m, n)$ is the rank of A.

$$\hat{A}_{[m,n]} = C_{[m,k]}\Sigma_{[k,k]}\left[E_{[n,k]}\right]' = \left[C_{[m,k]}\Sigma^{1/2}_{[k,k]}\right]\left[E_{[n,k]}\Sigma^{1/2}_{[k,k]}\right]' \tag{6.4}$$

The above equation shows the truncation part for k approximation where $k \ll r$. Here the eigenvalues in the diagonal matrix Σ are ordered in descending order, and likewise, columns of C and E are shifted around to keep up with the new ordering. $(r - k)$ smallest eigenvalues are omitted from Σ, and similarly, $(r - k)$ right-most columns of C and E are removed. The party or user P_a is not aware of the newly shifted columns and how the mapping between terms and columns has changed. P_a is also not aware of which terms were deemed unimportant towards the truncated approximation of A (i.e., $\hat{A}$ derived in Equation (6.4)), and hence dropped. Thus, when $C_{[m,k]}$ is received by P_a, it cannot make a deduction regarding any value since it does not understand which terms are being considered. Also transforming from low-dimensional representation $C_{[m,k]}$ to its accurate and high-dimension representation $C_{[m,r]}$, without any parametric information, is not possible according to the solvability of linear systems.

In summary, the actual data, i.e., P_b's term-document collection matrix $A_{[m,n]}$ is protected since it is a product of three matrices - $C_{[m,r]}$, $\Sigma_{[r,r]}$ and $[E_{[n,r]}]'$ out of which just a truncated form of $C_{[m,r]}$, i.e., $C_{[m,k]}$ is revealed to P_a. $\Sigma_{[r,r]}$ and $[E_{[n,r]}]$ are not revealed. Thus, reconstruction of $A_{[m,n]}$ is not possible since two out of three constituent factors are not available to P_a. Also, as per linear algebraic concepts or method for solving system of linear equations, there can be exponentially many solutions for $\Sigma_{[r,r]}$ and $E_{[n,r]}$.

If the above analysis is still not sufficient to justify the security of sharing $C_{[m,k]}$, we can adopt the randomization approach proposed in [91, 92]. Instead of sharing $C_{[m,k]}$ directly, a randomized version of $C_{[m,k]}$, e.g., $\hat{C}_{[m,k]}$, can be shared with P_a. $\hat{C}_{[m,k]}$ preserves the key characteristics of $C_{[m,k]}$ for data analytics without revealing any specific information about $C_{[m,k]}$. This technique has proven to be both information theoretically secure and effective [82, 92].

Complexity Analysis

At step 2 of the SCC protocol, P_a needs to perform $2k$ encryption operations. At step 4, P_a needs to perform $4n$ encryption operations. Thus, the computational cost for P_a is about $2k + 4n$ encryptions. At step 3, P_b needs to perform $2kn$ encryption exponentiations. In addition, step 4 requires P_b to perform $5n$ encryptions. As a result, the computational cost for P_b is about $5n$ encryptions and $2kn$ encryption exponentiations. The communication complexity is bounded by $(2k + n)t$ bits and $3nt$ bits for P_a and P_b respectively. SCC-LSI* also requires two rounds of communication between P_a and P_b.

A summary of the computation and communication complexities of the three SCC protocols is given in Table 6.1. The costs column indicates the cost unit. HEnc represents the homomorphic encryption operation, and Exp represents encryption exponentiation. As shown in the table, when $k \ll m$, the SCC-LSI protocol can be more efficient for P_b in comparison to the SCC protocol. The SCC-LSI* protocol is the most efficient for both parties.

Table 6.1: SCC Complexity

Protocols	Costs	P_a	P_b
SCC	HEnc	$m + 4n$	$5n$
	Exp	0	mn
	Bit	$(m + n)t$	3nt
	Round	2	2
SCC-LSI	HEnc	$2m + 4n$	$5n$
	Exp	0	$2k(m + n)$
	Bit	$(m + n)t$	3nt
	Round	2	2
SCC-LSI*	HEnc	$2k + 4n$	$5n$
	Exp	0	$2kn$
	Bit	$(2k + n)t$	3nt
	Round	2	2

On the other hand, Both SCC and SCC-LSI have provable security guarantees.

Algorithm 14 SCC-LSI($\langle P_a, q\rangle, \langle P_b, D\rangle) \rightarrow s_1, \ldots, s_n$

```
Require: The global vector space $G$, the size of the reduced vector, and the size of
    $D$ are known to both parties; $q$ is $P_a$'s private input, and $D$ is $P_b$'s private input
1: $P_b$
    1. For $1 \leq j \leq n$: compute $\vec{d_j}$ and $\vec{d'_j}$ based on $G$, where $n = |D|$, $m = |G|$
    2. Using the SVD algorithm to generate $C_{[m,k]}$ and $W_{[m,k]}$ based on $\vec{d_j}$s and $\vec{d'_j}$s
       respectively
    3. For $1 \leq j \leq n$: compute $\vec{d}_{jC}$ and $\vec{d'}_{jW}$ for every $\vec{d_j}$ and $\vec{d'_j}$ respectively
2: $P_a$
    1. Generate a private-public key pair $(pr, pu)$ of an HEnc scheme and send $pu$ to
       $P_b$
    2. Generate $\vec{q}$ and $\vec{q'}$ based on $G$
    3. Encrypt $\vec{q}$ and $\vec{q'}$ component-wise:
    4. $E_{pu}(\vec{q}[1]), \ldots, E_{pu}(\vec{q}[m])$
    5. $E_{pu}(\vec{q'}[1]), \ldots, E_{pu}(\vec{q'}[m])$
    6. Send them to $P_b$
3: $P_b$
    1. For $1 \leq j \leq k$:
    2. $E_{pu}(\vec{q}_C[j]) \leftarrow \prod_{i=1}^{m} E_{pu}(\vec{q}[i])^{C_{[m,k]}[i,j]}$
    3. $E_{pu}(\vec{q'}_W[j]) \leftarrow \prod_{i=1}^{m} E_{pu}(\vec{q'}[i])^{W_{[m,k]}[i,j]}$
    4. For $1 \leq j \leq n$:
    5. $s_{j_1} \leftarrow \prod_{i=1}^{k} E_{pu}(\vec{q}_C[i])^{\vec{d}_{jC}[i]}$
    6. $s_{j_2} \leftarrow \prod_{i=1}^{k} E_{pu}(\vec{q'}_W[i])^{\vec{d'}_{jW}[i]}$
4: $P_a$ and $P_b$ ($j = 1$ to $n$)
        $\langle P_a, s_j\rangle \leftarrow \mathrm{MP}(\langle P_a, pr\rangle, \langle P_b, pu, s_{j_1}, s_{j_2}\rangle)$, where $s_j = D_{pr}(s_{j_1}) \cdot D_{pr}(s_{j_2})$
5: $P_a$
        Send $s_1, \ldots, s_n$ to $P_b$
```

Algorithm 15 SCC-LSI*$(\langle P_a, q\rangle, \langle P_b, D\rangle) \rightarrow s_1, \ldots, s_n$

```
Require: The global vector space $G$, the size of the reduced vector, and the size of
    the document collection $D$ are known to both parties; $q$ is $P_a$'s private input, and
    $D$ is $P_b$'s private input
1: $P_b$
    1. For $1 \leq j \leq n$: compute $\vec{d_j}$ and $\vec{d'_j}$ based on $G$, where $n = |D|$, $m = |G|$
    2. Adopting SVD algorithm, generate $C_{[m,k]}$ and $W_{[m,k]}$ based on $\vec{d_j}$s and $\vec{d'_j}$s
       respectively; Send $C_{[m,k]}$ and $W_{[m,k]}$ to $P_a$
    3. For $1 \leq j \leq n$: compute $\vec{d}_{jC}$ and $\vec{d'}_{jW}$ for every $\vec{d_j}$ and $\vec{d'_j}$ respectively
2: $P_a$
    1. Generate a private-public key pair $(pr, pu)$ of an HEnc scheme and send $pu$ to
       $P_b$
    2. Generate $\vec{q}_C$ and $\vec{q'}_W$ based on $C_{[m,k]}$ and $W_{[m,k]}$
    3. Encrypt $\vec{q}_C$ and $\vec{q'}_W$ component-wise:
    4. $E_{pu}(\vec{q}_C[1]), \ldots, E_{pu}(\vec{q}_C[k])$
    5. $E_{pu}(\vec{q'}_W[1]), \ldots, E_{pu}(\vec{q'}_W[k])$
    6. Send them to $P_b$
3: $P_b$
       For $1 \leq j \leq n$:
    1. $s_{j_1} \leftarrow \prod_{i=1}^{k} E_{pu}(\vec{q}_C[i])^{\vec{d_j}[i]}$
    2. $s_{j_2} \leftarrow \prod_{i=1}^{k} E_{pu}(\vec{q'}_W[i])^{\vec{d'_j}[i]}$
4: $P_a$ and $P_b$ ($j = 1$ to $n$)
        $\langle P_a, s_j\rangle \leftarrow \text{MP}(\langle P_a, pr\rangle, \langle P_b, pu, s_{j_1}, s_{j_2}\rangle)$, where $s_j = D_{pr}(s_{j_1}) \cdot D_{pr}(s_{j_2})$
5: $P_a$
        Send $s_1, \ldots, s_n$ to $P_b$
```

6.5 Experimental Results

In this section, we empirically show the running time of the proposed protocols with a real dataset. As stated in Section 6.1, there does not exist a secure protocol that has the same functionality as the proposed protocols. Therefore, the main goal of this section is to show how efficient the SCC-LSI* protocol by comparison with the baseline, the SCC protocol. Since SCC-LSI* is proved to be theoretically faster than SCC-LSI (see Table 2 for SCC complexities), in order to ease use of notation and consequently readability, further references to LSI indicate the reference to SCC-LSI* (i.e., Algorithm 15), unless otherwise explicitly noted.

6.5.1 The Experiment Setup

The proposed protocols are implemented in a local area network. We use two separate machines to emulate the tasks performed by P_a and P_b. Additional information regarding the dataset and implementation details is given below:

- **Dataset** - We tested our system on a real data set [93]. The dataset consists of 889 unannotated, de-identified discharge summaries. This dataset was created as part of the Natural Language Processing (NLP) contest organized by the authors as part of the Informatics for Integrating Biology to the Bedside (i2b2) project.
- **Data extraction** - The dataset consists of a single file with multiple records in the form of discharge summaries. We split these records into their individual files. Using the Lucene analyzing package, we applied the lower case filter, stop word filter and the Porter stemming filter. The lower case filter converts all the tokens generated during the analysis phase to lowercase tokens. The stop word filter removes stop words or common terms such as “a“ and “the”. which act as noise and contribute little semantically. The Porter stemmer implements the Porter stemming algorithm [88].
- **Hardware details - Machine # 1**
  - Processor: 64-bit Intel® Xeon® CPU E2186G @ 3.80GHz 12CPU(s)

  - Memory: 62GiB DIMM DDR4 2666MHz (0.4 ns)
  - Hard disk: 1024GB PC400 NVMe SK hynix

- **Hardware details - Machine # 2**
  - Processor: 64-bit Intel® Xeon® CPU E2186G @ 3.80GHz 12CPU(s)
  - Memory: 62GiB DIMM DDR4 2666MHz (0.4 ns)
  - Hard disk: 1024GB PC400 NVMe SK hynix

- **Programming languages and software tools** - The protocols were implemented using Java and C. We used Java and the Apache Lucene ™ [84] Java information retrieval library to implement the secure similarity computation protocol. All the security related operations, such as encryption and decryption, were carried out using C and the GMP library [75].

- **Additive Homomorphic Encryption** - We use the Paillier [70] for its additive homomorphic property.

6.5.2 Empirical Analysis

To account for the difference in processors, we ran the experiment by alternating the client and server roles for these systems and averaged out the results. After indexing 889 documents we found the number of vector dimensions (i.e.,global terms) to be about 24,000. Such high dimensionality will significantly affect the computation performance. After applying the Porter stemming filter and eliminating words of size two and less, we were able to reduce the number of dimensions to approximately half. Thus, we have $n = 889$ and $m = 13,098$.

Figure 6.3 shows the execution time for the SCC protocol, plotted against the public key sizes. The evaluation is based on three key sizes: 512, 1,024 and 2,048. It can be seen that time required increases non-linearly with respect to the key sizes. It should be recognized that this figure only reports the total execution time between the two parties. Clearly, the execution time is very high for a key size of 2,048 which is not practical for

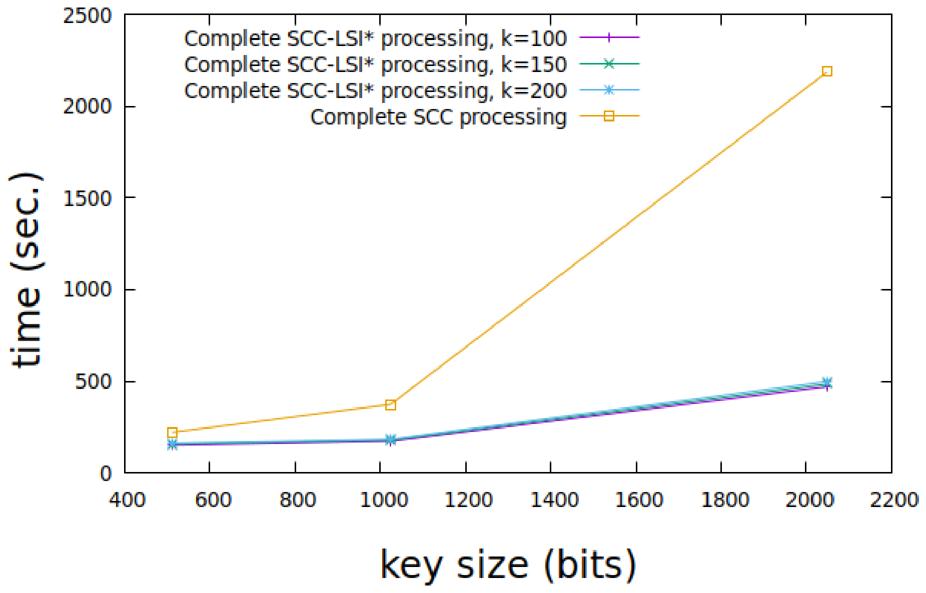

Figure 6.3: Run time complexity for SCC and SCC-LSI*

real-time applications. However, the applications discussed in Section 6.1 do not require real-time responses.

Next we show applying LSI improves the runtime efficiency of the SCC protocol. We overcome the dimensionality problem by reducing the dimension size from $m = 13,098$ to $k \in \{100, 150, 200\}$ using the LSI technique. More specifically, we report the results given in Figure 6.3 for the SCC-LSI* protocol. As can be seen, the running time grows with the key size. In addition, as k increases, the running time only increases slightly. For example, when k varies from 100 to 200, the running time changes from 470 to 498 seconds when the encryption key size is fixed to 2,048. The main reason the k variable does not affect the running time not as much is that k is smaller than n and encryption exponentiation in the proposed protocol is less costly than the encryption operation.

Overall, the SCC-LSI* protocol is much more efficient than the SCC protocol. For example, when the key size is 2,048, the running time for SCC-LSI* is about 498 seconds with $k = 200$. By contrast, the running time for SCC is about 2,186 seconds which is several times slower.

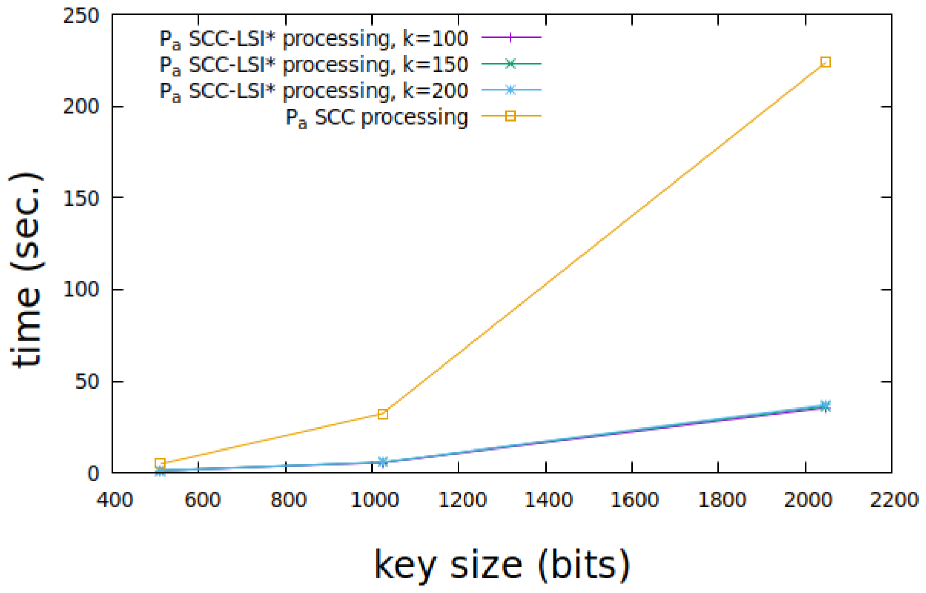

Figure 6.4: P_a's local running time vs. key size

Local Computation Time for P_a and P_b

The previous results only show the total running time of the entire protocol. It is not clear about the costs at each individual party. Figures 6.4, 6.5, and 6.6 summarize the running time results of each party excluding the communication costs. As shown in Figures 6.4 and 6.5, both parties' running time increases with the key size, and LSI can reduce the running time. Figure 6.6 shows the pre-processing time with or without LSI. Clearly, LSI adds more cost on producing the transformed vectors, but it helps on improving the efficiency of the overall protocol. The LSI pre-processing time also includes the Singular Value Decomposition on the P_b side. As evidenced by the empirical results, to protect data security and personal privacy, we have to sacrifice some efficiency. Almost all existing SMC-based privacy-preserving data analytical protocols are not suitable for real-time applications.

Three-party Secret Sharing Technique (SST)

SST is developed using secret sharing [94] based secure multi-party computation technique. The SST version requires three parties: P_0, P_1 and P_2. Our implementation utilizes Beaver multiplication triplets [82, 95] for computation of dot products, which form the basis for

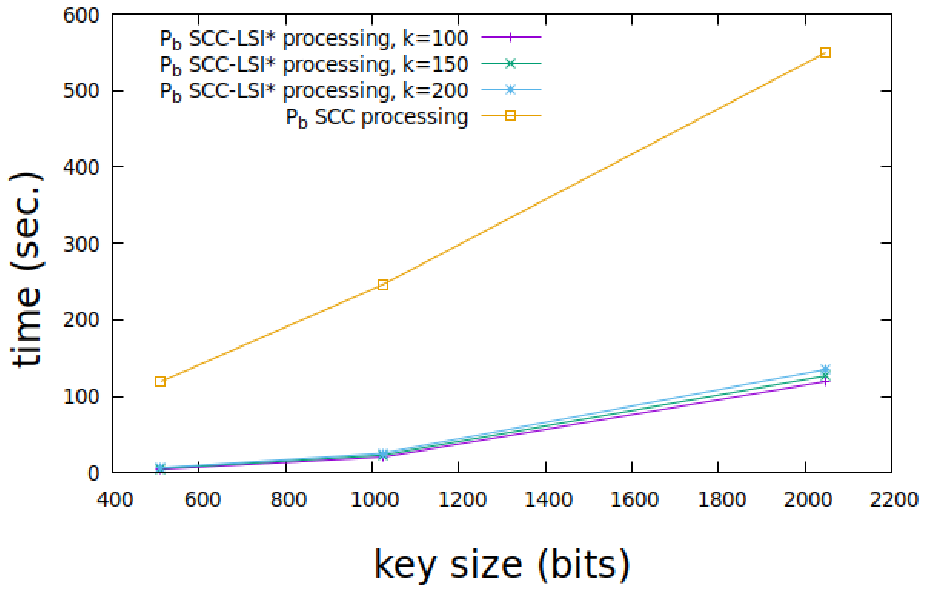

Figure 6.5: P_b's local running time vs. key size

Lucene's similarity score computation (Equation 2.2). P_0 is the party that generates the multiplication triplets required for multiplication purposes. It can be run before the actual SST protocol begins. Therefore, it can be considered a one-time computation as a preprocessing step. Workload for P_1 and P_2 is symmetrical in nature. In our problem domain, P_a and P_b can take on P_1 and P_2's roles respectively. They simply have to secretly share the query and the SVD matrix between each other.

Figures 6.7 and 6.8 show how SST performs against the homomorphic encryption based SCC and SCC-LSI*. P_1 and P_2 have identical workload due to the symmetrical nature of the Beaver triples-based multiplication protocol used for dot product computations. It can be seen clearly that SST performs exceptionally well. To run the protocol for 889 documents with 889 dimensions, P_1 and P_2 complete their processing part under a second, whereas SCC-LSI* does it under 62 seconds for P_b and under 8 seconds for P_a with a public key size of 1,024 bits. SCC-LSI* cannot be tested for dimension size k equal to $13,098$ as the rank r of Term-Document matrix in that case would be less than or equal to $min(13,098,889)$ and k is bounded by r. Additionally, for 889 documents with 13,098 dimensions, P_1 and P_2 complete their processing part under 11 seconds whereas SCC does it under 246 seconds for P_b and under 33 seconds for P_a with a public key size of 1,024 bits. From these results, it

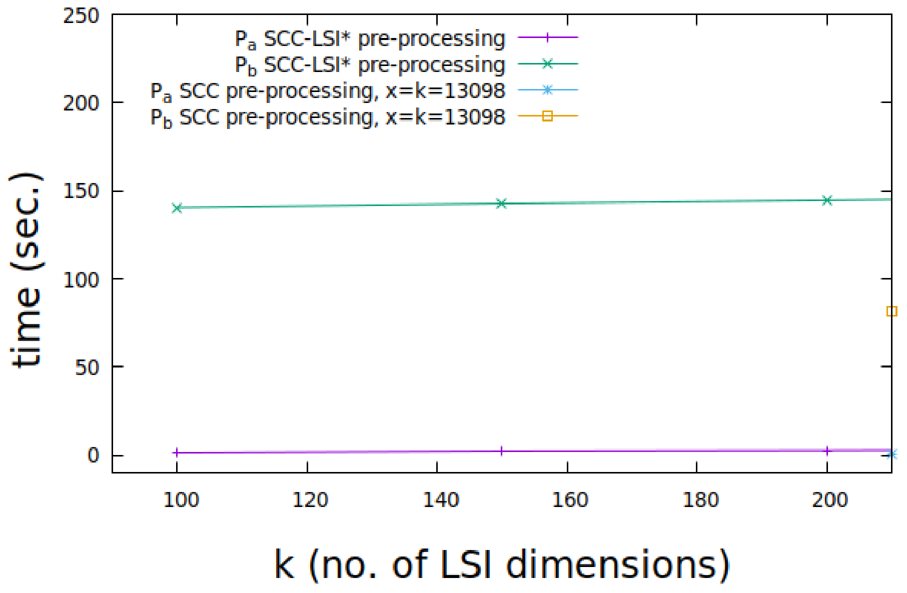

Figure 6.6: Pre-processing time for each party

seems that the LSI-based optimization may not be needed for SST. Nevertheless, LSI may improve result accuracy, and if the document collection does not change often, LSI can still provide a significant performance gain over a large amount of user queries.

For P_0 to generate the multiplication triples and their shares (one triple for each product operation in a pseudo-random manner), it took 0.54 seconds for 889 dimensions and under 8 seconds for $13,098$ dimensions. If n and m are number of documents and dimensions respectively, the number of triples required would be $(2m+1)n$, which includes the cumulative count of dot products of tf-idf and binary vectors, as well as the intermediate product of those two (performed by MP protocol in SCC) for all documents in the collection. Moreover, since the triple generation is independent in nature, P_0 can pre-compute the shares and send them to the other parties in a bulk manner, as a pre-processing step.

In short, at the cost of an additional party P_0, SST significantly outperforms both SCC and SCC-LSI* in terms of CPU time for processing purposes. Thus, depending on the number of involved parties and the underlying system architecture to be deployed, either SCC-LSI* or SST can be used.

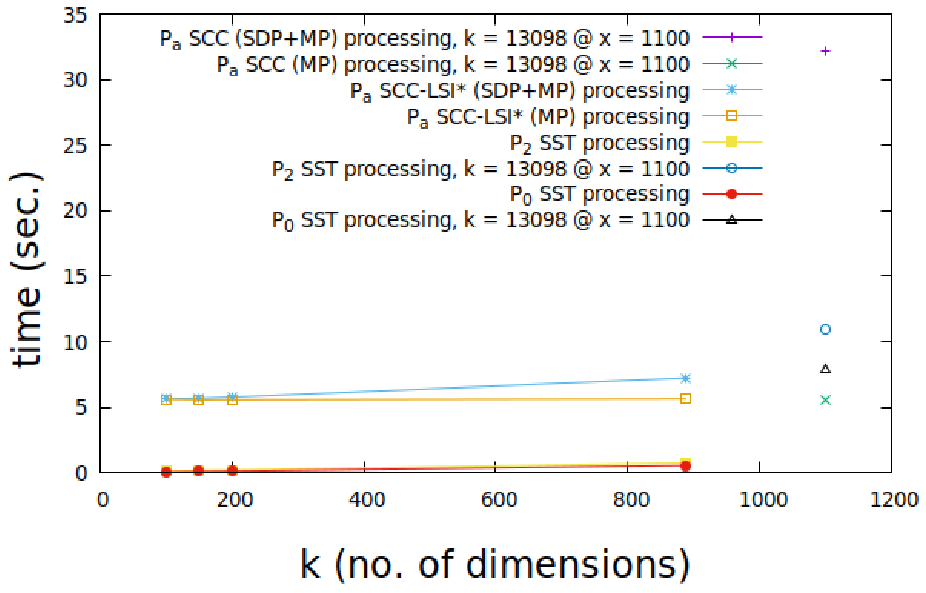

Figure 6.7: P_a's Processing time comparison for full and intermediate MP computation as well as the Secret Sharing Technique (SST)

MP - Intermediate Protocol Computation

Figures 6.7 and 6.8 also provide an idea about the amount of time spent, for both the parties, in running the intermediate secure Multiplication Protocol (MP) described in the work. Comparison is provided for SCC (found at $x = 1100$ and $k = 13,098$) and its optimized version SCC-LSI*, for both the parties P_a and P_b. It is observed that the CPU time required for executing MP protocol is very short when compared to the Secure Dot Product computation at Step 3 of SCC and SCC-LSI*.

The Performance on the Client Side

The computation on the P_a side is actually quite small. Under the homomorphic encryption (HEnc) approach, it performs a little over 10,000 HEnc operations without latent semantics indexing (LSI). With LSI, it performs multiplications between a vector and a matrix. In addition, it performs several hundred HEncs depending on the reduced dimension size. Under the secret sharing implementation, the user only needs to generate secret shares of his or her query vector, which is substantially more efficient than HEnc. The vector size

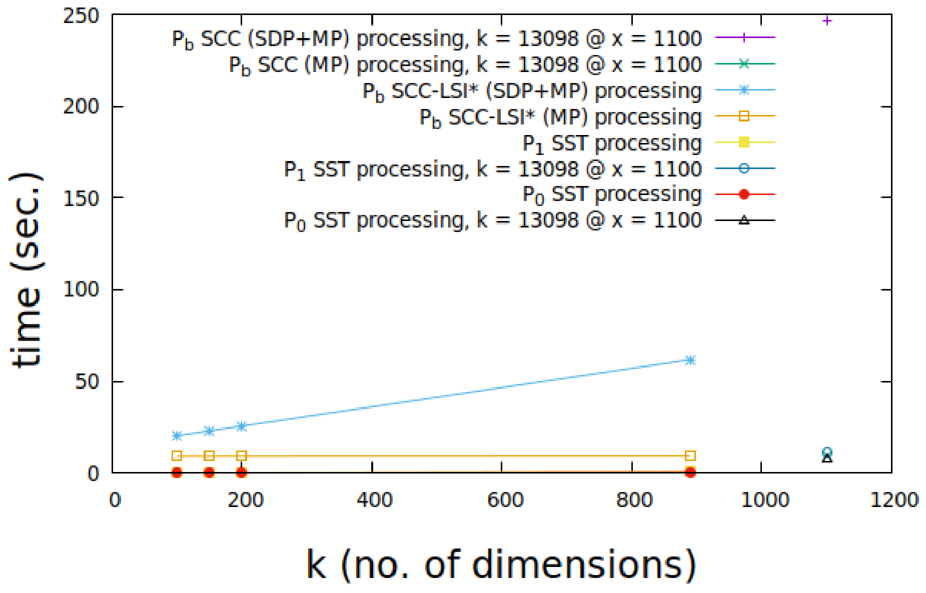

Figure 6.8: P_b's Processing time comparison for full and intermediate MP computation as well as the Secret Sharing Technique (SST)

only depends on the global vector space (G). As a result, when G becomes saturated, the computation overhead has minimal change when new documents are added. Even if a new document contains terms not in G, these terms may not be added due to the process of removing stop words and word stemming. Also, when using secret sharing, user queries can be efficiently generated from standard computing devices, such that using LSI to improve performance may no longer be necessary.

6.6 Extensions to the SCC Protocols

Our SCC protocols are secure under the semi-honest model and return a list of similarity scores. Here we propose ways to enhance the SCC protocols to make them secure under the malicious model. In addition, the list of similarity scores may allow Alice to infer too much information about Bob's document collection. We present strategies to mitigate this inference.

6.6.1 SCC under the Malicious Model

To make the SCC protocols secure under the malicious model, we will adopt the threshold based Paillier encryption scheme [96] and the innovative dual execution strategy [97] with the following modifications.

- All encryption and decryption operations need to be based on the threshold Paillier.
- Bob (P_b) also needs to encrypt the vectors representing his document collection and sends them to Alice (P_a).
- Dual execution: each party computes the similarity scores securely and independently.
- If the outcomes from their independent executions are not the same, the protocol aborts.

The main steps of the proposed extension are given in Algorithm 16 and Algorithm 17. In the following, we clarify several key points in the algorithms:

- **Step 1 (b)** - P_b encrypts his document collection and sends the component-wise encrypted vectors to P_a.
- **Steps 3 and 4** - These steps allow P_a to obtain the set of similarity scores and P_b to obtain the set of corresponding encrypted similarity scores. These scores are derived based on P_b's local information.
- **Steps 5 and 6** - These steps allow P_b to obtain the set of similarity scores and P_a to obtain the set of corresponding encrypted similarity scores. These scores are derived based on P_a's local information.
- **Steps 7 and 9** - P_a checks if the similarity scores computed at both sides are the same. If not, the execution aborts.
- **Steps 8 and 10** - P_b checks if the similarity scores computed at both sides are the same. If not, the execution aborts.

Technically speaking, the proposed extension does not lead to true secure protocol under the malicious model because to be secure against a malicious adversary, every step of the execution needs to be verified. Our extension verifies the computations in an aggregate fashion, and it can still detect malicious behaviors without the costs of a theoretically secure protocol under the malicious model. As we can see, the added cost is much higher than our original SCC protocols. This provides a clear indication that an SCC protocol secure under the malicious model may not have much practical value.

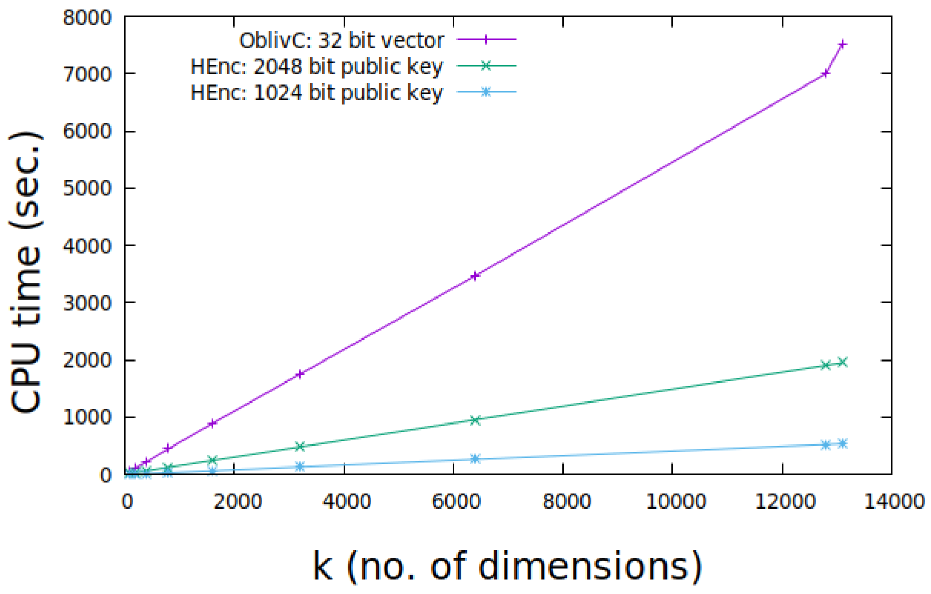

Figure 6.9: Garbled circuit vs. Paillier

6.6.2 Garbled Circuit, ECE, Private Keyword Search and Private Information Retrieval

Garbled circuit [98, 99] can be used to securely implement any polynomially bounded functionalities. When the functionality is small, garbled circuit can provide efficient implementation. We also tried the garbled circuit approach to implement the SCC protocol. Our results indicated that implementing SCC using garbled circuit is not as efficient as the approach based on Paillier encryption scheme. Note that without affecting the comparison outcome, our results were based on the key component of SCC, the dot product computation, for both garbled circuit and homomorphic encryption based approaches. Performance comparison results are shown in Figure 6.9. The garbled circuit based approach is implemented using Obliv-C [98] which is more efficient than FastGC [99]. For both implementations, we used 500 documents since the whole dataset would take a much longer time to execute. It is clear that the garbed circuit implementation is less efficient than the Paillier based approach even when the key size or the number of bits to represent N is fixed to 2,048 bits.

Elliptic Curve ElGamal (ECE) also has the additive homomorphic property and its encryption operation is more efficient than Paillier. On the other hand, the decryption operation (i.e., trying encrypting every value in the domain and compare with the encrypted value) is not very efficient when the domain size is large. For example, in the SCC-LSI protocol, the matrices contain real values. To preserve enough precision when converting them into integer values in $\mathbb{Z}_N$, we need to use several digits at least. Assume that we use five digits, the dot product will produce ten-digit numbers on average. Clearly, the decryption operation of ECE will be several order of magnitude less efficient than Paillier.

While Searchable Encryption (SE) [49, 51, 100–102] may be more efficient to conduct keyword search, it has several limitations in our problem domain. First of all, our main motivation is to support or extend the existing Information Retrieval libraries, such as Apache Lucene, which is widely used both independently or as part of larger projects, with privacy-preserving mechanisms to arrive at similar conclusions or results. Lucene similarity metrics utilize dot products, and it is not clear how SE can be used to compute a dot product. Secondly, deciding a set of key words is not straightforward, and it requires inputs from domain experts. Moreover, from the security perspective, SE leaks access patterns which can leak private information regarding the document collection [103].

Private Information Retrieval (PIR) [58,59,62,104,105] seems to solve a similar problem, but its application is quite different from ours. To perform PIR, the user needs to know the location of the information, and it cannot compute the scoring functions used in Lucene. In addition, the complexity of an information theoretically secure PIR to retrieve a single bit is bounded by $k \log_2 k \cdot n^{\frac{1}{\log_2 k}}$ [58], where k is the number of servers and n is the size of a bit string. For a two-server setting, the entire string has to be sent to the user. Thus, even if ignoring the inability of computing the similarity functions, it seems that to retrieve a single document using PIR, the entire document collection has to be transferred to the user.

On the other hand, in our solutions, the user receives similarity scores and several documents related to the top similarity scores. Because a document can have thousands of terms, a similarity score is generally much smaller than a single document. Under this circumstance, our solutions can be more efficient than PIR even for $k = 8$ or other constant values.

6.6.3 Mitigating the Inference Problem

Similarity scores can definitely leak some information regarding both the query document and the document collection. If both Alice and Bob are semi-honest, the similarity scores may not leak specific information about a particular term. Note that any information deduced from the output of an SMC protocol does not make the protocol less secure because the same information can be deduced from the ideal model, i.e., the trusted third party (TTP) model.

The inference problem becomes more severe when either party is malicious. For example, Alice could modify her input such that only one entry (e.g., for term t_i) in the query vector is 1 and the rest of entries are 0s. Alice can find out how many times t_i occurs in Bob's collection. Similarly, Bob could modify his document vectors to discover statistical information regarding Alice's query terms. Since input modification is not preventable even in the TTP model, this inference problem cannot be eliminated regardless if an SCC protocol is secure under the semi-honest or malicious model.

The inference problem can be mitigated to certain degree by restricting the output. For instance, the formulation given in Equation 1.2 can be changed to the following:

$$\text{SCC}(\langle P_a, q, k\rangle, \langle P_b, D\rangle) \rightarrow \hat{d}_{s_1}, \ldots, \hat{d}_{s_k} \tag{6.5}$$

Instead returning similarity scores, the new formulation returns the snippets of the top k most similar documents. Based on these snippets and the access control policies, Alice and

Bob can decide if they need to share the actual documents. This modification mitigates the inference problem which cannot be prevented completely due to the intrinsic nature of the information retrieval problem itself. To implement the top-k based SCC, we can adopt the Secure Minimum out of n Numbers (SMIN$_n$) protocol proposed in [106]. In addition, we can define a threshold t such that SCC returns snippets of documents whose similarity scores are above t. The proposed protocols can be easily modified to implement this threshold based formulation. Instead of directly returning the similarity scores, the dot product component of SCC can be simply modified to return secret shares of the similarity scores. Then a secure comparison sub-protocol takes the secret shares and t as inputs and returns document IDs to the server whose similarities are above t. After what the server can decide what to share with the user.

Another possible solution, in malicious settings, is to utilize Differential Privacy (DP) [79] mechanism of adding noise based on similarity score sensitivity and parameter ϵ to hide accurate similarity scores from the malicious adversary. The only downside is that scores would not be accurate and we would not have an exact and privacy-preserving solution to the non-secure capabilities offered by Lucene.

6.6.4 The Global Vector and LSI Matrices

In the proposed SCC protocols, the global vector space G needs to be shared between Alice and Bob. The G discloses the number of terms in Bob's document collection to Alice. If Bob's document collection is sufficiently large, G could contain all possible words. When this happens, information leakage regarding a specific document could be negligible. In general, how much information leaks from G regarding the individual documents in Bob's collection is impossible to analyze. In practical situations, we do not believe that the disclosure of G could seriously damage the security guarantee of the proposed SCC protocols. For situations where G cannot be shared, we may adopt the following strategies:

- Alice and Bob can adopt a universal vector space which is a super set that contains all the terms from Alice's query and Bob's document collection. All the terms from

Wordnet [107] could serve as such a universal vector space. However, this universal space could be too big to be practical.

- Alternatively, Alice and Bob could share a universal hash function that maps the terms into a different domain. This approach was proposed in [108] which hides G completely. On the other hand, the result accuracy could be affected a little bit but still acceptable for certain applications as analyzed in [108].
- A more secure and precise way is to develop an SMC based protocol to convert Alice's query into an encrypted vector representation without sharing G. This protocol can be implemented using an HEnc scheme, but additional computation and communication costs will be added to cause the SCC protocols less practical.

6.6.5 Handling Collection Update

When a document collection is built, its vector space is unlikely to experience much change as documents are added to the collection. This is because the total number of terms is fixed, just as in a dictionary. In other words, even if the global vector space G is large, once it is initialized and saved on the user side, the subsequent update cost is likely to be small. To confirm our suspicion, we performed several experiments to illustrate how an update may alter the size of G based on the i2b2 smoking discharge summaries dataset (consisting of 889 text files and 13098 terms in its global vector space). The results are summarized in Table 6.2, showing how the size of G changes when a batch of nine documents are indexed. It also provides the relative percent increase in total term counts when an indexing operation is performed. As can be seen, the increase in global term count percentage is quite high for the first few documents (i.e., around 90% and 24% for second and third batches, respectively). However, the change rate reduces drastically - to less than half of a percentage (i.e., 0.46% for the final batch). For larger document collections of related documents, we anticipate this will reduce even further. Therefore, when the index is sufficiently populated with related documents, the term or index update operation required for new documents becomes trivial in nature.

Document Count	Total Term Count	Term Addition Percent (%)
1	138	–
10	1330	89.62
19	1755	24.22
28	2090	16.03
⋮	⋮	⋮
199	6323	–
208	6460	2.12
217	6668	3.12
226	6734	0.98
⋮	⋮	⋮
433	9276	–
442	9359	0.89
451	9421	0.66
460	9474	0.56
⋮	⋮	⋮
658	11237	–
667	11322	0.75
676	11389	0.59
685	11478	0.78
⋮	⋮	⋮
865	12878	–
874	12959	0.63
883	13038	0.61
889	13098	0.46

Table 6.2: Growth states of the Lucene-constructed index of i2b2 dataset

Algorithm 16 $\mathrm{SCC}_m(\langle P_a, q, pr_a\rangle, \langle P_b, D, pr_b\rangle) \rightarrow (\langle P_a, s_1, \ldots, s_n\rangle, \langle P_b, s_1, \ldots, s_n\rangle)$

```
Require: The global vector space $G$ and the size $|D|$ are known to both parties; $q$
    is $P_a$'s private input, and $D$ is $P_b$'s private input; $pr_a$ and $pr_b$ are the threshold
    based private keys owned by $P_a$ and $P_b$
1: $P_b$
    1. Compute $\vec{d_j}$ and $\vec{d'_j}$ based on $G$, where $n = |D|$, $m = |G|$, $1 \leq j \leq n$
    2. Encrypt each $\vec{d_j}$ and $\vec{d'_j}$ component-wise, and send $E_{pu}(\vec{d_1}), \ldots, E_{pu}(\vec{d_n})$ and
       $E_{pu}(\vec{d'_1}), \ldots, E_{pu}(\vec{d'_n})$ to $P_a$
2: $P_a$
    1. Generate $\vec{q}$ based on $G$
    2. Encrypt $\vec{q}$ component-wise and send $E_{pu}(\vec{q})$ to $P_b$
3: $P_b$ ($j = 1$ to $n$)
    1. Compute $s_{j_1} \leftarrow \prod_{i=1 \wedge \vec{d_j}[i] \neq 0}^{m} E_{pu}(\vec{q}[i])$
    2. Compute $s_{j_2} \leftarrow \prod_{i=1}^{m} E_{pu}(\vec{q}[i])^{\vec{d'_j}[i]}$
4: $P_a$ and $P_b$ ($j = 1$ to $n$)
    $(\langle P_a, s_j\rangle, \langle P_b, E_{pu}(s_j)\rangle) \leftarrow \mathrm{MP}_m(\langle P_a, pr_a\rangle, \langle P_b, pr_b, s_{j_1}, s_{j_2}\rangle)$, where $s_j =$
    $E_{pr}(s_{j_1}) \cdot E_{pr}(s_{j_2})$
5: $P_a$ ($j = 1$ to $n$)
    1. Compute $s_{j_1} \leftarrow \prod_{i=1 \wedge \vec{q}[i] \neq 0}^{m} E_{pu}(\vec{d_j}[i])$
    2. Compute $s_{j_2} \leftarrow \prod_{i=1 \wedge \vec{q}[i] \neq 0}^{m} E_{pu}(\vec{d'_j}[i])$
6: $P_a$ and $P_b$ ($j = 1$ to $n$)
    $(\langle P_a, E_{pu}(s'_j)\rangle, \langle P_b, s'_j\rangle) \leftarrow \mathrm{MP}_m(\langle P_b, pr_b\rangle, \langle P_a, pr_a, s_{j_1}, s_{j_2}\rangle)$, where $s'_j =$
    $E_{pr}(s_{j_1}) \cdot E_{pr}(s_{j_2})$
7: $P_a$
    1. Compute $\tau \leftarrow \prod_{j=1}^{n} E_{pu}(s'_j) E_{pu}(N - s_j)$
    2. Send $D_{pr_a}(\tau^r)$ to $P_b$ where $r \in_R \mathbb{Z}_N$
8: $P_b$
    1. Compute $\tau' \leftarrow \prod_{j=1}^{n} E_{pu}(s_j) E_{pu}(N - s'_j)$
    2. Send $D_{pr_b}(\tau'^{r'})$ to $P_a$ where $r' \in_R \mathbb{Z}_N$
9: $P_a$
    1. Decrypt $D_{pr_b}(\tau'^{r'})$ with $pr_a$ to get $f_a$
    2. If $f_a \neq 0$ abort
```

```
10: $P_b$
        1. Decrypt $D_{pr_a}(\tau^r)$ with $pr_b$ to get $f_b$
        2. If $f_b \neq 0$ abort
```

Algorithm 17 $\text{MP}_m(\langle P_a, pr_a\rangle, \langle P_b, pr_b, E_{pu}(x), E_{pu}(y)\rangle) \rightarrow \langle P_a, x \cdot y\rangle$

```
Require: $P_a$: private key $pr$; $P_b$: $E_{pu}(x)$ and $E_{pu}(y)$, $pu$ is the public key of an HEnc
1: $P_b$
        1. Randomly generate $r_1$ and $r_2$ from $Z_N$
        2. Compute $D_{pr_b}(E_{pu}(x + r_1))$ and $D_{pr_b}(E_{pu}(y + r_2))$ and send them to $P_a$
2: $P_a$
        1. Decrypt $D_{pr_b}(E_{pu}(x + r_1))$ and $D_{pr_b}(E_{pu}(y + r_2))$ with $pr_a$ to get $x + r_1$ and
           $y + r_2$
        2. Compute $E_{pu}(xy + xr_2 + yr_1 + r_1r_2)$ and send it to $P_b$
3: $P_b$
        1. Compute $E_{pu}(xr_2)$, $E_{pu}(yr_1)$ and $E_{pu}(r_1r_2)$
        2. Derive $E_{pu}(xy)$ and send $D_{pr_b}(E_{pu}(xy))$ to $P_a$
4: $P_a$
        Decrypt $D_{pr_b}(E_{pu}(xy))$ with $pr_a$ to get $xy$
```

Chapter 7

Conclusion

In this work, we contribute to the privacy-preserving friend recommendation solutions in Online Social Networks (OSNs). More specifically, our first work, see chapter 3, addresses privacy concerns in collaborative environments involving integration partnerships among OSNs [6]. Similarly, our second work, see chapter 6, addresses privacy concerns in distributed environments involving document similarity computation by proposing Lucene-P^2, a privacy-preserving extension module to the widely used Apache Lucene Core, an information retrieval open-source library. This work is easily applicable for friend (or any entity) recommendation in OSNs by simply replacing documents with OSN user profiles.

In chapter 3, we present an approach that enables online social networks to form integration partnerships to provide friend recommendation services in a privacy-preserving manner. Our approach combines secure multiparty computation as well as differential privacy to give a holistic privacy guarantee. We analyze the complexity of the proposed approach as well as its security. A comprehensive experimental evaluation on real data shows the effectiveness of the approach in terms of both computation and communication cost, as well as utility.

In chapter 6, we presented the Lucene-P^2 framework which allows real/practical privacy-preserving information retrieval tasks. Under the framework, we developed three two-party privacy-preserving protocols that compute document similarities without disclosing the private input documents. We also discussed possible extensions to make the homomorphic encryption based protocols secure against other adversary models. Also, a secret sharing based method (SST) has been implemented into Lucene-P^2. However, SST requires an additional party as compared to SCC-LSI*, but it provides exceptionally improved performance.

7.1 Future Work

The Privacy-preserving Friend Recommendation work, presented in detail in chapters 3 and 6, can be extended any many directions. We plan to extend it in the following ways.

(a) We plan to implement the PPFR using secret sharing methods. In this work, see chapter 2, we have used additive homomorphic encryption and Differential Privacy. We determine that using additive secret sharing could further enhance its efficiency.

(b) We plan to investigate achievement of fault-tolerance in the malicious model for PPFR.

Acknowledgements

The authors wish to thank the anonymous reviewers for their invaluable feedback and suggestions. The content is solely the responsibility of the authors and does not necessarily represent the official views of the agencies funding the research. Grant details follow below.

- The 'Privacy-Preserving Friend Recommendation based on Common Friends' research, see chapter 3, reported in this publication was supported by the National Science Foundation under awards $CNS-1564034$ and the National Institutes of Health under awards R01GM118574 and R35GM134927.

- The 'Privacy-Preserving Friend Recommendation based on Similar Profiles' research, see chapter 6, was sponsored, in part, by grants from the National Institutes of Health (R01HG006844 and RM1HG009034) and National Science Foundation ($CNS-1855391$).

Bibliography

[1] Chao-Min Chiu, Meng-Hsiang Hsu, and Eric T.G. Wang. Understanding knowledge sharing in virtual communities: An integration of social capital and social cognitive theories. *Decision Support Systems*, 42(3):1872 – 1888, 2006.

[2] John T. Jost, Pablo Barberá, Richard Bonneau, Melanie Langer, Megan Metzger, Jonathan Nagler, Joanna Sterling, and Joshua A. Tucker. How social media facilitates political protest: Information, motivation, and social networks. *Political Psychology*, 39(S1):85–118, Feb. 2018.

[3] Panos Balatsoukas, Catriona M Kennedy, Iain Buchan, John Powell, and John Ainsworth. The Role of Social Network Technologies in Online Health Promotion: A Narrative Review of Theoretical and Empirical Factors Influencing Intervention Effectiveness. *Journal of Medical Internet Research*, 17(6):e141, jun 2015.

[4] Liliana Laranjo, Amaël Arguel, Ana L Neves, Aideen M Gallagher, Ruth Kaplan, Nathan Mortimer, Guilherme A Mendes, and Annie Y S Lau. The influence of social networking sites on health behavior change: a systematic review and meta-analysis. *Journal of the American Medical Informatics Association*, 22(1):243–256, 2015.

[5] Konstantinos Papamiltiadis. Let's clear up a few things about facebook's partners. `"https://about.fb.com/news/2018/12/facebooks-partners/"`, Dec 2018.

[6] Gabriel J. X. Dance, Michael LaForgia, and Nicholas Confessore. As facebook raised a privacy wall, it carved an opening for tech giants. *The New York Times*, December 2018.

[7] Jennifer Valentino-DeVries. 5 ways facebook shared your data. *The New York Times*, December 2018.

[8] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[9] Piotr Mardziel, Michael Hicks, Jonathan Katz, and Mudhakar Srivatsa. Knowledge-oriented secure multiparty computation. In *Proceedings of the 7th Workshop on Programming Languages and Analysis for Security*, pages 1–12, 2012.

[10] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 451–468, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[11] Genqiang Wu, Yeping He, JingZheng Wu, and Xianyao Xia. Inherit differential privacy in distributed setting: Multiparty randomized function computation. *CoRR*, abs/1604.03001, 2016.

[12] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[13] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.

[14] Oded Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols. Cambridge University Press, 2004.

[15] Xiaojian Zhang, Rui Chen, Jianliang Xu, Xiaofeng Meng, and Xie Yingtao. Towards accurate histogram publication under differential privacy. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 587–595. SDM, 04 2014.

[16] J. Xu, Z. Zhang, X. Xiao, Y. Yang, and G. Yu. Differentially private histogram publication. In *2012 IEEE 28th International Conference on Data Engineering*, pages 32–43, April 2012.

[17] Yu-Hsuan Kuo, Cho-Chun Chiu, Daniel Kifer, Michael Hay, and Ashwin Machanavajjhala. Differentially private hierarchical count-of-counts histograms. *Proc. VLDB Endow.*, 11(11):1509–1521, July 2018.

[18] Soheila Ghane, Lars Kulik, and Kotagiri Ramamohanarao. Publishing spatial histograms under differential privacy. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, SSDBM '18, New York, NY, USA, 2018. Association for Computing Machinery.

[19] Wei-Yen Day, Ninghui Li, and Min Lyu. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 123–138, New York, NY, USA, 2016. Association for Computing Machinery.

[20] Y. Mülle, C. Clifton, and K. Böhm. Privacy-integrated graph clustering through differential privacy. *CEUR Workshop Proceedings*, 1330:247–254, 01 2015.

[21] L. Guo, C. Zhang, and Y. Fang. A trust-based privacy-preserving friend recommendation scheme for online social networks. *IEEE Transactions on Dependable and Secure Computing*, 12(4):413–427, 2015.

[22] Xindi Ma, Jianfeng Ma, Hui Li, Qi Jiang, and Sheng Gao. Armor: A trust-based privacy-preserving framework for decentralized friend recommendation in online social networks. *Future Generation Computer Systems*, 79:82 – 94, 2018.

[23] Bharath K. Samanthula, Lei Cen, Wei Jiang, and Luo Si. Privacy-preserving and efficient friend recommendation in online social networks. *Trans. Data Privacy*, 8(2):141–171, August 2015.

[24] Shiwen Zhang, Xiong Li, Haowen Liu, Yaping Lin, and Arun Kumar Sangaiah. A privacy-preserving friend recommendation scheme in online social networks. *Sustainable Cities and Society*, 38:275 – 285, 2018.

[25] Udi Manber. Finding similar files in a large file system. In *USENIX Winter*, 1994.

[26] Yaniv Bernstein and Justin Zobel. Accurate discovery of co-derivative documents via duplicate text detection. *Inf. Syst.*, 31(7):595–609, November 2006.

[27] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, page 76–85, New York, NY, USA, 2003. Association for Computing Machinery.

[28] Federica Mandreoli, Riccardo Martoglia, and Paolo Tiberio. A document comparison scheme for secure duplicate detection. *International Journal on Digital Libraries*, 4(3):223–244, 2004.

[29] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29, 1997.

[30] Sergey Brin, James Davis, and Héctor García-Molina. Copy detection mechanisms for digital documents. *SIGMOD Rec.*, 24(2):398–409, May 1995.

[31] Antonio Si, Hong Va Leong, and Rynson W. H. Lau. Check: A document plagiarism detection system. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, SAC '97, page 70–77, New York, NY, USA, 1997. Association for Computing Machinery.

[32] Christian S. Collberg, Stephen G. Kobourov, Joshua Louie, and Thomas Slattery. Splat: A system for self-plagiarism detection. In *ICWI*, 2003.

[33] D. Sorokina, J. Gehrke, S. Warner, and P. Ginsparg. Plagiarism detection in arxiv. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 1070–1075, 2006.

[34] Yaniv Bernstein, Milad Shokouhi, and Justin Zobel. Compact features for detection of near-duplicates in distributed retrieval. In Fabio Crestani, Paolo Ferragina, and Mark Sanderson, editors, *String Processing and Information Retrieval*, pages 110–121, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[35] Narayanan Shivakumar and Hector Garcia-Molina. Scam: A copy detection mechanism for digital documents. In *DL*, 1995.

[36] Narayanan Shivakumar and Hector Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *Proceedings of the First ACM International Conference on Digital Libraries*, DL '96, page 160–168, New York, NY, USA, 1996. Association for Computing Machinery.

[37] Hui Yang and Jamie Callan. Near-duplicate detection by instance-level constrained clustering. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 421–428, New York, NY, USA, 2006. Association for Computing Machinery.

[38] Wei Jiang and Bharath K. Samanthula. N-gram based secure similar document detection. In Yingjiu Li, editor, *Data and Applications Security and Privacy XXV*, pages 239–246, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[39] Mummoorthy Murugesan, Wei Jiang, Chris Clifton, Luo Si, and Jaideep Vaidya. Efficient privacy-preserving similar document detection. *The VLDB Journal*, 19(4):457–475, 2010.

[40] W. Jiang, M. Murugesan, C. Clifton, and L. Si. Similar document detection with limited information disclosure. In *2008 IEEE 24th International Conference on Data Engineering*, pages 735–743, 2008.

[41] Nik Unger, Sahithi Thandra, and Ian Goldberg. Elxa: Scalable privacy-preserving plagiarism detection. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, WPES '16, page 153–164, New York, NY, USA, 2016. Association for Computing Machinery.

[42] H. H. Pang, X. Xiao, and J. Shen. Obfuscating the topical intention in enterprise text search. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1168–1179, 2012.

[43] Avi Arampatzis, George Drosatos, and Pavlos S Efraimidis. Versatile query scrambling for private web search. *Information Retrieval Journal*, 18(4):331–358, 2015.

[44] Marc Juarez and Vicenç Torra. *DisPA: An Intelligent Agent for Private Web Search*, pages 389–405. Springer International Publishing, Cham, 2015.

[45] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. Secure multidimensional range queries over outsourced data. *The VLDB Journal*, 21(3):333–358, 2012.

[46] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology – ICISC 2012*, pages 1–21, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[47] Yanbin Lu. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *NDSS*, 2012.

[48] Seung Geol Choi, Ariel Elbaz, Ari Juels, Tal Malkin, and Moti Yung. Two-party computing with encrypted data. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, pages 298–314, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[49] C. Wang, K. Ren, Shucheng Yu, and K. M. R. Urs. Achieving usable and privacy-assured similarity search over outsourced cloud data. In *2012 Proceedings IEEE INFOCOM*, pages 451–459, 2012.

[50] C. Liu, L. Zhu, L. Li, and Y. Tan. Fuzzy keyword search on encrypted cloud storage data with small index. In *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, pages 269–273, 2011.

[51] M. Kuzu, M. S. Islam, and M. Kantarcioglu. Efficient similarity search over encrypted data. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1156–1167, 2012.

[52] M. Raykova, A. Cui, B. Vo, B. Liu, T. Malkin, S. M. Bellovin, and S. J. Stolfo. Usable, secure, private search. *IEEE Security Privacy*, 10(5):53–60, 2012.

[53] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[54] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith. Public key encryption that allows pir queries. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, pages 50–67, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[55] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 353–373, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[56] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[57] G. R. Blakley and C. Meadows. A database encryption scheme which allows the computation of statistics using encrypted data. In *1985 IEEE Symposium on Security and Privacy*, pages 116–116, 1985.

[58] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998.

[59] A. Beimel, Y. Ishai, E. Kushilevitz, and J. . Raymond. Breaking the o(n/sup 1/(2k-1)/) barrier for information-theoretic private information retrieval. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 261–270, 2002.

[60] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *International Conference on*

the Theory and Applications of Cryptographic Techniques, pages 402–414. Springer, 1999.

[61] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *Information Security*, pages 314–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[62] Ryan Henry, Femi Olumofin, and Ian Goldberg. Practical pir for electronic commerce. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 677–690, New York, NY, USA, 2011. Association for Computing Machinery.

[63] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 151–160, New York, NY, USA, 1998. Association for Computing Machinery.

[64] Clement J McDonald, Paul Dexter, Gunther Schadow, Henry C Chueh, Greg Abernathy, John Hook, Lonnie Blevins, J Marc Overhage, and Jules J Berman. Spin query tools for de-identified research on a humongous database. In *AMIA Annual Symposium Proceedings*, volume 2005, page 515. American Medical Informatics Association, 2005.

[65] Ran Canetti, Yuval Ishai, Ravi Kumar, Michael K. Reiter, Ronitt Rubinfeld, and Rebecca N. Wright. Selective private function evaluation with applications to private statistics. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, PODC '01, page 293–304, New York, NY, USA, 2001. Association for Computing Machinery.

[66] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3&#8211;4):211–407, August 2014.

[67] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *Theory of Cryptography*, pages 137–156, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[68] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[69] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[70] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[71] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109 – 132, 2013.

[72] Jie Bao, Yu Zheng, David Wilkie, and Mohamed Mokbel. Recommendations in location-based social networks: a survey. *GeoInformatica*, 19(3):525–565, Jul 2015.

[73] Jiliang Tang, Yi Chang, and Huan Liu. Mining social media with social theories: A survey. *SIGKDD Explor. Newsl.*, 15(2):20–29, June 2014.

[74] Schloss Dagstuhl. dblp: computer science bibliography. `"https://dblp.uni-trier.de/"`, July 2018.

[75] `https://gmplib.org/`. Gnu multiple precision arithmetic library, February 2020.

[76] Yadolah Dodge. Spearman rank correlation coefficient. In *The Concise Encyclopedia of Statistics*, pages 502–505. Springer New York, New York, NY, 2008.

[77] Llukan Puka. *Kendall's Tau*, pages 713–715. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[78] Kai Ming Ting. *Precision and Recall*, pages 781–781. Springer US, Boston, MA, 2010.

[79] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, August 2014.

[80] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[81] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[82] Hyunghoon Cho, David J Wu, and Bonnie Berger. Secure genome-wide association analysis using multiparty computation. *Nature biotechnology*, 36(6):547–551, 2018.

[83] EU Directive. 95/46/ec of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the EC*, 23(6), 1995.

[84] Apache lucene ™. `http://lucene.apache.org/`. Accessed: 2016-06-07.

[85] Accountability Act. Health insurance portability and accountability act of 1996. *Public law*, 104:191, 1996.

[86] "45 cfr 164.512(i)(1)(ii)". HIPAA, "https://www.govinfo.gov/app/details/CFR-2004-title45-vol1/CFR-2004-title45-vol1-sec164-512". 2002-10-01.

[87] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 1st edition, July 2008.

[88] M. F. Porter. *An Algorithm for Suffix Stripping*, page 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[89] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407, 1990.

[90] Mark Sanderson. Christopher d. manning, prabhakar raghavan, hinrich schütze, introduction to information retrieval, cambridge university press. 2008. isbn-13 978-0-521-86571-5, xxi 482 pages. *Natural Language Engineering*, 16(1):100–103, 2010.

[91] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[92] Kun Liu, H. Kargupta, and J. Ryan. Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):92–106, 2006.

[93] Özlem Uzuner, Yuan Luo, and Peter Szolovits. Evaluating the state-of-the-art in automatic de-identification. *Journal of the American Medical Informatics Association*, 14(5):550–563, 2007.

[94] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[95] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[96] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient rsa key generation and threshold paillier in the two-party setting. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, pages 313–331, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[97] Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *2012 IEEE Symposium on Security and Privacy*, pages 272–284, 2012.

[98] Samee Zahur and David Evans. Obliv-c: A language for extensible data-oblivious computation. *IACR Cryptol. ePrint Arch.*, 2015:1153, 2015.

[99] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, page 35, USA, 2011. USENIX Association.

[100] Florian Kerschbaum and Alessandro Sorniotti. Searchable encryption for outsourced data analytics. In Jan Camenisch and Costas Lambrinoudakis, editors, *Public Key Infrastructures, Services and Applications*, pages 61–76, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[101] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *2011 Proceedings IEEE INFOCOM*, pages 829–837, 2011.

[102] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 253–262, 2010.

[103] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Ndss*, volume 20, page 12. Citeseer, 2012.

[104] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998.

[105] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. One-way functions are essential for single-server private information retrieval. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, page 89–98, New York, NY, USA, 1999. Association for Computing Machinery.

[106] B. K. Samanthula, Y. Elmehdwi, and W. Jiang. k-nearest neighbor classification over semantically secure encrypted relational data. *IEEE Transactions on Knowledge and Data Engineering*, 27(5):1261–1273, 2015.

[107] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[108] Bharath K Samanthula and Wei Jiang. Interest-driven private friend recommendation. *Knowledge and Information Systems*, 42(3):663–687, 2015.

VITA

Nitish Milind Uplavikar was born and raised in Pune, Maharashtra, India. He obtained a Bachelor of Engineering (BE) in Computer Engineering from the University of Pune. After gaining two years of experience in the industry, his passion for research made him enroll in the MS program for Computer Science at Missouri University of Science and Technology, Rolla, USA, where he converted into the Doctor of Philosophy (Ph.D.) program under the guidance of his thesis-advisor professor Dr. Wei Jiang. Later on in the program, he transferred, along with his advisor, to the University of Missouri, Columbia, USA, where he pursued the degree further by researching and addressing problems within the area of Privacy-Preserving Data Analytics using Applied Cryptography.

Nitish's brother, Pritish, is a Computer Scientist as well, while his parents Milind and Shubhangi, along with his grandparents, Bhalchandra and Mangala, operate a family business in his hometown. Following the successful completion of his course of study, Nitish would like to contribute towards research and development within the area of data privacy and security for social benefit.