PRIVACY-PRESERVING DATA SHARING AND UTILIZATION BETWEEN ENTITIES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Didem Demirağ
July 2017

PRIVACY-PRESERVING DATA SHARING AND UTILIZATION BETWEEN ENTITIES

By Didem Demirağ

July 2017

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Erman Ayday(Advisor)

Fazlı Can

Ali Aydın Selçuk

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

PRIVACY-PRESERVING DATA SHARING AND UTILIZATION BETWEEN ENTITIES

Didem Demirağ
M.S. in Computer Engineering
Advisor: Erman Ayday
July 2017

In this thesis, we aim to enable privacy-preserving data sharing between entities and propose two systems for this purpose: (i) a verifiable computation scheme that enables privacy-preserving similarity computation in the malicious setting and (ii) a privacy-preserving link prediction scheme in the semi-honest setting. Both of these schemes preserve the privacy of the involving parties, while performing some tasks to improve the service quality. In verifiable computation, we propose a centralized system, which involves a client and multiple servers. We specifically focus on the case, in which we want to compute the similarity of a patient's data across several hospitals. Client, who is the hospital that owns the patient data, sends the query to multiple servers, which are different hospitals. Client wants to find similar patients in these hospitals in order to learn about the treatment techniques applied to those patients. In our link prediction scheme, we have two social networks with common users in both of them. We choose two nodes to perform link prediction between them. We perform link prediction in a privacy-preserving way so that neither of the networks learn the structure of the other network. We apply different metrics to define the similarity of the nodes. While doing this, we utilize privacy-preserving integer comparison.

Keywords: Verifiable computation, link prediction, data privacy, cryptography, homomorphic encryption, security.

ÖZET

KURUMLARARASI GİZLİLİĞİ KORUYAN VERİ PAYLAŞIMI

Didem Demirağ
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Erman Ayday
Temmuz 2017

Bu tezde amacımız, gizliliği koruyan kurumlararası veri paylaşımını gerçekleştirmektir. Bu amaçla iki farklı sistem önerilmiştir: (i) kötü niyetli modelde gizliliği koruyan benzerlik testi yapmayı sağlayacak doğrulanabilir hesaplamaya dayanan bir sistem ve (ii) yarı güvenilir modelde bağlantı tahmini yapan bir sistem. Önerilen bu sistemler, servis kalitesini arttrmak için gerekli görevleri yaparken, sistemde yer alan tarafların gizliliğini korumayı amaçlar. Doğrulanabilir hesaplamaya dayanan sistemimizde, bir istemci ve birden fazla sunucunun olduğu merkezi bir sistem öneriyoruz. Bu çalışmada, farklı hastanelerde bulunan hastaların benzerliklerinin hesaplandığı durumu ele alıyoruz. Hasta verisinin sahibi olan istemci farklı hastanelerde bulunan birden fazla sunucuya istek gönderir. Istemcinin amacı, bu hastanelerdeki benzer hastaları bulmak ve onlara uygulanan tedavi yöntemlerini öğrenmektir. Bağlantı tahminine dayanan sistemimizde ise ortak kullanıcılara sahip olan iki sosyal ağ bulunmaktadır. Aralarında bağlantı tahmini yapmak için iki kullanıcı seçilir. Gizliliği koruyan bir şekilde bağlantı tahmini gerçekleştirilir. Böylece sosyal ağlardan hiçbiri karşı tarafın sosyal ağ yapısını öğrenmez. Farklı metrikler hesaplanarak kullanıcıların benzerliği belirlenirken gizliliği koruyan sayı karşılaştırması kullanılmıştır.

Anahtar sözcükler: Doğrulanabilir hesaplama, bağlantı tahmini, veri gizliliği, kriptografi, şifreleme, homomorfik şifreleme, güvenlik.

Acknowledgement

First and foremost, I would like to thank my advisor Asst. Prof. Dr. Erman Ayday for his support and diligence. His extensive knowledge and expertise in his research area has helped me to improve myself academically. His dedicated guidance paved the way for this process.

I would also like to thank my jury members Prof. Dr. Fazlı Can and Prof. Dr. Ali Aydın Selçuk for kindly accepting to be in my committee. I owe them gratitude for their valuable suggestions and insightful comments.

I would like to express my gratitude to Berrin Keyik Çelik whose endless support starting from my undergraduate studies always encouraged me. Her kindness and affection helped me to overcome the difficulties I had and appreciate all the nice things happened.

I would like to thank my friends Anisa, Nora and Saharnaz, for all the nice chats that gave me energy and hope. We learned a lot from each other, as we shared our ideas on academia, life and on many other topics. It was a pleasure to be accompanied by such nice friends.

I would like to show my gratitude to Dorukhan for exchanging ideas about different concepts that we worked on. The discussions for our school projects have been insightful to help me gain a new perspective. I had a chance to learn from his meticulous coding skills.

I would like to thank TUBITAK for supporting this thesis with the project entitled GENEMETRIC: Cloud Based Genetic Biometric Identification System (Project ID: 115E766).

I would like to express my gratitude to my parents for their love and support. I am grateful to them for presenting me so many opportunities throughout my life. I would like to thank my dear grandmother, who has a profound influence on who I am now. And lastly, the loving memory of my grandfather, who is the

part of my best childhood memories with my grandmother, won't be forgotten.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In our research, our main concern is maintaining privacy when sharing, and analyzing personal information. Service providers (SP) can analyze their own databases without any problem. However, when they want to analyze other similar SPs to provide better service, concerns regarding privacy arise. Some of them are:

- The SP which makes the query should only learn the result of this authorized query and nothing more.
- The SP which makes the query should make sure that the result of the query is computed correctly.
- The SP which receives query wants to be sure that its database is not visible to the SP which makes the query.

To this end, this thesis introduces two systems that enable privacy-preserving data sharing between entities: a verifiable computation scheme that performs privacy-preserving similarity calculation in the malicious setting and privacy-preserving link prediction in the semi-honest setting. Both systems enable involving parties to maintain their privacy, while performing the required tasks to improve their service quality.

Our verifiable computation scheme addresses the fact that the cloud computing service providers want to analyze personal and sensitive data like patient records, banking information or location data, in order to provide better service to their clients. Hence, these applications need to adopt a privacy-preserving approach to ensure the privacy of the individuals' data is not harmed. In our verifiable computation scheme, we consider the setting, where we perform privacy-preserving similarity check to determine patient similarity across different hospitals. Our aim is to find similar patients that received treatment for a similar disease in different hospitals. In order to increase the effectiveness of the treatment, data from similar patients across other hospitals can be utilized. However, concerns regarding privacy arise, as we need to keep the patient information in all of the hospitals private.

In semi-honest setting, the parties follow the protocol, but they may attempt to breach the privacy of the involving parties to learn more information that belong to them; whereas in the malicious setting the parties can deviate from the protocol in an arbitrary way to gain advantage. In the semi-honest setting, homomorphic encryption would be enough to maintain the privacy of the sensitive information. However, in the malicious setting, we need to combine several concepts like verifiable computation, and secret sharing mechanisms in order to maintain the privacy. Verifiable computation protocols help a client with limited computational capacity to outsource a computation to another party. Because of the concerns about privacy, the server needs to provide a proof that it made the correct calculation and the client should be able to verify this easily. Verification should take less effort than actually doing the computation.

Considering these privacy concerns, we also address the link prediction problem in online social networks and we propose privacy-preserving link prediction between two social network graphs. With the increase in the amount of research done about large networks, computational analysis of social networks is needed. As more social networks emerge, the popularity of these sites increases and more people start to have accounts in them. For instance, millions of users have accounts in services like Facebook or Twitter and huge amount of data accumulate each day, as users share content in their social network accounts. Therefore, the

need to analyse those networks arise and as a result graph mining has become a significant area as a branch of data mining.

Graph mining is used to analyze graph-structured data such as social networks. Graphs are structures that consist of nodes and edges between the nodes. A social network graph consists of nodes which represent entities and edges that define the relationships between the nodes. For instance, the social network graph for Twitter consists of nodes corresponding to users with their attributes and edges corresponding to the follower relationship between the nodes. Social networks are dynamic and go under change frequently. Their dynamic structure should be considered, while doing research on social network graphs.

Social Network Analysis (SNA) is the area of study that analyzes the social structures, social positions, and the roles. SNA is applied in the area of graph mining. There are different social network analysis methods like centrality analysis, community detection, or information diffusion. Another significant analysis done on social network graphs is link prediction. Link prediction defines the important linkages between the nodes. By utilizing the analysis of these linkages, we can predict the future connections or determine missing links between the nodes.

During this analysis concerns about privacy arise, since social networks contain personal and sensitive information that should be held secret. Threats against privacy can be categorized into three groups [1]: identity disclosure, link disclosure, and attribute disclosure. All these threats should be considered in a social network analysis algorithm. Considering these threats, we aim to perform link prediction without disclosing any information from both graphs. We find intersection and union between two neighbor sets and we use them in the calculation of different metrics for link prediction.

Hence, we examine the link prediction problem, while preserving privacy. Our aim is to develop algorithms for link prediction without disclosing any data from either graph. We propose schemes to compute the common neighbors of two nodes in order to predict whether there will be link between two nodes.

The thesis is organised as follows: In Chapter 2, we analyze the existing literature on the both verifiable computation and link prediction. In Chapter 3, we define the basic concepts that are used in our proposed schemes. In Chapter 4, the privacy-preserving similarity computation in malicious setting is explained. While we propose several solution to this problem, we give the details for the centralized solution by also providing the evaluation results. In Chapter 5, the proposed solution for privacy preserving link prediction is presented. While we discuss different use cases, we propose protocols for different metrics. In Chapter 6, we present our concluding remarks.

Chapter 2

Literature Survey

2.1 Techniques for verifiable computation

2.1.1 Privacy-preserving data analysis

Lindell and Pinkas investigated secure multi-party computation and discussed its relation with privacy-preserving data mining [2]. Rivest et al. proposed privacy homomorphism to make calculations on encrypted data [3]. They assumed that the encrypted data and the computation done on the data is chosen from an algebraic system. However, in this paper a proof about the correctness of the calculation, like in verifiable computation protocol, is not provided. In our proposed work, we make a computation on encrypted data like in this paper, but in addition to that, we also provide a proof that shows the computation is done correctly to the client. Gennaro and Wichs, proposed a system that anyone can do arbitrary calculations on authenticated data using fully homomorphic message authenticators [4]. They produce tags that authenticate the result of the computation. The tag can be generated even though the secret key is not known. However, techniques based on fully homomorphic encryption is not practical and there does not exist any practical application. Chung et al. offered a more improved scheme for delegating computation using fully homomorphic encryption [5] following the

work of Gennaro et al. De Cristofaro et al. discussed the problem of privacy-preserving sharing of sensitive information [6]. Proposed techniques serve as a privacy shield and it prevents both sides to reveal more than the minimum information required. Private set intersection is used in this solution. However the security proofs of the proposed solutions are done in the semi-honest setting. We propose a solution for malicious setting, where the client and the server don't trust each other about the results of the computations they make. Hence, they need to produce proofs for their computations and these proofs are verified by the other party.

Freedman et al. worked on the problem of determining the intersection of two private data sets [7]. They proposed a scheme based on homomorphic encryption and balanced hash under both semi-honest and malicious settings. However, they do not focus on the validity of the query and the inferences that can be made from the query's result are not considered. Moreover, this technique is limited to set intersection. Lastly, Hazay and Toft proposed a protocol for secure multi-party pattern matching [8]. Proposed scheme is based on ElGamal encryption and its security is proven under standard DDH (Decisional Diffie-Hellman) assumption. Full simulation is done under the presence of malicious adversaries. However, this work also has some similar disadvantages with the previous work.

2.1.2 One-sided verifiable computation techniques

Some works offered protocols to delegate computation using verifiable computation. Fiore et al worked on efficient and verifiable delegation of computation [9]. In this proposed solution, client can hold his encrypted data in the server and run statistical queries on his data. He receives the results in encrypted form and he can verify the correctness of the results. Gennaro et al. proposed a protocol for client to delegate his computation using garbled circuits [10]. Benabbas et al. worked on the problem of doing computation on data that is stored on an untrusted server [11]. Backes et al. discussed the setting that the client stores big amount of data in an untrusted server and asks the server to do computation

on his data [12]. Main contribution of this work is homomorphic MAC. Parno et al. proposed a system called Pinocchio [13]. In this system, client creates a public key that describes his computation. Worker evaluates a computation done on input data and uses the evaluation key to produce proof of correctness. Shoenmakers et. al. proposed a system called Trinocchio [14]. In this work, Pinoochio's verifiable computation scheme is improved by providing input privacy. Costello et al. proposed a system called Geppetto [15]. Geppetto aims to reduce prover overhead and increase prover flexibility. Lastly, Fournet et al. created a query language called ZQL to express simple computations done on private data [16]. In this system, the party that has the private personal information can do the computation on behalf of the other party who asks for that computation. Then, he will provide a proof, that shows correct data is used during the computation, to the other party. This can be done using zero-knowledge proofs. Compared to these schemes, we work on verifiable computation techniques where two or more parties, each keeping private data, are involved.

2.1.3 Two or more sided verifiable computation techniques

Baron et al. worked on the problem of wildcard and substring matching in malicious setting [17]. Server holds a text of length n and the client wants to match a pattern of length m with server's text. However, in this setting the practicality of the protocol will be low if client wants to send queries to multiple servers. Moreover, in our proposed work, we assume that both sides can be malicious. Gordon et al. worked on multi-client verifiable computation and aimed to have a stronger security [18]. In this work, N clients make a computation on common data. However, the security of the system is not defined for the setting where both of the parties act maliciously. In our proposed work, we assume that both client and server can be malicious. Therefore, both server and the client should provide a proof that they made the correct computations.

2.2 Techniques for link prediction

Leicht et al. defined a measure for the similarity of vertices in networks and they base their work on the fact that two nodes are similar if their immediate neighbors are also similar themselves [19]. They formulate the similarity by using an adjacency matrix. In our work, we utilize different similarity measures to perform link prediction between two graphs in a privacy-preserving manner.

In [20], a method based on local random walk with low complexity is proposed for missing link prediction problem. A random walk is a Markov chain that determines the sequence of nodes visited by a random walker. They propose two similarity indices: Local Random Walk (LRW) index and the Superposed Random Walk (SRW) index.

Yu et al. define Gaussian Processes (GPs) for directed, undirected, and bipartite networks [21]. The proposed framework indicates a connection between link prediction and transfer learning. Their algorithm can scale linearly to the number of edges. Their model can be applied to link prediction problem.

In [22], an effective general link formation prediction framework, Mli (Multi-network Link Identifier) is presented. This framework enhances the link prediction results in partially aligned networks. They solution utilizes the meta-path concept.

The study presented in [23], proposes a definition for link recommendation across heterogenous networks. In addition to supervised methods, they also use unsupervised methods like Common Neighbors, Adamic/Adar and Jaccard Index. They propose ranking factor graph model.

In [24], link prediction in coupled networks is studied and CoupledLP is proposed. They utilize the structure information of source network and the interactions between source and target networks. They want to predict missing link in the target network by using the structure information in the source.

Tang et al. propose a framework called TranFG to classify the social relationships by utilizing the information obtained from heterogenous networks [25]. Similar to the aforementioned work, they also try to predict the relationships in target network by observing the source network. While we are also aiming to perform link prediction, we also want to preserve the privacy of the involving parties without disclosing any information to either graph. When link prediction is applied in one network, there aren't any concerns about privacy, as there won't be the problem of disclosing any information to another party. However, when we want to perform link prediction in two different social networks, we need to address these concerns, as we don't want either party to learn the structure of the other party.

Chapter 3

Background

3.1 Verifiable computation

Verifiable computation protocols enable clients with limited computational capacity to outsource the computation of a function F on various inputs to one or more servers [26]. The server needs to provide a proof that it made the correct calculation and the client should be able to verify this easily. Verification should take less effort than actually doing the computation. Verifiable computation has several application areas. One of them can be the case of medical testing. For instance, Alice having her sequenced genome, which is her personal data, wants to perform a genetic testing. Hence, she provides her genome to the company that conducts the test. However, she also wants to make sure that the test results provided by the company are actually correct. In order to achieve this, company provides a proof along with the test result. Alice now can make sure that she received the correct result by verifying the proof.

Consider the example, where Alice sends her genome to a company to perform some tests. She has to be sure that the results she received is free of errors, otherwise the accidental errors can lead to some devastating results like wrong treatment or detrimental psychological effects. Also, the cloud services may have

the strong financial incentive to return incorrect answers, as producing them may require less computation power and the client may not have the chance to detect the error. For instance, in the case of Alice, she will not be able to understand the results regarding the analysis of her genome.

In order to address the aforementioned concerns, verifiable computation is utilized. A verifiable computation scheme consists of four algorithms as VC = (KeyGen,ProbGen,Compute,Verify) [26]:

1. "$KeyGen$"$(f, \lambda) \rightarrow (PK, SK)$: Randomized key generation algorithm produces a public key that encodes the function f depending on the security parameter λ and it is used by worker to calculate f. Client keeps the matching secret key as a secret.

2. "$ProbGen$"$_{SK}(x) \rightarrow (\sigma_x, \tau_x)$: Problem generation algorithm encodes the function input as public value σ_x using secret key SK and this value is given to the worker. Secret value τ_x is kept secret by the client.

3. "$Compute$"$_{PK}(\sigma_x) \rightarrow \sigma_y$: Worker computes encoded version of function's output value $y = f(x)$ using client's public key and the encoded input.

4. "$Verify$"$_{SK}(\tau_x, \sigma_y) \rightarrow y$: Verification algorithm converts the workers encoded output into the output of the function, $y = f(x)$, using secret key SK or the secret decoding value τ_x or outputs to show that σ_y does not represent a valid output of f on x.

3.2 Secret sharing

In secret sharing [27], data D is divided into n pieces and it is distributed among the parties in the protocol. Each party has a share of the secret and they have to gather the shares together to reconstruct the secret. The secret can be reconstructed from any k pieces. However, the data cannot be recovered even with the complete knowledge of k-1 shares.

3.3 Homomorphic encryption

Homomorphic encryption enables computation to be performed on encrypted data. The research done about computation on encrypted data mainly aimed to generalize the type of computations that can be done on encrypted data. Fully Homomorphic Encryption (FHE), which aimed this purpose, gained more attention with Gentry's work [28]. FHE proposes to support any kind of function that can be performed on encrypted data. Even though FHE proved to be theoretically possible, it has some shortcomings due to lacking efficiency [29]. In order to address the problem of efficiency, partially homomorphic encryption schemes are emerged. These schemes only allow certain types of operations to be performed on encrypted data. We use two partially homomorphic encryption schemes, namely Modified Paillier Cryptosystem and DGK Cryptosystem.

3.3.1 Modified Paillier cryptosystem

The Paillier cryptosystem [30] is a public key cryptosystem that supports some homomorphic operations. The public key is represented as $(n, g, h = g^x)$. The strong secret key is the factorization of $n = zy$ (z, y are safe primes), the weak secret key is $x \epsilon [1, n^2/2]$ and g of the order $(z-1)(y-1)/2$. By selecting a random $a \epsilon \mathbb{Z}^*_{n^2}$, g can be computed as $g = -a^{2n}$. Encryption, decryption and proxy re-encryption are explained as follows, where $[m]$ denotes the ciphertext corresponding to message m.

- Encryption: To encrypt a message $m \epsilon \mathbb{Z}_n$, we first select a random $r \epsilon [1, n/4]$ and generate the ciphertext pair (C_1, C_2) as below:

  $C_1 = g^r \ mod \ n^2$ and $C_2 = h^r(1 + mn) \ mod \ n^2$

- Decryption: The message m can be recovered from $[m]$, which denotes the encryption of m, as follows:

$m = \Delta(C_2/C_1^x)$ where $\Delta(u) = \frac{(u-1) \ mod \ n^2}{n}$, for all $u \ \epsilon\{u < n^2 \mid u = 1 \ mod \ u\}$

- Proxy re-encryption: Assume we randomly split the secret key in two shares $x_1 \ and \ x_2$, such that $x = x_1 + x_2$. The modified Paillier cryptosystem enables an encrypted message (C_1, C_2) to be partially decrypted to a ciphertext pair $(\tilde{C_1}, \tilde{C_2})$ using x_1 as below:

  $\tilde{C_1} = C_1$ and $\tilde{C_2} = C_2/C_1^{x_1} \ mod \ n^2$. Then, $(\tilde{C_1}, \tilde{C_2})$ can be decrypted using x_2 with the decryption function to recover the original message.

3.3.2 DGK cryptosystem

The DGK cryptosystem [31] is optimized for the secure comparison of integers. The key generation has three parameters k, t and L where $k > t > L$. The parameter k represents the number of bits of the RSA modulus n, t is the size of two small primes v_p and v_q, and L is the message space size in bits. Assume that p and q are two distinct primes of equal bit length, such that $p-1$ is divisible by v_p and $q-1$ is divisible by v_q. Then, the public key is represented as (n, g, h, u), where u is a L-bit prime. $g \ \epsilon \ \mathbb{Z}_n^*$ with order uv_pv_q, and h is an integer with order v_pv_q. Moreover, the private key is represented as (p, q, v_p, v_q).

3.3.3 Homomorphic properties

Both modified Paillier and DGK cryptosystems support some computations in ciphertext domain. Both cryptosystems have the following properties:

- The product of two ciphertexts is equal to the encryption of the sum of their corresponding plaintexts.
- A ciphertext raised to a constant number is equal to the encryption of the product of the corresponding plaintext and the constant.

Chapter 4

Privacy-Preserving Similarity Computation in Malicious Setting

In our setting, we aim to compute the similarity of a patient's data across several hospitals. Client is the hospital that owns the patient data about whom it will send the query to multiple servers. The servers from different hospitals contain data for different patients. Client's aim is to find similar patients in these hospitals so that it can learn about the similar treatment techniques applied to those patients. However, the client wants to trust the servers that they do this computation correctly and send the relevant result to the client's query. Thus, we utilize verifiable computation protocols among the client and the servers. After the client runs a query in multiple servers, it also needs to verify the proofs of those computations. If the client asks the query one by one to each server and then collects all the proofs, it will be inefficient for the client to verify all the proofs. In order to overcome this problem, we propose two different kinds of solutions. In distributed solution, after the first server receives the query from the client, it passes the result and the proof to the next server, until the result reaches the client. In the centralized solution, there is a proxy to collect all the results and the proofs to do the verification and pass the result to the client.

4.1 Centralized Solution

In this setting, we have a proxy to collect and verify all of the proofs (Figure 4.1). In this way, client will only verify one proof like in the distributed setting. Client sends his query to the proxy and then the proxy sends the query to these servers. Proxy collects the results and the corresponding proofs from the servers. It verifies all of the proofs and then creates one proof based on all of the data it received from the servers. Proxy verifies all of the results it received from the servers and generates only one proof corresponding to the results. Proxy sends the results and the proof to the client. As a result, the client has all the results from the servers and it will verify only one proof. Here, we also do not trust the proxy. That is why, the client checks proxy's proof created on the overall result.

In this section, single-server case is explained. We define a protocol between a server and a client. However, our scheme can easily be extended to multi-server setting. In the multi-server setting, the proxy will collect the results gathered from the servers. It means that the proxy will be responsible for the interaction with all of the servers. Proxy receives the matrix from the client and communicates with all of the servers for the results.

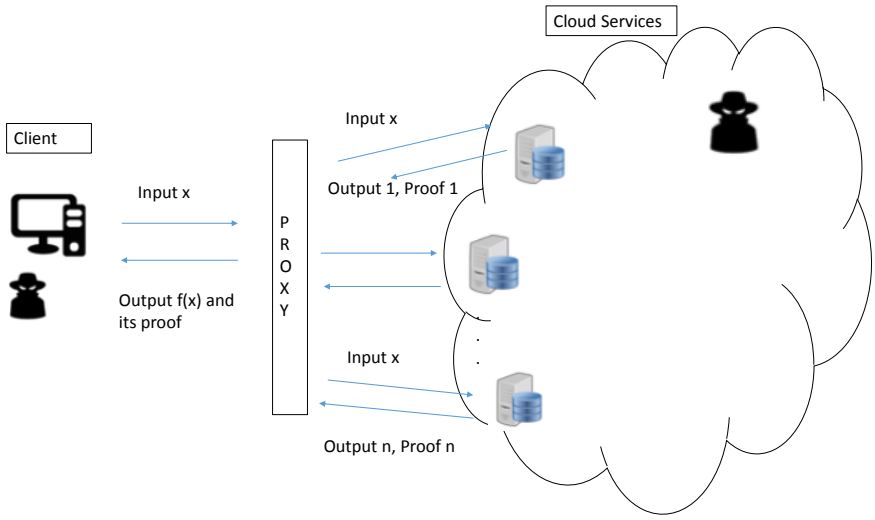


Figure 4.1: Centralized solution

Both client and server has medical data for patients. Client will make one

query about a patient in his database. However, on the server side calculation for all of the patients should be done using data that is sent by the client. Server has more data than the client, as server contains data for different patients. Client has only one record and server has m records. We can keep server's data in mxn matrix. Each row will correspond to a vector that is composed of binary numbers as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ z_{m1} & z_{m2} & z_{m3} & \dots & z_{mn} \end{bmatrix}$$

Client has a vector of size $1xn$, which is shown as follows:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \end{bmatrix}$$

However, the client also need to have the same size of the matrix at the server side, in order to perform the computation easily. That way, we can perform the operation row by row. Hence, the client will also keep his data in matrix. Matrix will again have the size of mxn and for each entry, the client will put the same data to match the size of the server's matrix as follows:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix}$$

where $x_{11} = x_{21} = ... = x_{m1}$, $x_{12} = x_{22} = ... = x_{m2}$ etc. Keeping client and server data in matrices enhances the step where we create signatures to prove the data is divided correctly and actually belongs to a certain party. Rather than dividing and signing each data separately, we can divide the matrix and create a signature over parts of the matrix.

Secret sharing is used to exchange the data. Client and server split their data into two parts. They send each other only one part of the data. So, both client

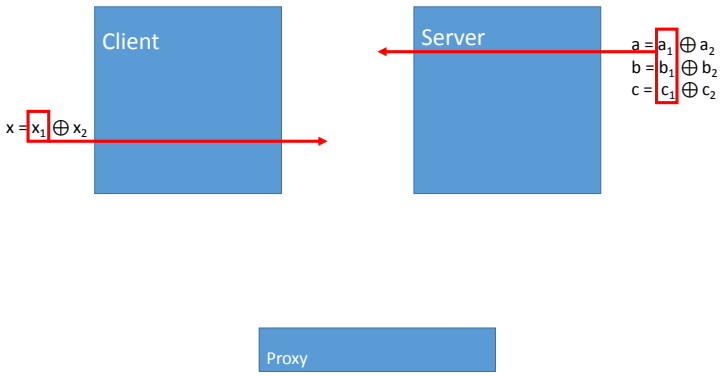


Figure 4.2: Secret sharing stage

and the server cannot recover the whole data that they received from the other party. For instance, the data at the server side can be divided as follows:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m1} & z_{m2} & \dots & z_{mk} \end{bmatrix}$$

and

$$\begin{bmatrix} a_{1(k+1)} & a_{1(k+2)} & \dots & a_{1n} \\ b_{2(k+1)} & b_{2(k+2)} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m(k+1)} & z_{m(k+2)} & \dots & z_{mn} \end{bmatrix}$$

where k is a number between 1 and n and chosen randomly. We also divide the client's matrix a similar way.

Moreover, we also assume that both parts of the data at each side are signed by an authority. It means that both parties use the data that they actually possess

and both parts belong to the whole data. As they are signed by an authority, we can easily verify them. For instance, let's consider the case where the binary data kept in the matrices correspond to sequenced DNA. In this case, the authority is the institution that sequences the DNA and signs it.

Client and server makes the computation on their sides and send the results to proxy. Proxy makes the rest of the computation with the partial results it received from both parties.

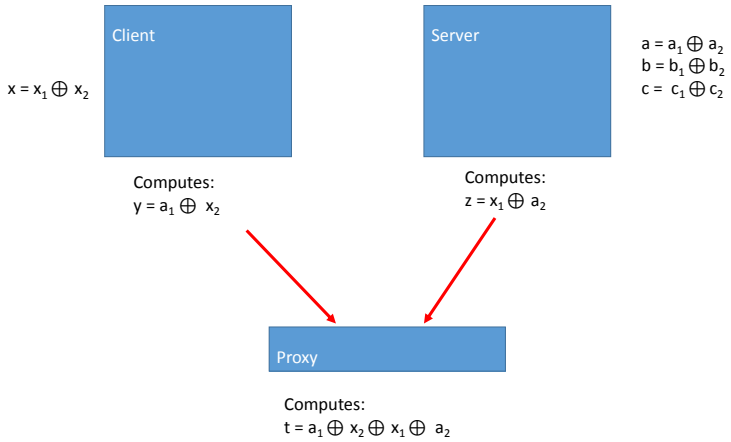


Figure 4.3: Proxy calculation

The protocol is executed as follows and the steps are also shown in Figure 4.4:

(1). Client and the server divide their data in two parts in such a way that no one can infer the whole data by only looking at the divided data.

(2). Client proves to server that he divided the data correctly and server proves to client that he divided the data correctly, by verifying the signatures over the parts of their data. The signature is applied over the ID of the patient together with the information about which part of the data is used.

(3). Client and server send each other one part of the matrix they divided. For simplicity, let's consider only one data row with only two elements: In Figure 4.2, client sends x_1 and server sends a_1.

(4). Client computes $y = a_1 \oplus x_2$ and server computes $z = x_1 \oplus a_2$, where $\oplus$ is the XOR operation. These computations will contribute to the overall result of the XOR operation on these patients data to determine similarity. As data on both sides are represented in binary format, performing XOR entry by entry will yield to the result of the similarity.

(5). Client proves to server that he did the calculation correctly and server creates the same proof.

(6). Client and server send the results of the computations to the proxy.

(7). Proxy checks the results that it received from the server and the client. It also checks whether client and server used the correct data to do the computation. If it is correct, proxy computes $y \oplus z$. Hence, it computes the overall result: $a_1 \oplus x_2 \oplus x_1 \oplus a_2$ (Figure 4.3). After calculating the overall result, proxy creates a proof of it.

(8). Proxy sends the result and the proof of its computation back to the client.

The aforementioned steps show the calculation for data with only two entries. When we work with matrices on both sides, we need to perform one more step at the proxy. Proxy uses the results he got from client and server in the final XOR operation and obtains a matrix that represents the overall results. However, he cannot send this matrix directly to the client, as client can reconstruct server's input from the result of the XOR operation and this would defeat our purpose of using secret sharing and introducing proxy to our system. Hence, in order to prevent this situation the proxy sums the entries in all of the rows and returns this vector back to the client. The vector has the following structure, where the entries s_1 to s_m correspond to the sum of each row:

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \end{bmatrix}$$

The client can determine a similarity threshold and determine which rows are higher than the threshold.

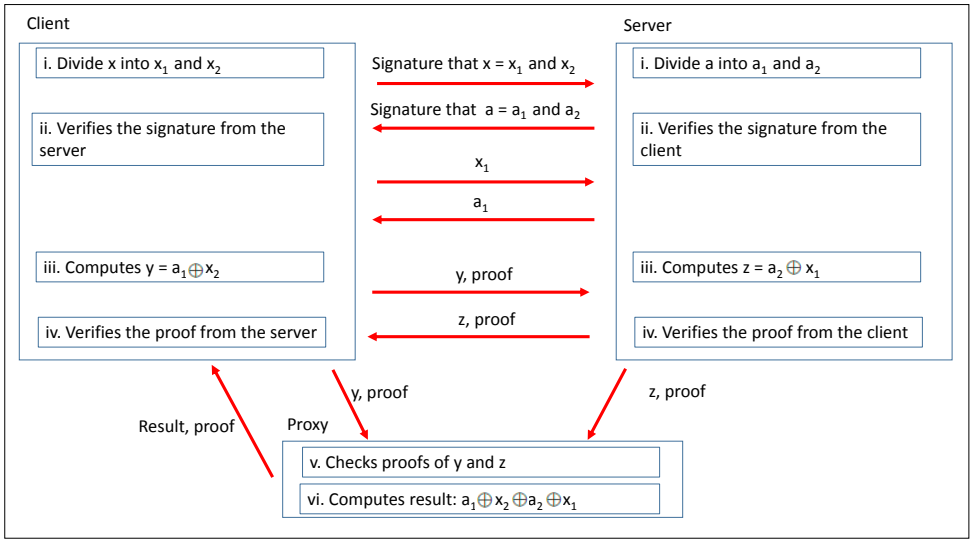


Figure 4.4: Overview of the protocol

4.2 Verifiable computation scheme

QAPs (Quadratic Arithmetic Programming) are used to define arithmetic operations. In order to use the QAPs utilized in Pinocchio's protocol, we can express XOR operation in terms of an arithmetic function:

$$a_1 \oplus a_2 = a_1(1 - a_2) + a_2(1 - a_1) \tag{4.1}$$

Verifiable Computation from strong QAPs [13] is defined as follows:

(1). $(EK_F, VK_F) \leftarrow KeyGen(F, 1^\lambda)$: F is a function with N input/output values from $\mathbb{F}$,which is the field of discrete logarithms of generator g of the group G. After converting F to an arithmetic circuit C, the corresponding QAP Q is created. Q with size m and degree d is defined as follows:

$$Q = (t(x),\ V,\ W,\ Y) \tag{4.2}$$

$I_{mid} = \{N+1, ..., m\}$ represents the non-IO-related indices. e is the non-trivial bilinear map that is defined as $e : G \times G \rightarrow G_T$. $s, \alpha, \beta_v, \beta_w, \beta_y, \gamma \xleftarrow{R} \mathbb{F}$ is chosen. Public evaluation key EK_F is defined as:

$(\{g^{v_k(s)}\}_{k \epsilon I_{mid}}, \{g^{w_k(s)}\}_{k \epsilon [m]}, \{g^{y_k(s)}\}_{k \epsilon [m]},$
$\{g^{\alpha v_k(s)}\}_{k \epsilon I_{mid}}, \{g^{\alpha w_k(s)}\}_{k \epsilon [m]}, \{g^{\alpha y_k(s)}\}_{k \epsilon [m]},$
$\{g^{\beta v_k(s)}\}_{k \epsilon I_{mid}}, \{g^{\beta w_k(s)}\}_{k \epsilon [m]}, \{g^{\beta y_k(s)}\}_{k \epsilon [m]},$
$\{g^{s^i}\}_{i \epsilon [d]}, \{g^{\alpha s^i}\}_{i \epsilon [d]})$
The public verification key is defined as
$VK_F = (g^l, g^\alpha, g^\gamma, g^{\beta_v \gamma}, g^{\beta_w \gamma}, g^{\beta_y \gamma}, g^{t(s)}, \{g^{v_k(s)}\}_{k \epsilon [N]}, g^{v_0(s)}, g^{w_0(s)}, g^{y_0(s)})$

(2). $(y, \pi_y) \leftarrow Compute(EK_F, u)$: The worker evaluates the circuit F on input u in order to obtain $y \leftarrow F(u)$. Consequently, he obtains the values $\{c_i\}_{i \epsilon [m]}$ of the circuit's wires. The worker solves for $h(x)$ in $p(x) = h(x).t(x)$. The proof π_y is computed as:
$(g^{v_{mid}(s)}, g^{w(s)}, g^{y(s)}, g^{h(s)}$
$g^{\alpha v_{mid}(s)}, g^{\alpha w(s)}, g^{\alpha y(s)}, g^{\alpha h(s)},$
$g^{\beta_v v(s) + \beta_w w(s) + \beta_y y(s)})$ where $v_{mid}(x) = \sum_{k \epsilon I_{mid}} c_k.v_k(x) = \sum_{k \epsilon [m]} c_k.v_k(x)$,
$w(x) = \sum_{k \epsilon [m]} c_k.w_k(x)$, and $y(x) = \sum_{k \epsilon [m]} c_k.y_k(x)$.

(3). $\{0, 1\} \leftarrow Verify(VK_F, u, y, \pi_y)$: Anyone who can access verification key VK_F can verify a proof. In order to do that, the pairing function e can be used to check that α and β proof terms are correct. For term α, we require 8 pairings and for β 3.

4.3 Evaluation

In our setting, we have two hospitals each has data for patients. The data are kept in binary format in matrices. We aim to perform XOR operation between to patient data of the client with all of the patients in the server. XOR is performed entry by entry and the result is again represented in a matrix. Matrices are created on Ubuntu 14.04.2 with 2GB of RAM, that runs on a virtual machine. In Figure 4.5, the time needed to create various-sized square matrices are shown.

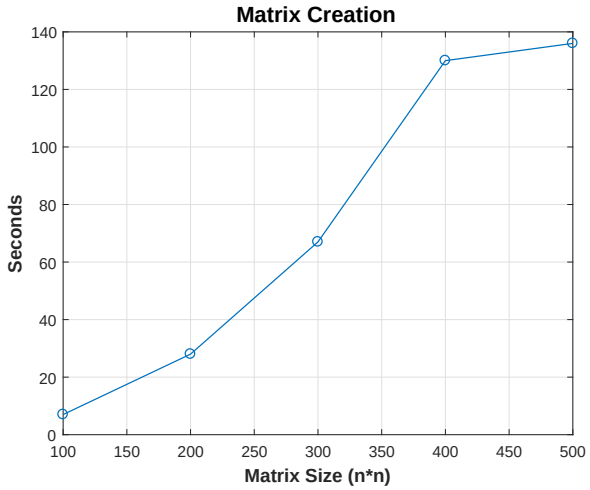


Figure 4.5: Matrix creation

We considered the case where there are 500 patients each of which has data in a vector with 1x1000000 size. We can divide each vector into 1x500 sized vectors. We can check 250 patients, using a 500x500 matrix whose first 250 entries are the same data of the patient and the second 250 entries are the patients from the hospitals. We need to create 2000 of these matrices in order to perform XOR operation on every entry of patient data. In total, we will need 4000 matrices, as we also need to check the rest of the patients in the hospitals. In order to obtain efficiency, we can parallelize the operations done on matrices. After creating the matrices, we run Pinocchio on Windows 8.1 platform with Intel(R) Core(TM) i7-4510U CPU @2.00GHz processor and 12.0 GB RAM and use these matrices as an input to Pinocchio.

In Figure 4.6, we show the amount of time needed to create the verification and evaluation keys for different sizes of square matrices. Figure 4.8 shows the total time needed to perform verifiable computation. Verification times for different sizes of matrices are shown in Figure 4.7. The computation at the client or at the server yield the same results in the previous figures, as they use the same-sized matrices.

After the calculation are done at client and server sides, proxy is responsible for doing the final calculation. He uses the results he got from client and server in the final XOR operation and obtains a matrix that represents the overall results.

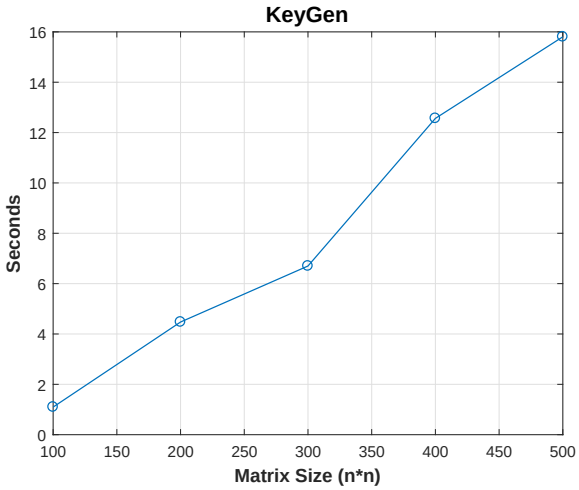


Figure 4.6: Key generation

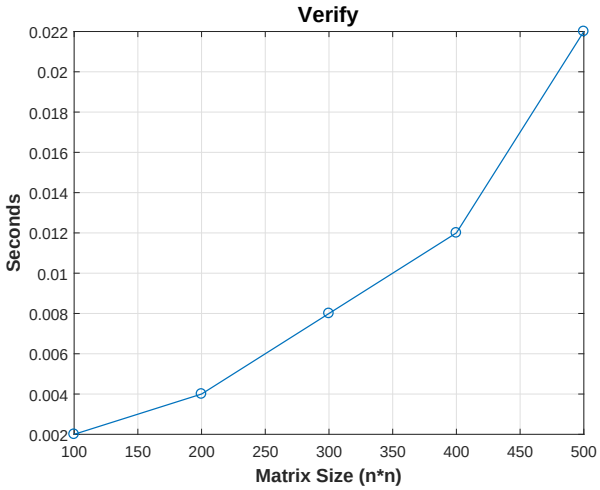


Figure 4.7: Verification

As he cannot send this matrix directly to the client, the proxy sums the entries in all of the rows and returns this vector back to the client. Figure 4.9 shows the time needed to generate the keys at the proxy. The total time needed to perform verifiable computation at proxy is shown in Figure 4.10.

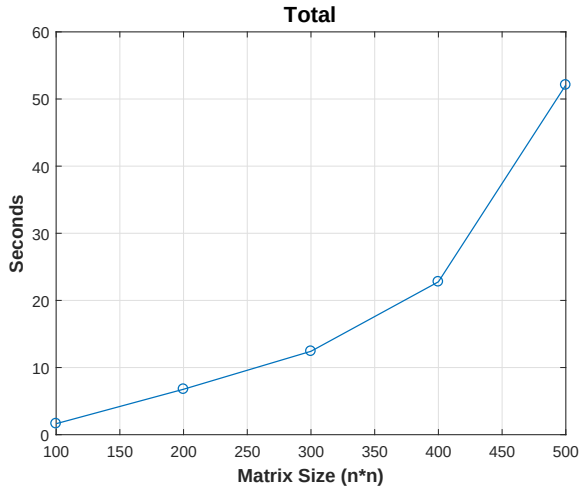


Figure 4.8: Total time to perform verifiable computation

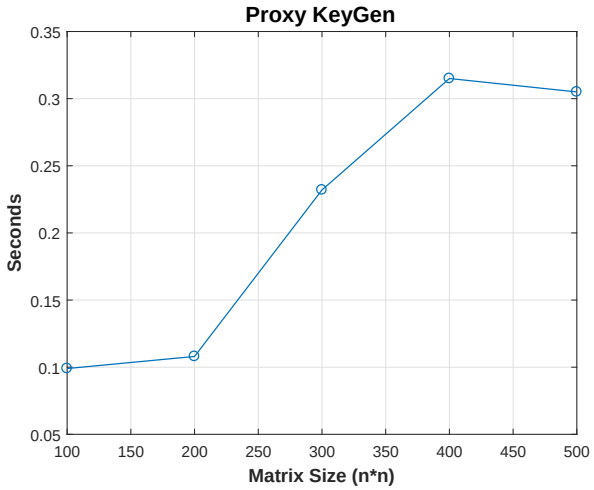


Figure 4.9: Key generation at proxy

4.4 Alternative Solutions

4.4.1 Distributed Solution

In this setting, client makes a query to one of the servers. The server prepares the result of the query and its proof and it sends them to the next server. This server verifies the computation and does the same computation and adds the result to the previous result and creates a proof over those results and this process goes on until the result and proof reaches to the client (Figure 4.11). In this way, client

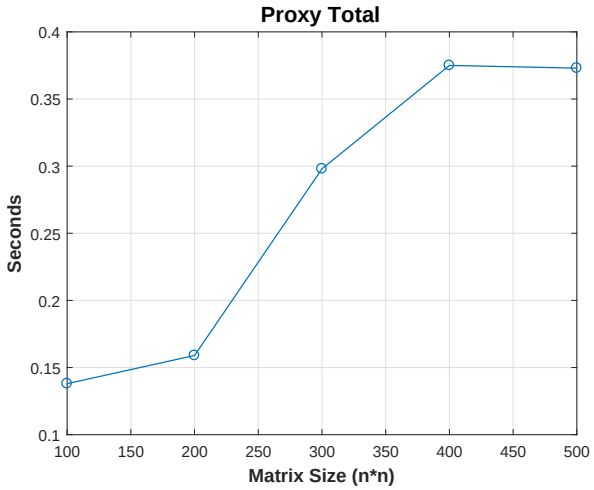


Figure 4.10: Total time to perform verifiable computation at proxy will only verify one proof.

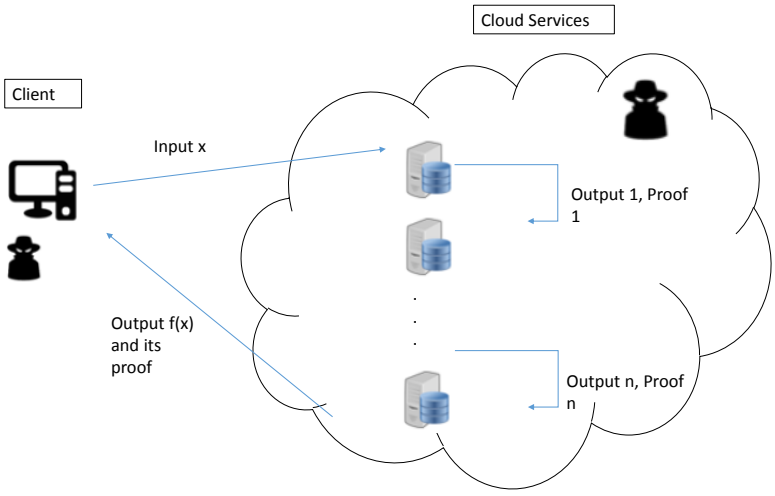


Figure 4.11: Distributed solution

4.4.2 Using Bloom Filter

Here, we can use the system proposed in [32] as an example. The hospital has IDs of its patients in the database and it creates a Bloom filter using these IDs. Let the length of the Bloom Filter be n and the number of hash functions be k.

The filter is encrypted. The client also puts the ID he wants to match in a filter and encrypts. Then they can multiply both filters and add the resulting entries. If the sum is equal to k, number of hash functions, then it means that there is a match. However, we need to handle the problem of false positives.

4.5 Future Work

Instead of using signatures to verify the data that involving parties possess, we can utilize Zero Knowledge Proofs (ZKP). Zero Knowledge Proofs (ZKPs) are used when the prover does not want to reveal anything about the proved statement. Hence, the only information the prover presents to the verifier is that the statement is true. We aim to adapt the zero knowledge proofs used in [33] and create ZKPs to show that the correct data is used during the computation. It is needed to prove that both sides divided the data correctly and these proofs should be done without revealing the data to the other party. Moreover, we need another zero knowledge proof to prove that the party actually used the other half of the data that he did not send to the other party. Our aim is to prevent a party from using a different data of his own and compute the digest with it.

We can also improve our system by using Geppetto [15] instead of Pinocchio [13], because Geppetto aims to reduce prover overhead and increase prover flexibility. We need to use Geppetto's proof generation for proving the correctness of the computation result. Hence, digest generation for calculations should be specified.

Our proposed system can also be applied to similar cases with different kinds of distributed systems. Some prominent examples of these systems can be SETI@Home [34], Folding@Home [35] and the Mersenne prime search [36]. All of these projects distribute computations to millions of clients in order to utilize their idle cycles. However, a significant problem arises with this advancement: dishonest clients who modify their software in such a way that they return results that are similar to the correct ones without actually performing any work [37].

These clients may be inclined to provide results without doing any computation with different incentives.

Chapter 5

Privacy-Preserving Link Prediction

Given a snapshot of a social network at time t, link prediction algorithms aim to predict the edges that will be formed in the network during the interval $t - t^{'}$, where $t^{'}$ represents a future time [38]. By defining the similarity between two nodes, link prediction algorithms will determine whether there will be a link between two nodes.

There exists two approaches to solve the link prediction problem: (i) In the first approach, the proximity of nodes the nodes are considered in the social networks. (ii) In the second approach, Bayesian probabilistic models, and probabilistic relational models are used [39] [40]. In Table 5.1, several different measures for calculating proximity is given.

Moreover, Common Neighbours, Jaccard's Coefficient and Adamic-Adar Index are regarded as the node-dependent indices and they are based on the node degree and the nearest neighborhood, whereas the Katz Index is defined as a path-dependent index that consider the global knowledge of the structure of the network [20].

Given two social networks, we have some common users in both of them. We

pick two nodes and apply different metrics to perform link prediction between these nodes. We aim to prevent the other network from having the knowledge of the whole graph. To this end, we propose a privacy-preserving approach to link prediction problem.

There can be several applications of privacy-preserving link prediction algorithms. Since we are performing similar computations on two different graphs, the graphs should have similar structures. For instance, there may be a phone operator that wants to propagate an advertisement about a service in one of the networks. The company wants to know which nodes are likely to form links between them, so that it can decide which nodes it will send the advertisement. It wants to maximize the number of nodes to whom it can offer a certain service. For this purpose, phone operator can utilize the similarity of certain nodes in a social network graph like Twitter or Facebook. Hence, we can perform privacy-preserving link prediction using a phone operator graph and a social network.

We can also perform link prediction operation between a service provider that provides streaming service such as Netflix, Amazon or Spotify and an online social network like Facebook. In order to make a good recommendation, we may utilize the information of what his friends on Facebook watch and how they rate them, while not revealing the friendship graph of Facebook to the Netflix network or vice versa.

5.1 Problem Definiton

In our problem setting, there are two social network graphs and they want to perform a graph mining task on their graphs without violating privacy. Both graphs contain nodes that correspond to users and edges between them that represent the relationship between them. They will compute the desired result without sharing their graphs. Our proposed system, prevents link disclosure and attribute disclosure attacks [1], as we don't let any party to learn the structure of the other party's graph. Moreover, in the Netflix use case, Section 5.3, we also

don't disclose the attributes, which are the movies that are watched and rated by a user, to Facebook graph.

We aim to execute link prediction algorithms in a privacy-preserving manner. Given two social networks, we want to find the similarity of two users, without either party disclosing their graphs to each other. We do the similarity computation by considering the structure of the graph. We mainly use privacy-preserving integer comparison [41], homomorphic encryption [30], and privacy-preserving set operations like intersection [42] [43]. The main technical challenge for privacy-preserving data mining is to make its algorithms scale and achieve higher accuracy while considering privacy [44].

Table 5.1: Similarity Metrics

similarity metric	**definition**
common neighbors	$\lvert\Gamma(x) \cap \Gamma(y)\rvert$
Jaccard's coefficient	$\frac{\lvert\Gamma(x)\cap\Gamma(y)\rvert}{\lvert\Gamma(x)\cup\Gamma(y)\rvert}$
Adamic/Adar	$\sum_{z\epsilon\lvert\Gamma(x)\cap\Gamma(y)\rvert} \frac{1}{log(\lvert\Gamma(z)\rvert)}$
$Katz_\beta$	$\sum_{l=1}^{\infty} \beta^l . \lvert path_{x,y}^{\langle l\rangle}\rvert$ where $path_{x,y}^{\langle l\rangle}$:= { paths of length exactly l from x to y} weighted: $path_{x,y}^{\langle l\rangle}$:= weight of the edge between x and y unweighted: $path_{x,y}^{\langle l\rangle}$:= 1 iff x and y are 1-hop neighbors

5.2 Proposed Solution

In link prediction, our aim is to predict whether there will be a link between two users. In order to achieve this, we need to define the similarity between these two nodes. Given two social networks, we have some common users in both of them. We pick two nodes, namely x and y, and define the common neighbors of them. The number of common neighbors will help us to perform link prediction. For this purpose, we apply different metrics to determine the similarity of two nodes. In Table 5.1, the similarity metrics that we use are shown. Here, the definitions

for different metrics to define the similarity of node pair $\langle x, y \rangle$ is given. $\Gamma(x)$ denotes the set of the neighbors of the node x [38].

A trivial solution for privacy-preserving link prediction can be holding a sorted list containing all of the nodes in the social network. If x is neighbor with a certain node, put one to the corresponding position of the that node otherwise put zero. However, this is not practical, since we need to keep a list which has a large size. The list contains many zeros, so it will be redundant to keep such a list. Moreover, it will not be practical to encrypt a big list and send it to the other party. Hence, we propose a more practical solution which does not include keeping a list of all nodes in a graph.

Networks are not independent of each other. In order to prevent the other network from having the knowledge of the whole graph, we can only send the subgraph containing the specific user with his neighbors. Therefore, we send the encrypted neighbor list of nodes to the other graph. We use different measures to determine links between nodes, while considering privacy. By trying these methods, we define whether privacy-preserving link prediction is computationally feasible.

While creating our scheme, we make the following assumptions:

(i). We assume that x and y are not neighbors in both of the graphs. Even if they are neighbors in one of the graphs, we do not consider that link.

(ii). The user IDs in both lists should match, otherwise we cannot make comparison between the neighbor lists of two graphs. We can use email or phone number to identify the users in both graphs.

(iii). Both graphs know that they are performing link prediction algorithm for nodes x and y.

In our setting, we have two online social network graphs. Let us denote our first graph as $G_1\langle V, E \rangle$ and second graph as $G_2\langle V, E \rangle$ where V denotes the set of nodes and E denotes the set of edges, i.e. follower relationship, between the

nodes (Figure 5.1). Let us represent the set of neighbors of a node i as $\Gamma(i)$. $G_1\langle V, E\rangle$ belongs to the client and $G_2\langle V, E\rangle$ belongs to the server. The client is the party that does the link prediction, namely Netflix and phone operator in our examples. The server is Facebook.

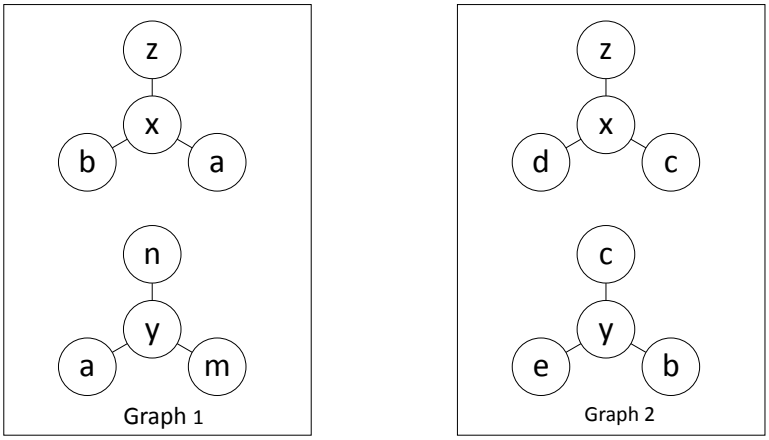


Figure 5.1: The neighbors of x and y in both graphs

We need to compare the IDs of the users in both graphs, as we will need to find the intersection and the union of the set of neighbors for calculating the similarity techniques which will be explained in the following sections in detail. For making comparison between the encrypted IDs in both networks, we use privacy-preserving integer comparison.

5.2.1 Privacy-preserving integer comparison

Here, we can compare two encrypted values without revealing the values to either side that is involved in the protocol. The result of the comparison is also encrypted.

We have $f(Enc(z), Enc(b))$, which denotes the comparison function between $Enc(z)$ and $Enc(b)$. $f(Enc(z), Enc(b)) = Enc(0)$ if $Enc(z) \geq Enc(b)$ and $f(Enc(z), Enc(b)) = Enc(1)$ if $Enc(z) < Enc(b)$.The details of the protocol is explained as follows:

Server's secret key x is randomly divided into x_1 and x_2, such that $x = x_1 + x_2$. x_1 is given to the server and x_2 is given to the client. Public and private keys of the server for the DGK cryptosystem are generated. The public key is shared with client. We also need to set symmetric keys to protect the message exchange between client and server from eavesdroppers. Let us denote the Paillier encryption of a as $Enc(a)$ and DGK encryption as $Enc_{DGK}(a)$. The protocol is as follows:

(i). At client: Client computes $Enc(z) = Enc(2^L + a - b)$. z_{L-1} represents the most significant bit of z. $z_{L-1} = 0$ if $a < b$ and $z_{L-1} = 1$ if $a \geq b$. Hence, the client needs to compute $Enc(z_{L-1}) = (z - (z \ mod \ 2^L))$. Since client can't compute this, it needs to start a privacy-preserving comparison protocol with the server. Client generates a random number r and computes $Enc(d) = Enc(z + r)$. Then, the client partially decrypts $Enc(d)$ using x_2 to obtain $\tilde{d}$ and sends it to server.

(ii). At server: Server decrypts $Enc(\tilde{d})$ using x_1 and obtains d. Then, it computes $(d \ mod \ 2^L)$, encrypts it to obtain $Enc(d \ mod \ 2^L)$. It uses client's public key and the modified Paillier cryptosystem. It sends the encrypted value to the client.

(iii). At client: Client computes $(r \ mod \ 2^L)$ and encrypts it to obtain $Enc(r \ mod \ 2^L)$. After that, it computes $\overline{Enc(z \ mod \ 2^L)} = Enc(d \ mod \ 2^L - r \ mod \ 2^L)$.
$\overline{Enc(z \ mod \ 2^L)} = Enc(z \ mod \ 2^L)$ if $Enc(d \ mod \ 2^L) \geq Enc(r \ mod \ 2^L)$. If $Enc(r \ mod \ 2^L) > Enc(d \ mod \ 2^L)$ underflow occurs, since the computation is done in modulo n. In order to prevent underflow, client should compute $Enc(z \ mod \ 2^L) = Enc(\overline{z \ mod \ 2^L} + \lambda 2^L)$, where $\lambda = 0$ if $(d \ mod \ 2^L) \geq (r \ mod \ 2^L)$, $\lambda = 1$ if $(r \ mod \ 2^L) > (d \ mod \ 2^L)$. Client needs to compute $Enc(\lambda)$ with the help of the server.

(iv). Computation of $Enc(\lambda)$: Here, DGK is used instead of Paillier, since it has an efficient multiplicative masking. Let $\hat{d} = (d \ mod \ 2^L)$, where $\hat{d}_i$ represents the i^{th} bit of $\hat{d}$, where $i \ \epsilon \ \{0, 1, ..., L-1\}$.

Server encrypts the bits of $\hat{d}$ using DGK to obtain $Enc_{DGK}(\hat{d_0}), ..., Enc_{DGK}(\hat{d_{L-1}})$ and sends them to client. Also, let $\hat{r} = (r \ mod \ 2^L)$ where $\hat{r_i}$ represents the i^{th} bit of $\hat{r}$, where $r \ \epsilon \ \{0, 1, ..., L-1\}$. Client encrypts the bits of $\hat{r}$ using public key of server and DGK encryption to obtain $Enc_{DGK}(\hat{r_0}), ..., Enc_{DGK}(\hat{r_{L-1}})$. Then,client chooses an integer s from the set $\{1, -1\}$ randomly and computes $C = \{Enc_{DGK}(c_0), ..., Enc_{DGK}(c_{L-1})\}$, where

$$Enc_{DGK}(c_i) = Enc_{DGK}(\hat{d_i} - \hat{r_i} + s + 3 \sum_{j=i+1}^{L-1} w_j) \tag{5.1}$$

where $w_j = \hat{d_j} \oplus \hat{r_j}$. $Enc_{DGK}(\hat{d_j} \oplus \hat{r_j}) = Enc_{DGK}(\hat{d_j} + \hat{r_j} - 2(\hat{r_j})\hat{d_j})$. This can be computed by the client. For each $Enc_{DGK}(c_i)$, the client selects random number α_i from $\mathbb{Z}_u$ and computes $Enc_{DGK}(e_i) = Enc_{DGK}(c_i \alpha_i)$, which masks $Enc_{DGK}(c_i)$ values. Client sends permuted $Enc_{DGK}(e_i)$ to server.

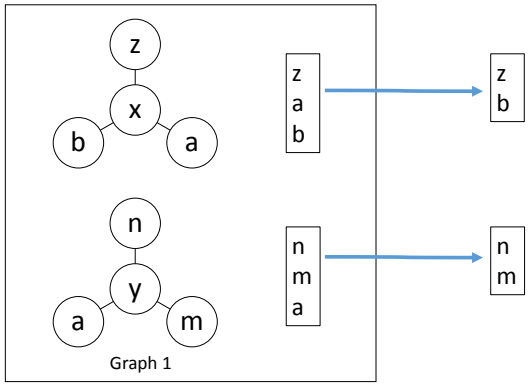
Server decrypts $Enc_{DGK}(e_i)$ with its private key. If all $Enc_{DGK}(e_i)$ are non-zero, server sets $a = 1$. If exactly one $Enc_{DGK}(e_i)$ value is different than zero then $a = 0$. Then server encrypts a and sends $Enc(a)$ to client.

If $a = 1$ and $s = 1$, where s is randomly selected by client, it means that $\hat{d} \geq \hat{r}$ and $\lambda = 0$ and if $a = 0$ and $s = 1$ it means that $\hat{r} > \hat{d}$ and $\lambda = 1$. Hence, if $s = 1$, client sets $Enc(\lambda) = Enc(1 - a)$. If $s = -1$, it sets $Enc(\lambda) = Enc(a)$. Client can compute $Enc(z \ mod \ 2^L)$ and $Enc(z_{L-1})$ using $Enc(\lambda)$.

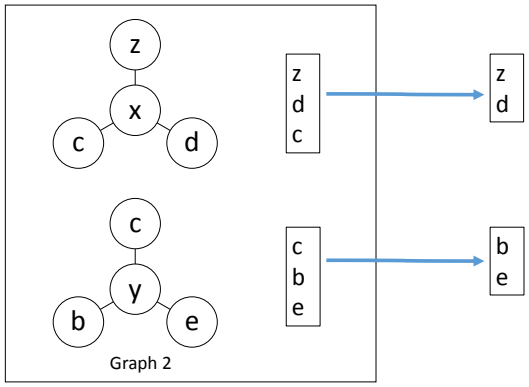
5.2.2 Common neighbors

We want to find the common neighbors of nodes x and y in both graphs. In order to compute this, we need to first create a list of neighbors of x and a list of neighbors of y in both graphs and according to those lists and we compute common neighbors as follows:

$$\text{common neighbors} = |\Gamma(x) \cap \Gamma(y)| \tag{5.2}$$



(a) Neighbor list for $G_1\langle V, E\rangle$



(b) Neighbor list for $G_2\langle V, E\rangle$

Figure 5.2: Finding list of neighbors of x and y and removing common neighbors

(i). When we find the common neighbors, we remove the common neighbors from the list, so that they are not counted twice as neighbors in the second graph. The common neighbors are added to the list. This procedure is done on both graphs as shown in Figure 5.2.

(ii). After the lists are created, client encrypts those lists with its public key and sends the encrypted lists to the server.

(iii). Server takes the encrypted neighbor list of x and makes a privacy-preserving integer comparison protocol (Section 5.2.1) with the neighbor list of y in its graph and adds to $|\Gamma(x) \cap \Gamma(y)|$. It does the same comparison between the encrypted neighbor list of y from the client with the neighbor list of x

in his graph. $|\Gamma(x) \cap \Gamma(y)|$ was equal to the number of common neighbors of x and y in server.

(iv). Before it sends the result to obtain the number of common neighbors of x and y in both graphs, one more step is needed. If the client adds both results directly, it might count common neighbors twice. For instance, assume a is also the common neighbor of x and y in the server. If the client directly adds two result, it will obtain the wrong value. In order to prevent this, we need another step.

(v). Client makes a list of common neighbors of x and y, in this case only a, and encrypts the list. Client sends the encrypted list to the server.

(vi). Server makes a privacy-preserving integer comparison with its own list of common neighbors of x and y and obtains the updated value of $|\Gamma(x) \cap \Gamma(y)|$ which does not contain the duplicate values.

(vii). Server sends the result back to client. Client can decrypt the result add to $|\Gamma(x) \cap \Gamma(y)|$. Here server obtains an encrypted result: $Enc(|\Gamma(x) \cap \Gamma(y)|)$. Therefore, server does not learn anything about the structure of the client.

(viii). Client decrypts the value it received from the server and adds to his previously computed value of $|\Gamma(x) \cap \Gamma(y)|$.

Let f be the comparison function, λ be an element from the set of neighbors in the neighbor list of the client and ϕ be an element from the set of neighbors in the neighbor list of the server. f takes two inputs and computes comparison protocol on them. So we can denote $Enc(|\Gamma(x) \cap \Gamma(y)|)$ as follows:

$$Enc(|\Gamma(x) \cap \Gamma(y)|) = \sum_{\lambda \epsilon \Gamma(x), \phi \epsilon \Gamma(y)} f(Enc(\lambda), Enc(\phi)) \tag{5.3}$$

In our work, we use the privacy-preserving integer comparison scheme proposed in [41]. According to this scheme, we have $f(Enc(z), Enc(b))$, which denotes the encrypted result of the comparison protocol between $Enc(z)$ and $Enc(b)$.

$f(Enc(z), Enc(b)) = Enc(1)$ if $Enc(z) \geq Enc(b)$ and $f(Enc(z), Enc(b)) = Enc(0)$ if $Enc(z) < Enc(b)$.

In our case, $f(Enc(z), Enc(b))$ consists of two functions f_1 and f_2 such that

$$f(Enc(z), Enc(b)) = f_1(1 - f_2) \tag{5.4}$$

where f_1 and f_2 denote the privacy-preserving integer comparison protocol for $Enc(z) \geq Enc(b)$ and $Enc(z) < Enc(b+1)$ respectively. By evaluating two inequalities, we will obtain the result for equality check. For instance, if $Enc(z) \doteq Enc(b)$, f_1 will yield 1 and f_2 will yield 0 to obtain f as 1.

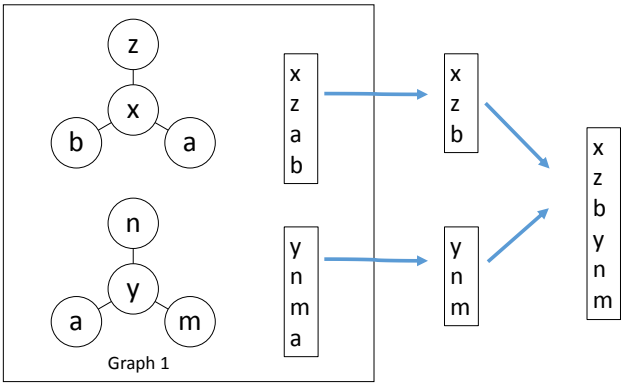
5.2.3 Jaccard's Coefficient

Another measure for calculating common neighbors is Jaccard's coefficient. It is computed as follows:

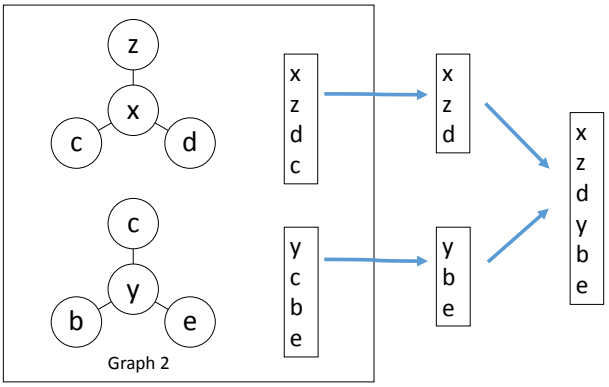
$$\text{Jaccard}'\text{s coefficient} = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|} \tag{5.5}$$

We can use the same procedure in the previous part to compute $|\Gamma(x) \cap \Gamma(y)|$. However, we need another scheme to compute the union. First of all, client needs to create the list of all neighbors of x and y. x and y should also be included in their respective lists. x compares each element in its list with y's list and eliminates the same nodes. In this case it is node a. Then client combines both lists to compute $|\Gamma(x) \cup \Gamma(y)|$. Server does the same procedure for x and y (Figure 5.3).

Then client encrypts the union list and sends to the server. Server compares its list with the list it received from the client. It obtains an encrypted result and sends back to the client.



(a) Neighbor list for $G_1\langle V, E\rangle$



(b) Neighbor list for $G_2\langle V, E\rangle$

Figure 5.3: Union operation

5.2.4 Adamic/Adar

In Adamic/Adar, we find $|\Gamma(x) \cap \Gamma(y)|$, with the same method in common neighbors. Then for each element in $|\Gamma(x) \cap \Gamma(y)|$, we sum the reciprocal of the logarithm of the number of neighbors of that element. In order to find the total number of neighbors of z in both graphs, namely client and server, we need to use the algorithm to find common neighbors in Section 5.2.2.

$$\text{Adamic/Adar} = \sum_{z \epsilon |\Gamma(x) \cap \Gamma(y)|} \frac{1}{log(|\Gamma(z)|)} \tag{5.6}$$

In order to compute Adamic/Adar, we need to find the union of the neighbors of each z, which is in the neighbor set of x, in both of the graphs. We can use the same algorithm proposed in Section 5.2.3 to find the union of the neighbors of z.

5.2.5 $Katz_\beta$

$Katz_\beta$ depends on computing the sum of all l-length paths between the nodes x and y. It is defined as follows:

$$\text{Katz}_\beta = \sum_{l=1}^{5} \beta^l . |path_{x,y}^{\langle l \rangle}| \tag{5.7}$$

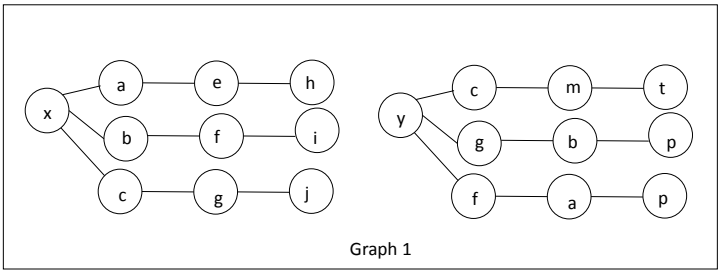
Here, $path_{x,y}^{\langle l \rangle}$ denotes the set of all l-length paths from x to y. We choose β in such a way that longer paths contribute less to the summation. According to [45], the average distance between two nodes is 4.7 for Facebook users and 4.3 for U.S. users. Hence, it will be enough to set l to at most 5.

For each l we find the following:

$$\text{N}_\text{n} = |\text{x's a}_\text{n} - \text{hop neighbors}| \cap |\text{y's b}_\text{n} - \text{hop neighbors}| \tag{5.8}$$

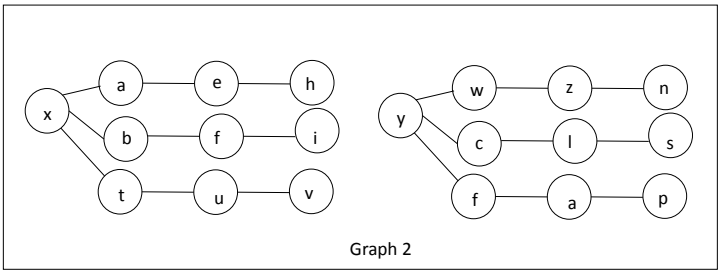
where $l = a_n + b_n$, $1 \leq a_n, b_n \leq l-1$ and $1 \leq n \leq l-1$. If any of the N_n values is greater than 0, then it means that x and y are l-hop neighbors of each other. For example, in order to find whether x and y have 3-hop neighbors, we need to find the value of N_1 for $a_1 = 1, b_n = 2$ and the value of N_2 for $a_1 = 2, b_n = 1$. If at least one of N_1 or N_2 is greater than 0 then it means that x and y are 3-hop neighbors of each other. In order to find n-hop neighbors, we need to first find the neighbors of our node. Then for each node in the neighbor set, we need to find their neighbors. We continue until we find n-hop neighbor list.

In order to find the number of l-length paths, we need to find the number of paths in the client and the server separately and then combine them to eliminate the duplicate ones (Figure 5.4). The algorithm is as follows:



|x's 2 hop neighbors| ∩ |y's 1 hop neighbors| = {f,g}
|x's 1 hop neighbors| ∩ |y's 2 hop neighbors| = {a,b}

(a) All possible neighbor lists for $G_1\langle V, E\rangle$



|x's 2 hop neighbors| ∩ |y's 1 hop neighbors| = {f}
|x's 1 hop neighbors| ∩ |y's 2 hop neighbors| = {a}

(b) All possible neighbor lists for $G_2\langle V, E\rangle$

Figure 5.4: Finding and combining all possible list of neighbors

(1). At the client, we find all values of N_n. We find $\sum_{n=1}^{l-1} N_n$. Client keeps the list of the nodes for each N_n.

(2). At the server too, we find all values of N_n. Server also keeps the list of the nodes for each N_n.

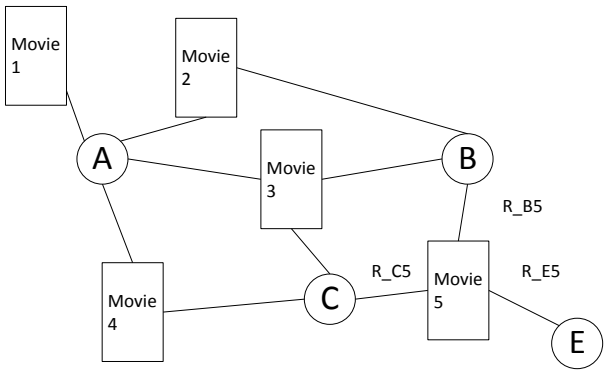
(3). We should also find the values for N_n for x from G_1 and y is from G_2 and vice versa.

(4). We can't directly add the two results together, since we should first eliminate the duplicate results. Client encrypts the lists. Client sends the encrypted lists to the server.

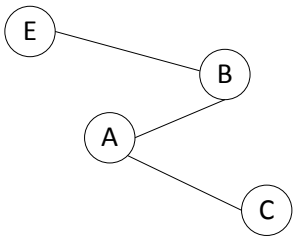
(5). Server makes a privacy-preserving integer comparison with its own lists and obtains the updated value of l-length neighbors which does not contain the duplicate values. In our example, we will have 4 in total, since the values we obtained at the server side are for duplicate paths.

5.3 Netflix and Facebook Case

In Netflix, we have the users as nodes and the edges between movies and users, as ratings. We can assume that the users have the same IDs in both networks, as they can be identified from their email addresses or phone numbers in both networks. Let's assume we want to recommend a movie to the user A. In order to make a good recommendation, we may utilize the information of what his friends on Facebook watch and how they rate them. However, we also do not want to reveal the friendship graph of Facebook to the Netflix network. Hence, we encrypt both the user IDs and the ratings corresponding to the users in the Netflix network (Figure 5.5a) and send the list to the Facebook (Figure 5.5b). We assume that the information that the person who will receive the recommendation is known by Facebook and it is user A in this case. However, we aim to hide A's friend from Netflix and hide his likes from Facebook. On Facebook graph, we can make a privacy-preserving integer comparison between the user IDs. If there is a match, we add the corresponding encrypted ratings for the movies rated by those users. We send the encrypted total amount back and in the Netflix network we decrypt the value to decide whether the movie is worth recommending to the user A. However, we need to prevent the case, where Netflix graph figures out which friends that a user has in Facebook. This can be understood when the user has only one friend. When Facebook sends the total rating back to Netflix , it will decrypt and see that the rating belongs to a certain user in the graph. Hence, we need to utilize differential privacy and add a certain amount of noise to the total rating, that is calculated at the Facebook side. Common neighbors, Jaccard's coefficient, Adamic/Adar and $Katz_\beta$ can also be applied to this use case as different similarity metrics.



(a) Netflix Graph



(b) Facebook Graph

Figure 5.5: Netflix and Facebook graph structures

5.4 Evaluation

We present the evaluation results in addition to the number of comparisons and complexity of the different similarity metrics. First, we define the initialization of the parameters of the encryption schemes that we use, namely Paillier and DGK. The size of the security parameter n in Paillier cryptosystem is 4096 bits. The security parameters of the DGK cryptosystem are set to the following values: L = 16, t = 160, k = 1024.

5.4.1 Comparisons and Complexity

In this section, we determine the number of comparisons that are needed to calculate the different techniques and accordingly, we define the complexity for each metric. According to [45], on average a person can have 214 friends in Facebook and the average distance is 4.7 for Facebook users. We can set number of friends as n = 214 and the longest distance between two nodes as m = 5. Now, we can define the number of comparisons needed for each metric:

5.4.1.1 Common Neighbors

For G_1, x and y can have at most n number of friends. For each element in x's neighbor list, we need to make comparison against all elements in y's neighbor list. In total, we will have n^2 comparisons for G_1. We have the same situation for G_2 too. In total, we will have n^2 comparisons for G_2 too. We also make another comparison between the neighbor lists of x and y in both graphs to eliminate the duplicate ones. It will also be n^2 comparisons. In total, we will have $3n^2$ comparisons. This algorithm runs in $O(n^2)$.

5.4.1.2 Jaccard's Coefficient

Since, we find $|\Gamma(x) \cap \Gamma(y)|$ first, we will have the same number of number of comparisons with the previous part which is $3n^2$. For $|\Gamma(x) \cup \Gamma(y)|$, we will again have $3n^2$ comparisons. In total, we will have $6n^2$ comparisons. This algorithm runs in $O(n^2)$.

5.4.1.3 Adamic/Adar

Since, we find $|\Gamma(x) \cap \Gamma(y)|$ first, we will have $3n^2$ comparisons. Then for each z in the set $\Gamma(x) \cap \Gamma(y)$, we will find the total number of neighbors in G_1 and G_2. We need to compare the neighbor set of z in G_1 with the neighbor set of z in G_2.

It consists of n^2 comparisons. Hence, in total we will have $4n^2$ comparisons. This algorithm runs in $O(n^2)$.

5.4.1.4 $Katz_\beta$

l = 1: 0 since we assumed that x and y are not neighbors. l = 2: n^2 l = 3: $2n^3$ l = 4: $3n^4$

Let C be:

$$\mathrm{C} = \sum_{l=1}^{m} (l-1)n^l \tag{5.9}$$

Value of C is for only one graph, so for two graphs we will have $2C$ comparisons. We should add the number of comparisons that are made to eliminate the duplicate neighbors to $2C$.

This algorithm runs in $O(n^m)$.

5.4.2 Performance

We perform our test on Windows 8.1 platform with Intel(R) Core(TM) i7-4510U CPU @2.00GHz processor and 12.0 GB RAM. Both of the graphs are generated in MATLAB as an adjacency matrix. Each graph has 1000 nodes. Later, this matrix is converted to the list of the IDs of all nodes in the network and their corresponding neighbors. We determined intervals for the possible number of neighbors of a node. For instance, 5-10 means that every node has a random number of neighbors between 5 and 10. We implemented the metrics in Java and the relevant codes can be found in github[1]. In Figure 5.6, the performance of common neighbor's metric is shown and in Figure 5.7 the performance of Jaccard's coefficient is shown for the same intervals for the degree of each node.

[1]https://github.com/ddm3/linkprediction

In can be deducted from the results that computing Jaccard's coefficient takes more time than computing common neighbors, as in addition to the privacy-preserving calculation of intersection in common neighbors, we also have to perform privacy-preserving calculation for union.

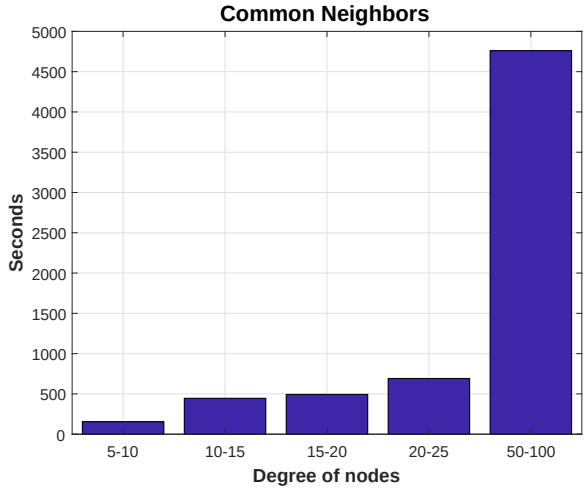


Figure 5.6: Performance of common neighbors

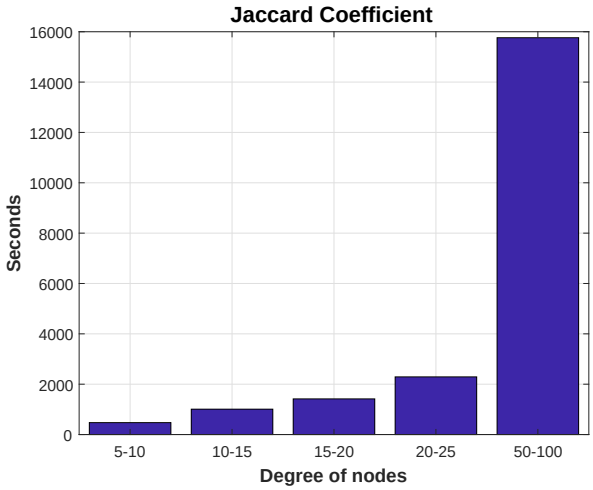


Figure 5.7: Performance of Jaccard's coefficient

5.5 Future Work

In our scheme, server learns the nodes on which the client performs link prediction. In order to prevent this, a scheme that allows the server to ask the client to

compute common neighbors of x and y without letting it know which nodes it wants to find common neighbors can be proposed.

Chapter 6

Conclusion

In this thesis, we proposed two systems that can perform data sharing between entities without violating privacy. We define a system under malicious setting that is based on verifiable computation and a privacy-preserving link prediction scheme under semi-honest setting.

The system that is based on verifiable computation aims to help the cloud computing services to analyze personal and sensitive data without violating the privacy of the involving parties, as such data like patient records, banking or location information, can reveal sensitive information about the individuals; and their disclosure may end up in serious problems about privacy. The system performs privacy-preserving similarity check of patients across different hospitals. By utilizing secret sharing and signature schemes, we obtained a system that can preserve privacy in the malicious setting.

Privacy-preserving link prediction scheme addresses the fact that there is an increase in the analysis done on social network graphs and link prediction is one of the prominent areas. Moreover, the privacy concerns about performing link prediction in two different graphs should be addressed. Therefore, we proposed a privacy-preserving link prediction scheme. Given two social network graphs, we computed different similarity metrics of two nodes. For this computation,

we used privacy-preserving integer comparison. Using this scheme, we created a protocol that finds the neighbor lists in both graphs and combines them without violating privacy.

Bibliography

[1] X. Wu, X. Ying, K. Liu, and L. Chen, "A survey of privacy-preservation of graphs and social networks," in *Managing and mining graph data*, pp. 421–453, Springer, 2010.

[2] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *Journal of Privacy and Confidentiality*, vol. 1, no. 1, p. 5, 2009.

[3] R. Rivest, L. Adleman, and M. Dertouzos, "Foundations of secure computation," 1978.

[4] R. Gennaro and D. Wichs, "Fully homomorphic message authenticators," in *Advances in Cryptology-ASIACRYPT 2013*, pp. 301–320, Springer, 2013.

[5] K.-M. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Advances in Cryptology–CRYPTO 2010*, pp. 483–501, Springer, 2010.

[6] E. De Cristofaro, Y. Lu, and G. Tsudik, "Efficient techniques for privacy-preserving sharing of sensitive information," in *Trust and Trustworthy Computing*, pp. 239–253, Springer, 2011.

[7] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology-EUROCRYPT 2004*, pp. 1–19, Springer, 2004.

[8] C. Hazay and T. Toft, "Computationally secure pattern matching in the presence of malicious adversaries," *Journal of cryptology*, vol. 27, no. 2, pp. 358–395, 2014.

[9] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 844–855, ACM, 2014.

[10] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology–CRYPTO 2010*, pp. 465–482, Springer, 2010.

[11] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Advances in Cryptology–CRYPTO 2011*, pp. 111–131, Springer, 2011.

[12] M. Backes, D. Fiore, and R. M. Reischuk, "Verifiable delegation of computation on outsourced data," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 863–874, ACM, 2013.

[13] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 238–252, IEEE, 2013.

[14] B. Schoenmakers, M. Veeningen, and N. de Vreede, "Trinocchio: Privacy-friendly outsourcing by distributed verifiable computation.," *IACR Cryptology ePrint Archive*, vol. 2015, p. 480, 2015.

[15] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," in *Security and Privacy (SP), 2015 IEEE Symposium on*, pp. 253–270, IEEE, 2015.

[16] C. Fournet, M. Kohlweiss, G. Danezis, Z. Luo, *et al.*, "Zql: A compiler for privacy-preserving data processing.," in *USENIX Security*, pp. 163–178, Citeseer, 2013.

[17] J. Baron, K. El Defrawy, K. Minkovich, R. Ostrovsky, and E. Tressler, "5pm: Secure pattern matching," in *Security and Cryptography for Networks*, pp. 222–240, Springer, 2012.

[18] S. D. Gordon, J. Katz, F.-H. Liu, E. Shi, and H.-S. Zhou, "Multi-client verifiable computation with stronger security guarantees," in *Theory of Cryptography*, pp. 144–168, Springer, 2015.

[19] E. A. Leicht, P. Holme, and M. E. Newman, "Vertex similarity in networks," *Physical Review E*, vol. 73, no. 2, p. 026120, 2006.

[20] W. Liu and L. Lü, "Link prediction based on local random walk," *EPL (Europhysics Letters)*, vol. 89, no. 5, p. 58007, 2010.

[21] K. Yu and W. Chu, "Gaussian process models for link analysis and transfer learning," in *Advances in Neural Information Processing Systems*, pp. 1657–1664, 2008.

[22] J. Zhang, P. S. Yu, and Z.-H. Zhou, "Meta-path based multi-network collective link prediction," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1286–1295, ACM, 2014.

[23] Y. Dong, J. Tang, S. Wu, J. Tian, N. V. Chawla, J. Rao, and H. Cao, "Link prediction and recommendation across heterogeneous social networks," in *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pp. 181–190, IEEE, 2012.

[24] Y. Dong, J. Zhang, J. Tang, N. V. Chawla, and B. Wang, "Coupledlp: Link prediction in coupled networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 199–208, ACM, 2015.

[25] J. Tang, T. Lou, J. Kleinberg, and S. Wu, "Transfer learning to infer social ties across heterogeneous networks," *ACM Transactions on Information Systems (TOIS)*, vol. 34, no. 2, p. 7, 2016.

[26] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Annual Cryptology Conference*, pp. 465–482, Springer, 2010.

[27] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[28] C. Gentry, *A fully homomorphic encryption scheme*. Stanford University, 2009.

[29] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption.," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1192, 2015.

[30] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in cryptologyEUROCRYPT99*, pp. 223–238, Springer, 1999.

[31] I. Damgard, M. Geisler, and M. Kroigard, "Homomorphic encryption and secure comparison," *International Journal of Applied Cryptography*, vol. 1, no. 1, pp. 22–31, 2008.

[32] A. Essex, J. Clark, and U. Hengartner, "Cobra: Toward concurrent ballot authorization for internet voting.," in *EVT/WOTE*, 2012.

[33] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy (SP)*, pp. 459–474, IEEE, 2014.

[34] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@ home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[35] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande, "Folding@ home and genome@ home: Using distributed computing to tackle previously intractable problems in computational biology," *arXiv preprint arXiv:0901.0866*, 2009.

[36] G. Woltman, S. Kurowski, *et al.*, "The great internet mersenne prime search," *Online,(1997, March 23) available http://www. mersenne. org*, 2004.

[37] D. Molnar, "The seti@ home problem," *ACM Crossroads*, vol. 7, p. 55, 2000.

[38] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[39] H. Kashima and N. Abe, "A parameterized probabilistic model of network evolution for supervised link prediction," in *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pp. 340–349, IEEE, 2006.

[40] C. Wang, V. Satuluri, and S. Parthasarathy, "Local probabilistic models for link prediction," in *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pp. 322–331, IEEE, 2007.

[41] J.-P. Hubaux, J. Fellay, E. Ayday, M. Laren, J. Raisaro, P. Jack, *et al.*, "Privacy-preserving computation of disease risk by using genomic, clinical, and environmental data," in *Proceedings of USENIX Security Workshop on Health Information Technologies (HealthTech 13), number EPFL-CONF-187118*, 2013.

[42] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology-EUROCRYPT 2004*, pp. 1–19, Springer, 2004.

[43] L. Kissner and D. Song, "Privacy-preserving set operations," in *Advances in Cryptology–CRYPTO 2005*, pp. 241–257, Springer, 2005.

[44] A. Evfimievski and T. Grandison, "Privacy preserving data mining," *IGI Global*, pp. 1–8, 2009.

[45] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The anatomy of the facebook social graph," *arXiv preprint arXiv:1111.4503*, 2011.