DISSERTATION

PRIVACY PRESERVING LINKAGE AND SHARING OF SENSITIVE DATA

Submitted by

Ibrahim Meftah Lazrig

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2018

Doctoral Committee:

Advisor: Indrakshi Ray
Co-advisor: Indrajit Ray

Yashwant Malaiya
Leo Vijayasarathy
Toan Ong

Copyright by Ibrahim Meftah Lazrig 2018
All Rights Reserved

ABSTRACT

PRIVACY PRESERVING LINKAGE AND SHARING OF SENSITIVE DATA

Sensitive data, such as personal and business information, is collected by many service providers nowadays. This data is considered as a rich source of information for research purposes that could benefit individuals, researchers and service providers. However, because of the sensitivity of such data, privacy concerns, legislations, and conflict of interests, data holders are reluctant to share their data with others. Data holders typically filter out or obliterate privacy related sensitive information from their data before sharing it, which limits the utility of this data and affects the accuracy of research. Such practice will protect individuals' privacy; however it prevents researchers from linking records belonging to the same individual across different sources. This is commonly referred to as record linkage problem by the healthcare industry.

In this dissertation, our main focus is on designing and implementing efficient privacy preserving methods that will encourage sensitive information sources to share their data with researchers without compromising the privacy of the clients or affecting the quality of the research data. The proposed solution should be scalable and efficient for real-world deployments and provide good privacy assurance. While this problem has been investigated before, most of the proposed solutions were either considered as partial solutions, not accurate, or impractical, and therefore subject to further improvements. We have identified several issues and limitations in the state of the art solutions and provided a number of contributions that improve upon existing solutions. Our first contribution is the design of privacy preserving record linkage protocol using semi-trusted third party. The protocol allows a set of data publishers (data holders) who compete with each other, to share sensitive information with subscribers (researchers) while preserving the privacy of their clients and without sharing encryption keys. Our second contribution is the design and implementation of a probabilistic

privacy preserving record linkage protocol, that accommodates discrepancies and errors in the data such as typos. This work builds upon the previous work by linking the records that are similar, where the similarity range is formally defined. Our third contribution is a protocol that performs information integration and sharing without third party services. We use garbled circuits secure computation to design and build a system to perform the record linkages between two parties without sharing their data. Our design uses Bloom filters as inputs to the garbled circuits and performs a probabilistic record linkage using the Dice coefficient similarity measure. As garbled circuits are known for their expensive computations, we propose new approaches that reduce the computation overhead needed, to achieve a given level of privacy. We built a scalable record linkage system using garbled circuits, that could be deployed in a distributed computation environment like the cloud, and evaluated its security and performance. One of the performance issues for linking large datasets is the amount of secure computation to compare every pair of records across the linked datasets to find all possible record matches. To reduce the amount of computations a method, known as blocking, is used to filter out as much as possible of the record pairs that will not match, and limit the comparison to a subset of the record pairs (called candidate pairs) that possibly match. Most of the current blocking methods either require the parties to share blocking keys (called blocks identifiers), extracted from the domain of some record attributes (termed blocking variables), or share reference data points to group their records around these points using some similarity measures. Though these methods reduce the computation substantially, they leak too much information about the records within each block. Toward this end, we proposed a novel privacy preserving approximate blocking scheme that allows parties to generate the list of candidate pairs with high accuracy, while protecting the privacy of the records in each block. Our scheme is configurable such that the level of performance and accuracy could be achieved according to the required level of privacy. We analyzed the accuracy and privacy of our scheme, implemented a prototype of the scheme, and experimentally evaluated its accuracy and performance against different levels of privacy.

ACKNOWLEDGEMENTS

Praise is to God by whose grace good deeds are completed. After praising and thanking God, I would like to thank my parents for all their love and encouragement, for raising me to be a good man, and for their support in all my pursuits. I must also express my very profound gratitude to my wife and my kids for providing me with unfailing support and continuous encouragement throughout my years of study . This accomplishment would not have been possible without their support.

I also would like to express my sincere gratitude to my advisors, Prof. Indrakshi Ray, and Prof. Indrajit Ray, for their continuous support of my Ph.D study and research, for their patience, motivation, kindness, and immense knowledge. Their guidance helped me in all the time of research and writing of this dissertation. I could not have imagined having better advisors for my Ph.D study. Their advice and support on both research as well as on my career have been invaluable.

I am also grateful to the members of my doctoral committee : Prof. Yashwant Malaiya, and Prof. Leo Vijayasarathy, for their reviews, insightful comments, patience, kindness, and encouragement.

My sincere thanks also goes to Prof. Michael Kahn, and Dr. Toan Ong, from University of Colorado- Denver Medical campus, who provided me an opportunity to join their team as intern, and who gave access to their resources of research. Without their precious support it would not be possible to conduct this research. I appreciate all their contributions of time, ideas, and funding to make my Ph.D. experience productive and stimulating.

Last but not least, my time at CSU was enriched by those with whom I have had the pleasure to hang-out or work with, our security research group (DBSec). So I would like to thank my fellow students for their feedback, cooperation and of course friendship, specially my collaborators Tarik Moataz, and Diptendu Kar.

DEDICATION

To my parents, Noorya and Meftah

To my wife, Hala

To my daughters, Noor, Lojain, Fatima, and Maram

To my sons, Hadi and Meftah

To my siblings, Mokhtar, Lutfiya, Abdelwahid, Younis, Hanan, Hawa, Khadeeja, and Mohammed

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

With the advancement of technology, we are shifting into an era where services are offered electronically through interconnected providers. Such services often collect sensitive information about individuals.

These technology advancements will provide environments that help organizations to fulfill their application requirements like availability, scalability, and cost effectiveness. In addition, these environments could boost the collaboration between individuals/organizations seeking to share some information. However, despite the huge information sharing benefits, this shift is still at its slowest pace because of privacy concerns.

Privacy could be breached in different ways, and achieving privacy is not limited to anonymity or hiding person's online activities. Private information about individuals are stored in digital forms around multiple locations, each of which has a certain degree of sensitivity. For example, medical health records are more sensitive than personal hobbies in social networks profile. The disclosure of private information is irreversible action and has a great effect on person's emotional, financial and legal status, so people always seek to protect their information.

On the other hand, data linkage and sharing across multiple parties, who have access to individuals' data, has many useful usages, starting from finding mutual interests in social networks, scientific research, and up to national security. Privacy issues that might occur because of such sharing hinder data collectors (owners) from participation in some useful information sharing applications, or they might participate by using obliterated data that is not very useful. Thus, the question becomes how we build protocols that enable such useful integration and sharing of data while preserving the individuals' privacy. Trial and error strategy, where a sharing protocol is used and then later replaced if it didn't provide enough privacy, is not an option. If private information is leaked, it is not recoverable after the fact.

For this reason, data holders and service providers should provide security mechanisms that ensure the protection of the consumers' sensitive information from inadvertent disclosure or unauthorized access.

Secure channels with access control are not sufficient when the data is outsourced or shared with other parties. Therefore, new efficient privacy preserving data linkage and sharing protocols are needed, and it is currently considered as one of major research challenges to design such protocols.

Researchers realized the need for privacy preserving and security mechanisms to safely utilize the new robust, highly available, and low cost technology like the cloud services to store and share information. As a result, various approaches have been proposed to protect the privacy of shared and outsourced information based on cryptographic and non-cryptographic primitives. Some of the non-cryptographic approaches tend to remove, generalize and anonymize some information or add noise to data in order to protect privacy. Though those techniques achieve some levels of privacy, they limit the research applications of this data. The cryptographic approaches rely on cryptographic primitives that require shared keys, or need very computation intensive operations, which limit their scalability.

In general, the state of the art cryptography-based privacy preserving data sharing approaches are *scenario-tailored* or *general-purpose*. On one hand, *general-purpose* approaches define protocols to privately and securely compute arbitrary function on the data to be shared, such as intersection and summation. The participants of the sharing protocol, each having his own private inputs, are interested in computing a function f over their inputs. Each party will learn the result of evaluating this function f on the inputs of all the participants and nothing else. This may be achieved using a trusted party, where each party could send its private inputs to this trusted party, and he will evaluate the result of the function f and send it back to them. However, it is hard to find a party that could be trusted by all parties. Consequently, researchers have worked on protocols that can evaluate the function f without the trusted third party. The general purpose protocols are called *Secure Multi-party Computation, (SMC)* [34] and *Oblivious Function Evaluation, (OFE)* [41] [37], and the main

drawback of these protocols is lack of scalability due to the computation and communication overhead incurred by the cryptographic operations involved.

On the other hand, *scenario-tailored* approaches are specialized protocols to perform specific functionalities rather than generic ones. For example *Querying Encrypted Database* allow parties to share information with others by querying, searching, or even computing some functions over their encrypted data. These protocols could be adapted and deployed on existing database management systems (DBMS) and do not incur too much overhead, which made them more popular and practical, for example searchable encryption constructions [17]. However, a new design of specific protocol for each scenario is needed whenever the application or the requirements change.

In the first part of this dissertation we will focus on both scenario-tailored and general-purpose cryptography-based approaches, because of the advantages specific to each of them. We will discuss how we build and design privacy preserving linkage and sharing of sensitive data (PPLSSD) systems based on both approaches, and how we alleviate their drawbacks for certain scenarios.

Specifically, we investigate how multiple and competing data providers can link and share their data in a privacy preserving manner. This falls in the category of privacy preserving record linkage (PPRL). Record linkage is the task of identifying which records from one or more data providers refer to the same entity. Each entity has some private attributes that may directly or indirectly identify it. Examples of such attributes include those in personally identifiable information (PII), IP address, and GPS location. These identifying attributes are usually used to link the records. Note that, sometimes a data provider cannot share an entity's identifying attributes because of policies and regulations such as Health Insurance Portability and Accountability Act (HIPAA) in the healthcare sector.

The use of cryptographic tools to secure the private attributes appears to be a viable approach. However, there are many constraints that prevent us from adopting simple cryptographic solutions. First is that the different data providers are autonomous and often competing entities. Thus, we cannot expect them to use a shared key that is common to all

data providers for data encryption. Second is that if the data providers use different keys for encryption, then it may be impossible to link the data belonging to the same entity but coming from different providers. Third is that the data maintained by different providers come from different sources and they may contain errors, typos, and missing values. Consequently, any direct transformation and encryption of the linkage attributes will make the record linkage problem harder, even if the same keys are used in encryption. Another challenge we investigate is how parties could link their data without the need of trusted third party. In this case parties do not want their data to leave their institution even in encrypted form, and yet be able to link their data. We designed some protocols to address some of these issues of privacy preserving record linkage.

In the second part of this dissertation, we investigate privacy preserving record linkage scalability issues. We study how to improve the record linkage process using indexing methods known as *blocking*. Blocking is the process of grouping the records of each party involved in the linkage such that only records in the corresponding blocks will be considered and tested for matching. Since traditional blocking schemes leak too much information about the records in each block, we designed a novel blocking method that reduce the amount of leaked information. That is, we designed a blocking method that achieves good level of privacy and reduces the number of records needed to be compared for record linkage.

1.1 Motivation Applications

As we mentioned before, data linkage and sharing across multiple parties has tremendous benefits for individuals, research community, and the data holders themselves. The development of privacy preserving data linkage and sharing protocols will encourage data holders to share more valuable information, based on privacy guarantee levels provided by these protocols. The following are some examples that motivate the need for privacy preserving data linkage and sharing schemes in different scenarios.

1. *Healthcare*: My research has been motivated by a real-world scenario in which multiple healthcare providers need to share clinical data with researchers. The researchers may

be studying the effects of some experimental drugs on a patient. In such scenarios, the researchers need clinical data from all the healthcare providers who are treating the patient. Clinical data contains personally identifiable information (PII) which is extremely sensitive as it may be used to uniquely identify an individual. Thus, it is important to link the data from the multiple healthcare providers and share it with the researchers in a privacy preserving manner such that PII or other sensitive information is not revealed. The analysis performed by the researcher may contain useful information that must be propagated back to the patients.

One traditional approach for solving this problem is using data obliteration where each provider removes PII information before handing the data to the researcher. This prevents linking the data belonging to the same patient and so the researcher does not get the complete medical history of the patients. The alternative approach involves giving all the sensitive data to a trusted third party who links data belonging to the same patient, sanitizes it, and then gives it to the researchers. An association is maintained between the sanitized data and the patient identifier which is used to convey the research results to the patient. The problem with this alternative approach is that if the trusted third party is compromised then all the patients' sensitive data will be leaked. In this dissertation we provide an improved solution to this problem.

2. *Law Enforcement*: Suppose there is a list of suspects, and law enforcement agency (LEA) would like to investigate their bank accounts at certain bank. The LEA is not allowed to disclose the list of suspects to the bank, and the bank on the other hand cannot disclose all of its customers' accounts information and trust the LEA to extract only the relevant information. Furthermore, the bank might ask for an authorized list by mutually trusted party, so the LEA can obtain information of pre-authorized lists only.

3. *Interest Sharing and Private Matching*: Sometimes two or more users would like to share their common interests, hobbies, activities, availability, locations and so on, without exposing their privacy beyond the matched common ones.

4. *Network DOS Attack Detection*: In this application, companies targeted by DOS attacks might want to collaborate to distinguish between malicious and legitimate source IP addresses during the attack. Usually the source of attacking hosts are common between different companies' logged data, and finding them will be easier if the logs of these companies are analysed together. However, companies refrain from sharing their logs because of privacy issues related to the companies' reputation and their customers. If there exists a privacy preserving protocol that reveals only the suspicious source IP addresses and nothing else, then the companies will voluntarily participate in such secure information sharing.

1.2 Privacy and performance Requirements

For mutual information sharing between parties or publishers/subscribers sharing scenarios, where all data owners need to share their data with certain levels of privacy guarantee and sharing on a need-to-know basis (minimum sharing) we define the following list of requirements, that privacy preserving information sharing protocols strive to fulfill.

- *Anonymity*: the identities of subjects (whom the data are about) cannot be revealed. Moreover, unauthorized entities should not be able to verify if any subject identity exists in the data or not.

- *Owner Privacy*: none of the parties should learn anything about information stored in the other parties' database beyond the results of the sharing process, or what could be inferred from these results. Sometimes a stronger privacy requirement is needed for the publisher/subscriber scenario, where the identity of the owner (source of information) should not be revealed.

- *Querier/Client Privacy*: the data source (owner) should learn nothing about the client's query (what the client is looking for).
- *Correctness*: the parties receive correct results at the end of the sharing protocol.
- *Authorization*: the shared information is authorized by some trusted third party (TTP), so none of the participating parties could use fake data to infer information about other party's data (No-Cheating).
- *Fairness*: for mutual sharing, either all parties get the result or none of them do.
- *Efficiency*: protocol should scale (both in terms of computation and communication complexity) to large number of participants each having large data sets.
- *Flexibility*: the protocol is preferred to support multiple functions, rather than single function (generality vs. specificity).

Sometimes the parties participating in the sharing protocol specify more requirements about the environment used during the sharing process. For example, no third party shall be involved in the process, and no data, in any form (even encrypted), is directly shared with the other parties. However, this might effect the performance and scalability of the protocol. Researchers stress that it is very important to design protocols that are efficient in practice and meet the given privacy requirements. Sometimes, it might be acceptable (depending on the application and the assumptions about the participating parties) to relax the privacy requirements and use less strict (weak) definition of security models, instead of designing protocols using strong security models that fulfill all security and privacy requirements and are inefficient in practice. In addition some information may be learned during the data linkage and sharing process, so the question of what could be considered as acceptable leakage must be asked before the sharing process is started, in order to select the proper security model.

1.3 Publish/Subscribe Data Linkage and Sharing via Third Party

Many applications exist where a group of data sources (publishers) continuously generate sensitive data, periodically update the same, and share the data with another group of data analyzers (subscribers). To protect the privacy of the clients of the publishers, the data sharing needs to occur in a privacy-preserving manner, which in its simplest form is enabled by removing identifying information from the data. An example of such data sharing is observed in the so-called clinical data-sharing networks. Different health care providers (e.g., medical clinics, laboratories, hospitals and pharmacies) are the publishers of the data for the networks while clinical researchers are the subscribers of the data. Unlike the traditional privacy-preserving data publishing domain, the data in such clinical data-sharing networks are not static but are updated every time a patient interacts with a data publisher.

Owing to the updatable nature of the data, a unique and challenging situation occurs in such applications that is not observed in traditional privacy preserved data publishing setups. Any updates to a record on the publisher side must be pushed to the corresponding record on the subscriber side even though these two records have been delinked via sanitization algorithms. Consider the medical data warehouse example. Assume that a clinical researcher needs data related to a specific demographic. In this case, patients' identification information (such as SSN, driver's license number, date of birth etc.) are typically removed when the medical information is shared with the researcher. To provide the most relevant and current data, patients' progress under treatment regimens would need to be propagated to the clinical researcher. Similarly, the researcher should also be able to pull updates from the publisher or at a minimum be able to query the publisher for updates. To allow such sharing of information, records at the publisher need to be somehow linked to records at the subscriber in a privacy preserving manner.

Things become more complicated if this sharing needs to be carried out between multiple publishers and multiple subscribers. Publishers are often business competitors and unwilling

to reveal to each other that they might be sharing clients between themselves. In such cases, two publishers should not know that they have a common group of clients. (Sharing such information under explicit directives from a client is allowed and is not considered here.) For privacy reasons, two subscribers should not be able to determine that they have clients in common; they should not be able to link or trace two sanitized records to the same client. When a publisher has more than one record for the same client, the same number of sanitized records should be available at the subscriber and be updated as needed. This occurs, for example, when a patient has repeated visits to the doctor for treatment.

Not much work has been done in this area of privacy preserving record linkage in dynamic setting. Some existing techniques that partially address the problem require encryption of linkage information using a shared key between data publishers to find if matched individuals' data exist across multiple sites. However, this technique works for small communities; it is expensive to deploy in large heterogeneous setups. In addition, shared keys among a large number of entities increases the chances of key leakage. An alternative technique that is used in the medical community is to utilize the services of a trusted third party, called the Honest Broker. The third party maintains the linking information between the subscriber data and the publisher data in a non deidentified manner. The problem with this approach is that the Honest Broker has all information, which makes it a lucrative target for attackers. If the Honest Broker is compromised it will cause a catastrophic damage to both data publishers as well as to individual clients.

1.4 Privacy Preserving Linkage of corrupted Data

If the data is encrypted with a shared (or same) key, the matching process could be performed deterministically. However, in a typical real-world setting, where the data publishers operate independently of each other, there is no guarantee that the same datum is consistent across different publishers. This occurs owing to factors such as, different structures (for example, FName versus First Name), semantics (whooping cough vs. pertussis), typographic and formatting errors (John vs. Jhon), or the way the data is captured, entered and main-

tained by each entity. As a result, deterministic matching becomes difficult or ineffective. If the data is encrypted with different keys (as would typically be done by independent data publishers that do not trust each other), finding matches between the records are even more challenging. For privacy preserving record matching to be effective under such circumstances, we need to consider as many of the discrepancies in the data as possible that can potentially occur during acquisition (input) such as spelling mistakes, formatting errors, abbreviations, punctuations, pronunciations, users background, or different underlying assumptions about the structure of the data, and attempt to determine how similar the encrypted data are.

1.5 Privacy Preserving Data Linkage without Third Party

Multiple PPRL methods have been proposed (see chapter 3 for a comprehensive discussion). Most work follow the centralized linkage approach, which requires the participation of a trusted third party (TTP) or a semi-trusted third party (STP) [66]. However, the TTP or STP is susceptible to attacks [23] resulting in information leakage. Consequently, many data providers are unwilling to let sensitive information out of their institutions even when it is anonymized/encrypted with the best known techniques.

In such cases, parties are willing to share data based on the need-to-know concept (minimum sharing). We use an example from the medical domain to describe such linkage scenario. Two databases of patients records belonging to different health-care providers. The data contain personally identifiable information (PII) of the patients such as, first name, last name, date of birth, U.S. Social Security Number (SSN), and zip/postal code. Combinations of these PII are used for the linkage process. For research purposes both parties want to find if they have medical records that belong to the same patient without revealing the patient information. In order to protect the privacy of patients who are not members of both the databases, the two parties do not want to release their records, even encrypted, to each other or to a third party to perform the linkage process.

1.6 Blocking Overview

Data blocking problem is viewed as the partitioning of n records with d attributes (fields) into blocks (classes) such that, records in each block have similar values in corresponding attributes. A set of attributes is predetermined and selected as grouping criteria and they are referred to as blocking variables. Record similarity may be defined using similarity/distance measures on the values of their blocking variables [26]. Blocks are assigned identifiers (ID) that are created from the values of the blocking variables.

Blocking is used in databases to improve the performance of database operations that require checking one dataset against the other. For example to link the records of two or more datasets using some of their attributes, we need to match the similarity of each record of one dataset against all the records of the other dataset. If blocking is used, the matching process is limited to the records of the corresponding blocks, that is, blocks with the same ID.

In record linkage (RL), where two or more parties have their databases compared against each other to find records belonging to the same person [29], blocking plays a vital rule in performance. Parties use blocking to group their records based on agreed upon scheme, and then they use the block IDs to create a set of candidate record pairs for each block. Then the linkage process is restricted to those candidate pairs only, i.e quickly remove obvious non-matched records from linkage process, and hence reduce the computation. To measure the efficiency of blocking schemes when used to speed up the comparison of two datasets A and B, two well known measures are used. *Reduction Rate* (RR), which measures how effective the blocking scheme is in reducing the number of computations, and and *Pair Completeness* (PC), which measures how effective the blocking scheme is in not removing the actual matched record pairs [55].

Blocking schemes can be categorized as deterministic or probabilistic (approximate) blocking. Deterministic blocking is used to group the records based on the exact match of their selected blocking attribute(s) values. For example group the personal records based

on the hash values of their Surname attribute. Approximate blocking is used to group the records based on the similarity of their selected blocking attribute(s) values. For example using the Soundex values of the Surname attribute [16]. Since data is prone to errors, deterministic blocking is not very effective to capture all possible matched pairs, and record linkages using deterministic blocking will have too many false negatives (FN) (i.e. records that matched will be missed because of typographic errors).

For efficient record linkage, blocking is needed to be performed in the same manner for both the parties , so parties need to share the blocking information (e.g. Block IDs) with each other or with a trusted third party (TTP), and may leak some information. For example, blocking using 'lastname' will reveal the lastnames of all records in that block. To avoid this, secure or less leaking blocking techniques are required.

1.7 Research Objectives and Contributions

In this work, we have designed protocols for efficient privacy preserving linkage and sharing of sensitive data. We investigated the cases of publish/subscribe, and mutual sharing models. In the publish/subscribe, we used the service of a semi-trusted third party (STP), which is not trusted to keep a secret confidential, but follows the protocol and does not collude with any of the parties. We also investigated the linkage problem when the data has typos and errors, and how the standard secure data linkage methods fail to link the data when the values of data attributes used in the linkage contains such discrepancies and typos. We designed a protocol that securely and probabilistically link corrupted data based on the values similarity, and with high accuracy. In the mutual sharing case, we investigated two-party sharing scenario, where none of the parties wish to release its data even in an encrypted form, yet they want to find if they have some data in common. Towards this end, we studied the secure computation method and proposed solution using garbled circuit and Bloom filters for probabilistic data linkage. Since garbled circuits are not efficient by design, we proposed some design optimizations to make it possible to use it for this kind of applications. Finally, we studied the current indexing (blocking) methods used to to speed

up the linkage process, and we proposed and built a privacy preserving blocking scheme that allow parties to block their data with minimum privacy level guaranteed. The following sections briefly explain each of these contributions.

1.7.1 Privacy Preserving Record Matching using Automated Semi-Trusted Broker

Our first contribution is a protocol for record linkage using semi-trusted third party (semi-TTP). Our approach consists of a publish/subscribe architecture, where the publishers are the data sources, and subscribers are users who are interested in the linked records form these sources. In addition to the publishers and subscribers, we have a semi-TTP who correctly follows the protocol and does not collude with other parties. The data publishers do not need to share encryption keys but they need to cooperate at the beginning and only once to setup some key-converters using homomorphic encryption. Then each data source will encrypt the private attributes of each record with its private key, which guarantee that no other party can decrypt these attributes if he got access to them. The Semi-TTP will use the key-converters to re-encrypt the encrypted attributes, such that all parties' data will be encrypted under a key that does not exist (a ghost key). Then the semi-TTP will be able to link the records since their linkage attributes are encrypted under the same key, and forward the linked records to the subscribers. The subscribers can issue retrospective queries about any received records using the double encrypted attributes. Using the services of semi-TTP we achieved multiple goals like, hiding the source of data form the subscribers, hiding the linkage results from the sources, and protecting the privacy of the entities since no one can decrypt their private data. We implemented this protocol and tested its accuracy and performance [45, 57].

1.7.2 Privacy Preserving Probabilistic Record Linkage using Locality Sensitive Hashes

Our second contribution is a protocol to perform privacy preserving probabilistic record linkage to overcome the inconsistencies in the linkage attributes due to errors and typos. In

this protocol, we allow any pair of records to be linked if the similarity of their corresponding linkage attributes is above a predefined threshold. The next step is to create encrypted signatures from the linkage attributes and use these signatures to compute the similarity values of the original attributes in a privacy preserving manner using our previous protocol [58].

1.7.3 Privacy Preserving Probabilistic Record Linkage without Honest Brokers

We designed probabilistic privacy preserving record linkage protocol using Bloom filters and secure computation, and without a third party. We encoded the private attributes of each record in Bloom filter, then we designed garbled circuits (GC) to compute the Dice coefficient (DC) of every pair of records' Bloom filters. Based on the computed DC value and a pre-defined threshold, the GC will securely decide if the pair of records can be linked. We designed some heuristic to optimize and parallelize the GC computation to reduce the overhead. We built a proof of concept of this protocol and tested its performance in a local network, and we also deployed a distributed version of this protocol on Google Cloud (this work is in submission).

1.7.4 Privacy Preserving Approximate Blocking for Record Linkage

Blocking is the process of grouping the records of each party performing the linkage such that only records in the corresponding blocks will be considered and tested for matching. Since traditional blocking schemes leak too much information about the records in each block, we designed a novel blocking method that reduces the amount of leaked information. That is, we designed a blocking method that achieves good level of privacy and reduces the number of records that needed to be compared for record linkage (this work is in submission).

1.7.5 Scalable Distributed Privacy Preserving Probabilistic Record Linkage using Bloom Filters and Garbled Circuits

In order to build an efficient garbled circuit based record linkage system, we proposed some novel heuristic approaches to improve our GC designs, and build an efficient GC module to perform the DC computations in a reasonable time. We improved the scalability of this record linkage system by executing it in a distributed computation environment to perform the record linkage process in a parallel fashion. Our evaluation results show that the designed system is very effective and efficient (this work is in submission).

1.8 Dissertation Outline

The rest of this dissertation is organized as follows. In Chapter 2, we briefly give the background of some cryptographic primitives that we used in this research. In Chapter 3, we list some of the most relevant works, and their shortcomings that we addressed in this research. In Chapter 4, we introduce our first solution to the problem of record linkage without shared encryption key with the help of semi-trusted third party. In chapter 5, we addressed the problem of matching corrupted data, and propose our solution for the probabilistic data linkage. In Chapter 6, we present our work that does privacy preserving record linkage without the trusted third party. In Chapter 7, we introduce our privacy preserving record linkage scheme that uses approximate blocking, and how we can use it for scalable garbled circuits based record linkage that we introduced in chapter 8. We conclude and list some of potential future work in chapter 9.

Chapter 2

Preliminaries

In this section we introduce some of cryptographic primitives that we use to build some of the discussed schemes.

2.1 ElGamal Cryptosystem

ElGamal public-key cryptosystem [33] is defined over finite field $\mathbb{F}_q$ of a prime order q. The public key pk equals (G, q, g, h), where G is a cyclic group of order q with g as a generator, and $h = g^x$. The private key sk equals $x \xleftarrow{R} \{1, \cdots, q-1\}$, i.e., x is randomly sampled from numbers between 1 and $q-1$.

The encryption of a message m using the public key is (c_1, c_2) and computed such that $(c_1, c_2) = \mathsf{Enc}_{\mathsf{pk}}(m) = (g^r, m \cdot h^r)$, where $r \xleftarrow{R} \{0, \ldots, q-1\}$. To decrypt the ciphertext (c_1, c_2) and retrieve $m = \mathsf{Dec}_{\mathsf{sk}}(c_1, c_2)$, the private key is needed as follows: first compute $s = c_1^{\mathsf{sk}} = (g^r)^x$. The decrypted message m equals $m = c_2 \cdot s^{-1} = m \cdot h^r \cdot g^{-rx} = m \cdot (g^x)^r \cdot g^{-rx} = m$.

2.2 Elliptic Curve Cryptography and the Discrete Logarithm Problem (ECDLP)

An Elliptic Curve [54] E over a finite field $\mathbb{F}_q$ of prime order q, is a set of points with coordinates from that field defined by an equation of the form $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ for all $a_i \in \mathbb{F}_q$, or of the simpler form $y^2 = x^3 + ax + b$, with $a, b \in \mathbb{F}_q$ for finite fields of order different from 2 or 3. The coefficients define the shape of the curve and are the parameters of the curve. The set of points together with a special point called the point at infinity $\mathcal{O}$, form a group under the addition of points operation. Multiplication of points is not defined; however, multiplying a point P by a scalar u is defined as the addition of the point P u number times, i.e. $uP = \underbrace{P + P + \cdots + P}_{u \text{ times}}$. The cyclic subgroup $E(\mathbb{F}_q)$ is defined

by its generator (base point) P with order n, which is the smallest positive integer such that $nP = \mathcal{O}$. This subgroup $E(\mathbb{F}_q) = \{\mathcal{O}, P, 2P, \ldots, (n-1)P\}$ is denoted by its domain parameters (q, a, b, P, n).

Given an Elliptic Curve $E(\mathbb{F}_q)$ over a finite field and two points $P, Q \in E$, it is hard to find an integer $x \in \mathbb{Z}_q$ such that $Q = xP$. This is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). ECDLP is believed to be harder than finding Discrete Logarithms for finite fields, which is why many public key cryptosystems uses Elliptic Curves (EC) as the underlying group. For our purposes, the ElGamal cryptosystem and its variations are defined using EC as follows.

First, the communicating parties agree on the EC parameters and the corresponding field, i.e., $E(\mathbb{F}_q)$, the generator point G, and its order n. From these parameters each party generates its private key as $\mathsf{sk}_i = x, x \xleftarrow{R} \{1, \cdots, n-1\}$, and its public key as the point $\mathsf{pk}_i = xG$. The messages to be encrypted must be encoded as points on the curve in order to apply the group addition, or the messages must be integer scalars in the range $\{1, \cdots, n-1\}$ to use multiplications of points by scalars. The encryption of encoded message M is then performed by the sender A using the recipient B's public key sk_B as $C_1 = r \cdot G$, and $C_2 = r \cdot \mathsf{pk}_B + M$, where $r \xleftarrow{R} \{1, \cdots, n-1\}$. The decryption at the receiver side B is done using its private key sk_B as $M = C_2 - \mathsf{sk}_B \cdot C_1 = r \cdot \mathsf{pk}_B + M - sk_B \cdot r \cdot G = r \cdot k \cdot G - k \cdot r \cdot G + M = M$.

2.3 Homomorphic Encryption

Homomorphic encryption allows arithmetic operations to be carried out on ciphertext in such a way that the decrypted result matches the result of the operations when performed on the plaintext. *Partially* Homomorphic Encryption system (PHE) allows either addition or multiplication but not both. In this work, we focus only on the multiplicative property.

The homomorphic property of ElGamal cryptosystem over a finite field ensures that the product of two encrypted messages $\mathsf{Enc}_{\mathsf{pk}}(m_1)$ and $\mathsf{Enc}_{\mathsf{pk}}(m_2)$ will decrypt to the product of their corresponding plaintext messages $m_1 \cdot m_2$,

$$\begin{aligned}\mathsf{Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2) &= (g^{r_1}, m_1 \cdot h^{r_1}) \cdot (g^{r_2}, m_2 \cdot h^{r_2}) \\ &= (g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2}) \\ &= \mathsf{Enc}_{\mathsf{pk}}(m_1 \cdot m_2).\end{aligned}$$

If $s = c_1^{\mathsf{sk}} = g^{(r_1+r_2)x}$ then $s^{-1} = g^{-(r_1+r_2)x}$ and the decryption will result in

$$\begin{aligned}\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2)) &= \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m_1 \cdot m_2)) \\ &= \mathsf{Dec}_{\mathsf{sk}}(g^{r_1+r_2}, (m_1 \cdot m_2)h^{r_1+r_2}) \\ &= (m_1 \cdot m_2) \cdot g^{(r_1+r_2)x} \cdot s^{-1} \\ &= (m_1 \cdot m_2) \cdot g^{(r_1+r_2)x} \cdot g^{-(r_1+r_2)x} \\ &= m_1 \cdot m_2\end{aligned}$$

2.4 Threats and Privacy Issues in Sharing Encrypted Data

In this work, the threats to privacy we are concerned about are the ones that occur when the data is shared between multiple parties or when the data is outsourced to a third party. The threat comes from adversaries within the sharing system, i.e., from the participating parties. Outside adversaries are not considered here, since standard network security techniques can be used to mitigate their actions. Preserving privacy is not limited to the confidentiality of the data. Even when the data is encrypted, privacy may be violated. For example, when the data is outsourced to an untrusted cloud server, the operations on the data could reveal some information like the access pattern. Therefore the data is susceptible to unauthorized access and attacks. In order to describe or model threats to a designed privacy preserving information sharing protocol, an adversarial model depicting the trust assumptions associated with the environment and participants of the protocol is assumed. Then the protocol is proved to be secure in that adversarial model.

2.4.1 Security Model

The security model describes the adversarial actions that the participating parties are allowed to do during protocol execution. In general there are two types of adversaries, semi-honest, and malicious [35]. For the semi-honest adversarial model (also called "honest-but-curious" and "passive"), the corrupted parties correctly follow the protocol specification, however the adversary who controls those parties will attempt to use some of the corrupted parties internal states to learn information that should remain private. This model is considered weak form cryptographic perspective, however there are some realistic settings where it is sufficient to model the threats using this model.

In the malicious adversarial model (also called "active" adversaries), the adversary, who controls the corrupted parties, is allowed to make the parties arbitrarily deviate from the protocol. In order to ensure that no adversarial attack can succeed, it is preferred for any secure protocol to provide security in the malicious model. However it is very difficult to design efficient and practical protocols in the malicious model.

2.4.2 Privacy under the semi-honest model

The semi-honest model is widely used because of its efficiency and considered sufficient in many applications. It is more appropriate for settings where the participating parties have some mutual trust, like none of them will change its inputs in order to infer some information about the other's inputs. The semi-honest model ensures that no inadvertent leakage of information could happen as long as the parties are honest. In our motivation application example of healthcare providers who wish to perform record linkage on their patient records for research purposes, if one of the healthcare providers get compromised after the record linkage protocol executed, it is guaranteed that nothing is revealed about the other healthcare providers non-linked data. Following is the formal definition of privacy under this model adapted from Lindell et al. work [59].

Definition 1. *Privacy under the semi-honest model: Let f be a function. We say that a protocol π computes f in a privacy preserving manner in the the presence of static semi-*

honest adversaries if there exists a set of probabilistic polynomial-time algorithms $\{S_i\}$, $1 \leq i \leq t$ *such that for every input* $x_i \in \{0,1\}^c$ *in the set of inputs* $X = \{x_1, \cdots, x_t\}$, *and* $|x_i| = c$ *we have* $\{S_i(1^n, x_i, f(X)), f(X))\}_{n\in\mathcal{N}} \stackrel{c}{\equiv} \{(view_i^\pi(n, X), output^\pi(n, X))\}$, *where* $view_i^\pi(n, X) = (1^n, x_i, r_i, m_1^i, \cdots, m_t^i)$, *is the view of party* i, m_j^i *is the* j*th message party* i *received,* r_i *is its internal state, and* n *is security parameter. This means that, the view of a party* i *can be simulated by a probabilistic polynomial-time algorithm given access to the party's input* x_i *and the output only.*

2.5 Oblivious Transfer

Oblivious Transfer (OT) is one of basic primitives used in many security protocols. There are many OT primitives, represented by a number of selected messages k out of all messages n ($k/n - OT$), where $k < n$. In $k/n - OT$ protocol party B learns k of n secret messages held by party A, without A learning which secret messages B obtains. For example, in the most used, one-out-of-two Oblivious Transfer protocol ($1/2 - OT$), Alice has two secret bits (messages) b_0, b_1 unknown to Bob, and Bob can select to receive exactly one of the two bits. Alice will not know which bit was chosen, and Bob will not obtain information about the bit that he doesn't select. The probability that Alice can guess the bit selected by Bob is 1/2, in this case.

The $1/2 - OT$, is a secure function which inputs are A's bits denoted as a_0, a_1, and B's integer index of the bit of his choice $b \in \{0,1\}$ to select one of A's inputs. The function outputs for A, B is $(\perp, a_b)$, that is A gets nothing, and B gets the message corresponding to his selection.

A $1/k - OT$ can be implemented using one-way trapdoor function, and illustrated as follows:

Settings: Alice (the sender) has $b_1, \cdots, b_k$, Bob (the receiver) wants to receive $b_i, 1 \leq i \leq k$.

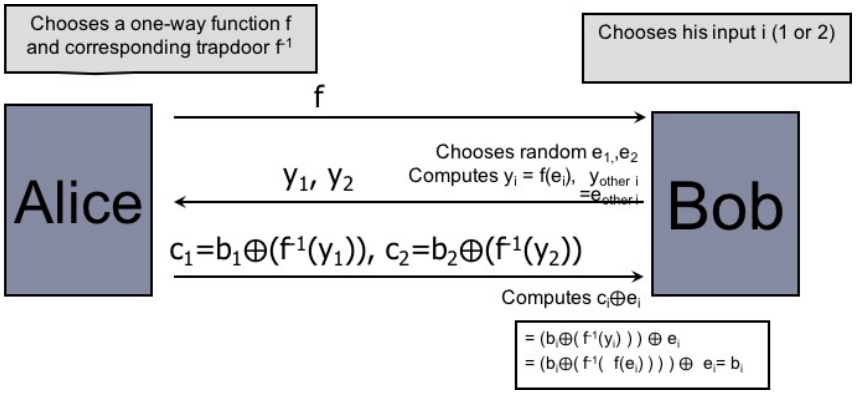


Figure 2.1: An example of 1 out of 2 OT protocol using one-way trapdoor function

1. Alice chooses trapdoor function f, and its trapdoor to compute f^{-1} , (e.g public key encryption scheme, such as RSA) and informs Bob about f. Only Alice knows how to compute f^{-1} efficiently.

2. Bob picks at random $e_1, \cdots, e_k$ and sends Alice the values $y_1 = e_1, \cdots, y_{i-1} = e_{i-1}, y_i = f(e_i), y_{i+1} = e_{i+1}, \cdots, y_k = e_k$

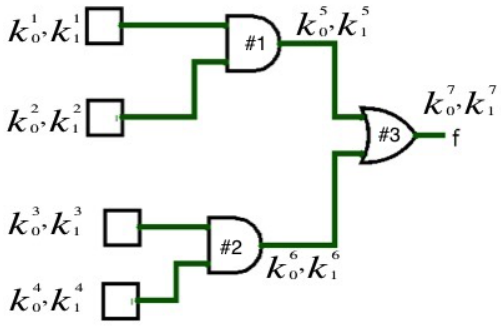
3. Alice computes and sends to Bob $x_j = (f^{-1}(y_j) \oplus b_j) \forall 1 \leq j \leq k$.

4. Bob computes $b_i = x_i \oplus e_i$.

Figure 2.1 shows an illustration of these steps for $k = 2$. Since $x_i \oplus e_i = ((f^{-1}(f(e_i)) \oplus b_i) \oplus e_i = (e_i \oplus b_i) \oplus e_i = b_i$ will be correct only if $y_i = f(e_i)$, then Bob will be able to get the correct value. However such scheme is not secure if Bob cheated and sent $y_i = f(e_i)$ *for all* $1 \leq i \leq k$, so he will be able to get all b_i values. We used this example OT protocol for its simplicity and illustration purposes, however for real-world applications, more secure OT protocols are used, and we refer to [6, 14, 63] for some of these secure OT protocols.

2.6 Yao's garbled circuit protocol

Yao's garbled circuit protocol allows parties A and B, who hold secret inputs x and y respectively, to jointly compute any function $f(x, y)$ without revealing their secret inputs x and y [90]. In Yao's protocol, the parties compute this function using a set of logic gates



(a) An example of a logic circuit for the function $F(x, y) = a_1 \cdot b_1 + a_2 \cdot b_2$, $x = a_1 a_2, y = b_1 b_2$

1st input	2nd input	Output	Garbled Output
k_0^1	k_0^2	k_0^5	$Enc_{k_0^1}(Enc_{k_0^2}(k_0^5))$
k_0^1	k_1^2	k_0^5	$Enc_{k_0^1}(Enc_{k_1^2}(k_0^5))$
k_1^1	k_0^2	k_0^5	$Enc_{k_1^1}(Enc_{k_0^2}(k_0^5))$
k_1^1	k_1^2	k_1^5	$Enc_{k_1^1}(Enc_{k_1^2}(k_1^5))$

(b) A garbled AND gate of the gate #1 in the circuit on the left

Figure 2.2: An Example of Yao's Garbled Circuit

with encrypted inputs, called a garbled circuit. Garbled circuits can be used to compute any function; however, they are computationally expensive operations. In Yao's protocol, the two parties engage in the computation process as a generator and an evaluator of the garbled circuit. Upon agreement between the two parties, the function is converted into a Boolean circuit with a series of logic gates. Then for each row in the truth table of every gate, the generator obfuscates the inputs by generating random keys to encode the input bits and double encrypts the corresponding output bit using these keys. Figure 2.2 shows an example of a logical circuit and one of its garbled "AND" gates.

The generator sends the evaluator the garbled circuit (as a set of garbled truth tables of the gates), after shuffling the rows to hide the usual order, along with the keys corresponding to the generator's inputs. The two parties then engage in an oblivious transfer process where the evaluator obtains the keys corresponding to his inputs and the generator gets nothing. Because the generator has the two possible keys k_0^i, k_1^i, corresponding to the possible bit values $\{0, 1\}$ of every garbled gate's input line i, the evaluator needs to get the key corresponding to his input bit value for line i without disclosing his bit value to the generator, i.e., for his input bit value $b \in \{0, 1\}$ of the input line i, he wants to obtain the corresponding key k_b^i without disclosing whether b is Zero or One, so the generator cannot guess the choice of the evaluator with probability more than 0.5. In addition, the generator wants to guarantee that the evaluator obtains only the key he chooses, and not the other key, because if the evaluator knows both keys, he can guess the generator's input from

the intermediate results when he uses these keys as his inputs. After obtaining the keys corresponding to both parties' inputs, the evaluator is able to evaluate the circuit and get the encrypted output of the circuit, for which he needs the help of the generator to decrypt.

For Yao's protocol we need an encryption scheme such that if the wrong key is used to decrypt the ciphertext, it will output nothing. One technique is padding the message with some predefined prefix/suffix, for example a string of n zeros as follows.

$$
\begin{aligned}
Gen(1^n) &: k \leftarrow \mathcal{U}_n \\
Enc(m) &: r \leftarrow \mathcal{U}_n. Output(c_1, c_2) \leftarrow (r, 0^n || m \oplus f_k(r)). \\
Dec(c_1, c_2) &: \texttt{Compute} \quad m_0 || m_1 \leftarrow c_2 \oplus f_k(c_1) \quad where \quad |m_0| = n. \\
&\qquad If\ m_0 = 0^n,\ then\ output\ m_1. Otherwise, output\ \perp .
\end{aligned}
$$

The evaluator will attempt to decrypt all the rows with the keys he has, and only one will succeed. The computation overhead incurred because of the oblivious transfer and gate evaluation. The size of B's inputs effects the overhead caused by the oblivious transfer stage, where A and B engaging in OT protocol for every input wire of the circuit that is associated with B's input. Also A is sending B tables of size linear in the size of the circuit, and B is decrypting a constant number of ciphertexts for every gate of the circuit (this is the cost incurred in evaluating the gates). The evaluation of the gates uses symmetric encryption which is very efficient compared to oblivious transfers that require modular exponentiations. So if the size of the evaluated circuit is small, the overhead of the OT will dominate. The computation overhead is therefore roughly linear in the length of B's input. The communication overhead is linear in the size of the circuit and number of OTs (i.e size of B's inputs).

Chapter 3

Related Work

Privacy preserving data sharing has been well studied, particularly in the context of sharing information from databases controlled by multiple parties. The participating parties in a privacy preserving database querying system are: the *data owner*, variously called the data source or publisher of the data, who provides access to its data to others, the *data querier* who generates query against the publisher's data and receives the results, and the *host* who (potentially) stores the publisher's data, and executes the query by performing relevant computations.

If the querier is the owner of the data itself, and the host is an untrusted outsourced third party, searchable encryption schemes or Oblivious RAM (ORAM) are used to protect the data and maintain the owner privacy. Considerable amount of research has been done towards this problem that resulted in very efficient and expressive schemes [11, 12, 17, 62, 78, 79]. However, in searchable encryption schemes, the data to be joined must be encrypted under the same key that has been shared among the owners. So one limitation of these searchable encryption schemes is that a shared key is needed, and hence it cannot be applied directly when the participating parties are mutually distrustful.

In this work we mainly focus on the works that allow parties to link their data using some sensitive attributes while protecting these attributes. This problem is known as privacy preserving record/data linkage (PPRL).

The record linkage problem is defined as the process of finding records belonging to the same entity, across two or more data sets [30]. Some of the protocols allow the parties to directly communicate and perform the matching process, and some utilize the service of a third party to accomplish the task of record matching.

PPRL methods can be classified into two major categories: deterministic and probabilistic. Deterministic PPRL methods establish the linkage between two records based on the

exact agreement/disagreement of one or more hash values of the linkage variables [24, 43]. Deterministic PPRL methods have very low number of false positives [36, 71] and are scalable owing to the simplicity of value similarity computation (equality). However, deterministic PPRL are unable to match records with typographical or phonetic errors.

We consider the mutual (direct) sharing and the publish/subscribe via third party scenarios. We consider methods that allow data linkage based on equality of the attributes (deterministic linkage) and methods that use similarity of the attributes (probabilistic linkage). In addition, we investigated works that speed-up the linkage process, specifically a method that indexes the data known as blocking. In the following sections we list the most notable related works.

3.1 Bloom Filter and Dice Coefficient

A Bloom filter is a data structure for checking set-membership [7]. It can be used to link records [21, 23, 72, 76]. The basic steps are: 1) tokenize the value of a linkage variable into n-gram tokens, 2) hash the tokens with a family of hash functions and map the resulting hash values to a Bloom filter bit vector, and 3) based on the strings of binary values in two Bloom filter bit vectors, compute the approximate similarity.

Dice coefficient (DC) is often used as a similarity score to compare two Bloom filters [23]. Linkage variables can be either individual values (e.g., first name, last name) or a combination of multiple values [76]. The Bloom filter linkage method when used with an effective blocking scheme has been proven to be a scalable solution for probabilistic PPRL [72]. However, the hash values in the form of Bloom filter bit vectors are susceptible to frequency-based attacks that potentially can reveal the original clear-text values [66].

Bloom filter has been used for probabilistic PPRL by Schnell, et al. in [76]. In Schnell's work, the parties create Bloom filters from their record using some mutually agreed upon parameters like (type and number of hash functions) and send those Bloom filters to a TTP to compute the respective Dice coefficient. The assumption is that the TTP will be unable to infer any sensitive information from the Bloom filters. However, other works show that it

is possible to launch frequency attacks on the Bloom filters and information can be leaked [66].

3.2 Deterministic Linkage

One of the sharing scenarios we are interested in is the publish/subscribe. In this setting, we respectively denote by publisher, subscriber and broker the data owner, data querier and the host. The challenging scenario addressed in this work considers many competitor publishers that do not want to reveal any information about their data to each other but would like to anonymously and securely share some information with the subscriber. In addition, the subscriber is not only interested in querying their data separately, but jointly in order to find connected records across the databases. Furthermore, the subscriber wants to be able to retrieve updates about some previously queried entities.

Solutions proposed in [8, 19] introduce a third party (Honest Broker) who will work as a mediator between the querier and the owners. This solution is impractical and not secure. In fact, although these solutions could be easy to deploy, non-cryptographic hash usage makes unauthorized users able to compute the hash; if the domain of the information is small or well known then it is possible to find the original information or at least to verify if some value exists or not. If pseudo-random functions are used instead, then a shared secret key must be used, which is undesirable in our setup because of the competition between different publishers. In addition, owing to the significant amount of trust required on the Honest Broker it becomes an appealing target for the attackers and malicious insiders [2, 42].

Yin and Yau [91] propose a privacy preserving repository for data integration across data sharing services that allows owners to specify integration requirements and data sharing services to safeguard their privacy. The owners determine who and how their data can be used. The authors presented what they call context aware data sharing to help data services to share data with the repository. However, the matching process between records is done using the hash value of their identification information, which is not secure and does not preserve privacy. This scheme suffers from the same underlying problems of the previous construc-

tion, namely, a mandatory secret sharing between competing parties. Carbunar and Sion [10] introduce a mechanism for executing a general binary join operation on an outsourced relational database with low overhead using predefined binary finite match predicates for computed match set. However they assume that the keys of both columns used in the join are known to the data owner. If the data belongs to two different owners, this key must be shared beforehand – essentially the same problem as earlier.

Chow et al. [15] propose a model for performing privacy preserving operations in a distributed database environment. The model, called Two-Party Query computation, comprises a randomizer and computing engine that do not reveal any information between themselves. The model in essence emulates a central party with the functions split into two entities. In order to guarantee that the randomizer and the computing engine do not collude, the authors proposes to use key agreement protocol among the participating entities. This protocol has limited applicability to our scenario since it doesn't not support the retrospective queries.

Some research developed protocols targeted at private set intersection [18, 32, 38] and secure multi-party computation (SMC) [4, 22, 28, 37] to solve the record linkage problem. However these techniques do not fit directly in our scenario because either they require a shared key or the parties involved learns the result of the computation. Further more, some of these techniques do not scale very well with the number of parties and the size of the data sets, and incur large computation or communication overhead [84].

Finally, a solution presented by Tassa et al. [81] targets gathering data between horizontally or vertically divided dataset while preserving sensitive information. The computation is over sanitized data set using k-anonymity or l-diversity sanitizing techniques. While this technique does not require a third party, it requires many assumptions over the structure of the data residing in the different parties.

3.3 Probabilistic Linkage

Approximate comparison of string values and distance measures allow researchers to deal with data discrepancies and errors that causes such apparent differences in duplicate records [1, 9, 64, 70].

Probabilistic linkage methods determine the likelihood that two records refer to the same entity. The most widely used probabilistic PPRL is the Fellegi-Sunter (FS) method which uses conditional probabilities to estimate match and non-match numeric scores for each value of a linkage variable [25, 30]. Based on these scores, a normalized summation weight is assigned to each linkage variable indicating its significance in contributing to an overall similarity score [44]. The similarity between the values of two linkage variables in each pair of records is quantified by *distance*. For example, Levenshtein distance is used to measure the distance of clear-text values [65]. The overall similarity score between two records is computed as the weighted sum of the distances of all linkage variables. For encrypted values, distance is usually estimated based on number of overlapping hashed consecutive letters (e.g., n-gram) generated from the original clear-text values. Well-known methods for distance computation for encrypted data include Jaccard similarity and Bloom filter with Dice coefficient [21, 58, 76].

In [49], a framework for privacy-preserving approximate record linkage is proposed. The framework is based on a combination of secure blocking and secure matching. The secure blocking is based on phonetic algorithms statistically enhanced to improve security, and the secure matching is performed using a private approach of the Levenshtein Distance algorithm. The main advantage of blocking is that it results in a significant decrease of record comparisons.

A three-party protocol, proposed in [75], relies on identical encryption at the sources and uses Bloom filters to perform similarity search. The linkage center performs probabilistic record linkage with encrypted personally identifiable information and plain non-sensitive variables. To guarantee similar quality and format of variables and identical encryption

procedure at each site, the linkage center generates semi-automated pre-processing and encryption templates based on information obtained via a new method called data masking that hides personal information.

Similarity preserving data transformation approaches like [67, 73, 77, 88] have been also presented.

In [73], a protocol that provides privacy at the levels of both data and schema matching was presented. In this protocol, records are mapped into an Euclidean space using a set of pre-defined reference values and distance function, while preserving the distances between record values. Then the semi-trusted (HBC) third party will compare the records in the metric space in order to decide their matching. The secure schema matching requires global schema, provided by the third party, on which the parties map their own local schemas. In [88], an approach based on the work in [73], that doesn't need a third party was presented. A complex plain is created then an adjustable width slab is moved within the complex plain to compute likely matched pairs. In [67], public reference tables are used to encode names as the distance to all reference points.Then a third party will estimate the distance between pairs based on their encoding.

In [77], Bloom filters with a set of cryptographic hash functions are used to encode the attributes of the data records. The encoded records are then compared via a set-based similarity measure.

Composites of multiple linkage values, also referred to as hash keys were proposed to link erroneous data [52, 72]. The number of hash keys depends on the availability of shared linkage variables among data sources.

Most of these protocols where either theoretically or experimentally proven to be efficient. However most of these works do not fit directly into our publish/subscribe scenario [4, 18, 32, 38, 84] because the data source parties participate in the matching process will know the matching results, or they work for deterministic match [18, 32], require a shared key [75], or rely on expensive SMC operations that do not scale very well with the size of the data [4, 22, 28, 37].

3.4 Secure Multi-party Computation PPRL

In [56] an SMC based PPRL is proposed. First data is blocked (indexed or grouped into blocks such that records in each block have similar values in some of their attributes) using publicly available identifiers, then using a third party's (TP) public key and Paillier encryption scheme, party A encrypts his encoded data and sends it to party B. Party B then uses his data and the TP's public key to perform some homomorphic operations on A's encrypted data, and sends the encrypted results to the TP. The TP then uses its private key to compute the final similarity values between the record pairs. Though this work uses secure computation, which is computationally expensive (a 10K data set processing took around 23.3 hours), it differs from our methods in many ways. First, SMC PPRL needs a TP and our linkage scheme does not. Second, linkage attribute that is of type string requires the values to be encoded as a bit vector of all possible bi-grams where the cells of the bi-grams that exists in the data values are set to 1's, and the rest are set to 0's and then the Jaccard similarity is used as similarity measure. We use a set of Bloom filters to encode different combinations of linkage identifiers and use the Dice coefficient as similarity measure. Third, since we do not use a TP, compromising TP or its public key and causing information leakage is not possible in our scheme.

Wen et al. [87] presents two protocols in the semi-honest model. The first protocol is deterministic PPRL based on Oblivious Bloom filter Intersection. The second protocol is probabilistic PPRL, which extends their exact PPRL protocol by incorporating Locality Sensitive Hash (LSH) functions. In their protocol each party encodes its entire database as one Bloom filter (BF), then they use these Bloom filters to find the Ids of the matched records. Inserting large set of record ids in one BF will require large BF in order to control the false positives (FPs). In our method, we use four Bloom filters, of size 1k-bits each, to encode four, well studied [52], different combinations of some attributes of each record. Then the similarity of any pair of records will depend on the dice coefficients (DCs) of their

corresponding Bloom filters. The FPs in our method will depend on the preset threshold that compared with the DCs.

Vatsalan et al. [84] proposed a two-party protocol that uses reference values. In their work, phonetic encoding (such as Soundex) was used to generate blocks to improve the efficiency of the matching process. The quality of the results will be highly affected by the selection of the public reference lists.

For efficient record linkage of large datasets, an indexing method called blocking is usually used to reduce the number of compared record pairs. Inan et al. [39] proposed a blocking protocol that provides strong data protection compliant with differential privacy [27]. In their work, the datasets are partitioned into subsets, using d-dimensional hyper-rectangles constructed from ranges of the domain values of some d attributes. Random noise (adding fake records, or suppressing some records) is injected in each partition to hide the actual record counts. Then using the shared subset extents, each party will filter out record pairs in the subsets that their extents do not intersect.

3.5 Blocking

Many schemes of privacy preserving blocking have been proposed for scalable PPRL [16]. However, most of these schemes require either TTP [3, 46, 47, 50, 51, 55, 83], a shared reference set, where each reference represents a block, and each record is compared to all the references and assigned to the block with highest similarity to its reference [46, 50, 55, 74, 83, 85, 86], or need a pre-defined blocking variables, so the block ids will be created by the distinct values each variable can have [3, 39, 40, 46, 47, 48, 50, 51, 55, 61, 74, 83, 85, 86, 89].

In [61], clustering techniques were used for privacy preserving blocking. They used approximate distance measure to efficiently partition the data into overlapping groups called canopies.

In [50], nearest neighbor clustering is used to cluster the records using a shared reference table among the parties. Then a TTP will gather all clusters formed by each party and

merge the clusters with corresponding ids, and performs the PPRL only between members of each cluster.

In [46], authors propose a privacy preserving blocking technique based on the use of reference sets. The idea is to repeat the blocking procedure with multiple, different reference set samples, so as to attain multiple record to block assignments. The blocking attributes are shared among the parties.

Another method using a common reference set is proposed in [74]. In this method strings are embedded in a Euclidean space, then similar vectors is found using multidimensional tree-based index. This method has performance drawback.

A phonetic encoding-based blocking method is proposed in [47] that generalizes strings using phonetic encoding functions then sends them to TTP, who builds blocks using common codes from both data sets. Phonetic codes do not work with other data types, and are considered inadequate in representing similar strings effectively.

A hierarchical clustering and public reference sets based blocking method is proposed in [55]. TTP generates the base clusters on some attribute using the reference sets, then each data owner assigns his records to these base clusters according to their similarity to the members of each cluster. Random noise drawn from a zero-mean Laplace distribution is added to each cluster to hide its cardinality and achieve desired level of differential privacy.

In [85], a two-party blocking method using reference sets is proposed. The two parties independently generate clusters using reference values, then those reference values are exchanged, merged and sorted. Then, the sorted nearest neighborhood method is applied on this sorted list of reference values to create the candidate record pairs.

The main drawback of reference sets based blocking methods is that its accuracy is strongly dependent on these reference sets, which, in most cases, should be a subset of the values used in the data sets for better accuracy. In addition, its security require the presence of TTP to match the blocks, otherwise the data owners will gain too much information about each other's data using the common reference sets.

A method based on Hamming Locality-Sensitive Hashing technique HLSH, which uses Bloom filter representation of the data and the service of TTP, is proposed in [51]. It basically assigns the Bloom filters to a fixed number of independent hash tables buckets. If both parties use the same settings, then similar Bloom filters will be assigned to at least one same hash table bucket. Candidate record pairs are formulated by scanning the buckets.

Authors in [40] proposed a hybrid approach that combines sanitization and cryptographic techniques that enable users to trade off between privacy, accuracy, and cost. A blocking phase, that operates over sanitized data, is applied to filter out, in a privacy preserving manner, pairs of records that do not satisfy the matching condition. The blocking phase compares pairs of partitions produced in the sanitization phase against one another based on the regions covered by each partition. Sanitization techniques such as as k-anonymity [80], or permuting with noise to achieve differential privacy [27] were suggested to be used. The final step in the blocking phase uses secure computations such as [90] [82],[31], [53] to label any pair of records that were not classified as match or non-match in the previous blocking step.

Chapter 4

Privacy Preserving Record Matching using Automated Semi-Trusted Broker

In this chapter, we present a novel scheme that allows multiple data publishers that continuously generate new data and periodically update existing data, to share sensitive individual records with multiple data analyzers while protecting the privacy of their clients. An example of such sharing is that of health care providers sharing patients' records with clinical researchers. Traditionally, such sharing is performed by sanitizing identifying information from individual records. However, removing identifying information prevents any updates to the source information to be easily propagated to the sanitized records, or sanitized records belonging to the same client to be linked together. We solve this problem by utilizing the services of a third party, which has very limited capabilities in terms of its abilities to keep a secret, confidential and by encrypting the identification part used to link individual records with different keys. The scheme is based on strong security primitives that don't require shared encryption keys.

4.1 Contribution Summary

In this work, we address this problem of privacy preserving record linkage by proposing a secure scheme based on partially homomorphic encryption and a third party. The third party is responsible just for automatically and blindly performing record matching. It is honest in the sense that it follows the protocol correctly but is not trusted to keep a secret confidential. It is curious about the sensitive information contained in individual records and can act accordingly. However, our protocol ensures that it is prevented from getting such

information without colluding with publishers. The third party knows that two publishers or subscribers have clients in common; however, it does not know the identities of these clients.

The main idea behind our protocol is as follows. Each data publisher creates a "key converter" in collaboration with other data publishers. Each key converter is then given to the third party (henceforth referred to simply as the broker). Each data publisher encrypts critical identification information of its data using it own secret key. Later, the broker uses the "key converters" to blindly transform all publisher-encrypted identification information to an alternate encrypted form under some other key that is not known to any party including the broker itself. The broker needs to collude with at least one of the publishers to find that key. Once the linkage information is encrypted under the same key, the broker can easily determine matching records. The broker can also keep track of the individuals found at different sites for retrospective update queries purposes. Since the data is encrypted at the source with different keys that the broker does not have access to, the risk of privacy breach in case of the broker getting compromised is limited.

4.2 Scheme Construction

Our scheme works in three phases: the *setup* phase, the *encryption of query results* phase, and the *secure record matching* phase. The setup phase is executed only once by the data publishers to create the so-called publishers' "key converters". It utilizes properties of homomorphic encryption to create these "key converters". The encryption of query results phase occurs at the publisher side whenever it executes a query, to encrypt the identifying part of the query results. This encryption is performed using the publisher's secret non-shared key before sending the results to the broker. The secure record matching phase occurs at the broker side to determine the linkages between the records coming from the publishers after executing the queries.

We begin by discussing the adversarial model and privacy requirements, then we explain each phase in details.

4.2.1 Adversary model and privacy requirements

Data publishers are considered competitors who do not want to share data with each other. Each publisher tries to determine information about the clients of competitor publisher, or at a minimum, tries to determine if any one of its clients is also a client of its competitor. However, publishers are also honest in the execution of the protocol steps and willing to share information with subscribers *privately*, that is, without revealing real identities, and *securely*, that is, without any leakage of information to other data publishers, and without revealing the publisher identity to the subscribers.

A data subscriber, on the other hand, needs to determine if any information that came from different publishers belong to the same individual so they could be grouped together as such and treated accordingly. For example, if a researcher is looking for the side effects of a new drug used for skin treatment on patients who has kidneys problems, then he has to match patients from the (Dermatology) and (kidney diseases) departments to find patients under these conditions. We need to allow such grouping at the subscriber side.

Further more, the subscriber is allowed to issue *retrospective* queries regarding some individual client, for example, update queries regarding the progress of treatment of certain patients. Subscribers (researchers) are considered curious in the sense they will try to determine the real identities of the individuals. Some information about individual identities might be leaked from their non-identification information (i.e. eye color, age, weight, etc.) using statistical inference techniques. This is a separate problem that needs to be addressed with anonymization (i.e. k-anonymity) or other sanitization methods, and is not considered in this work.

The broker is honest in the sense that it will not collude with any of the parties, but is curious and not trusted to keep a secret, secret. The broker will work as a mediator between the data publishers and the subscribers by honestly performing the following tasks:

- Hide the source of information (publishers and clients' identities) from the subscribers.

- Blindly determine record linkages among the encrypted publishers' records and assign alternate random identifiers to the linked records before sharing them with the subscribers. The broker will just know the linkages without knowing the real identifiers.

4.2.2 Setup phase

In order to create its key converter, each publisher is required to initially interact with other publishers participating in the sharing system, keeping in mind that other publishers are competitors. Every publisher needs to go through the setup protocol once at the beginning of the system instantiation when it joins the sharing system. An existing publisher also needs to embark upon the setup phase if a refreshing of keys is required when new publishers join the system. These key converters will be delegated to the third party (broker) and will be used to convert records encrypted under different keys of different publishers, to records encrypted under a common key. This common key is such that it cannot not be re-constructed by any of the parties, namely, individual publishers, broker and subscribers. This means that the encrypted data is safe if there is no collusion between any of the publishers and the broker at the instantiation time. In this scheme, we propose a secure protocol to create the key converters among the publishers using ElGamal homomorphic cryptosystem that supports product operation over the encrypted keys. At the end of this phase, every publisher is associated with a special key converter that allows the broker to perform the matching process.

Each publisher from the set of N publishers $D = \{d_i\}_{i\in[N]}$ has its secret key $\mathsf{sk_i}$, and the broker which has a public key pk and private key sk pair. The setup phase is illustrated in Fig.4.1 and works as follows.

- **Step 1**: The broker broadcasts its public key $\mathsf{pk} = (G, q, g, g^{\mathsf{sk}})$ for ElGamal homomorphic encryption to all publishers.

- **Step 2**: Each publisher d_i generates an initial random secret $r_i \xleftarrow{R} \{1, \cdots, q-1\}$, where q is the order of G, encrypts it using the master key pk. Let $t_{i\to j}$ denote the temporary encrypted key converter of publisher i when being processed by publisher

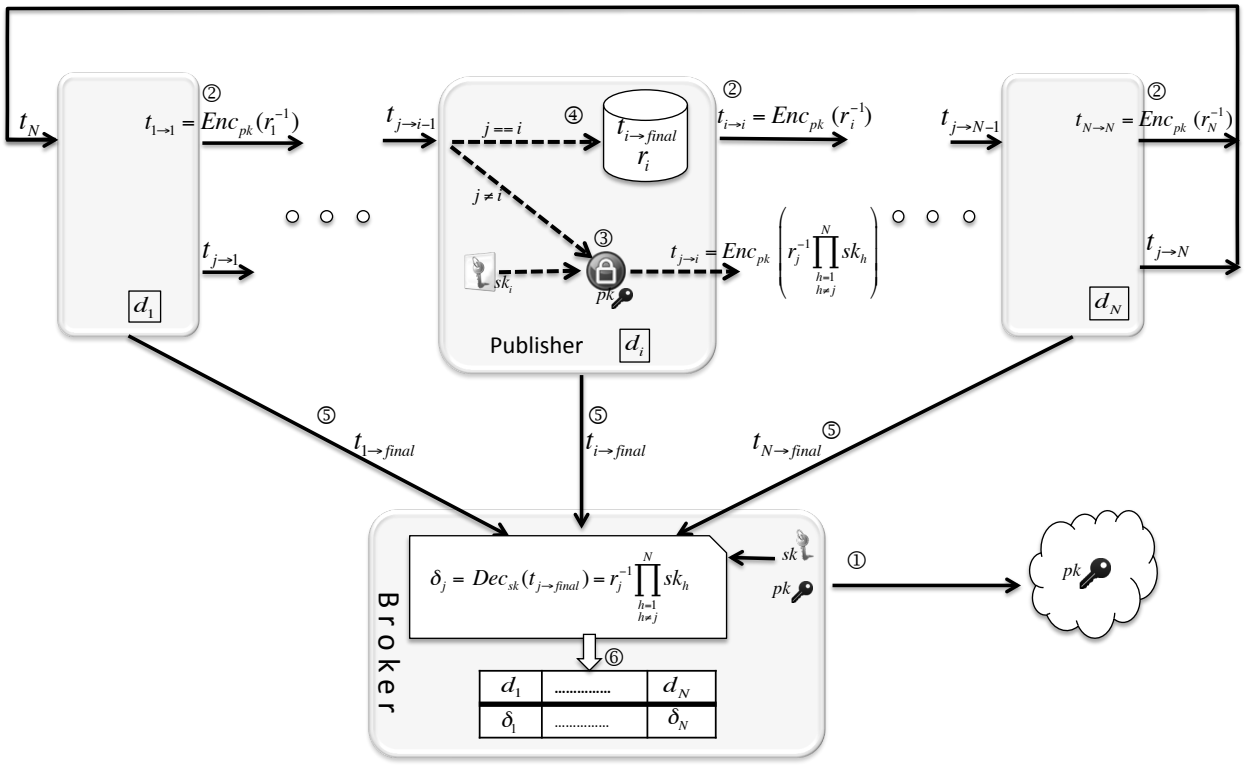


Figure 4.1: High level setup phase illustration

j, and $t_{i\to final}$ the final converter when it gets back to the publisher i after being processed by all parties. Publisher d_i generates the initial temporary encrypted key converter $t_{i\to i} = \mathsf{Enc_{pk}}(r_i^{-1})$, then forwards it to the next publisher $d_{(i \mod (N))+1}$.

- **Step 3**: Each publisher d_j receives a value $t_{i\to j}$ from its upstream neighbor d_i $(i \neq j)$, securely multiplies it using ElGamal homomorphic cryptosystem with its secret key $\mathsf{sk_j}$ encrypted under the broker's public key pk as follows:

$$\begin{aligned} t_{i\to i+1} &= t_{i\to i} \cdot \mathsf{Enc_{pk}}(\mathsf{sk_{i+1}}) \\ &= \mathsf{Enc_{pk}}(r_i^{-1} \cdot \mathsf{sk_{i+1}}) \end{aligned}$$

This operation is repeated through $N-1$ publishers. The temporary value of publisher d_i generated by the j^{th} publisher equals:

$$t_{i\to j} = \mathsf{Enc_{pk}}(r_i^{-1} \cdot \prod_{c=i+1}^{j} \mathsf{sk_c})$$

- **Step 4**: After $N-1$ transitions, the initials publisher receives the final key converter $t_{i\to final}$ as

$$t_{i\to final} = \mathsf{Enc_{pk}}\left(r_i^{-1} \prod_{\substack{j=1 \\ j\neq i}}^{N} \mathsf{sk_j}\right)$$

- **Step 5**: After each publisher is able to generate its key converter value after being processed by all other publishers, the publisher d_i sends its $t_{i\to final}$ value to the broker, and saves a copy of it for future updates in case new publishers joined the system.

- **Step 6**: For each $t_{i\to final}$, the broker extracts the key conversion factor δ_i such that:

$$\delta_i = \mathsf{Dec_{sk}}(t_{i\to final}) = r_i^{-1} \prod_{\substack{j=1 \\ j\neq i}}^{N} sk_j$$

Key converter refresh: If a new publisher d_{N+1} joins the system, then it follows the previous steps to create its own key converter t_{N+1}, while other publishers just need to update their own key converters. To accommodate the key of the new publisher, old publishers send their previously created $t_{i\to final}$ values to the new one, which in turn securely adds its secret key sk_{N+1} to this value $t_{i\to final}$ to get $t'_{i\to final}$, and sends it back to the source. Each publisher d_i then refreshes its $t'_{i\to final}$ with a new random value r'_i and sends the updated $t'_{i\to final}$ to the broker. This new random value r'_i is used to prevent the broker from extracting information about the newly added secret key of the new party d_{N+1} by comparing the new with the old $t_{i\to final}$ values. Each publisher updates its secret key with this new random value too.

Note: Careful readers might think that it would be simpler if publishers broadcast their keys and then locally compute the key converters. It is true that in term of communication overhead, this method also involves $O(n^2)$ interactions, however, in term of information shared, it leaks more. In this solution, each publisher submits its key encrypted with the public key of the broker. If the broker can somehow eavesdrops the communication between the publisher, it can decrypt and obtains the key of all publishers.

4.2.3 Encryption of query results phase

This phase is triggered by a query sent by a subscriber requesting information from publishers. This represents a *data pull* model; however, our scheme can be also used in a *data push* mode where publishers send data directly to the broker which redirects it to the corresponding subscribers. After executing the received query, each publisher encrypts the identifying parts of the query results using any cryptosystem that relies on DDH (Decisional Diffie-Hellman) or DL (Discrete Logarithm) hardness assumptions, such as ElGamal cryptosystem. For performance reasons, our construction uses Elliptic Curves (EC) instead of Finite Groups of large primes as the underlying group for the used cryptosystem.

Each publisher has the publicly known ElGamal EC parameters, i.e., the curve parameters $E(\mathbb{F}_q)$ and the point on the curve P of prime order n. The public/private key pair will be $(r_i \cdot \mathsf{sk}_i \cdot P, r_i \cdot \mathsf{sk}_i)$ and both of th keys are kept secret. The message to be encrypted, id, in our multiplicative scheme needs to be a scalar. We denote by $\mathsf{E}(.)$ the encryption of ElGamal based on EC.

The publisher first hashes the identifying part of every record in the result set using a universal hash function H. The result set is the data outputted by executing the subscriber query. The publisher uses its secret key multiplied by the corresponding random value, $(r_i \cdot \mathsf{sk_i})$, to encrypt the resulting hash. That is, the encryption of any identifier id will be:

$$\mathsf{E}_{(r_i \cdot \mathsf{sk_i})}(H(id)) = H(id) \cdot r_i \cdot \mathsf{sk_i} \cdot P$$

Finally, the publisher substitutes the real identifying part, id, by $\mathsf{E}_{(r_i \cdot \mathsf{sk_i})}(H(id))$ for all records in the result set. Finally, each record is composed of the encrypted identification part, plus, the other client's information. The data in plaintext in each record will be sanitized if necessary, according to the publisher's policy, before being sent to the broker. Sanitizing techniques details are out of scope of this work.

Publishers can avoid having the broker store the key converter $(\delta_i)_{i \in [N]}$. For this purpose, each publisher encrypts the identifiers of the query results with a new random value r_i, updates the key converter $t_{i \to final}$, then sends these results to the broker accompanied with

the new key converter. This solution adds negligible communication overhead, but ensures a zero-key stored on the broker side.

4.2.4 Secure record matching phase

The broker receives the encrypted identifiers with different keys from different publishers. The broker's job is to merge similar clients' records from different publishers such that they will map to the same newly generated identifier. The broker will use the key converters δ_i to change the encryption key in such a way that similar data will be deterministically encrypted with the same key without requiring any decryption to be performed along the way.

The broker uses the δ_i values to convert any identifier id encrypted by publisher d_i under its secret key $(r_i \cdot \mathsf{sk}_i)$, to a value encrypted under a different secret key Δ, i.e., $\mathsf{E}_{\Delta}(H(id))$. The key $\Delta = \prod_{i=1}^{N} \mathsf{sk}_i$ is resulting from the product of all the secret keys of all publishers. In order to perform the secure record matching, the broker re-encrypts the encrypted identifying parts of the records coming from the publisher d_i using the corresponding key converter δ_i as:

$$\mathsf{E}_{\delta_i}\left(\mathsf{E}_{(r_i \cdot \mathsf{sk}_i)}(H(id))\right) = \mathsf{E}_{\Delta}(H(id))$$

That this process does indeed perform correct matching is shown by the fact that:

$$\begin{aligned}
\mathsf{E}_{\delta_i}\left(\mathsf{E}_{(r_i \cdot \mathsf{sk}_i)}(H(id))\right) &= \mathsf{E}_{\delta_i}\left(H(id) \cdot r_i \cdot \mathsf{sk}_i \cdot P\right) \\
&= H(id) \cdot r_i \cdot \mathsf{sk}_i \cdot P \cdot \delta_i \\
&= H(id) \cdot r_i \cdot \mathsf{sk}_i \cdot P \cdot r_i^{-1} \prod_{j=1, j \neq i}^{N} \mathsf{sk}_j \\
&= H(id) \cdot \prod_{j=1}^{N} \mathsf{sk}_j \cdot P \\
&= H(id) \cdot \Delta \cdot P = \mathsf{E}_{\Delta}(H(id))
\end{aligned}$$

In order to maintain the linkages between publishers' data records and the randomly generated identifiers for subscribers, the broker keeps track of the processed identifiers for both flows, i.e., from publishers to subscribers and vice versa. The aim of this mapping is two folds: first we do not want to give the ability to the subscribers to know whether they

share the same client and second give the ability to the broker to map back these random values to the same client. For this purpose, the broker builds two secure inverted indexes, one to map the encrypted identifiers after conversion (i.e. $\mathsf{E}_{\Delta}(H(id))$) to their corresponding subscriber identifiers such that for each we generate a random value rid concatenated to the subscriber identifier sid. The second maps $rid \| sid$ to the set of corresponding encrypted clients' identifiers concatenated with their publisher id such that $\mathsf{E}_{\mathsf{r_i \cdot sk_i}}(H(id))||d_i$, for $i \in [N]$, see Table. 4.1. These secure inverted indexes can be constructed following searchable encryption data structure instantiation, see [12, 17].

Table 4.1: Conceptual representation of secure inverted indexes

$\mathsf{E}_{\Delta}(H(id))$	$rid \| sid$
0xAB4542..24	0x354AE2..16 ‖ 1, 0xF14598..24 ‖ 5
0xC2C6A5..59	0x413F56..AE ‖ 2
.....	..

$rid \| sid$	$\mathsf{E}_{r_i \cdot \mathsf{sk}_i}(H(id))\|\|d_i$
0x354AE2..16 ‖ 1	0x6547A..6A ‖ 2, 0x45CA4..B2 ‖ 5
0x413F56..AE ‖ 2	0x48F53..12 ‖ 11
..	.....

4.2.5 Matching phase walk-through

We now summarize all the interactions between the three parties, namely, publishers, subscribers and the broker to describe how privacy preserving record matching system works to serve the subscriber's needs. These interactions schematically shown in Fig. 4.2, with each of following steps corresponding to the numbers shown in the figure.

1. The subscriber sends a query Q to the broker.

2. The broker sanitizes the query if necessary, and checks if this query Q is a query that seeks information about new clients or a retrospective query (requesting more information about an individual whose data has been seen before). If it is a new query, it forwards the query to publishers and wait for the answers.

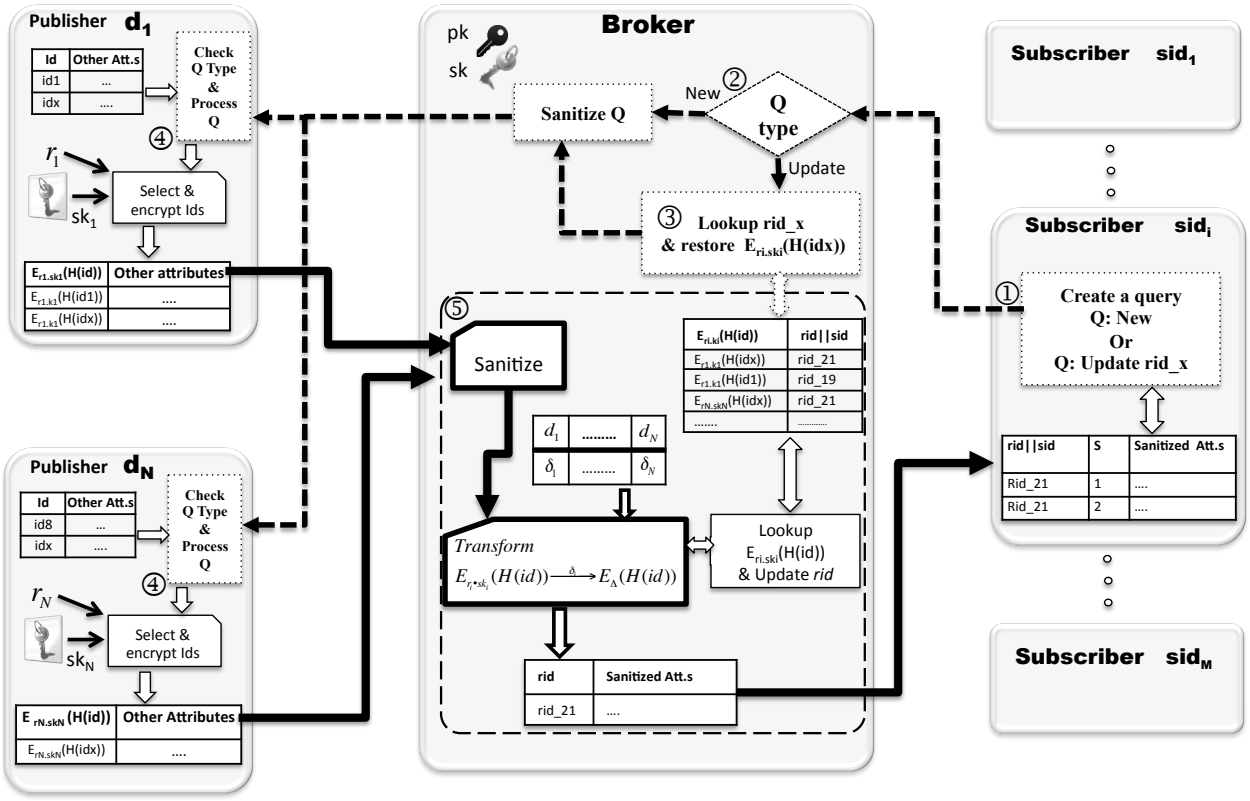


Figure 4.2: Matching phase walk-through illustration

3. If the query Q is a retrospective query, the broker first looks up the $rid||sid$ in the inverted index for the corresponding encrypted identifiers and their associated publisher identities, i.e. $\mathsf{E}_{\mathsf{r_i \cdot sk_i}}(H(id))||d_i$. The broker then replaces $rid||sid$ with the corresponding $\mathsf{E}_{\mathsf{r_i \cdot sk_i}}(H(id))$ and finally forwards the query Q only to their associated publishers d_i.

4. For each query Q it receives, the publisher d_i checks if it is a new query or a retrospective query. If it is a retrospective query, the publisher can directly look up the matching records while for a new query a search has to be done to find the corresponding clients. In either case, the publisher applies its local sanitizing policies to the results, encrypts the identification part using its secret key $(r_i \cdot \mathsf{sk}_i)$, and finally sends the results to the broker.

5. Upon receiving the results of the forwarded query from the publishers, the broker further sanitizes the results according to the general sharing policies and regulations. Then it performs the secure record matching process, and updates its inverted indexes as follows:

   - Using the first inverted index, and for each distinct $\mathsf{E}_{\mathsf{r_i \cdot sk_i}}(H(id))$ in the query result, apply the key converters to get $\mathsf{E}_{\Delta}(H(id))$. If a match is found then this identifier has been sent before (it might be from a different publisher though). So the broker retrieves the corresponding $\{rid \| sid\}$, and updates the second inverted index with the encrypted id $\mathsf{E}_{\mathsf{r_i \cdot sk_i}}(H(id)) || d_i$ in case it has been previously sent from a different publisher.
   - If the converted encrypted $\mathsf{E}_{\Delta}(H(id))$ is not found, then it means that this identifier has not been sent before by any publisher. The broker adds this converted value to the first inverted index, with a new subscriber and random identifier $rid \| sid$. The second inverted index is updated accordingly.
   - The encrypted identifier $\mathsf{E}_{\mathsf{r_i \cdot sk_i}}(H(id))$ in each record in the results is then replaced with its corresponding $rid \| sid$ before being sent to the corresponding subscribers.

   As we have mentioned earlier, these inverted indexes are encrypted under the broker's secret keys, and all the searching and update operations are performed on encrypted data using any adaptive secure searchable encryption scheme data structures.

4.3 Complexity and Security Analysis

Complexity: Our scheme is practicable and can be efficiently implemented in real scenarios. The construction is composed of three main steps, the setup, the encryption of query results and the secure record matching phase. The setup phase depends on the number of publishers N. The entire setup phase results in a total communication overhead which is $O(N^2)$. To generate *one* key converter, the publishers needs to transfer one message each, while the computation is constant per each transfer. The total communication and compu-

tation overhead per publisher is in $O(N)$. The setup phase is performed just once and is done in an off-line manner. The encryption of query results overhead depends on the matching set and the data structure that the publisher is using for its own data storage. The search complexity depends on the query sent by the subscriber through the broker. Also the complexity of this phase greatly depends on the size of the database of the publishers and the sanitizing protocols used as well. For this reason, we consider only the communication overhead for this phase. For the last phase performed by the broker, given all identifiers that match the query, the goal consists of retrieving all subscribers identifiers as well as the randomized identifiers. Using SSE, the search is optimal and similar to plaintext data. Given an identifier, the search complexity will be in the number of matching results.

To sum up, the construction does not induce any overhead during the matching phase more than the one expected on plaintext data. The linear construction of the key converters is done once during the instantiation of the protocol.

Security: The key converter is based on ElGamal homomorphic encryption which is semantically secure. However, we want to make sure that the output of the key converter received by the broker will not leak any information about the secret keys of any publisher. The broker receives N key converters for N different publishers. Every key converter is composed of $N-1$ secret key and a random value. The broker then has, at the end of the setup phase, a linear system of N equations with $2N$ unknowns. This is an under-determined system which is information theoretically secure. This property ensures that the broker cannot, based on the key converters, recover the secret keys of the publishers. Note that this is true as long as the publishers do not collude with the broker. We can enhance this key converter generation with much secure schemes such as multi-party computation where we can take into consideration malicious parties.

During the encryption of query results phase, every publisher is encrypting the records with a different secret key. Thus, even if a publisher eavesdrops over the communication between publishers and broker, it cannot infer anything about the clients identity.

For the storage of the encrypted identifiers at the broker side, a conversion of the encryption is necessary. The broker therefore knows that this encrypted identifier (whatever be the identifier) is linked to the same client in different publishers. This feature is very important for the scheme correctness since it enables the broker to aggregate the exact subscriber query results. From a security perspective, we want to hide not only the association between encrypted identifiers and subscribers but also the mapping between the publisher identifiers and the encrypted identifiers. For this purpose, we use a symmetric searchable encryption inverted index that enables to store securely these mappings. Consequently, even if the data at the broker is somehow leaked, the entire inverted indexes are encrypted and the mapping will not be disclosed.

Chapter 5

PPPRL: Privacy Preserving Probabilistic Record Linkage using Locality Sensitive Hashes

In practice, the records to be matched often contain typographic errors and inconsistencies arising out of formatting and pronunciation incompatibilities, as well as incomplete information. When encryption is applied on these records, similarity search for record linkage is rendered impossible. The central idea behind our work is to create characterizing signatures for the linkage of attributes of each record using minhashes and locality sensitive hash functions before encrypting those attributes. Then, using a privacy preserving record linkage protocol we perform probabilistic matching based on Jaccard similarity measure. We have developed a proof-of-concept for this protocol and we show some experimental results based on synthentic, but realistic, data. Our protocol is easily generalizable to other application areas, such as, privately sharing Internet monitoring data sets, and record de-duplication.

5.1 Contribution Summary

In this work, we build upon our previous secure record matching protocol [57], that performs a deterministic record matching using the services of a *semi-trusted* broker, in order to construct a *probabilistic* record matching protocol. It takes into account the different types of discrepancies mentioned above. Probabilistic in the context of this work means that the matching is performed based on the likelihood of similarity with certain probability. We call the set of attributes used to perform the record matching, the *linkage attributes*. In order to hide the identities of the matched entities from all of the parties, the matching process is performed on encrypted data under different keys. Any two records of different data sets are said to be matched if they belong to the same individual/entity. In our demonstration,

we use personal identification information as the linkage attributes (e.g. names. SSN, email address, DOB, Driving License number, etc.) for finding candidates for record matching. However, our protocol is easily generalizable to any other attributes that are of interest.

5.2 Problem Formulation, Definitions and Background

We abstract the problem as that of a group of $n \geq 2$ publishers (data sources) $\{P_1, \cdots, P_n\}$, with n databases $\{D_1, \cdots, D_n\}$ who would like to link and share their data with another group of $m \geq 1$ subscribers (e.g. researchers) $\{S_1, \cdots, S_m\}$ while protecting the privacy of their clients. At the end of the linkage process, we require that 1) the subscriber(s) gets the linked records identified by some random non-real ids, 2) none of the publishers knows the result of the linkage process, or that a certain client's information exists at other party's records, 3) none of the parties, including the party conducting the linkage process, should know the real identity of any client that it did not know a priori 4) any pair of records with similarity above a pre-set threshold must be linked.

Definition 2. - *Linkage Fields:* *For a database D_i of publisher P_i with a schema $C = (A_1, \cdots, A_w)$ that has a set of w attributes, we define the set of linkage fields $L \subseteq C$ as a subset of $t \leq w$ attributes that uniquely define the personal identity of the database records. We denote the linkage fields set as $L = \{L_1, \cdots, L_t\}$ and the set of records of D_i as $R = \{R_1, \cdots, R_v\}$. We refer to the value of the linkage field L_j of the record R_k by $R_k \cdot L_j$.*

Definition 3. - *Linkage Parameters:* *We associate with the linkage fields L sets of linkage parameters K, St, W, such that for each linkage field $L_i \in L$, we define some linkage parameters, $K_i \in K$ that represents the number of hash functions used for similarity calculations of the values of that field, similarity threshold $0 \leq St_i \leq 1$ which defines the minimum similarity threshold to consider the values of the linkage field as a match, and Linkage Field weight $w_i \in W : \sum_{1 \leq j \leq t} w_j = 1$ that represents the importance of that field in the matching decision.*

Definition 4. - *Field Match Score:* *Given two values $R_j \cdot L_i, R_k \cdot L_i$ of linkage field L_i of two records R_j, R_k, similarity function Sim, field similarity threshold St_i, and field weight w_i, a field match score FM_i is defined as:*

$$FM_i(R_j \cdot L_i, R_k \cdot L_i) = \begin{cases} 0 & \text{if } Sim(R_j \cdot L_i, R_k \cdot L_i) < St_i \\ w_i \cdot Sim(R_j \cdot L_i, R_k \cdot L_i) & \text{otherwise} \end{cases}$$

Definition 5. - *Record Match Decision:* *Given a similarity threshold TR and a set of Field Match scores $\{FM_i : 1 \leq i \leq t\}$ of two records R_j, R_k, the record match decision RM of R_j, R_k is defined as:*

$$RM(R_j, R_k) = \begin{cases} Match & \text{if } \sum_{1 \leq i \leq t} FM_i \geq TR \\ Non - Match & \text{otherwise} \end{cases}$$

5.2.1 Minhash Functions and Similarity

To calculate the similarity of two strings A and B, we use the Jaccard Similarity measure. First the two strings are converted to a set representation S_A, S_B respectively (e.g. using bi-grams). Then the Jaccard similarity, $Sim(A, B)$, is calculated on S_A and S_B as:

$$Sim(A, B) = \frac{\mid S_A \cap S_B \mid}{\mid S_A \cup S_B \mid}$$

If $Sim(A, B)$ is close to 1, then A and B are very similar. Let Ω be the set of all possible values of the set representation of the strings (e.g. all possible bi-grams of the alphabet), then MinHash (also called min-wise hash) is computed by applying a random permutation $\pi : \Omega \to \Omega$ on any given set S and selecting the minimum value.

$$MinHash_\pi(S) = min\{\pi(S)\}$$

It was proven by A. Broder [9] that the probability of the minhashes of any two sets S_A and S_B being the same is equal to the Jaccard similarity of the two sets; that is:

$$Pr(MinHash_\pi(S_A) = MinHash_\pi(S_B)) = \frac{\mid S_A \cap S_B \mid}{\mid S_A \cup S_B \mid} = Sim(A, B)$$

Thus, we can estimate the Jacard similarity of the two strings A and B by choosing a set of n (e.g. $n = 100$) independent random permutations, apply them to each string set representation to get a set of n minhashes (we also use the term signatures) of those strings, and compute how many corresponding minhashes of the two sets are equal. It is possible to simulate the n random permutations by n independent random hash functions (e.g. selecting a hash function and use n different random seeds).

5.2.2 Locality Sensitive Hashing – LSH

Locality sensitive hashing or LSH is a well known technique in the information retrieval community, that is used for determining similar items by hashing them in such a way that their hash values will collide with high probability. Let St_1, St_2 be a similarity metric (e.g. Jaccard similarity) and P_1, P_2 be two probabilities, then a family of functions $\mathcal{H}$ (e.g. minhashes) is called the (St_1, St_2, P_1, P_2)-sensitive LSH if for every $h \in \mathcal{H}$ and any x, y :

1. if $Sim(x, y) \geq St_1$, then the probability that $h(x) = h(y)$ is at least P_1.

2. if $Sim(x, y) \leq St_2$, then the probability that $h(x) = h(y)$ is at most P_2.

From above, a family of K minhash functions with a similarity threshold St_0 is a $(St_0, 1 - St_0, St_0, 1 - St_0)$-sensitive LSH for Jaccard similarity. For a better accuracy of the similarity estimation it is preferred to have St_1 and St_2 as close as possible and P_1 and P_2 as far as possible. We construct a $(St_0, 1 - St_0, \gamma, \beta)$-sensitive LSH family from a family of K minhash functions by grouping the minhashes (while keeping the order) into b groups of size r each, such that $K = b \cdot r$, and then hash each group to get b new signatures. Hence, $\gamma = 1 - (1 - St_0^r)^b$ and $\beta = (1 - St_0^r)^b$, where the probability that the new b signatures of x and y agree in at least one signature is γ.

We can further reduce the number of false positives by requiring that the b-signatures of x and y agree on at least m signatures (e.g. $m = 2$), then the probability of the b signatures to agree in at least m signatures will be:

$$\sum_{m \leqslant i \leqslant b} \binom{b}{i} St_0^{r \cdot i} (1 - St_0^r)^{b-i}$$

However, experiments show that $m = 1$ is sufficient.

5.3 The Secure Probabilistic Matching Protocol

In order to allow the parties to perform secure similarity match based on Jaccard measure, we improve our previous deterministic protocol to perform the match probabilistically with a predefined threshold. We improve the string set representation of the matching (linkage) attributes, then create signatures based on Locality Sensitive Hash scheme explained below. To prevent any correlation analysis, the signatures are lexicographically sorted and encrypted using the publishers' secret keys.

5.3.1 String Set Representation

In order to apply the LSH technique to get the signatures of the strings, we first need to convert the strings to sets of Q-grams (e.g. Bi-grams , for Q=2). In our implementation we extend the bi-gram set representation of our strings in such a way that it accounts for missing and flipped character positions. We add bi-grams constructed from tri-grams with the middle character being deleted. For example, the *the extended set representation* of the string **abcde** is constructed as follows:

- create the set of bi-grams B as $B = \{\_a, ab, bc, cd, de, e\_\}$
- create the set of Tri-grams T as $T = \{\_ab, abc, bcd, cde, de\_\}$
- From T create the set of extensions X as $X = \{\_b, ac, bd, ce, d\_\}$
- The Extended set E will be $E = B \cup X$

With this extension the possibility of matching strings with flipped character increases; for example the two strings **John** and **Jhon** will be considered a match with .75 Jaccard similarity using this technique. While the same strings without this extension will have

.25 Jaccard similarity. We represent dates as strings and append some characters before and after the month, day and year components to make the bi-grams generated form those components look different in case they have the same digits.

5.3.2 Choosing LSH parameters

For a similarity threshold St_0, and K minhash functions we set the values of b (number of groups/signatures) and r (number of minhash values in each group) such that for any two strings x,y that have $Sim(x, y) \geq St_0$, the probability of getting at least m out of b of their corresponding signatures to match is greater than 0.5. This is done as follows:

$m \simeq b \cdot St_0^r$ and $b = K/r$ so $r = K \cdot St_0^r/m$ or $r = ln(r \cdot m/K)/ln(St_0)$, solve for r then use $b = \lfloor K/r \rfloor$ to calculate b. All the values K,b, and r must be integers. The similarity threshold could also be approximated based on r and b as $St_0 \simeq (1/b)^{1/r}$

5.3.3 Creation of LSH Signatures

During the setup phase, all data sources (publishers) agree on the set of size t linkage fields $L = \{L_1, \cdots, L_t\}$ and their corresponding similarity computation parameters, which are K_i hash functions (or,it could be one hash function with K_i different seeds), and similarity threshold St_i for each linkage field $L_i \in L$. We would like to emphasize here that those linkage fields not necessarily a single database attribute, they might be concatenations of certain attributes, as it is the case in many record matching works. Then each party calculates b_i (number of groups/signatures) and r_i (number of minhash values in each group) for each record linkage field L_i as discussed in section 5.3.2 above. The steps for signatures creation are shown in figure 5.1 and following described now: We denote by $Sig_j^{(i)}$ the set of signatures of the column i of record j, and $ESig_j^{(i)}$ is its corresponding encrypted signatures. Each party A with records set $R_A = \{R_1, \cdots, R_n\}$ having linkage fields $L = \{L_1, \cdots, L_t\}$, creates the linkage field signatures for each record $R_j \in R_A$, $1 \leq j \leq n$ as follows:

- For each linkage field value $R_j.L_i \in R_j$ create its corresponding Encrypted signature $ESig_j^{(i)}$ as follows:

1. Convert the value $R_j.L_i \in R_j$ to an extended bi-grams set E
2. Apply K_i hash functions to the set E and get the K_i minhash values $minh = \{min(h_f(E)) : 1 \leq f \leq K_i\}$
3. Group the set $minh$ into b_i groups of r_i items each, and without changing their order, i.e. $G_x = \{minh[(x-1)*r_i+1]\| \cdots \|minh[x*r_i]\}, 1 \leq x \leq b_i$. Then, using a universal hash function H, hash each group into one value getting the set of b_i signatures of the value $R_j.L_i$ as: $Sig_j^{(i)} = \{H(G_g) : 1 \leq g \leq b_i\}$
4. The encrypted signatures of the value $R_j.L_i$ will be:

$$ESig_j^{(i)} = \{E_{sk_{A,L_i}}(Sig_j^{(i)}[g]||g||L_i) : 1 \leq g \leq b_i\}$$

, where sk_{A,L_i} is the secret key of party A for linkage attribute L_i. This secret key could be the same for all attributes.

- sort $ESig_j^{(i)}$ values in lexicographical order and append them as new field to the record.

To better explain how the protocol works, consider a database that has a set of linkage fields, and one of them is first name (fName). The encrypted signatures construction steps of this linkage field example is shown in figure 5.2. At the beginning the shared configuration information is used to compute the number of signatures and minhashes for each signature; then the *fName* field of all records is processed according to the steps shown until we get the encrypted signatures of each field.

5.3.4 Record Matching Phase

Each data source (publisher) will send the encrypted signatures of the linkage fields (columns) (as separate or combined) table(s) along with random record identifiers of its data set to the broker. To determine if record $A.R_k$ matches the record $B.R_j$, the broker executes the following steps:

- For each set of encrypted signatures of each linkage column ($L_i \in L$) of both records i.e. $A.R_k.ESig_k^{(i)}$, and $B.R_j.ESig_j^{(i)}$, apply the key-conversion as follows:

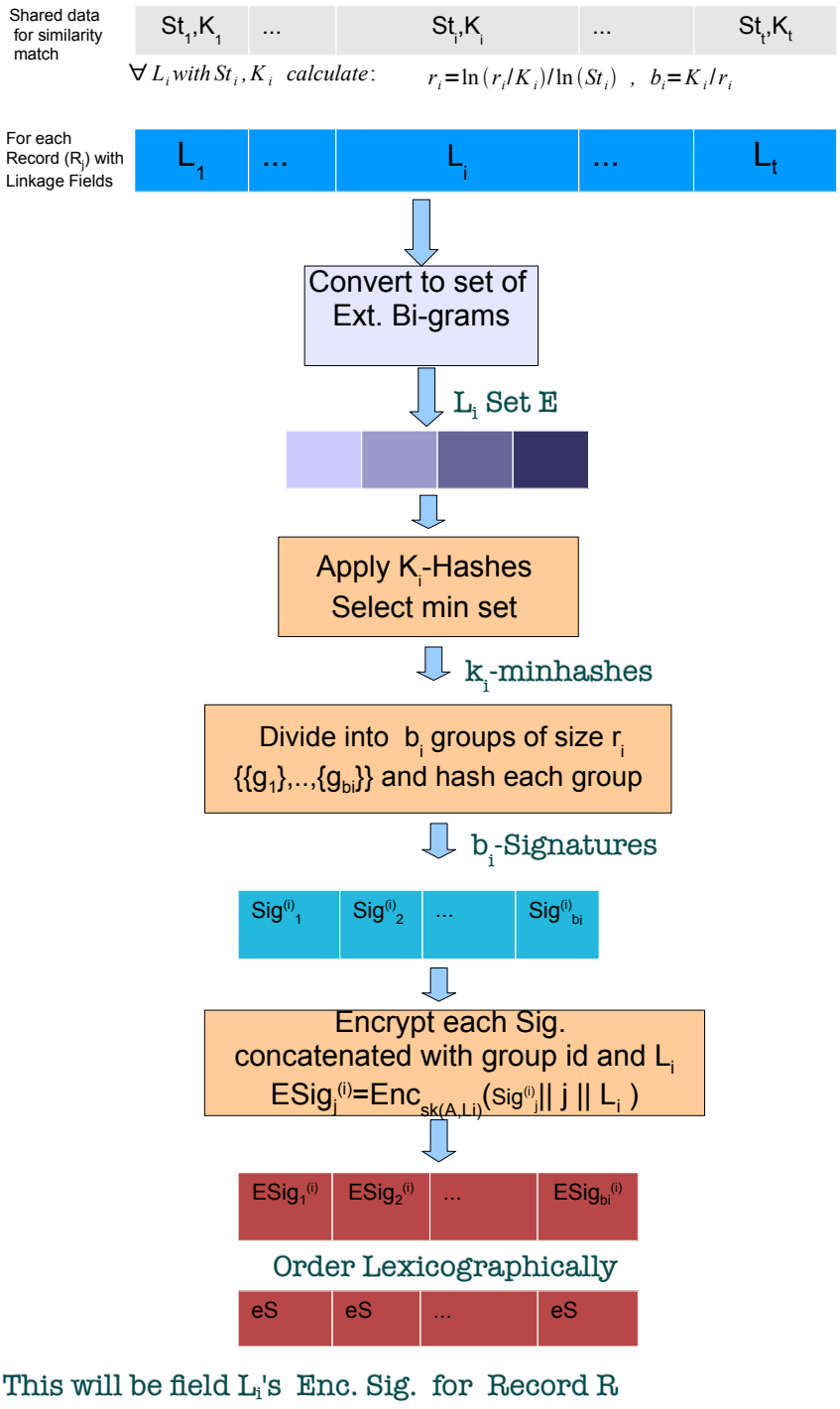


Figure 5.1: Signature Creation

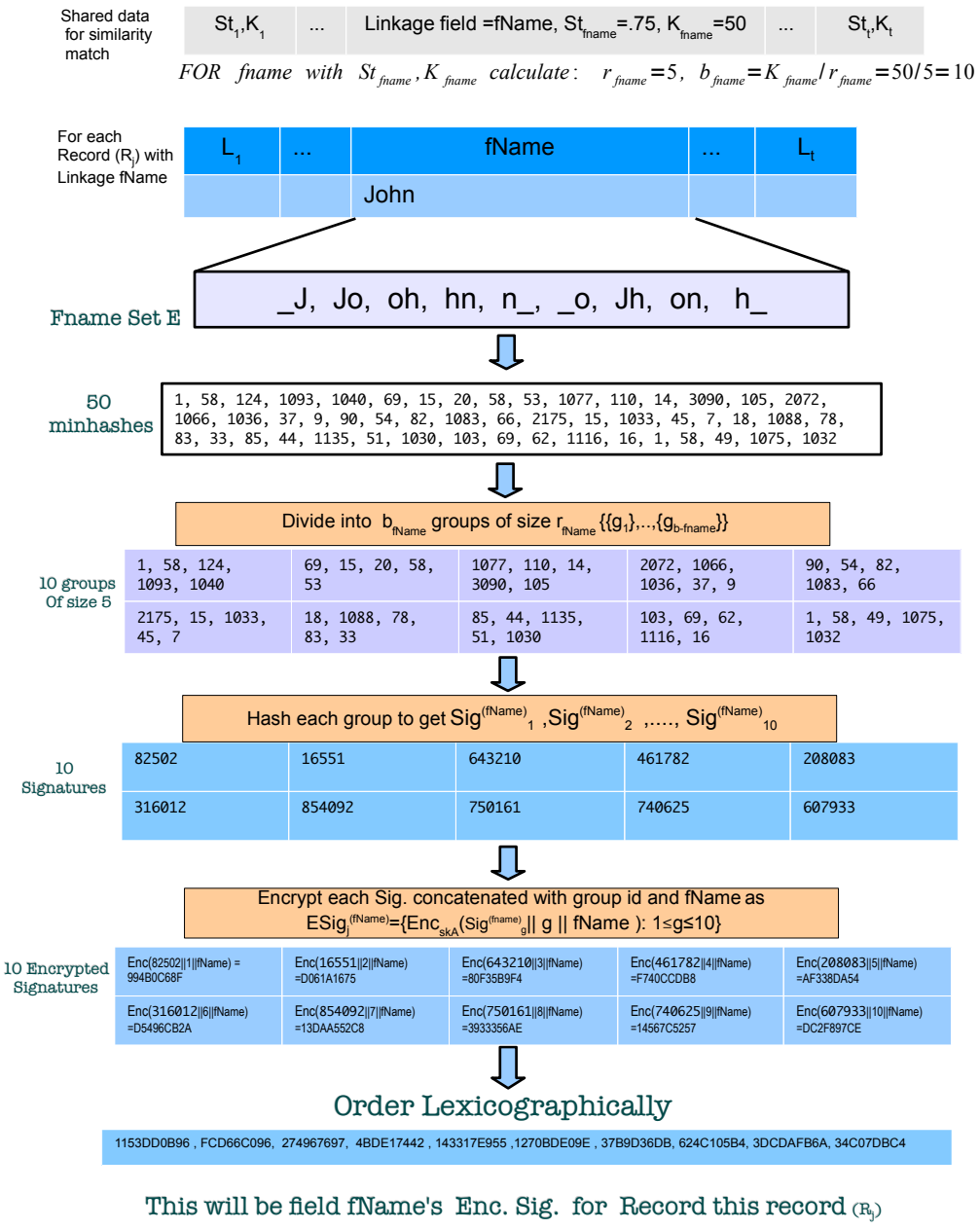


Figure 5.2: Signature Creation Example

Use the key converter of A (δ_A values) to convert the signatures of linkage fields $A.R_k.ESig_k^{(i)}$ of the record $A.R_k$ encrypted by publisher A under its secret key sk_A, to signatures encrypted under a different secret key Δ, i.e., $\mathsf{E}_\Delta(Sig_k^{(i)})$ for every linkage field L_i. The key $\Delta = \prod_{p=1}^{N} \mathsf{sk}_p$ is resulting from the product of all the secret keys of all parties (publishers). That is,

$$\mathsf{E}_{\delta_A}\left(ESig_k^{(i)}\right) = \mathsf{E}_{\delta_A}\left(\mathsf{E}_{\mathsf{sk}_A}(Sig_k^{(i)})\right) = \mathsf{E}_\Delta(Sig_k^{(i)})$$

The same applies to B, that is,

$$\mathsf{E}_{\delta_B}\left(ESig_j^{(i)}\right) = \mathsf{E}_{\delta_B}\left(\mathsf{E}_{\mathsf{sk}_B}(Sig_j^{(i)})\right) = \mathsf{E}_\Delta(Sig_j^{(i)})$$

- Compare the resulting signature sets encrypted under the same key Δ, $B.\mathsf{E}_\Delta(Sig_j^{(i)})$ and $A.\mathsf{E}_\Delta(Sig_k^{(i)})$ to find the number of equal signatures es_i and update the results table for this linkage field L_i. If the number of equal signatures is zero (0), then the similarity between these two values of this linkage field is less than the pre-set similarity threshold St_i, and hence the field match score $FM_i(R_j \cdot L_i, R_k \cdot L_i)$ will be zero. Otherwise, calculate the Estimated similarity of the two records for this linkage field L_i as $S_i \simeq (es_i/b_i)^{1/r_i}$, then calculate the field match score based on its weight w_i using the equation 4 shown in section 5.2. $FM_i(R_j \cdot L_i, R_k \cdot L_i) = w_i \cdot S_i$

- To make a record match decision $RM(R_j, R_k)$ and declare the records R_j and R_k as a match or not, first compute the overall record match score using the field match scores computed in the previous step.

$$RecScore = \sum_{1 \le i \le t} FM_i$$

  Then compare the record match score with the pre-set record match threshold TR as shown in equation 5 in section 5.2. If a match is found, save the matched record identifiers as a matched pair in the results table.

At the end, the broker will have an association (mapping) between the record random identifiers of the publishers based on the scores computed from the matched signatures of

the corresponding fields. For example, suppose we have two publishers A, and B that send the encrypted signatures of their records as tables with the following schema:

Publisher A: $A.col_1SigTable(ARecId, Col_1Sigs)$, $A.col_2SigTable(ARecId, Col_2Sigs)$, $\cdots$, $A.col_tSigTable(ARecId, Col_tSigs)$

Publisher B: $B.col_1SigTable(BRecId, Col_2Sigs)$, $B.col_2SigTable(BRecId, Col_2Sigs)$, $\cdots$, $B.col_tSigTable(BRecId, Col_tSigs)$

Then the Broker will find the matched records and create the mappings between A's record Ids and B's record Ids and save them as new table with a schema similar to this,

$matchingResultTable(ARecId, BRecid, Score)$.

5.4 Implementation and Results

To evaluate the performance and the accuracy of this protocol, we implemented it and tested it against some data sets of different sizes and using combinations of different linkage fields. We adopted the following two approaches, 1) using each single attribute as a linkage field, and 2) using concatenation of some attributes as a linkage field. We evaluated the accuracy and performance based on these two approaches.

We define accuracy in terms of *precision* and *recall* based on true positive (TP), false positive (FP), true negative (TN) and false negative(FN). **TP** (True Positive) is the number of originally-matching records that are correctly identified as match. **FP** (False Positivse) is the number of originally-non-matching records that are falsely identified as match. **TN** (True Negative) is the number of originally-non-matched records correctly identified as a non-match. Finally, **FN** (False Negative) is the number of originally-matched records falsely identified as a non-match. Based on these values, we compute the following:

1. **TPR** (True Positive Rate) Or Sensitivity/recall: Portion of originally matched records that correctly declared as match by the system: $TPR = \frac{TP}{TP+FN}$

2. **TNR** (True Negative Rate) Or Specificity: Portion of originally non-matched records that correctly declared as not-match by the system: $TNR = \frac{TN}{TN+FP}$

3. **PPV** (Positive Predictive Value) Or Precision: Portion of correctly matched records from all records declared as a match by the system: $PPV = \frac{TP}{TP+FP}$

4. **ACC** (Accuracy): Total Number of Correctly declared as a match and correctly declared as a not-match records out off all records: $ACC = \frac{TP+TN}{TP+TN+FP+FN}$

5. F_1 **score** (F-Measure) is the harmonic mean of precision (PPV) and recall (TPR), calculated as $F_1 score = 2 \times \frac{(PPV \times TPR)}{(PPV+TPR)}$.

5.4.1 Results of using realistic synthetic Dataset

Using two Datasets (one is the corrupted version of the other) generated by *Mocaroo* realistic data generator (www.Mocaroo.com), that consists of 10K records each, and 6K of the records are true matches. The record attributes (fields) used in this data were (ID,SSN,FirstName,LastName,email,DOB,Gender,ZipCode). We used two versions of the datasets, one is more corrupted than the other.

We opt to use realistic synthetic Datasets for many reasons, like known linkage results, shareable dataset for reproducibility (i.e., synthetic data don't require IRB approval), and controlled error/missing values rate.

We ran an extensive set of experiments with different configurations on both versions of the datasets (less corrupted and highly corrupted), and for convenience we will discuss only some of them here. After some experiments using signatures for single attributes, we picked similarity threshold for each attribute (low threshold to allow more permissiveness, and high threshold for strictness in the matching criteria).

1. **Using the Less corrupted Vs. the highly corrupted Datasets:**

   The more corrupted the data the lower the similarity between the records will be. So the similarity threshold of each linkage field that effect its signatures creation will effect the matching results as well. For more permissiveness the threshold should be low, and vice versa. However the lower threshold will effect the False positive rate, so it is better to keep false positive as low as possible.

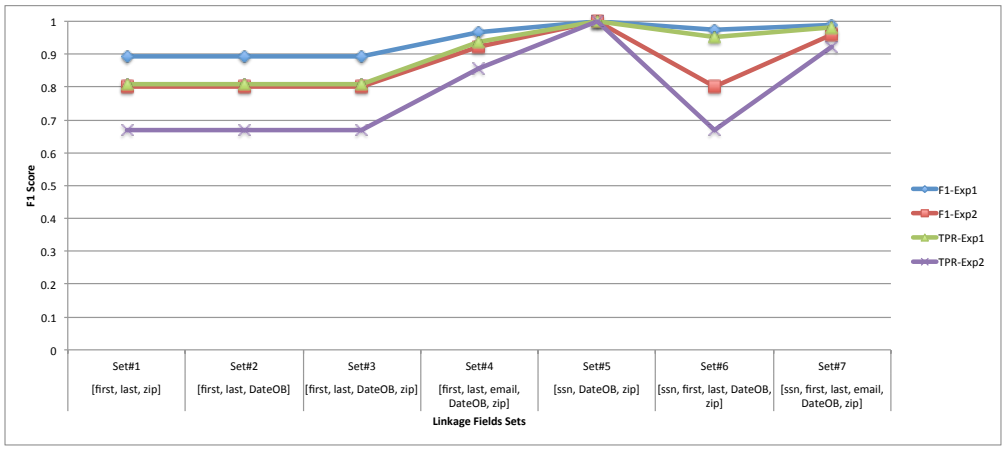


Figure 5.3: F1 score using low (Exp1) and high (Exp2) Similarity thresholds for single attributes, and different combinations

For example, the F1-score result of two experiments of low (ssn= 0.85, first= 0.65, last= 0.65, email= 0.75, DateOB= 0.85, zip= 0.85) and high (ssn= 0.85, first= 0.75, last= 0.75, email= 0.75, DateOB= 0.85, zip= 0.85) thresholds settings of seven different sets of combinations of the single attributes, and using the less corrupted data is shown in figure 5.3 , where the sets used are:

- Set#1: [first, last, zip]
- Set#2: [first, last, DateOB]
- Set#3: [first, last, DateOB, zip]
- Set#4: [first, last, email, DateOB, zip]
- Set#5: [ssn, DateOB, zip]
- Set#6: [ssn, first, last, DateOB, zip]
- Set#7: [ssn, first, last, email, DateOB, zip]

2. **Using Single attributes Vs. Concatenated attributes as Linkage Fields**

   Using logical expressions constructed from combinations of single attributes gives more flexibility to form matching criteria, and allows fine tuning of the matching threshold of each attribute separately. However it incurs more computation and storage overhead

and make the signatures more susceptible to frequency attacks on some attributes. On the other hand, using the concatenated values of certain attributes before creating the signatures will limit the matching criteria to the combinations of these concatenations and reduce the flexibility of matching conditions. However such concatenated attributes, if properly constructed, will reduce the computations required in both signature creation and matching process, and limit the frequency attacks.

In our experiments we used the first technique to evaluate the accuracy of the protocol, and since the signatures are encrypted from the source with different keys, the frequency analysis is only possible if conducted by the broker. If the publishers use new random with each signature, update their key-converter δ and include it with the signatures, they can limit the broker frequency attack to those matched attributes, though will add communication overhead.

Table 5.1 shows the results of using four different combinations of the set of attributes. The top part of the table for the combinations using single attributes signatures, each has its similarity threshold, the bottom part of the table for the concatenated attributes and the similarity threshold is set for the whole concatenated attributes of each combination. The combinations are as follows:

- comb1: First Name, Last Name, Date of birth
- comb2: Date of birth, SSN
- comb3: Last name, SSN
- comb4: Three Letters First Name, Three Letters Last Name, Soundex(First Name), Soundex (Last Name), Date of birth, SSN

Any two records matched by any of the above combinations is considered a match.

From the results in table 5.1, we were able to fine tune the threshold similarity of each attribute, using the single attribute technique, to control the true positives and false positive and get better results. It is much harder to do that in the second technique.

Table 5.1: Matching quality results of using single and concatenated combinations of attributes with different similarity thresholds settings

Using Signatures of Single attributes						
Sim. Threshold for (SSN,First,Last,DOB,Special1)	TP	FP	PPV	ACC	F1	Time (ms)
(0.85, 0.85, 0.85, 0.85, 0.85)	5161	14	0.997	0.915	0.924	384589
(0.85, 0.85, 0.85, 0.95, 0.85)	4916	2	1	0.891	0.901	261302
(0.85, 0.75, 0.75, 0.98, 0.85)	5022	0	1	0.902	0.911	498038
Using Signatures of Concatenated attributes						
Sim. Threshold for (comb1, comb2, comb3, comb4)	TP	FP	PPV	ACC	F1	Time (ms)
(0.95, 0.95, 0.95, 0.95)	5140	9	0.998	0.913	0.922	3895
(0.98, 0.988, 0.99, 0.99)	4710	0	1	0.871	0.880	2009
(0.98, 0.98, 0.98, 0.95)	4848	0	1	0.885	0.894	2364

The combination (Comb4) where constructed by creating a new attribute (we named "Special1") that consists of the concatenations of the first four parts of comb4 above (i.e. "Three Letters First Name" + "Three Letters Last Name" + "Soundex (First Name)" + "Soundex (Last Name)"), then create the signatures for it.

Finally, table 5.1 also shows the time used in the matching process using both techniques. It is evident how the second technique out performed the first, for the reasons mentioned above. In conclusion, our system achieved good matching quality results with good performance.

Chapter 6

Privacy Preserving Probabilistic Record Linkage Without Trusted Third Party

Most of PPRL work focusses on deterministic linkages where the identifying attributes of two records must be equal in order to declare them to belong to the same individual. Moreover, most of these methods require the active participation of a trusted third party (TTP). If this TTP is compromised, it makes the data from all participating parties vulnerable to information leakage. The proposed work improves upon the existing methods in two ways. First, we propose a protocol which does not require two records to have an exact match on identifying attributes in order to be declared as belonging to the same individual. Second, we investigate probabilistic PPRL in the two-party setting without resorting to any TTP.

In our method, linkages are determined based upon the approximate similarity between Bloom filters, measured by the Dice coefficient (DC). Yao's Garbled Circuits (GC) are used to compute the DC of every pair of records and compare it to a pre-set threshold in a secure manner. The parties do not have to exchange their Bloom filters, or even share the DC values. The output of the computation could be the value *"True"* if two records can be linked (that is, they belong to the same entity), or the value *"False"* otherwise. Since the Dice coefficient values may leak some information, we set a threshold value. We have a garbled circuit that compares the computed Dice coefficient to the threshold and it returns *"True"* if the computed value is greater than or equal to the threshold, hence reducing information leakage. Our protocol improves upon the approach of Schnell, et. al in [76] for record linkage by removing the TTP, and using optimized garbled circuit design to keep each party's data on its site to prevent Bloom filters cryptanalysis. To alleviate the computation and communication overhead of Yao's protocol, we leverage data blocking methods and

optimize the computation. We provide a security proof of our method and experimentally evaluate the performance gained on large benchmark datasets.

6.1 Contribution Summary

In this work, we proposed solution to the privacy preserving record linkage problem without TTP, and without sharing keys. Our solution is based on designing garbled circuits (GCs) to securely compute the Dice coefficient of the records Bloom filters, and compare it with a pre-set threshold. Garbled circuits are considered inefficient because they incur excessive amount of computations. We proposed a number of optimizations in order to alleviate the computation overhead in our garbled circuits. The first optimization is converting the real number operations into integer number operations. Integer operations require smaller circuits, and compute much faster. The second optimization is partitioning the Bloom filters, and process the partitions sequentially. We sequentially compute the DC contribution of each partition, and test if the required DC threshold will be met or not. The process of the following partitions is contingent on the DC contribution fulfilment of the previous partitions. The partitioning method helped in speeding up the process in two ways, 1) it made the computation on smaller circuits, and reduce the computation of partitions when the DC contributions are not met by the first partitions. The third optimization is using blocking, where the records of each party involved in the linkage are indexed based on some similar attributes, then the record linkage is performed on the corresponding block pairs. Our optimizations techniques improve the efficiency of the garbled circuit based record linkage methods. Blocking speeds up the process of large datasets, and BF partitioning speeds up the process within each block, and the integer computation reduced the garbled circuits sizes.

6.2 Probabilistic PPRL Protocol Overview

We have two datasets X = $\{x_1, x_2, \ldots, x_m\}$ and Y = $\{y_1, y_2, \ldots, y_n\}$ having PIIs stored on two servers located at two different institutions, A and B. A and B need to perform probabilistic PPRL on datasets X and Y in a secure manner without resorting to a TP. Our protocol consists of two steps: (i) Bloom filter generation and (ii) Matching records using Dice coefficient. We describe the details of each step below.

6.2.1 Bloom Filter Generation

A and B first agree upon k combinations of attributes that will be used for matching. Each of the two parties, A and B, constructs k Bloom filters denoted as $\{BF_1, BF_2, \cdots, BF_k\}$ for each record. Each Bloom filter represents one of k different combinations of the data fields included in the dataset. The combinations used in this paper were adopted from Kho et al. [52] although the method can be generalized for any combinations of linkage variables. We give the set of combinations that we use in this work.

- **[Comb 1]**: First Name + Last name + Date of Birth
- **[Comb 2]**: Date of Birth + SSN
- **[Comb 3]**: Last name + SSN
- **[Comb 4]**: Three Letter First Name + Three Letter Last Name + Soundex First Name + Soundex Last Name + Date of Birth + SSN

We followed the standard protocols to generate Bloom filter of each combination in each record [21, 76]. To ensure comparability, the Bloom filters from both parties A and B must be generated using the same mechanism and salts. We set the size of each Bloom filter to 1000-bit. The Bloom filter generation process, used by each party, is described below.

1. The value of each combination is created from the concatenation of the text values of the individual variables.
2. Tokenizing the text value of the combination into bi-grams.

3. Hashing clear-text bi-grams using a family of one-way hash functions with added random string salts.
4. Mapping the resulting hash values to a Bloom filter.

6.2.2 Matching using Dice Coefficient

Once each party has generated the k Bloom filters corresponding to the k combinations for each record, the next step is to compare them. This is done by computing the Dice coefficient (DC) for the corresponding Bloom filters obtained from the two parties, and checking if any

Suppose A and B have m and n records respectively. Let the records of A be denoted by x_i, where $i \in \{1, 2, \ldots, m\}$. Let records of B be denoted by y_j, where $j \in \{1, 2, \ldots, n\}$. We now describe the process of matching the pair of records (x_i, y_j). We begin by computing the DC for the corresponding Bloom filters. Recall that we have k Bloom filters corresponding to the k combination of matching attributes. Let $x_i \cdot BF_h$ and $y_j \cdot BF_h$ be the Bloom filters corresponding to combination h for x_i and y_j respectively where $h \in \{1, 2, \ldots, k\}$ and let $DC_h(x_i \cdot BF_h, y_j \cdot BF_h)$ be the corresponding DC. Two records are considered a match when $DC_h(x_i \cdot BF_h, y_j \cdot BF_h) > DC_t$ where DC_t is a given threshold. Formally,

$$\exists DC_h(x_i \cdot BF_h, y_j \cdot BF_h) \geq DC_t, 1 \leq h \leq k \Rightarrow x_i \; matches \; y_j$$

The Dice coefficient (DC_h) of two Bloom filters $x_i \cdot BF_h$ and $y_j \cdot BF_h$, belonging to records x_i, y_j respectively, is computed as follows:

$$DC_h(x_i \cdot BF_h, y_j \cdot BF_h) = \frac{2 \times |x_i \cdot BF_h \wedge y_j \cdot BF_h|}{|x_i \cdot BF_h| + |y_j \cdot BF_h|} \tag{6.1}$$

where $|.|$ is the number of 1's and $a \wedge b$ denotes bitwise-AND operation on Bloom filters a and b.

6.3 Computation Methods

6.3.1 Garbled Circuit to Compute and Compare DC

We use garbled circuits (GC) to compute the Dice coefficient (DC_h) of any two Bloom filters $x_i \cdot BF_h$, and $y_j \cdot BF_h$, then compare the result to the threshold DC_t. Algorithm 1 describes this process.

Algorithm 1 Computing/Comparing Dice Coefficient of Bloom Filters

Inputs:

- DC_t ▷ *Dice Coef. threshold to compare with*
- $x_i \cdot BF_h$ ▷ *party A's BF*
- $y_j \cdot BF_h$ ▷ *party B's BF*

The Bloom filters are represented as Boolean vectors.

Outputs: **True** if $DC(x_i \cdot BF_h, y_i \cdot BF_h) \geq DC_t$, else **False**

Steps:

1- $ti = \text{int}(DC_t \times 128)$ ▷ *Convert DC_t to integer out of 128 instead of real number*

2-Compute $\mathcal{N} = |x_i \cdot BF_h \wedge y_j \cdot BF_h| \ll 2$ ▷ *to compute the numerator of equation 6.1*

▷ Where $\wedge$ is bitwise-AND, $\ll$ is left-shift , $|.|$ is #1s

3-$\mathcal{D} = |x_i \cdot BF_h| + |y_j \cdot BF_h|$ denominator of equation 6.1 is computed as follows:

Compute $\mathcal{D} = |x_i \cdot BF_h \vee y_j \cdot BF_h| + \mathcal{N}$ ▷ Where $\vee$ is bitwise-OR, $|.|$ is #1s

4-Compute cmp $= ((2 \times \mathcal{N}) \ll 7) \geq (\mathcal{D} \times ti)$ ▷ *cmp will get the value True or False*

return cmp

A garbled circuit can be constructed, such as the one shown in Algorithm 1, to return the computed DC to both parties. However, the DC value could reveal information about the compared records. For example when DC is close to 1, the parties can infer the matching attributes of the other party. When some record has small DC values that matches with all other records, it can be singled out as a record with less frequent attributes.

An alternative approach is to compare the computed DC to a pre-set matching dice coefficient threshold DC_t and return *"True"* or *"False"* to represent the match or non-match decision. If a threshold is used to filter out results, then a *comparator* circuit is needed to compare the floating point values of the threshold and the DC (see Step 4 in the algorithm). To improve efficiency, we used integers instead of floating point representations and achieve this comparison without any loss of accuracy. The constant threshold DC_t is represented as fraction of 128 instead of a fraction of 100 (to use left shift 7 times instead of multiplication);

that is, the new numerator of the new fraction is the integer value $ti = \text{int}(DC_t \times 128)$ and the denominator is 128. The *comparator* circuit cmp is as follows:

$$\text{cmp} = ((2 \times \mathcal{N}) \ll 7) \geq (\mathcal{D} \times ti) \tag{6.2}$$

where the comparator circuit will return True if the DC is greater than or equal to the threshold. Figure 6.1 shows the DC-computation and comparison with threshold in the form of GC, generated by party A. We assume that Party A is the generator and Party B is the evaluator.

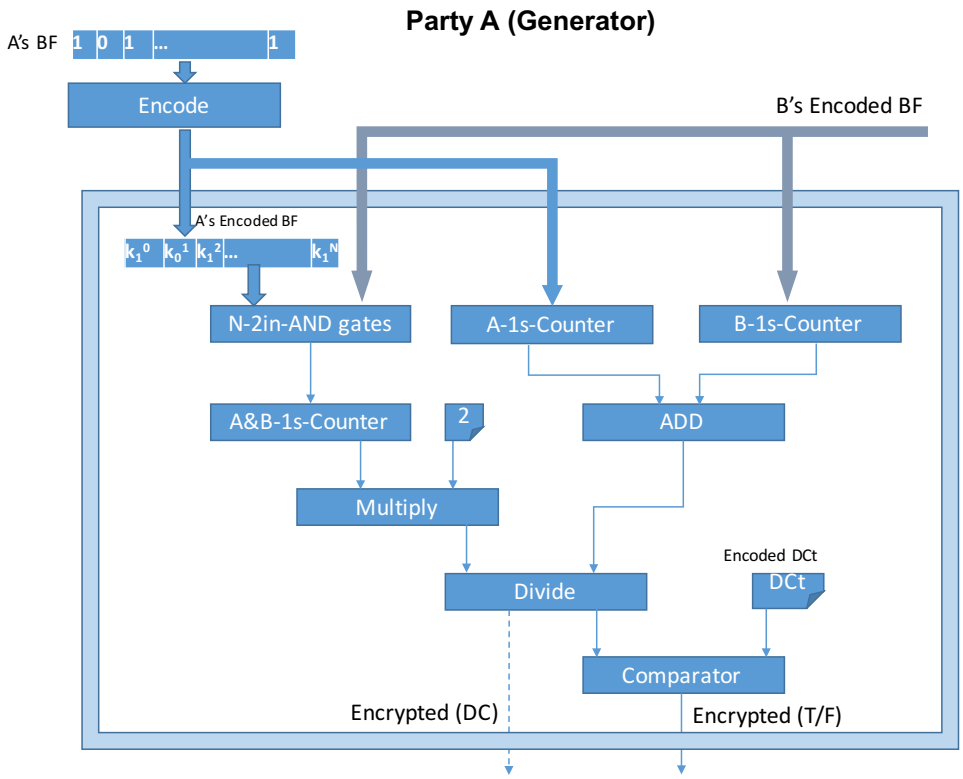


Figure 6.1: Garbled circuit that computes DC of two Bloom filters and compares it with a threshold

For each pair of Bloom filters a, b that the two parties wants to compare they need a garbled circuit and oblivious transfer (OT). The process is shown in Figure 6.2 and described below.

1. Party A generates the GC and maps (encodes) each bit $\mathsf{a}_i \in \{0,1\}, i = 0 \ldots N$, of its Bloom filter a to a distinct encryption key $k_a^i \in_R \{0,1\}^\lambda$, where N is the size of the Bloom filter and λ is the security parameter (the length of encryption keys).
2. Party A sends the GC and his encoded Bloom filter to party B.
3. Party B obtains the mappings of its Bloom filter b's bits to their keys from party A, without disclosing the bits of b to party A via the OT protocol. For each bit $\mathsf{b}_i \in \{0,1\}, i = 0 \ldots N$, party B receives k_b^i (the key corresponding to its bit b_i) from A.
4. Party B evaluates the GC using the keys of its encoded Bloom filter (obtained in step 3) as its input to the GC and obtains an encrypted result from the GC evaluation.
5. Party B sends the obtained result to Party A for decryption.
6. Party A decrypts the result and sends it to Party B, so both can output the result of the matching process.
7. Parties A and B need to repeat these steps for every pair of Bloom filters that they have created.

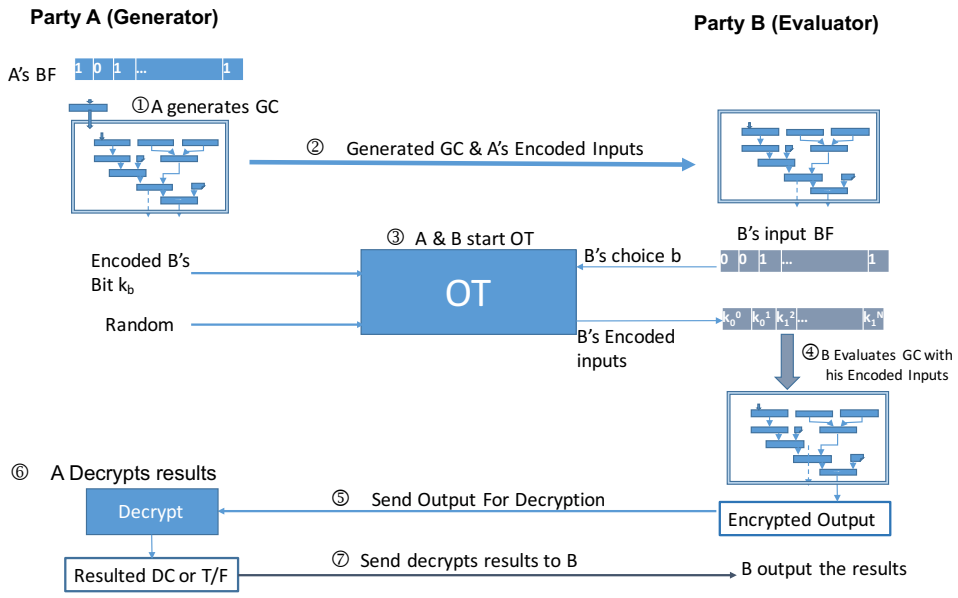


Figure 6.2: Dice coefficient computation using garbled circuit and oblivious transfer

6.3.2 Optimizing the computation

At the end of the record linkage process, both parties' output will be a list of the corresponding record indexes that satisfy the matching DC threshold. The two parties will be able to map those indexes to their respective record IDs maintained at each end.

Despite the improvements of the GC constructions [69], the comprehensive (n-to-m) linkage process using GC is still inefficient for matching large datasets. Since we have k encoded Bloom filters for each record, the matching process computes k DCs for each pair of records in the processed datasets. To declare any pair of records a match, at least one of their DCs must be greater than or equal to a pre-set threshold DC_t. For two datasets X and Y of sizes m and n respectively, and each having k Bloom filters, the matching process needs $m \times n \times k$ DC computations. DC calculations using GC and oblivious transfer operations take significant amount of time. We now propose two heuristic approaches to reduce the problem size and enable more efficient computation.

6.3.2.1 Improvement using blocking

In record linkage, blocking is the process of grouping records with certain characteristics together. With blocking the matching process is limited to the use of those groups that share the same characteristics, and hence reduces the overhead of GC computation and OT operations.

Any secure blocking scheme that provides data protection compliant with differential privacy, such as the one mentioned in [39], could be used as a preprocessing step with our scheme. Blocking restricts the comparisons to the blocks having similar linkage variable values, and ends up with $B \times s^2$ comparisons, where B is the number of matching blocks and s is the average number of records in each block.

6.3.2.2 Improvement using partitioned Bloom filters

In this step, each party partitions his BF into π smaller parts. Then, based on the number of ones in each part, compute its contribution to the overall DC. As the computation

progresses we determine sequentially if it is worth continuing the computation for the other parts. In other words, our GC securely computes the contribution C_i of both parties' partitions $P_i, 1 \leq i \leq \pi$ sequentially and examine whether it fulfills its contribution towards the pre-set threshold DC_t. If the partitions passed the contribution fulfilment test both parties proceed to the next partition; otherwise, they stop and do not process the remaining parts.

Let us denote the two BFs for which parties A and B are calculating their DCs as a and b, and their parts, $a = \{a_1||a_2||\ldots||a_\pi\}$ and $b = \{b_1||b_2||\ldots||b_\pi\}$, where $||$ is the concatenation operator. In such a case, the contribution C_i of the corresponding partitions $(a_i, b_i), 1 \leq i \leq \pi$, to the overall DC_t is computed by the formula given below. Here $|a_i \wedge b_i|$ denotes the number of 1's in the intersection of partitions a_i and b_i computed using bitwise-AND operation. $|a|$ and $|b|$ denote the number of 1's in the Bloom filter a and b respectively.

$$C_i = 2 \times \frac{|a_i \wedge b_i|}{|a|+|b|}$$

The sum of all contributions must be greater than or equal to the pre-set threshold of Dice coefficient DC_t in order to consider the two records a match by these two BFs. That is,

$$\sum_{i=1}^{\pi} C_i \geq DC_t$$

The contribution C_i depends on the number of ones in both partitions and proportional to the size of their intersection. Note that,

$$(|a_i \wedge b_i| \leq min(|a_i|, |b_i|)$$

Each party only knows its Bloom filter but not that of the other party. Consequently, when computing the contribution C_p, $1 \leq p \leq \pi$, each party has to estimate the value of $|a_j \wedge b_j|$ for the remaining partitions $p+1 \leq j \leq \pi$. Let us denote this estimate as $\widehat{|a_j \wedge b_j|}$. Then, the following constraints hold: $\widehat{|a_j \wedge b_j|} \leq max(|a_j|, |b_j|)$ and $|a_j \wedge b_j| \leq \widehat{|a_j \wedge b_j|}$

Thus, party A and B can substitute $\widehat{|a_j| \wedge |b_j|}$ with $|a_j|$ and $|b_j|$ respectively.

If the contributions of the partitions are calculated in sequence (in order $1 \leq i \leq \pi$), then the computation could be stopped at any point p if the sum of the minimum contributions

needed from these processed p partitions is not larger than DC_t minus the summation of the estimated maximum contributions C_j's of the remaining partitions $p+1 \leq j \leq \pi$:

$$\sum_{i=1}^{p} C_i \geq (DC_t \ - \sum_{j=p+1}^{\pi} \text{est.max}(C_j))$$

where

$$\begin{aligned} \text{est.max}(C_j) &= \frac{|\widehat{a_j \wedge b_j}|}{|a|+|b|} \\ &= \frac{|a_j|}{|a|+|b|} \text{ by party A, or } \frac{|b_j|}{|a|+|b|} \text{by party B} \end{aligned}$$

6.3.2.3 Performance analysis

The overhead due to GC depends directly on the input size and the number of operations for calculating the computed function. Partitioning the inputs into smaller parts will result in smaller sub-circuits. Note that, when we partition the inputs, some subsequent sub-circuits may not be evaluated depending on the value of the preceding sub-circuits. This will reduce the total overhead.

We assume (without loss of generality) that the two datasets under consideration are each of size n, and the number of Bloom filters in each record of these datasets is $k = 1$. Let t denote the average time to compute DC of any pair of Bloom filters. Thus, without any optimization (non-blocked, non-partitioned) time needed for DC computation is $n \times n \times t$. Let π denote the number of partitions. Thus, the average time for computation of each partition is t/π. Let P_i denote the i^{th} partition and r_i the size of the remaining computation after processing partition P_i (that is, the output size). Let $\alpha_i = \frac{r_i}{r_{i-1}}$ be the reduction factor, that is, the ratio of output size r_i to the input size r_{i-1}. $\alpha_0 = 1, r_0 = n^2$. We need these for determining those Bloom filter pairs that will not satisfy the DC threshold at the stage of computation of partition P_i, $1 \leq i \leq \pi$. The partial DC computation time t_i needed by Partition i is as follows:

1. **[For Partition P_1:]** $t_1 = \alpha_0 \cdot \frac{t}{\pi} \cdot n^2 = \frac{t}{\pi} \cdot n^2$

2. [**For Partition P_2:**] $t_2 = \frac{t}{\pi} \cdot \alpha_0 \cdot \alpha_1 \cdot n^2 = \frac{t}{\pi} \cdot \frac{r_1}{r_0} \cdot n^2 = \frac{t}{\pi} \cdot r_1$
3. [**For Partition P_i:**] $t_i = \frac{t}{\pi} \cdot \prod_{i=0}^{\pi-1}(\alpha_i) \cdot n^2 = \frac{t}{\pi} \cdot r_{i-1}$
4. [**Total Time:**] $T = \sum_{i=1}^{\pi} t_i = \frac{t}{\pi} \cdot (n^2 + r_1 + r_2 + \ldots + r_{\pi-1})$

The total time depends on the size of the Bloom Filters, the number of partitions (which depends on the size of each partition), and the size of the intermediate results of each partition computations r_i (which is the input to the computation of the next partition and depends on the matching threshold). The results from experiments in the next section show that a higher matching threshold leads to more reductions in the intermediate results. In the worst case scenario (zero reduction), the computation is the same as the original (non-partitioned) case. Consequently, picking the number of partitions together with the suitable threshold will affect the intermediate results and hence total performance.

6.4 Experimentation and Results

6.4.1 Implementation

We leveraged the Java code base developed at the UCSD [13] to implement our probabilistic garbled circuit. Our additions to the existing UCSD code base include the implementation of 1) Dice coefficient computation, 2) partitioned Bloom filters, and 3) blocking schemes.

For our experiments we ran the two parties (generator and evaluator) Java codes on a local network with two machines equipped with 4GB RAM, and Intel i5 processor. To verify the scalability of the system, we ran a distributed version of the system on a set of machines and reported the results for comparison.

The datasets were generated using the Mockaroo synthetic data generator (`http://www.mockaroo.com`), which uses real names and addresses. The data fields included in both datasets are First name, Last name, Social Security Number (SSN), Date of birth and Identification (ID). The data in each field in these datasets were randomly corrupted to emulate basic typographical errors (i.e., insertion, deletion, substitution and transposition). For each field, the corruption rate ranges from 10% to 20%. In order to validate the results,

we use an identification number (ID) for each record such that all records that belong to the same individual have the same identification number. Each dataset contains 10K records and both datasets have 6K records in common. A record which is in both datasets has the same value in the ID field. Therefore, the values of the ID field are used as the true answer to verify linkage performance.

6.4.2 Evaluation Metrics

We use the two metrics given below to evaluate our methodology.

6.4.2.1 Accuracy

The accuracy of the matching process is measured by the True Positive Rate TPR or *recall* value which is defined by

$$recall = \frac{TP}{TP + FN}$$

where TP is the True Positive which is the number of records correctly identified as a match, and FN is False Negatives which is the number of records incorrectly declared as a non-match. To show the advantage of using the probabilistic PPRL over the deterministic PPRL, we compared probabilistic PPRL using Bloom filters, with the deterministic PPRL using the hash of the data.

6.4.2.2 Runtime Improvement:

We performed tests to demonstrate the runtime improvement by blocking and partitioning methods:

1. *Blocking Without BF-partitioning*: We used blocking to divide the datasets into blocks and reduce the computations needed to link different sizes datasets. We set this test as our baseline to compare with the optimized version where the BFs are partitioned.
2. *Blocking with BF-partitioning*: To demonstrate the impact of our BF partitioning and computation reduction technique on both efficiency and accuracy, we ran the same previous baseline test, using our BF partitioning technique, and compared the results.

3. *Effect of block size*: To understand the effect of block size on both basic and partitioned GC-RL, we tested linking non-blocked datasets with different sizes (from 5 records to 1k records).
4. *Effect of BF Partitioning*: To further understand the performance gain and reduced computations of our BF partitioning technique, we ran analysis tests on one dataset and measured execution time and the number of partitions processed during each phase of computation.
5. *Scalability of our approach*: to test the scalability of our design, we tested a distributed version of it and reported the results.

6.4.3 Results

In each test, two datasets with the same number of records are linked. Each test is run 3 times and the average runtime is recorded. Each record of the datasets consists of 4 Bloom filters, constructed using the combinations explained in section 6.2, and each test computes 4 DCs of the corresponding BFs of each pair of records. A match is declared if any DC exceeds DC_t =0.9 threshold value.

6.4.3.1 Basic-GC-RL with Blocking Only (Baseline Case)

Performing GC-RL directly on large datasets is not practical. For example, using GC-RL to link just 1K $\times$ 1K non-blocked datasets took around 261,893.426 seconds (72.75 hours). Thus, like most other record linkage algorithm we divide the datasets into smaller blocks.

Blocking significantly reduces matching time because each record will be compared to only records in the corresponding block rather than to all records in a dataset. However, blocking may affect the accuracy of the matching process if the "should-have-been-linked" records are grouped into different blocks. Using test datasets with different sizes allows us to better understand the performance and scalability of our optimization schemes. Table 6.1 shows the performance of our basic-GC-RL when blocking is used to link four different sizes dataset-pairs.

Table 6.1: performance of the Baseline (Basic-GC-RL) Vs the optimized (Partition-GC-RL) and Using Blocking

Data Size/ Actual Matches	Accuracy				Basic-GC-RL (No partitions)	Partition-GCRL(4parts)
	TP	FP	FN	recall	Time/Sec	Time/Sec
1k x 1k / 600	596	0	4	0.99	1773	3429
Deterministic Match	460	0	140	0.77		
2k x 2k / 1200	1189	0	11	0.99	5613	4694
Deterministic Match	911	0	289	0.75		
5k x 5k / 3000	2976	2	24	0.99	25816	11066
Deterministic Match	2308	0	692	0.77		
10k x 10k / 6000	5948	16	52	0.99	89656	30283
Deterministic Match	4608	0	1392	0.77		

6.4.3.2 Partition-GC-RL With Blocking

In this experiment, we link the same datasets used in the baseline performance test (Blocking Basic-GC-RL) using the optimized version with BF partitioning method (Blocking Partition-GC-RL). Table 6.1 shows comparison between the performance of the two methods, and figure 6.3 shows that the partition-GCRL outperform the Basic-GCRL when the dataset size increases. The performance gain becomes evident when using the Partition-GC-RL with large datasets, as shown in figure 6.4.

6.4.3.3 Effect of block size on optimized GC-RL

We tested the non-blocked version of both the Basic-GC-RL and the Partition-GC-RL on 10 datasets with sizes ranging from 5 records to 1K records each, and recorded their execution times and accuracy. For datasets with larger sizes, the runtime can be extrapolated based on smaller sets. From the results in Table 6.2, the runtime of Basic-GC-RL increases rapidly with the size of the datasets, which makes the basic version of the GC-RL method to be very inefficient and impractical in real-world datasets.

With very small size blocks (less than 10 records), Basic-GC-RL runs faster than Partition-GC-RL, because the partitioning overhead exceeds the benefits. As the size of the block increases, the advantage of Partition-GC-RL method over the Basic-GC-RL method is more

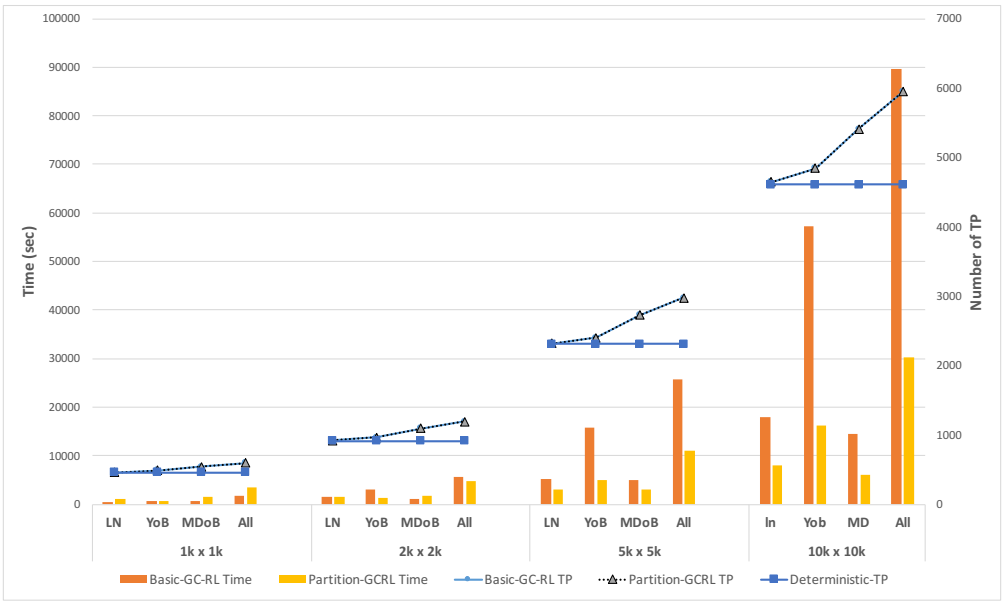


Figure 6.3: Performance and accuracy of the Basic-GCRL and the Partition-GCRL schemes (for Dataset sizes 1K – 10K)

evident with speedup factor of 9x, as shown in figure 6.5. For all sizes of the datasets in this experiment, the linkage accuracy of the Basic-GC-RL and the Partition-GC-RL is the same. However, even with partitioning, if the size of the dataset is 1K or more, GC is not efficient without blocking.

6.4.3.4 Partition-GC-RL Performance Gain

To further explain the reason for the Partition-GC-RL performance gain, Table 6.3 shows a detailed dissection of the times taken by each partition for linking two datasets of 1K each using only one Bloom filter. Size-In (n = 1M) of BF Partition 1 represents the initial number of record pairs to be linked. After comparing the first partition, Size-Out (n = 901,394) of BF Partition 1 represents number of record pairs which have sufficient evidence for their BF Partition 2 to be considered. This means that a number (n = 98,606) pairs were eliminated as a result of Partition 1 comparison. The process moves on to all four partitions. For the last partition, the number of record pairs was reduced to 1,016.

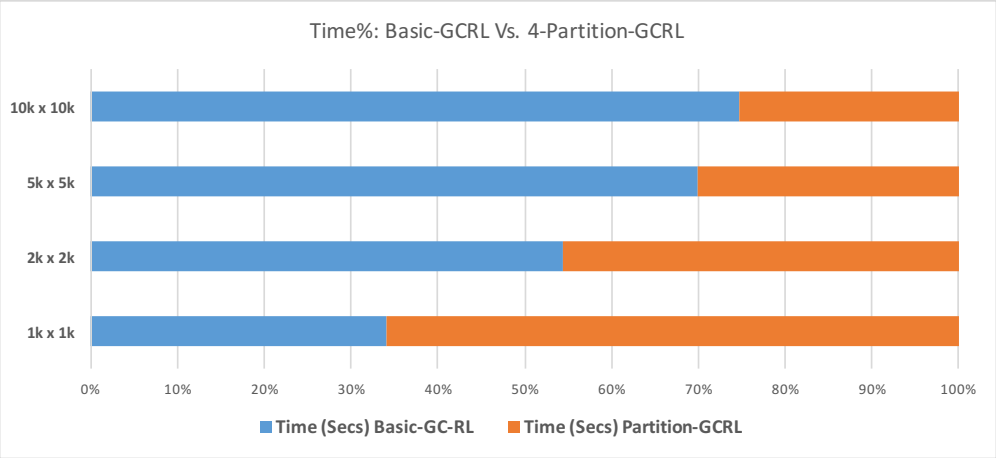


Figure 6.4: Percentage of time taken by the Basic-GCRL and the Partition-GCRL schemes (for Dataset sizes 1K – 10K)

Table 6.2: Block size effect on the Basic-GC-RL and Partition-GC-RL

# Recs AxB	Basic-GC-RL Time (sec)	Partition-GC-RL Time(sec)	Speedup %
5x5	5.905	8.357	-41.52
10x10	12.176	10.946	10.10
20x20	24.187	17.477	27.74
30x30	75.913	28.826	62.03
40x40	91.935	35.539	61.34
50x50	157.375	53.973	65.70
100x100	614.953	191.717	68.82
200x200	2185.048	647.0377	70.39
400x400	8683.934	2431.429	72.00
1kx1k	261893.426	27987.46	89.31

6.4.3.5 Scalability of our scheme

Since the blocking method allow the process of linking similar pair of blocks separately from the rest of other blocks, it is possible to distribute the blocks over multiple nodes to link them simultaneously. For this purpose we build a distributed version of the scheme and test its performance with the 10k datasets. We used three machines on each side with the following configurations (HP-Z800-XeonE5645-SAS, CPU 12 x 2.4G, RAM 96Gb, OS Linux-Fedora), each machine runs 7 nodes, i.e 21 nodes in total on each side. The total time taken to link the 10k $\times$ 10k datasets, was 2199 seconds (about 36 min), while the non-distributed garbled circuit took about 30283 seconds (8.5 hours), see table 6.1, to link the same datasets.

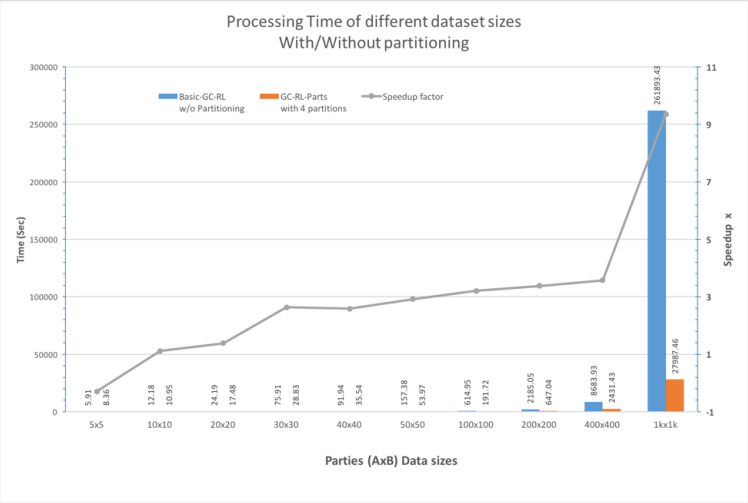


Figure 6.5: Partition-GC-RL performance gain

Table 6.3: Average number of computations and time for each partition (1-BF, 1K X 1K)

BF Partition	Computations		Time (secs)
	Size In	Size Out	
1	1,000,000	901,394	4,437.69
2	901,394	44,537	2,517.50
3	44,537	1,016	29.79
4	1,016	941	11.89
Total time for linking 1K X 1K datasets using 1 BF			6,996.86

6.5 Analysis and Observations

6.5.1 Privacy analysis

We now address how the parties using the scheme could infer information about the non-matched Bloom filters. First we will analyze the Basic-GCRL scheme, and prove it achieves the maximum level of privacy for record linkage in the semi-honest model. Then we will discuss the partition-GCRL scheme, and analyze the leakage from knowing matching results of some partitions of the BFs, and how our scheme could minimize this leakage in order to guarantee the desired level of privacy.

6.5.1.1 Bloom filter analysis

The false positive rate $f_p = \left(1 - e^{\frac{-kn}{m}}\right)^k$, and is controlled by, m the size of the Bloom filter, n the number of items to be inserted in the filter, and k the number of hash functions used to hash the items into the filter indexes. In order to minimize the f_p, we need to set k and m such that $k = \frac{m}{n} \cdot ln2$ [7]. In our RL application, a string of characters representing the values of record attributes is converted into bi-grams and then inserted into the record BF. Given that we have 26 letters, 10 digits and the padding character, the number of all possible bi-grams $|\Sigma| = 37^2$. This means if we set $n = |\Sigma|$, and fix k, then the size of the BF that minimizes f_p will be $\frac{37^2 \times k}{ln2}$, which is very large. For example, many RL applications use $k = 15$ hash functions, which means the ideal BF size will be about 59,000-bits, and it will not be efficient. However, in RL, small Bloom filters are created from a set of bi-grams of short strings (one or more attributes values of a single record), and this set of bi-grams is a small subset of Σ, so if the above formula is used to set m, then most of the cells of the BF will be 0's. On the other hand, if we use small m (e.g 1000-bits) we expect to have very large number of false positives. However this will not be always the case when the length of the strings is small, and the number k of hash functions is set such that no two different bi-grams could be hashed to the similar BF-indexes by all of the k functions. There are $\binom{m}{k}$ possible different patterns of 0s and 1s that could be produced by setting k-1s in m-bits. The record bloom filter is the aggregation (bit-OR) of these patterns. This means that, the larger k value, the less possible the bi-grams collide, however more 1's will be set in the BF, and this will make f_p increase. In addition, limiting the size of the BF m to small value will make some of the bits set multiple times by different bi-grams (since we have less locations than the number of items). From privacy perspective, this one-bit to many-bi-grams assignment makes it difficult to exactly map back this bit to the bi-gram that sets it. That is, given a bit location of the BF that is set as 1, the probability to infer the bi-gram that sets it, without knowing the other bits, will be small.

In our partition-GCRL scheme, no individual bit location is revealed, however the partitioning could allow the adversary to exclude some of the bi-grams if the partition size doesn't

cover all bits that are possibly set by all bi-grams. The ideal case is to have each partition to include at least one bit that could be set by every bi-gram. If the indexes generated by the k hash functions is uniformly distributed across the BF, then the minimum partition size that could cover all bi-grams will be m/k. This means, that we can have at most k partitions in order to have each partition to cover all possible bi-grams. In our experiments we set the number of partitions β much smaller than k to guarantee the partition coverage of all possible bi-grams.

6.5.1.2 Scheme Leakage analysis

We want some guarantee that none of the parties should learn more than what the others are willing to share. Any extra information learned by any of the parties is considered a leakage, and this leakage could violate the privacy. To quantify this leakage, we define a leakage function that captures the extra-information learned by the parties from the output and the intermediate messages they receive during the RL protocol execution. While it is highly desirable (higher privacy) to prevent any leakage, the design of such scheme that is efficient and practical will be more complicated and challenging. Thus, it may be acceptable to have some well studied leakages.

Definition 6. *response from the partition-GCRL* $\mathcal{R}_{\pi_{part}}(BF^i, BF^j)$*:*
If we denote the result of DC contribution fulfilment test of the partitions $BF^i.p, BF^j.p$ *of the two Bloom filters* BF^i, BF^j *of the parties* i *and* j *as the pair* (p, r) *where* $r \in \{T, F\}$ *represents the fulfilment test result as* T *(True), and* F *(False), for the partition* $1 \leq p \leq \beta$*, then for the* β *partitions we define the response* $\mathcal{R}$ *from the partition-GCRL scheme* π_{part} *denoted* $\mathcal{R}_{\pi_{part}}(BF^i, BF^j)$ *as* $\{(1, T), \cdots, (u-1, T), (u, F)\}$*, where* $1 \leq u \leq \beta$*. That is, all the partitions numbers until a contribution fulfilment fails.*

Definition 7. *non-matched partition leakage function* $\mathcal{L}_p(\beta)$*:*
Given the response $\mathcal{R}_{\pi_{part}}(BF^i, BF^j)$ *during the DC computation of the two* β*-partitioned Bloom filters* BF^i, BF^j *of parties* i*,and* j *using the partition-GCRL scheme* π_{part}*, we define the non-matched partition leakage function as* $\mathcal{L}_p(\beta) = |\mathcal{R}_{\pi_{part}}(BF^i, BF^j)|$*. That is the index*

of the partition that failed the DC contribution test and caused the scheme to stop processing the remaining partitions.

Form the above leakage definition we notice that, as the index increases, the leakage increases as well because more information from the matched partitions is revealed (the adversary will know how these matched partitions will look like based on its partitions), and as the number of remaining unknown partitions is getting smaller the probability of guessing the contents of these remaining partitions increases, i.e the uncertainty about the remaining bits of the Bloom filter is less.

Definition 8. *ρ-privacy preserving RL scheme:*

For RL scheme using Bloom filters of size n, partitioned into β partitions, we say that the scheme is ρ-privacy preserving scheme if the probability of guessing the Bloom filter bits pattern is upper bounded by $\frac{1}{p^\rho}$, where $p = \binom{n/\beta}{n/2\beta}^\beta$, $0 < \rho \leq 1$, assuming half of the BF bits are set as 1s. The ρ-privacy preserving RL scheme sets the minimum level of privacy guarantee achieved by the scheme.

Theorem 1. *The partition-GCRL scheme π_{part} is $1/\beta$-privacy preserving RL scheme for the none matched records.*

Proof 1. *In our partition-GCRL scheme π_{part}, if the Bloom filter's size is n, the number of partitions is β, and the leakage function is $\mathcal{L}_p(\beta)$, and for $p = \binom{n/\beta}{n/2\beta}^\beta$ we have*

- *the number of remaining partitions that might be different after receiving the final result as "F" will be $\gamma = \beta - (\mathcal{L}_p(\beta) - 1)$*
- *the minimum number of all possible patterns $p_{min} = \binom{n/\beta}{n/2\beta}^\gamma$, assuming half of the bits are set as 1s.*
- *the maximum leakage value for the leakage function will occur when the non-matched partition is the last one, i.e $l_{max} = \beta$, so $\gamma = \beta - (\beta - 1) = 1$,*

- $\overline{p}_{min} = \binom{n/\beta}{n/2\beta}^1 = p^{1/\beta}$
- *the probability of guessing the remaining bits of the Bloom filter will be at most* $\frac{1}{\overline{p}_{min}} = \frac{1}{p^{1/\beta}} = \frac{1}{p^{\rho}}$ *for* $\rho = 1/\beta$

Therefore π_{part} *is* $1/\beta$*-privacy preserving RL scheme.*

Corollary 1. *Our Basic-GCRL scheme* π_{Basic} *is 1-privacy preserving scheme.*

Proof 2. *Since the Basic-GCRL scheme is a special case of the partition-GCRL scheme, then the same proof applies, and by setting* $\beta = 1$ *we get* $\rho = 1/\beta = 1$.

Note that when $\beta = 1$ (no partitions) we get full privacy preserving scheme with $\rho = 1$, and the privacy level decreases when the number of partitions increases.

Since we have different garbled circuit for every pair of Bloom filters, none of the parties knows if the same Bloom filter of the other party is being compared over and over again. This means the history of comparing any pair of BFs is not recorded, and the only history we care about is the comparison of the partitions.

6.5.1.3 GCRL-Scheme Privacy

In this scheme our main concern is about the privacy of the non-matched records. Our ideal case is to find the matched records without revealing any information about the non-matched records. We say that RL protocol is privacy preserving if whatever can be learned by a party participating in the protocol about the other party's records can be learned based on its input record and the RL function's output only. That is, we require that a party's view in a protocol execution can be simulated by probabilistic polynomial time algorithm given only its input and output.

1. The GC-DC computation is secure:

   Since our scheme used garbled circuits to compute the DC as a function of the parties' Bloom filters and compares it to a pre-set threshold value DC_t, and only returns the result of comparison as *True* or *False*, therefore nothing about these Bloom filters bits

location is revealed. Computations using Garbled Circuits (GC) is proved secure in [60], so our scheme security is defined by sequential modular composition theorem [60], where we assume that there is a trusted party computes the DC functionality.

2. Basic-GCRL Scheme Privacy:

   In our 2-party Basic-GCRL protocol any of the parties could be the adversary (who wants to learn more information than allowed about the other party's input by examining the messages it receives). The main purpose of our protocol is to protect the privacy of the honest party, therefore we will assume that the first party denoted P_H is the honest party, and the second party denoted P_A is the adversary. Let $DC(BF_1, BF_2)$ be a function that computes the Dice coefficient (DC) of any two Bloom filters $BF_1,\ BF_2$, $f_{DC}(BF_1, BF_2, DC_t)$ be a function that, for some DC threshold DC_t, it returns F if $DC(BF_1, BF_2) < DC_t$, and returns T otherwise, and let π_{Basic} be our Basic-GCRL protocol that, securely computes this function using garbled circuits. Furthermore, let $view_i^{\pi_{Basic}}(BF_H, BF_A, DC_t)$ be the view of party $P_i,\ i \in \{H, A\}$ during the execution of the protocol π_{Basic}, which consists of $(BF_i, r_i, \{m_i\})$, where BF_i is the i^{th} party's input BF, r_i is his internal coin tosses, $\{m_i^j : 1 \leq j \leq t\}$ is the set of messages he received, and let $output_i^{\pi_{Basic}}(BF_H, BF_A, DC_t)$ be the output of party P_i during the execution of π_{Basic}, and denote

$$\begin{aligned} &output^{\pi_{Basic}}(BF_H, BF_A, DC_t) \\ &\quad = (output_H^{\pi_{Basic}}(BF_H, BF_A, DC_t), output_A^{\pi_{Basic}}(BF_H, BF_A, DC_t)) \,. \end{aligned}$$

   We say that π_{Basic} is privacy preserving RL protocol in the presence of semi-honest adversaries if there exist a probabilistic polynomial time algorithm denoted $\mathcal{S}$, such that for any pair of BFs BF_H, BF_A such that $f_{DC}(BF_H, BF_A, DC_t) = F$, the following holds

$$\begin{aligned} &\{(\mathcal{S}(BF_A, f_{DC}(BF_H, BF_A, DC_t)), f_{DC}(BF_H, BF_A, DC_t))\} \\ &\quad \stackrel{c}{\equiv} \left\{\left(view_{\pi_{Basic}}^A(BF_H, BF_A, DC_t), output^{\pi_{Basic}}(BF_H, BF_A, DC_t)\right)\right\} \end{aligned}$$

   That is the transcript generated by the view of P_A can be simulated by $\mathcal{S}$ given its inputs and the output only.

Or alternatively

$$\{(view^{A}_{\pi_{Basic}}(BF_H, BF_A, DC_t))\} \stackrel{c}{\equiv} \{(view^{A}_{\pi_{Basic}}(\overline{BF}_H, BF_A, DC_t))\}$$

Which means, the view of P_A when P_H uses BF_H as his input, and the view of P_A when P_H uses different input $\overline{BF}_H$, such that $f_{DC}(\overline{BF}_H, BF_A, DC_t) = F$, is computationally indistinguishable.

Theorem 2. *Our Basic-GCRL scheme π_{Basic} is privacy preserving RL protocol in the presence of semi-honest adversaries*

Proof 3. *In the ideal (with the help of TTP) and the real (using π_{Basic}) executions the only messages the adversary P_A receives is the final result of the function $f_{DC}(\overline{BF}_H, BF_A, DC_t)$, and the view of the adversary could be simulated by ppt algorithm that is indistinguishable from the real execution. Or alternatively, the view of the adversary in both real and ideal executions will be indistinguishable. So no extra information is revealed beyond the RL result.*

3. Partition-GCRL Scheme Privacy:

   The above definition will not work with our partition-GCRL scheme because there are some intermediate messages returned about the compared partitions, which are the DC contribution fulfilment test results (T or F) messages, which we defined by our leakage function $\mathcal{L}_p(\beta)$. As we mentioned before, the adversary will be able to filter-out some of the possible constructs of BFs based on the intermediate results, and narrow his search scope to the BFs that differ in locations starting from the unmatched partition. However, our goal is to prevent the Adversary from pinpointing individual records with high probability. That is to achieve the minimum privacy defined in definition 8.

   So we need to adjust our definition to accommodate this leakage.

   Given a Bloom filters BF_A of party P_A, and two similar BF^1_H, BF^2_H of party P_H, such that $f_{DC}(BF^i_H, BF_A, DC_t) = F,\ i \in \{1, 2\}$, and $f_{DC}(BF^1_H, BF^2_H, DC_t) = T$, we say

that π_{Part}, with leakage upper-bounded by the function $\mathcal{L}_p(\beta)$, is ρ-privacy preserving RL protocol in the presence of semi-honest adversaries if the following holds

$$\begin{aligned}&\left\{\left(view^{\pi_{Part}}_{A(\mathcal{L}_p(\beta))}(BF^1_H, BF_A, DC_t), output^{\pi_{Part}}(\beta, BF^1_H, BF_A, DC_t)\right)\right\}\\ &\stackrel{c}{\equiv}\left\{\left(view^{\pi_{Part}}_{A(\mathcal{L}_p(\beta))}(BF^2_H, BF_A, DC_t), output^{\pi_{Part}}_{P_A}(\beta, BF^2_H, BF_A, DC_t)\right)\right\}\end{aligned}$$

That is, the the views of the adversary P_A during the scheme execution are indistinguishable when the party P_H uses two inputs that have the same leakage, with probability higher than $\frac{1}{p^\rho}$ for $\rho = 1/\beta$, and $\binom{p=n/\beta}{n/2\beta^\beta}$. The probability of success for P_A to guess which of the two inputs (BF^1_H, BF^2_H) party P_H uses is given by

$$Succ_{P_A} = Pr\left[b = \bar{b} \;\middle|\; \begin{array}{l} BF^b_H \leftarrow P_H(b \in_r \{1,2\}, BF^1_H, BF^2_h) \\ \bar{b} \leftarrow P_A(P_H(BF^b_H, DC_t) \stackrel{\pi_{Part}}{\longleftrightarrow} P_A(BF_A, DC_t)) \end{array}\right] < \frac{1}{p^\rho}$$

Theorem 3. *Our Partition-GCRL scheme π_{Part} is ρ-privacy preserving RL protocol in the presence of semi-honest adversaries.*

Proof 4. *We consider an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ according to the following game*

- *At the beginning $\mathcal{A}$ and $\mathcal{C}$ agree on the Bloom filter size and the number of partitions β according to the analysis in 6.5.1.1.*
- $\mathcal{A}$: *Picks BF_A, and two datasets D^1 and D^2 such that*

  $$D^x = \{BF^x_i : f_{DC}(BF^x_i, BF_A, DC_t) = F, \forall 1 \leq i \leq n\},\ x \in \{1,2\}$$

  and for every $BF^1_i \in D^1$ there is at least δ $BF^2_j \in D^2$ such that

  $$f_{DC}(BF^1_i, BF^2_j, DC_t) = T$$

  for some $i, j \in [1, n]$. δ could be set to achieve certain level of deferential privacy. Then send D^1 and D^2 to the challenger $\mathcal{C}$
- $\mathcal{C}$: *picks at random $x \stackrel{\$}{\leftarrow} \{1,2\}$, and $b \stackrel{\$}{\leftarrow} [1,n]$, and sets his $BF_c = BF^x_b$*

- $\mathcal{A}\&\mathcal{C}$: *Starts* π_{Part} *scheme to compare their inputs and at the end* $\mathcal{A}$ *will get the result as "F" (false or no-match) and will learn the extra information* $\mathcal{L}_p(\beta)$. *Then* $\mathcal{A}$ *outputs his guess* $\bar{x}$ *and* $\bar{b}$ *of* $\mathcal{C}$*'s selections of* x *and* b. *That is*

$$(\bar{x}, \bar{b}) \leftarrow \mathcal{A}((\mathcal{L}_p(\beta), F) \leftarrow (\mathcal{C}(BF_c, DC_t) \xleftrightarrow{\pi_{Part}} \mathcal{A}(BF_A, DC_t)))$$

- *Using the extra information* $\mathcal{L}_p(\beta)$, $\mathcal{A}$ *will be able to guess* b *with probability* $\frac{1}{\delta}$, *however* $\mathcal{A}$ *will not be able to guess* x *with probability more than* 0.5 *because the same leakage will produced by two BFs from the two datasets* D^1 *and* D^2. *This means if* $\mathcal{A}$ *doesn't have access to both datasets, and that is the case for record linkage, then based on the information he learns from the RL scheme* π_{Part} *he has to guess at least the contents of the last partition that cause the* π_{Part} *to return "F" as the linkage result. This ends our proof.*

6.5.2 Observations

Probabilistic PPRL methods are superior to their deterministic counterparts because the accuracy of the linkage is better when the data is not very clean, as shown in figure 6.6, where the recall values of the probabilistic method are close to 100% with all the datasets, while the deterministic method was never above 77%.

Performing PPRL using GC allows obviates the need for TTP but it is computationally expensive. Incorporating Bloom filters and Dice coefficient into GC to perform probabilistic PPRL adds more complexity to the process and makes it infeasible in practice. Towards this end, we demonstrated how blocking and our BF partitioning can significantly improve the performance.

Blocking significantly reduces matching time, since each record is compared to only the records of the corresponding group rather than to all records in a dataset. Using BF partitioning substantially shortens the runtime because of two reasons, 1) the GC to compute the DC is small compared to the GC of the whole BF, and 2) computing the DC contribution of each partition reduces the problem size for the following partition, and hence decreases its computation time. The total time to compute the DC of all of the partitions depends on the

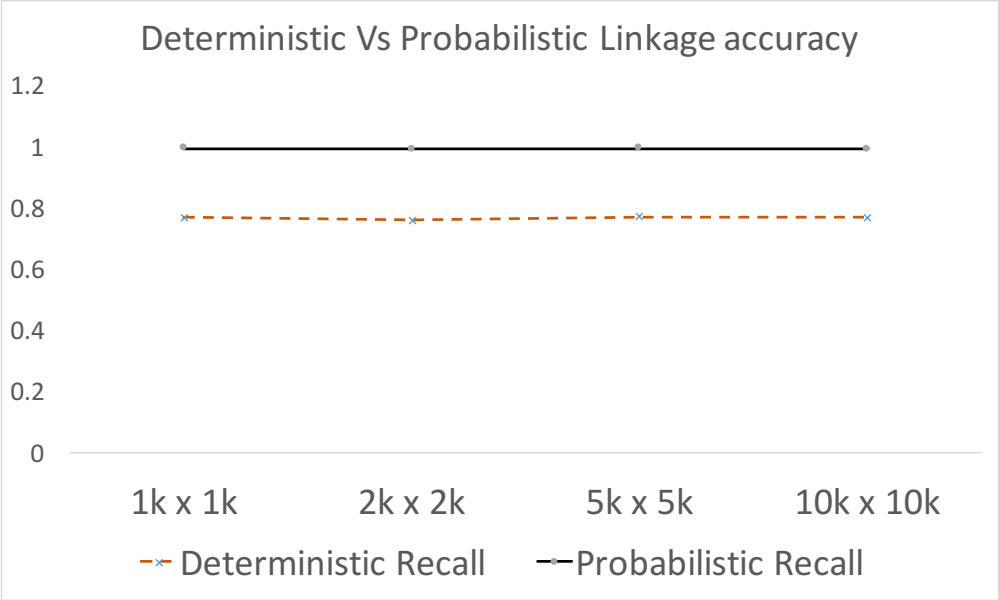


Figure 6.6: Accuracy of our deterministic and probabilistic record linkage schemes

reduction of the problem size, which, in turn, depends on the DC contribution of each BF partition. Indeed, the Dice coefficient computation with Bloom Filter partitioning technique is much faster than the direct DC computation techniques, and without affecting the record matching accuracy. Combining the two techniques (Blocking, and BF partitioning) improves the efficiency of the garbled circuit based record linkage methods. Blocking will speed up the process of large datasets, and BF partitioning will speed up the process within each block.

Chapter 7

Privacy Preserving Approximate Blocking for Record Linkage

In this chapter we propose a Hierarchical Record Blocking scheme using anomymized representation of record Bloom filters. We introduce a mechanism to anonymize and encode the record Bloom filters (BFs) into smaller binary string representation dubbed anonymized Bloom filters (aBFs). These aBFs have pairwise similarity close to the pairwise similarity of their original BFs. To perform record blocking, the scheme performs hierarchical clustering of the records using these aBFs and creates a *Block Pattern* (BP) for each cluster. The hierarchy is represented as multi-layers of weighted graphs. The clustering is based on configurable similarity measure between the aBFs. At each level, a new node that represent the aggregated nodes in the lower levels is created with a new aBF constructed from the aggregated nodes. We introduce the concept of Block Pattern that is used to represent each block and it is constructed from the aBFs during the clustering. The number of edges, and the way of creating the Blocks Patterns are configurable based on privacy and efficiency thresholds. The hierarchy levels could be controlled based on the desired number of blocks or the properties of the patterns. The block patterns are used to group the aBFs of both sides when used to guide and speed up the record linkage process. The blocks could overlap in a controlled manner to strike a balance between performance and the quality of blocking, which are measured by reduction rate and pair completeness respectively. We analysed the correctness and privacy level of the encoding and the blocking schemes, and run thorough tests and verify the results.

7.1 Contribution Summary

In this work, we propose a privacy preserving blocking scheme that does not need a pre-defined blocking key, or any shared reference set. Our scheme may use the service of TTP for extra level of privacy, however the level of privacy achieved by our scheme without TTP is sufficient in most practices. For privacy purposes, we first created an encoded anonymized version of the original BFs (aBFs) for each party, then we designed a blocking technique that uses the anonymized BFs (aBFs) to create a set of block patterns (BPs) that are used to represent the BFs in each block. The encoding is done using unary encoding of the binned ranges of the 1s counts of each segment of the BF, which adds some uncertainty and some false positives. In contrast to other blocking techniques [3, 39, 40, 46, 47, 48, 50, 51, 55, 61, 74, 83, 85, 86, 89], our scheme does not require blocking variable(s) to be pre-defined. We model the similarity relationship between the encoded aBFs as a graph, and use some graph processing algorithms to hierarchically cluster the nodes and create the block patterns. We also designed our clustering algorithm using different strategies and criteria to select the nodes to be merged during the clustering process.

In addition, the number of blocks and the average block size is configurable in our technique to achieve anonymity versus performance balance. Our design allows blocks to overlap in a controlled manner, such that the false negatives could be minimized. However, high overlapping will increase the block sizes and hence will affect the performance by reducing the reduction rate. In addition, parties can tweak the overlapping parameter to achieve higher levels of privacy. We analyze and experimentally study the effect of different BF anonymization configurations, select the best ones, and use them to perform the blocking and measure the scheme quality and performance. Our scheme can be applied in different types of settings: those that use a trusted third party (TTP) and also those that do not use any third party.

In order to have good blocking scheme (high reduction rate), blocking may only be used in a weaker security model and not the security in the cryptographic model, since

the blocked data is not indistinguishable from any arbitrary random dataset (i.e there are common features that aggregate the records in blocks, which can distinguish one block from another). A secure blocking may be designed for PPRL, however it will not perform significantly better than its non-blocking counter part. Nevertheless, anonymization and differential privacy techniques provide sufficient levels of privacy in most cases.

7.2 Binary Data Similarity Measures Overview

There are numerous similarity and distance measures for binary data proposed for various purposes [68]. The selected similarity measure has a great effect on binary data analysis, because each measure targets specific properties of the binary strings. Usually, the similarity is measured based on the counts of positions with positive match (where both strings have the value '1'), the negative match (where both strings have the value '0'), and the mismatch (where one string has a '0' while the other one has a 1). Some measures ignore the negative match, and assume that negative matches do not necessarily mean any similarity between the two inputs. For example *Dice* and *Jaccard* do not use the negative match counts in their measures, while *Yule,Russel&Rao* use them [68]. Also the contribution of the negative match may be used differently among the similarity measures (i.e., to increase or decrease the similarity value). For example, some measures, like *Faith*, assign weights with the negative match counts to increase/decrease their effect.

In our scheme, we need similarity measures during the creation of the representatives (aBFr's), and during creation of blocks to assign each aBF to the block(s) that it has the highest similarity with its representative aBFr. Since representatives creation phase is based on the common 1s locations among the binary strings (aBFs), we need a measure that emphasizes the positive match, and does not ignore the negative match. Some of the measures are not so good in distinguishing between the strings with many ones from the strings with small number of ones, so we need to avoid them in this phase. In blocking phase, we need a measure that emphasizes the common ones between the aBFr's and the aBF's. More details on this will be discussed in section 7.4.3.

For two strings S_1, S_2 of length l, if we denote their positive match count by a, their negative match count by d, their mismatch count when S_1 has 1 and S_2 has '0' by b, and their mismatch count when S_1 has '0' and S_2 has 1 by c, then a general similarity measure [5] can be written as

$$Sim(S_1, S_2) = \frac{\alpha a + \beta d}{a + b + c + d}$$

where α and β are the weights assigned to positive and negative match counts respectively.

The following well known similarity measures may be used with our scheme:

- The intersection (Bitwise AND):

$$Sim_{intersection}(S_1, S_2) = a$$

  or it could be normalised by the length of the strings, which is aka Russel&Rao, where $l = a + b + c + d$.

$$Sim_{Russel\&Rao}(S_1, S_2) = \frac{a}{l}$$

- The Faith:

$$Sim_{Faith}(S_1, S_2) = \frac{2a + d}{2l} = \frac{a + 0.5d}{l}$$

- Dice asymmetric

$$Sim_{Dice_{A|B}}(S_1, S_2) = \frac{a}{a + b}$$
$$Sim_{Dice_{B|A}}(S_1, S_2) = \frac{a}{a + c}$$

- Dice

$$Sim_{Dice}(S_1, S_2) = \frac{2a}{2a + b + c}$$

- Jaccard

$$Sim_{Jaccard}(S_1, S_2) = \frac{a}{a + b + c}$$

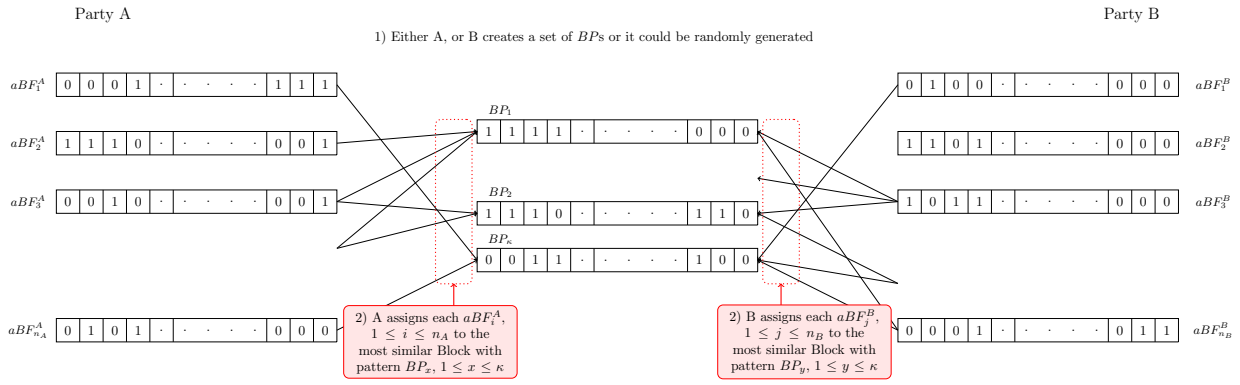


Figure 7.1: Symmetrical Blocking: Both parties use the same set of block patterns aBFrs to assign their aBFs to the Blocks.

- The overlap:

$$Sim_{overlap}(S_1, S_2) = \frac{a}{min((a+b),(a+c))}$$

7.3 Privacy Preserving Blocking

In the following we define the problem of privacy preserving blocking, and the two approaches we design to perform our privacy preserving blocking.

7.3.1 Symmetrical Blocking

In Symmetrical Blocking one of the parties groups his anonymized aBFs, based on some predefined similarity measure, into κ groups (blocks). Then for each block t, he creates a unique block pattern (BP_t) and sets the unique index $1 \leq t \leq \kappa$ as the block Id. Then, the set of unique patterns $\{(t, BP_t)\}$ is shared with the other parties to use it as a reference to group their sets of aBFs. As a result, the sets of aBFs on each side are grouped using the same set of unique patterns and Ids. Figure 7.1 shows an illustration of this process. Note that, each aBF could be assigned to a zero or more blocks (see section 7.4.2 for details). We also would like to mention that, it is possible to randomly generate the set of unique patterns $\{(t, BP_t)\}$ (securely using secure computations, or via TTP) instead of generating them from one of the parties' data.

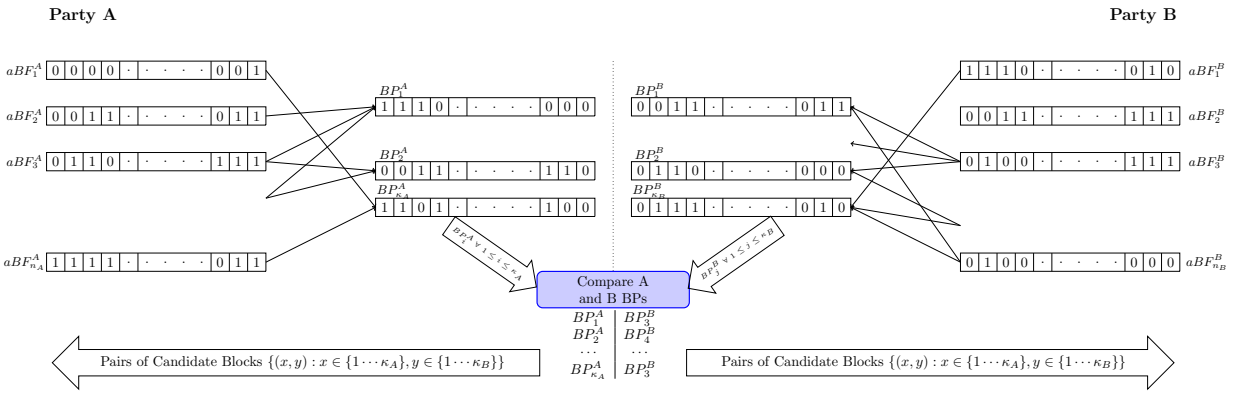


Figure 7.2: Asymmetrical Blocking: Each party $P_i \in \{A, B, C, ..\}$ creates his set of blocks patterns $BP_t^{P_i}$ and sets the index $1 \leq t \leq \kappa_i$ as the block Id, and assigns his aBFs to the corresponding Blocks. Then the parties collaboratively (or Via TTP) compare their BPs and create all possible block pairs.

7.3.2 Asymmetrical Blocking

In asymmetrical Blocking each party $P_i \in \{A, B, C, ..\}$, with dataset of n_i anonymized aBFs $T_i = \{aBF_x^{P_i}\}$, $1 \leq x \leq n_i$, independently groups his dataset T_i, based on some predefined similarity measure, into κ_i groups (blocks). Then for each block t, he creates a unique block pattern $BP_t^{P_i}$ and sets the index $1 \leq t \leq \kappa_i$ as the block Id. Then the parties collaboratively create a set of all possible similar block pairs, by comparing their sets of block patterns. The next step is to create all possible candidate record pairs from the block pairs. The resulted blocking is asymmetrical in the sense that there could be many to many high similarity relationship between the blocks. Figure 7.2 shows an illustration of this process.

Definition 9. ***Privacy Preserving Probabilistic Blocking Problem:***

Without loss of generality, let us assume that we have two parties A and B with their respective large datasets T_A with size n_A, and T_B with size n_B, who want to perform PPRL on their datasets. Both parties agree to use a set of Bloom filter representations of their records to perform probabilistic PPRL, by securely computing the Dice Coefficients of each of the $n_A \times n_B$ record pairs. However, to do that in an efficient manner and reduce the unnecessary comparisons or records that will not be linked, they want to group their records in blocks such that only records in similar blocks are compared. Using traditional blocking methods is not an

option because the common blocking variable reveals too much information about the records in each block. So they need a blocking scheme that groups their Bloom filters in blocks with Blocks IDs that do not reveal much information about the BFs in each block. i.e. given any Block Id, there should be no method that could determine (with high certainty) the BFs in that Block.

7.4 Proposed solution

To solve the problem of Privacy Preserving Probabilistic Blocking (PPPB), and allow both parties to perform the blocking in a similar manner, and without revealing too much information (i.e. the shared blocking information must provide a certain level of anonymity), we propose the following method:

- Each party creates anonymized version of their BFs which we term aBFs, such that knowing any aBF will not identify the corresponding original BF with probability higher than certain threshold (level of privacy)

- Parties collaboratively decide on the usage of symmetrical or asymmetrical blocking, and how the blocks patterns (BPs) will be generated (by one of the parties, randomly via secure computations, or randomly via TTP).

- One of the parties (if symmetrical blocking) or both parties (if asymmetrical blocking), hierarchically cluster their anonymized aBFs, and create anonymized block pattern (BP) of each cluster. The BP of each cluster is an aBF that has a high similarity value with each aBF in that cluster. This similarity value must be above a pre-defined similarity threshold δ that is set based on the required level of anonymity. The lower the similarity value, the higher the level of anonymity we get and less efficient blocking (lower reduction rate).

- Based on selected blocking scheme (symmetrical or asymmetrical blocking), they create blocks pairs (possible blocks to compare) by 1) directly exchanging the BPs, 2) securely

performing one side blocking, 3) securely computing the similarity of their BPs, or 4) utilizing the service of TTP to compare the BPs. We will elaborate on these solutions in section 7.4.4.

- using the corresponding block pairs, they create a list of candidate record pairs from each block, and start PPRL process using this record pairs list.

7.4.1 Anonymizing and Encoding The Bloom filters

In this section we introduce our anonymization scheme based on binned counts of 1s in partitioned Bloom filters. The bin ranges and partition sizes are configured to achieve the required level of anonymity and tolerated similarity error.

Definition 10. ***Thresholded Unary Encoding-*** **Uencode***(v,$\mathcal{B}$,min,max):* *Given min, max values, a value $min \leq v \leq max$, and a set of q thresholds $\mathcal{B} = \{b_i : 1 \leq i \leq q, b_i < b_{i+1}\}$, the Unary code string u of the value v is defined as follows:*

$$u = \begin{cases} min \leq v < b_1 & \underbrace{\text{"}00\cdots0\text{"}}_{q\ 0s} \\ b_i \leq v < b_{i+1} & \underbrace{\text{"}11\cdots1}_{i\ 1s}\underbrace{00\cdots0\text{"}}_{q-i\ 0s},\ 1 \leq i \leq q \\ b_q \leq v \leq max & \underbrace{\text{"}11\cdots1\text{"}}_{q\ 1s} \end{cases}$$

Definition 11. ***Odd/Even counts*** **- OEcounts***(s):* *Given a string s of length l, we define c_o and c_e as the count of 1s in the odd and even locations of the string s respectively as follows*

$$c_o = \sum_{\substack{j=1,\\ j+=2}}^{l} s[j], \quad c_e = \sum_{\substack{j=2,\\ j+=2}}^{l} s[j]$$

In order to perform the blocking using the Bloom filters BFs, and hide the actual locations of the 1s in the original BFs, we introduce the idea of anonymized BFs by encoding the 1s counts of partitioned BFs. First, we divide the BF of size n into k partitions $p_1, \cdots, p_k$ of size l bits each, i.e $n = k * l$ (assuming k divides n). let min_{1s}, and max_{1s} represent the expected maximum and minimum number of 1s in each partition respectively. For example,

if we expect half of the bits of the BF will be 1s, and the bits are uniformly distributed, then we might expect the number of 1s in each partition to be around half the size of that partition, i.e. $l/2$. In the extreme cases we might expect the maximum number of 1s to be the size of the partition l, and the minimum number of 1s to be zero.

Then to create the anonymized code u^i for the partition $p_i, 1 \leq i \leq k$ do the following:

- count the number of 1s in the partition, as a total c^i, or separately count the 1s in the Odd and Even locations using **OEcounts**(p_i) as c_o^i for the Odd count, and c_e^i as the Even count. Counting the 1s in the Odd and Even locations will increase the length of the encoded BF, but it will improve the accuracy of the encoding, which in turn will improve the accuracy of our similarity measures, and hence the quality of the blocking as we will explain later.

- based on the expected min_{1s} and max_{1s} values of the partition count(s) and the required levels of anonymization and accuracy of the encoding, specify the set of bin thresholds $\mathcal{B}$ (the upper limits of counts ranges), and whether or not to use odd/even counts.The code length (how many bits used to encode the counts) will be equal to the number of bins thresholds if no odd/even counts is used, and double that if odd/even counts is used. The set of bin thresholds $\mathcal{B}$ is used to define the range of 1s count values (in the whole partition or in the odd/even bit locations of the partition) to be represented by each code. Note that when Odd and Even counts are used, the min_{1s} and max_{1s} values will be set as half of the values used for the total partition count (assuming half of the bits are in the odd locations and half of them are in the even locations).

- with the partition total 1s counts c^i or odd/even locations 1s counts (c_o^i, c_e^i), the set of bin thresholds $\mathcal{B}$, and min_{1s},max_{1s} values, get the corresponding partition code u^i using our unary encoding explained in definition 10 as: $u^i =$ **Uencode**$(c^i,\mathcal{B},min_{1s},max_{1s})$ if total count is used, or $u^i = u_o^i|u_e^i$, if Odd and Even counts are used, where $|$ denotes concatenation, $u_o^i =$**Uencode**$(c_o^i,\mathcal{B},min_{1s},max_{1s})$, and $u_e^i =$**Uencode**$(c_e^i,\mathcal{B},min_{1s},max_{1s})$.

- The length of u^i will be $l_u = |\mathcal{B}|$ if no Odd/Even counts are used and $l_u = 2 \times |\mathcal{B}|$ if Odd/Even counts are used. The size of the resulting encoded aBF will be $S_{aBF} = l_u \times k$.

Specifying the number and the values of the bin thresholds is very critical in our anonymization encoding. The number of bin thresholds will set the code length, and hence the resolution of the code. The threshold values will set the range of values to be represented by each code, which will affect the tolerance or error (uncertainty) between the values that would be considered the same after encoding. For example, for 1-bit code, we need to set only one threshold $\mathcal{B} = \{b_1\}$, then our values range will be the intervals $[min_{1s}, b_1), [b_1, max_{1s}]$ and encode the counts in the first interval (below b_1) as '0' and the counts in the second intervals (above or equals b_1) as '1'. If we increase the code length we can have more informative encoding. For example, a 2-bit code allows us to set 2 thresholds $\mathcal{B} = \{b_1, b_2\}$, and divide our range into the three intervals $[min_{1s}, b_1)$ encoded as '00', $[b_1, b_2)$ encoded as '10', and $[b_2, max_{1s}]$ encoded as '11'. The threshold values b_i's will control the number of possible count values, and hence will effect the anonymity level. See section 7.5.2 for more details.

Figure 7.3 shows an example of encoding a BF using this scheme.

7.4.2 The Blocking Algorithm

Our blocking method groups the aBFs of the records based on a common feature, which is common 1s, that is measured by their similarity values. We create a hierarchical structure using this common feature among the similar aBFs. In this structure the common feature is used as a pattern to identify each group on each level. Since the distribution of 1s in each BF will be controlled by the data (in this case the Bi-grams) inserted in it, and hence the corresponding aBF will reflect this distribution, we expect that some of the aBFs will have some features shared with many other aBFs, and consequently there will be many possible ways to assign these aBFs to groups and create the pattern of these groups. That is, we will have some aBFs that could participate in more than one group. What we aim for is to have the aBFs at the center (very similar to the pattern) of the group participate only in that group, while the ones a little bit far from the center (at the edges) to possibly participate

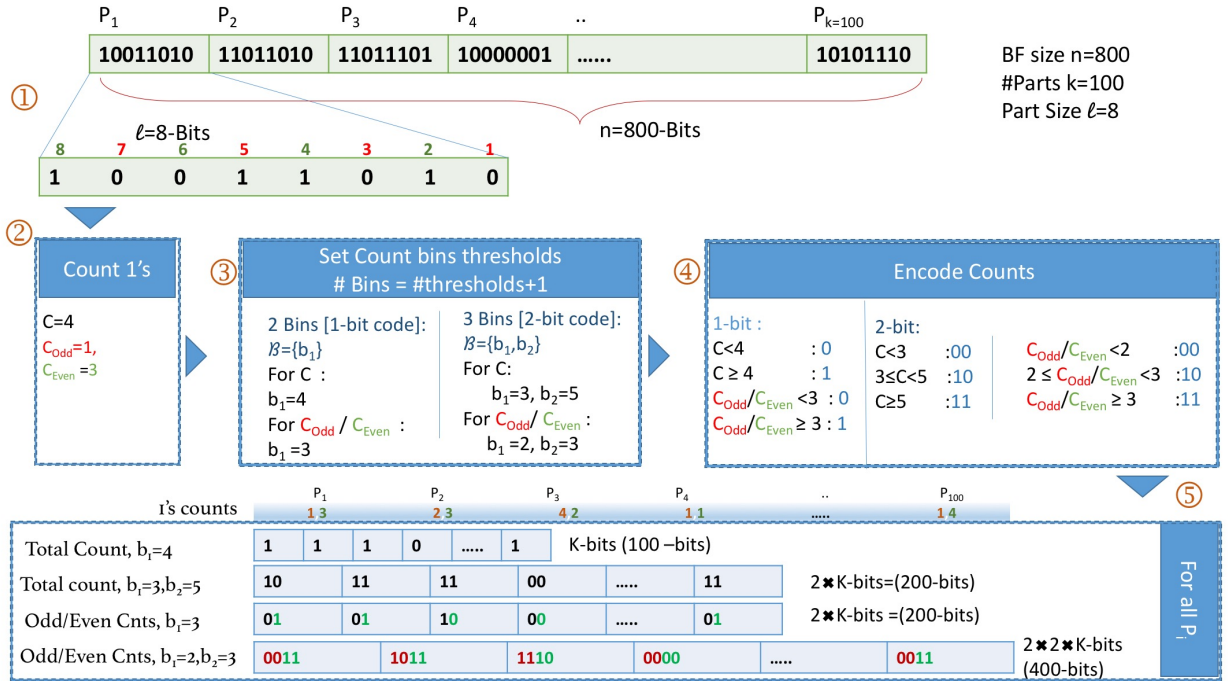


Figure 7.3: Bloom Filter anonymization and Encoding example.

in more than one group. However, such practice of having overlapped groups will effect the performance of our blocking such that large overlaps will result in less efficient blocking (many aBFs will be processed multiple times in the linkage process, and hence low reduction rate). However, the chances to miss any candidate record pairs will be less, and this will lead to higher pair completeness. On the other hand, overlapping will help in getting better privacy.

Our algorithm uses graphs to represent the relation among the patterns of each level of the hierarchy, and the nodes of similar groups will be merged as we go up the hierarchy until the desired number of clusters is reached, or we hit a minimum merging similarity threshold.

7.4.2.1 Modeling the first layer

Each data record is represented by an Id and a binary string of its encoded anonymized aBF. Based on configured similarity measure the pairwise similarity of all aBFs are calculated. Then the Ids of each pair of records with their aBFs similarity above a cut-off threshold (a user-specified minimum similarity threshold to reduce graph edges) are inserted

Algorithm 2 Hierarchical BF clustering Algorithm

Inputs:

- IDList ▷ *List of IDs,*
- G0 ▷ *Level 0 Graph created from the pairwise similarities of all aBFs, where each node has the recID, aBF, and edges labled with similarity values.*
- reqNumbOfCls ▷ *Required number of clusters*
- SimMergeTh ▷ *Similarity Merge threshold (lowest Sim. value to merge)*
- Patt1sDelta ▷ *percentage of cluster members to have ones in each cell to consider the patt of the cluster to have one.*

Output: Hierarchical Clusters of aBFs and their Patterns

Initialization:

$\sigma \leftarrow 0.05$ ▷ *Similarity decrement value*

MergeSim $\leftarrow 0.99$ ▷ *Start merging high similar nodes first, then decrease by* σ

numCls $\leftarrow$ len(IDList) ▷ initially each rec in one cluster

level $\leftarrow 1$

Clusters[0]$\leftarrow$ G0

while numCls > reqNumbOfCls **and** MergeSim > SimMergeTh **do**:

Clusters[level] $\leftarrow$ ClusterUsingGraphMaxSim(Clusters[level-1],MergeSim,reqNumbOfCls,Patt1sDelta)

numCls $\leftarrow$ len(Clusters[level])

if len(Clusters[level]) == len(Clusters[level-1]) **then** ▷ *no change, i.e. no merges*

if numCls > (reqNumbOfCls+ err): **then** ▷ *if nCls is not close enough*

▷ reduce the MergeSim by σ

if MergeSim > SimMergeTh: **then**

MergeSim $\leftarrow$ NergeSim-σ ▷ *Reducing MergeSim by* σ

level $\leftarrow$ level+ 1

end if

else ▷ *close enough*

Break

end if

end if

end while

return Clusters

Algorithm 3 ClusterUsingGraphMaxSim

procedure ClusterUsingGraphMaxSim

Inputs:

- pClusters ▷ *Graph of previous clusters*
- MergeSim
- Patt1sDelta

```
cnt ← minMergesPerLevel
level ← 0
nCls ← len(G.nodes())
nLevel ← new Graph                          ▷ new empty graph for the new level clusters
cnt ← 0
i ← 0
sortedElist ← sorted(Clusters.edges)        ▷ Sort the graph edges on their weights, largest first
w ← sortedElist[0]
while i < len(sortedElist) and w ≥ MergeSim: do
    n1,n2,n1n2w ← sortedElist[i]
    cnt ← cnt+1
    newNode←mergeNodes(nLevel, n1, n2, Patt1sDelta)
    i← i+1
end while
                  ▷ calculate pairwise similarity and add edges between the new level nodes' aBF
if cnt > 0 then createSimEdges(nLevel)
end if
 return nLevel
end procedure
```

Algorithm 4 mergeNodes

procedure mergeNodes

Inputs:

- G ▷ *Graph of clusters*
- node1, node2 ▷ *nodes to merge*
- Patt1sDelta

```
nnId ← 'M'+node1                            ▷ new Node's Id
n1PattSum ← G.node[node1]['PattSum']
n2PattSum ← G.node[node2]['PattSum']
nnPattSum ← n1PattSum+n2PattSum             ▷ the sum of all corresponding cells in all cluster Patts
nIdList ← G.node[node1]['IdList']
nIdList ← nIdList + G.node[node2]['IdList']  ▷ append the two ID lists of the merged nodes
nBFs ← len(nIdList)                         ▷ Number of Patts in this cluster
                  ▷ Set the new node aBF's cells as 1 if their sum is ≥ percentage of the cluster size
nPatt ← (nnPattSum ≥ int(round(Patt1sDelta*nBFs)))
G.add_node(nn, Pattp=nPatt, PattSum=nnPattSum, IdList=nIdList)
end procedurereturn nnId
```

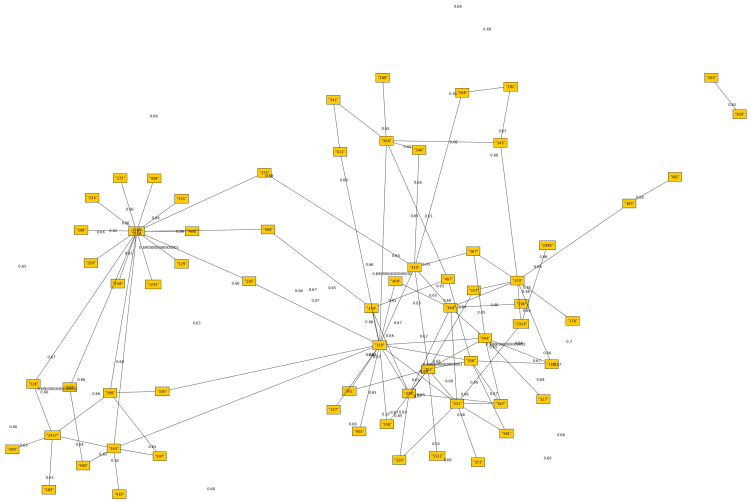


Figure 7.4: A part of the 10k test dataset's similarity graph (Level 0).

into a weighted edge list, where the weight is the similarity value. A new graph of the first layer is created from the edge list with the record Id as the label of the node, the aBF string as the node *pattern*, and the similarity values as the edge weight between similar nodes. Figure 7.4 shows a partial graph created from the similarity edge list of a 10k test dataset.

7.4.2.2 Building more layers

To build the clustering hierarchy, a new layer is created by merging similar nodes from the previous layer. We propose different strategies for the clustering process. We also define a limit for the number of levels, or set the desired number of clusters to control the clustering process. Our method can also use the number of ones in the patterns as a threshold to stop merging the nodes during the construction of the higher layers.

User-specified similarity thresholds are set as the minimum similarity value between the nodes to be merged. For example, if there is no pair of nodes with similarity higher than the threshold, we stop merging at that level, and the number of nodes at that level will be the final number of clusters. That is, we can control the number of levels according to the desired number of clusters and the minimum similarity threshold. Each time we merge two nodes, the new node created in the upper level will have a new pattern constructed from

the merged nodes. The new pattern will be used to calculate the similarity between the new nodes, and creating the new edges between the nodes.

Because a node might have too many edges that connect it to its similar nodes, finding optimum way to merge the nodes of the graph in the next level is quite complicated, and could effect the clustering process in terms of efficiency and performance. Following are some strategies we adopted in this work to select the nodes to merge.

- **Using Maximum Weight Match Algorithm** A matching M in G is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex. A matching M of a graph G is maximal if every edge in G has a non-empty intersection with at least one edge in M [20]. Starting from the first level, the maximum weight matching [20] is used to select which pair of nodes to merge in the next level.

- **Using Sorted Edge Weights** In this ad hoc approach, we sorted the graph edges based on their weights in descending order (largest first), and start picking the top pair of nodes to merge. Since nodes with highest similarity values are most likely to produce a pattern with highest number of ones when merged. This means less loss in the similarity between the pattern and the cluster members (see section 7.4.3 for details on new node pattern creation).

- **Using Sorted Nodes Degrees** In this case the nodes are sorted in an ascending order according to their degree, and the node with least number of neighbours is picked first. Since the node with least number of neighbours does not have many options to be merged, we can guarantee its selection in at least one cluster (of course if it fulfils other requirements, like minimum merge similarity).

- **Using Sorted Number of Common Neighbors** start with some main node (for example, using sorted degrees), and create a list of its neighbors, and then find the neighbor node which has the highest number of common nodes with this main node (i.e. the node that creates as much triangles as possible with it). The candidate nodes for the merge then will be the main node, the neighbor node, and all the common

nodes. This will lead to better similarity between the nodes, and hence better new node pattern with high similarity to the merged nodes. Alternatively, we can find the main node by selecting the two nodes that has the highest number of neighbors in the whole graph (by calculating the square of the adjacency matrix of the graph, and pick the pair of nodes with the highest value if (or if not) directly connected).

7.4.3 Creating the New Node Pattern

Initially (at level 0) the node patterns are the aBFs themselves. When two nodes (at any level) are merged into a new node (in an upper level), a new pattern, for the new node, is created from the patterns in these merged nodes. The new node will point to all the nodes in the lower levels, and its pattern will represent all the aBFs of these merged nodes (cluster). Creating the new pattern is a pivotal process in this clustering technique. Since the next level merging decision will be made based on the similarity of the patterns of the nodes, these patterns must be created in such a way that they stay within similarity threshold from all the members of the cluster. For example if our measure is the number of common ones, then each of the aBFs in the cluster must have at least a pre-set value of ones in common with the cluster pattern. In this work, we created the new nodes' patterns based on the similarity measure and using the following approaches:

1. **Using strict intersection (ANDing the patterns)** In this approach, each bit in the new pattern will be set as 1 only if all the corresponding bits have 1 in the patterns of all nodes in the cluster. That is, we apply the bitwise-AND operation between the patterns of all nodes in the cluster. In this case we guarantee that each member of the cluster will have at least the same ones in the cluster's pattern. However the slightest difference in the pairwise similarity will reduce the number of ones in the pattern, which in turn will lead to lower accuracy of the similarity between clusters patterns in the upper levels. To control the accuracy in this case a threshold for the minimum ones in the pattern is set, so the number of 1s in the patterns will not go below that threshold.

2. **Using permissive intersection (Setting a minimum threshold)** In this approach, the corresponding bits of all the aBFs in the cluster are summed, and if their summation is greater than a percentage threshold of the size of the cluster, then the corresponding bit in the pattern is set to '1' otherwise it is set to '0'. This process is repeated every time a new member is added or removed from the cluster. If we denote the size of the new formed cluster (Cl) as S_c, the number of bits in each node pattern BP as d, and the percentage threshold as Δ, then the d-bits summation of the corresponding bits of all the S_c patterns of the nodes in the cluster Cl will be:

   $CPat_{Sum}[i]_{1 \leq i \leq d} = \sum_{\forall BP \in Cl}^{S_c} BP[i]$

   and the d-bits of the new pattern will be set as follows:

$$BP[i]_{1 \leq i \leq d} = \begin{cases} 1, & \text{if } CPat_{Sum}[i] \geq \lfloor S_c * \Delta \rfloor \\ 0, & \text{otherwise.} \end{cases}$$

   The Δ value could be chosen such that the pairwise similarity between the members and the pattern will not go below Δ. In our experiments, we adopted this technique to construct the pattern of the new node for the reasons explained in section 7.5.3.

7.4.4 Exchanging the Blocking Information

In order to have an efficient blocking (high reduction rate), both parties need to either share the final set of clusters' patterns, or compute their similarities as we explained in the symmetrical and asymmetrical blocking approaches. To accomplish that according to the desired level of privacy we suggest the following:

1. One party do the clustering and create the clusters patterns (symmetrical blocking): The advantage of this approach is that both parties will use the same patterns and block ids (using the patterns or their indexes), and the blocking is consistent on both sides, which will substantially improve the record linkage performance. let's assume party A created the set of blocks patterns BP, with size $\kappa = |BP|$ patterns, that is κ is the number of blocks. Since the patterns were created with some anonymity (i.e

clustering is done on anonymized, coded, and short versions of the actual BFs, and the bits of the pattern were set under threshold condition), our privacy model of this sharing approach will depend on the following two scenarios:

- If party A will tolerate the amount of information in the patterns to be leaked, then party A forwards the set of patterns BP to party B, and B will create κ blocks by calculating the pairwise similarity between every pattern in BP and every aBF_j^B in his dataset, and using the same overlap and similarity thresholds used by A. In this case, B will perform $n_B \times \kappa$ local similarity computations to do the aBFs to block assignments. This method is very efficient performance wise, however party B will learn some information about A's records. A can add some randomization to the patterns to increase the anonymity of his records. For example, by changing some bits in the patterns or by lowering the value of Δ used in the cluster representative creation (which will make block sizes larger).
- Party A and B start secure computation scheme to perform secure blocking, where party B creates his blocks based on A's patterns without sharing their inputs.That is, they compute the similarity of each $aBF_j^B \in B$ with every pattern from A's side. Using Garbled Circuits (GC) B starts as the *Generator* and creates circuits to compute the the similarity values between his aBFs and the patterns of A. A will be the *Evaluator* of the circuits, and his inputs are the set of κ patterns. Since B is the generator, he will be the only one to know the results of the computation, and they are just the similarity values. In this case A's patterns are kept secret. Though, this scheme will be computationally less efficient than the previous scheme, its level of privacy is much greater. In addition, the aBFs sizes are small (compared to the original BFs), and the number of blocks κ is small compared to the size of the dataset n_A, which makes the GCs smaller and the blocking time tolerable.

2. Both parties separately do the clustering and create their own sets of patterns (asymmetrical blocking):

   Since the patterns are created separately, there is a high chance that they are different, and the blocking on both sides will not be consistent. Thus the two parties need to first compare their set of patterns to create all possible blocks pairs (i.e., there could be some blocks to be compared with more than one block). Then from the blocks pairs they construct the candidate record pairs. To do this type of asymmetrical blocking, they need to follow one of the following approaches:

   - Exchange their blocks patterns, so they can calculate their similarities or select which ones to use.
   - Use TTP to compute the similarities of their block patterns, and create a list of candidate block pairs.
   - Use secure computations to compare their block patterns, and create a list of candidate block pairs

   Then for the record linkages they need to compare the actual BFs of each candidate record pair. Since multiple blocks may be paired together, the reduction rate (RR) will be less than that of the symmetrical approach, and hence less efficient blocking. In our experiments we used the symmetrical approach where one party shares his block patterns with the other party. As future work we intend to compare the two approaches.

7.5 Scheme analysis

In this section we analyze the scheme to see how correctly the anonymized Bloom filters (aBFs) represent the original BFs, and what level of anonymity the anonymization and encoding scheme can achieve by configuring the partition size, the bin thresholds, and whether to use odd/even locations 1s counting. We will focus our analysis on using odd/even loca-

tions 1s counts, since it gives better resolution, however using total counts follow the same analysis but will incur larger error margins.

7.5.1 Correctness:

To analyze how likely similar BFs get encoded to similar (not necessarily the same) aBFs, and non-similar BFs get encoded to non-similar aBFs with acceptable margin of error (i.e. to what level the similarity between the BFs is preserved after anonymizing and encoding them), we analyze the probabilities of matching the BFs given that their corresponding aBFs are matched as follows.

For two parties A and B with bloom filters a and b respectively, we analyze the probabilities that their corresponding anonymized and encoded aBFs $\hat{a}, \hat{b}$ will match as follows:

Assuming uniformly distributed bin thresholds, each party will divide its BF into k partitions of l bits. Our encoding string of the partition 1s count will gradually change as we move from the first bin to the last bin by starting as string of all 0s for the first bin, and then adding ones gradually up to string of all 1s for the last bin. In other words, the coding strings of the adjacent bins are very similar, while the similarity decreases as the bins get far from each other (see section 7.4.1 for the details of the encoding procedure).

When the codes are similar, this will result in *negative matches* (matched by 0s) for corresponding code strings in the lower bins, and *positive matches* (matched by 1s) for the upper bins, and a mix of negative and positive matches in the middle bins, which reflects the tendency of the corresponding partitions to positively or negatively match. That is, if the original partitions have more 0s than 1s, then they will tend to negatively match more than positively match, and the same will happen to their corresponding codes. The same applies to the positive matches when the number of 1s are greater than the number of 0s. In the middle bins, where the number of 1s is close to the number of 0s, there will be a mix of negative and positive matches.

Using the odd/even locations 1s counts will increase the resolution of the codes by specifying where would be the positive/negative matches when the counts are close. For example,

if the total number of 1s in A's partition is 12, and 9 of these 1s where in odd locations, and its code (based on some bin threshold) was '10' (1 for odd count, 0 for even count), and if the corresponding partition from B has 17 1s, and was encoded as '11', then these two partitions is more likely to positively match in the odd locations and not match in the even locations, and so are the codes.

If we denote the corresponding partitions a^i, and b^i, their odd/even counts a^i_{oc}, a^i_{ec}, and b^i_{oc}, b^i_{ec}, and their encoded partitions counts of $\hat{a}^i_c$ and $\hat{b}^i_c$, then for the two partitions a^i and b^i we have the following two cases:

1. *both codes are the exactly the same:*

   $\hat{a}^i_c = \hat{b}^i_c$ (i.e., the counts are within the same bin, and the aBFs $\hat{a}, \hat{b}$ will exactly match at those codes).

   Without loss of generality, let us assume that the number of odd locations is half the partition size (i.e. partition size is even). That is, the number of odd and even locations equals $l/2$ each, and we expect the number of ones in each partition to be approximated to a constant value ($C = \frac{Average\ number\ of\ 1s\ in\ BF}{number\ of\ partitions}$), that is, for party A we expect $a^i_{oc} + a^i_{ec} \simeq C$, and the same for party B. The other cases where the number of 1s is smaller than C (encoded as 0s most of the time) are less frequent and they will be negatively matched, so they will introduce some small false positive errors, but will not influence our analysis much.

   Thus we have either (1) the odd counts equals the even counts (i.e the 1s are distributed evenly over the odd and even locations), or (2) the odd counts greater/smaller than the even counts (more 1s in either locations). In (1), if half or less of the odd (resp. even) locations in the corresponding BFs partitions a^i and b^i are occupied, then there is a probability that a^i and b^i will not intersect at any bit, which means this will be a false positive (since the codes match). However this probability is small, for example this happens when all bits are in opposite locations on both sides as shown in figure

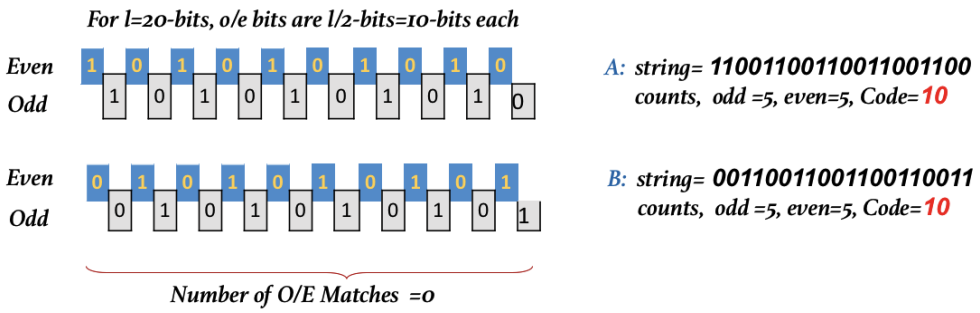


Figure 7.5: An example of similarly encoded partitions with *Zero* matched cells when both counts (Even/Odd) are less than or equal $l/4$, where l is the partition size.

7.5. Then probability of the two partitions to intersect in more than d bits of the odd or the even locations will be:

$$p(|a^i \wedge b^i|_{odd} > d) = 1 - \frac{\sum_{j=0}^{d} \binom{l/2}{j}}{\sum_{j=0}^{l/2} \binom{l/2}{j}}$$

that is, 1 minus the probability they will not intersect in d or more bits location.

In (2) one of the counts is larger than the other, and hence most of the odd or most of the even locations will be ones. The minimum number of matched bits in the original partitions will be either the minimum bits matched in the odd locations $M_{odd} = a^i_{oc} - [l/2 - b^i_{oc}] = a^i_{oc} + b^i_{oc} - l/2$, or in the even locations $M_{even} = a^i_{ec} + b^i_{ec} - l/2$, and the number of minimum matches increases as one of the counts exceeds the other (i.e the difference between them increases). Figure 7.6 shows an example for similarly encoded partitions, and the even count is greater than the odd count for each partition. In this example, the partition size $l = 20$, for party A the odd count is 4, and the even count is 6, and for party B the odd count is 4, and the even count is 7, and the min number of matches will be 6-[20/2 -7]=6-3=3.

2. *the codes are different:*

   In this case the counts are different, and this might result from the following two cases:

   - *the counts are within bins far from each other:*

     If the count values lie into non-neighbor bins, then they are quite different from

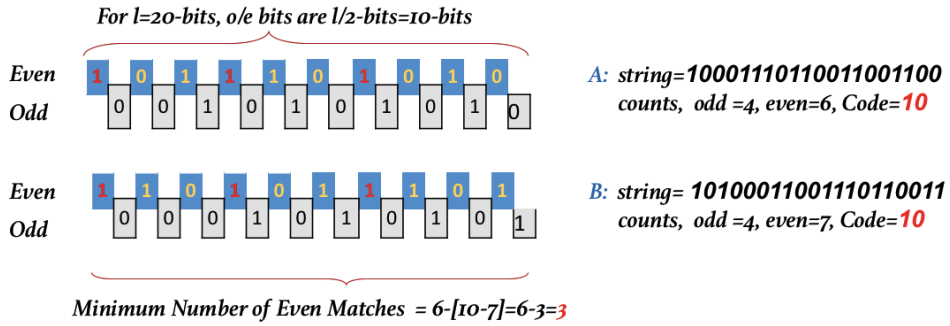


Figure 7.6: An example of minimum matched cells when one count (Even) is larger than the other (Odd) for similarly encoded partitions.

each other, and their codes will be different too. For example, if we have 2-bit coding, and a^i partition count value is in the first bin, and b^i's count value is in the third bin, then their corresponding codes will be '00' and '11' respectively, and the codes will not match. This reflects the fact that if the counts assigned to separate bins, the chances of matching the 1s in the original partitions will be low, and even if there is a match it will be in small number of bits (could be neglected, it wouldn't effect the similarity results a lot), and the same should happen for the corresponding codes (i.e codes will not match as well).

- *the counts are within intervals close to each other:*

  In this case the the codes will slightly overlap, which means that there is a possibility of some 1s in the original partitions will match, and the match in the codes represents that possibility. For example, if one count value was in the second bin (coded as '10') and the other count value was in the third bin (coded as '11'), then the two codes match in the 1st bit, and hence some of the 1s in the original partitions may match.

We experimentally verified our analysis, and measured the errors between the similarity values of the original Bloom filters (BFs) and the similarity values of the anonymized and encoded Bloom filters (aBFs). The experiment details and results will be discussed in the Experiments section 7.6.2.

7.5.2 Anonymity of the encoding scheme:

To model the amount of anonymity that our encoding scheme provides, we analyze the probability of knowing the pattern of 0s and 1s in any partition of the original BF from knowing the corresponding code our scheme produced for that partition in the aBF. We assume our adversary here has access to the BF construction procedure or can create a BF for any record of his choice, for example he might be one of the parties. For adversaries without this kind of access, getting information about the original BFs from the aBFs is hard.

Let ρ_i be the pattern in odd, even, or all locations, of partition P_i of the BF, $\mathbf{c}_i$ be the code corresponding to this partition in the aBF, generated using the pattern's 1s counts and bin b, and let l_m, l_u be the lower and upper limits for the range of values of bin b, then the number of all possible patterns $\theta_{\mathbf{c}_i}$ that will generate the code $\mathbf{c}_i$ will be $\theta_{\mathbf{c}_i} = \sum_{j=l_m}^{l_u} \binom{s}{j}$, where $s = |\rho_i|$, is the size of the pattern. Note that $s = l$ when all locations count is used, or $s = l/2$ if odd/even counts are used, for the partition size l.

If all patterns are equally likely to occur, then given $\mathbf{c}_i$, the probability of guessing ρ_i will be $Pr(\rho_i/\mathbf{c}_i) = 1/\theta_{\mathbf{c}_i}$.

Note that, the value of $\theta_{\mathbf{c}_i}$ will depend on the size of the pattern s and the bin size ($|l_u - l_m|$). When s is large, the number of possible patterns is large too, and when bin size is large, the number of patterns represented by the same code will be large as well, and hence the probability of guessing the pattern will be smaller.

If all bits in the original BF are independent (i.e knowing some bits does not add any information about the other bits) then the probability of knowing the whole original BF from an aBF, $Pr(BF/aBF)$ could be evaluated as:

$$P(BF/aBF) = \prod_{i=1}^{t} Pr(\rho_i/\mathbf{c}_i) = \prod_{i=1}^{t} \frac{1}{\theta_{\mathbf{c}_i}}$$

where $t = k$ when all locations count is used, or $t = 2k$ when odd/even counts are used, and for number of partitions k.

Our observation is that, the counts of the 1s in each bloom filter partition do not reveal much information about their locations. In addition, the sensitive 1s counts (small counts and large counts) are masked by the codes generated for the bin ranges of counts.

To measure the amount of information leaked by our encoding scheme, we used the mutual information measure, which measures the amount of information the original partition and the corresponding code have in common. That is how much knowing the codes will reduce the amount of uncertainty about the original corresponding partitions, and hence will make it easier to guess their patterns.

Let X_i denote the random variable that represents the information in the original BF partition P_i, (i.e all possible patterns of 0s and 1s that could be in that partition in the entire set of BFs), and Y_i denote the information in P_i's corresponding cells in the anonymized and encoded aBF (i.e., all possible codes generated by our scheme for P_i). Then the mutual information I of the variables X_i and Y_i is defined as:

$$I(X_i, Y_i) = H(X_i) + H(Y_i) - H(X_i, Y_i)$$

where H is the entropy function, which could be computed by

$$H(X_i) = -\sum_{x \in X_i} p(x)\ log\ p(x),\ H(Y_i) = -\sum_{y \in Y_i} p(y)\ log\ p(y)$$

$$H(X_i, Y_i) = -\sum_{y \in Y_i} \sum_{x \in X_i} p(x, y)\ log\ p(x, y)$$

That is, $I(X_i, Y_i)$ could be computed by

$$I(X_i, Y_i) = \sum_{y \in Y_i} \sum_{x \in X_i} p(x, y)\ log\ \frac{p(x, y)}{p(x)p(y)}$$

This means that, as the number of values in each bin increases, the joint entropy (the uncertainty) $H(X_i, Y_i)$ of variables Y_i (of the aBFs) and X_i (of the original BFs) will increase, and hence the mutual information $I(X_i, Y_i)$ decreases, and hence a higher level of anonymity will be obtained. However the accuracy of the encoding decreases as the number of values in

each bin increases, and this will make our blocking method produces large size blocks, and eventually will lower the reduction rate (less efficient blocking).

In other words, our encoding is not deterministic function of the actual ones location of the partitions in the original BF (each code represents an interval of bin values, i.e., one-code to many-patterns mapping), and hence the information obtained from the codes is less than that obtained from the original BFs. By carefully crafting the bin thresholds, and the partition size, one can strike a balance between privacy, accuracy of the encoding, and the blocking performance, as we will show in our experiments in section 7.6.

7.5.3 Anonymity of the Block Patterns

Our encoding scheme will essentially be used for privacy preserving approximate blocking, (i.e., we encode the BFs as aBFs, then use the aBFs to do the blocking by creating a unique block pattern BP for each block). At some point of the record linkage process, the BPs will be shared (or matched) between the parties, which will leak some information about the aBFs in each block. The aBFs in turn leak some information about the original BFs (even though we proved the amount of information leaked is very small and could be controlled by setting the bin thresholds and partition size). Our goal is to build blocking scheme that, minimizes the leaked information as much as possible, while keeping the blocking efficient such that, most of the non-matchable BF pairs will be filtered out from the list of candidate record pairs (i.e., reduce the overall record linkage computations).

In our design, we build the block patterns (BPs) such that some additional randomness is added to these BPs (see section 7.4.3). That is, every bit set as '1' in the BP does not necessarily mean that all corresponding bits of the aBFs in that block must have '1'. There could be some aBFs, in that block, still have '0' in that bit location, however the number of these -set as 0- bits is below certain threshold Δ. The same applies to the '0' bits in the BP, i.e., a '0' in some bit does not necessarily mean that all aBFs of that block has '0' in these corresponding bits. If the adversary obtains the BP of some block, he will not be certain about the aBFs in that block.

This additional uncertainty could be controlled by the threshold Δ, that we use for setting the bits of each block pattern as '0' or '1', to achieve the desired level of anonymity. In addition, each party can increase/decrease the similarity threshold to have a minimum block size, or completely remove blocks of size below certain number of records and assign their aBFs to a block with most similar pattern.

Furthermore, the assignment of aBFs to blocks is not deterministic, because any aBF could be assigned to upto a pre-set number of blocks (overlapping threshold). This will add uncertainty about the aBFs represented by each block pattern, and also will improve the pairs completeness (reduce the false negative).

7.6 Experiments

As a proof of concept, we build the proposed scheme, and designed some tests to evaluate the accuracy of the encoding, the level of anonymity achieved, and the accuracy of our blocking using the anonymized Bloom filters (aBFs). In the following sections we discuss the details and results of those experiments.

7.6.1 The Datasets

We used two synthetic datasets of real personal identification information (names, addresses, etc) generated using the Mockaroo synthetic data generator (`http://www.mockaroo.com`). The data fields included in both datasets are First name, Last name, Social Security Number (SSN), Date of birth and Identification (ID). The data in each field in these datasets were randomly corrupted to emulate basic typographical errors (i.e., insertion, deletion, substitution and transposition). For each field, the corruption rate ranges from 10% to 20%. Each dataset has 10k records, and for each record, four differently constructed Bloom filters were created from different combination of the record linkage attributes. Each Bloom filter has 1k bits, and represents one of four different combinations of the data fields included in the dataset. The combinations used in this paper were adopted from the work of Kho et al.

[52] although the method is not dependent on the specific combinations selected as linkage variables.

Combination 1: First Name + Last name + Date of Birth

Combination 2: Date of Birth + SSN

Combination 3: Last name + SSN

Combination 4: Three Letter First Name + Three Letter Last Name + Soundex First Name + Soundex Last Name + Date of Birth + SSN

The value of each combination is created from the concatenation of the text values of the individual variables. We followed the standard protocols introduced in previous works to generate Bloom filter of each combination in each record [21, 76]. The Bloom filter generation process includes: 1) tokenizing the text value of the combination into bi-grams, 2) hashing clear-text bi-grams using a family of one-way hash functions (e.g., SHA 512) with added random string salts, and 3) mapping the resulting hash values to a Bloom filter. To ensure comparability, the Bloom filters from both parties A and B must be generated using the same mechanism and salts.

To maintain the linkage source of truth, all records that belong to the same individual have the same identification number. Both datasets have 6K records in common, and any similar pair of records in both datasets has the same value in the ID field. Therefore, the values of the ID field are used as the true answer to verify blocking performance.

We used synthetic data in these experiments for several reasons. First, synthetic data can be easily shared among method developers for method benchmarking and reproducibility. Second, synthetic data can be manipulated to simulate different levels of data quality (e.g., errors or missingness). Third, correct linkages are controlled with the use of synthetic datasets. Therefore, the use of synthetic datasets are necessary first steps in record linkage method development prior to testing on real data.

7.6.2 Accuracy of the Encoding

We expect the level of similarity between original BFs will also hold for their corresponding aBFs. That is, if any pair of original BFs has a high similarity, then we expect the anonymization and encoding mechanism to keep similar relationship among the corresponding pair of aBFs, and the aBFs will have high similarity too. The same principle applies to low similarity values. To evaluate the effectiveness of our anonymization and encoding scheme to preserve the similarity relationship among the original BFs after anonymization, we calculate the Mean Square Error (MSE) between the similarity values of the original BFs and the similarity values of their corresponding aBFs. For all possible pairs of original BFs (BF_i, BF_j), and their corresponding aBFs (aBF_i, aBF_j), $1 \leq i \leq n-1, i+1 \leq j \leq n$ we define the MSE as follows:

$$MSE = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(Sim(BF_i, BF_j) - Sim(aBF_i, aBF_j)\right)^2$$

Using the four bloom filters of each record in the dataset, we ran a set of tests with different settings for the *number of partitions*, and a set of different *encoding bins*. The number of partitions used are 10, 20, 50, and 100, which produce partitions with sizes 100, 50, 20, and 10 respectively. The set of bins for each experiment e is represented by a list of thresholds $b_e = [q_i : 0 \leq q_i \leq 1, q_i < q_{i+1}\}$ that define the separation between the bins, and the number of bins will be $|b_e|+1$. Each threshold in b_e represents the percentage of the $maxValue$ of 1s in each bin, for example if b_e is given as [0.5] then the size of this set is 2 bins, and the first bin holds 1s count values from $minValue$ and upto (not including) $0.5 \times maxValue$, and the second bin holds 1s count values from $0.5 \times maxValue$ and upto $maxValue$ inclusive. Where $minValue$ is the minimum 1s count a partition can have (e.g. $minValue = 0$), and $maxValue$ is the maximum value a partition 1s count can have (e.g. it could be the size of the partition, assuming all bits can be set as '1'). The sets of different bins used in this work are as follows :

b1 [0.5]: two bins [0-0.5), and [0.5-1.0]

b2 [0.75]: two bins [0-0.75), and [0.75-1.0]

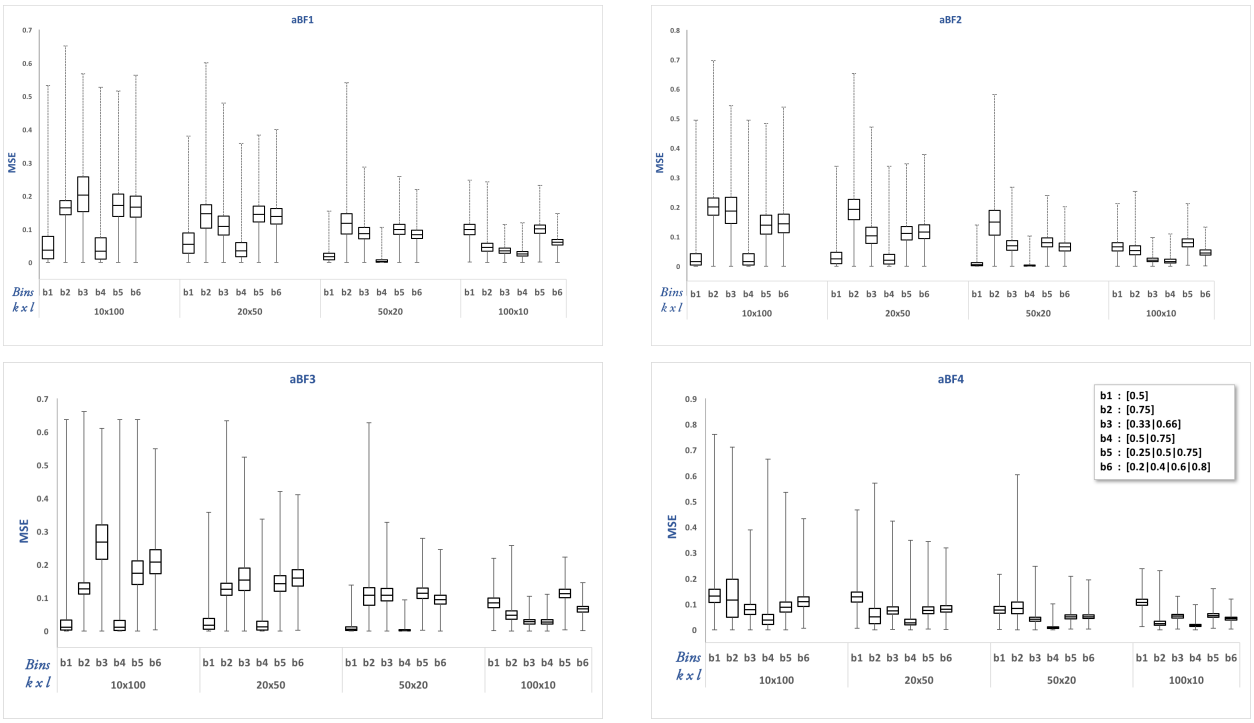


Figure 7.7: MSE values obtained using different encoding configurations for each of the four Bloom filters. Where k is number of partitions, and l is the partition size in bits.

b3 [0.33,0.66]: three bins [0-0.33), [0.33-0.66), and [0.66-1.0]

b4 [0.5,0.75]: three bins [0-0.5), [0.5-0.75), and [0.75-1.0]

b5 [0.25,0.5,0.75]: four bins [0-0.25), [0.25-0.5), [0.5-0.75), and [0.75-1.0]

b6 [0.2,0.4,0.6,0.8]: five bins [0-0.2), [0.2-0.4), [0.4-0.6), [0.6,0.8), and [0.8-1.0]

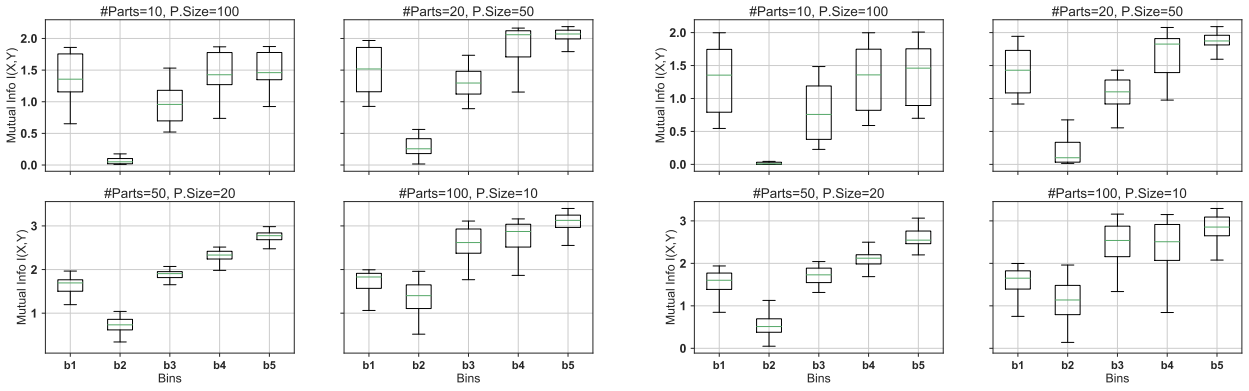
The results of this experiment, shown in figure 7.7, confirm our analysis in section 7.5.1, where the accuracy of encoding improved when the partitions are small, and it also increased with the distribution and number of bins. For example, the lowest MSE resulted from the smallest partition size, i.e., when $k \times l = 100 \times 10$, where k is number of partitions, and l is partition size in bits. Also bin sets $\{b_3, b_4, b_5, b_6\}$, give better results than bin sets $\{b_1, b_2\}$, however this will negatively affect the anonymity levels as we will see in the next section.

7.6.3 Evaluation of the Anonymization Scheme

In this experiment we measure the amount of information that may be inferred from the aBFs. We calculated the mutual information between the original BFs and their corresponding aBFs using the steps discussed in section 7.5.2. We used different setting for our anonymization and encoding scheme, to visualize the relationship between security and accuracy of the encoding. Since we have four BFs, and each of them is constructed from different set of attributes, we applied the technique using each of the four BFs in order to measure the amount of information leaked by each of them. The results shown in figure 7.8 comply with our analysis in section 7.5.2, where the mutual information decreases when block size is large and the number of bins is small. The mutual information is also affected by the bin thresholds, which determine the range of values each bin might have. However, using our scheme the mutual information was small in all the test settings.

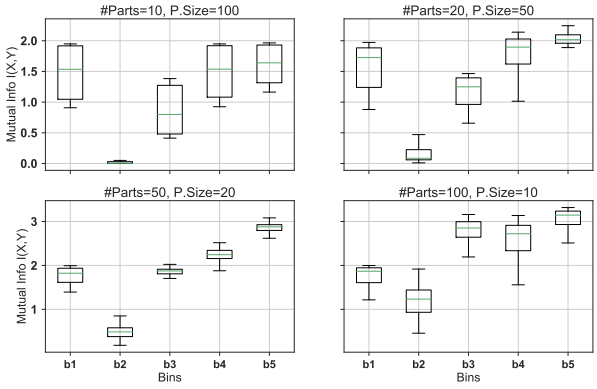
7.6.4 Evaluation of the Blocking Quality

In this experiment, we measure the effectiveness of our blocking method by using two commonly known measures. The reduction rate (RR) and the pair completeness (PC). Reduction rate is computed as $RR = 1 - c_p/pp$, where c_p is the number of candidate record pairs produced by the blocking scheme, and pp is the number of all possible pairs, i.e., $pp = |T_A|.|T_B|$ for the two datasets T_A, and T_B. Pair completeness is computed by $PC = cp_m/p_m$, where cp_m is the true matches among the c_p candidate pairs, and p_m is the true matches among all pairs. In our blocking scheme, we allow any aBF to be assigned to a single or multiple blocks (blocks overlapping) based on its similarity to the blocks patterns BPs of these blocks. The configured number of maximum overlap and the minimum similarity threshold control how the aBFs are assigned to the blocks. For example, if the maximum overlap is 5, the minimum similarity threshold is 0.85, and there is an aBF which is similar to 10 BPs with values greater than 0.85, then it will be assigned only to the blocks with the top 5 similarity values. In addition, every aBF is assigned to at least one block (the block with highest similarity value, even if it is below the threshold). Any block with small number of

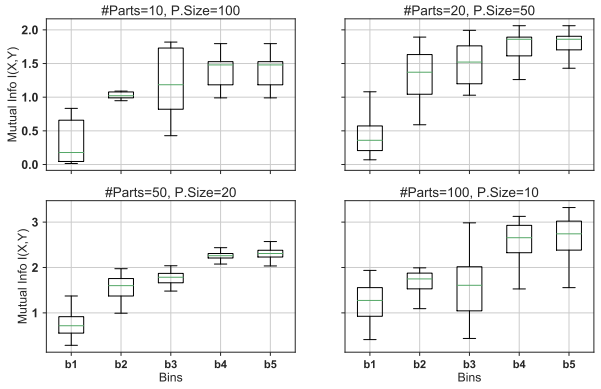


(a) BF1

(b) BF2



(c) BF3



(d) BF4

Figure 7.8: Mutual information shared between anonymized-encoded (aBFs) and the original BFs.

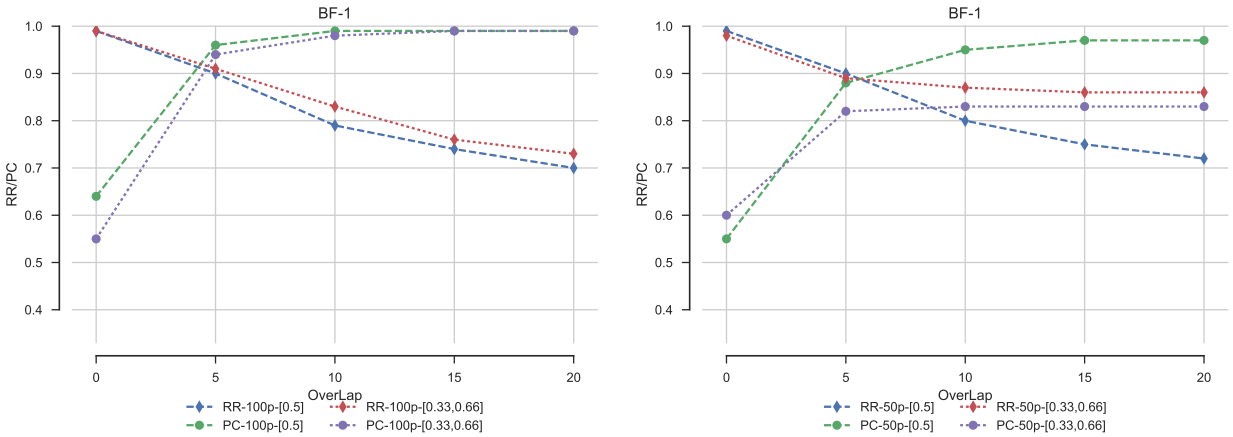


Figure 7.9: Pair Completeness (PC), and Reduction rate (RR) for different block overlap settings.

aBFs could be removed, and its aBFs are assigned to other blocks with high similarity to their BPs. Figure 7.10 shows the RR and PC values using the selected encoding configurations and variable block overlaps. The experiment is done using each of the four different aBF representations of the records. From the results, our blocking method achieved high pair completeness, especially with high overlap. The reduction rate is slightly degraded with high overlap setting, however it never get below 60%. An optimal results of reduction rate and pair completeness could be achieved with reasonable overlap, for example when overlap was 5 and using data of BF1, we achieved 90% of reduction rate, and 96% of pair completeness, as shown in figure 7.9.

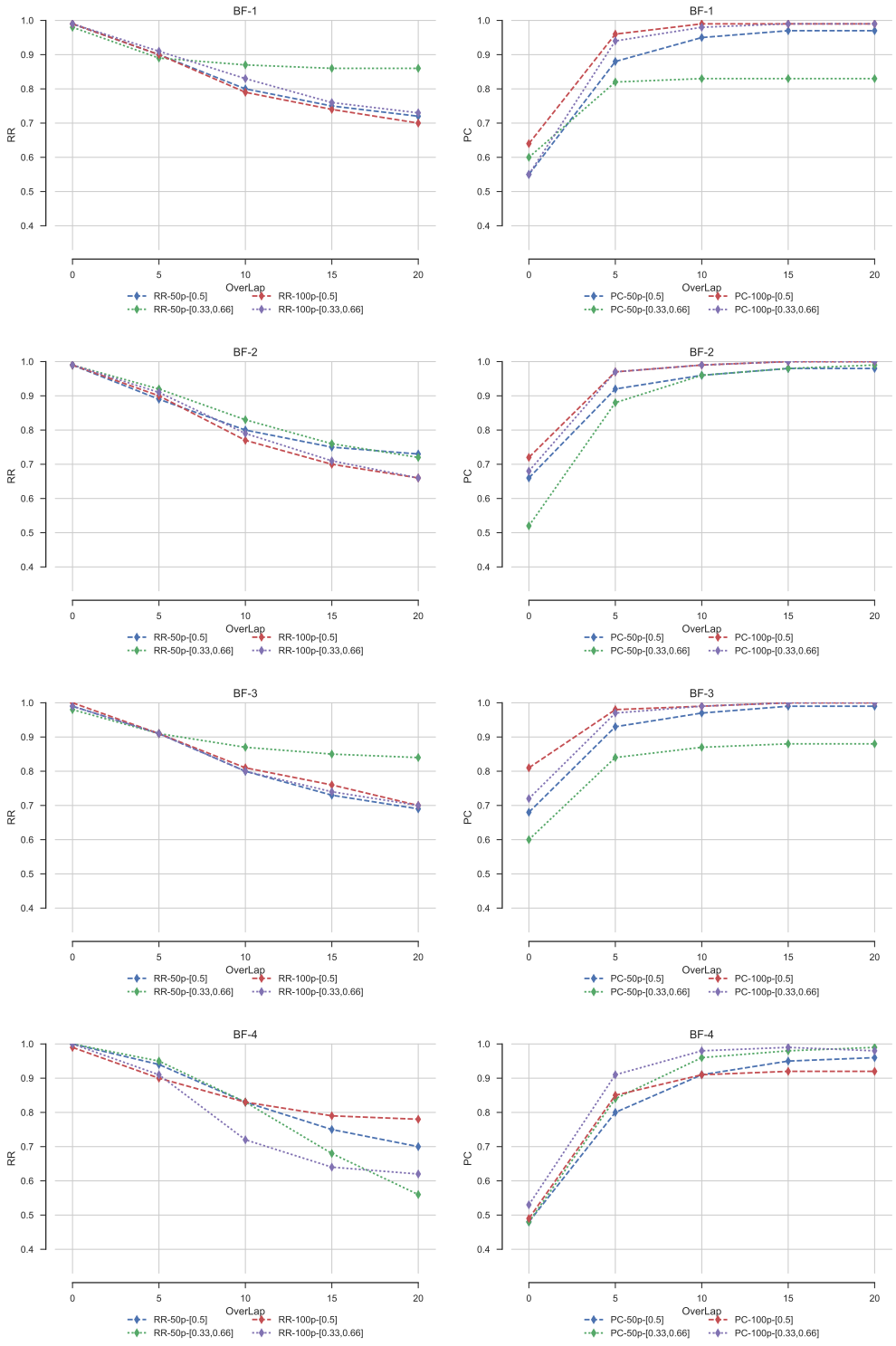


Figure 7.10: Blocking effectiveness measured by Reduction Rate (RR), and Accuracy measured by Pair Completeness (PC) values for the tested configurations and using each of the four different aBF representations of the records.

Chapter 8

Large-scale Distributed Privacy preserving probabilistic record matching using Bloom Filters and Garbled Circuits

Using Garbled Circuits, we showed that it is possible to build a system that allow two parties to compute the DCs of their corresponding encoded records without actually sharing these encoded records. Because there are no BFs exchanged between the parties, none of them will be able to launch any attack on the other's BFs, and hence the privacy of their patients is protected. GCs are very expensive secure computation environments in terms of computation and communication, and hence have inefficient performance and limited scope of applications. In order to build an efficient garbled circuit based record linkage system, we proposed some novel heuristic approaches to improve our GC designs, and build an efficient GC module to perform the DC computations in a reasonable time. In this work, we further improve upon our scheme presented in chapter 6 by making it more efficient. We designed a scalable record linkage system that works in a distributed computation environment to perform the record linkage process in a parallel fashion. Our evaluation results show that the designed system is very effective and efficient.

8.1 Contribution Summary

Our objective in this work is to evaluate the feasibility of our record linkage garbled circuit approach GCPPRL, presented in chapter 6, in practice using high capacity compute engines. Our garbled circuit record linkage approach, leveraged Yao's garbled circuits to compute similarity between any pair of Bloom filters without the need of a trusted third party (TTP), and without revealing these Bloom filters. In this work, we further improved our GCPPRL

method by using parallel processing framework in distributed processing environment, like local network or Google Cloud Compute Engines (GCCE). Using blocking, data records are indexed as separate independent groups (called blocks, or chunks), and then the blocks could be processed simultaneously.

Linkage between corresponding pair of blocks from each party, is processed by a worker on network node or VM (virtual machine) instance. Each node/VM will run a number of workers according to the availability of computing resources. The more resources we have, the more blocks the system will be capable to process. We tested the improved process in various scenarios with variation in hardware and software configurations.

8.2 Building a scalable distributed PPPRL System

Blocking allow parties to assign their records to a number of blocks (subsets), each have distinguished block id. Then only records of the corresponding blocks (blocks with the same id) of both parties are matched against each other. Since blocks are independent from each other, the record linkage process could be performed on these blocks in parallel to reduce the processing time. Another computation reduction method we used in order to reduce the overall GC computation is elimination. Since each record is represented by multiple Bloom filters, sometimes it is sufficient to consider the a pair of records as a match if one of their Bloom filters is similar with high DC value, for example their DC ≥ 0.98. To reduce the block sizes during the matching process, we added elimination process to remove the records matched in the previous steps (either by the previous Bloom filters or the previously processed blocks) if their matching DC scores are above pre-set elimination threshold. This elimination threshold works as an indicator for how likely the matched records are not a false positive, and hence no need to compare them with other records, or even compare the other Bloom filters of these two records.

8.2.1 System Overview

The scalable design of the system is based on the blocking technique. Since blocks with different Ids are independent of each other, multiple blocks could be processed simultaneously. That is, both parties can launch the matching process of multiple blocks at the same time, i.e in parallel. We designed and build a system that allow the Generator and the Evaluator to perform the matching process of the corresponding blocks in a distributed fashion, and hence utilize their computation resources to the maximum and reduce the whole record matching time.

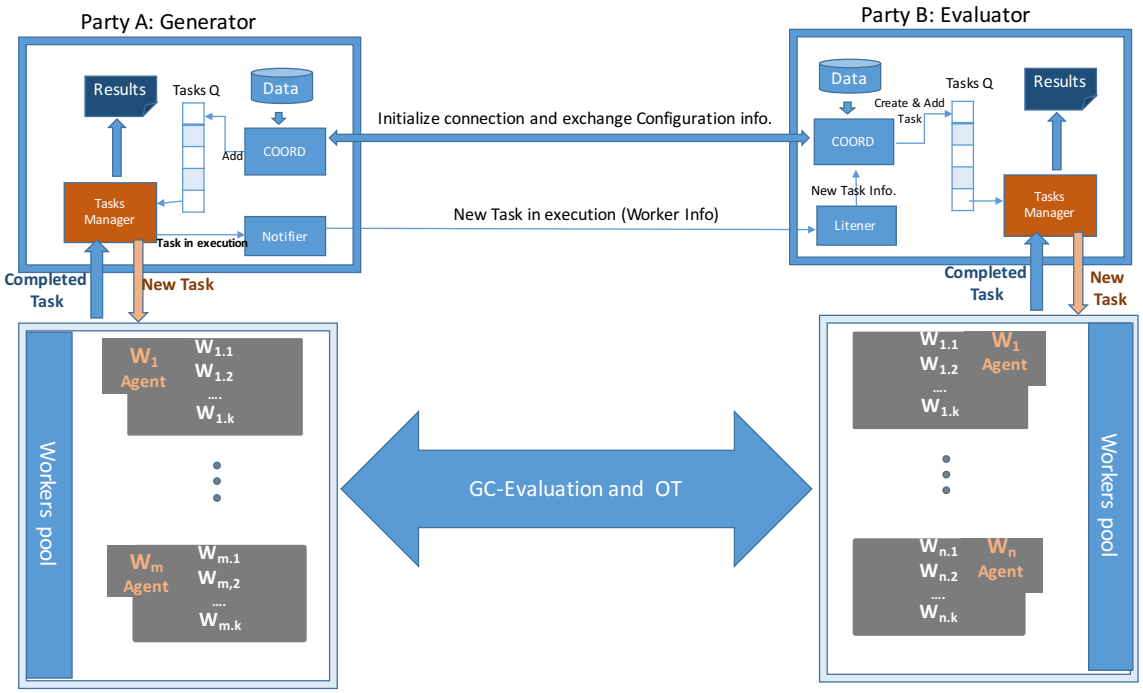


Figure 8.1: Schematic diagram of the distributed record matching system components

The system structure is shown in figure 8.1. The system consists of three modules, coordinator, tasks manger, and work agents. These modules could run on a single machine or on different machines. Based on the resources available to it, each party identifies a set of machines as work agents to run the matching process. Each party also needs to identify one machine as the coordinator, and one or more machines as tasks manager(s). During the matching process, each party assigns a block or set of blocks to a work agent or set of work

agents, running on the same or different machines. Each work agent dynamically (based on the current work load and available resources) runs one or more processes dubbed workers. Then workers that process the same block on both parties' side communicate directly with each other to perform the matching process on these blocks of data assigned to them. The *notifier* from the Generator party's side will notify the *listener* on the Evaluator party's side whenever a new block is assigned to a worker. The *listener* in turn, extract, from the notification message, the needed information to identify the Generator party's worker and allow some Evaluator-side worker to communicate with that worker.

In the following sections we provide some technical details about the modules of the distributed system we created to parallelize the matching process.

8.2.2 The coordinators

The coordinator (say, at party A's side) starts the matching process, arranges the tasks assignments, and collects the results. The Generator starts by running its coordinator to accept connections from the Evaluator's coordinator (at party B's side). When Evaluator coordinator connects to the Generator coordinator, they exchange some information about the matching processes. This information includes, the Blocks sizes and their Ids, the size of the workers bool (number of workers on each side), and the blocking schemes.

Then the Generator coordinator will loop over the blocking schemes, groups/partitions its data based on the block IDs of the selected blocking scheme, and creates new task(s) for each block ID. The task is the GC code to compute the DC with a subset (block) of the data that has this block ID. The coordinator adds these tasks to its tasks queue (Q) for execution on the workers bool.

When there are no more tasks to execute, each party will wait for the completion of all the pending tasks of the current blocking scheme, then proceed to the next blocking scheme. If elimination is enabled, each party will filter out the matched records with DC $\geqslant$ the pre-set elimination threshold, then loop to process the next Blocking scheme.

Following are the Generator and the Evaluator coordinators algorithms

Algorithm 5 Generator Coordinator (Party A)

```
procedure GenCoord
    Inputs:
    • BlkSchemes                                   ▷ Blocking schemes to use
    • maxBlkSize                                   ▷ maximum block size
    • Data                                         ▷ Party A's Data Records BFs
    TM ← connectToTaskManager()
    TM.InitWorkersPool()
    Ws ← TM.getWorkersInfo()
    EvalCoord ← connectToEvalCoord()               ▷ Wait for Eval. Coord. to connect
    BlksInf ← getDataBlockedUsing(BlkSchemes,Data)
    ComBlkInfo ← XchgBlokingInfo(EvalCoord,BlksInf)
    for s ∈ BlkSchemes do
        for blk ∈ ComBlkInfo do
            DataBlk ← getDataBlocks(s,blk,Data)
            subBlks ← partitionBlk(DataBlk, maxBlkSize)
            for sblk ∈ subBlks do
                taskId ← Blk.Id + sblk.part
                nTask ← CreateNewTask(taskId, Ws, GCGenTaskCode,sblk, blk.id)
                sentTasks ← sentTasks + TM.addTask(nTask)
                results ← results + getCompletedTaskResults()
                if TM.Q is full then
                    sleep(100)
                end if
            end for
        end for
    end for
end procedurereturn
```

Algorithm 6 Evaluator Coordinator (Party B)

procedure EVALCOORD

Inputs:

- BlkSchemes ▷ *Blocking schemes to use*
- maxBlkSize ▷ *maximum block size*
- Data ▷ *Party B's Data Records BFs*
- GCconnInf ▷ *Gen. Coord. connection info (IP,Port, auth,..)*

```
TM ← connectToTaskManager()
TM.InitWorkersPool()
Ws ← TM.getWorkersInfo()
GenCoord ← connectToGenCoord(GCconnInf)          ▷ connect to Generator Coord.
listener ← startListener(GenCoord.notifier)
BlksInf ← getDataBlockedUsing(BlkSchemes,Data)
ComBlkInfo ← XchgBlokingInfo(GenCoord,BlksInf)
while (! Done) do
    nProcBlkInf ← listener.getNextProcessedBlkInfo()
    DataBlk ← getDataBlocks(nProcBlkInf.s, nProcBlkInf.Id, Data)
    subBlks ← partitionBlk(DataBlk, maxBlkSize)
    for sblk ∈ subBlks do
        taskId ← nProcBlkInf.Id + sblk.part
        nTask ← CreateNewTask(taskId, Ws, GCEvalTaskCode, sblk, nProcBlkInf.Id, nProcBlk-
Inf.GenWorkerInfo)
        sentTasks ← sentTasks + TM.addTask(nTask)
        results ← results + getCompletedTaskResults()
        if TM.Q is full then
            sleep(100)
        end if
    end for
end while
end procedurereturn
```

8.2.3 The Work agents

Before the data processing starts, each party initializes its processing environment (bool of workers) by allocating some machine(s) to perform the matching process, and runs a work agent on each machine. These work agents communicate with the task manager, and starts/stops worker processes based on demand and/or the required settings (configurations). Each work agent can run multiple workers to process multiple tasks at the same time, however the performance will depend on the available resources on that machine.

Algorithm 7 Agent

```
procedure AGENT
    Inputs:
    • nWs                                   ▷ number of workers
    • TMcon                                 ▷ Task Manager to work for
    TM ← connectToTaskManager(TMcon)
    wList ← InitWorkersPool(nWs)
    TM.send(AgentId,wList)
    taskListener ← startTaskListner(TM)
    while (1) do
        nTask ← taskListener.getNextTask()
        freeWorker ← wList.getNextFreeWorker()
        result ← freeWorker.run(nTask)
        TM.send(nTask.Id, result)
    end while
end procedurereturn
```

8.2.4 The Tasks managers

The task manager of the generator will pick the tasks from the coordinator queue and send them to the workers pool via the work agents. The task manager keeps track of the tasks (status: waiting, running, cancelled, completed, terminated) being executed, and the workers pool occupancy. The tasks on the generator side will wait in the "waiting mode" at the assigned workers until the corresponding evaluator worker start the process. The task manager must collect the results of the completed tasks, and forward them to the coordinator, re-run the cancelled or crashed tasks.

Algorithm 8 Task Manager

```
procedure TaskManager
    AGs ← listenToAgentConn()
    Coord ← waitForCoordConn()
    Q ← Coord.tasksQueue
    while (Q.isNotEmpty() and (nw=AGs.getNextFreeWorker()!=null) ) do
        nTask ← Q.next()
        sentTasks ←sentTasks+ sendTaskToWorker(nw)
        if ( Coord.name == Generator ) then
            notifyCoord(Coord, nTask.Id, nw, "task is in exec")
        end if
        taskId, result ← getCompletedTasks(sentTasks)
        Coord.send(task.Id, result)
    end while
end procedurereturn
```

8.2.5 The Generator Notifier

Whenever a task is started on any of the workers the information about that worker (its IP, and Port number,) and the block ID of the data in the task is sent to the coordinator's notifier. This information is needed by party B (evaluator) to know which block is being processed and by which worker. For example, each worker on the party B's side, who assigned to process certain block, need the IP address and port number of its peer worker (on A's side) processing that block. The generator notifier will send this information (blocks to workers assignments on A's side) to the evaluator listener as soon as they became available.

8.2.6 The Evaluator Listener

The Evaluator listener will continuously listen for notification from the Generator about new tasks under execution, and extract the worker information and the block ID from these notification and forward them to its coordinator. The coordinator will create new task with the evaluator GC code and the subset of the data for this block ID, and send it to its tasks queue. The task manger of the evaluator will continuously execute the tasks in the queue based on the available workers.

8.2.7 Creating and distributing the Tasks

Each task is a wrapped GC-RL code with the a subset of the dataset (the block of data) to be linked (matched). The generator loops through the blocking schemes and use the block Id of each block to create the new tasks. If the block size is larger than a pre-set value, the block is partitioned and the new tasks are created accordingly. The id of the new task consists of the blocking scheme number, the block id, and a sequence number if the block was further partitioned. The task id is used to communicate information about the task (like its status: running, completed, terminated, etc.) locally (between the task manager and the work agents) or with the other party (evaluator) to notify it about the tasks being executed at the generator side. Once a task created on the generator side, it is placed on the execution queue. The task manager will continuously check the workers status, and picks the tasks from the queue and send them to the available workers. When a generator worker receives a task, it executes its code, and prepares the data for the GC computation, sends a "Task is executing" message to the notifier with the task id, and waits for its peer evaluator worker to connect and start the GC-RL process. The notifier immediately, sends a "task in execution" message, with info about the worker executing it, to the evaluator listener. The evaluator listener sends the received task information to the coordinator, which extracts information about the blocking scheme, block id and the worker, and then creates a new evaluator task and places it in its execution queue. The task manager of the evaluator will continuously pull tasks from the queue and send them to the available evaluator workers. The evaluator workers will unwrap the task object, connect to its generator peer, and start GC-RL process. When a task completed at each side, the task manager will collect the results and store them on the coordinator, mark the task as completed, and move to execute other tasks. If a task crashed or interrupted (for any reason, e.g. connection failure) the worker agent will notify the tasks manager, which will notify the other party and re-execute the failed task.

8.3 Experimental Evaluation

We ran some experiments without distribution to evaluate the effect of Block sizes and the elimination technique on the performance, then we deployed our system on a local network, and Google Cloud. Our results show that the parallel process of the garbled circuits greatly improved the runtime in the local network and the cloud-based infrastructure. In addition to single compute engine, a cluster of compute engines can also be leveraged to reduce the runtime of data linkage operations. The capacity of cloud-based infrastructure will overcome the trade-off between security and efficiency, allowing more sophistical methods to be implemented and used in practice.

8.3.1 Effect of Block sizes distribution on the non-Distributed version

Blocking helped in dividing the problem of comparing large datasets against each other into a number of smaller problems of comparing smaller corresponding blocks. However, the resulted blocks vary in sizes and cannot guarantee all candidate for linkage record pairs to be in the corresponding blocks. On one hand, The variations in block sizes caused the difference in performance between the blocking schemes as can be seen in figure 8.2. Each blocking scheme produces different block size distribution which shown in figure 8.3. For example, Year of birth (scheme 3), took the longest time because it has the largest block sizes. On the other hand, the assignment of candidate records to different blocks will cause some false negatives (FN). If the blocking is done deterministically, the slightest difference in the values of blocking variable of any record pair will make them assigned to two different blocks, and consequently miss the comparison when such deterministic blocking scheme is used.

8.3.2 Effect of Elimination on the non-Distributed version

In this test we ran to experiments, one to evaluate the effect of the elimination threshold value, and the other to evaluate the effect of using elimination on both time and accuracy.

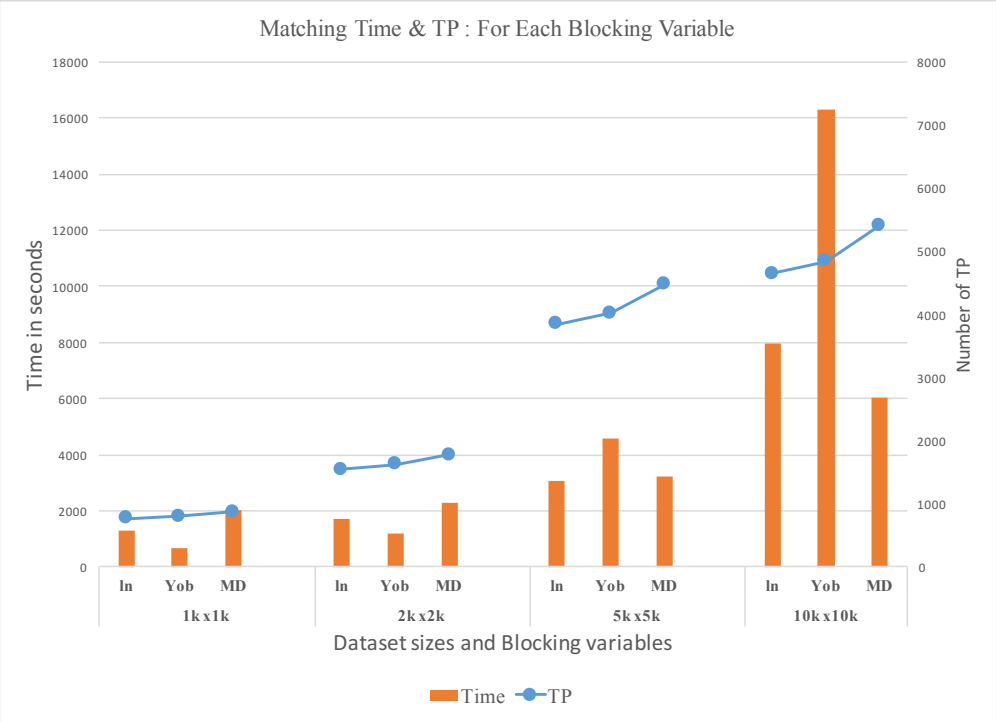


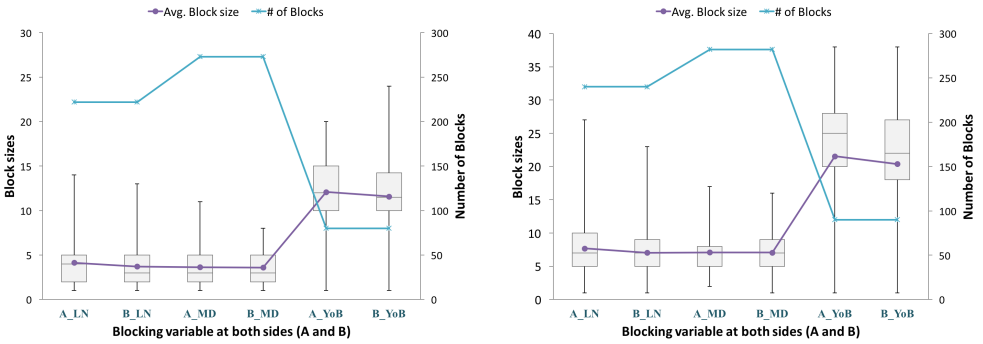
Figure 8.2: Matching process time and TP of optimized non-distributed GC-RL using each Blocking variable and DC threshold = 0.9.

- **Effect of elimination thershold:**

  In this experiment, we used the 10k x10k datasets to investigate the effect of changing the elimination threshold on time and accuracy. The results in figure 8.4 show that the time has slightly increased when the elimination threshold increased. This is as expected because the higher the elimination threshold means less records will be removed from the data set for the subsequent iterations. The FP was similar and high for both elimination thresholds because of the low (0.85) DC used, however the TP decreased for the low Elimination threshold (0.86) because some of the record eliminated in the first iteration were missed in the subsequent ones.

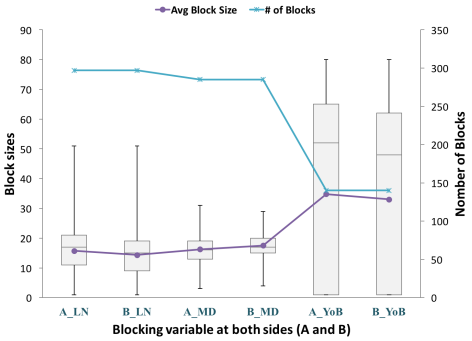
- **Effect of using Elimination :**

  In this experiment we compare the matching time and accuracy of the results using elimination with the results of not using elimination. Figure 8.5 shows the results of this comparison, where the elimination technique reduced the total matching time

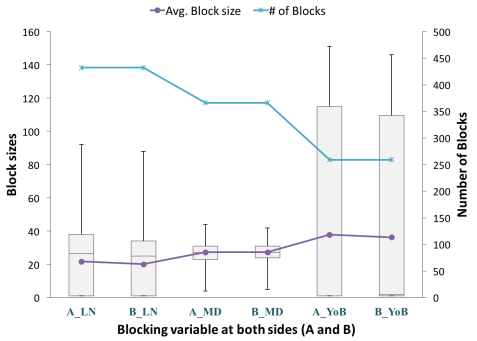


(a) 1k

(b) 2K



(c) 5K



(d) 10K

Figure 8.3: Block size distribution for each blocking variable (Dataset sizes 1K – 10K).

substantially (almost to the half in the large problem 10k˙10k). In figure 7, we can see that the TP is not affected at all with the elimination and the FP is slightly improved.

8.3.3 Distributed System Results

After blocking, each blocking scheme produces different block sizes, and the size of some of the produced blocks is still large for a single worker to process quickly. To solve this, we set a maximum block size threshold, such that any block larger than that size is arbitrary divided into smaller sub-blocks of sizes smaller than or equal to the maximum block size. By doing so, each sub-block need to be matched against all other sub-blocks of the corresponding block on the other side, which will increase the number of blocks but the number of computations will be the same. The only benefit of this, is the computations of the large tasks (with large blocks) could be parallelized (by executing multiple small sub-tasks) when there are some computation resources available, and will be completed faster.

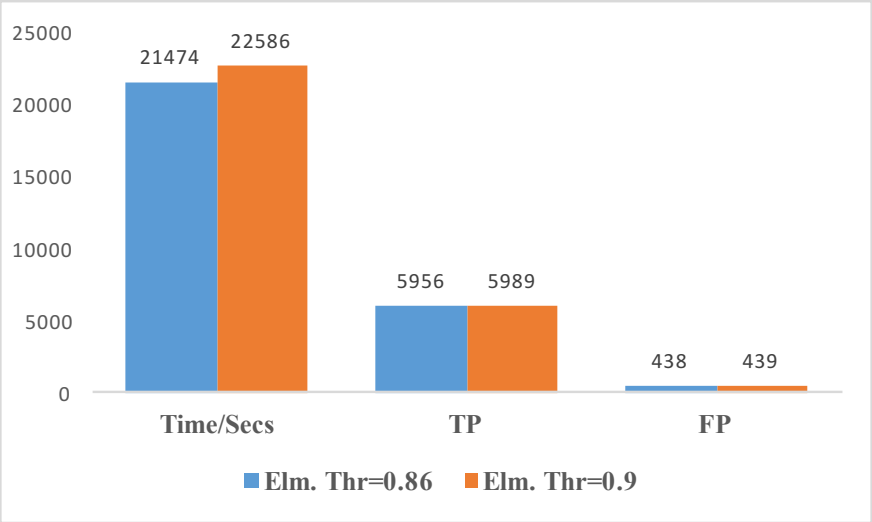


Figure 8.4: Effect of elimination threshold on time, true positives TP, and false positives FP.

8.3.3.1 Test 1: 10k x 10k on Local network

Our first test of the system was on our computer science department network, where the machines were not dedicated to our system, and all the processes were run with low priority. The purpose of this test is to see how the system perform in uncontrolled environment. This experiment was executed on the network with the following configuration:

- **Generator:**

  Coordinator: on Maserati HP-Z440-XeonE5-1650v3 6x3.5Gh 32Gb Linux(Fedora)

  TaskManager: on eggs HP-DL580-G7-XeonE7-4830 16x2.13 128Gb Linux(Fedora)

  Agents:

  1. eggs HP-DL580-G7-XeonE7-4830 16x2.13 128Gb Linux(Fedora) x 7 workers
  2. coconuts HP-Z800-XeonE5645-SAS 12x2.4G 96Gb Linux(Fedora) x 7 workers
  3. mustang HP-Z440-XeonE5-1650v3 6x3.5Gh 32Gb Linux(Fedora) x 7 workers

- **Evaluator:**

  Coordinator: on bacon HP-DL580-G7-XeonE7-4830 16x2.13 128Gb Linux(Fedora)

  TaskManager: on lamborghini HP-Z440-XeonE5-1650v3 6x3.5Gh 32Gb Linux(Fedora)

  Agents:

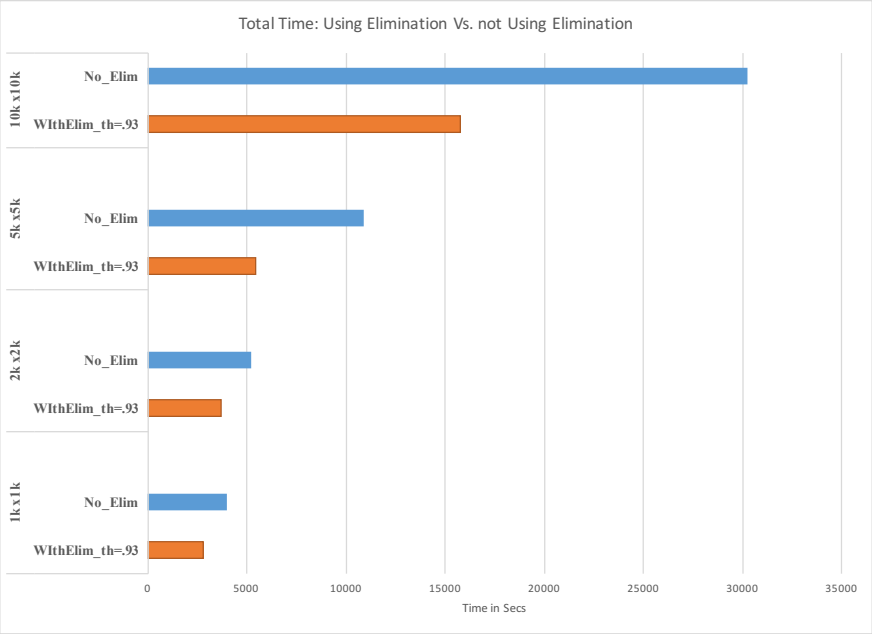


Figure 8.5: Effect of using elimination, with threshold of 0.93, on total time of linking 10k x 10k datasets.

1. bacon HP-DL580-G7-XeonE7-4830 16x2.13 128Gb Linux(Fedora) x 7 workers
2. bananas HP-Z800-XeonE5645-SAS 12x2.4G 96Gb Linux(Fedora) x 7 workers
3. corvette HP-Z440-XeonE5-1650v3 6x3.5Gh 32Gb Linux(Fedora) x 7 workers

The total time taken to link the 10k × 10k datasets, using parallelization in the previously mentioned environment, was 2199 seconds (about 36 min), while the non-parallelized garbled circuit took about 30285 seconds (8.5 hours) to link the same datasets. The linkage results and the time taken are shown in table 8.1. As we can see from this table, the total processing time is substantially reduced using the parallel processing, which makes the garbled circuit a viable solution to the privacy preserving record linkage.

Table 8.1: results of linking 10k × 10k datasets on local network with 21 workers on each side.

Blocking Var.	Time/Secs	Matches Found	Unique Matches	False Positives
ln	811.779	4658	4643	15
YoB	1068.069	1437	1436	1
MDoB	319.448	947	946	1
Total	2199.296	7042	5948	16

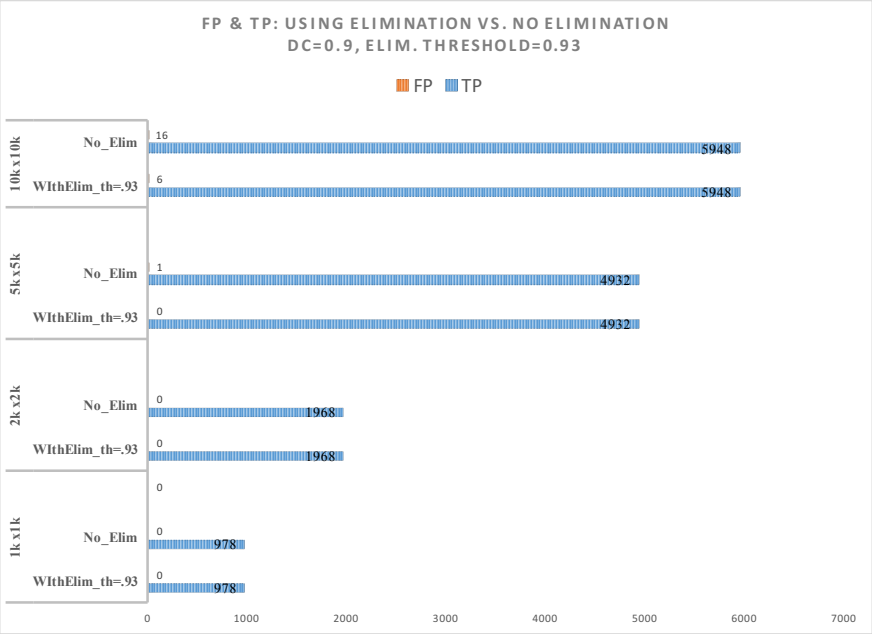


Figure 8.6: Effect of using elimination, with threshold of 0.93, on TP and FP of linking 10k x 10k datasets.

8.3.3.2 Test 2: 10k x 10k on Google Cloud

We deployed our system on Google cloud and run some experiments using the same two synthetic 10K datasets we used before. Each record is represented by an Id, and four different Bloom filters (each Bloom filter represent a combination of the PII). We used the same blocking schemes we used before.

We run our tests using 12 different software and hardware configurations. The parameters of the configurations include: number of CPU cores (range: 4 to 32), memory size (15GB – 28.8GB), number of workers (6-61), and maximum block size (50-200) records.

Our experiments results are shown in figure 8.7. With the minimum configurations (config 0: 4 cores and 15GB of memory, 6 workers), it took 8,062.4s (2.25 hours) to complete the linkage process of the two datasets, while it took only 1,271.6s (21 minutes) when the maximum configurations (config 11: 32 cores and 28.8GB memory, 61 workers) of the 2 VMs is used. When we used more VMs (config 12: 4 on each side), the time drops to 854.0s (14 minutes). The configurations details and the tests results are shown in table 8.2. The results showed that increasing number of threads or change the chunk size without providing more

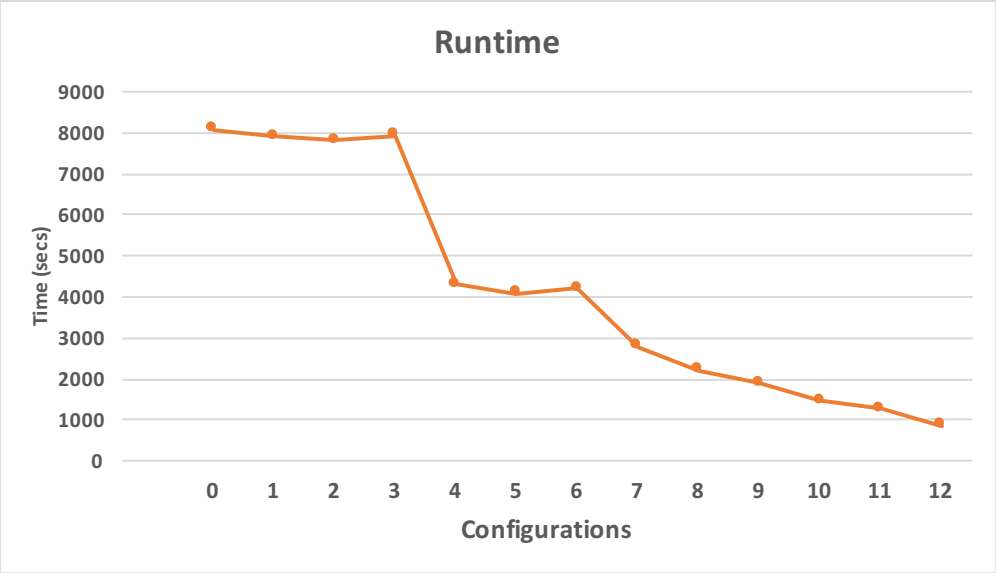


Figure 8.7: Total record linkage time for 10k ×10k datasets, using different configuration of the scalable GCRL system deployed on Google cloud.

CPU cores and memory did not improve the efficiency. Efficiency is improved on average by 39.81% and 79% when the number of cores and memory on the both sides are doubled. The CPU utilization on both sides is maximized (near 100%) when the computing power of the generator is double of that of the evaluator.

Table 8.2: System performance using the 12 different hardware/software configurations.

Configs	# CPUs		Mem	# VMs	used Mem		%CPUs used		Network (MB)		# workers		Max	Runtime
	Gen	Eval	(GB)		Gen	Eval	Gen	Eval	Gen	Eval	Gen	Eval	Blk.size	(sec)
0	4	4	15	2	2	2	100.00	40.00	22.99	22.69	6	6	200	8062.4
1	4	4	15	2	2	2	100.00	41.00	22.72	22.40	11	11	200	7914.6
2	4	4	15	2	2	2	100.00	45.65	23.07	23.17	11	11	50	7807.3
3	4	4	15	2	2	2	100.00	44.30	22.30	22.20	21	21	50	7928.9
4	8	4	15	2	5.3	5.2	100.00	97.00	42.80	40.00	21	21	50	4298.1
5	8	4	15	2	5.3	5.2	97.46	94.65	45.28	45.35	21	21	100	4071.3
6	8	4	15	2	5.8	5.2	99.18	93.25	44.19	43.79	21	21	200	4191.6
7	12	6	15	2	6.6	6	99.40	99.49	66.87	67.33	21	21	100	2773.2
8	16	8	15	2	7	5.9	99.47	98.95	83.69	67.33	21	21	100	2214.1
9	32	16	28.8	2	7.7	7.2	97.70	64.15	93.34	93.16	21	21	100	1898.1
10	32	16	28.8	2	12.6	12	97.70	80.45	99.34	99.16	41	41	100	1454.1
11	32	16	28.8	2	13	7.8	96.80	95.60	101.00	101.75	61	61	100	1271.6
12	32	16	28.8	4	10.3	10.8	95.85	99.56	122.6	121.18	40	40	100	854.0

Chapter 9

Conclusion and Future Work

In this research, we introduced the problem of privacy preserving linkage and sharing of sensitive data, and we highlighted the record linkage as one challenging application of such important research. We investigated the state of the art of the field, the challenges in privacy preserving linkage and sharing of sensitive information, and the techniques and solutions targeted to this problem. Throughout this work we mainly focussed on two main privacy preserving data linkage and sharing techniques; using a third party services, and direct two parties secure computations. We used the third party technique because of its scalability, and we introduced a novel technique to protect the data while it is being processed by the third party. The secure two parties computations technique is very computationally extensive operation, however the data do not leave each party's perimeter, which makes this technique more secure. We used the service of semi-trusted third party to allow multiple parties to share their data with researchers without revealing their actual data, and without using shared encryption keys. We used garbled circuits for two-party secure computation to design an efficient two-party record linkage protocol. We introduced new techniques to improve the computation of our garbled circuits design, and make the protocol more efficient. We also introduced the problem of privacy preserving probabilistic matching, and its applicability to the record linkage problem. Data is not always consistent, and it is very important to consider the inconsistencies in the data that might occur during data acquisition procedures at each party. We designed a privacy preserving probabilistic record linkage protocol based on the similarity of the linkage attributes. This protocol increased the chances of finding matched records when the encrypted linkage attributes are not perfectly the same. We provided proof of concept and experimental evaluations for our designs to demonstrate their efficiency and applicability.

We also investigated blocking techniques to minimize the number of comparisons required during the usage of secure matching processes for faster record linkage, and proposed a new blocking method that preserves privacy. We provided new Bloom filter anonymization technique, which we used to build our hierarchical blocking scheme.

Blocking significantly reduced the garbled circuit record linkage (GC-RL) computations overhead, and allow the distributed processing of the data. Using our design we show that GC-RL is a viable solution for parties who do not want to exchange their records yet want to perform record linkage on their data. The deployment of the system on the Cloud allowed the parties to process the data fast, and allocate the resources as needed.

For future work, we plan to 1) further improve our semi-trusted third party protocols and make them robust to collusion and, 2) adapt our privacy preserving record linkage protocols to new applications, like secure storage de-duplication. 3) We also plan to come up with new designs to reduce the setup phase steps in our first protocol, and develop new protocols using bi-linear pairing, where we can get rid of the setup phase. 4) Improving the matching quality results by incorporating automation methods for selecting the best matching criteria and properly adjusting the similarity thresholds and weights of the attributes. 5) We also plan to further study the effect on using different combinations of linkage variables. 6) Further develop and optimize the new blocking technique and try different clustering strategies. 7) Use our Bloom filter anonymization technique to perform approximate record linkage and de-duplication. 8) We also want to investigate the use of differential privacy in the construction of the block patterns.

Bibliography

[1] *Mining of Massive Datasets*. http://scpd.stanford.edu/public/category /courseCategoryCertificateProfile.do?method=load&certificateId=10555807, 2010.

[2] S. Ajmani, R. Morris, and B. Liskov. A trusted third-party computation service. Technical report, 2001.

[3] A. Al-Lawati, D. Lee, and P. D. McDaniel. Blocking-aware private record linkage. In *IQIS 2005, International Workshop on Information Quality in Information Systems, 17 June 2005, Baltimore, Maryland, USA (SIGMOD 2005 Workshop)*, pages 59–68, 2005.

[4] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, WPES '03, pages 39–44, Washington, DC, 2003. ACM.

[5] F. B. Baulieu. A classification of presence/absence based dissimilarity coefficients. *Journal of Classification*, 6(1):233–246, Dec 1989.

[6] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 547–557, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[7] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, July 1970.

[8] A. D. Boyd, P. R. Saxman, D. A. Hunscher, K. A. Smith, T. D. Morris, M. Kaston, F. Bayoff, B. Rogers, P. Hayes, N. Rajeev, E. Kline-Rogers, K. Eagle, D. Clauw, J. F. Greden, L. A. Green, and B. D. Athey. The university of michigan honest broker: A web-based service for clinical and translational research and practice. *Journal of the American Medical Informatics Association : JAMIA*, 16(6):784–791, Nov-Dec 2009.

[9] A. Z. Broder. Identifying and filtering near-duplicate documents. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, COM '00, pages 1–10, London, UK, UK, 2000. Springer-Verlag.

[10] B. Carbunar and R. Sion. Toward private joins on outsourced data. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1699–1710, Sept 2012.

[11] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference*, pages 353–373, Santa Barbara, CA, USA, 2013.

[12] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*, pages 577–594, Singapore, 2010.

[13] F. Chen, X. Jiang, S. Wang, L. M. Schilling, D. Meeker, T. Ong, M. E. Matheny, J. N. Doctor, L. Ohno-Machado, and J. Vaidya. Perfectly secure and efficient two-party electronic-health-record linkage. *IEEE Internet Computing*, 22(2):32–41, 2018.

[14] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *Proceedings of the 4th International Conference on Progress in Cryptology – LATINCRYPT 2015 - Volume 9230*, pages 40–58, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

[15] S. S. Chow, J.-H. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In *NDSS*, 2009.

[16] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555, Sept 2012.

[17] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM*

Conference on Computer and Communications Security, pages 79–88, Alexandria, VA, USA, 2006.

[18] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security*, FC'10, pages 143–159, Tenerife, Spain, 2010. Springer-Verlag.

[19] R. Dhir, A. A. Patel, S. Winters, M. Bisceglia, D. Swanson, R. Aamodt, and M. J. Becich. A multidisciplinary approach to honest broker services for tissue banks and clinical data. *Cancer*, 113(7):1705–1715, 2008.

[20] D. E. Drake and S. Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85:211–213, 2003.

[21] E. Durham, Y. Xue, M. Kantarcioglu, and B. Malin. Private Medical Record Linkage with Approximate Matching. *AMIA Annual Symposium Proceedings*, 2010:182–186, 2010.

[22] E. Durham, Y. Xue, M. Kantarcioglu, and B. Malin. Quantifying the correctness, computational complexity, and security of privacy-preserving string comparators for record linkage. *Information Fusion*, 13(4):245–259, Oct. 2012.

[23] E. A. Durham, M. Kantarcioglu, Y. Xue, C. Toth, M. Kuzu, and B. Malin. Composite Bloom Filters for Secure Record Linkage. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):2956–2968, Dec. 2014.

[24] S. B. Dusetzina, S. Tyree, A.-M. Meyer, A. Meyer, L. Green, and W. R. Carpenter. *An Overview of Record Linkage Methods*. Agency for Healthcare Research and Quality (US), Sept. 2014.

[25] S. L. DuVall, R. A. Kerber, and A. Thomas. Extending the Fellegi–Sunter probabilistic record linkage method for approximate field comparators. *Journal of Biomedical Informatics*, 43(1):24–30, Feb. 2010.

[26] S. L. DuVall, R. A. Kerber, and A. Thomas. Extending the fellegi–sunter probabilistic record linkage method for approximate field comparators. *Journal of Biomedical Informatics*, 43(1):24 – 30, 2010.

[27] C. Dwork. *Differential Privacy: A Survey of Results*, pages 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[28] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. *ACM Transactions on Algorithms*, 2(3):435–472, July 2006.

[29] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[30] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[31] M. J. FREEDMAN. Efficient private matching and set intersection. *Advances in Cryptology, EUROCRYPT, 2004*, 2004.

[32] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, (EUROCRYPT)*, pages 1–19, Interlaken, Switzerland, May 2004.

[33] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO '84*, pages 10–18, Santa Barbara, California, USA, 1984.

[34] O. Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 1998.

[35] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[36] S. J. Grannis, J. M. Overhage, and C. J. McDonald. Analysis of identifier performance using a deterministic linkage algorithm. *Proceedings. AMIA Symposium*, pages 305–309, 2002.

[37] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Proceedings of the 5th Conference on Theory of Cryptography*, TCC'08, pages 155–175, New York, USA, 2008. Springer-Verlag.

[38] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, San Diego, California, February 2012.

[39] A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino. Private record matching using differential privacy. In *EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, March 22-26, 2010, Proceedings*, pages 123–134, 2010.

[40] A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino. A hybrid approach to private record matching. *IEEE Trans. Dependable Sec. Comput.*, 9(5):684–698, 2012.

[41] S. l. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 577–594, Berlin, Heidelberg, 2009. Springer-Verlag.

[42] N. Jefferies, C. J. Mitchell, and M. Walker. A proposed architecture for trusted third party services. In *Proceedings of the International Conference on Cryptography: Policy and Algorithms*, pages 98–104, 1995.

[43] S. B. Johnson, G. Whitney, M. McAuliffe, H. Wang, E. McCreedy, L. Rozenblit, and C. C. Evans. Using global unique identifiers to link autism collections. *Journal of the American Medical Informatics Association: JAMIA*, 17(6):689–695, Dec. 2010.

[44] P. Jurczyk, J. J. Lu, L. Xiong, J. D. Cragan, and A. Correa. FRIL: A Tool for Comparative Record Linkage. *AMIA Annual Symposium Proceedings*, 2008:440–444, 2008.

[45] D. M. Kar, I. Lazrig, I. Ray, and I. Ray. POSTER: priremat: A distributed tool for privacy preserving record linking in healthcare. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2507–2509, 2017.

[46] A. Karakasidis, G. Koloniari, and V. S. Verykios. Scalable blocking for privacy preserving record linkage. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 527–536, New York, NY, USA, 2015. ACM.

[47] A. Karakasidis and V. S. Verykios. Privacy preserving record linkage using phonetic codes. In *Proceedings of the 2009 Fourth Balkan Conference in Informatics*, BCI '09, pages 101–106, Washington, DC, USA, 2009. IEEE Computer Society.

[48] A. Karakasidis and V. S. Verykios. Secure blocking + secure matching = secure record linkage. *JCSE*, 5(3):223–235, 2011.

[49] A. Karakasidis and V. S. Verykios. Secure blocking+ secure matching= secure record linkage. *Journal of Computing Science and Engineering*, 5(3):223–235, 2011.

[50] A. Karakasidis and V. S. Verykios. Reference table based k-anonymous private blocking. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 859–864, New York, NY, USA, 2012. ACM.

[51] D. Karapiperis and V. S. Verykios. An lsh-based blocking approach with a homomorphic matching technique for privacy-preserving record linkage. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):909–921, April 2015.

[52] A. N. Kho, J. P. Cashy, K. L. Jackson, A. R. Pah, S. Goel, J. Boehnke, J. E. Humphries, S. D. Kominers, B. N. Hota, S. A. Sims, B. A. Malin, D. D. French, T. L. Walunas, D. O.

Meltzer, E. O. Kaleba, R. C. Jones, and W. L. Galanter. Design and implementation of a privacy preserving electronic health record linkage tool in Chicago. *Journal of the American Medical Informatics Association: JAMIA*, 22(5):1072–1080, Sept. 2015.

[53] L. Kissner and D. Song. Privacy-preserving set operations. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, CRYPTO'05, pages 241–257, Berlin, Heidelberg, 2005. Springer-Verlag.

[54] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[55] M. Kuzu, M. Kantarcioglu, A. Inan, E. Bertino, E. Durham, and B. Malin. Efficient privacy-aware record integration. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 167–178, New York, NY, USA, 2013. ACM.

[56] M. Kuzu, M. Kantarcioglu, A. Inan, E. Bertino, E. Durham, and B. Malin. Efficient privacy-aware record integration. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 167–178, New York, NY, USA, 2013. ACM.

[57] I. Lazrig, T. Moataz, I. Ray, I. Ray, T. Ong, M. G. Kahn, F. Cuppens, and N. Cuppens-Boulahia. Privacy preserving record matching using automated semi-trusted broker. In *Proceedings of the 29th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy*, pages 103–118, Fairfax, VA, July 2015.

[58] I. Lazrig, T. Ong, I. Ray, I. Ray, and M. Kahn. Privacy Preserving Probabilistic Record Linkage Using Locality Sensitive Hashes. In *Data and Applications Security and Privacy XXX*, pages 61–76. Springer, Cham, July 2016.

[59] Y. Lindell and B. Pinkas. Privacy preserving data mining. In M. Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pages 36–54, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[60] Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, Apr 2009.

[61] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM.

[62] T. Moataz and A. Shikfa. Boolean symmetric searchable encryption. In *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security*, pages 265–276, Hangzhou, China, 2013.

[63] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 448–457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[64] G. Navarro. A guided tour to approximate string matching. *ACM Computing Survey*, 33(1):31–88, Mar. 2001.

[65] G. Navarro. A Guided Tour to Approximate String Matching. *ACM Computing Surveys*, 33(1):31–88, Mar. 2001.

[66] F. Niedermeyer, S. Steinmetzer, M. Kroll, and R. Schnell. Cryptanalysis of Basic Bloom Filters Used for Privacy Preserving Record Linkage. *Journal of Privacy and Confidentiality*, 6(2), Dec. 2014.

[67] C. Pang, L. Gu, D. Hansen, and A. Maeder. *Privacy-Preserving Fuzzy Matching Using a Public Reference Table*, pages 71–89. Springer Berlin Heidelberg, 2009.

[68] D. Pfitzner, R. Leibbrandt, and D. Powers. Characterization and evaluation of similarity measures for pairs of clusterings. *Knowledge and Information Systems*, 19(3):361, Jul 2008.

[69] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In M. Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pages 250–267, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[70] E. H. Porter, W. E. Winkler, B. O. T. Census, and B. O. T. Census. Approximate string comparison and its effect on an advanced record linkage system. Research report, U.S. Bureau of the Census, 1997.

[71] A. L. Potosky, G. F. Riley, J. D. Lubitz, R. M. Mentnech, and L. G. Kessler. Potential for cancer related health services research using a linked Medicare-tumor registry database. *Medical Care*, 31(8):732–748, Aug. 1993.

[72] S. M. Randall, A. M. Ferrante, J. H. Boyd, J. K. Bauer, and J. B. Semmens. Privacy-preserving record linkage on large real world datasets. *Journal of Biomedical Informatics*, 50:205–212, Aug. 2014.

[73] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid. Privacy preserving schema and data matching. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 653–664, Beijing, China, 2007. ACM.

[74] M. Scannapieco, I. Figotin, E. Bertino, and A. K. Elmagarmid. Privacy preserving schema and data matching. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 653–664, New York, NY, USA, 2007. ACM.

[75] K. Schmidlin, K. M. Clough-Gorr, and A. Spoerri. Privacy preserving probabilistic record linkage (p3rl): A novel method for linking existing health-related data and maintaining participant confidentiality. *BMC Medical Research Methodology*, 15(1):1–10, 2015.

[76] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using Bloom filters. *BMC Medical Informatics and Decision Making*, 9:41, 2009.

[77] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9(1):1–11, 2009.

[78] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *ACM Conference on Computer and Communications Security*, pages 299–310, 2013.

[79] M. Strizhov and I. Ray. Multi-keyword similarity search over encrypted cloud data. In *Proceedings of 29th IFIP TC 11 International Conference*, pages 52–65, Marrakech, Morocco, 2014.

[80] L. Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, Oct. 2002.

[81] T. Tassa and E. Gudes. Secure distributed computation of anonymized views of shared databases. *ACM Transactions on Database Systems (TODS)*, 37(2):11, 2012.

[82] J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *J. Comput. Secur.*, 13(4):593–622, July 2005.

[83] D. Vatsalan and P. Christen. Sorted nearest neighborhood clustering for efficient private blocking. In *Advances in Knowledge Discovery and Data Mining, 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II*, pages 341–352, 2013.

[84] D. Vatsalan, P. Christen, and V. S. Verykios. An efficient two-party protocol for approximate matching in private record linkage. In *Proceedings of the Ninth Australasian Data Mining Conference - Volume 121*, AusDM '11, pages 125–136, Darlinghurst, Australia, 2011. Australian Computer Society.

[85] D. Vatsalan, P. Christen, and V. S. Verykios. An efficient two-party protocol for approximate matching in private record linkage. In *Ninth Australasian Data Mining Conference, AusDM 2011, Ballarat, Australia, December 2011*, pages 125–136, 2011.

[86] D. Vatsalan, P. Christen, and V. S. Verykios. Efficient two-party private blocking based on sorted nearest neighborhood clustering. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 1949–1958, 2013.

[87] Z. Wen and C. Dong. Efficient protocols for private record linkage. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1688–1694, New York, NY, USA, 2014. ACM.

[88] M. Yakout, M. J. Atallah, and A. Elmagarmid. Efficient private record linkage. In *Proceedings of the 25th IEEE International Conference on Data Engineering, ICDE '09.*, pages 1283–1286, March 2009.

[89] M. Yakout, M. J. Atallah, and A. Elmagarmid. Efficient and practical approach for private record linkage. *J. Data and Information Quality*, 3(3):5:1–5:28, Aug. 2012.

[90] A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

[91] S. Yau and Y. Yin. A privacy preserving repository for data integration across data sharing services. *Services Computing, IEEE Transactions on*, 1(3):130–140, July 2008.