



**Technische Universität Berlin**

Chair of Database Systems and Information Management

Master's Thesis

# **Secure Data Matching in Federated Learning**

Ali Arous

Degree Program: Computer Science (Master) (Erasmus Mundus - BDMA)

Matriculation Number: 452833

## **Reviewers**

Prof. Dr. Volker Markl

Prof. Dr. Odej Kao

## **Advisor**

Behrouz Derakhshan (behrouz.derakhshan@dfki.de)

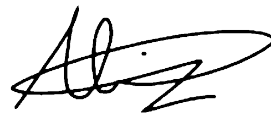
## **Submission Date**

31.08.2021



Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 31.08.2021

A handwritten signature in black ink, appearing to be 'Ali Arous', written in a cursive style.

.....  
*Ali Arous*

# Zusammenfassung

Der Erfolg von *föderiertem Lernen* in den letzten Jahren hat dem maschinellen Lernen von verteilten Daten einen deutlichen Schub verliehen. Die allgemeine Einstellung vertikal partitionierter Daten, bei der mehrere Datenbesitzer unterschiedliche Merkmale über gemeinsame Entitäten besitzen, erfordert eine Datensatzverknüpfung, *Record Linkage*, als Vorverarbeitungsschritt, um entsprechende Datensätze zu finden. Aufgrund des Fehlens eindeutiger Entitätskennungen für eine solche Verknüpfung müssen stattdessen sensible persönliche Identifikatoren wie Namen, Geburtsdaten und Adressen genutzt werden. Die entsprechenden Techniken, die eine sichere Nutzung dieser sensiblen Identifikatoren für die Verknüpfung ermöglichen, werden als *Privacy-Preserving Record Linkage (PPRL)* bezeichnet. Die Einführung von PPRL wird jedoch durch das Fehlen von Evaluierungsrahmen und einheitlichen Werkzeugen behindert, die vergleichende Bewertungen neu entwickelter Techniken ermöglichen.

In dieser Arbeit wird eine empirische Evaluierungsstudie zu modernsten PPRL-Techniken zur Verfügung gestellt, in der ihre Fähigkeiten und Grenzen in den Aspekten Linkage-Qualität und Skalierbarkeit bewertet werden. Weiterhin wird ein Prototyp eines Evaluations-Frameworks, das die Durchführung analoger Evaluationen mit Fokus auf Modularität, Erweiterbarkeit und Benutzerfreundlichkeit, entwickelt. Die empirische Bewertung zeigt die Durchführbarkeit des vorgeschlagenen Frameworks, bei dem Techniken aus verschiedenen Bibliotheken verglichen und ihre Bewertungen anhand verschiedener Messgrößen aufgeführt wurden. Die Ergebnisse zeigen, dass modernste PPRL-Techniken mit fein abgestimmten Parametern mit nicht vertraulicher Record Linkage mit geringfügigen Qualitätsverlusten vergleichbar sein können, während sie das nicht vertrauliche Schema in Bezug auf die Skalierbarkeit übertreffen.

# Abstract

The success of *federated learning* in the recent years has given a significant boost to machine learning over distributed data. The common setting of vertically partitioned data, where multiple data owners hold different features about common entities, requires *record linkage* as a preprocessing step to find correspondent records. Due to the lack of unique entity identifiers for such linkage, sensitive personal identifiers such as names, dates of birth and addresses need to be exploited instead. The corresponding set of techniques that allow to securely leverage these sensitive identifiers in the context of the linkage are referred to as *privacy-preserving record linkage (PPRL)*. The adoption of PPRL is, however, hampered by the lack of evaluation frameworks and uniform tools to enable comparative evaluations of newly developed techniques.

In this work, we provide an empirical evaluation study of state-of-the-art PPRL techniques, assessing their capabilities and limitations in the two aspects of linkage quality and scalability. We also develop a prototype of an evaluation framework that facilitates running analogous evaluations with a focus on modularity, extensibility and ease of use. Our empirical evaluation showcased the viability of our proposed framework where techniques from different libraries were compared and their scores against different measures were reported. Results showed that state-of-the-art PPRL techniques with fine-tuned parameters can be comparable to non-private record linkage with marginal losses in terms of quality, whereas they outperform the non-private scheme in scalability.

# Acknowledgments

This work would not have been possible without the support of my advisor, Behrouz Derakhshan. His enthusiasm and attention to detail have been an inspiration and kept my work on track. I also thank Martin Franke and Florens Rohde from the Database Group at the University of Leipzig for the thoughtful conversation on the experimental evaluation. Lastly, my special thank goes for the BDMA consortium, including professors and staff members, for the generous scholarship and exceptional attention throughout the last two years that concluded with the accomplishment of this work.

# Contents

<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Abbreviations</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation	1
1.2 Research Challenge	2
1.3 Novelty	3
1.4 Anticipated Impact	4
<b>2 Scientific Background</b>	<b>5</b>
2.1 Record Linkage Overview	5
2.2 Privacy Preserving Record Linkage (PPRL)	6
2.3 PPRL Process	7
2.4 Privacy Aspects and Techniques for PPRL	10
2.4.1 PPRL Protocols: Role and Number of Parties	11
2.4.2 Adversary Models	12
2.4.3 Privacy Techniques	13
2.4.4 Attacks	18
2.5 Bloom Filter based Encoding	19
2.5.1 Hashing Techniques	20
Double Hashing	20
Triple Hashing	20
Enhanced Double Hashing	20
Random Hashing	21
2.5.2 Attribute Level Bloom Filter Encoding	21
2.5.3 Cryptographic Long-term Key	21
2.5.4 Record Level Bloom Filters	22
2.5.5 CLK-RBF	23
2.6 Blocking/Filtering Methods	24
2.6.1 LSH-based Blocking Methods	24
MinHash LSH	25



Hamming distance based LSH . . . . .	26
2.6.2 Other Blocking Methods . . . . .	27
Multibit Trees . . . . .	27
2.6.3 Filtering Methods . . . . .	27
Length Filter . . . . .	27
P4Join . . . . .	28
2.7 Evaluation Measures . . . . .	28
2.7.1 Linkage Quality Measures . . . . .	28
2.7.2 Linkage Scalability Measures . . . . .	30
<b>3 Research Problem . . . . .</b>	<b>31</b>
<b>4 Implementation Details and Evaluation Framework . . . . .</b>	<b>33</b>
4.1 Methodology . . . . .	33
4.2 Implementation Details . . . . .	34
4.3 Evaluation Framework: Definition and Usage . . . . .	37
4.3.1 System Architecture . . . . .	38
4.3.2 Usage . . . . .	44
<b>5 Experimental Evaluation . . . . .</b>	<b>48</b>
5.1 Experimental Setup . . . . .	48
5.1.1 Data and Workloads . . . . .	48
Dataset Generation and Statistics . . . . .	49
5.1.2 Evaluation Metrics . . . . .	52
5.1.3 Evaluation Environment . . . . .	52
5.1.4 Methods and Parameters . . . . .	52
Bloom Filter Based Encoding: . . . . .	53
LSH Based Blocking: . . . . .	54
5.2 Experiments on Encoding and Linkage Quality . . . . .	54
5.2.1 Testing the Effect of Padding with 2-grams . . . . .	54
Experimental Design . . . . .	54
Interpretation of the Results . . . . .	55
5.2.2 Testing the Effect of Padding with 3-grams . . . . .	63
Experimental Design . . . . .	63
Interpretation of the Results . . . . .	64
5.2.3 Testing Generalizability to a Second Dataset . . . . .	67
Experimental Design . . . . .	67
Interpretation of the Results . . . . .	67
5.2.4 Testing with Different Subsets of Attributes . . . . .	71
Experimental Design . . . . .	71
Interpretation of the Results . . . . .	72



5.2.5	Testing with Weighted Attributes . . . . .	74
	Experimental Design . . . . .	74
	Interpretation of the Results . . . . .	74
5.3	Experiments on Blocking, Linkage Quality and Scalability . . . . .	76
5.3.1	Testing MinHash Blocking for Parameters Tuning . . . . .	76
	Experimental Design . . . . .	76
	Interpretation of the Results . . . . .	77
5.3.2	Testing MinHash Blocking for Quality and Scalability . . . . .	78
	Experimental Design . . . . .	78
	Interpretation of the Results . . . . .	79
5.3.3	Testing HLSH Blocking for Quality and Scalability . . . . .	81
	Experimental Design . . . . .	81
	Interpretation of the Results . . . . .	82
5.3.4	Testing HLSH Blocking on Big Data . . . . .	86
	Experimental Design . . . . .	86
	Interpretation of the Results . . . . .	87
5.4	Discussion . . . . .	90
<b>6</b>	<b>Related Work . . . . .</b>	<b>92</b>
6.1	Evaluation Work on Linkage Quality: . . . . .	92
6.2	Evaluation Work on Linkage Quality and Scalability: . . . . .	94
6.3	PPRL Research Software Projects . . . . .	97
<b>7</b>	<b>Conclusion . . . . .</b>	<b>100</b>
	<b>Bibliography . . . . .</b>	<b>102</b>



# List of Figures

1	Outline of the general PPRL process [85]. . . . .	7
2	A taxonomy of methodologies and technologies used in PPRL [84] .	11
3	Example of the Dice coefficient similarity calculation [23] . . . . .	17
4	An example illustration of the cryptographic long-term key (CLK) encoding method [23] . . . . .	22
5	An example illustration of record level Bloom filter (RBF) generation [23] . . . . .	23
6	An illustration of MinHash calculations for two sets [23] . . . . .	26
7	Class Diagram for our proposed evaluation framework . . . . .	38
8	An example of a linkage quality report displayed by a <i>Report</i> instance	42
9	An example of a blocking report displayed by a <i>Report</i> instance . .	43
10	Structure of our proposed framework, composed of 5 packages . . .	44
11	Linkage quality measures for NCVR 5K with padding and $k = 10$ .	55
12	Linkage quality measures for NCVR 5K with padding and $k = 15$ .	57
13	Linkage quality measures for NCVR 5K with padding and $k = 20$ .	58
14	Linkage quality measures for NCVR 5K with padding and $k = \text{'opt'}$	60
15	Linkage quality measures for NCVR 5K and $k = \text{'10'}$ , without padding	61
16	Linkage quality measures for NCVR 5K and $k = \text{'opt'}$ , without padding . . . . .	62
17	Linkage quality measures for NCVR 5K with padding, $k = 10$ and $q = 3$ . . . . .	64
18	Linkage quality measures for NCVR 5K without padding, $k = 10$ and $q = 3$ . . . . .	66
19	Linkage quality measures for FEBRL with padding, $k = 10$ and $q = 2$	68
20	Linkage quality for FEBRL and NCVR 5K with increasing $k$ . . . .	69

# List of Tables

1	Schema mapping of NCVR dataset snapshot in 2020 to its 2021 version . . . . .	49
2	Linkage quality measures at optimal $\tau$ with $k \in \{10, 15\}$ . . . . .	58
3	Linkage quality measures at optimal $\tau$ with $k \in \{15, 20\}$ . . . . .	59
4	Linkage quality measures at optimal $\tau$ with $k \in \{10, 'opt'\}$ in addition to $k = 15$ for RBF . . . . .	61
5	Linkage quality measures over unpadded tokens at optimal $\tau$ with $k \in \{10, 15, 20, 'opt'\}$ . . . . .	63
6	Linkage quality measures using different subsets of attributes on NCVR 5K . . . . .	72
7	Linkage quality measures using different attributes weighting on NCVR 5K . . . . .	75
8	Blocking measures for MinHash-4-100 and MinHash-3-50 on NCVR 5K . . . . .	77
9	Linkage time (in seconds) with blocking using MinHash-4-100 and MinHash-3-50 on NCVR 5K . . . . .	77
10	Linkage quality measures using MinHash-4-100 and MinHash-3-50 on NCVR 5K . . . . .	78
11	Blocking measures for MinHash-3-5 on NCVR 5K, 10K and 20K . . . . .	79
12	Linkage time (in seconds) with blocking using MinHash-3-50 on NCVR 5K, 10K and 20K . . . . .	80
13	Linkage quality measures using MinHash-3-50 on NCVR 5K, 10K and 20K . . . . .	80
14	Blocking measures for HLSH on NCVR 5K, 10K, 20K, 50K, 100K and 200K . . . . .	82
15	Linkage time (in seconds) with blocking using HLSH on NCVR 5K, 10K, 20K, 50K, 100K and 200K . . . . .	83
16	Linkage time (in seconds) with HLSH-50 blocking and MinHash-3-50 blocking on NCVR 5K, 10K and 20K . . . . .	84
17	Linkage quality measures using HLSH on NCVR 5K, 10K, 20K, 50K, 100K and 200K . . . . .	85
18	Blocking measures for HLSH on NCVR 200K, 400K, 800K, 1.6M and 3.2M . . . . .	87



19	Linkage quality measures using HLSH on NCVR 200K, 400K, 800K, 1.6M and 3.2M . . . . .	89
----	--	----

# List of Abbreviations

**RL** Record Linkage

**PPRL** Privacy-Preserving Record Linkage

**ML** Machine Learning

**FL** Federated Learning

**GDPR** European General Data Protection Regulation

**HIPAA** Health Insurance Portability and Accountability Act in USA

**API** Application Programming Interface

**QIDs** Quasi-identifiers

**DO** Database Owner

**LU** Linkage Unit

**HBC** Honest-but-curious

**SMC** secure Multi-party Computation

**MD** Message Digest

**SHA** Secure Hash Algorithms

**HMAC** Hashed Message Authentication Code

**ALC** Anonymous Linkage Code

**SLK** Statistical Linkage Key

**DH** Double Hashing

**TH** Triple Hashing

**EDH** Enhanced Double Hashing

<b>RH</b>	Random Hashing
<b>ABF</b>	Attribute level Bloom Filter
<b>CLK</b>	Cryptographic Long-term Key
<b>RBF</b>	Record level Bloom Filter
<b>LSH</b>	Locality-Sensitive Hashing
<b>HLSH</b>	Hamming Locality-Sensitive Hashing
<b>ELSH</b>	Euclidean Locality-Sensitive Hashing
<b>NCVR</b>	North Carolina Voter Record
<b>FEBRL</b>	Freely Extensible Biomedical Record Linkage
<b>TP</b>	True Positive
<b>FP</b>	False Positive
<b>TN</b>	True Negative
<b>FN</b>	False Negative
<b>PC</b>	Pairs Completeness
<b>PQ</b>	Pairs Quality
<b>RR</b>	Reduction Ratio

# 1 Introduction

## 1.1 Motivation

With the increased computing power and storage capacities in the recent years, massive amounts of data were being collected and maintained by organizations from different domains. Such data often need to be shared and integrated between organizations in order to improve data quality and allow for advanced data mining and machine learning tasks. For example, hospitals could harness the medical history of common patients in order to prevent chronic diseases and minimize the risks of future hospitalization [60], for which federated learning could be leveraged.

Federated learning (FL) addresses the problem of training machine learning (ML) models using data collected and maintained by different parties in a decentralized setting, while taking into account the privacy constraints accompanied [47]. It has become a necessity in the recent years, expanding through other fields like databases and privacy [60]. FL can be categorized into horizontal FL and vertical FL, based on how the data is being partitioned. In horizontal FL, data providers record the same features for different entities, whereas in vertical FL, they can record different features for the same entities. Vertical FL is more challenging as parties have to figure out the correspondence between entities of the different datasets in order to learn from the union of all features. A way to overcome this problem is to join the datasets using a family of techniques referred to as entity resolution, entity matching or record linkage [18].

In addition to being a preprocessing step for vertical FL, record linkage (RL) represents a prerequisite for any successful collaboration among partner businesses, when records across their databases will need to be joined together. Its numerous applications span to healthcare where researchers aggregate health databases for epidemiological studies, government services as in the linking of census data, fraud detection to effectively identifying individuals who have committed fraud and also in the internet applications to identify web pages that are about the same topic or are written by the same person [82].

Linking records across organizations that hold personal data often poses the issue of how to protect privacy and confidentiality of such sensitive data due to laws or regulations. Some example Acts that describe the legal restrictions of disclosing personal or sensitive data are: the European General Data Protection Regulation

(GDPR) in the EU countries<sup>1</sup>, the Health Insurance Portability and Accountability Act (HIPAA) in the USA<sup>2</sup>, and the Data-Matching Program Act in Australia<sup>3</sup>. In other words, when record linkage is conducted between organizations, it needs to be guaranteed that the parties involved in the linkage process learn just a limited amount of information that is necessary to achieve the linkage, but not any actual data from other parties' databases [85].

The increasing need to link large databases across organizations while, at the same time, preserving the privacy of the records holding personal data of people, has led to the development of privacy-preserving record linkage (PPRL) techniques [82].

## 1.2 Research Challenge

In general, record linkage is a challenging task due to the lack of unique entity identifiers across databases to be linked, in which case other common available attributes need to be leveraged for the linkage. For databases that store records about people, personally identifying attributes such as names, addresses and dates of birth can be used and are often referred to as pseudo-identifiers [74] or quasi-identifiers (QIDs) [85]. These values, in combination, not only allow uniquely identifying individuals but also reveal private and sensitive information about them [82].

Using these attributes, however, comes with a challenge as they are prone to various variations and errors that could result from different sources and for different reasons. Some of such reasons include changes of addresses as people move to new residences, changes of family names due to marriage or sometimes typos caused by human or machine when entering the information to the system. Missing values can also appear if one of the datasets is not recording a certain attribute at some or all of its records. If exact matching was used, any error or variation would result in the record pairs corresponding to the same entity being classified as non-matches [15].

To overcome this, an approximate matching scheme can be used instead, where different variations of the same attribute value can be matched with a controllable error margin. However, measuring linkage quality by the number of false and missing matches remains one of the major challenges for record linkage applications due to the lack of ground truth information for real-world databases.

Another main challenge is the inherent quadratic complexity of the linkage problem, where every record of the first source needs to be compared with all records

---

<sup>1</sup><https://gdpr-info.eu/>

<sup>2</sup><https://www.hhs.gov/hipaa/>

<sup>3</sup><https://www.oaic.gov.au/privacy/other-legislation/government-data-matching/>

of the other [34], which becomes a scalability bottleneck for linking large data volumes with millions of records. To mitigate this, blocking or filtering methods as well as parallel and distributed processing need to be leveraged to reduce the complexity of the linkage, and speed up the similarity computations [81].

The privacy requirement in privacy-preserving record linkage adds the third challenge of privacy and confidentiality to the previous two challenges, where simply hashing or encrypting two attribute values, with one error in common, can result in two completely different codes. Hence, state-of-the-art encoding techniques, like Bloom filter [7] based ones, need to be used to allow for approximate matching while, at the same time, reducing the risk of data breaches [82].

Numerous PPRL schemes have been proposed in the last decade. However, their practical use in real applications is still limited due to the lack of comparative evaluations that test their relative strengths and weaknesses across the three main challenges of PPRL. There is, therefore, a strong need for easy-to-use, modular and open-source evaluation frameworks to allow practitioners and researchers to test and evaluate different PPRL methods and parameter settings, in order to identify effective PPRL workflows and appropriately balance efficiency and privacy [31].

## 1.3 Novelty

While most of the existing research focused on methods relating to one step in PPRL (i.e, encoding methods or blocking and filtering methods), we survey in this work different existing private encoding and matching techniques for record linkage and present a generic pipeline that captures these methods into sequential phases. We then conduct an empirical evaluation study to comparatively assess the capabilities and limitations of some of these methods in terms of linkage quality and scalability, against non-private record linkage. To the best of our knowledge, no previous evaluation study has been conducted covering the same techniques as ours, in the same breadth and depth. Lastly, we offer our developed code in the form of a modular evaluation prototype of a framework that facilitates running similar evaluations with extensibility options, to allow the convenient and easy integration of new methods, that could be developed in the future.

Despite the existence of some PPRL software projects and toolkits implementations in the literature [53, 77, 41, 43], the majority of them do not offer enough functionalities for a practical use in an evaluation framework, like the capability to properly construct and configure PPRL workflows and the extensibility to accommodate newly developed techniques. While one work [81] contributed a theoretical grounding of an evaluation framework, the focus there was on devising new numerical scores and conducting experimental evaluations instead of describing a system design for a PPRL framework. To the best of our knowledge, we are offering in this



work the first PPRL evaluation framework in the literature, with clearly defined APIs and uniform testing functionalities.

## 1.4 Anticipated Impact

Offering such an evaluation framework will help to boost the pace of testing newly developed PPRL methods and identifying efficient and scalable PPRL pipelines, which will allow better integration of big datasets across different organizations. This will consequently improve the predictive capacity of newly developed ML models, affecting decision-making in various businesses. For example, banks and insurance companies could collaborate for spotting fraudulent activities, while hospitals and medical facilities could leverage the medical history of common patients for more accurate diagnoses capabilities and enhanced hospitalization.

We structure the rest of the thesis as follows. In the following chapter we provide the background about PPRL in overall, and the various techniques connected to it. Then, in Chapter 3 we formalize our research problem, specify its scope and describe the goals that will be explored. We present our evaluation framework and the implementation details in Chapter 4, while in Chapter 5 we provide our comparative empirical evaluation between state-of-the-art PPRL encoding and blocking techniques. We present a review of the literature of existing comparative evaluation studies and software projects for PPRL in Chapter 6, and finally, we conclude our work in Chapter 7 by summarizing our findings and discussing future research directions.

## 2 Scientific Background

In this chapter, we present background material that contributes to the understanding of the preliminaries and provides basic knowledge for the following chapters of this thesis. We start by an overview on record linkage in Section 2.1, which we extend in Section 2.2 to PPRL. In Section 2.3, we describe the different steps involved in the PPRL process, while we provide concepts and definitions related to the privacy aspect in Section 2.4, which will serve for our scope definition and discussions on the privacy preservation of different methods. In Section 2.5, we present four Bloom filter based encoding methods, three of which will be used later in our experimental evaluation. Afterwards, we present blocking and filtering methods in Section 2.6, including two blocking methods used in our experimental evaluation. Finally, we finish this chapter with Section 2.7 by defining various evaluation measures for linkage quality and scalability that we use for our assessment of the results in the experimental evaluation.

### 2.1 Record Linkage Overview

*Record linkage (RL)*, also known as *data linkage*, *data matching*, or *entity resolution*, is the process that aims to identify and link records that correspond to the same real-world entities within one or across several datasets. When records about the same entity need to be identified in a single dataset, it is called *duplicate detection*, or *deduplication* [18].

RL has been applied to a wide range of applications areas including government services, healthcare, fraud detection and business applications [82] where linking records from different datasets aims to improve data quality or enrich data for advanced data mining and machine learning tasks. For example, many health researchers need to combine health databases from different hospitals and health organizations in order to conduct a quality health data mining like in epidemiological studies [12].

*Deterministic RL*, where the linkage is based on exact agreement or disagreement of attributes, can be applied to achieve an exact matching in case unique entity identifiers of high quality were available in the datasets to be linked. If such entity identifiers were missing, *probabilistic RL* that leverages available personal attributes, like names and addresses to achieve an approximate matching, is used

instead [30]. However, linkage quality for probabilistic RL is largely affected by the quality of the input data which often contains typographical errors and heterogeneity in structure [37]. Therefore, preprocessing steps to clean and standardize the input data are usually conducted to mitigate such problems. The scalability to large datasets, is usually addressed by blocking and filtering techniques [30] that alleviate the inherent quadratic complexity of the matching when every record has to be compared with every other record.

Privacy and confidentiality are not of big concern when record linkage is applied within a single organization. However, when data from different organizations need to be linked, privacy and confidentiality should be carefully considered, as the following scenario illustrates [82]:

If a group of public health researchers need to investigate the types of injuries caused by car accidents, in order to uncover correlations between the two, they will require data from hospitals, the police, as well as public and private health insurers. However, neither of these parties is willing or allowed by law to provide their databases to the researchers. On the other hand, the researchers only need to access to some attributes of the records that are relevant to their research and are matched across the different databases, such as the medical details and basic biographic information, like age and gender of people who were involved in car accidents.

## 2.2 Privacy Preserving Record Linkage (PPRL)

As shown in the illustrative scenario in Section 2.1, sharing or exchanging of sensitive personal data between organizations is often not feasible due to privacy concerns, legal restrictions, or commercial interests. Therefore, databases need to be linked in a privacy-preserving way, whereby no sensitive information is being revealed to any of the organizations involved in the linkage, and no adversary is able to learn anything about these sensitive data. This problem has been addressed by the emerging research area of privacy-preserving record linkage [82, 85].

Given several databases containing person-specific data held by different organizations, *privacy-preserving record linkage (PPRL)* is the process that aims at identifying and linking of records that correspond to the same entity/individual across different databases based on the matching of personally identifying attributes, such as names and addresses, without revealing the actual values in these attributes due to privacy concerns. The output of PPRL is a set of matching clusters containing records of the same entity [84].

The problem of PPRL can be formally defined as follows [82]:

Assume  $O_1, \dots, O_m$  are  $m$  data owners with their respective databases  $D_1, \dots, D_m$ . They wish to determine which of their records  $r_1^i \in D_1, r_2^i \in D_2, \dots, r_m^k \in D_m$  match

according to a decision model  $C(r_1^i, r_2^i, \dots, r_m^k)$  that classifies pairs (or groups) of records into one of the two classes  $M$  of matches and  $U$  of non-matches.  $O_1, \dots, O_m$  do not wish to reveal their actual records  $r_1^i, \dots, r_m^k$  with any other party. They, however, are prepared to disclose to a selected party, or to an external party the actual values of some selected attributes of the record pairs that are in class  $M$  to allow further analysis.

The basic idea behind PPRL is that participating parties mask (encode) their records in a uniform fashion and only share the masked data for conducting the linkage, so that no sensitive information is ever exchanged in the clear between organizations involved in a PPRL protocol, nor revealed to any other party [85]. At the end of the PPRL process, database owners (DOs) only learn which of their own records match with a high similarity with records from the other databases. Following that, values in selected attributes of the matched records are exchanged between DOs or sent to a third party who requires the linked data, such as a researcher [82].

## 2.3 PPRL Process

A typical PPRL process involves several steps as illustrated in the following figure, where the steps shown in dark-outlined boxes are to be conducted on masked database records, while dotted arrows show alternative data flows between steps:

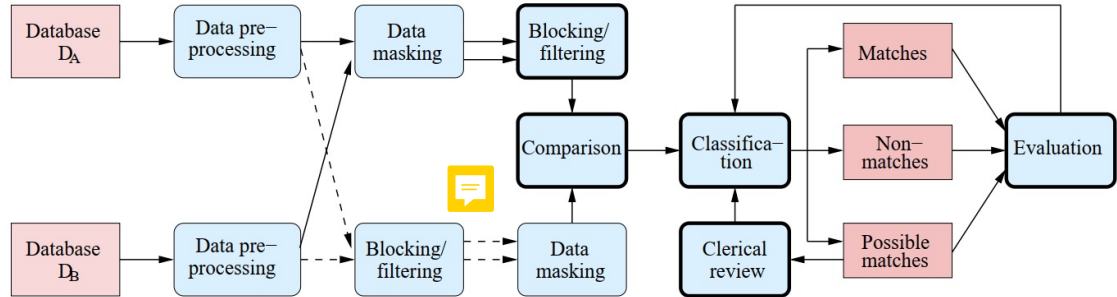


Figure 1: Outline of the general PPRL process [85].

In this section we discuss the steps and the techniques involved in the PPRL process, as shown in Figure 1:

- **Data Preprocessing and Masking:** Since most real-world data are noisy, incomplete and inconsistent [68, 63], data preprocessing is often applied first in the PPRL process to improve linkage quality. This could include filling in missing data and transforming data into consistent forms [14, 24]. As

data preprocessing usually takes place independently at different DOs, it is important that the databases to be linked use the same process of data cleaning and standardization by agreeing on the techniques to be used and the set of attributes to be applied on [24].

Data masking (encoding) needs to be applied as a next step in PPRL, to ensure that no sensitive information is being revealed to any of the parties involved in the linkage or to any other external party, and that no adversary is able to infer anything about the sensitive data. As with data preprocessing, data masking can also be conducted independently with database owners agreeing on the encoding technique and the relevant parameter settings to be used. This will ensure that the same encoding step is applied across all the databases to be linked. Therefore, an exchange of information is often required between the parties involved in the linkage at these two steps [82].

- **Blocking/filtering:** As discussed in Section 1.2, record linkage has an inherent quadratic complexity as every record of the first source needs to be compared with all records of the other during the linkage process. It also has an exponential complexity with regard to the number of databases taking part in the linkage, which all leads to PPRL being not scalable when applied to multiple databases of big sizes.

Blocking and filtering techniques are used in the linkage process with the aim to improve scalability. Both types of techniques reduce the number of comparisons that need to be conducted between records by removing as many record tuples as possible that are unlikely to correspond to matches [6, 17]. As a result, expensive similarity calculations are run on a smaller set of record tuple comparisons.

Blocking groups records into blocks according to a blocking criteria such that comparisons in the next step will be limited to records in the same block or in similar blocks. Filtering, on the other hand, prunes potential non-matching record tuples based on their properties (e.g. lengths of QIDs) [17]. The output of this step are the candidate record pairs (or record sets) that have potentially matching records, which need to be looked at and compared in more detail.

Blocking/filtering can be either conducted on masked records at the linkage unit (LU) or locally on unmasked records by the DOs.

- **Comparison:** As shown in Figure 1, after the blocking step is finished, candidate record tuples need to be compared in the comparison step using comparison (similarity) functions [21]. In record linkage, different comparison functions have been used including Levenshtein edit distance, Jaro-Winkler

comparison, Soft-TFIDF string comparison, and set-based comparison using the Overlap, Dice, or Jaccard coefficient [19]. These functions return a numerical value, usually normalized in the interval  $[0,1]$ , that represents the similarity of the compared QID values, where a similarity of 1 means that two values being exactly the same, and 0 corresponds to two values being completely different.

Several QIDs are usually used when comparing records, which results in a weight vector of numerical similarity values of all compared QIDs per record pair. However, in PPRL, and as the QIDs of records often contain variations and errors, simply masking these values with a secure one-way hash-encoding function could generate completely different encoded values which will highly impact the linkage quality. Therefore, an effective masking approach and approximate comparison functions should be used to securely and accurately calculate the approximate similarity of QID values. Several approximate comparison functions have been adapted for PPRL, including the Levenshtein edit distance [42] and the Overlap, Dice, or Jaccard coefficients [82].

- **Classification:** As discussed in Section 2.1, the goal of the linkage is to classify the generated candidate record tuples into matches, non-matches, or possible matches. In the classification step, the similarity vectors of the candidate record tuples are given as input to an appropriate decision model [35]. Several classification techniques have been developed for record linkage including threshold based [18], rule based [78], machine learning [16] and probabilistic [48] classification techniques. Most PPRL techniques developed so far employ a simple threshold-based classification [82].

In PPRL, the classification technique used should only output the record tuples that have been classified as matches in order to ensure the privacy of the records that do not match. Furthermore, the similarity values of the compared record tuples as well as the similarity distribution across all compared tuples must not be revealed to any party taking part in the linkage to secure against possible privacy attacks [10, 82]. We will provide details about different privacy attacks in Section 2.4.4.

- **Clerical review:** In this step, the record tuples that were classified as possible matches get manually assessed and classified into matches or non-matches [87]. The clerical review in its current form is not possible in a PPRL scenario as the actual QID values of records cannot be inspected to not reveal sensitive private information. A recent work in PPRL suggested a human-machine interaction approach to improve the quality of linkage in PPRL without compromising the privacy [50].

- **Evaluation:** As illustrated in Figure 1, the final step in the PPRL process is the evaluation, which measures the performance of a linkage project in application, in terms of efficiency and effectiveness. Efficiency provides a quantitative measure on the scalability of the techniques used in a linkage process [18], while effectiveness, on the other hand, measures the accuracy of the classification performed by the linkage process. A variety of measures have been proposed for evaluating the scalability and linkage quality [17, 22]. Evaluation of linkage quality in a privacy-preserving context is more challenging given that in a practical record linkage application, the true match status of the compared record pairs are unlikely to be known and accessing the actual record values would likely reveal sensitive private information. Evaluating the amount of privacy protection provided by a PPRL technique using a set of standard measures is still an immature aspect in the PPRL research [81].

In the following sections, we use the term *record pairs* instead of *record tuples* following the basic assumption of record linkage between two parties.

## 2.4 Privacy Aspects and Techniques for PPRL

The privacy aspect of PPRL has several dimensions to be considered. The important ones being: the number of parties and their roles, adversary models, privacy attacks, and data masking (encoding) techniques. In Sections 2.4.1 - 2.4.4, we describe these four privacy dimensions, and in Section 2.5, we discuss Bloom filter-based data encoding, a widely used technique in PPRL. In the following figure, we show a taxonomy of privacy aspects and methodologies used for PPRL:

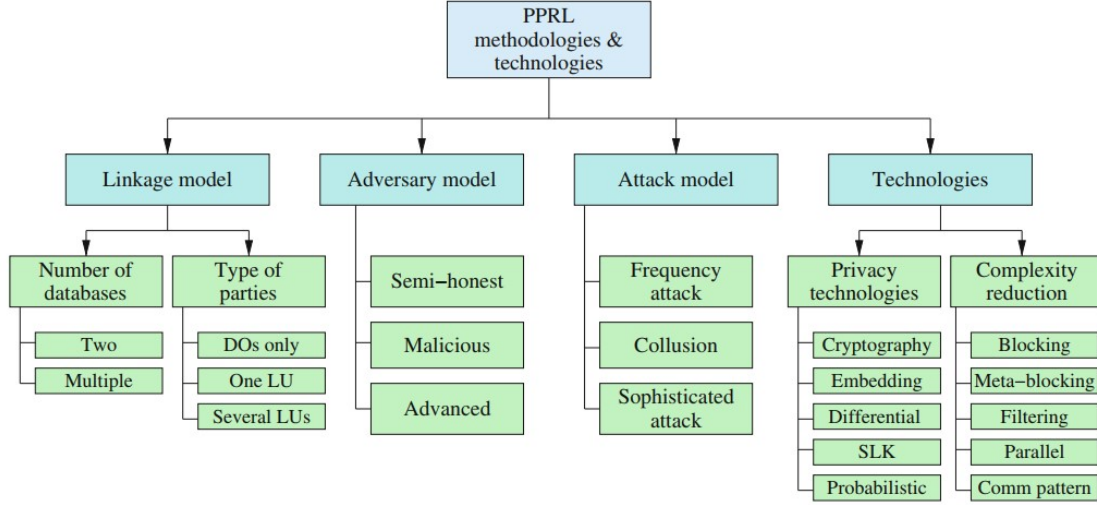


Figure 2: A taxonomy of methodologies and technologies used in PPRL [84]

### 2.4.1 PPRL Protocols: Role and Number of Parties

Considering the number and type of parties involved [84], the linkage models can be categorized as:

- **Two database owners (DOs)** participate in a two-party protocol where they communicate directly with each other in order to identify the matching records in their databases. This protocol is more secure (compared to two DOs with a LU, that we discuss after) as no collusion can happen between the DOs. However, this linkage model is more complex and expensive in terms of computation and communication costs as it requires the use of sophisticated techniques for encoding and encryption to ensure that the parties do not learn any sensitive information about each other's data. Linking two databases has a quadratic complexity in terms of databases' sizes.
- **Two DOs with a linkage unit (LU)** involve in a three-party protocol where a trusted LU (third party) conducts the linkage. The two DOs start by exchanging necessary information such as secret keys, and parameters for preprocessing and encoding methods. After that, both DOs send their encoded records to the LU. The LU performs the matching by calculating similarities between the encoded records, and sends the results of the matching back to the DOs. Even though LU-based protocols are more efficient due to the centralized processing of records, they have a major drawback of having a requirement for a trusted LU. This is to avoid possible collusion



attacks where the LU may collude with a DO in order to infer the attribute values for the other DO. We explain privacy attacks in Section 2.4.4.

- **Two DOs with several LUs** is the model where more than one LU take part in the linkage of two databases. For example, one LU could be responsible for the management of secret keys, and another one for conducting the complexity reduction step, while matching of records may be handled by a third LU. The idea behind separating the tasks across several LUs is to reduce the amount of information learned by a single party. However, collusion could compromise the privacy in this setting.
- **Multiple DOs without or with LU(s)** involve in multi-party linkage model. In this model, the aim is to come up with clusters of matching records from multiple databases that may refer to the same entity. The naive complexity of this model is exponential with regard to the number of databases, and the linkage process becomes more complicated with the more databases need to be linked. Therefore, the linkage process needs to be conducted using more scalable and efficient techniques, while processing can be distributed among multiple parties to improve efficiency as well as privacy.

## 2.4.2 Adversary Models

Different adversary models are assumed in PPRL methodologies [82]:

- **Honest-but-curious (HBC)/semi-honest model:** This is the most commonly used adversary model in existing PPRL techniques [82]. In this model, parties are considered curious in the sense that they try to learn as much information as possible about the other party's input to the protocol, while being honest in following the protocol steps [84].

The protocol is considered to be secure in the HBC model if, at the end of the protocol, all involved parties had no new knowledge above which record pairs have been classified as matches, that they learn from the output. It is important to note, however, that while HBC model assumes a non-colluding coordinator to act according to the protocol, it does not prevent the other parties from colluding with each other with the aim to learn about another party's sensitive information [85]. Due to its weak assumption of privacy against adversarial attacks, this model is not very realistic for real-world applications [84].

- **Malicious model:** In this model, malicious parties can behave arbitrarily by refusing to participate in a protocol, deviating from the protocol by not

following the steps in the specified way, sending false values for their data input, or aborting the protocol at any time [85]. By this, the malicious model provides a strong assumption of privacy where more complex and advanced privacy techniques are needed to make the protocols resistant against such malicious adversaries [84].

Only limited work has been done in PPRL assuming the malicious adversary model [54, 59], while evaluating privacy under this model remains very difficult, due to the potential unpredictable ways for malicious parties to deviate from the protocol.

- **Advanced models:** These are hybrid models developed to overcome the problems associated with the semi-honest model, which is not realistic, and the malicious model, which requires computationally expensive techniques. Two examples of such advanced models are: accountable computing and covert model [84]. In the covert model, the model guarantees that the honest parties will be able to detect, with a high probability, the misbehavior of an adversary [4], while the accountable computing model provides accountability for privacy compromises by the adversaries. This accountability eliminates the need for excessive complexity and cost that usually incur with the malicious model [40]. Further research is needed in the direction of transforming existing HBC and malicious PPRL protocols into these models and proving privacy of solutions under them [85].

### 2.4.3 Privacy Techniques

- **Embedding approaches:** allow QID values to be mapped into a multidimensional metric space [84] (such as Euclidean [8] or Hamming [46]) while preserving the distances between original data points using a set of pivot values that span the multidimensional space [84, 85]. A drawback of this approach is that it is often difficult to determine the appropriate dimensionality for the metric space and to select suitable pivot values [85]. This technique was used in a recent work [44] that proposed a framework for embedding string and numerical data with theoretical guarantees for the specification of space dimensionality.
- **Generalization techniques:** These techniques try to overcome the problem of re-identification of individual records by generalizing the values of QID attributes in a database in such a way that re-identification from the generalized database is not feasible [23]. A variety of such generalization techniques have been developed, including: k-anonymity [76], l-diversity [57] and t-closeness [55]. In k-anonymity privacy model, a k-anonymised database

is said to have the  $k$ -anonymity property if an individual that is represented by a record in the database cannot be distinguished from at least  $k - 1$  other individuals whose information also appears in the database [76].  $K$ -anonymity can be achieved in databases that hold sensitive data by making sure a group of  $k$  records in a database have the same values in their QID attributes.

- **Secure multi-party computation (SMC):** is a cryptographic protocol that has been introduced to overcome the problem of performing secure computations on sensitive data across different parties [56]. SMC enables the collaborating parties to compute a joint function of their private sensitive input values, where at the end of the protocol no party learns anything about any other party's private input but the final result of the computation. SMC techniques have been used in a variety of ways in the context of comparing sensitive values [23]:
  - **One-way hash encoding:** A one-way hash function converts an input value into a hash-code (e.g., 'john' to '61409aa1fd47d4a5332de23cbf59a36f') in such a way that makes it nearly impossible, with current computing technology, to learn the original input value from the hash-code. The common types of one-way hash functions are the Message Digest (MD) and Secure Hash Algorithms (SHA), which include: MD5, SHA-1, and SHA-2 [70]. A keyed hash encoding approach can also be used to prevent dictionary attacks, where an adversary hash-encodes values from a large list of common words using existing hash encoding functions until a matching hash-code is found. This approach significantly improves the security of the one-way hash encoding, as without knowing the secret key, a dictionary attack will not be successful. However, frequency attacks are still possible, where the frequency distribution of a set of hash-codes is matched with the distribution of known attribute values, such as surnames [82]. An example of this approach is the Hashed Message Authentication Code (HMAC) function [38].
  - **Oblivious transfer protocols:** An oblivious transfer protocol allows one party to send several of its sensitive input values to a receiver while ensuring that the sender does not know which of its input values gets selected by the receiver, and the receiver, on the other hand, does not learn any information about the other input values sent by the sender beside the value it has selected. In this way, the protocol protects these sensitive values from both the sender and receiver. Oblivious transfer protocols are, in general, expensive in terms of computation and communication requirements, which might make them the efficiency

bottleneck in the protocol design [23].

- **Homomorphic encryption:** is a form of encryption that allows calculations to be conducted on encrypted data (ciphertext) and to generate a result of these calculations that is also encrypted [1]. The decrypted result of a calculation conducted on homomorphically encrypted values will be exact to the result of the same calculation performed on the original data. In the context of PPRL, homomorphic encryption has been used to calculate the similarities of different data types (e.g., numerical, categorical, and strings) between sensitive values held by different DOs [23].
- **Secure summation:** is a type of SMC protocol that allows multiple cooperating DOs to calculate the sum of their sensitive numeric input values without the need to reveal them to any other parties that take part in the protocol, while all parties learn the final summed result [23]. This protocol does not work for two parties, as one of the parties can easily calculate the input value of the other by simply subtracting its own value from the the sum of the two numbers.
- **Secure set intersection:** Secure set intersection protocols can be used to obtain the intersection of sets of values or records held by different databases without any participating DO having to reveal its full set of values. These protocols have been used in techniques to link sensitive databases in PPRL context [23], and they generally use either commutative [2] or homomorphic [61] encryption schemes.
- **Differential Privacy:** is a rigorous definition that provides guarantees of indistinguishability of an individual in the data with a high probability [84]. It was initially proposed to allow interactive querying of databases that hold sensitive data by adding noise to each result of a statistical query, such as: Count (return the number of records that fulfill a certain criteria) or Sum (return the summation of values of a selected attribute in the records that fulfill a certain criteria). Such queries can, in some occasions, return a small number of values, causing some individuals to become identifiable [23]. Differential privacy has been used in PPRL to perturb the input databases in a differential private fashion before they are being used in a linkage protocol [39], and also to harden Bloom filters by randomly flipping bits in a differential private manner [23].
- **Anonymous Linkage Codes (ALCs):** ALC is an encrypted single string formed by concatenating substrings or functions of different QIDs [74]. ALCs are often constructed from first name, surname, date of birth and gender.

The resulting combination is encrypted using cryptographic hash functions where the resulting hashed string is used as the linkage key. Two records are said to be representing the same entity if their corresponding ALCs exactly matched. Different variations of ALCs exist, ranging from the basic one where QIDs are preprocessed, combined in one string and then encrypted, to the error tolerant variant called Statistical Linkage Key (SLK), where the concatenated string consist of the second, third and fifth character of the surname, the second and third character of the first name, the gender and the date of birth, which has shown best performance when compared to other variants [67]. The encrypted version of SLK is denoted 581-Key, which is the widely used ALC variant.

- **Bloom filters:**

Bloom filter encoding is one probabilistic technique that has been widely used in PPRL solutions [29, 65, 71, 75, 83, 79]. A Bloom filter is a space efficient data structure proposed by Bloom [7] in 1970 for checking element membership in a set [11]. A Bloom filter  $b_i$  is a bit array of length  $l$  bits where all bits are initially set to 0.  $k$  independent hash functions,  $h_j$ , with  $1 \leq j \leq k$ , each with range  $[0, \dots, l-1]$ , are used to map each of the elements  $s$  in a set  $S$  into the Bloom filter by setting the bit positions returned by hash functions  $h_j(s)$ ,  $1 \leq j \leq k$  to 1 [23]. Due to hash collisions, a given bit can be set to 1 in a Bloom filter by more than one element of the input set, which can lead to false positives [14], as querying the Bloom filter will indicate a certain element is in the set while in fact it is not. However, false negatives are never possible in Bloom filters, because if any of the  $k$  hashed positions for a query element  $s_q$  is 0 then  $s_q$  cannot be in the set  $S$  [23].

It was shown in [58] that half of the bits in a Bloom filter should ideally be set to 1 to achieve lowest collision probability. The optimal number of hash functions that lead to this low collision probability, and hence low false positive rate,  $f$ , can be calculated for a given Bloom filter of length,  $l$ , and a number of elements to be hashed,  $n$ , as:

$$k_{opt} = \frac{l}{n} \ln(2) \quad (2.1)$$

Which leads to a false positive rate of:

$$f = \left(\frac{1}{2^{\ln(2)}}\right)^{l/n} \quad (2.2)$$

When using Bloom filters in PPRL context, each string attribute value is first converted into a set  $S = \{s_1, s_2, \dots, s_n\}$  of substrings of length  $q$ , known

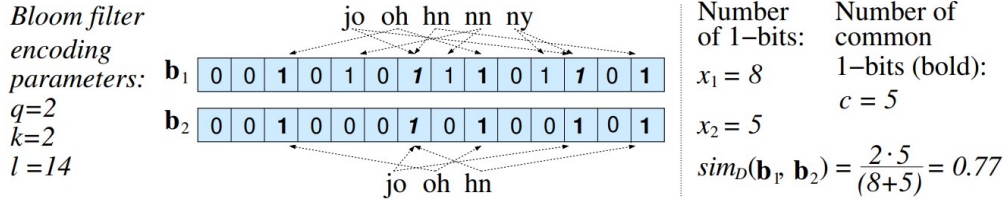


Figure 3: Example of the Dice coefficient similarity calculation [23]

as q-grams [14] using a sliding window approach (e.g., if  $q = 2$ , 'Johnny' gets converted into the list of q-grams [jo, oh, hn, nn, ny]). Then, using a set of hash functions, these sets of q-grams are encoded into Bloom filters. Next, a set-based similarity function, such as the Jaccard or Dice coefficient [14] can be used to calculate the similarity between two Bloom filters, with the Dice coefficient being more commonly used due to its insensitivity to matching zero bits in long Bloom filters [71].

As shown in Figure 3, for two Bloom filters,  $\mathbf{b}_1$  and  $\mathbf{b}_2$ , the Dice coefficient similarity is calculated as:

$$sim_D(\mathbf{b}_1, \mathbf{b}_2) = \frac{2 \cdot c}{x_1 + x_2} \quad (2.3)$$

where  $c$  is the number of bit positions that are set to 1 in both Bloom filters (i.e., the common 1-bits), and  $x_1$  and  $x_2$  are the number of 1-bits in  $\mathbf{b}_1$  and  $\mathbf{b}_2$ , respectively. The number of 1-bits in a bit vector is also referred to as the Hamming weight [23]. The matching can be done either by a LU [29, 71] or directly compared across the DOs [83, 79].

- **Count-min sketches:** are probabilistic data structures, similar to Bloom filters, that are used to hash-map values while recording the number of their occurrences in a sub-linear space [25]. In PPRL, they have been used in situations where the frequency of a matching tuple needs to be identified [45, 66]. However, Count-min sketches only support exact matching of categorical values [85].
- **Reference values:** can be some randomly generated values, or any other values available in public databases, such as telephone directories or voter lists. These values are used as references in several PPRL approaches [69, 62], where a list of reference values need to be known to all DOs who can then calculate the similarities (or distances) between their QID values and this list. The calculated similarities are then either shared with other DOs or sent to a LU where the triangular inequality is leveraged, in order to calculate

the similarities between the sensitive values at different DOs using the shared distances from the reference list [23].

- **Phonetic encoding:** are algorithms used mainly in linkage protocols to group values that have a similar pronunciation together. By using phonetic encoding methods to encode similar attribute values into the same phonetic code (many-to-one encoding), they can provide privacy for these sensitive values [23]. Furthermore, using phonetic encoding to only compare record tuples with the same phonetic code, can lead to a reduction in the total number of record tuples that need to be compared, thereby increasing the scalability while at the same time allowing for an approximate matching of values that have the same phonetic encoding. However, when different attribute values map to the same phonetic code through this method, they get classified as matches, resulting in false positives being generated that eventually reduce the linkage quality [23].

#### 2.4.4 Attacks

Several attack models have been developed to investigate the resistance of different PPRL techniques [84]:

- **Frequency attack:** A frequency attack exploits the frequency distribution of a set of encoded values and compare it to the frequency distribution of a set of known plain text values. By doing this, it is possible that some encoded values can be re-identified even without knowing the encoding method and its parameters used during the encoding process [23]. Frequency attack is among the most commonly used attacks [84].
- **Dictionary attack:** This attack assumes that the adversary has gained access to an already encoded database, and is aware of the encoding method and the parameters used during the encoding process. In order to reveal the actual values that are encoded, the adversary can ideally use a large publicly available database covering a full population, along with their knowledge of the encoding method and its parameters to simply apply this encoding to all values in the database, in order to see which generated encoding would exactly match with a value in the encoded sensitive database. If the adversary succeeds to find such a match, then he will learn the corresponding original value from the plain-text database. To make it more difficult for this attack to succeed, each value can be combined with a secret key that is only known to the organizations that encode their sensitive values [23].



- **Composition attack:** combines knowledge from multiple independent encoded datasets to learn sensitive values of certain records [33]. For example, if a distance-preserving perturbation technique [78] gets attacked with a knowledge about mutual distances between values, it might be the case that the original values can be re-identified with a high level of confidence [85].
- **Cryptanalysis attack:** is a special type of frequency attack that takes place on Bloom filter encoding techniques that may become vulnerable to this attack at certain parameter choices of the Bloom filter, such as the number of hash functions and the number of set bits in a Bloom filter. A previous study [51] has shown that using a constrained satisfaction solver allows the iterative mapping of individual masked values back to their original values [85].
- **Collusion:** is another vulnerability associated with multi-party or LU-based PPRL models [85], where some of the parties involved in the protocol work together to learn about another party's data [84]. In recent times, data related applications from different domains including online rating, auctioning, and mobile computing, that involve multiple parties, have witnessed acts of collusion as some of the parties participating in the protocol were trying to access unauthorised data, to gain certain benefits [3, 9]. In the context of PPRL, the aim of a collusion would be to learn some sensitive data of one party by the colluding with other parties who will share their data and parameters in the aim of revealing the targeted party's data [23].

## 2.5 Bloom Filter based Encoding

Bloom filter encoding is the most popular technique used in practical applications of PPRL [84]. Its popularity comes from its efficiency, simplicity to implement, and its ability to facilitate accurate and approximate similarity calculations of encoded records.

Several techniques have been developed to encode single or several textual attribute values into Bloom filters [23] that we describe starting from Section 2.5.2. These techniques interpret all attribute values as strings and expect them to be converted into sets of q-grams. Then, one of the hashing methods, that we describe in the following, is used to map each q-gram to bit positions in a Bloom filter, which is known as encoding.



## 2.5.1 Hashing Techniques

We describe here four different hashing methods that can be used in Bloom filter based encoding methods. We assume a Bloom filter of length  $l$  bit positions, where the bit positions are numbered from 0 to  $l - 1$ .

### Double Hashing

The idea behind double hashing (DH) scheme is to use two hash functions and an index value  $i$  to simulate a group of  $k$  hash functions. For a given element  $x$ , the DH scheme computes the  $k$  bit positions  $g_i(x)$  to be set to 1, where  $1 \leq i \leq k$ , as follows:

$$g_i(x) = (h_1(x) + i \cdot h_2(x)) \bmod l \quad (2.4)$$

where  $h_1()$  and  $h_2()$  are two independent hash functions [23]. Recent research [49] has shown that DH is vulnerable to cryptanalysis attacks which can lead to re-identification of encoded q-grams in a Bloom filter.

### Triple Hashing

The triple hashing (TH) scheme is an extension to double hashing that employs a third hash function when generating a hash value for an element being hashed [23]. Given an element  $x$  and three independent hash functions  $h_1()$ ,  $h_2()$  and  $h_3()$ , the TH scheme computes the  $k$  bit positions  $g_i(x)$  to be set to 1, where  $1 \leq i \leq k$ , as follows:

$$g_i(x) = (h_1(x) + i \cdot h_2(x) + \frac{i(i-1)}{2} \cdot h_3(x)) \bmod l \quad (2.5)$$

where  $l$  is the length of the Bloom filter.

While TH achieves a lower probability of collision and, therefore, more hashing accuracy than DH [27], its computational cost is 50% higher due to the additional hash function employed.

### Enhanced Double Hashing

Enhanced double hashing (EDH) is an improved version of DH with a similar computational cost, that overcomes the drawbacks associated with both DH and TH [23]. EDH uses two hash functions as in DH, and calculates the bit positions to be set to 1,  $g_i(x)$  where  $1 \leq i \leq k$ , as follows:

$$g_i(x) = (h_1(x) + i \cdot h_2(x) + \frac{i^3 - i}{6}) \bmod l \quad (2.6)$$

where  $l$  is the length of the Bloom filter, and  $h_1()$  and  $h_2()$  are two independent hash functions.

## Random Hashing

The idea behind random hashing (RH) scheme is to use a pseudo-random number generator, that generates a sequence of  $k$  random numbers based on a random seed specific to each element being hashed. These  $k$  random numbers are used afterwards to generate the bit positions to be set to 1 in a Bloom filter. The pseudo-random number generator will always generate the same sequence if initialized to a certain seed value [73]. For a given element  $x$ , RH calculates the bit positions to be set to 1,  $g_i(x)$  where  $1 \leq i \leq k$ , as follows:

$$seed(x), g_i(x) = randint(i, l - 1), \text{ with } 1 \leq i \leq k \quad (2.7)$$

where the function  $randint(start, end)$  returns an integer value in the interval  $[start, end]$  with uniform probability. RH does not generate repeatable patterns, unlike to DH, as it is not based on a linear combination of two hash values. This property makes Bloom filters generated with RH more resilient to certain types of attacks as discussed in [23].

### 2.5.2 Attribute Level Bloom Filter Encoding

In attribute level Bloom filter encoding (ABF) [71], QID values get converted into sets of q-grams, where tokens of each set, representing one attribute, are hashed into one Bloom filter, using a suitable value for the Bloom filter length,  $l$ , and number of hash functions,  $k$ , used to hash each token. QID values can be padded with a special character to allow for the first and last characters of the string to appear in more than one q-gram, giving them similar emphasize as other characters. The outcome of this encoding method is one ABF for each QID attribute. Later in the comparison step of the linkage process, corresponding ABFs across candidate records will be compared together. Even though when initially proposed, ABFs showed accurate linkage results with a reasonable time [71], recent research [49, 51, 52] has shown that they are vulnerable to frequency-based cryptanalysis attacks discussed in Section 2.4.4.

### 2.5.3 Cryptographic Long-term Key

Cryptographic Long-term Key (CLK) [72] encodes multiple attribute values of a single record into one Bloom filter, where different types and numbers of hashing methods can be used for different attributes. CLK procedure is illustrated in Figure 4.

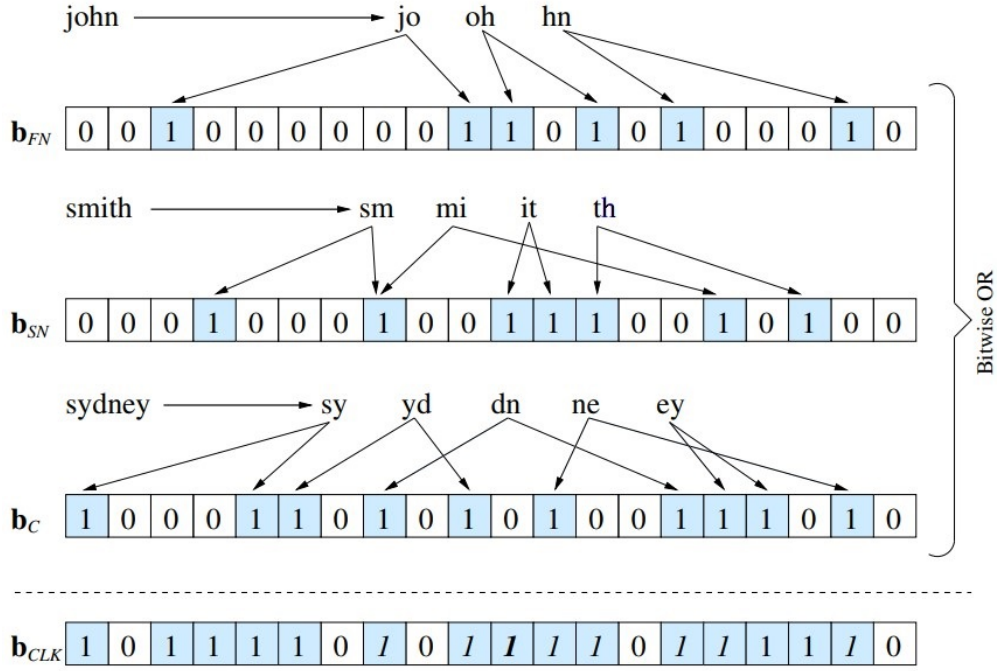


Figure 4: An example illustration of the cryptographic long-term key (CLK) encoding method [23]

As we can see from the figure, CLK is applied using three attribute values from FirstName (john), Surname (smith), and City (sydney), using  $q = 2$  (bigrams),  $k = 2$  and  $l = 20$ . Several collisions are highlighted in italics in  $b_{CLK}$ . For example, both ‘oh’ (from FirstName) and ‘it’ (from Surname) are hashed to the eleventh bit position. Similar to ABF and all other Bloom filter based encoding methods, attribute values can be padded before they are converted into  $q$ -gram sets. However, unlike to ABF where similarity is calculated between values of each QID, in CLK only one similarity is calculated between two records. This could lead to CLK resulting in a reduced linkage quality. However, CLK is more efficient compared to ABF and also provides increased privacy protection [23].

#### 2.5.4 Record Level Bloom Filters

Record level Bloom filter (RBF) [28, 29] is an alternative method to CLK that encodes values from multiple attributes into one Bloom filter. The difference between the two is that in CLK all QID values get hashed directly to one Bloom filter per record, while in RBF, first, an attribute level Bloom filter, ABF, is generated for each QID attribute. After that, random bit positions from the generated ABFs are sampled and combined, after being permuted in a final step, into one Bloom

filter per record. The length of each ABF usually depends on the length of the corresponding QID, where longer Bloom filters can be generated for longer attributes to obtain a similar percentage of 1-bits. Database owners use some form of secure communication to agree on the hash functions, the selections of bit positions, the attribute weights, and the permutation of bit positions, before starting to encode their databases. We illustrate the process of RBF generation in Figure 5.

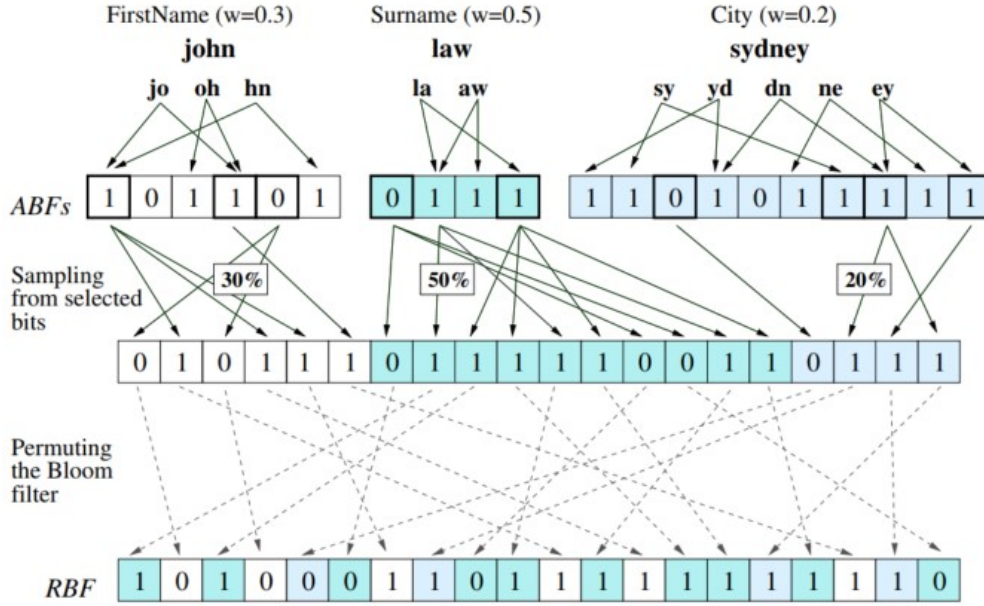


Figure 5: An example illustration of record level Bloom filter (RBF) generation [23]

As shown in the figure, as a first step, the values in the three attributes First-Name, Surname, and City are encoded into separate ABFs with lengths  $l_{FN} = 6$ ,  $l_{SN} = 4$  and  $l_C = 10$ , using the same number of hash functions per q-gram token,  $k = 2$ . After that, certain bit positions (shown in bold) are selected, from which 30%, 50%, and 20% bits are sampled following the respective weights,  $w_{FN} = 0.3$ ,  $w_{SN} = 0.5$  and  $w_C = 0.2$ , for the three attributes. The sampled bit positions are then concatenated into one Bloom filter for the record, which is finally permuted to obtain the final RBF of length 20 bits.

### 2.5.5 CLK-RBF

The CLK-RBF encoding [72, 81] modifies the basic idea of CLK which was initially using the same number of hash functions  $k$  for all QID attributes, to make it use a different  $k$  for different attributes that are to be encoded into one CLK Bloom

filter. The use of different number of hash functions per attribute in CLK-RBF can be seen as weighting of the attributes based on their importance, which can improve the linkage quality, while still providing privacy thanks to mapping all attribute values into the same Bloom filter. For example, if a *first name* attribute is considered more important than a *city* attribute, then more hash functions can be used to encode q-grams of the *first name* values compared to the ones used to hash q-grams from *city* values.

## 2.6 Blocking/Filtering Methods

An important aspect of a PPRL protocol is the number of record pairs that are compared and classified which determines its complexity. A naive pair-wise comparison of two databases is of quadratic complexity in the size of the databases [6]. Private blocking techniques, including the ones we discuss below, are used at the beginning of a PPRL protocol to reduce the number of comparisons by removing the pairs that are unlikely to refer to matches, and only compare in detail the ones that are likely to be matches, in the next step [81].

### 2.6.1 LSH-based Blocking Methods

Locality sensitive hashing (LSH) is a commonly used technique for indexing and searching nearest neighbours in high dimensional data, that has been used as a blocking technique in PPRL to efficiently generate candidate record pairs [23]. LSH uses a family of hash functions (known as LSH family) to map data objects into buckets, so that similar objects get mapped to the same bucket with a high probability, and dissimilar objects are guaranteed to be mapped to different buckets with a high probability as well. By offering these properties, LSH is able to preserve data characteristics in the hashing space and guarantees the collision probability between similar objects.

Formally, an LSH hash family is defined for any two data objects  $x$  and  $y$  as follows:

- if  $d(x, y) \leq t$  then  $h(x) = h(y)$  with probability at least  $p_1$
- if  $d(x, y) > c.t$  then  $h(x) = h(y)$  with probability at most  $p_2$

Where  $d$  is a specific metric distance function,  $t > 0$  is a threshold,  $c > 0$  is an approximation factor and  $h$  is a hash function from the LSH family. An LSH Family is considered interesting for the hashing if  $p_1 > p_2$ .

Each LSH family is sensitive to a specific distance function for specific type of objects:

- Euclidean LSH (ELSH) is locality sensitive to the Euclidean distance metric.
- Hamming LSH (HLSH) is locality sensitive to the Hamming distance which is often used for bit vectors in the Hamming space.
- Min-wise independent permutations LSH (MinHash LSH) allows efficient estimation of the Jaccard similarity between two sets by only comparing the minimum hash values.

To conduct the blocking step of the PPRL pipeline, each DO generates MinHash or HLSH representations for their database records, and sends these representations to the LU where the LSH based similarity calculations are run. Records mapped to the same bucket due to their similar LSH hash values are considered to be candidates. These candidate pairs are later compared in more detail in the linkage step.

Due to the wide adoption of Bloom filter based encoding techniques in PPRL applications, Hamming LSH has ranked as one of the efficient blocking techniques that run directly on bit vectors encoding of data [23].

### MinHash LSH

The basic idea behind MinHash is that the probability of two sets to generate the same MinHash values (known as MinHash signatures) is equal to the Jaccard similarity ( $sim_J$ ) of those two sets if a sufficiently large number of suitable hash functions are used [23].

The probability  $p$  that two sets  $s_1$  and  $s_2$  generate the same MinHash value for a given hash function  $h$  is defined as follows:

$$p[h(s_1) = h(s_2)] = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} = sim_J(s_1, s_2). \quad (2.8)$$

If  $k$  hash functions were employed, the estimated Jaccard similarity ( $sim_{J_e}$ ) between  $s_1$  and  $s_2$  can be computed as:

$$sim_{J_e}(s_1, s_2) = \frac{\sum_{i=1}^k [h_i(s_1) = h_i(s_2)]}{k} \approx sim_J(s_1, s_2), \quad (2.9)$$

where  $[h_i(s_1) = h_i(s_2)]$  returns 1 and  $[h_i(s_1) \neq h_i(s_2)]$  returns 0. Let us assume we have two sets  $s_1 = \{B, C, D\}$  and  $s_2 = \{A, B, C\}$  that have a Jaccard similarity of  $sim_J(s_1, s_2) = 1/2$ . Suppose we select two random permutations  $RP_1 = (2,3,0,1)$  and  $RP_2 = (0,3,1,2)$  to represent the two hash functions  $h_1$  and  $h_2$  respectively. For each hash function, we can select the index of the element of the first row that is 1 (which means the set contains the element) as the MinHash value of this hash function. We illustrate this process in the following figure (Figure 6):

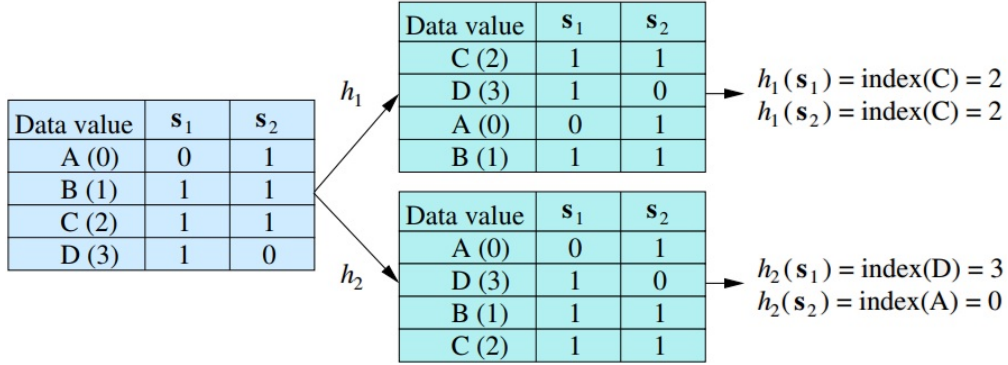


Figure 6: An illustration of MinHash calculations for two sets [23]

To generate a MinHash signature for each set, we concatenate these MinHash values into a list. In our case, we generate the MinHash signatures  $Sig_1 = [2,3]$  and  $Sig_2 = [2,0]$  for sets  $s_1$  and  $s_2$  respectively. We can estimate the Jaccard similarity between the original sets based on the fraction of MinHash values in common between the two sets, which represents the similarity between the two MinHash signatures. The Jaccard similarity estimation in this example will be  $sim_{je}(s_1, s_2) = 1/2$

### Hamming distance based LSH

The idea of Hamming LSH (HLSH) is to group Bloom filters that have similar bit patterns in common into blocks, and to only compare the full Bloom filters that reside in the same block. Similar to MinHash, HLSH generates multiple hash values and combines them into an LSH signature for each Bloom filter. Each hash value in the signature is generated by randomly sampling  $\lambda$  bit positions from a Bloom filter of length  $l$ , where  $1 \leq \lambda < l$ . Increasing  $\lambda$  leads to blocks with more similar Bloom filters, which means higher precision but lower recall. By applying the bit sampling  $\mu > 1$  times, a number of such blocks are generated from each Bloom filter, where in each iteration, a different set of  $\lambda$  bit positions is sampled from the  $l$  positions of the Bloom filter.

For example, given two Bloom filters  $b_1 = [1, 1, 0, 0, 1, 0, 1, 0, 1, 1]$  and  $b_2 = [1, 1, 1, 0, 1, 0, 1, 0, 0, 1]$ , and two sets ( $\mu = 2$ ) of randomly sampled bit positions,  $P_1 = \{0, 2, 4, 6\}$  and  $P_2 = \{0, 1, 5, 7\}$ , of length  $\lambda = 4$ , the first sampling would result in the bit pattern '1011' for  $b_1$  and '1111' for  $b_2$ . This means that the two Bloom filters will be added to different blocks. However, the second sampling gives '1100' for both  $b_1$  and  $b_2$ , which yields to the two Bloom filters being grouped together in the same block for a later detailed comparison [23].

We note here that the textbook [23], which we depend on in our explanation,



confuses  $\lambda$  and  $\mu$ , and reuses the same symbols later, as well as in its accompanying code, to refer to the number of bands to which LSH signatures can be divided to by  $\lambda$ , and the band size which specifies the number of signatures a band spans by  $\mu$ . We add here our disambiguation to the two symbols:  $\lambda$  is used throughout our work, except for the above explanation of Hamming HLSH, to refer to the number of bands, whereas  $\mu$  is used to refer to the band size. Another connected confusion can be made between the *band* and *block* terms. The difference between the two is that a block contains a set of Bloom filters (or plain-text records) that got mapped to it by having at least one identical band (segment) in their LSH signatures.

## 2.6.2 Other Blocking Methods

### Multibit Trees

A multibit tree is a binary tree data structure that assigns bit vectors such as Bloom filters to its nodes based on selected bit positions referred to as *match bits*. At each node in the tree, match bits are selected in a way that, approximately half of the bit vectors have their match bits set to one, while the other half exhibit a value of zero, which will keep the tree as balanced as possible. Each of these halves is called a leaf. The splitting-in-half process is repeated until a user-defined minimum number of bit vectors in each half is reached, which is usually set between one to eight [13].

Multibit trees can be used as a blocking method for PPRL, where a pair of Bloom filters,  $b_1$  and  $b_2$ , is compared using the Jaccard or Tanimoto similarity, as follows:

$$sim_J(\mathbf{b}_1, \mathbf{b}_2) = \frac{|b_1 \cap b_2|}{|b_1 \cup b_2|} = \frac{|b_1 \text{ and } b_2|}{|b_1 \text{ or } b_2|} \quad (2.10)$$

To find similar pairs in terms of Tanimoto similarity, the tree structure is first constructed for the larger dataset, then every record in the smaller dataset gets queried sequentially against the constructed tree. An upper bound of the Tanimoto similarity can be estimated for each record by comparing the values at the bit positions of each leaf in the tree, before the actual similarity calculation. Leaves with a similarity under a user-defined Tanimoto threshold are disregarded in the calculation of the similarities, whereby the search space gets reduced drastically.

## 2.6.3 Filtering Methods

### Length Filter

The Jaccard (as well as Dice) similarity function allows the application of a simple *length filter* to reduce the search space [85], as the minimal similarity is only



achieved if the numbers of Bloom filter set bits, known as *lengths*, of the two input records do not deviate too much. Formally, for two records  $r_i$  and  $r_j$  with  $|r_i| \leq |r_j|$ , it holds that

$$\text{sim}_j(r_i, r_j) \geq s_t \Rightarrow |r_i| \geq \lceil s_t \cdot |r_j| \rceil \quad (2.11)$$

Where  $|r_i|$  is the length of the record, measured by the number of set bits in its corresponding Bloom filter, and  $s_t$  is the Jaccard similarity threshold.

### P4Join

P4Join is the privacy preserving version of the previous PPJoin [75]. This method utilizes the length filter in addition to prefix and position filters to reduce the search space. The *prefix filter* bases on the fact that matching bit vectors need a high degree of overlap in order to satisfy a predefined threshold. Thus, pairs of records can be excluded from the detailed comparison if they did not have a sufficient overlap, which can be checked on a relatively small subset of bit positions, such as the ones at the beginning, known as prefix [85]. P4Join reorders the positions of bit vectors in ascending order of their frequency counts, so that the prefixes of bit vectors contain infrequently set bit positions, which reduces the likelihood of an overlap with other bit vectors. The *position filter* of P4Join, can avoid the comparison of two records even if their prefixes overlap, depending of the positions where the overlap occurs [75].

## 2.7 Evaluation Measures

The PPRL process can be evaluated on three aspects [23]:

1. Linkage quality and completeness
2. Linkage complexity and scalability
3. Privacy protection

In the following, we only consider the first two aspects, as the third aspect has no accepted standard measures yet, and is out of our scope in this work.

### 2.7.1 Linkage Quality Measures

Similar to the evaluation of supervised classifiers in Machine Learning and many other domains, ground truth data in the form of true matches and true non-matches are needed in order to be able to evaluate the quality of the PPRL process [82].

PPRL has also to deal with a trade-off between privacy preservation and the linkage quality. Maintaining more privacy usually means compromising data quality due to information lost in the masked data as compared to the original data.

In the context of record linkage, we consider two records to represent a match, if they are assumed to refer to the same entity, and a non-match if they are assumed to refer to different entities. In case the set of ground truth record pairs in the form of true matches and true non-matches was available, we can calculate the following four classification outcomes:

- True Positives (TP): are the compared record pairs that have been classified as matches and represent true matches.
- False Positives (FP): are the compared record pairs that have been wrongly classified as matches while they are true non-matches.
- True Negatives (TN): are the compared record pairs that have been classified as non-matches, and they represent true non-matches.
- False Negatives (FN): are the compared record pairs that have been classified as non-matches, but they are actually true matches.

The end goal of the classification step of record linkage is to correctly classify as many true matches (TP) as possible while, at the same time, keeping the number of false matches (FP) and missed matches (FN) as small as possible [23].

Based on the four counts of TP, FP, TN and FN, we can calculate a variety of quality measures for evaluating PPRL. The most commonly used ones being:

- Precision: measures how many of the classified matches are in fact true matches. It is calculated as  $P = TP / (TP + FP)$ .
- Recall: measures how many of the true matches are correctly classified as matches. It is calculated as  $R = TP / (TP + FN)$
- F-measure: is a quality measure that combines precision and recall into a single number by using their harmonic mean. It is calculated as  $F = 2 \times (P \times R) / (P + R)$

Accuracy is not a suitable measure of linkage quality, because classifying record pairs is generally an imbalanced classification problem, where the many more non-matching record pairs compared to matching pairs can significantly distort the accuracy measure [82].

It is important to note that there is a trade-off between precision and recall that is directly affected by setting the classification threshold lower or higher. Since

F-measure is a type of averaging that is only high if both precision and recall are high, it is often used to obtain a compromise between the two measures [23].

In practice, measuring the linkage quality is challenging, because no ground truth data for the the actual matches are available in most real-world applications. The linkage quality can, however, still be assessed in a pilot study using data with known match status, or by using manual classification results obtained by clerical review in a record linkage process [82].

## 2.7.2 Linkage Scalability Measures

While linkage quality is the most important aspect of a linkage project with regard to the obtained linked dataset [23], it is also important to assess the complexity of a linkage process which will determine its scalability to larger datasets. The most common way to evaluate linkage complexity (and hence scalability) is based on the number of candidate record pairs generated in the blocking step and their quality [81]. If we denote the number of true matches and true non-matches included in the candidate record pairs generated using a blocking technique by  $B_M$  and  $B_N$ , and the total number of true matches and true non-matches in the full record pairs by  $N_M$  and  $N_N$  respectively, then we can define the following three complexity measures:

- Reduction ratio (RR): measures how much a blocking technique is able to reduce the number of candidate record pairs that are being generated compared to all possible record pairs. It is calculated as  $RR = 1 - (B_M + B_N)/(N_M + N_N)$ . A high reduction ratio means a blocking technique has removed many record pairs. This measure, however, does not take the quality of these candidate record pairs into account.
- Pairs Completeness (PC): measures the number of true matching candidate record pairs preserved by the blocking technique over all true matching pairs across the two datasets being linked. This measure corresponds to recall and it is calculated as  $PC = B_M/N_M$ . A lower pairs completeness value means more lost true matching record pairs during the blocking process which leads to a lower linkage quality [23].
- Pairs Quality (PQ): measures the number of true matching candidate record pairs preserved in blocking over the total number of candidate record pairs generated. It is calculated as  $PQ = B_M/(B_M + B_N)$

### 3 Research Problem

Our goal in this work is two folds:

1. We aim at conducting an experimental evaluation of state-of-the-art encoding and blocking techniques for PPRL, that would contribute new findings to existing work.
2. To come up with a prototype of a generic and easy-to-use evaluation framework that would facilitate conducting similar experimental evaluations in the future on various other techniques.

The design of an extensive experimental evaluation is challenging due to the lack of standards among researchers, i.e., different studies do not necessarily agree on one best way of conducting each aspect of the evaluation. On the other hand, the expected evaluation framework need to provide enough functionalities to facilitate the comparative evaluation of different PPRL techniques with regard to linkage quality, scalability and a future extensibility to the privacy aspect.

Designing such a framework to compare existing techniques and accommodate for future ones is a challenging task for several reasons:

- Researchers use different evaluation measures for linkage quality and scalability assessment.
- Researchers use a variety of datasets with different preprocessing requirements including real-world and synthetic ones for PPRL.
- For many developed techniques, there is no clear separation between the PPRL pipeline steps, where different aspects are wrangled together in their implementation.

The scope of our work is a three-party, probabilistic PPRL protocol with a semi-honest (HBC) coordinator, which is commonly used in the literature [13, 23, 32, 81] As we explained in Section 2.4.2, this protocol assumes that the coordinator acts according to the protocol and does not collude with any of the other two parties. The requirements of running this three-party privacy protocol is that at the end of the linkage, only a limited amount of information is revealed to any of the parties that conducted the linkage, or to an external party that requires the linked data.



The future application environment is to run PPRL in a distributed setting as a previous step of a federated learning task, where data do not leave its premises, and data providers protect their privacy against a central aggregator.

## 4 Implementation Details and Evaluation Framework

In this chapter we present our research methodology, including how we addressed the different challenges accompanied with the research problem, then we provide an elaborate explanation of our implementation and design choices, and we end with describing the system architecture and usage of our developed evaluation framework.

### 4.1 Methodology

Our solution approach builds on firstly surveying the literature of PPRL in order to pinpoint the differences and similarities between theoretical and experimental evaluation studies, which we build on for the design of our comparative evaluation as well as our generic evaluation framework. In particular, we aim at assessing the PPRL techniques by:

- Theoretically surveying them and their different characteristics including the privacy aspect, as presented in Chapter 2.
- Implementing or integrating some of them, taking into account the heterogeneity between the different libraries where they reside.
- Empirically assessing them for efficiency and scalability, against different measures related to classified record pairs and time.
- Drawing conclusions on the best configurations of techniques to achieve efficient and effective privacy-preserving record linkage.

We follow the most commonly used approaches for experimental design including the choice of parameters, the derivation of different configurations, and the assessment of the results with different evaluation measures. While different studies do not necessarily agree on one way of conducting every aspect of the evaluation, we follow the more common ways that are being widely used in the recent years.

Beside conducting our experimental evaluation in an incremental fashion, we propose a set of design requirements for a PPRL evaluation framework, that we

compose from surveying the literature as well as from our experimental work. We then try to progressively reach to those requirements in our aim to come up with an initial prototype of a framework. We achieve this by adding new implementations and design features along the way as we proceed with our experimental evaluation.

Our proposed requirements are derived from the challenges we identified to be accompanied with the design of such a framework, where existing techniques need to be compared, while future ones need to be also accommodated. We list here the most relevant requirements for such design in comparison to the challenges discussed in Chapter 3:

- As researchers use different evaluation measures for linkage quality and scalability assessment, the framework needs to identify and support the commonly used ones, and allow a loosely coupled replacement of measures.
- As researchers use a variety of datasets including real-world and synthetic ones for PPRL, this requires a generic preprocessing scheme to restructure these datasets into a uniform input to the framework.
- As there is no clear separation between the different PPRL steps in the implementations of many of the developed techniques, an interface-based solution is required, where each technique is regarded as a black box that satisfies an interface contract.

In addition to the previous points, the prospected framework should also allow the users to easily plug in their implementations of new techniques and test them against the already existing ones under uniform conditions.

## 4.2 Implementation Details

We start our implementation by building upon the empirical evaluation code provided by [23]. The codebase includes (among other files): a data generation file *datasetPairGenerator.py*, a data encoding file *encoding.py* where two of the encoding methods that we use, CLK and RBF, were implemented, and a linkage evaluation file *evalLinkage.py*, which defines the overall procedure for running a linkage process. Since the code was written in the older Python version 2.7, we upgrade it to version 3.7 by conducting the necessary fixes on the syntax with the help of Python *2to3* software<sup>1</sup>.

The code is initially designed to be runnable from the command line, which we keep as a possibility. However, since our aim is to extend this code to a more

---

<sup>1</sup><https://docs.python.org/3/library/2to3.html>

mature evaluation framework, we restructure it as a library to allow importing and calling its methods from a Python Jupyter Notebook<sup>2</sup>.

In addition to the previous code, we use the Python library *clckhash*<sup>3</sup> which includes an implementation of CLK-RBF method that we use in the linkage evaluation experiments, next to CLK and RBF. Due to the difference in the API between the *clckhash* library and the code we are leveraging, we write a wrapper function to unify the call of CLK-RBF implementation, within the *clckhash* library, with the other two methods. Our defined function *gen\_bf\_dict\_external()* takes as parameters the same ones taken by the other two methods, including:

- A dictionary of records, where the key is a voter id (ncid) and the value is a list of QID attribute values
- A list of column headers
- The Bloom filter length
- The length of the q-gram tokens
- The number of hash functions used to hash each q-gram token
- An optional boolean parameter, if set to True, each attribute is assigned an equal weight in the resulting Bloom filter, regardless of the number of q-gram tokens it comprises.

The function called on a certain configuration of parameters, runs the generation of CLK-RBF Bloom filters in terms of API calls over the *clckhash* library, which includes, as stated in *clckhash* documentation, the definition of a linkage schema that specifies attributes weights and the type of weighting, followed by a call to the encoding method. Our wrapper function returns back results in the same expected format as the other functions, which is a dictionary of voter ids as keys and Bloom filters representing the encoded records as values. To integrate the CLK-RBF encoding method in the linkage evaluation code, we adapted the encoding part, blocking part, and the results accumulating part.

In addition to the previous integration step, we made the following contributions to the exiting code:

We expanded the collected results in the experiment runs to include, in addition to Precision and Recall, the F-measure, true positive (TP), false positive (FP), true negative (TN) and false negative (FN) in the linkage quality experiments, and Pairs Completeness (PC), Pairs Quality (PQ) and Reduction Ration (RR) in the blocking and scalability experiments. This involved various code additions and

---

<sup>2</sup><https://jupyter.org/>

<sup>3</sup><https://pypi.org/project/clckhash/>



modifications in terms of measurement and collection of results. Despite reported, we later discarded the last two measures from discussions sufficing with the rest.

To properly present the results in a comprehensive way that allows comparing multiple methods at a glance, we wrote three functions:

- *display\_df()*, which takes a dictionary of varying structure of stored linkage quality results and presents them in a tabular way, using multi-indexing and highlighting to pinpoint optimal values and highest scores.
- *display\_blocking()*, which is similar to the previous one, but is dedicated for blocking results that are presented in a different tabular design from the linkage quality results. We implemented this function in a way that enables the reporting of accumulated results across multiple runs.
- *plot\_it()*, which takes as parameters: lists of similarity thresholds, precision, recall, and f-measure values, collected for the different encoding techniques. Then it plots them in three adjacent figures, for each evaluation measure. Each figure shows how the different encoding methods compare to each others, and to the plain q-gram linkage. This function can either plot to the screen or save the plot as an image file.

In addition to the previous functions, we wrote a *save\_table()* function, which translates a formatted pandas dataframe into LATEX<sup>4</sup> markup, and writes it into a *latex\_tables.tex* file on the hard disk drive. We called this function inside our first *display\_df()* function in a parameterized way, so that users can decide whether they want to switch on or off the logging of their generated tables in a latex markup. Later, we also added the possibility to save the results in an formatted Excel file. We note here that most of the tables included in our experimental evaluation were automatically generated with this function.

In another context, we overloaded the *load\_data\_set()* function, which was initially reading a specified number of records form a .csv file and returns them in a dictionary. In our implementation, we extended this functionality by allowing the user to specify the amount of non-matching records desired between the datasets, which helps to test the effect of having datasets with different ratios of matching and non-matching record pairs, on the quality of the linkage.

Later in the course of our experimental design, and as a final addition to the encoding part, we introduced necessary code logic to the linkage evaluation in order to implement the weighting of RBF and CLK-RBF methods.

Talking about blocking techniques, the provided implementation of *MinHash* blocking function starts with generating a MinHash signature of length  $\lambda \cdot \mu$  hash

---

<sup>4</sup><https://www.latex-project.org/>

values for each record. Then it divides these signatures into  $\lambda$  bands each with  $\mu$  MinHash values. As a result, records with the same MinHash value in a certain band will be inserted into the same block in the following step. The MinHash blocking code sets  $\lambda = 50$  and  $\mu = 3$ , respectively (which we changed later to test with other values). Therefore, every record was inserted into 50 blocks. The implementation limits the size of generated blocks to a maximum of 100 records per block in order to avoid large blocks that are expensive to process in terms of time and memory. Finally, the code compares record pairs residing in corresponding blocks.

As for *Hamming LSH (HLSH)* implementation: instead of sampling random permutations of bit positions, the code partitions the Bloom filter into *num\_seg* segments, where each segment represents one sampled permutation of bit positions of consecutive bits. This corresponds to inspecting Bloom filters at their different segments and grouping the ones that look the same at a certain segment as candidates for a detailed comparison step. Here, each segment represents a key for one block. The maximum size of a block is also controlled with *max\_block\_size* where all blocks larger than this limit, will be removed to limit memory use and runtimes. In this implementation, the *num\_seg* corresponds to  $\lambda$  in our description in Section 2.6.1, whereas  $\mu$  is determined by the length of the Bloom filter divided by *num\_seg*.

We modified the MinHash blocking function to fix some broken code related to the hashing of q-grams, whereas for HLSH, we replaced one scalability bottleneck step in the linkage implementation, that was causing the original code to crash on larger datasets, with a more efficient and scalable solution, that we explain in more detail in Section 5.3.4.

Following that, we fixed the linkage evaluation code to allow for the smooth switching between the two blocking methods HLSH and MinHash LSH, as the code was not properly defined in this aspect and was pre-set to only run blocking with HLSH.

With the help of the resulting code, we did various experiments to compare encoding and blocking PPRL techniques, while meanwhile progressively reformulating it in a form of a modular evaluation framework that caters to the flexibility and extensibility requirements of our proposed design guidelines.

## 4.3 Evaluation Framework: Definition and Usage

We explain in this section the definition and usage of our evaluation framework prototype and the user API that it offers. Our proposed design for the evaluation framework allows the customization of a PPRL pipeline, integration of new meth-

ods at each step, and the generation of different evaluation measures for PPRL linkage quality and scalability aspects.

### 4.3.1 System Architecture

We show below the class diagram for our solution, highlighting the most important functions:

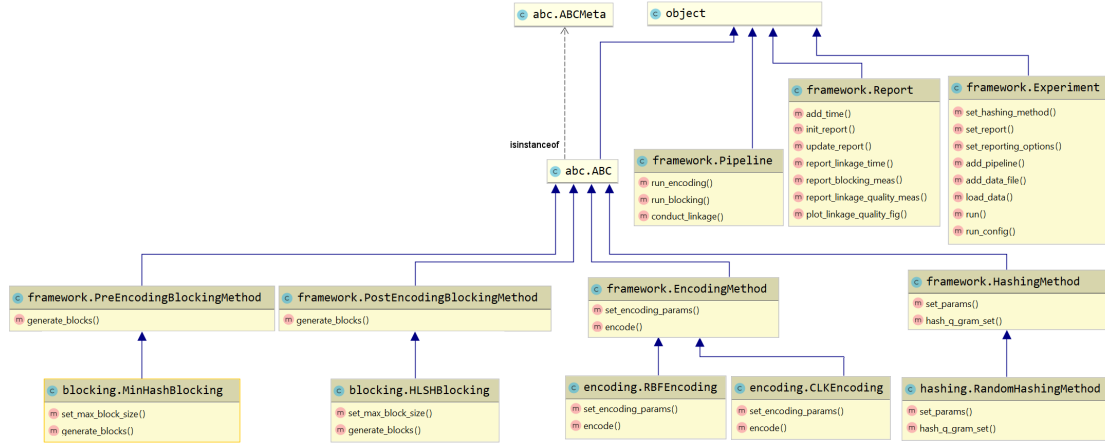


Figure 7: Class Diagram for our proposed evaluation framework

As illustrated in Figure 7, the framework comprises the following main packages:

- *framework.py* file, which contains the core classes of the framework: *HashingMethod*, *EncodingMethod*, *PreEncodingBockingMethod*, *PostEncodingBlockingMethod*, which are abstract classes, and *Pipeline*, *Experiment* and *Report*, which are concrete classes.
- *hashing.py* file, which contains available hashing classes each of which extends the *framework.HashingMethod* abstract class.
- *encoding.py* file, which contains available encoding classes each of which extends the *framework.EncodingMethod* abstract class.
- *blocking.py* file, which contains available blocking classes each of which extends either the *framework.PreEncodingBockingMethod* or the *framework.PostEncodingBockingMethod* abstract classes.

We discuss in the following the contracts and functionalities offered by the *framework.py* classes in detail.

**HashingMethod class:**

This abstract class represents a blueprint to be extended by any class offering a hashing functionality. The concrete class must implement the *hash\_q\_gram\_set()* function, which takes a set of q-gram tokens and returns a *bit vector* of the hashed tokens.

**EncodingMethod class:**

This abstract class represents a blueprint to be extended by any class offering an encoding functionality. The concrete class must implement the *encode()* function, which takes a list of attribute values and returns a *bit vector* representing the values encoded in a Bloom filter.

**PreEncodingBlockingMethod class:**

This abstract class represents a blueprint to be extended by any class offering a blocking functionality over q-gram tokens (i.e., before the encoding step). The concrete class must implement the *generate\_blocks()* function, which takes a dictionary of record ids as keys and set of q-gram tokens as values.

**PostEncodingBlockingMethod class:**

This abstract class represents a blueprint to be extended by any class offering a blocking functionality over Bloom filters of encoded data. The concrete class must implement the *generate\_blocks()* function, which takes a dictionary of record ids as keys and Bloom filters as values.

**Pipeline class:**

This class represents an abstraction of a PPRL pipeline that is composed of three main steps: encoding, blocking and matching. It can be initialized by passing one instance of an encoding method and one instance of a blocking method to its constructor, or by only passing an encoding method instance, in case no blocking was required. The Pipeline instance can, afterwards, run the encoding process by calling its member function *run\_encoding()* on a dictionary of records attribute values. The *run\_encoding()* updates the class field members with the intermediate results of encoded data in the form of Bloom filter dictionaries. The Pipeline instance achieves blocking, by calling its member function *run\_blocking()* in one of three scenarios:

- In case the Pipeline was constructed with no blocking method, the *run\_blocking()* function assigns all records to one block.

- In case the Pipeline was constructed with a `PreEncodingBlockingMethod` instance, the `run_blocking()` function expects two dictionaries of q-gram values in the input and delegates the blocking of each party's dictionary to the provided blocking method in the Pipeline. The function updates the Pipeline member fields with block dictionaries at the end of the blocking process.
- In case the Pipeline was constructed with a `PostEncodingBlockingMethod` instance, the `run_blocking()` function delegates the blocking of the Bloom filter dictionaries resulted from the encoding step to the provided blocking method in the Pipeline.

Lastly, the Pipeline instance runs the `conduct_linkage()` member function to do the matching, and the classification of record pairs into matches and non-matches. This function takes in input a similarity threshold and returns a dictionary of matched record pairs that share a similarity equal or higher than the passed threshold.

### Experiment class:

This class represents an abstraction of a PPRL experiment. The modeling of the experiment in this class is composed of:

- A set of pipelines that are to be compared together.
- A set of dataset files that correspond to data hold by different parties interested in linking their data securely.
- A set of configurations each of which represents an interesting combination of PPRL parameters to test the different pipelines against.
- A reporting utility, modeled by the `Report` class, to collect scores from different pipelines at different configuration runs and present them in a comprehensive format.

The Experiment class uses its member function `add_pipeline()` to add a new pipeline construct to its set of pipelines to be compared. It uses its member function `add_data_file()` to assign a data source path to a member field. This function is designed in a way that allows extending its use to multiple datasets (i.e., more than two) as it takes a dataset file path in the first argument, and a number representing the order of the party in the linkage, in the second argument. After the data sources get assigned, the Experiment class can call its member function `load_data()` to read the records from the parties' datasets. To run the experiment, a set of arguments need to be provided to the `run()` member function. The expected format of each parameter is a list of interesting values to test with. The

*run()* function, afterwards, loops through all combinations of the provided parameters, calling at each combination the *run\_config()* member function over a set of primitive parameters' values. This flexible design of the *run()* function makes it easy to test with various combinations of parameters' values while, at the same time, offering the possibility to fix a value for a certain parameter, by passing it within a single-element list. Finally, the *run\_config()* member function models one experiment run over a certain configuration of parameters' values. This function loops through the ready-to-compare pipelines in the Experiment instance, and for each pipeline, it executes the different steps of encoding, blocking and matching, while capturing the generated scores with the help of a *Report class* instance.

### **Report class:**

This class represents a reporting utility that can be used to capture various evaluation measures in a PPRL process. The class is designed to comprise a set of handy member functions:

- *add\_time()*: This member function can be called from anywhere in a PPRL process to capture a time value. Examples of such values are: linkage time and blocking time. It takes as a first argument a textual description of the value being captured, while the actual value is passed in the second argument. The function then adds the key-value pair to a member field dictionary in the Report instance.
- *init\_report()*: This member function is to be used to initialize the report scores for the first time. This can be useful in situations where the scores need to be hold across different iterations or experiment runs. In such scenarios, calling this function or the analogous *init\_blocking\_report()* outside the repetitive part of the code, will create the containers that will be later updated by calls to *update\_report()* member function.
- *update\_report()*: This member function is used inside a repetitive part (e.g., accumulating linkage quality scores across different encoding methods). Once called, it appends the passed scores to initialized member field containers. In our use case, this function takes as a parameter a dictionary of classified record pairs against a list of similarity thresholds. The function then extracts the TP, FP, FN and TN rates, and uses their values to calculate precision, recall and F-measure. These three linkage quality measures get, afterwards, accumulatively stored in respective containers for the different encoding methods.

- *report\_linkage\_time()*: This member function prints the time used for conducting the linkage over the different encoding methods and clear-text q-grams.
- *report\_linkage\_quality\_meas()*: This member function displays a comprehensive table of quality measures across different configurations and experiment runs. It uses our implementation of *display\_df()* function that we already described in Section 4.2. We show an example of the output of this function in Figure 8.

		TP	FP	FN	Precision	Recall	F-Meas.
Method	Sim. thres.						
q-gram	0.7	9561	7741	0	55.26%	100.00%	71.18%
	0.75	9515	3103	46	75.41%	99.52%	85.80%
	0.8	9479	668	82	93.42%	99.14%	96.19%
	0.85	9453	74	108	99.22%	98.87%	99.05%
	0.9	9370	16	191	99.83%	98.00%	98.91%
	0.95	8353	1	1208	99.99%	87.37%	93.25%
rbf	0.7	9579	9102	0	51.28%	100.00%	67.79%
	0.75	9555	3146	24	75.23%	99.75%	85.77%
	0.8	9509	696	70	93.18%	99.27%	96.13%
	0.85	9452	91	127	99.05%	98.67%	98.86%
	0.9	9387	21	192	99.78%	98.00%	98.88%
	0.95	8298	5	1281	99.94%	86.63%	92.81%
clk	0.7	9674	51249	0	15.88%	100.00%	27.41%
	0.75	9663	17445	11	35.65%	99.89%	52.54%
	0.8	9634	4427	40	68.52%	99.59%	81.18%
	0.85	9568	441	106	95.59%	98.90%	97.22%
	0.9	9512	33	162	99.65%	98.33%	98.99%
	0.95	9241	9	433	99.90%	95.52%	97.66%
clk-rbf	0.7	9681	71445	0	11.93%	100.00%	21.32%
	0.75	9669	24545	12	28.26%	99.88%	44.06%
	0.8	9651	5504	30	63.68%	99.69%	77.72%
	0.85	9578	576	103	94.33%	98.94%	96.58%
	0.9	9520	39	161	99.59%	98.34%	98.96%
	0.95	9302	9	379	99.90%	96.09%	97.96%

Figure 8: An example of a linkage quality report displayed by a *Report* instance

- *report\_blocking\_meas()*: This member function displays a table of blocking measures accumulated across multiple runs using our implementation of *display\_blocking()* function, described in Section 4.2. We show an example of the output of this function in Figure 9.

		5000	10000	20000
Measures	Method			
Blocking time (sec)	rbf	0.37	0.84	1.98
	clk	1.32	2.64	5.02
	clk-rbf	1.30	2.31	5.37
Linkage time (sec)	rbf	2.32	3.99	5.30
	clk	0.64	2.24	7.82
	clk-rbf	0.73	2.80	9.09
Number of matches found	rbf	4881	9662	18292
	clk	4841	9699	19290
	clk-rbf	4850	9708	19334
Number of candidate pairs	rbf	1235739	2076110	2743946
	clk	334770	1050646	3524003
	clk-rbf	386225	1227396	4034286
Pairs Completeness	rbf	0.98	0.97	0.91
	clk	0.97	0.97	0.96
	clk-rbf	0.97	0.97	0.97
Reduction Ratio	rbf	0.95	0.98	0.99
	clk	0.99	0.99	0.99
	clk-rbf	0.98	0.99	0.99

Figure 9: An example of a blocking report displayed by a *Report* instance

- *plot\_linkage\_quality\_fig()*: This member functions uses our implementation of *plot\_it()* function that we already described in Section 4.2. An example of the output of this function is Figure 11 or any similar figure in Chapter 5.

The *Report* class can be extended to more sophisticated reporting functionalities as we discuss in Section 4.3.2. Each *Experiment* instance composes one *Report* instance, that can be set through the *Experiment* member function *set\_report()*. This allows the Experiment to be decoupled from the reporting logic, which was one of the biggest drawbacks of the initial implementation. The *Experiment* class comes also with the member function *set\_reporting\_options()*, which allows the user to specify which elements to show or hide from the final report.

Based on the core classes in *framework.py*, we defined a set of concrete classes for hashing, encoding and blocking. In order to achieve this, we leveraged existing implementation from the original codebase which we composed inside wrapper classes of our definition. These concrete classes adhere to our framework API by extending the abstract classes. Examples of these classes include:

- *hashing.RandomHashingMethod* that uses a member field instance of *Ran-*



*domHashing* class, to which is delegates the hashing of q-gram sets inside its overridden *hash\_q\_gram\_set()* member function.

- *encoding.CLKEncoding* and *encoding.RBFEncoding* that use member field instances of *CryptoLongtermKeyBFEncoding* and *RecordBFEncoding* classes, respectively. Each of the wrapper classes delegates the encoding calls to its member field encoding instance inside its overridden *encode()* member function.
- *blocking.HLSHBlocking* that extends *framework.PostEncodingBlockingMethod* and calls the existing *hlsh\_blocking()* function, in our composed *linkageUtility.py* package, to conduct the blocking over Bloom filters inside its overridden member function *generate\_blocks()*. This class does not have a blocking class instance among its member fields as the existing code defines HLSH blocking as a function. However, we try to show here how this step is making heterogeneous implementations of different methods uniform in terms of our framework.
- *blocking.MinHashBlocking* that extends *framework.PreEncodingBlockingMethod* and calls *init\_minhash()* and *minhash\_blocking()* from *linkageUtility.py* package to conduct blocking over q-grams inside its overridden member function *generate\_blocks()*. The instance of this class gets initialized with the blocking parameters, which are for this method: *lsh\_band\_size* and *lsh\_num\_band*.

We finally show in Figure 10 the files structure of our developed framework, which will help us to explain its usage in the next section.

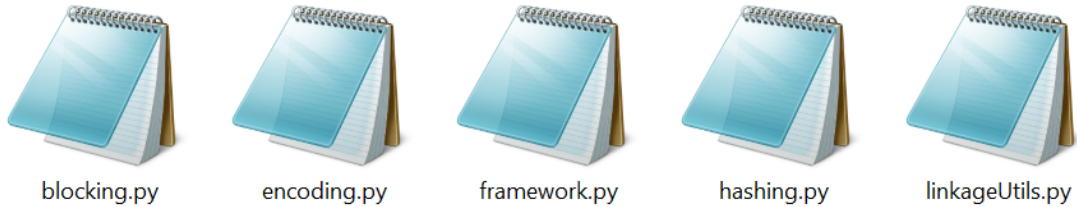


Figure 10: Structure of our proposed framework, composed of 5 packages

### 4.3.2 Usage

We show in this section how to extend our proposed framework, and how to use its classes to construct pipelines and design comparative evaluation experiments



than run different configurations and report varied results. First, we start by explaining how we extended the abstract core classes to define concrete classes for the different hashing, encoding and blocking methods. We illustrate that with the following selected examples:

We extended `HashingMethod` to define our concrete `CLKEncoding` class that composes an instance of an older `CryptoLongtermKeyBFEncoding` class as follows:

```
class CLKEncoding(EncodingMethod):
    # ...
    def set_encoding_params(self, attr_encode_tuple_list):
        self.encoder = CryptoLongtermKeyBFEncoding(attr_encode_tuple_list)

    def encode(self, attr_val_list):
        self.encoder.encode(attr_val_list)
```

We extended `PostEncodingBlockingMethod` to define our concrete `HLSHBlocking` class as follows:

```
class HLSHBlocking(PostEncodingBlockingMethod):
    def __init__(self, block_hlsh_num_seg):
        self.block_hlsh_num_seg = block_hlsh_num_seg
        self.MAX_BLOCK_SIZE = 100

    def set_max_block_size(self, size):
        self.MAX_BLOCK_SIZE = size

    def generate_blocks(self, rec_bf_dict):
        return hlsh_blocking(rec_bf_dict, self.block_hlsh_num_seg,
                               ↪ self.MAX_BLOCK_SIZE)
```

Similarly, the user can extend the relevant abstract class in order to make its new PPRL method satisfy the functions signatures expected by our framework. If the user has their newly developed technique already implemented in a class, they can define a wrapper class that embeds the functionality offered by their technique but exposes a uniform API to the framework, as we did for *CLKEncoding*. On the other hand, if the technique to be integrated is implemented in a function, the user can define a concrete class that calls the technique's function inside the class overridden member function, as we did for *HLSHBlocking*. It is worth mentioning here that newly added hashing, encoding and blocking methods are supposed to reside in *hashing.py*, *encoding.py* and *blocking.py* files, respectively. Any helper function that may be called inside a member function of a class, needs to be added



to *linkageUtils.py*. This latter file contains all implementations of utility functions used in different parts of the framework, e.g., *calc\_precision()*. This file was not depicted in our framework's Class Diagram in Figure 7 as it serves for helper functions and does not contain classes definitions.

Talking about the further extensibility of the framework, the user can do the following:

- Extend the *Pipeline* class to add further steps, e.g., preprocessing or post-processing, or to override the existing logic of each step.
- Extend the *Report* class to add further reporting functionalities or override the original implementation of each reporting functionality.
- Extend the *Experiment* class to modify the structure of the experiment design and to override existing functionalities. e.g., the way *run\_config()* function processes each run.

We show below an example of our framework usage inside a *main()* function:

```
# dataset files
data_set_file_name1 = 'Data/febr14_A_sorted.csv'
data_set_file_name2 = 'Data/febr14_B_sorted.csv'

# create an experiment instance and add the data files to it
exp = Experiment()
exp.add_data_file(data_set_file_name1, 1) # first file
exp.add_data_file(data_set_file_name2, 2) # second file

# list of targeted column indexes in the dataset
attr_num_list = [1, 2, 3, 4, 5, 6, 7, 8]

# list of dataset sizes (i.e., number of records to be loaded)
num_rec = [5000]

# list of similarity thresholds to classify candidate record pairs
↪ against
sim_thres_list = [0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8]

# list of number of hash functions to be used for Bloom filter
↪ construction
k = [10, 15, 20, 'opt']
```



```
# list of q-gram sizes (i.e., 2-grams or 3-grams)
q = [2, 3]

# list of padding options
padded = [True, False]

# create a hashing instance from RandomHashingMethod class
hash_method = hashing.RandomHashingMethod()

# set the hashing instance as default for the experiment
exp.set_hashing_method(hash_method)

#### Build pipelines ####
c1 = encoding.CLKEncoding()      # create an encoding instance
b1 = blocking.HLSHBlocking(50)  # create a blocking instance
p1 = Pipeline(c1, b1) # construct the pipeline
exp.add_pipeline(p1) # add the pipeline to the experiment

c2 = encoding.RBFEncoding()
b2 = blocking.MinHashBlocking(3,50)
p2 = Pipeline(c2, b2)
exp.add_pipeline(p2)

# run the experiment across the different configurations
exp.run(attr_num_list, sim_thres_list, num_rec, k, q, padded)
```

Our framework will be made available for public use on the following repository:  
<https://github.com/ali-arous/pprl-evaluation-framework>

# 5 Experimental Evaluation

## 5.1 Experimental Setup

In this section, we present our selected datasets for the experimental evaluation, and their workloads. We explain the different preprocessing steps we conducted to have them in an evaluation-ready state, and we discuss, after, our evaluation metrics for different experiment types. Finally, we describe the specifications of two evaluation environments that we tested on and discuss the methods and parameters involved in the experimental evaluation.

### 5.1.1 Data and Workloads

We used for our evaluation two types of datasets from two different domains. The first set of datasets are based on a US voter database for North Carolina state (NCVR) which is available for download at: <https://dl.ncsbe.gov/>.

This dataset covers over seven million voters in North Carolina state, where each record has an NCID attribute, representing a fixed and unique entity identifier for each individual voter. In addition to the full-state dataset (file: *ncvoter\_Statewise.zip*), voter records are available for download by counties (in smaller files from *ncvoter1.zip* to *ncvoter99.zip*). In order to conduct the linkage, two snapshots of the state-wide file, or smaller county files, need to be downloaded. The larger time difference between the two snapshots, the more changes in voter data can be expected mostly because people might move to new addresses or change their last names (due to marriage).

The second dataset pair is the *Freely Extensible Biomedical Record Linkage (FEBRL)*<sup>1</sup>, which is generated by the *FEBRL* software [20] as one data set with 10,000 records (5000 original record and 5000 duplicates, with one duplicate record per original one). Later, the 5000 original records are separated into the dataset file *dataset4a.csv*, similarly to the 5000 duplicate records which are kept in *dataset4b.csv*. This dataset pair (for short, *dataset*) can be used for testing record linkage, and has been used already by many record linkage studies [36, 86, 5]. We use this dataset to validate the observations we make on the NCVR dataset for the linkage

---

<sup>1</sup><https://recordlinkage.readthedocs.io/en/latest/ref-datasets.html>

quality experiments, and to assess the generalizability of our findings to datasets from different domains.

## Dataset Generation and Statistics

In order to generate a pair of datasets for NCVR with different sizes and controllable rate of error, we utilize the Python code in *datasetPairGenerator.py* provided by [23] and adapted by us to serve for our needs. This script takes as input two CSV files representing two voter snapshots in North Carolina state, finds voter IDs in common between them, and groups their corresponding records by the number of non-identical attribute values they exhibit. The script afterwards generates two linkage datasets spanning records from different error categories (based on the error rate). For this script to work properly, it is important that the two CSV files at the input follow the same structure, where columns representing corresponding attributes come in the same order in both files.

After investigating files from different NCVR snapshots, we noticed that the schema of the files was changing over time. For files between 2020 and 2021, that we decided to use for the datasets generation, we found the following differences in the attributes of our interest:

NCVR 2021		NCVR 2020	
Field	Position	Field	Position
ncid	68	ncid	4
last_name	9	last_name	11
first_name	10	first_name	12
middle_name	11	midl_name	13
res_street_address	13	res_city_desc	23
		state_cd	24
		zip_code	25
full_phone_number	24	area_cd	33
		phone_num	34
gender_code	28	sex_code	41
birth_age	29	age	43
birth_year	67	2020 - age	43

Table 1: Schema mapping of NCVR dataset snapshot in 2020 to its 2021 version

To reconcile both schemas, we conducted the following preprocessing steps on NCVR 2020:



- Concatenated the columns *res\_city\_desc*, *state\_cd* and *zip\_code* into a new column *res\_street\_address*.
- Concatenated the columns *area\_cd* and *phone\_num* into a new column *full\_phone\_number*.
- Derived the new column *birth\_year* by subtracting the age value from 2020.
- Moved each relevant column (including the newly created ones) to their corresponding positions in NCVR 2021.

After the previous steps, both files will become ready for ingestion by the datasets generation script. We note here that we only care about the positions of the columns of our interest (i.e., linkage columns), as the rest of the code picks each attribute value by the index of its column in the CSV file. We, hence, do not change the order of other non-relevant columns in NCVR 2020.

We applied the schema reconciliation and data preprocessing steps explained above on two pairs of files. First pair was a subset of the NCVR dataset comprising voter records in *YADKIN* county, which is the file *ncvoter99.zip* in the download page. This file corresponds to the most up-to-date voter data in that county for 2021. In order to have its counterpart from 2020, and since NCVR website did not store partial files per county for previous snapshots, we had to download the full state-wide 2020 NCVR snapshot, and extract the portion of it related to county id 99 (i.e., *YADKIN* county). We added the following steps to our preprocessing code in order to do so:

- Loaded the 2020 NCVR snapshot in a Pandas DataFrame in Python.
- Filtered for only the rows where *county\_id* = 99.
- From the remaining records, filtered for only rows with *voter\_status\_desc* = 'ACTIVE' (As we noticed a lot of records labeled as 'REMOVED' or 'INACTIVE' which did not have a correspondence in the 2021 file).
- Deleted the column *snapshot\_dt* from the DataFrame as it is a snapshot specific column.
- Saved the resulting DataFrame in the CSV file *'ncvoter99\_2020.csv'*.
- Following that, a few post-processing steps were conducted on the generated file, in order to correct encoding and remove empty lines.

From the pair of files *ncvoter99.csv* and *ncvoter99\_2020.csv*, we generated pairs of linkage datasets of varying sizes: 5000, 10,000 and 20,000.

We did most of the linkage quality experiments on these datasets as we saw later that the findings on them generalize to bigger dataset sizes on which analogous trends were observed.

The second pair of files on which the schema reconciliation and data preprocessing steps were run are: the full-state voter records file, namely *ncvoter\_Statewide.csv*, and its counterpart file of 2020 state snapshot. In this scenario, voter records for counties with ids from 80 until 99 were used to generate a dataset pair of 200,000 records and a dataset pair of 1,500,000 records (which we denote as 200K and 1.5M, respectively, and follow this convention for other datasets). After that, counties with ids from 50 to 99 were used to generate a dataset pair of 3,500,000 (3.5M) records. The voter attributes included in the generated datasets as QIDs for the linkage, are the ones at positions: {9, 10, 11, 13, 24, 28, 67} in each voter file. These positions correspond to: last\_name, first\_name, middle\_name, res\_street\_address, full\_phone\_number, gender\_code and birth\_year for 2021 files, and similarly but with: midl\_name instead of middle\_name, and sex\_code instead of gender\_code for 2020 files.

We show below the distribution of errors for the datasets with 20K, 1.5M and 3.5M records:

# same attributes	# matching record pairs		
	20K	1.5M	3.5M
7	3,220	1,108,046	2,645,832
6	15,900	325,595	779,871
5	762	29,777	75,165
4	609	2,158	6,402
3	39	196	657
2	9	13	41



We chose to illustrate these sizes as they represent the full sizes of the their respective files. These files were used throughout the experiments to sample datasets of different smaller sizes. For example, the dataset of size 5K, used in our linkage quality experiments, represents a subset of the dataset file with a total of 20K records.

For FEBRL dataset, the distribution of error is as follows (for 5000 records): 1563 records have 8 same attributes out of 9, 1692 records have 7, 1104 records have 6, 439 records have 5, 119 records have 4 and 29 records have 3. This distribution is more flat than for NCVR with more records distributed on the different error categories. However, we consider FEBRL as a cleaner dataset than NCVR as most errors in FEBRL are caused by one character change between corresponding attributes of records belonging to the same person, unlike to NCVR.



### 5.1.2 Evaluation Metrics

In our experimental setup, we follow the evaluation metrics settings discussed in [82] and widely used in previous PPRL evaluations [32, 13, 23].

For linkage quality metrics, Precision, Recall and F-measure (F1-Score) are used, and their values are justified by looking at the True Positive (TP), False Positive (FP) and False Negative (FN) rates. True Negative (TN) rate can be calculated by subtracting the other three rates from the dataset size. Whenever plotted, the figures of Precision, Recall and F-measure are shown side by side, with a fixed x and y scales.

When blocking methods are assessed, we planned to use Pairs Completeness (PC) as a measure of the effectiveness of the blocking technique (similar to recall) [81], Pairs Quality (PQ) as a measure of the efficiency of the blocking technique in terms of linkage quality (similar to precision), and Reduction Ratio (RR) which measures the effectiveness in terms of scalability to larger sizes of datasets, as explained in Section 2.7. We also record the runtime used for encoding, blocking and linkage in all experiments, but only include it in results and discussions when relevant. We later dropped PQ and RR from discussions as PC was more relevant for our results, and RR was always being reported at 0.99 across the different tests.

### 5.1.3 Evaluation Environment

All experiments on small and medium-sized datasets (i.e., up to 200,000 records) were conducted on a desktop machine with a 64-bit Intel Core i7 (1.80GHz) CPU, and 16GB of main memory. Experiments on larger datasets (i.e., up to 3.2 million records) were conducted on a server machine with 64-bit Intel Xeon (2.60GHz) CPU, and 128GB of main memory.

Python 3.7.3 was used for implementation and running of the experiments, including the libraries: pandas@1.3.0, numpy@1.21.1, clhash@0.16.1 and matplotlib@3.3.4.

### 5.1.4 Methods and Parameters

In our experiments, and as previously discussed in Chapter 3, we implement the PPRL process following the commonly used three-party protocol [13, 23, 32, 81], under the HBC model for the coordinator. That is, we assume to have two parties that represent data owners (DOs) who want to link their data with the help of a non-colluding linkage unit (LU). The other protocols, including two parties communicating directly with each others or multiple parties, are more complex and expensive in terms of computation and communication as they require more sophisticated encoding or encryption techniques [82, 23].

QID attributes included in the linkage are, as discussed in Section 5.1.1, selected using their positions (indexes) in the database file. We follow [23] for NCVR dataset by including QIDs at positions  $\{1, 2, 3, 4, 5, 6, 7, 8\}$  which correspond to:  $\{last\ name, first\ name, middle\ name, street\ address, city, zip\ code, gender, and\ birth\ year\}$ . For FEBRL we include the QIDs at positions  $\{1, 2, 3, 4, 5, 6, 7, 9\}$  which correspond to:  $\{given\ name, surname, street\ number, address\ 1, address\ 2, suburb, postcode, date\ of\ birth\}$ . We treat all QIDs as strings, that we either pad, *padded* = *True*, or not. These strings will be later split into q-grams where we test with  $q = 2$  and  $q = 3$ . In our experiments, we test with different dataset sizes, by either changing the dataset files to bigger ones, or by controlling the number of records read from a pair of files using *num\_rec* parameter. Later we add another parameter, *non\_matching*, to specify the number of records from one file that do not have any match in the other file.

### Bloom Filter Based Encoding:

We set the length of our generated Bloom filters *bf\_len* to the commonly used value of 1024, following [32]. Other work used the value of 1000 instead [74, 13, 23]. *Random hashing (RH)* is used to map q-gram tokens into Bloom filters as it provides a more secure alternative to the double hashing (DH) scheme [32, 23]. We hash each token a certain number of times by setting *num\_hash\_func*, which we denote as  $k$ , to varying values  $\{10, 15, 20, 'opt'\}$  where 'opt' refers to the dynamically calculated number of hash functions following the Formula 2.1.

Our selected Bloom filter based encoding techniques to take part in the evaluation are: *Record-level Bloom filter (RBF)*, *Cryptographic Long-term Key (CLK)*, and the hybrid *CLK-RBF*. Except for Experiment 5.2.5, we use RBF without attribute weighting (i.e., all attributes have the same weight of  $bf\_len / number\_of\_attributes$ ). Same arrangement is set for CLK-RBF. The difference between the two methods is that RBF uses the attribute weight to sample bits from its respective *attribute Bloom filter (ABF)*, whereas for the CLK-RBF implementation that we use, this weight can either be assigned as the number of set bits per each attribute in the resulting Bloom filter, or the number of set bits per each q-gram token. Both weights can vary per attribute/token. In case of equally weighting per token, it corresponds to the number of hash functions,  $k$ , used to hash tokens in CLK and RBF, which we follow for CLK-RBF, except for Experiment 5.2.5 where we introduce weighting per attribute. We previously presented the three methods in Section 2.5.

The similarity between generated Bloom filter pairs is calculated using the *Dice coefficient* following the Formula 2.3. We use the threshold based classification model, where all compared pairs sharing a similarity higher than a threshold  $\tau$  are classified as matches. We tested against similarity thresholds of:  $\{0.7, 0.75, 0.80,$

0.85, 0.90, 0.95} for NCVR dataset, and of: {0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.9} for FEBRL.

### LSH Based Blocking:

We use blocking to overcome the quadratic complexity of the linkage for larger datasets when assessing the linkage quality, but also to evaluate the quality and scalability of the blocking techniques themselves. For this, we pick two LSH based blocking, namely: MinHash LSH and Hamming LSH, previously presented in Section 2.6.1. We empirically determined the optimal parameters that lead to high blocking efficiency and effectiveness, which align with the recommended parameters in [23]. As a result, we set the LSH band size  $\mu = 3$ , and the number of LSH bands  $\lambda = 50$ .

## 5.2 Experiments on Encoding and Linkage Quality

In this class of experiments we conduct linkage quality evaluation over Bloom filters generated by each of CLK, RBF and CLK-RBF encoding methods, while changing other parameters that contribute to the overall linkage quality. The goal is to find an optimal set of parameters that can be recommended for similar settings as ours. We use such optimal parameters set to design further experiments that test interesting aspects of PPRL, such as the scalability of the solution and the generalizability of the results.

### 5.2.1 Testing the Effect of Padding with 2-grams

#### Experimental Design

In this experiment, we run the PPRL process in two scenarios, the first one with padded tokens, which we consider the default, and the other with unpadded tokens. For each scenario, we vary the number of hash functions,  $k$ , and set the other parameters to their default values as described in Section 5.1.4. We compare the encoding quality of the three Bloom filter based methods against the non-private record linkage over  $q$ -grams. We aim from this evaluation to compare the three encoding methods with each others, understand the effect of padded and unpadded tokens on the linkage quality with a changing number of hash functions, and see which PPRL configuration gives closer results to the non-private scenario. We think that deciding to use padding or not has an important impact on the generated Bloom filters, as it contributes to the overall number of set bits. Different techniques might have different responses to the increase or decrease in this number of set bits.

## Interpretation of the Results

After running the PPRL process over padded tokens, with  $k = 10$ , we get the results depicted in Figure 11:

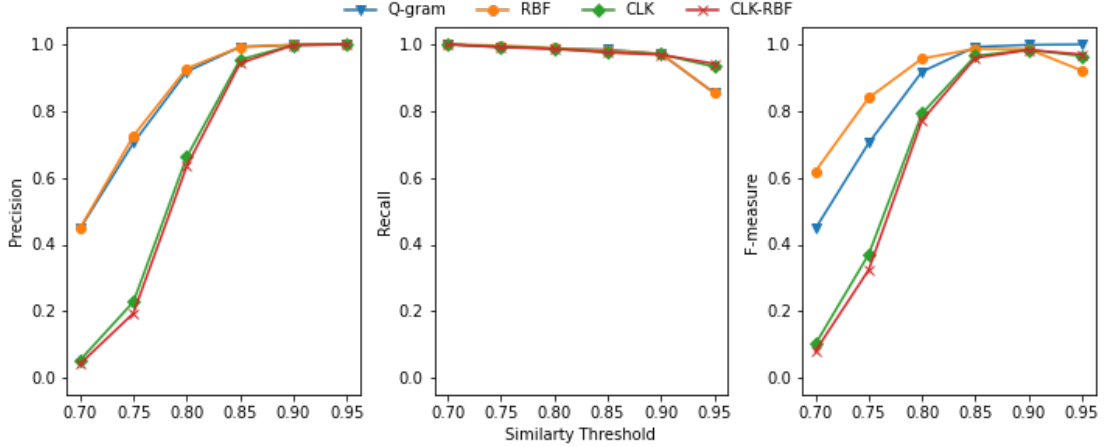


Figure 11: Linkage quality measures for NCVR 5K with padding and  $k = 10$

We see from the figure that q-gram and RBF have almost the same trend on precision, as is the case for CLK and CLK-RBF. The two pairs of methods start divergent from each other with a clear gap at lower thresholds, then converge at around  $\tau = 0.85$ . Looking at recall, we see that all methods start with a very high value (i.e., almost 1) and have a slight steady decrease until  $\tau = 0.9$ , where q-gram and RBF drop more than the other two methods. Combining both precision and recall trends we get the F-measure where we see a divergence between q-gram and RBF at lower thresholds and different drops among the four methods after  $\tau = 0.9$ .

To better understand the plots, we have a closer look at the numerical results of the experiment run. We notice that the highest linkage quality was achieved for q-gram at similarity threshold  $\tau = 0.85$ . This is where the linkage quality achieves the best compromise between the increasing precision (due to drops in FP rate) and the decreasing recall (due to increases in FN). In other words, by  $\tau = 0.85$ , linkage over q-grams would have removed enough FP candidate pairs to have a satisfying precision, while the side effect of increased TN candidate pairs is still tolerable. In this experiment run, this similarity threshold for q-gram corresponds to 99.25% precision, 98.43% recall and 98.84% F-measure. The great improvement of precision is due to the drastic drop in FP samples from 426 for the previous threshold of  $\tau = 0.8$  to just 36. Whereas the slight decrease in recall is caused

by the increased FN samples from 60 to 76. In all other thresholds, RBF gives slightly better scores than q-gram. Similarly, for RBF, the best linkage quality was recorded at threshold 0.85, with a very slight decrease of around 0.20% from q-gram measures of precision, recall and F-measure. For CLK, the linkage quality was recorded highest at  $\tau = 0.9$ , with better precision than RBF by 0.5%, but lower recall with 1% (both due to the increased threshold). The F-measure was around 0.30% lower than in RBF. However, at threshold  $\tau = 0.85$ , CLK still achieves its second best quality with F-measure of 96.56% (less than 2% drop from  $\tau = 0.9$ ), improved recall by 0.70% and decreased precision by 4.35%.

In general, we notice that the increase of 0.5 in threshold results in growing reduction in FP rate, starting at  $1/4$  or  $1/5$  of the rate at the previous threshold. CLK-RBF follows CLK in the trend and almost mimic its behavior at the different thresholds, scoring the highest linkage quality at  $\tau = 0.9$ . In all the different thresholds, CLK-RBF achieves slightly lower than CLK with a very small margin of 0.06% on F-measure at  $\tau = 0.9$ .

One observation from this experiment run, is that q-gram and RBF start at threshold  $\tau = 0.7$  with a FP rate of 5973 and 5953 respectively, which is around  $\times 1.2$  of the TP rate. Whereas CLK and CLK-RBF start at a FP rate of 86,929 and 112,279, which is around  $\times 18$  and  $\times 23$ , respectively. Because of this, the last two methods need one more round of threshold increment to achieve the similar reduction of FP rate and, hence, a comparable precision and F-measure to q-gram and RBF. When it comes to recall at  $\tau = 0.95$ , all methods demonstrate a drop due to the increase of FN rate. However, CLK and CLK-RBF result in less than half the FN rate compared to q-gram and RBF, which makes them more robust at high thresholds. In total, we find from this experiment run that RBF gives better overall linkage quality than CLK and CLK-RBF that is closer to the linkage quality over plain text, i.e., q-grams.

Re-running the experiment with  $k = 15$ , we get the results depicted in Figure 12:

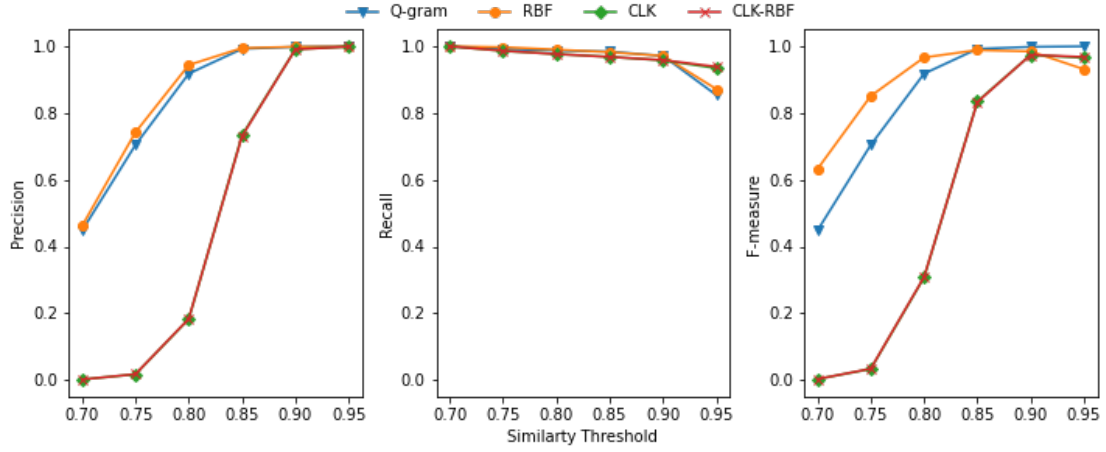


Figure 12: Linkage quality measures for NCVR 5K with padding and  $k = 15$

We observe a larger gap between the curves on precision and F-measure caused by a lowered precision for CLK and CLK-RBF at the earlier thresholds. In this experiment as with the subsequent ones, the q-gram will give the same results as it is independent of the changes in  $k$  value. For RBF, a slight increase can be observed in the measures, at the optimal threshold value, compared to the previous experiment run (with  $k = 10$ ). The reasoning here is that the increased rate of set bits, per hashed q-gram, boosts the discriminative power of the attributes and results in a slightly higher TP rate, and a slightly lower FP and TN rate (max. of 10 samples difference). However, the optimal threshold value remains at 0.85 with an F-measure of 98.88% and high precision and recall of 99.50% and 98.27%, respectively.

For CLK and CLK-RBF, and compared to the previous results with  $k = 10$ , we notice a drastic increase in the FP rate by  $\times 53$  and  $\times 42$  times, respectively, at the first threshold of  $\tau = 0.7$ . This corresponds to an initial precision approaching zero (0.11% and 0.10%), which can be explained by the collisions caused by the increased number of set bits. Only a slight increase in FN rate is observed which corresponds to a slightly lower recall (i.e., by around 1%). Again here, the threshold  $\tau = 0.9$  is the optimal one with an F-measure of 97.45% and 97.47% for RBF and RBF-CLK, respectively, and high precision and recall for both of them. In total,  $k = 15$  had slightly better results for RBF ( $\sim 0.20\%$  on F-measure) but lower results for CLK and CLK-RBF ( $\sim 1\%$  on F-measure). We present the numerical results recorded at the optimal thresholds of the three methods between  $k = 10$  and  $k = 15$  in Table 2.

Method	$k$	opt. $\tau$	TP	FP	FN	Precision	Recall	F-Meas.
rbf	10	0.85	4755	38	90	99.21%	98.14%	98.67%
	15	0.85	4765	24	84	99.50%	98.27%	98.88%
clk	10	0.9	4737	11	145	99.77%	97.03%	98.38%
	15	0.9	4757	41	208	99.15%	95.81%	97.45%
clk-rbf	10	0.9	4744	10	152	99.79%	96.90%	98.32%
	15	0.9	4759	43	204	99.10%	95.89%	97.47%

Table 2: Linkage quality measures at optimal  $\tau$  with  $k \in \{10, 15\}$

Re-running the experiment with  $k = 20$ , we get the results depicted in Figure 13:

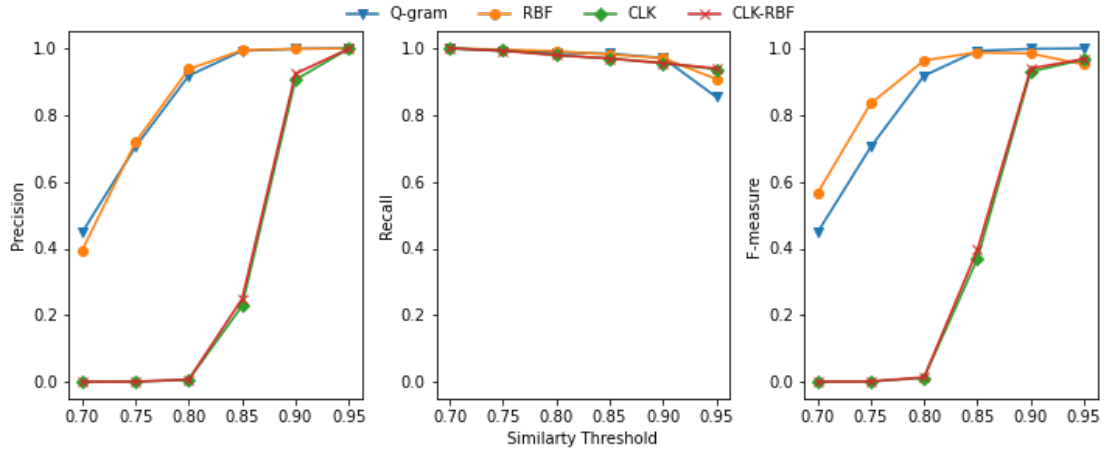


Figure 13: Linkage quality measures for NCVR 5K with padding and  $k = 20$

In this experiment run, we see that the gap between the curves on precision and F-measure gets bigger than for  $k = 15$ . We also observe a higher FP rate for RBF which stabilizes at  $\tau = 0.85$ , resulting in a slightly lower quality compared to the previous experiment run ( $k = 15$ ). However, for CLK and CLK-RBF, the FP rates start, at  $\tau = 0.7$ , with very high values of  $\times 5$  and  $\times 10$  times, respectively, of their initial values in the previous run, and only stabilize at  $\tau = 0.95$ . Hence,  $\tau = 0.95$  becomes the optimal threshold at  $k = 20$  for CLK and CLK-RBF, where both methods achieve best F-measure scores. At this threshold, and in comparison to  $\tau = 0.9$  in the same run, precision is boosted by around 9.5% for CLK and 7.5%

for CLK-RBF, while recall is decreased by around 2% for both methods. In total, best F-measure scores in this run are slightly lower than in the previous run (by less than 1%). We conclude that increasing the number of set bits per q-gram token, by increasing  $k$  above 15, results in a higher FP rate for the three encoding methods with much bigger influence on CLK and CLK-RBF. This high FP rate, which we attribute to the increased bit collisions, affects precision negatively and results, along with a slightly increased FN rate, in lower scores on the three quality measures. In addition, increasing  $k$  to 20 raises the optimal threshold for CLK and CLK-RBF to  $\tau = 0.95$ , at which recall drops by around 2% compared to optimal  $\tau = 0.9$  in the previous run. We present the numerical results recorded at the optimal thresholds of the three methods between  $k = 15$  and  $k = 20$  in Table 3.

Method	$k$	opt. $\tau$	TP	FP	FN	Precision	Recall	F-Meas.
rbf	15	0.85	4765	24	84	99.50%	98.27%	98.88%
	20	0.85	4770	27	89	99.44%	98.17%	98.80%
clk	15	0.9	4757	41	208	99.15%	95.81%	97.45%
	20	0.95	4682	6	313	99.87%	93.73%	96.71%
clk-rbf	15	0.9	4759	43	204	99.10%	95.89%	97.47%
	20	0.95	4694	5	301	99.89%	93.97%	96.84%

Table 3: Linkage quality measures at optimal  $\tau$  with  $k \in \{15, 20\}$

As a general conclusion from the the last three experiment runs, we find that  $k = 10$  results in the best scores on the three measures for CLK and CLK-RBF at threshold  $\tau = 0.9$ , and  $k = 15$  results in best scores on the three measures for RBF at threshold  $\tau = 0.85$ .

Re-running the experiment with  $k = \text{'opt'}$  we get the results depicted in Figure 14:



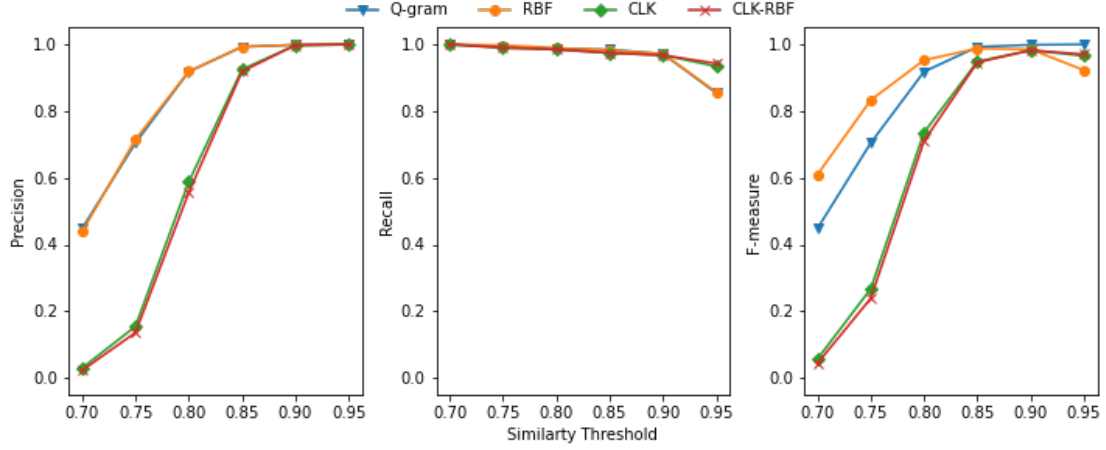


Figure 14: Linkage quality measures for NCVR 5K with padding and  $k = \text{'opt'}$

In this experiment we compare the results with the ones of the first experiment run ( $k = 10$ ) as the optimal number of hash functions,  $k_{opt}$ , was calculated at  $k = 11$ . We observe, for RBF, slightly higher scores for the quality measures at 0.85 threshold ( $\sim 0.10\%$ ), whereas for CLK and CLK-RBF, slightly lower scores ( $\sim 0.10\%$ ) at 0.9 threshold. Since RBF has already achieved its best scores at  $k = 15$ , we also compare the scores of this method with that experiment run. We observe that the difference is 0.11% behind the F-measure score for  $k = 15$ . These observations lead us to conclude that the calculated optimal value for  $k$  based on Formula 2.1 does not necessarily give the best scores for the different methods. This reasoning is supported by the fact that different methods demonstrate different behaviors under the same test settings as we saw from the previous experiment runs. We present the numerical results recorded at the optimal thresholds of the three methods at the relevant  $k$  values in Table 4.

Method	$k$	opt. $\tau$	TP	FP	FN	Precision	Recall	F-Meas.
rbf	10	0.85	4755	38	90	99.21%	98.14%	98.67%
	11	0.85	4760	33	86	99.31%	98.23%	98.77%
	15	0.85	4765	24	84	99.50%	98.27%	98.88%
clk	10	0.9	4737	11	145	99.77%	97.03%	98.38%
	11	0.9	4746	13	158	99.73%	96.78%	98.23%
clk-rbf	10	0.9	4744	10	152	99.79%	96.90%	98.32%
	11	0.9	4747	12	160	99.75%	96.74%	98.22%

Table 4: Linkage quality measures at optimal  $\tau$  with  $k \in \{10, 'opt'\}$  in addition to  $k = 15$  for RBF

After running the PPRL process over *unpadded* tokens, with  $k = 10$ , we get the results depicted in Figure 15:

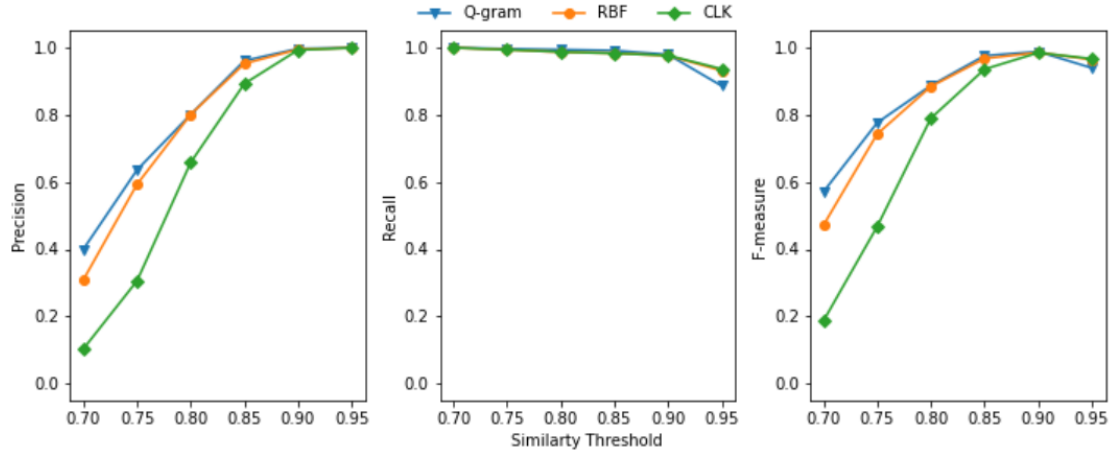


Figure 15: Linkage quality measures for NCVR 5K and  $k = '10'$ , without padding

We see in Figure 15 that q-gram and RBF show a divergence on the precision measure compared to Figure 11, while CLK becomes nearer to them than before. We observe in this experiment run that for  $k = 10$  with unpadded tokens, the scores in general went slightly lower at all threshold values for q-gram and RBF. The reason for this is a noticeable increase in FP rate at  $\tau = 0.7$  by a factor of  $\times 1.2$  for q-gram and  $\times 1.8$  for RBF, in addition to a little decrease in a few samples in the TP rate. This increase in FP rate is attributed to the less emphasize given to the first and last characters of each attribute in the unpadded setting. On the other hand, for CLK, the three scores went a bit higher despite the similar

decrease in TP. The observation behind this is the drop in FP rate by half, which contributes to a higher precision, and the slight decrease in FN rate which results in a bit higher recall. Both measures contribute in overall to a marginally higher F-measure (by 0.13%) at the optimal threshold of 0.9. We explain this drastic drop in FP rate in the unpadded setting by lower collision rate in bit positions. Since in the 2-gram setting, there will be two tokens less per attribute, the resulting Bloom filter ends up having  $2 \times k \times \text{number\_of\_attributes}$  less hashed positions, which increases the precision of CLK. We recall here that for settings where  $k$  was 15 or 20, this encoding method showed a great increase in FP rate caused by the higher ratio of collision. The CLK-RBF has no option to switch padding off, so we exclude it from this experiment. The optimal threshold for all methods here was  $\tau = 0.9$ .

Re-running the experiment (excluding CLK-RBF) on unpadded tokens with  $k = \text{'opt'}$  we get the results depicted in Figure 16:

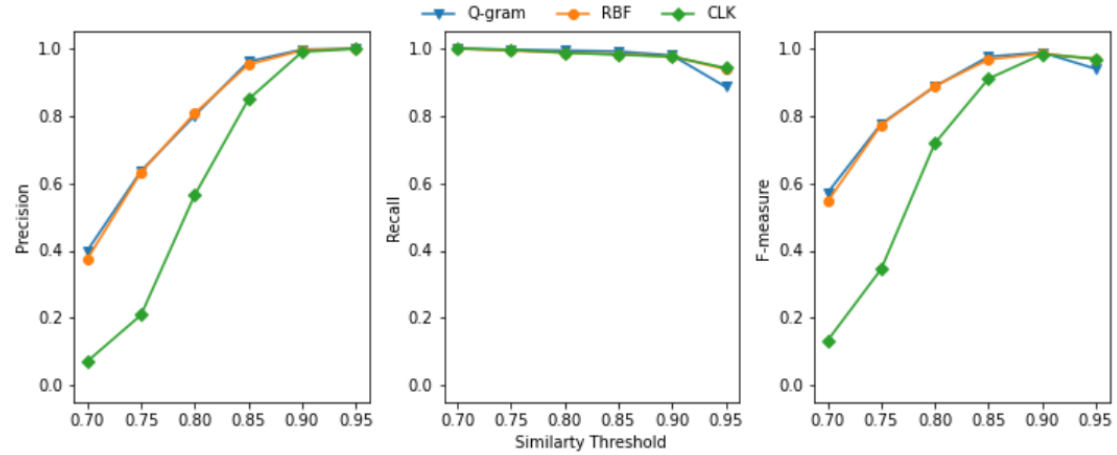


Figure 16: Linkage quality measures for NCVR 5K and  $k = \text{'opt'}$ , without padding

In this experiment run, we observe the same trend as with the previous one. We see, however, that the optimal number of hash functions,  $k_{opt}$ , was calculated at  $k = 12$  this time. This is, as we mentioned earlier, according to the Formula 2.1 where  $k$  depends on the number of tokens to be encoded,  $n$ , and in this unpadded scenario, we have a smaller  $n$ . By comparing the values of RBF and CLK between  $k = 12$  and  $k = 10$ , we see that for RBF it gives slightly better overall scores for  $k = 12$  at different thresholds, while the best score keeps the same as in  $k = 10$ . For CLK it gives slightly lower scores for  $k = 12$ . This goes a long with the observation made at  $k_{opt} = 11$  for the padded setting.

Having a look at the optimal thresholds for  $k = 15$  and  $k = 20$ , we see that RBF has slightly lower recall and F-measure scores for  $k = 15$ . However, we generally find in this unpadded setting that changing  $k$  in the set of  $\{10, 12, 15, 20\}$  has less impact on the linkage quality over RBF, compared to the padded setting. In both settings, RBF demonstrates resilience to the change in the number of hash functions. CLK also exhibits less impact on the linkage quality with the increase in the number of hash functions up to 20, compared to the padded setting. The reasoning is that having less tokens to encode per record (for 2-gram,  $2 \times \text{number\_of\_attributes}$ ) attenuates the effect of increasing  $k$  on the number of set bits in the Bloom filter and, hence, on the bit collision probability. We present the numerical results for the unpadded setting, recorded at the optimal thresholds of RBF and CLK methods for  $k \in \{10, 12, 15, 20\}$  in Table 5.

Method	$k$	opt. $\tau$	TP	FP	FN	Precision	Recall	F-Meas.
rbf	10	0.9	4720	23	114	99.52%	97.64%	98.57%
	12	0.9	4720	25	112	99.47%	97.68%	98.57%
	15	0.9	4721	23	123	99.52%	97.46%	98.48%
	20	0.9	4728	17	119	99.64%	97.54%	98.58%
clk	10	0.9	4734	32	111	99.33%	97.71%	98.51%
	12	0.9	4738	47	116	99.02%	97.61%	98.31%
	15	0.9	4745	92	132	98.10%	97.29%	97.69%
	20	0.95	4654	3	254	99.94%	94.82%	97.31%

Table 5: Linkage quality measures over unpadded tokens at optimal  $\tau$  with  $k \in \{10, 15, 20, 'opt'\}$

For the following experiments and based on the results discussed in the current experiment runs, we decide to use  $k = 10$  for the three encoding methods: RBF, CLK and CLK-RBF. While this value does not give the highest scores for RBF in the padded setting, it still allows this method to outperform the other two in terms of linkage quality. Moreover, having the same value for  $k$  will help to assess other parameters under uniform conditions, especially that a bigger value of  $k$  means a higher number of set bits which may have an overlapping effect with other parameters

## 5.2.2 Testing the Effect of Padding with 3-grams

### Experimental Design

In this experiment, and similarly to what we did in Experiment 5.2.1, we run the PPRL process in the two scenarios of with and without padding using  $q = 3$ . For

each scenario we fix  $k = 10$ , and set the other parameters to their default values as described in Section 5.1.4. We compare the encoding quality of the three Bloom filter based methods: RBF, CLK and CLK-RBF against the non-private record linkage over q-grams. We aim from this evaluation to understand the effect of increased token size,  $q$ , with or without padding, on the linkage quality against different similarity thresholds. We also want to find which PPRL configuration gives closer results to the non-private scenario. We think that the choice of  $q$  has an important impact on the generated Bloom filters, as it controls the tolerance for typographical errors in the attribute values [32]. In general, larger values for  $q$  are more sensitive to single character differences, e.g., the values 'Bloom' and 'Blaom' will have two bigrams ( $q = 2$ ) in common, which are: 'Bl' and 'om', while they have no common trigrams ( $q = 3$ ), in the unpadded setting.

### Interpretation of the Results

After running the PPRL process over padded trigrams, with  $k = 10$ , we get the results depicted in Figure 17:

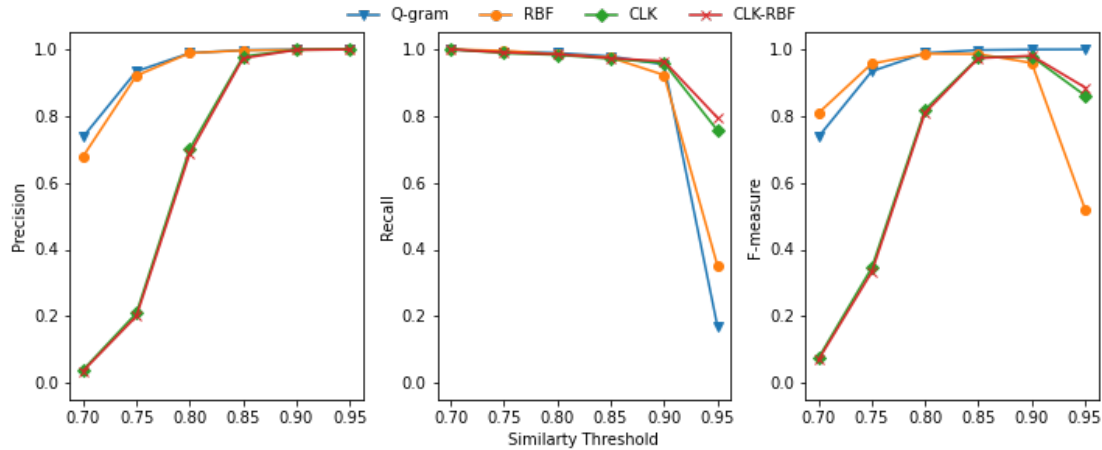


Figure 17: Linkage quality measures for NCVR 5K with padding,  $k = 10$  and  $q = 3$

We notice comparable best scores to Experiment 5.2.1 with  $q = 2$  setting. For q-gram and RBF, the scores are slightly higher for  $q = 3$  and achieve their maximum at  $\tau = 0.8$ . For CLK and CLK-RBF, they are slightly lower and achieve their maximum at  $\tau = 0.9$ . We also see a divergence between the curves at the smaller thresholds, where for  $\tau = 0.7$ , q-gram and RBF start with higher precision values of 73.75% and 67.91%, respectively, compared to only 3.86% and 3.55% for CLK

and CLK-RBF. The reason behind that is the FP rate, which is reduced by a factor of  $\times 0.28$  for q-gram and  $\times 0.38$  for RBF, from its recorded value at  $q = 2$ . This allows the linkage over the two methods to start with a higher precision and increase it with the progressing threshold. As a result, precision reaches the best compromise with the decreasing recall early at  $\tau = 0.8$ . Recall as well as F-measure are, hence, also higher for q-gram and RBF in this experiment run. On the other hand, for CLK and CLK-RBF, the FP rate relatively increases for  $q = 3$  at  $\tau = 0.7$  compared to its recorded value for  $q = 2$ . On parallel, we notice that the reduction in FP rate between consecutive thresholds is faster in this run, which yields a marginally higher precision. Recall, however, is a bit attenuated by a small increase in FN rate, and as a result, the F-measure for the two methods is a little less than recorded for  $q = 2$ . We explain the low FP rate for q-gram and RBF at  $\tau = 0.7$  by the longer token sequence, which makes it harder for corresponding attribute values to match if they contain errors, since they will have less tokens in common, as we explained earlier. This is also demonstrated by a slightly lower TP rate for  $q = 3$  compared to  $q = 2$ . The relative increase in FP rate for CLK and CLK-RBF at  $\tau = 0.7$  compared to  $q = 2$  can be attributed to increased number of set bits, and hence, increased collisions. The reasoning is that longer q-gram sequence results in more generated tokens from padded attribute values than a shorter sequence, e.g., 'Bloom' if padded will generate the 7 following trigram tokens: { \_B, \_Bl, Blo, loo, oom, om\_, m\_ }, whereas it only generates 6 bigram tokens: { \_B, Bl, lo, oo, om, m\_ }. The number of q-gram tokens resulting after padding from a string of length  $m$  is calculated as:  $m + (q - 1)$ .

Also, we notice that the drop in recall at  $\tau = 0.95$  is drastic for all methods but more drastic for q-gram and RBF with (95% to 17%, and 92% to 35%, respectively), in comparison to (96% to 76% and 96% to 80%) for CLK and CLK-RBF, respectively. This is caused by the big increase in FN rate at higher thresholds due to the decreased tolerance for spelling errors for 3-grams compared to 2-grams. The effect of collisions in CLK and CLK-RBF attenuates this decrease in tolerance.

Re-running the experiment (excluding CLK-RBF) on unpadded tokens with  $k = 10$ , we get the results depicted in Figure 18:

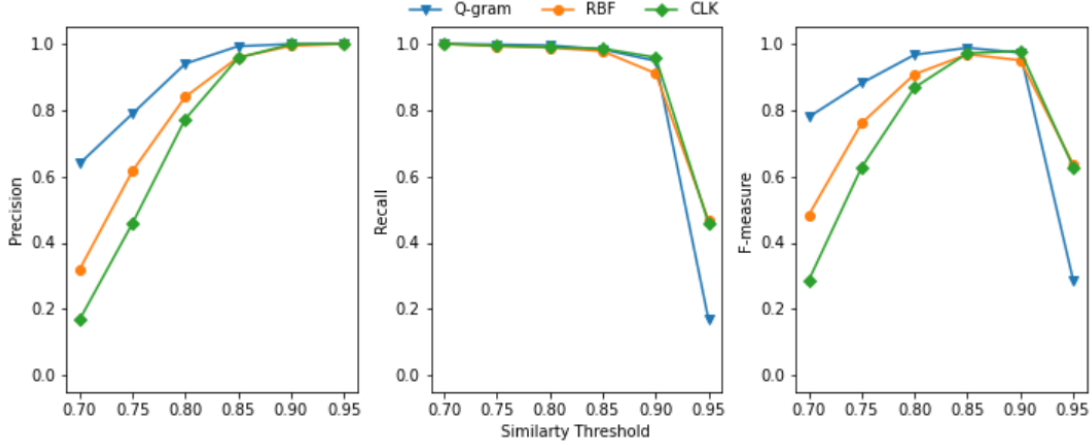


Figure 18: Linkage quality measures for NCVR 5K without padding,  $k = 10$  and  $q = 3$

We see, similarly to our observations in Experiment 5.2.1 for ( $k = 10$ ,  $q = 2$ , *padded* = *False*), that scores went in general slightly lower for almost all threshold values for q-gram and RBF. The reason for that is, as before, the noticeable increase in FP rate by a factor of ( $\times 1.5$  for q-gram and  $\times 4.5$  for RBF) compared to ( $\times 1.2$  for q-gram and  $\times 1.8$  for RBF for  $q = 2$ ) and the little decrease in a few samples in the TP rate. However, for CLK, and despite the similar decrease in TP rate as with the other two methods, the scores went higher by a factor of  $\times 2$  for  $\tau = 0.7$  and  $\tau = 0.75$ , and stabilized at  $\tau = 0.85$  and  $\tau = 0.9$ , with the maximum F-measure of 97.87% recorded at  $\tau = 0.9$  for ( $k = 10$ ,  $q = 3$ , *padded* = *False*) experiment. This F-measure value has a marginal difference of only 0.02% from the padded scenario. The observation behind this is, the decrease in FP rate by a factor of  $\times 0.2$  (compared to  $\times 0.5$  for  $q=2$ ), which contributes to a higher precision. This supports our finding from the previous run, that the bigger number of tokens caused by padding on trigrams resulted in more collisions, which went much less in this run, decreasing the FP rate (by a factor of  $\times 0.2$ ) as no padding was introduced. The CLK-RBF has no option to switch padding off, so we exclude it from this experiment as we did for  $q = 2$ .

In total, we conclude from this experiment run that using padding for trigrams results in a better overall linkage quality despite the differences observed between the methods.

We find from this experiment in its two runs, that using trigrams instead of bigrams in the padded setting have a marginal improvement for q-gram and RBF (with a shift in the optimal thresholds), and a marginal deterioration on CLK and CLK-RBF (even though this margin is bigger for CLK with 0.5% on F-measure).

Thus, we conclude that increasing the size of q-grams increases the variability between the methods, and between the optimal thresholds while not contributing to an observable improvement on linkage quality. In addition, using padding for trigrams results in a better overall linkage quality despite the differences observed between the methods. This leads us to using  $q = 2$  as a more stable value for comparing the encoding methods.

### 5.2.3 Testing Generalizability to a Second Dataset

In this experiment, we aim at re-running the Experiment 5.2.1 with the recommended setting of using bigrams and padding on the FEBRL [20] dataset, in order to gauge the generalizability of our findings to data from a different domain and with a different error rate. We think that this is very important for applicability of PPRL techniques to real-world projects.

#### Experimental Design

While we decided in Experiment 5.2.1 to use  $k = 10$  for the next experiments, we still need to validate here if that finding holds for other datasets. That is why we run again for the different values of  $k \in \{10, 15, 20, 'opt'\}$ . Based on initial runs, we modified the set of similarity thresholds at which we were testing the linkage quality to become  $\{0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.9\}$ . We explain in the following interpretation section the motivation behind this change. The set of QID attributes we include in our evaluation is  $\{\text{given\_name}, \text{surname}, \text{street\_number}, \text{address\_1}, \text{address\_2}, \text{suburb}, \text{postcode}, \text{date\_of\_birth}\}$  which excludes the *state* attribute. We decided on this subset in order to keep the number of the included attributes similar to the previous dataset, and since the *state* attribute did not exist in the NCVR dataset. All other parameters are set to their defaults including the use of  $q = 2$  and padding.

#### Interpretation of the Results

We depict the results of this experiment run in Figure 19:



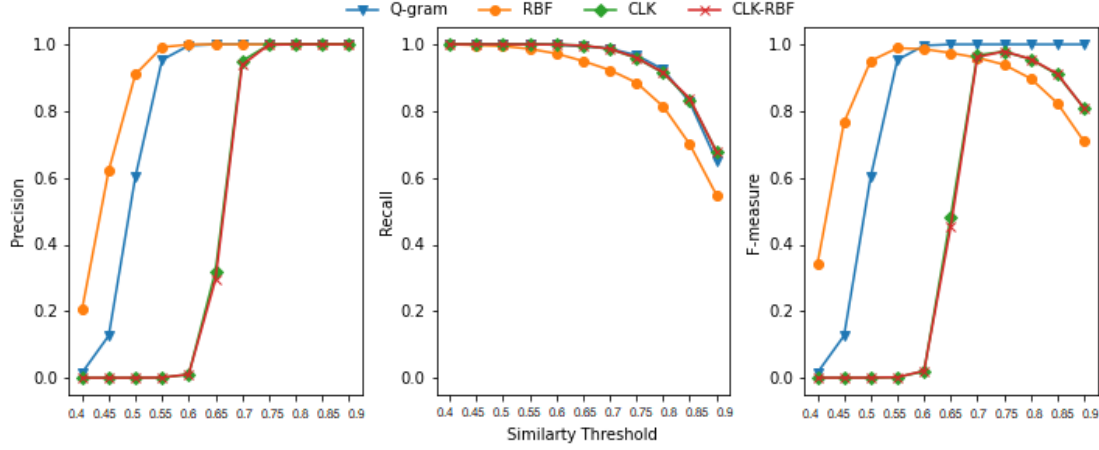


Figure 19: Linkage quality measures for FEBRL with padding,  $k = 10$  and  $q = 2$

Our observations in this experiment are analogous to the ones we made for NCVR dataset with some threshold shifts for the different methods. For example, q-gram on FEBRL dataset achieves its best scores at  $\tau = 0.6$  with 99.64% precision, 99.9% recall and 99.77% F-measure. These are slightly higher than the best scores recorded at  $\tau = 0.85$  on NCVR dataset with 99.25% precision 98.43% recall and 98.84% F-measure. The improved score for FEBRL dataset are caused by a higher TP, lower FP and a lower FN than in NCVR. Similarly, for RBF, the best linkage quality was recorded at  $\tau = 0.55$  with a slight decrease (less than 1%) from q-gram measures of precision, recall and F-measure. Compared to its best scores on NCVR at  $\tau = 0.85$ , we see that RBF performs better on FEBRL with slightly improved recall and F-measure and a slightly decreased precision (a margin of less than 0.5% on the three measures). As we have seen before for NCVR, CLK and CLK-RBF require a higher threshold to demonstrate comparable linkage quality, where CLK achieves its best F-measure at  $\tau = 0.75$ , with slightly better precision than RBF but a lower recall of 2.74%. This drop in recall is bigger than what was recorded on NCVR (around 1%), and results in a slightly lower linkage quality on FEBRL than on NCVR. CLK-RBF follows CLK with achieving its best scores at  $\tau = 0.75$  on FEBRL, with slightly better linkage quality than CLK on the three measures (less than 0.15%), but slightly lower recall and F-measure than its performance on NCVR. For both CLK and CLK-RBF, an improved recall can be attained by threshold  $\tau = 0.7$  with compromising the precision, which gives better linkage quality for CLK than CLK-RBF. Talking about the difference in FP rate that we observed on NCVR, we see that it also applies to FEBRL. Both CLK and CLK-RBF demonstrate much higher FP rate at  $\tau = 0.45$  compared to q-gram

and RBF where the FP is even less than TP. In general, we see that the different methods follow similar trends on FEBRL to what they demonstrated on NCVR. One interesting observation here is that RBF, while still closer to q-gram than others, shows a slight divergence from its curve on FEBRL, whereas on NCVR it almost mimics q-gram curve. We explain the shifts in optimal thresholds in this experiment run compared to NCVR by the different nature of data, which contains a different error rate and different attributes.

We depict in Figure 20 the linkage quality in terms of F-measure recorded at optimal thresholds over FEBRL dataset in comparison to NCVR with the increasing value of  $k$ .

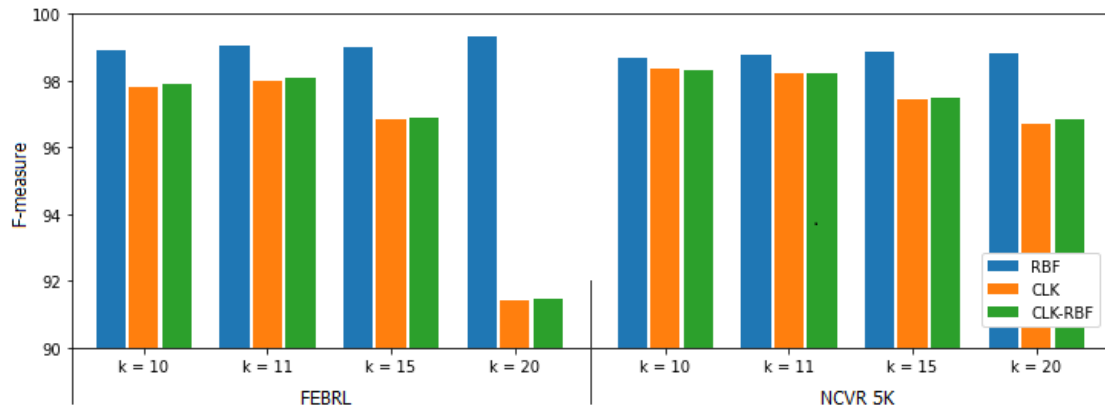


Figure 20: Linkage quality for FEBRL and NCVR 5K with increasing  $k$

For  $k = 15$ , and as seen before with applying the same settings on NCVR, the q-gram will give the same results as it is independent of the changes in  $k$  value. For the remaining three methods, the same general conclusions that we made before on NCVR apply here, with the following observations:

- RBF achieves its best score at  $\tau = 0.6$  with 99.02% on F-measure compared to 98.90% for  $k = 10$  achieved at  $\tau = 0.55$ .
- CLK achieves its best score at  $\tau = 0.8$  with 96.85% on F-measure compared to 97.83% for  $k = 10$  achieved at  $\tau = 0.75$ .
- CLK-RBF achieves its best score at  $\tau = 0.8$  with 96.88% on F-measure compared to 97.92% for  $k = 10$  achieved at  $\tau = 0.75$ .
- The increase in  $k$  corresponds to a slightly higher TP rate for the three encoding methods as seen before on NCVR dataset.

- The increase in  $k$  corresponds to a lower FN rate unlike to the observation on NCVR where the FN rate went slightly higher.
- The observation of drastic increase in FP rate for CLK and CLK-RBF on the NCVR data holds here as well, with a factor of  $\times 45$  for CLK and  $\times 47$  for CLK-RBF at  $\tau = 0.6$ , instead of  $\times 53$  for CLK and  $\times 42$  for CLK-RBF at  $\tau = 0.7$  on NCVR.
- In general, increasing the number of hash functions dampens the precision, especially at earlier thresholds, and boosts the recall.

For  $k = 20$ , the same general conclusions that we made before on NCVR apply here, with the following observations:

- RBF starts at a high FP rate and stabilizes at  $\tau = 0.6$  (compared to  $\tau = 0.85$  on NCVR), resulting in a slightly higher linkage quality of 99.31% on F-measure composed of (99.94% precision and 98.68% recall) compared to 98.63% F-measure on NCVR (99.59% precision, 97.68% recall).
- For CLK and CLK-RBF, the threshold at which the very high initial value of FP rate starts to drastically decrease becomes at 0.75 compared to 0.65 for  $k = 15$ , which postpones the recording of the best scores until  $\tau = 0.9$ . This is similar to the observation on NCVR where the best scores were recorded at  $\tau = 0.95$  (we recall here that there is a threshold shift by which the scores start to be analogous between the two datasets). However, CLK and CLK-RBF show a decreased F-measure score (by around 5%) from its best value at  $k = 15$  on FEBRL compared to a slight decrease by less than 1% on F-measure from  $k = 15$  to  $k = 20$  on NCVR.
- The FN rate is not increased in this experiment with the increase of  $k$  from 15 to 20, but rather slightly decreased. This diverges from the observed behaviour on NCVR. The reasoning is that the increased number of hashed bit positions caused by a larger  $k$  does not compromise the recall in spite of the higher collision rate. It rather boosts it a little if the records shared higher similarities with each others (i.e., coming from a cleaner dataset as with FEBRL compared to NCVR).

As a general observation, we see that at the different values of  $k$ , and for the increasing threshold starting from 0.4, q-gram and RBF demonstrate a relatively small FP rate compared to CLK and CLK-RBF. At lower values of  $k$ , beginning at 10, q-gram still gives a FP rate that is 17 times bigger than the one recorded for RBF. However, as  $k$  increases towards 20, this gap becomes progressively smaller, as the FP rate for RBF grows to reach the same magnitude of q-gram's at  $k = 20$ .

(we recall here that q-gram is independent of  $k$ ), where FP rate for q-gram only equals 3 times the FP rate for RBF. This explains the difference in the optimal thresholds between the two methods, with 0.6 for q-gram, and 0.55 for RBF at  $k = 10$ . On the other hand, CLK and CLK-RBF start at a very high FP rate for all  $k$  values, while the smaller the  $k$ , the faster the FP rate drops between consecutive thresholds. This explains why the best scores are recorded for  $k = 10$  at 0.75, for  $k = 15$  at 0.8, and for  $k = 20$  at 0.9, for both methods.

For  $k = opt$ , it corresponds to 11 in this experiment, similar to its calculated value on NCVR. We observe a slight improvement in linkage quality for RBF at its optimal threshold (by 0.16% on F-measure), compared to the recorded scores at  $k = 10$ . Similarly, both CLK and CLK-RBF achieve a slightly higher linkage quality at  $k = 11$  (by 0.18% for both methods on F-measure). This is caused by improved recall but a slightly decreased precision for CLK and CLK-RBF. In comparison to NCVR, these two methods showed a slightly increased linkage quality at the higher  $k$  value of 11. However, as we concluded before for the effect of higher  $k$  values over clean data, it can contribute to a higher recall which in the end improves the overall F-measure.

## 5.2.4 Testing with Different Subsets of Attributes

In this experiment, we aim at understanding the impact of the different QID attributes on the overall linkage quality. For this purpose, we test with different subsets of attributes by excluding some of them from the main set and observing the linkage quality after running the PPRL process on the resulting set. We think that finding an optimal set of QID attributes, and making conclusions on the impact of specific attributes on the linkage quality has a great importance on designing future PPRL pipelines.

### Experimental Design

In this experiment, we run the PPRL process over NCVR 5K dataset, following the default values of the parameters that we fixed for this dataset in the previous experiments. Hence, we set  $k = 10$  to be used with  $q = 2$  in the padded setting. The similarity thresholds vary from 0.7 to 0.95 with a step of 0.5. The attributes that we decide on progressively excluding in this experiment are mostly the ones that hold volatile information, e.g., *res\_street\_address* [13]. These attributes are prone to changes over time which is why they are considered volatile, compared to non-volatile ones, e.g., date of birth. We ranked such attributes from the most volatile one to the least volatile one as follows: *res\_street\_address*, *res\_city\_desc*, *zip\_code*, *last\_name* (due to marriage) and *middle\_name* (due to missing values).

## Interpretation of the Results

We show in Table 6 the linkage quality results of running the PPRL process on different subsets of QID attributes. The entries for the subsets are listed incrementally, i.e., each entry shows which QID attribute is excluded from the main set or the previous subsets. We denote the main attribute set as  $S$ , while referring to the following attributes  $\{res\_street\_address, res\_city\_desc, zip\_code, last\_name, middle\_name\}$  by their original indexes in the main set  $\{5, 4, 6, 1, 3\}$ , respectively.

QID subset	Method	opt. $\tau$	TP	FP	FN	Precision	Recall	F-Meas.
$S$	q-gram	0.85	4761	36	76	99.25%	98.43%	98.84%
	RBF	0.85	4755	38	90	99.21%	98.14%	98.67%
	CLK	0.9	4737	11	145	99.77%	97.03%	98.38%
	CLK-RBF	0.9	4744	10	152	99.79%	96.90%	98.32%
$A = S - \{5\}$	q-gram	0.9	4783	11	175	99.77%	96.47%	98.09%
	RBF	0.9	4775	12	123	99.75%	97.49%	98.61%
	CLK	0.9	4786	18	201	99.63%	95.97%	97.76%
	CLK-RBF	0.9	4789	19	202	99.60%	95.95%	97.74%
$B = A - \{4\}$	q-gram	0.8	4949	70	46	98.61%	99.08%	98.84%
	RBF	0.85	4838	23	149	99.53%	97.01%	98.25%
	CLK	0.85	4877	19	113	99.61%	97.74%	98.66%
	CLK-RBF	0.85	4896	23	95	99.53%	98.10%	98.81%
$C = B - \{6\}$	q-gram	0.85	4922	18	58	99.64%	98.84%	99.23%
	RBF	0.85	4921	22	54	99.55%	98.91%	99.23%
	CLK	0.85	4929	24	49	99.52%	99.02%	99.26%
	CLK-RBF	0.85	4933	27	47	99.46%	99.06%	99.26%
$D = C - \{1\}$	q-gram	0.85	4787	1023	178	82.39%	96.41%	88.85%
	RBF	0.85	4702	1211	265	79.52%	94.66%	86.43%
	CLK	0.85	4848	1074	118	81.86%	97.62%	89.05%
	CLK-RBF	0.85	4845	1029	125	82.48%	97.48%	89.36%
$E = C - \{3\}$	q-gram	0.85	4902	305	74	94.14%	98.51%	96.28%
	RBF	0.85	4838	315	137	93.89%	97.25%	95.54%
	CLK	0.85	4924	326	55	93.79%	98.90%	96.28%
	CLK-RBF	0.85	4931	335	48	93.64%	99.04%	96.26%

Table 6: Linkage quality measures using different subsets of attributes on NCVR 5K

We make the following observations from the results in Table 6:

- Excluding *res\_street\_address* (index 5) lowers the recall for all methods by around 1%, while it boosts the precision for q-gram and RBF by around 0.5%. As a result, F-measure is decreased for all methods. In this setting we notice that the optimal threshold was reached at 0.9 for all methods.

- Excluding *res\_city\_desc* (index 4) boosts the recall for all methods but RBF by around 2% while slightly lowering the precision. As a result, F-measure is improved for all methods but RBF. Optimal threshold for q-gram is reached at 0.8 while for the rest is reached at 0.85.
- Excluding *zip\_code* (index 6) boosts the recall for all methods but q-gram by 1-2% and only lowers the precision by less than 0.1%. F-measure is, hence, improved for all methods but q-gram, and the optimal threshold is reached at 0.85 for all methods.
- Excluding *last\_name* (index 1) deteriorates quality scores on the three measures for all methods, while F-measure score is 10-13% less than the previous setting.
- Including *last\_name* again but excluding *middle\_name* (index 3) lowers the precision by 5-6% and slightly the recall by 0.33-1.66%. F-measure is, as a result, decreased by around 3-4%

Based on the previous observations and by comparing the scores in Table 6, we conclude that excluding *res\_street\_address*, *res\_city\_desc* and *zip\_code* from the main set of attributes results in the best linkage quality, making the subset  $\{last\_name, first\_name, middle\_name, gender\_code, birth\_year\}$  the optimal set of attributes for conducting a PPRL process over NCVR 5K. Indeed, this set includes the most stable QID attributes that describe the voter’s personal details. These details are less likely to change over time compared to e.g., the address or city of residence information. We see that excluding one of the volatile attributes like the *res\_street\_address* while keeping other correlated ones, like *res\_city\_desc* or *zip\_code* does not help in improving the discriminative power but rather deteriorates it. Only when the three address-related attributes are excluded, the classifier is able to decide with a better certainty on the correct matches. This optimal set also normalizes the optimal threshold at 0.85 for all methods with very similar linkage quality scores (margin of 0.03%). Another interesting conclusion is that the last name, and in contrast to other arguments [13], does not hold volatile information, as removing it will drastically drop the quality measures. This means that it is very less likely for people to change their last names for marriage or whatever other reason in North Carolina state. On the other hand, the middle name is, as we expected, less important than the first and last names. Removing it will decrease the linkage quality by 1/3 the extent caused by a missing last name.

## 5.2.5 Testing with Weighted Attributes

### Experimental Design

In this experiment, we aim at understanding the impact of assigning different weights to different QID attributes on the overall linkage quality. For this purpose, we modify the usage of RBF and CLK-RBF to enable attribute weighting, while excluding CLK as it does not support this setting. We run over three configurations:

1. We include all attributes in the default attributes set i.e., of indexes  $\{1, 2, 3, 4, 5, 6, 7, 8\}$  with the corresponding weight vector  $(0.2, 0.2, 0.1, 0.05, 0.025, 0.025, 0.2, 0.2)$ . This gives equal higher importance to the last and first name, gender and year of birth. Middle name is given half the weight of last and first names, based on our conclusion about its impact from Experiment 5.2.4. Street address is given half the weight of middle name, and each of city and zip code are given half importance of street address. We denote this configuration as  $W1$ .
2. Similar to the previous configuration but with a weigh vector of  $(0.2, 0.2, 0.1, 0.033, 0.033, 0.033, 0.2, 0.2)$ . Here we give similar less importance to the address, city and zip code attributes, while the rest take the same weights as in  $W1$ . We denote this configuration as  $W2$ .
3. We include attributes of indexes  $\{1, 2, 3, 7, 8\}$  that correspond to the best attributes subset  $\{last\_name, first\_name, middle\_name, gender\_code, birth\_year\}$  found in Experiment 5.2.4. The weigh vector is set to  $(0.3, 0.3, 0.1, 0.15, 0.15)$  by which we emphasize the first and last names and give higher importance to birth year and gender than middle name. We denote this configuration as  $W3$ .

### Interpretation of the Results

We show in Table 7 the linkage quality results of running the PPRL process on the three configurations:  $W1$ ,  $W2$  and  $W3$ , in comparison to the corresponding arrangements from Experiment 5.2.4:  $S$  that represents a non-weighted setting for  $W1$  and  $W2$ , and  $C$  that represents the same for  $W3$ .



Config.	Method	opt. $\tau$	TP	FP	FN	Precision	Recall	F-Meas.
$S$	RBF	0.85	4755	38	90	99.21%	98.14%	98.67%
	CLK-RBF	0.9	4744	10	152	99.79%	96.90%	98.32%
$W1$	RBF	0.75	4935	79	38	98.42%	99.24%	98.83%
	CLK-RBF	0.9	4938	23	61	99.54%	98.78%	99.16%
$W2$	RBF	0.8	4909	36	71	99.27%	98.57%	98.92%
	CLK-RBF	0.9	4942	27	57	99.46%	98.86%	99.16%
$C$	RBF	0.85	4921	22	54	99.55%	98.91%	99.23%
	CLK-RBF	0.85	4933	27	47	99.46%	99.06%	99.26%
$W3$	RBF	0.9	4883	30	91	99.39%	98.17%	98.78%
	CLK-RBF	0.9	4963	233	36	95.52%	99.28%	97.36%

Table 7: Linkage quality measures using different attributes weighting on NCVR 5K

We see from the table that the first weighting scheme of  $W1$  boosts recall by around 1% for RBF and 2% for CLK-RBF while slightly compromising precision by less than 1% for both methods. This improved the overall linkage quality on F-measure score. On the other hand, the second weighting scheme of  $W2$  only improved a little for RBF due to slightly improved precision and less decreased recall, while for CLK-RBF it remained at the same quality as in  $W1$  with marginally better recall and less better precision. We find that  $W1$  is very close to this one with contrasting behaviors demonstrated by the two methods across the two configurations. For  $W3$ , and compared to the non-weighted setting of  $C$ , we find that linkage quality over both methods drops with more impact observed on CLK-RBF of around 2% on F-measure compared to less than 0.5% for RBF. The drop on CLK-RBF is attributed to an observable increase in FP rate from 27 to 233 which lowers the precision by around 4%, whereas for RBF it is due to an increase in FN rate and a less increase in FP rate, which lower both recall and precision. This means that the five personal attributes used in this setting are more or less equally important for the discriminative power.

We conclude from the previous observations that weighting has an important impact on the overall linkage quality of a PPRL process. However, it needs to be introduced with care not to compromise an already good achieved quality. We think that weighting can be used as a more flexible alternative to the exclusion of less important attributes, as it can attenuate or emphasize attributes of choice. This gives a wider range of options than eliminating some attributes that could still be relevant. For example, CLK-RBF gives approximate results in  $W1$  and  $W2$  configurations, with all attributes included, to its recorded values for the non-weighted best attributes subset,  $C$ , with only 0.1% less on F-measure.



## 5.3 Experiments on Blocking, Linkage Quality and Scalability

In the following experiments we conduct blocking using LSH in two scenarios, namely Min-hash based LSH blocking and Hamming based LSH blocking (HLSH), that were previously presented in Section 2.6.1. The first method applies on q-grams and generates blocking dictionaries that can be used by different encoding methods in the PPRL process. In other words, the blocking in this setting happens once over q-grams before the encoding step and the generated blocks are used by all methods to encode the records within. This corresponds to the alternative flow in Figure 1. On the other hand, Hamming based LSH blocking runs over Bloom filters after the encoding step, which means that blocking takes place over different Bloom filters for each encoding method. This corresponds to the main flow in Figure 1. In this setting, linkage over q-gram can use blocks generated by one of the encoding methods.

### 5.3.1 Testing MinHash Blocking for Parameters Tuning

#### Experimental Design

In this experiment we run the PPRL process on NCVR 5K dataset using Min-Hash LSH as a blocking technique and RBF, CLK and CLK-RBF as encoding techniques, in addition to the clear-text q-grams. We aim from this experiment at understanding the effect of MinHash blocking on the linkage quality, which will help to assess the possibility of mapping the results obtained from bigger datasets (using MinHash blocking) to the no-blocking scenario. Moreover, since we already conducted previous experiments on NCVR 5K dataset without blocking, we are interested to see the reduction in linkage time over each encoding method if blocking using MinHash was employed. We compare two recommended settings [23] of MinHash: The first is with band size,  $\mu = 4$ , and number of bands,  $\lambda = 100$ , that we denote as *MinHash-4-100*, while the second is with  $\mu = 3$  and  $\lambda = 50$ , which we denote as *MinHash-3-50*. For the encoding methods, we use the selected parameters from the linkage quality experiments, namely  $q = 2$  and  $k = 10$  with padding, which we will consider as defaults for the subsequent experiments. To avoid varieties in the testing environment when we measure the time, we run each configuration in three identical runs and record the average time among the three runs. This holds for all subsequent experiments with the exception of Experiment 5.3.4.

## Interpretation of the Results

Running the PPRL process using each of MinHash-4-100 and MinHash-3-50, and measuring: the average time used for hashing each dataset in the pair,  $Avg. t_{hashing}$ , the time used for blocking,  $t_{blocking}$ , the number of preserved true match pairs,  $TM$ , and pairs completeness,  $PC$ , we record the results in Table 8.

Method	Avg. $t_{hashing}$ (sec)	$t_{blocking}$ (sec)	TM (/4999)	PC (%)
MinHash-4-100	48.41	97	4975	99.51
MinHash-3-50	23.41	47	4981	99.64

Table 8: Blocking measures for MinHash-4-100 and MinHash-3-50 on NCVR 5K

From this comparison we see that MinHash-3-50 takes less than half the average time for calculating the records MinHash signatures compared to MinHash-4-100. It also takes less than half the time used for blocking and gives a better blocking quality with a higher number of preserved matches and, hence, a higher pairs completeness measure. This can be explained by the smaller band size and smaller number of blocks, respectively.

Measuring the time taken by the linkage over different encoding methods, we get the results presented in Table 9

Method	No Blocking	MinHash-4-100	MinHash-3-50
q-gram	141.96	11.03	11.37
RBF	55.47	5.11	5.31
CLK	54.93	5.06	5.50
CLK-RBF	58.74	5.72	5.71



Table 9: Linkage time (in seconds) with blocking using MinHash-4-100 and MinHash-3-50 on NCVR 5K

As we can see, both blocking settings reduce the linkage time drastically by more than  $\times 0.1$ . MinHash-3-50 takes slightly more time because it generates fewer but larger blocks than MinHash-4-100. Linkage over larger blocks takes more time inside each block compared to smaller blocks, which can eventually lead to a larger overall linkage time.

Measuring the linkage quality for the three encoding techniques and q-gram using the two blocking settings compared to the no-blocking scenario, we get the results presented in Table 10.

	method	opt. $\tau$	TP	FP	FN	Precision	Recall	F-Meas.
No Blocking	q-gram	0.85	4761	36	76	99.25%	98.43%	98.84%
	RBF	0.85	4755	38	90	99.21%	98.14%	98.67%
	CLK	0.9	4737	11	145	99.77%	97.03%	98.38%
	CLK-RBF	0.9	4744	10	152	99.79%	96.90%	98.32%
MinHash-4-100	q-gram	0.85	4760	36	76	99.25%	98.43%	98.84%
	RBF	0.85	4754	38	90	99.21%	98.14%	98.67%
	CLK	0.9	4736	11	141	99.77%	97.11%	98.42%
	CLK-RBF	0.9	4743	10	147	99.79%	96.99%	98.37%
MinHash-3-50	q-gram	0.85	4760	36	76	99.25%	98.43%	98.84%
	RBF	0.85	4754	38	90	99.21%	98.14%	98.67%
	CLK	0.9	4736	11	143	99.77%	97.07%	98.40%
	CLK-RBF	0.9	4743	10	150	99.79%	96.93%	98.34%

Table 10: Linkage quality measures using MinHash-4-100 and MinHash-3-50 on NCVR 5K

As we can see from TP and FP rates, only one sample drop is observed in TP for all methods after MinHash blocking, while FP keeps the same values before and after. The drop of one sample on TP means that the records removed by the blocking technique only affect one true match pair. However, this one sample drop has no observable effect on precision (i.e., the difference is less than two decimal places), especially that FP is not impacted by the blocking. On the other hand, and while q-gram and RBF keep the same FN before and after blocking (preserving the same recall and F-measure scores), we notice that CLK and CLK-RBF witness some drops in a few samples. MinHash-4-100 causes sample drops of (4 for CLK and 5 for CLK-RBF) compared to MinHash-3-50 (2 for both). This contributes to a slightly enhanced recall for MinHash-4-100 and, consequently, a slightly enhanced F-measure score.

Our conclusion from this experiment on NCVR 5K dataset, is that the effect of blocking on the linkage quality is marginal. Therefore, comparing linkage quality on bigger dataset sizes, using blocking, is expected to correspond to approximate results to the no-blocking setting, while the benefit being a much reduced runtime complexity.

### 5.3.2 Testing MinHash Blocking for Quality and Scalability

#### Experimental Design

In this experiment, we use MinHash-3-50 as the better performing setting of MinHash blocking, and test its scalability on varying sizes of NCVR datasets, and the resulting linkage quality over the different encoding methods. In particular,

we test, in addition to, on 5K dataset, on 10K and 20K dataset which comprise 10,000 and 20,000 records, respectively. We set all encoding parameters to their defaults as we stated in the previous experiment 5.3.1.

## Interpretation of the Results

After running the PPRL process using MinHash-3-50 and measuring the average time used for hashing each dataset in the pair, the time used for blocking, the number of preserved true match pairs, and pairs completeness, we record the results presented in Table 11.

Dataset size	5K	10K	20K
Average records hashing time (sec)	23.41	42.77	78.92
Time used for blocking (sec)	47	86	158
Number of found matches	4,981	9,956	19,842
Pairs Completeness (%)	99.64	99.56	99.21

Table 11: Blocking measures for MinHash-3-5 on NCVR 5K, 10K and 20K

Looking at the results in Table 11, we observe that as the dataset size grows by a factor of  $\times 2$ :

- Average hashing time increases by a factor of  $\times 1.8$ .
- Blocking time increases by a factor of  $\times 1.8$  as well.
- Number of found matches increases by a factor of  $\times 1.99$ .
- And pairs completeness decreases by a step proportional to the size difference between consecutive datasets, while it starts to decrease by 0.08 between 5K and 10K.

Looking at the time spent on the linkage at the different-sized datasets using the three encoding methods and plain text, we record the results shown in Table 12. The time here is, as stated earlier, the average time of three runs of each configuration.

Method	5K	10K	20K
q-gram	11.37	27.26	56.84
RBF	5.31	13.24	27.53
CLK	5.50	14.05	28
CLK-RBF	5.71	14.27	28.37

Table 12: Linkage time (in seconds) with blocking using MinHash-3-50 on NCVR 5K, 10K and 20K

We see from Table 12 that the linkage time at 10K is around  $\times 2.5$  its value at 5K and around half its value at 20K. This means that blocking with MinHash is able to reduce linkage runtime complexity from quadratic to almost linear with the increasing dataset size from 5K to 10K and then to 20K, which makes the linkage step more scalable. However, this comes with the overhead of blocking complexity which as we have seen before also appears to be linear in time with the growing dataset size from 5K to 20K.

In order to assess the effect on linkage quality using MinHash-3-50 on different sizes of NCVR dataset, we fix the similarity thresholds that were previously reported optimal with no blocking employed on NCVR 5K, and compare with the quality at the same thresholds on larger sizes of 10K and 20K, for the different encoding methods. In case higher scores were recorded on a different threshold value for a certain method, we mention that explicitly. We compile the results in Table 13.

Dataset	Method	Threshold	TP	FP	FN	Precision	Recall	F-Meas.
NCVR 5K	q-gram	0.85	4,760	36	76	99.25%	98.43%	98.84%
	RBF	0.85	4,754	38	90	99.21%	98.14%	98.67%
	CLK	0.9	4,736	11	143	99.77%	97.07%	98.40%
	CLK-RBF	0.9	4,743	10	150	99.79%	96.93%	98.34%
NCVR 10K	q-gram	0.85	9,547	89	132	99.08%	98.64%	98.86%
	RBF	0.85	9,537	96	163	99.00%	98.32%	98.66%
	CLK	0.9	9,512	33	255	99.65%	97.39%	98.51%
	CLK-RBF	0.9	9,520	39	271	99.59%	97.23%	98.40%
NCVR 20K	q-gram	0.85	18,912	586	353	96.99%	98.17%	97.58%
	RBF	0.85	18,901	605	416	96.90%	97.85%	97.37%
	CLK	0.9	18,841	256	647	98.66%	96.68%	97.66%
	CLK-RBF	0.9	18,861	265	663	98.61%	96.60%	97.60%

Table 13: Linkage quality measures using MinHash-3-50 on NCVR 5K, 10K and 20K

We note the following deviations for the recorded optimal thresholds:

- On NCVR 10K, we see that RBF achieves a maximum F-measure at  $\tau = 0.9$  with 99.78% precision, 97.61% recall and 98.68% F-measure. At this threshold, the improvement is only 0.02% higher than the previous one, and it compromises the recall by 0.71% to increase precision by 0.78%.
- On NCVR 20K, we see that q-gram and RBF record a maximum F-measure at  $\tau = 0.9$  with 99.21% precision, 97.07% recall and 98.13% F-measure for q-gram, and 99.04% precision, 97.08% recall and 98.05% F-measure for RBF. Similar to our previous observation for NCVR 10K, this improvement compromises recall in favor of precision.

From the previous results, we notice that scaling up the dataset and applying MinHash blocking does not result in significant changes on the linkage quality measures (Precision, Recall and F-Measure) if compared at the same similarity thresholds that were reported as optimal on the initial dataset. This is because all encoding methods keep the same trends they exhibited on smaller sizes, while some marginal changes could be attributed to the nature of the new record pairs in the bigger datasets, and how similar or different they are from each others. Choosing whether one of two close thresholds, in their quality of linkage, should be recognized at optimal is application dependent. This is because the impact will be on the balance between precision and recall. As we have seen before, favoring a higher threshold will boost the precision and deteriorate the recall.

### 5.3.3 Testing HLSH Blocking for Quality and Scalability

#### Experimental Design

In this experiment, we redo the previous scenarios conducted with MinHash, but this time using Hamming LSH (HLSH) as the second type of LSH blocking we are evaluating. We run HLSH on varying sizes of NCVR dataset: 5K, 10K and 20K as we did for MinHash in Experiment 5.3.2. Moreover, we also run on larger dataset sizes of 50K, 100K and 200K to further test its scalability. We think that the outcomes from this experiment using HLSH blocking for larger dataset sizes will be projectable on MinHash as well. We also record the impact observed on linkage quality using HLSH blocking with the different encoding techniques, while excluding clear-text q-gram as HLSH blocking takes place on Bloom filters after the encoding step. In addition, we compare HLSH to MinHash-3-50 in terms of blocking performance, and the impact on linkage time and quality using the three encoding techniques. We set all encoding parameters to their defaults as we did in the previous experiment, and we choose the number of bands similar to MinHash and as suggested in [23] to  $\lambda = 50$ . The band size  $\mu$  which was a set parameter in MinHash, gets calculated for HLSH based on  $\lambda$  and the length of the Bloom filter.

## Interpretation of the Results

As explained before, this blocking method runs over Bloom filters generated by different encoding methods and, hence, the results vary by the encoding method used.

We report in Table 14 the blocking measures for HLSH on different sizes of NCVR dataset and the different encoding methods.

	Method	5K	10K	20K	50K	100K	200K
Blocking time (sec)	RBF	0.42	0.73	1.55	4.01	9.19	15.52
	CLK	1.47	2.73	5.07	9.83	21.60	33.79
	CLK-RBF	1.26	2.63	4.90	10.05	20.57	73.77
Num. of found matches	RBF	4881	9662	18292	40583	72195	119976
	CLK	4841	9699	19290	48168	96358	193048
	CLK-RBF	4850	9708	19334	48302	96629	193444
Pairs Completeness (%)	RBF	97.64	96.63	91.46	81.17	72.2	59.99
	CLK	96.84	97	96.45	96.33	96.36	96.52
	CLK-RBF	97.01	97.08	96.67	96.6	96.63	96.72

Table 14: Blocking measures for HLSH on NCVR 5K, 10K, 20K, 50K, 100K and 200K

As we can see from the results in Table 14, RBF encoding results in the shortest time for blocking. We also see that CLK requires a bit more time than CLK-RBF at smaller sizes, but as the dataset size gets bigger, the difference in blocking time becomes smaller between the two. At size of 200K, blocking over CLK takes less than half the time taken by blocking over CLK-RBF. For RBF at that size (but also at 50K and 100K), it takes less than half the time of CLK. Beside that, We notice that the three methods double their blocking time when the dataset size doubles from 50K to 100K. However, doubling from 100K to 200K increases the blocking time to approximately  $\times 1.5$  its previous value for RBF and CLK, whereas for CLK-RBF, blocking time scales up by  $\times 3.5$ .

In terms of the number of true matches preserved by the blocking technique, we see that at the smaller dataset size of 5K, HLSH over RBF Bloom filters preserves 40 and 30 more true matches than CLK and CLK-RBF, respectively. However, as the dataset size gets bigger by a factor of  $\times 2$ , CLK and CLK-RBF outperform RBF. The gap grows as the dataset size multiplies by 2. At 200K we find a pair completeness value of only 59.99% for RBF, compared to 96.52% for CLK and 96.72% for CLK-RBF. The source of this deterioration is that HLSH removes many more candidate pairs, including many true matches, through blocking over RBF than what it does over the other two encoding methods. By looking at the logs of the three methods, we see that HLSH over RBF is removing many blocks

from consideration when they exceed the size of 100 elements per block. For example, at dataset size of 20K, 660 and 672 blocks were removed from the first and second parties conducting the linkage, respectively. For CLK, however, only one block exceeding 100 elements was removed, and for CLK-RBF, 11 and 9 blocks were removed from the first and second parties, respectively. This also explains the difference in the HLSH blocking time over the three methods, and leads to a conclusion that HLSH over RBF encoding results in more similar bit patterns among its Bloom filter segments compared to CLK and CLK-RBF, which results in many large blocks being generated and, hence, removed along with the true matches they may contain.

Comparing the blocking measures for HLSH with the ones recorded for MinHash-3-50 in Table 11, we see that HLSH over CLK Bloom filters (which takes longer than the other two methods) concludes in 1/32 the time needed for MinHash at different dataset sizes. However, the blocking quality for MinHash is higher than for HLSH with more than 99% pairs completeness compared to around 97% for HLSH over the best performing encoding. Talking from an implementation perspective, HLSH is faster than MinHash as it does not do any further hashing over Bloom filters, but rather segments each Bloom filter into bands and block records according to shared bands among them.

We present in Table 15 the time spent on linkage using the different encoding methods based on blocks generated by HLSH.

	5K	10K	20K	50K	100K	200K
RBF	2.06	3.59	5.21	6.60	13.00	17.61
CLK	0.57	2.04	7.36	20.73	88.91	393.82
CLK-RBF	0.80	2.35	8.47	21.86	91.60	447.37

Table 15: Linkage time (in seconds) with blocking using HLSH on NCVR 5K, 10K, 20K, 50K, 100K and 200K

We see that at the initial dataset size of 5K, linkage over CLK encoding concludes 0.13 seconds earlier than CLK-RBF and is around 1/4 the time taken by linkage over RBF. As the dataset size doubles, we see that for RBF, the linkage time demonstrates mostly a slow increase (i.e., scales up by a varying factor in the range [1.2, 2.2]), whereas linkage time over CLK and CLK-RBF increases by a factor of  $\times 4$  with the dataset size. This can be explained again by the reduction ratio achieved by HLSH over each of the three methods. In other words, for smaller dataset sizes of 5K and 10K, linkage over RBF takes longer time than over CLK and CLK-RBF because it happens over larger but fewer blocks generated by



HLSH blocking. However, the bigger the dataset size gets, the smaller the relative number of generated blocks becomes for CLK and CLK-RBF. For example, for CLK at size of 5K, HLSH generates 200K out of 250K possible blocks (calculated as  $5,000 \text{ records} \times 50 \text{ bands}$ ), whereas at size of 10K, it generates 300K out of 500K possible blocks, and at size of 20K, it generates 500K out of 1 million possible blocks. At dataset size of 20K, the linkage reaches a point where fewer larger blocks over RBF Bloom filters take less time than the more smaller blocks over CLK and CLK-RBF Bloom filters.

We present in Table 16 a comparison of linkage time based on blocks generated by HLSH and MinHash-3-50 on NCVR 5K, 10K and 20K.

	5K		10K		20K	
	MinHash	HLSH	MinHash	HLSH	MinHash	HLSH
RBF	5.31	2.06	13.24	3.59	27.53	5.21
CLK	5.50	0.57	14.05	2.04	28	7.36
CLK-RBF	5.71	0.80	14.27	2.35	28.37	8.47

Table 16: Linkage time (in seconds) with HLSH-50 blocking and MinHash-3-50 blocking on NCVR 5K, 10K and 20K

In comparison to MinHash, and as we can see from Table 16: At dataset size of 5K, linkage using HLSH over RBF takes less than half the time taken for linkage using MinHash. Also, the linkage time using HLSH over CLK and CLK-RBF equals  $1/10$  and  $1/7$  the linkage time using MinHash, respectively. At dataset size of 10K, however, these factors become  $1/3$  for RBF, and  $1/6$  for CLK and CLK-RBF, respectively. Lastly, at dataset size of 20K, the factors become  $1/5$  for RBF and  $1/4$  and  $1/3$  for CLK and CLK-RBF, respectively. By looking at these numbers, we conclude that the bigger the dataset size, the smaller the difference in linkage time between HLSH over CLK or CLK-RBF and MinHash. This difference gets bigger for HLSH over RBF due to removal of big blocks. Nevertheless, the difference between the two blocking methods remains big in terms of linkage time, where, in overall, linkage over HLSH blocks takes between  $1/3$  to  $1/4$  the time taken over MinHash blocks. After inspecting the logs of the runs we found that MinHash generates more uniform blocks with an average size of 3 records per block at size 5K, that increases to 3.6 and 4.4 at sizes 10K and 20K, respectively. The number of generated blocks is 72,636, 108,246, 159,781 for the three sizes, respectively. If we multiply the square of the average block size with the number of blocks at a certain dataset size, we get a larger number of estimated comparisons for MinHash than the ones calculated for HLSH for the different encoding methods. At larger sizes, HLSH generates many small blocks but not necessarily

all have a correspondence in the other party. This interesting finding justifies why MinHash has a better blocking quality than HLSH, as it does more comparisons across more uniform blocks.

In order to assess the effect on linkage quality using HLSH on different sizes of NCVR dataset, we fix the similarity thresholds that were previously reported optimal with no blocking on NCVR 5K, and compare with the quality at the same thresholds on NCVR 5K and larger sizes of up to 200K, for the different encoding methods, when HLSH blocking is employed. In case higher scores were recorded at a different threshold value for a certain method, we mention that explicitly. We present the results in Table 17.

Dataset	Method	$\tau$	TP	FP	FN	Precision	Recall	F-Meas.
NCVR 5K	RBF	0.85	4,746	38	79	99.21%	98.36%	98.78%
	CLK	0.9	4,736	11	92	99.77%	98.09%	98.92%
	CLK-RBF	0.9	4,743	10	91	99.79%	98.12%	98.95%
NCVR 10K	RBF	0.85	9,452	91	127	99.05%	98.67%	98.86%
	CLK	0.9	9,512	33	162	99.65%	98.33%	98.99%
	CLK-RBF	0.9	9,520	39	161	99.59%	98.34%	98.96%
NCVR 20K	RBF	0.85	17,915	486	249	97.36%	98.63%	97.99%
	CLK	0.9	18,841	256	400	98.66%	97.92%	98.29%
	CLK-RBF	0.9	18,861	265	418	98.61%	97.83%	98.22%
NCVR 50K	RBF	0.85	40,001	66	404	99.84%	99.00%	99.42%
	CLK	0.9	47,047	40	1,035	99.92%	97.85%	98.87%
	CLK-RBF	0.9	47,184	30	1,019	99.94%	97.89%	98.90%
NCVR 100K	RBF	0.85	71,356	119	609	99.83%	99.15%	99.49%
	CLK	0.9	94,076	94	2,092	99.90%	97.82%	98.85%
	CLK-RBF	0.9	94,349	84	2,068	99.91%	97.86%	98.87%
NCVR 200K	RBF	0.85	118,860	220	829	99.82%	99.31%	99.56%
	CLK	0.9	188,614	208	4,092	99.89%	97.88%	98.87%
	CLK-RBF	0.9	189,186	197	3,869	99.90%	98.00%	98.94%

Table 17: Linkage quality measures using HLSH on NCVR 5K, 10K, 20K, 50K, 100K and 200K

We see from Table 17, and in comparison to the results in Table 13, that scaling up the dataset to  $\times 2$  and  $\times 4$ , while employing HLSH blocking:

- Results in a slight drop in precision for bigger sizes than 5K (where it improves) compared to MinHash.
- Preserves the same TP and FP record pairs for CLK and CLK-RBF that were preserved in MinHash.

- Loses some TP and FP record pairs for RBF, which somehow results in an increase in the quality measures of the classification, but an actual decrease in the blocking quality.

Based on the previous observations, we can conclude that HLSH gives better results than MinHash in terms of blocking quality for CLK and CLK-RBF which reflects on the linkage quality. On the other hand, MinHash gives better results for RBF where HLSH fails in the blocking quality measures due to significant losses in the TP record pairs on bigger dataset sizes.

As a general remark, we find that scaling the dataset size, while using the suitable blocking technique for each encoding method gives comparable linkage quality results to the ones made with no blocking over the initial dataset of size 5K. Specific improvements or deteriorations recorded at any point are attributed to the blocking technique used. This yields to our findings on the initial dataset being roughly generalizable to bigger dataset sizes. Similar conclusions were made by [65] on very big datasets of 7 and 20 million records.

When we scale the dataset to 50K, 100K and 200K, consecutively, we see the scores marginally improve keeping the same trends we have seen on the smaller datasets. We find that at the bigger sizes of 50K, 100K, and 200K, CLK-RBF gives slightly better precision, recall and F-measure, as it preserves more TP record pairs than CLK while removing more of the FP and FN ones. Eventually this means a better pairs completeness measure for CLK-RBF. However, as we have seen before, this encoding method requires more blocking time than CLK. While the difference in the blocking time is quite small (within a second) for dataset size of 50K and 100K, it becomes more than the double at 200K. The linkage time gets also bigger than in CLK with around 50 seconds at dataset size of 200K, in comparison to 1 second for 50K and 3 seconds for 100K. We conclude, generally, that with blocking employed, higher linkage quality is accompanied with higher blocking and linkage times. This is justified before for MinHash in comparison to HLSH, and is analogous to what is observed here for HLSH over CLK-RBF in comparison to HLH over CLK.

### 5.3.4 Testing HLSH Blocking on Big Data

#### Experimental Design

In this experiment, we redo previous scenarios conducted on smaller dataset sizes using HLSH, and report the results in a summarized way. We aim from this to test the generalizability of the results to big datasets of millions of records. We conduct the runs of this experiment on a server machine and, hence, the time recorded is not comparable to the previous experiments. However, we consider the baseline here as the experiment run on the first big dataset (i.e., of 400,000

records). The other datasets that we test on are of 800,000, 1,600,000 and 3,200,000 records, which we denote as NCVR 400K, 800K, 1.6M and 3.2M, respectively. For this experiment, we exclude CLK-RBF due to implementation issues with the *clckhash* library underneath, i.e., it crashes on large dataset sizes starting from NCVR 400K. For the rest of the code, we fixed one scalability bottleneck that was causing the original code to crash on larger datasets (starting from NCVR 800K) in the *conduct\_linkage()* function, where we replaced a Python’s dictionary used to record the already compared record pairs with a *bitarray* and a hashing scheme to achieve the memorization step efficiently.

## Interpretation of the Results

We report in Table 18 the blocking measures for HLSH over the increasing volumes of data from 400,000 to 3,200,000 records of NCVR dataset. We also include NCVR 200K for reference. To help us explain some of the newly observed trends in this experiment, we include in our reported results the number of candidate pairs, which was not included in the previous results. We also append the linkage time to the blocking measures table for a more concise presentation.

	Method	200K	400K	800K	1.6M	3.2M
Blocking time (sec)	RBF	16	39	75	173	359
	CLK	34	70	133	249	568
Num. of found matches	RBF	119,976	168,284	220,857	294,447	407,160
	CLK	193,048	386,769	771,253	1,533,566	2,749,104
Num. of candidate pairs	RBF	9,465,436	11,098,118	12,216,994	15,666,933	22,122,435
	CLK	134,934,647	512,110,646	1,432,583,629	2,193,764,474	1,234,245,721
Pairs Completeness (%)	RBF	60	42.07	27.6	18.4	12.72
	CLK	97	96.69	96.41	95.85	85.9
Linkage time (sec)	RBF	17.61	22.79	57.08	76.27	113.9
	CLK	393.82	1,328.85	6,535.97	10,493.73	6,176.97

Table 18: Blocking measures for HLSH on NCVR 200K, 400K, 800K, 1.6M and 3.2M

As we can see from results in Table 18, HLSH blocking over RBF takes around half the time needed for blocking over CLK at dataset sizes of 200K and 400K. The difference between the two shrinks as the dataset scales up, to reach its smallest at 1.6M with HLSH blocking time over RBF equals  $\times 0.7$  of its value over CLK . This goes along with our conclusion from the previous experiment 5.3.3.

In terms of the number of true matches preserved by the blocking technique, we see that while HLSH over CLK Bloom filters is preserving between 97% to 96% between 200K and 1.6M datasets. Blocking over RBF, on the other hand, magnifies its losses as the dataset size gets bigger, to only preserve 18% and 13% at sizes 1.6M and 3.2M, respectively. This leads us to conclude that HLSH over

RBF is not scalable to larger datasets as it leads to drastic drops in the blocking quality over them. As we recorded in Experiment 5.3.3, linkage quality for HLSH over RBF starts to deteriorate after size 10K.

By looking at the time spent on the linkage using the two encoding methods over HLSH-generated blocks, we find that RBF increases by a factor between [1.3, 2.5] with the dataset size. This goes along with our observation from Experiment 5.3.3. For CLK we see that on NCVR 400K and 800K, the linkage time increases by a factor of  $\times 3.4$  and  $\times 4.9$ , respectively, and has a roughly similar trend to the previous Experiment 5.3.3 on less big dataset sizes. Starting from size 1.6M, it only increases by a factor of  $\times 1.6$ . Interestingly, we also find at 3.2M that the linkage time is less than its value at both 1.6M and 800K, and is equal to only 0.59 to its value at 1.6M. Here is the point where we decided to look at the number of candidate pairs after including it in the table. As can be seen from the results, HLSH over CLK at 1.6M keeps  $\times 1.6$  the amount of candidate record pairs kept at 800K, which is proportional to the increase in linkage time between the two sizes. On the other hand, at dataset size of 3.2M, it only keeps around 1,234 million candidate record pairs, which is 0.56 and 0.86 of what it preserves at 1.6M and 800K, respectively. This leads us to conclude that the HLSH blocking with number of bands  $\lambda = 50$  is only scalable up to size 1.6M (among our size options) with satisfactory results. The decreased blocking quality at 3.2M can be attributed to the block removal step at that size as explained previously for MinHash. This means that a larger number of blocks exceed the size of 100 elements at 3.2M and, hence, get removed with some of removed ones containing record pairs that do not occur elsewhere. This effect of block removal results in a decaying factor of preserved candidate record pairs with the growing dataset size.

In order to assess the effect on the linkage quality using HLSH over big sizes of NCVR dataset, we report in Table 19 the optimal thresholds along with their corresponding linkage quality scores. We also include here results on NCVR 200K for reference.

Dataset	Method	Opt. $\tau$	TP	FP	FN	Precision	Recall	F-Meas.
NCVR 200K	RBF	0.85	118,860	220	829	99.82%	99.31%	99.56%
	CLK	0.9	188,614	208	4,092	99.89%	97.88%	98.87%
NCVR 400K	RBF	0.85	167,104	752	855	99.55%	99.49%	99.52%
	CLK	0.9	378,907	1,650	7,224	99.57%	98.13%	98.84%
NCVR 800K	RBF	0.85	219,514	1,223	973	99.45%	99.56%	99.50%
	CLK	0.9	757,489	4,183	12,883	99.45%	98.33%	98.89%
NCVR 1.6M	RBF	0.85	292,884	2,091	1,140	99.29%	99.61%	99.45%
	CLK	0.9	1,509,962	12,019	16,242	99.21%	98.94%	99.07%
NCVR 3.2M	RBF	0.85	405,359	3,254	1,290	99.20%	99.68%	99.44%
	CLK	0.9	2,719,852	18,447	9,124	99.33%	99.67%	99.50%

Table 19: Linkage quality measures using HLSH on NCVR 200K, 400K, 800K, 1.6M and 3.2M

As we can see from Table 19, linkage quality over RBF shows high scores, even higher than over CLK, despite the very low *pairs completeness* observed for it in Table 18. As we previously hinted, this linkage quality is based on the samples returned after blocking and represent a subset of the original dataset. e.g., at size of 1.6M, the returned samples only represent 18.4% from the original dataset. We can make an intuitive conclusion here, that linkage quality, when blocking is employed, should always be assessed in conjunction with the blocking quality, to avoid misleading high scores for the linkage over low quality blocking outcomes, as in HLSH over RBF case. For CLK, we see that the optimal threshold remains at 0.9 for all sizes, with recall slightly improved when the size increases by  $\times 2$  and precision slightly attenuated. On the other hand, we see that F-measure slightly improves from size 400K to 1.6M, while achieving a relatively bigger improve by 0.43% at size 3.2M thanks to 0.73% improve on recall and 0.12% on precision. This improved linkage quality is, however, analogous to the high linkage quality of HLSH over RBF that we discussed above. At size of 3.2M the blocking quality for HLSH over CLK deteriorates by around 10% on the pairs completeness measure.

The observations we made in this experiment between 400K and 1.6M dataset sizes confirm our preliminary conclusion from Experiment 5.3.3 on the generalizability of our findings on linkage quality from dataset NCVR 5K to big datasets with millions of records. While we could only test with CLK at these big sizes, the previous results at NCVR 10K, 20K, 50K, 100K and 200K support this conclusion. For dataset size of 3.2M, we think that the same holds as for other dataset sizes. However, this size needs a different parameter tuning for the blocking technique due to the big number of candidate record pairs to be handled.

## 5.4 Discussion

Based on the different experiments that we conducted in this study, we came up with many interesting findings and conclusions that we will discuss here and build upon to give some recommendations for future studies.

Starting with the basic case of applying PPRL on a modest dataset of size 5,000 records, we found that RBF encoding method results in more representative Bloom filters, that contribute to a better overall linkage quality than CLK and CLK-RBF, and make PPRL over RBF approximate to clear-text record linkage. This was later confirmed on larger dataset sizes as well.

By raising the number of hash functions,  $k$ , we noticed divergent trends between RBF on one side and CLK and CLK-RBF on the other. While RBF witnessed improved scores with the increase of  $k$  up to 15, a deteriorated quality was observed for CLK and CLK-RBF due to the higher collision rate. The optimal  $k$  was recorded at 10 for the latter two methods and at 15 for RBF. In general, we found that the change of  $k$  has less impact on quality if tokens were not padded, and that RBF has more overall resilience to this change compared to CLK and CLK-RBF.

In a connected observation, we found that the calculated optimal value for  $k$ , based on the number of hashed elements, does not necessarily give the best scores as the linkage quality is dependent on the method employed. In general, our results showed that  $k = 10$  gave better overall scores than other values including ‘opt’.

Testing the effects of padding and q-gram size led us to find that padded tokens give better results, in general, along with  $k = 10$ . Increasing the size of q-grams, on the other hand, magnified the variability between the different encoding methods, as RBF responded positively to an increase from  $q = 2$  to  $q = 3$  in contrast to the other two methods. However, CLK and CLK-RBF were observed to be more tolerant to spelling errors with 3-grams at higher similarity thresholds. In general, the increase from  $q = 2$  to  $q = 3$  did not contribute to an observable linkage quality, which led us to consider  $q = 2$  as a more stable value for the size of q-grams. Based on our previous remarks, we decided on the best performing configuration of  $k = 10$ ,  $q = 2$ , with padding, as our default setting, which helped us to assess the different methods under uniform conditions. Comparing our results on NCVR dataset with the ones recorded on FEBRL, we observed similar trends for the three methods across the different values of  $k$  and similarity thresholds. This confirmed our initial findings and raised our confidence about its generalizability to data from a different domain.

In another context, testing with different subsets of QID attributes led to the set of personal attributes including last, first and middle names, gender and birth year to be the optimal attributes set. PPRL using the three encoding methods over this optimal set demonstrated the most accurate classification results and the most converged trends and values among the three methods. In a connected



observation, we found that the last name attribute is as important as the first name, in a divergence from arguments made in other studies about its volatility due to marriages. We found, in general, that weighting can be introduced as a flexible alternative to eliminating volatile attributes by attenuating or emphasizing attributes of choice. This was supported by our observation on CLK-RBF, which was able to give approximate results, with two weighting schemes and all attributes included, to the results made on the non-weighted optimal attributes subset.

When we introduced blocking to the PPRL process, we found that its effect on linkage quality was marginal for most settings, which supported the generalizability of our findings on NCVR 5K to larger dataset sizes. This was confirmed later by actual tests on big datasets with millions of records. Comparing between MinHash-3-50 and MinHash-4-100 blocking, we found that the first was faster and gave better performance which qualified it for further tests and comparisons with HLSH blocking. In such comparisons, HLSH appeared to be significantly faster than MinHash but with a relatively lower blocking quality. Linkage time was also lower over HLSH blocks by more than the third compared to over MinHash blocks. This was due to a large number of uniform blocks generated with MinHash, which resulted in bigger number of candidate record pairs compared to HLSH. However, HLSH over RBF led to big losses on the true matches as the dataset size increased, which rendered this setting of HLSH impractical, and non-scalable to larger sizes than 10K. In general, we found that HLSH achieved different reductions over the three encoding methods, which impacted the linkage time over the resulting blocks. At larger sizes, HLSH seemed to generate many small blocks that do not all have a correspondence in the other party which justified the better quality observed for MinHash over it. In overall, we concluded that HLSH works well over CLK and CLK-RBF, with relatively high linkage quality, while MinHash is better suited for linkage quality if RBF encoding method was used. In the same context, we found at larger dataset sizes of up to 200K, that HLSH over CLK-RBF gave slightly better scores on the quality measures, while requiring more time than HLSH over CLK. Scaling up to 3.2 million records, confirmed conclusions we made early with 5,000 records. However, it showed that HLSH was not scalable to that size with its used parameter value of 50 for the number of bands.

In conclusion, we made a general remark that linkage quality, when blocking is employed, should always be assessed in conjunction with the blocking quality, to avoid misleading high scores for the linkage over low quality blocking outcomes.



## 6 Related Work

In this chapter, we survey existing evaluation studies and software tools in the literature, that are connected to our work. We discuss for each of them the methodology used, the limitations, in addition to the differentiation in our work to it. We split this chapter into three sections: two sections based on the focus of the evaluation and one section dedicated for the software PPRL projects that could be leveraged in evaluations.

### 6.1 Evaluation Work on Linkage Quality:

Randall et al. [65] applied privacy preserving linkage using the attribute Bloom filters (ABFs) for approximate string comparison to large, population-level data collections. The datasets included ten years of West Australian Hospital Admissions data (around 7 million records) and ten years of the New South Wales Admitted Patient Data Collection (around 20 million records). The linkage strategy included blocking using *Soundex* of surname with first initial, and date of birth. Comparison was conducted using a mix of approximate and exact matches for different attributes. Linkage quality was measured at different thresholds with highest result reported. The results showed that probabilistic linkage using Bloom filter masked fields is equally effective as probabilistic linkage using unmasked fields. Using ABFs for encoding, however, has shown to be vulnerable to frequency attacks by recent work [32, 23]. In our evaluation study, we employ more advanced Bloom filter based methods that show robustness to most privacy attacks and provide better security than ABFs.

Schnell et al. [74] compared the performance of CLKs with the more traditional ALCs in terms of precision and recall using real-world data from a large regional breast cancer screening program. The database contained around 200K records. Different encoding variants and matching strategies were used in the experiments and additional identifiers were included to compare their effect. The results showed better recall but lower precision for CLK than best-performing ALCs. However, the impact of increased false positive was not considered a limiting factor for the application of CLKs thanks to their flexibility for fine-tuning. Moreover, CLKs were expected to exceed ALCs when more stable identifiers are involved in the

linkage. According to the authors, after fine-tuning parameters and identifier sets, PPRL linkage quality over CLKs preserves the precision and becomes comparable to clear-text linkage. This work, in general, limits to evaluating linkage quality between one chosen Bloom filter based encoding method, i.e., CLK, and the traditional ALCs. While CLK still represents one of the state-of-the-art encoding methods for PPRL, we show in our work that for some settings, CLK can be outperformed in terms of linkage quality by RBF or the hybrid CLK-RBF.

In a recent and connected study on Bloom filters encoding [32] Franke et al. reviewed, categorized and comprehensively evaluated the Bloom filter variants and hardening techniques in terms of both linkage quality and privacy following their proposition of three privacy measures that allow assessing the privacy properties of Bloom filter encodings. Two real-world datasets obtained from the North Carolina voter registration database (NCVR) and the Ohio voter files were used in the experiments. Two snapshots of each dataset were selected at different points in time, which allowed for errors and inconsistencies between the snapshots due to the time difference. Bloom filters with size 1024 were used, and Hamming LSH was employed as a blocking technique. The pair-wise similarity was assessed using Jaccard coefficient and the simple threshold classification model. Different configurations were tested, including the use of bigrams and trigrams, with or without padding, at different values of hash functions and different similarity thresholds. Linkage quality measures of precision, recall and F-measure were recorded. For their evaluation of linkage quality, the authors selected the best configuration using trigrams and padding with number of hash functions equal to 10 as a baseline for their privacy experiments. The results showed that multiple hardening techniques drastically reduce linkage quality and therefore are not applicable in real-world scenarios. On the other hand, two particular techniques, namely salting and xor-folding, seemed to maintain high linkage quality while drastically reducing frequency information. The authors recommended a careful selection of Bloom filter parameters in combination with these hardening techniques to highly lower the probability of frequency-based cryptanalysis attacks. In this work, however, the authors did not make it clear that they were testing on CLK encoding, despite having described the difference between it and other methods. Moreover, and while their work provides a very good evaluation on tuning Bloom filters for linkage quality and privacy, the effect of the blocking technique on the linkage quality, which represents an important part of our evaluation work, was overlooked in theirs.

Heidt et al. [36] tested with weighted attributes over a pair of synthetic datasets, each consisted of 5000 records generated by the FEBRL tool [20]. The aim was

to find the best configuration for their proposed algorithm for federated record linkage over multiple parties with a trusted third party that they integrated into an end-to-end pseudonymization framework for medical data sharing. Record-level Bloom filters (RBFs) were used in the experiments to encode nine already agreed-on QIDs: first name, last name, gender, date of birth, city, zip code, street name, country and insurance number. All of them being formatted as strings that were split into bigrams. Number of hash functions was set to 15 with a length of 500 bits for the generated Bloom filters. Different combinations of weights were evaluated using a grid search, where the weighting of an attribute specified the sampling of its attribute Bloom filter (ABF) in the record level one (RBF). Different combinations of hashing methods were also compared in the double-hashing scheme. Results showed the best linkage results to have been achieved using a combination of SHA2 and SHA3 with 32-bit HMACs in the double hashing scheme at threshold of 0.8 and with optimal weight matrix was 0.1 for (first name, last name, birthday, zip code and insurance), 0.2 for (gender and country), and 0.05 for city. One limitation of this work is the use of a single synthetic dataset to draw the final conclusions, which were not proven to be generalizable. In addition, it uses an uncommon length of 500 bits for Bloom filters, while comparing different hashing functions for an already vulnerable hashing scheme, namely the double-hashing. In our work, we employ the more secure random hashing, use more than one dataset, and set the Bloom filter size to the recommended and commonly used value of 1024.

## 6.2 Evaluation Work on Linkage Quality and Scalability:

Brown et al. [13] evaluated the performance of CLKs using multibit trees against gold-standard derived from clear-text probabilistic record linkage on large-scale data that is a subset of the Australian hospital admission data. The final dataset was of 20 million records from which datasets of 5, 10 and 15 million records were randomly sampled. The accuracy and efficiency of multibit tree over CLKs with a leaf limit value of one was tested in the experiments where linkage quality measures and the execution time was recorded. Results showed that the clear-text linkage resulted in marginally higher precision and recall than the fine-tuned CLKs and concluded in less computing time. The authors argue that having the privacy aspect of PPRL comes with the price of small losses in the linkage quality but also a large increase in computational time on large datasets which seems to be acceptable in applied settings. The results in general show that the use of multibit trees for indexing/blocking of CLK data has great potential. On another observation,

the authors found that including more stable identifiers without missing data in the CLKs will usually increase the discriminative power, leading to increasing the precision without sacrificing recall. While this work provides an evaluation of linkage quality and scalability using CLK with multibit trees, it does not address each of the two methods individually, but rather together as one combined method. This is different from the way we conduct our evaluation as we try to make a clear separation between the two phases of encoding and blocking, study the quality and performance of each method alone, and then in conjunction with other methods, which we believe to provide a more holistic approach of the evaluation.

Christen et al. [23] conducted an empirical evaluation of selected Bloom filter based encoding and hardening techniques with regard to their quality, scalability and privacy. Datasets were based on the North Carolina Voter Registration database (NCVR) where records from two of its snapshots were extracted between April 2018 and October 2019. The QID fields included in the linkage were: first name, middle name, surname, birth year, gender, street, city and zip code as they represent commonly used attributes for record linkage project. Three different variations of pairs of dataset were generated (CLN-500K, CLN-100K and DRT-100K) where CLN-500K and CLN-100K included up to 2 different QIDs for the same voter across the two snapshots, with 500,000 and 100,000 records respectively. DRT-100K contained 100,000 records, where each voter had between one and three different QIDs across snapshots. CLN-100K and DRT-100K were used in the linkage experiments while CLN-100K and CLN-500K were used for the scalability experiments. Experiments setting included a Bloom filter length of 1000 bits, and a varying number of hash functions for q-gram encoding. All QIDS were assumed to be strings that were converted into bigrams and hashed using double and random hashing. For encoding methods, CLKs and weighted RBFs were used and evaluated for linkage quality using a dice coefficient similarly function in comparison to the clear-text linkage. The authors also implemented a blocking approach using MinHash LSH with number of bands equals to 50 and a band size of 3. Block size was limited to 100 records to prevent very large blocks from causing much longer runtimes. For scalability, 1, 2 or 4 attributes were encoded with increasing number of hash functions for which the runtime was recorded. Results showed no difference in hashing methods on the linkage quality, whereby random hashing was recommended as a more secure alternative to double hashing. In encoding, CLK showed similar linkage quality compared to the clear-text q-grams with a small decrease in linkage quality as the number of hash functions increases. The best linkage quality for CLK was achieved with the smallest number of hash functions  $k = 10$ . On the other hand, RBF encoding outperformed CLK encoding for the different numbers of hash functions,  $k$ , where the largest number of hash

functions used ( $k = 30$ ) achieved the best linkage results on both the clean and dirty data sets. This was attributed to the use of weighting scheme for the RBF. Among the hardening techniques applied, XOR folding and Rule 90 performed best, even resulting in increased linkage quality when CLK encoded Bloom filters were used. In the scalability experiments, runtimes for different encoding methods were increased with the number of hash functions used. RBF required much longer runtime compared to ABF and CLK. This work however, did not address the scalability of the blocking technique used nor did it compare blocking techniques to each others. While the work studied the scalability of the encoding methods to larger data, the limitation remained in doing this for selected attributes instead of the whole set using during the linkage. We think that the authors for some limitation in their implementation decided not to include ABF in the linkage quality experiments but did include it for the scalability ones, which made somehow a lack of consistency. We built on this study to cover different techniques for linkage quality and to study the scalability in conjunction with quality for both the encoding and blocking techniques.

Vatsalan et al. [81] presented a theoretical grounding for an evaluation framework for PPRL. The framework provides an overall numerical score that enables the assessment of PPRL solutions against scalability, linkage quality and privacy. The authors used the proposed framework to experimentally evaluate six private blocking including HLSH and four private comparison and classification techniques using real-world datasets. The first dataset represented an Australian telephone directory that contained around 7 million record, from which four commonly used attributes for record linkage were extracted: given name, surname, suburb name, and postcode. Datasets of different sizes were generated by sampling 0.1%, 1%, 10% and 100% of the records in the full database. Random noise was introduced to the dataset to generate a version of dirty data for each one. The second dataset was the NCVR database previously described, from which the four attributes of (first name, surname, city and zipcode) were also extracted. The efficiency of blocking was measured by reduction ration (RR), whereas the effectiveness was measured by pairs completeness (PC) on both datasets. Results showed that HLSH performs better by achieving high values for both RR and PC. It was also shown that HLSH approach generates overlapping blocks of smaller sizes and the variance between block sizes is comparatively very high. For the experiments on private comparison and classification techniques included CLK, RBF, and CLK-RBF. CLK was used with number of hash functions  $k = 30$ , length of Bloom filter  $l = 1000$  and  $q = 2$  (bigrams). RBF was used in the weighted setting using the same number of hash functions as CLK whereas CLK-RBF was used with similar Bloom filter length as CLK and a different number of hash functions reflecting different attribute weights.

Results showed that RBF encoding requires more blocking iterations to converge but in the end achieves a higher recall compared to CLK which completes in a smaller number of iterations. The hybrid CLK-RBF, however, achieves a higher recall in a smaller number of iterations. The authors concluded that in general, CLK-RBF encoding method used with blocking outperforms the other two encoding solutions by achieving higher linkage quality, and better privacy in terms of bit distribution and pruning of non-matches. Also in terms of scalability, the CLK-RBF was faster than CLK and RBF because it required a smaller number of iterations to converge compared to the other two encoding methods. This study while presenting a theoretical base for conducting similar comparative evaluations in the future, it, however, models this evaluation from the perspective of a privacy attack which represents the core part of the evaluation. It also does not address the implementation aspect of an evaluation framework including how to integrate future techniques with it and how to use its API, neither does it offer a public access to the code used in the study despite expressing the willingness to offer it. We compare similar encoding methods to this study but from different libraries, and we provide a concrete open-source evaluation framework that could be easily used, extended and modified by other researchers.

## 6.3 PPRL Research Software Projects

We list below some of the research projects that have developed free available software tools to enable building of PPRL pipelines using a variety of implemented algorithms and, in some cases, evaluating them [23]:

PRIMAT [31] is a software toolbox for PPRL techniques developed by the database group at the University of Leipzig. The toolbox includes Java implementations of previously developed methods for fast and scalable PPRL based on the use of blocking. PRIMAT follows the assumption of a trusted linkage unit and offers different linkage modes, protocols and components that support the creation and execution of tailored PPRL flows. Its implemented techniques are grouped in two packages, one for the data owners and another for the linkage unit. The functions provided include: data generation and corruption, data cleaning, data encoding using Bloom filter encoding and hardening techniques, HSLH blocking and metric space based indexing and filtering techniques, in addition to post-processing methods. PRIMAT also has an evaluation package that contains class definitions for linkage quality measures. However, it is not clear how to leverage this package nor how to build a PPRL pipeline for the evaluation. The code also lacks standardization as it compiles techniques developed by different people

at different periods of time, and a proper documentation is still largely missing. We try in ours to address most such shortcomings and we provide our solution in Python.

The 'PPRL' R package<sup>1</sup> is a toolbox developed as a software package in R<sup>2</sup> statistical programming language and implemented in both R and c++ by the University of Duisburg-Essen. The package combines the functionalities of the earlier Merge ToolBox developed by the same university with both exact and probabilistic PPRL linkage techniques. The package offers a wide range of encoding methods and their variants including statistical and encrypted linkage keys, in addition to Bloom filter based encoding methods. The latter includes CLK, RBF and different hardened versions of them. Blocking was implemented in a basic blocking function that is also provided in the package. However, this package lacks any utility for evaluation of the encoding methods it offers, and also lacks efficient blocking to allow experiments on scalability. Whereas our solution is focused on the evaluation part on both encoding and blocking.

The Anonlink<sup>3</sup> project [26] is a PPRL project that includes an efficient implementation of the CLK encoding method in the Python CLKhash<sup>4</sup> library, in addition to a hashing-based external blocking method that supports multi-party linkage to speed up the comparison of Bloom filter sets. The Anonlink project is developed by the Confidential Computing team at Data61 unit of the Australian national science agency. It is written in C++ with a Python interface and supports secret keys through the HMAC<sup>5</sup> keyed-hash function for the secure encoding of sensitive QID values. This project, similar to the previous one, also lacks any evaluation utility. We use the CLK implementation provided in this project for the CLK-RBF implementation in our work.

Lastly, we mention the experimental evaluation code provided in [23] which we used as a starting point for developing our comparative evaluation framework. The code includes modules for: Three Bloom filter encoding techniques (ABF, RBF and CLK), datasets generation, linkage evaluation workflow, and privacy attack methods. The code was implemented in Python 2.7 and is callable by command line where necessary parameters need to be passed to each file. Our contribution to this code is already presented in Chapter 5.

---

<sup>1</sup><https://cran.r-project.org/web/packages/PPRL/>

<sup>2</sup><https://www.r-project.org/>

<sup>3</sup><https://github.com/data61/anonlink>

<sup>4</sup><http://clkhash.readthedocs.io/>

<sup>5</sup><https://datatracker.ietf.org/doc/html/rfc2104>

In our comparative evaluation, we based our design on ideas drawn from most of the aforementioned work [65, 81, 74, 13, 32, 23], while for our implementation, we combined techniques developed by [23] and [26]. We differentiate from the previous work in providing a well-defined prototype of an evaluation framework, with a clear interface to integrate new techniques which none of the previous work has offered. We also present a comparative empirical evaluation that spans to implementations from different libraries and codebases, and provides the first extensive evaluation of the Bloom filter encoding techniques CLK, RBF and CLK-RBF against the clear-text baseline. Instead of studying the scalability of these methods in terms of time taken for encoding a set of attributes as in [23], we compare both their linkage quality and linkage runtime beside the blocking runtime using different blocking approaches and varying sizes of datasets. This leads to our scalability evaluation covering both the encoding methods and the blocking methods, where we provide detailed analysis of the two categories. We think that in terms of compared elements and looked at measures for both quality and scalability, we are providing a holistic evaluation study that showcase the viability of our proposed evaluation framework.



## 7 Conclusion

In this work, we considered the problem of privacy-preserving record linkage as a preprocessing step for federated learning and a data enrichment tool for enhanced data quality, where PPRL could bring enormous potential for businesses, governments, and research organizations. Under this topic, we addressed the sparsity issue of comparative evaluation studies caused by the lack of well-defined evaluation frameworks that could provide uniform means of testing and benchmarking of various PPRL techniques. Such frameworks are paramount to better understand the advantages and weaknesses of different encoding and linkage methods with regard to linkage quality and scalability to large datasets. Without them it will be difficult for practitioners to make an informed decision about what type of linkage techniques to employ within their organisations [23].

Our contribution to this resembles in an extensive empirical evaluation study for state-of-the-art PPRL techniques, namely CLK, RBF and CLK-RBF for encoding, and MinHash LSH and Hamming LSH for blocking. We designed our empirical evaluation study based on surveying the literature and identifying the commonly used and well-established evaluation measures. Beside that, we proposed a set of design requirements for PPRL evaluation frameworks, that we used as guidelines to design a modular and easy-to-use prototype of an evaluation framework that is first of its kind in the literature. Our framework facilitates the integration of new developed techniques and benchmarking them against gold standards.

Our empirical evaluation showcased the viability of our proposed framework prototype via comparing different techniques obtained from two different software libraries. On the other hand, the results of the evaluation demonstrated a comparable performance between private and non-private techniques, which shows that current state-of-the-art PPRL techniques can be applied in real-world applications with a marginal loss of quality, but with the advantage of an enhanced scalability. Fine-tuning is necessary, however, to reach satisfactory results. These findings are in line with other experimental work on PPRL techniques [64, 65].

One limitation for applying PPRL in real-world applications lays within the lack of unique entity identifiers that serve as ground truth in experimental datasets, which precludes the performance evaluation of the tailored solution on the real-world data. To mitigate this inherent limitation, parameters of the PPRL pipeline can be fine-tuned in an evaluation study conducted over synthetic data that can be generated to hold similar characteristics to the real-world dataset of interest

[82].

We focused in our study on two aspects of PPRL which are linkage quality and scalability. While we presented supportive materials for privacy, related to the adversary models and types of attacks in Chapter 2, and also discussed some privacy aspects related to the resilience of the used techniques, evaluating the privacy of these techniques was outside the scope of our study, mainly because this dimension of PPRL is still immature in terms of standard measures and evaluation means [23, 32].

Since we used two datasets for our linkage quality experiments and proved the correspondence of the findings on larger dataset sizes that reached millions of records, we believe that our conclusions can be generalized to other datasets from other sources and of different characteristics. However, further evaluation is still needed to incorporate a wider set of techniques on many other different data sets with various data quality characteristics. Such an evaluation would help in obtaining a better overall understanding of the performance of different encoding and blocking techniques with regard to linkage quality and scalability.

For the future, we plan to extend our framework to include the privacy aspect, and to test the effect of missing values on the linkage quality. Also we plan to study the challenges connected to encoding of numerical and date attribute values instead of treating them as of textual type. Some studies already started addressing this aspect which is not yet mature in the application of PPRL [80].

# Bibliography

- [1] Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)* 51(4), 1–35 (2018)
- [2] Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. pp. 86–97 (2003)
- [3] Allahbakhsh, M., Ignjatovic, A., Benatallah, B., Bertino, E., Foo, N., et al.: Collusion detection in online rating systems. In: *Asia-Pacific Web Conference*. pp. 196–207. Springer (2013)
- [4] Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology* 23(2), 281–343 (2010)
- [5] Balaji, J., Javed, F., Kejriwal, M., Min, C., Sander, S., Ozturk, O.: An ensemble blocking scheme for entity resolution of large and sparse datasets. *arXiv preprint arXiv:1609.06265* (2016)
- [6] Baxter, R., Christen, P., Churches, T.: A comparison of fast blocking methods for record linkage; erschienen in: *Proceedings of the workshop on data cleaning, record linkage and object consolidation at the ninth acm sigkdd international conference on knowledge discovery and data mining; washington dc; 2003; o*
- [7] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13(7), 422–426 (1970)
- [8] Bonomi, L., Xiong, L., Chen, R., Fung, B.C.: Frequent grams based embedding for privacy preserving record linkage. In: *Proceedings of the 21st acm international conference on information and knowledge management*. pp. 1597–1601 (2012)
- [9] Bosu, A., Liu, F., Yao, D., Wang, G.: Collusive data leak and more: Large-scale threat analysis of inter-app communications. In: *Proceedings of*

the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 71–85 (2017)

- [10] Boyd, J.H., Randall, S.M., Ferrante, A.M.: Application of privacy-preserving techniques in operational record linkage centres. *Medical data privacy handbook* pp. 267–287 (2015)
- [11] Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. *Internet mathematics* 1(4), 485–509 (2004)
- [12] Brook, E.L., Rosman, D.L., Holman, C.D.J.: Public good through data linkage: measuring research outputs from the western australian data linkage system. *Australian and New Zealand journal of public health* 32(1), 19–23 (2008)
- [13] Brown, A.P., Borgs, C., Randall, S.M., Schnell, R.: Evaluating privacy-preserving record linkage using cryptographic long-term keys and multibit trees on large medical datasets. *BMC medical informatics and decision making* 17(1), 1–7 (2017)
- [14] Christen, P.: *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. 2012
- [15] Christen, P.: A comparison of personal name matching: Techniques and practical issues. In: *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*. pp. 290–294. IEEE (2006)
- [16] Christen, P.: Automatic record linkage using seeded nearest neighbour and support vector machine classification. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. pp. 151–159 (2008)
- [17] Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering* 24(9), 1537–1555 (2011)
- [18] Christen, P.: The data matching process. In: *Data matching*, pp. 23–35. Springer (2012)
- [19] Christen, P.: Data linkage: The big picture. *Harvard Data Science Review* 1(2) (2019)
- [20] Christen, P., Churches, T., Hegland, M.: Febrl—a parallel open source data linkage system. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. pp. 638–647. Springer (2004)

- [21] Christen, P., Gayler, R., Hawking, D.: Similarity-aware indexing for real-time entity resolution. In: Proceedings of the 18th ACM conference on Information and knowledge management. pp. 1565–1568 (2009)
- [22] Christen, P., Goiser, K.: Quality and complexity measures for data linkage and deduplication. In: Quality measures in data mining, pp. 127–151. Springer (2007)
- [23] Christen, P., Ranbaduge, T., Schnell, R.: Linking Sensitive Data: Methods and Techniques for Practical Privacy-Preserving Information Sharing. Springer International Publishing AG (2020)
- [24] Churches, T., Christen, P., Lim, K., Zhu, J.X.: Preparation of name and address data for record linkage using hidden markov models. BMC Medical Informatics and Decision Making 2(1), 1–16 (2002)
- [25] Dal Bianco, G., Galante, R., Heuser, C.A.: A fast approach for parallel deduplication on multicore processors. In: Proceedings of the 2011 acm symposium on applied computing. pp. 1027–1032 (2011)
- [26] Data61, C.: Anonlink private record linkage system. <https://github.com/data61/anonlink> (2017)
- [27] Dillinger, P.C., Manolios, P.: Fast and accurate bitstate verification for spin. In: International SPIN Workshop on Model Checking of Software. pp. 57–75. Springer (2004)
- [28] Durham, E.A., Kantarcioglu, M., Xue, Y., Toth, C., Kuzu, M., Malin, B.: Composite bloom filters for secure record linkage. IEEE transactions on knowledge and data engineering 26(12), 2956–2968 (2013)
- [29] Durham, E.A.: A framework for accurate, efficient private record linkage. Ph.D. thesis (2012)
- [30] Franke, M., Gladbach, M., Sehili, Z., Rohde, F., Rahm, E.: Scads research on scalable privacy-preserving record linkage. Datenbank-Spektrum 19(1), 31–40 (2019)
- [31] Franke, M., Sehili, Z., Rahm, E.: Primat: a toolbox for fast privacy-preserving matching. Proceedings of the VLDB Endowment 12(12), 1826–1829 (2019)
- [32] Franke, M., Sehili, Z., Rohde, F., Rahm, E.: Evaluation of hardening techniques for privacy-preserving record linkage. In: EDBT. pp. 289–300 (2021)

- [33] Ganta, S.R., Kasiviswanathan, S.P., Smith, A.: Composition attacks and auxiliary information in data privacy. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 265–273 (2008)
- [34] Gladbach, M., Sehili, Z., Kudrass, T., Christen, P., Rahm, E.: Distributed privacy-preserving record linkage using pivot-based filter techniques. In: 2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW). pp. 33–38. IEEE (2018)
- [35] Gu, L., Baxter, R.: Decision models for record linkage. In: Data mining. pp. 146–160. Springer (2006)
- [36] Heidt, C.M., Hund, H., Fegeler, C.: A federated record linkage algorithm for secure medical data sharing. In: German Medical Data Sciences: Bringing Data to Life, pp. 142–149. IOS Press (2021)
- [37] Herzog, T.N., Scheuren, F.J., Winkler, W.E.: Data quality and record linkage techniques. Springer Science & Business Media (2007)
- [38] Hmac, H.: Keyed-hashing for message authentication (1997)
- [39] Inan, A., Kantarcioglu, M., Ghinita, G., Bertino, E.: Private record matching using differential privacy. In: Proceedings of the 13th International Conference on Extending Database Technology. pp. 123–134 (2010)
- [40] Jiang, W., Clifton, C., Kantarcioglu, M.: Transforming semi-honest protocols to ensure accountability. Data & Knowledge Engineering 65(1), 57–74 (2008)
- [41] Karakasidis, A., Koloniari, G., Verykios, V.S.: Privateer: A private record linkage toolkit. In: CAiSe Forum. pp. 197–204 (2015)
- [42] Karakasidis, A., Verykios, V.S.: Secure blocking+ secure matching= secure record linkage. Journal of Computing Science and Engineering 5(3), 223–235 (2011)
- [43] Karapiperis, D., Gkoulalas-Divanis, A., Verykios, V.S.: Lshdb: a parallel and distributed engine for record linkage and similarity search. In: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW). pp. 1–4. IEEE (2016)
- [44] Karapiperis, D., Gkoulalas-Divanis, A., Verykios, V.S.: Federal: A framework for distance-aware privacy-preserving record linkage. IEEE Transactions on Knowledge and Data Engineering 30(2), 292–304 (2017)

- [45] Karapiperis, D., Vatsalan, D., Verykios, V.S., Christen, P.: Large-scale multi-party counting set intersection using a space efficient global synopsis. In: International Conference on Database Systems for Advanced Applications. pp. 329–345. Springer (2015)
- [46] Karapiperis, D., Vatsalan, D., Verykios, V.S., Christen, P.: Efficient record linkage using a compact hamming space. In: EDBT. pp. 209–220 (2016)
- [47] Konečný, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
- [48] Kong, C., Gao, M., Xu, C., Qian, W., Zhou, A.: Entity matching across multiple heterogeneous data sources. In: International Conference on Database Systems for Advanced Applications. pp. 133–146. Springer (2016)
- [49] Kroll, M., Steinmetzer, S.: Who is 1011011111....1110110010? automated cryptanalysis of bloom filter encryptions of databases with several personal identifiers. In: International Joint Conference on Biomedical Engineering Systems and Technologies. pp. 341–356. Springer (2015)
- [50] Kum, H.C., Krishnamurthy, A., Machanavajjhala, A., Reiter, M.K., Ahalt, S.: Privacy preserving interactive record linkage (ppirl). Journal of the American Medical Informatics Association 21(2), 212–220 (2014)
- [51] Kuzu, M., Kantarcioglu, M., Durham, E., Malin, B.: A constraint satisfaction cryptanalysis of bloom filters in private record linkage. In: International Symposium on Privacy Enhancing Technologies Symposium. pp. 226–245. Springer (2011)
- [52] Kuzu, M., Kantarcioglu, M., Durham, E.A., Toth, C., Malin, B.: A practical approach to achieve private medical record linkage in light of public resources. Journal of the American Medical Informatics Association 20(2), 285–292 (2013)
- [53] Lablans, M., Borg, A., Ückert, F.: A restful interface to pseudonymization services in modern web applications. BMC medical informatics and decision making 15(1), 1–10 (2015)
- [54] Li, F., Chen, Y., Luo, B., Lee, D., Liu, P.: Privacy preserving group linkage. In: International Conference on Scientific and Statistical Database Management. pp. 432–450. Springer (2011)

- [55] Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: 2007 IEEE 23rd International Conference on Data Engineering. pp. 106–115. IEEE (2007)
- [56] Lindell, Y.: Secure multiparty computation for privacy preserving data mining. In: Encyclopedia of Data Warehousing and Mining, pp. 1005–1009. IGI global (2005)
- [57] Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: l-diversity: Privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD) 1(1), 3–es (2007)
- [58] Mitzenmacher, M., Upfal, E.: Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis. Cambridge university press (2017)
- [59] Mohammed, N., Fung, B.C., Debbabi, M.: Anonymity meets game theory: secure data integration with malicious participants. The VLDB Journal 20(4), 567–588 (2011)
- [60] Nock, R., Hardy, S., Henecka, W., Ivey-Law, H., Patrini, G., Smith, G., Thorne, B.: Entity resolution and federated learning get a federated resolution. arXiv preprint arXiv:1803.04035 (2018)
- [61] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques. pp. 223–238. Springer (1999)
- [62] Pang, C., Hansen, D., et al.: Improved record linkage for encrypted identifying data. In: Proceedings of the 14th Annual Health Informatics Conference. pp. 164–168. Citeseer (2006)
- [63] Rahm, E., Do, H.H.: Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. 23(4), 3–13 (2000)
- [64] Ranbaduge, T., Vatsalan, D., Randall, S., Christen, P.: Evaluation of advanced techniques for multi-party privacy-preserving record linkage on real-world health databases: Ijpd (2017) issue 1, vol 1: 087, proceedings of the ipdln conference (august 2016). International Journal of Population Data Science 1(1) (2017)
- [65] Randall, S.M., Ferrante, A.M., Boyd, J.H., Bauer, J.K., Semmens, J.B.: Privacy-preserving record linkage on large real world datasets. Journal of biomedical informatics 50, 205–212 (2014)



- [66] Roughan, M., Zhang, Y.: Secure distributed data-mining and its application to large-scale network measurements. *ACM SIGCOMM Computer Communication Review* 36(1), 7–14 (2006)
- [67] Ryan, T., Holmes, B., Gibson, D.: A national minimum data set for home and community care. Australian Institute of Health and Welfare (1999)
- [68] Scannapieco, M.: *Data Quality: Concepts, Methodologies and Techniques. Data-Centric Systems and Applications*. Springer (2006)
- [69] Scannapieco, M., Figotin, I., Bertino, E., Elmagarmid, A.K.: Privacy preserving schema and data matching. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. pp. 653–664 (2007)
- [70] Schneier, B.: *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley & Sons (2007)
- [71] Schnell, R., Bachteler, T., Reiher, J.: Privacy-preserving record linkage using bloom filters. *BMC medical informatics and decision making* 9(1), 1–11 (2009)
- [72] Schnell, R., Bachteler, T., Reiher, J.: A novel error-tolerant anonymous linking code. Available at SSRN 3549247 (2011)
- [73] Schnell, R., Borgs, C.: Randomized response and balanced bloom filters for privacy preserving record linkage. In: *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. pp. 218–224. IEEE (2016)
- [74] Schnell, R., Richter, A., Borgs, C.: A comparison of statistical linkage keys with bloom filter-based encryptions for privacy-preserving record linkage using real-world mammography data. In: *HEALTHINF*. pp. 276–283 (2017)
- [75] Sehili, Z., Kolb, L., Borgs, C., Schnell, R., Rahm, E.: Privacy preserving record linkage with ppjoin. *Datenbanksysteme für Business, Technologie und Web (BTW 2015)* (2015)
- [76] Sweeney, L.: k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10(05), 557–570 (2002)
- [77] Toth, C., Durham, E., Kantarcioglu, M., Xue, Y., Malin, B.: Soempi: a secure open enterprise master patient index software toolkit for private

- record linkage. In: AMIA Annual Symposium Proceedings. vol. 2014, p. 1105. American Medical Informatics Association (2014)
- [78] Turgay, E.O., Pedersen, T.B., Saygın, Y., Savaş, E., Levi, A.: Disclosure risks of distance preserving data transformations. In: International Conference on Scientific and Statistical Database Management. pp. 79–94. Springer (2008)
  - [79] Vatsalan, D., Christen, P.: Scalable privacy-preserving record linkage for multiple databases. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. pp. 1795–1798 (2014)
  - [80] Vatsalan, D., Christen, P.: Privacy-preserving matching of similar patients. *Journal of biomedical informatics* 59, 285–298 (2016)
  - [81] Vatsalan, D., Christen, P., O’Keefe, C.M., Verykios, V.S.: An evaluation framework for privacy-preserving record linkage. *Journal of Privacy and Confidentiality* 6(1) (2014)
  - [82] Vatsalan, D., Christen, P., Verykios, V.S.: A taxonomy of privacy-preserving record linkage techniques. *Information Systems* 38(6), 946–969 (2013)
  - [83] Vatsalan, D., Christen, P., et al.: An iterative two-party protocol for scalable privacy-preserving record linkage (2012)
  - [84] Vatsalan, D., Karapiperis, D., Verykios, V.S.: Privacy-preserving record linkage. (2019)
  - [85] Vatsalan, D., Sehili, Z., Christen, P., Rahm, E.: Privacy-preserving record linkage for big data: Current approaches and research challenges. In: *Handbook of Big Data Technologies*, pp. 851–895. Springer (2017)
  - [86] Vo, K., Jonnagaddala, J., Liaw, S.T.: Statistical supervised meta-ensemble algorithm for medical record linkage. *Journal of biomedical informatics* 95, 103220 (2019)
  - [87] Winkler, W.E.: Methods for evaluating and creating data quality. *Information Systems* 29(7), 531–550 (2004)