# Secure and Accurate Two-step Hash Encoding for Privacy-Preserving Record Linkage ⋆

Thilina Ranbaduge[1], Peter Christen[1], and Rainer Schnell[2]

[1] Research School of Computer Science, The Australian National University,
Canberra, ACT 2600, Australia. thilina.ranbaduge@anu.edu.au
[2] Methodology Research Group, University Duisburg-Essen, Germany.

**Abstract.** In order to discover new insights from data, there is a growing need to share information that is distributed across multiple databases that are often held by different organisations. One key task in data integration is the calculation of similarities between records to identify pairs or sets of records that correspond to the same real-world entities. Due to privacy and confidentiality concerns, however, the owners of sensitive databases are often not allowed or willing to exchange or share their data with other organisations to allow such similarity calculations. In this paper we propose a novel privacy-preserving encoding technique that can be used to securely calculate similarities between sensitive values held in different databases. Our technique uses two-step hashing to encode values into an integer set representation that provides strong privacy guarantees and allows accurate similarity calculations. We provide a theoretical analysis of the accuracy and privacy of our encoding technique, and conduct an empirical study on large real databases containing several millions records. Our results show that our technique provides high security against privacy attacks and achieves better similarity accuracy compared to two state-of-the-art encoding techniques.

**Keywords:** Hashing · Jaccard similarity · Integer representation

## 1 Introduction

Application domains such as banking, health, and national security, increasingly require data from multiple sources to be integrated to allow efficient and accurate decision making [1]. Integrating databases can help to identify similar records that correspond to the same real-world entities. Linked records allow improvement of data quality, enrichment of the information known about individual entities, and facilitate the discovery of novel patterns and relationships between the entities that are represented by records in linked databases.

Because there is often a lack of common entity identifiers across the databases to be linked, the linking of records is commonly based on personal identifying attributes (known as *quasi-identifiers* [13]), such as first and last names, address details, and dates of birth [1]. Data quality issues, such as typographical errors

⋆ This work is funded by the Australian Research Council under DP160101934.

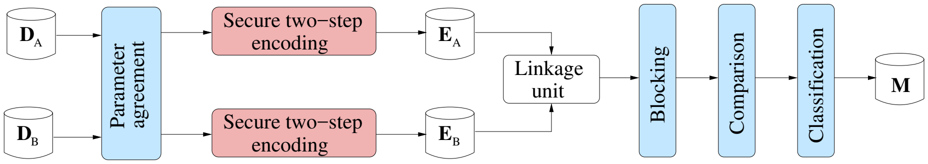

Fig. 1: Overview of the main steps of a secure privacy-preserving record linkage protocol using our proposed two-step encoding technique. After the initial parameter agreement phase each database owner applies two-step encoding for each record of their respective database. The encoded databases, $\mathbf{E}_A$ and $\mathbf{E}_B$, are then sent to a linkage unit (third party) which conducts blocking, comparison, and finally classification, resulting in the set $\mathbf{M}$ of matched record pairs.

and variations, in such attributes mean that approximate similarities need to be calculated between quasi-identifiers. Pairs of records that are highly similar can then be classified as matches (correspond to the same entity) [1,13].

However, quasi-identifying attributes can contain enough information to allow identification of unique individuals which can raise privacy and confidentiality concerns when the databases to be linked belong to different organisations. This is especially the case when the participating organisations do not trust each other, as is commonly the case in public-private sector collaboration [13].

Due to privacy and confidentially concerns, organisations are often not willing or authorised to exchange or share any sensitive data about the entities stored in their databases with any other party. This can severely limit or even prohibit the integration of databases across organisations [8]. Research in the area of *privacy-preserving record linkage* (PPRL) aims to develop techniques that facilitate the linking of databases without the need of any sensitive data to be shared between the organisations involved in the linkage process [5,13].

As shown in Fig. 1, PPRL is conducted by encoding or encrypting sensitive data at the database owners (DOs) before being exchanged with other third party organisations (such a linkage unit [5,13]) to calculate the similarities between records. At the end of such a PPRL process, only limited information about those compared record pairs that were classified as matches is revealed to the DOs [13]. Any PPRL technique must guarantee that no participating party can learn anything about the sensitive data in any of the databases. The PPRL process must also be secure such that no external adversary can learn any sensitive information about the entities in the databases that are being linked [13].

Any encoding or encryption method used in PPRL must facilitate approximate similarity calculations between sensitive values without the need for sharing the actual values [13]. Various techniques to securely calculate similarities between values have been proposed. They either rely on expensive security computations to achieve strong privacy guarantees, or they use efficient data masking or perturbation techniques that, however, can be vulnerable to cryptanalysis attacks that can re-identify sensitive values in an encoded database [2].

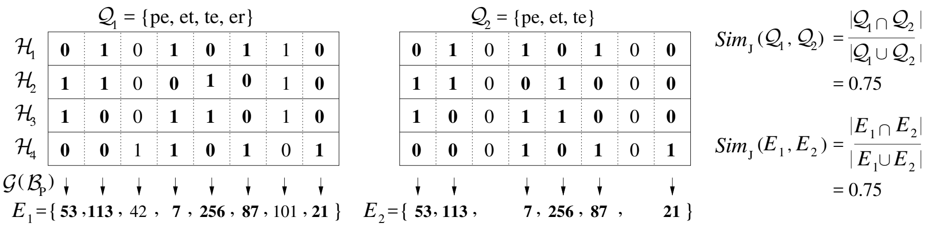
$\mathcal{Q}_1$ = {pe, et, te, er}
$\mathcal{Q}_2$ = {pe, et, te}
$\mathcal{H}_1$ 0 1 0 1 0 1 1 0
$\mathcal{H}_2$ 1 1 0 0 1 0 1 0
$\mathcal{H}_3$ 1 0 0 1 1 0 1 0
$\mathcal{H}_4$ 0 0 1 1 0 1 0 1
0 1 0 1 0 1 0 0
1 1 0 0 1 0 0 0
1 0 0 1 1 0 0 0
0 0 0 1 0 1 0 1
$\mathcal{G}(\mathcal{B}_p)$
$E_1$={ 53 ,113 , 42 , 7 , 256 , 87 , 101 , 21 }
$E_2$={ 53, 113 , 7 , 256 , 87 , 21 }
$Sim_J(\mathcal{Q}_1, \mathcal{Q}_2) = \frac{|\mathcal{Q}_1 \cap \mathcal{Q}_2|}{|\mathcal{Q}_1 \cup \mathcal{Q}_2|}$ = 0.75
$Sim_J(E_1, E_2) = \frac{|E_1 \cap E_2|}{|E_1 \cup E_2|}$ = 0.75

Fig. 2: An example of the basic idea of our two-step encoding technique, as described in Sect. 3. The two input values, ‘peter’ and ‘pete’, are first converted into character q-gram sets, $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and then hashed into bit vectors (rows) using the four hash functions $\mathcal{H}_1$ to $\mathcal{H}_4$, resulting in the two shown bit matrices. All non-zero column bit patterns, $\mathcal{B}_p$, are then hashed again using a second hash function, $\mathcal{G}$, which maps bit patterns to integers that are unique per position (for example, ‘0110’ from positions $p$=1 and $p$=5 is hashed into 53 and 256, respectively). The sets of integers, $E_1$ and $E_2$, are then the encodings of the input values which we use to privatly calculate the Jaccard similarity.

As illustrated in Fig. 2, we propose a novel efficient encoding technique that can be used to securely encode sensitive values while facilitating accurate similarity calculations between encoded values. Unlike existing encoding techniques [8,11], the aim of our technique is to provide privacy guarantees against recently proposed re-identification attacks [2,14] while at the same time allowing the accurate and efficient calculation of similarities between encoded values.

As we describe in Sect. 4, frequency attacks, a known weakness of the Bloom filter encoding technique [2,6,14], are prevented in our encoding technique because of the two hashing steps, while dictionary attacks [13] are prevented by adding a secret salt value to the second hashing step of our technique.

**Contribution**: We propose an encoding technique that allows the efficient calculation of similarities between sensitive string values in a privacy-preserving manner. We analyse the accuracy and privacy protection of our technique and apply it in the context of PPRL. We show that our encoding technique is significantly more secure than popular Bloom filter encoding [8,13,13], which is known to be vulnerable to cryptanalysis attacks [2,6,14]. We evaluate our technique on large real databases, which validates that, compared to existing PPRL encodings [8,11], this technique can efficiently and accurately calculate similarities between encoded values while providing privacy against cryptanalysis attacks.

## 2 Related Work

Encoding techniques for PPRL can be categorised into *perturbation* and *secure multi-party computation* (SMC) based techniques [13]. While SMC techniques are provably secure, they often incur high computation and communication costs. Perturbation based techniques are generally more efficient than SMC based techniques but have a trade-off between linkage quality, scalability, and privacy.

Schnell et al. [8] proposed a PPRL approach using Bloom filter (BF) encoding. A BF is a bit vector where all positions are initialised to 0. The elements of a set are mapped into a BF using $k > 1$ hash functions that set certain bit positions to 1. In PPRL, commonly character q-grams extracted from quasi-identifiers are hashed into BFs. Due to its efficiency of encoding and approximate matching of attribute values, BFs has almost become a standard for PPRL [13].

However, BF encoding can be susceptible to privacy attacks [2,6,14]. Due to the mapping of q-grams into bit positions, information about how the encoding has been performed can be learnt [2]. This can allow the re-identification of the encoded values. Sensitive values that occur frequently in an encoded database can lead to frequent bit patterns in BFs that can be identified [6], and even individual frequent q-grams can be found using pattern mining techniques [14]. This raises serious concerns about the applicability of BF encoding for PPRL.

BF hardening techniques, such as salting [10], balancing [9], and XOR-folding [9] have been proposed to improve the privacy of BF encoding. These techniques have a trade-off between improved resilience to attacks at the costs of linkage quality because similarities between hardened BFs are likely distorted. No detailed studies have been conducted on these trade-offs, and we therefore do not consider hardening in our evaluation but leave experiments for future work.

Smith [11] recently proposed a tabulation Min-hash based approach which showed both improved similarity and privacy protection on small data sets. To generate a bit vector of length $l$, first $l$ hash tables each containing random 64-bit strings are generated. For each value to be encoded, $l$ random bit strings are selected based on the Min-hash values generated from the value's q-gram set. From the bit strings the least significant bits are then concatenated to generate the final bit vector for the value. This prevents the mapping of 1-bits to q-grams, and existing privacy attacks will not be successful on this encoding method.

However, as we discuss in Sect. 5, this tabulation hash encoding approach [11] has significantly higher computational complexity compared to BF encoding. As a result, for encoding very large databases the practical use of this approach is questionable. Our proposed two-step hash encoding technique provides both high privacy protection while also being computationally efficient.

## 3 Secure Two-Step Hash Encoding of Sensitive Data

Our encoding technique aims to prevent an adversary from learning any information about what sensitive values have been encoded. To achieve this, we employ a two-step hashing process where a sensitive value is first hashed into a bit matrix, and then columns in this matrix are hashed again to generate an integer set representation of the sensitive input value, as shown in Fig. 2.

Algorithm 1 outlines the main steps involved in our encoding approach. First the quasi-identifier attributes, $A$, from each record $r$ in the database $\mathbf{D}$ are converted into sets of character q-grams of length $q$ in line 4 using the function *genQGramSet*() which returns one q-gram set $\mathcal{Q}$ for each record $r \in \mathbf{D}$.

**Algorithm 1:** ***Secure Two-step Hash Encoding***

```
Input:
- $\mathbf{D}$:  The database to be encoded                         - $l$:  Length of bit vectors (rows)
- $A$:  List of quasi-identifier attributes                        - $q$:  Q-gram length
- $\boldsymbol{\mathcal{H}}$:  List of hash functions, $\mathcal{H}_i$, with $1 \leq i \leq k$    - $s$:  Secret salt value
- $\mathcal{G}$:  Hash function used in the second step
Output:
- $\mathbf{E}$:  Encoded database
1:  $\mathbf{E} = \{\}$                        // Initialise an inverted index for the encoded database
2:  $k = |\boldsymbol{\mathcal{H}}|$                         // Get number of bit vectors (rows) to be generated
3:  foreach $r \in \mathbf{D}$ do:                     // Loop over all records $r$ in the database
4:    $\mathcal{Q} = genQGramSet(r, A, q)$           // Generate q-grams from attribute values in $r$
5:    $E = \{\}$, $\mathbf{B} = [\ ]$               // Initialise empty integer set and bit vector list for $r$
6:    foreach $\mathcal{H}_i \in \boldsymbol{\mathcal{H}}, 1 \leq i \leq k$ do:       // Step 1: Loop over each hash function in $\boldsymbol{\mathcal{H}}$
7:      $\mathcal{B}_i = genBitVector(l)$           // Generate a bit vector of length $l$ with all 0s
8:      $\mathcal{B}_i = hashQGrams(\mathcal{B}_i, \mathcal{H}_i, Q)$         // Hash the q-gram set into the bit vector
9:      $\mathbf{B}[i] = \mathcal{B}_i$                   // Add the bit vector to the list $\mathbf{B}$ (a new row)
10:   foreach $p \in [1, \cdots, l]$ do:    // Step 2: Loop over all positions (columns) in the matrix
11:     $\mathcal{B}_p = genColumnVector(\mathbf{B}, p)$        // Generate the bit vector for position $p$
12:     if $HW(\mathcal{B}_p) > 0$ do:           // Check the position bit vector has at least one 1-bit
13:       $e_p = \mathcal{G}(\mathcal{B}_p, p, s)$              // Hash the position bit vector with secret salt
14:       $E = E \cup \{e_p\}$                 // Add the integer representation to the set $E$
15:   $\mathbf{E}[r.id] = E$                // Add the set of encoded values for $r$ to the database
16: return $\mathbf{E}$
```

Similar to BF encoding [8], as the first step of our encoding process, in line 6 we iterate over the $k$ hash functions $\mathcal{H}_i$, with $1 \leq i \leq k$. For each $\mathcal{H}_i$, we first generate an empty (all 0s) bit vector $\mathcal{B}_i$ of length $l$ in line 7, and then hash each q-gram in the set $\mathcal{Q}$ into $\mathcal{B}_i$ using $\mathcal{H}_i$ in the function *hashQGrams*() in line 8. The resulting bit vector $\mathcal{B}_i$ is then added to the list $\mathbf{B}$ in line 9. This first hashing step generates a bit matrix with $k$ rows and $l$ columns, as shown in Fig. 2.

The second step of our encoding process starts in line 10, where we loop over the $l$ positions in the bit matrix. For each position $1 \leq p \leq l$, we first generate its column bit vector, $\mathcal{B}_p$, in line 11 using the function $genColumnVector()$. We then check if $\mathcal{B}_p$ contains at least one 1-bit (i.e. its Hamming weight (HW) is larger than 0). This check is required because generating an integer value from a column bit vector that contains only 0-bits can lead to incorrect Jaccard similarity calculations. If we do generate integers from 0-bit columns that are common to two bit matrices then we will have additional common integer values in their encodings, while not common 0-bit columns would lead to not common integer values. This can be seen in Fig. 2, where no integers are generated for positions 3 and 6 in $\mathbf{B}_2$. If there would, then these two extra integers in $E_2$ would result in a wrong Jaccard similarity between $E_1$ and $E_2$, calculated as 6/10.

In line 13 in Algo. 1, if a column bit vector, $\mathcal{B}_p$, does contain at least one 1-bit, then we use it, together with the position number, $p$, and the secret salt, $s$, to generate the integer, $e_p$, for that position using the hash function $\mathcal{G}$. The secret salt value is only known (and needs to be agreed upon) by the database owners before starting the encoding process. Using a secret salt provides security against dictionary attacks, because even if an adversary knows the hash function $\mathcal{G}$ she cannot simply try all possible bit patterns of length $k$ combined with all $l$ position numbers because they are not an input to $\mathcal{G}$ by themselves.

The requirement on the hash function $\mathcal{G}$ is that it returns integer values in a certain range, and that the probability of a hash collision is very low. One possibility for $\mathcal{G}$ is to use a hash function from the SHA family [1], such as $e_p = int(sha(g_p))$, where $g_p = str(\mathcal{B}_p) \oplus str(p) \oplus s$, where $str()$ converts it input into a string, and $\oplus$ indicates the string concatenation. The collision probability for such an approach will be very low, however the resulting integers will be very large and thus require more storage and communication time.

An alternative approach for $\mathcal{G}$ is to use the value $g_p$ calculated above as the seed for a pseudo random number generator [4] which generates integer values within a certain range, such as $[1, r]$. For a given range $r$, the probability of a collision (that two different inputs $g_p$ result in the same integer value) can then be calculated using the birthday paradox [12]. The total number of unique possible values of $g_p$ (the concatenation of $\mathcal{B}_p$ with $p$ and $s$ as described above) is $x = l \cdot (2^k - 1)$. The collision probability then is [12]: $P(x, r) = 1 - \prod_{i=1}^{x-1} (1 - (i/r)) \approx 1 - e^{-x(x-1)/2r}$. As we show in our experiments in Sect. 5, this collision probability will be less than 3%. Note that a single collision will only affect the estimated Jaccard similarity by a minor amount of around $\pm 1/l$.

It is even possible to construct a function for $\mathcal{G}$ that ensures there are no collisions at all by mapping the bit patterns from each position into a distinct, non-overlapping range of integers. Using again a pseudo random number generator, where $g_p$ is again the seed, we set the range for the possible integers generated for position $p$ as the interval $[e_p^{min}, e_p^{max}]$. We set $e_p^{min} = (p-1) \cdot r_p$ and $e_p^{max} = p \cdot r_p - 1$, and the range for a position as $r_p \gg 2^k - 1$.

Back to Algo. 1, once a column bit vector is hashed, in line 14 the generated integer, $e_p$, is added into the set of encoding values, $E$. Each set $E$ is then added into the encoded database $\mathbf{E}$ using the corresponding record identifier ($r.id$) as the key (in line 15). Finally, $\mathbf{E}$ can be used to compare encodings from another database where the same attributes $A$ have been encoded, and the same hash functions $\boldsymbol{\mathcal{H}}$ and $\mathcal{G}$, and the same secret salt value, $s$, have been used.

By assuming each hash operation is of $O(1)$ complexity, for a record $r$ that contains $n$ q-grams we require $n \cdot k$ hash operations in the first step and $l$ hash operations in the second step. Hence, the encoding of all $|\mathbf{D}|$ records in a database $\mathbf{D}$ in Algo. 1 has an overall complexity of $O(|\mathbf{D}| \cdot (n \cdot k + l))$.

## 4 Similarity Estimation and Privacy Analysis

We now discuss the two main aspects to of our two-step encoding technique: (1) How accurate the Jaccard similarity calculations based on integer set representations are, and (2) how strong the privacy protection of our encoding is.

**Accuracy of Similarity Estimation**: Our encoding approach hashes similar q-gram sets into integer sets that have more values in common, while dissimilar q-gram sets are hashed into integer sets that have less values in common. Two q-gram sets with a non-empty intersection should therefore be hashed into two integer sets that have values in common with some non-negligible probability.

Ideally, the Jaccard similarities [1] calculated on encoded integer sets should be the same as the Jaccard similarity between corresponding q-gram sets. However, as with most hashing techniques [12], there is a chance of collisions especially in the first step of our encoding approach which might result in somewhat different estimated Jaccard similarities.

Formally, given two q-gram sets $\mathcal{Q}_1$ and $\mathcal{Q}_2$, with $\mathcal{Q}_1 \neq \mathcal{Q}_2$ and $\mathcal{Q}_1 \cap \mathcal{Q}_2 \neq \emptyset$, when using our encoding approach then the Jaccard similarity is calculated using the encodings, $E_1$ and $E_2$. Therefore, if $sim_J(\mathcal{Q}_1, \mathcal{Q}_2) \neq sim_E(E_1, E_2)$, where $sim_J()$ and $sim_E()$ are the q-gram and integer sets Jaccard similarity functions, respectively, then in the context of PPRL either a false match ($sim_E > sim_J$) or a false non-match ($sim_J > sim_E$) might occur.

By assuming a family $\boldsymbol{\mathcal{H}}$ of independent hash functions is used in step 1, we can estimate $sim_E$ between two integer sets compared to $sim_J$ between their corresponding q-gram sets. We conduct this estimation based on the number of hash collisions. Let us assume $c = |\mathcal{Q}_1 \cap \mathcal{Q}_2|$ is the number of common q-grams, and $d = |(\mathcal{Q}_1 \cup \mathcal{Q}_2) \setminus (\mathcal{Q}_1 \cap \mathcal{Q}_2)|$ the number of different q-grams that occur in only one of the two q-gram sets. $sim_J$ between $\mathcal{Q}_1$ and $\mathcal{Q}_2$ can now be calculated as $sim_J(\mathcal{Q}_1, \mathcal{Q}_2) = c/(c + d)$. We assume $k$ hash functions are used to hash q-gram sets into bit vectors of length $l$, and that there are no hash collisions in the second step of our approach, as we discussed in the previous section.

The likelihood that none of the $k \cdot c$ hashes of the $c$ common q-grams are hashed to a certain position in the bit vectors is $(\frac{l-1}{l})^{kc}$. The expected number of non-zero bit positions that are set by the $c$ common q-grams will then be $n_c = l \cdot (1 - \frac{l-1}{l})^{kc}$. Similarly, the number of non-zero bit positions set by all $c+d$ q-grams is $n_a = l \cdot (1 - \frac{l-1}{l})^{k(c+d)}$. This allows us to calculate the estimated Jaccard similarity $sim_E$ between $E_1$ and $E_2$ as $sim_E(E_1, E_2) = n_c/n_a$.

Using the actual and estimated Jaccard similarities, $sim_J$ and $sim_E$, calculated between $\mathcal{Q}_1$ and $\mathcal{Q}_2$ and between $E_1$ and $E_2$, respectively, we can now calculate the difference $\delta = sim_J(\mathcal{Q}_1, \mathcal{Q}_2) - sim_E(E_1, E_2)$. For the values of $k$=$[10, 20, 30]$ and $l$=$[250, 500, 1000]$ we use in the experiments in Sect. 5, the average difference was $\delta = 0.03$ when $sim_J \geq 0.5$ are considered ($c \geq d$). This is a reasonable assumption for PPRL where one is only interested in highly similar values such as names that share at least half of their q-grams [1].

The estimated Jaccard similarities are unlikely to be higher than the actual similarities, i.e. $sim_E \leq sim_J$. This means for PPRL there can be missed matches, however false matches will unlikely be generated by our encoding method. This is different with popular Bloom filter encoding, which commonly generates false matches [8], as can be seen in our experimental results in Fig. 4.

Preventing false matches and false non-matches are important in many applications of PPRL, such as health or national security, where linking people wrongly or missing links of people that likely are a match can result in difficult to correct biases and thus wrong conclusions [7]. When the length of the bit vectors, $l$, increases the difference $\delta$ between the estimated and the actual Jaccard similarities gets lower. $\delta$ is also smaller for more similar values being compared. Longer bit vectors will therefore lead to higher linkage quality.

**Privacy Analysis**: The use of a two-step hashing process prohibits that an adversary can construct the corresponding list $\mathbf{B}$ of bit vectors using the frequencies of integer values in the sets $E$. Each bit position can result in $2^k - 1$ possible non-zero bit patterns, and since the function $\mathcal{G}$ can be constructed in different ways and it includes a secret salt value, the adversary cannot directly map an integer value to its corresponding correct bit pattern. Over the $l$ bit positions, the total number of unique non-zero bit patterns in $\mathbf{B}$ is $l^{2^k-1}$, while a BF of length $l$ can (only) hold $2^l - 1$ possible non-zero bit patterns.

A brute-force attack by an adversary to try to reconstruct $\mathbf{B}$ created in the first step of our approach is therefore extremely difficult, especially when larger values for $k$ and $l$ are used. Furthermore, even if the adversary can reconstruct $\mathbf{B}$, she still needs to then re-identify the way q-grams are hashed into bit rows using the list of hash functions $\boldsymbol{\mathcal{H}}$ in the first step of our approach.

An adversary can however try to conduct a frequency attack on the integer values generated by our encoding approach to try to identify which q-gram(s) are represented by which integer values(s). For Bloom filter encoding, it has been shown that frequent encoded values and even frequent q-grams can result in frequent bit patterns that can be successfully re-identified [2,6,14].

In our encoding approach, such a frequency based attack is only possible if a single q-gram is hashed to a certain position. In such a case the frequency of the resulting integer value would correspond to the frequency of the single q-gram hashed to this position. However, such an attack becomes challenging if several q-grams are hashed into the same position in the first step of our encoding approach. This is because the combinations and frequencies of bit patterns will be a mix based on several q-grams, thereby preventing an adversary from correctly aligning the frequencies of integer values with the frequencies of q-grams.

To analyse the privacy provided by our approach we therefore calculate the likelihood that several q-grams are hashed to the same position. We assume each of the $n = |\mathcal{Q}|$ q-grams in a q-gram set $\mathcal{Q}$ is mapped into a different position for each hash function $\mathcal{H}_i$, with $i \leq i \leq k$, such that the probability of these hash mappings are uniformly distributed across the $l$ positions. The probability that more than one (from $n$) q-gram in $\mathcal{Q}$ is hashed to the same position in $\mathbf{B}$ is $P(k, l, n) = 1 - \left(\frac{l-1}{l}\right)^{k(n-1)}$. Assuming one of the $n$ q-grams has been hashed to a certain position, the factor in this equation is the probability that none of the other $n - 1$ q-grams (each hashed $k$ times) hits this position. One minus this is then the probability that at least one other q-gram hits the position.

Assuming $k = [10, 20, 30]$ and $l = [250, 500, 1000]$, and with $n = [10, 20]$, we obtain probabilities ranging from $P = 0.1$ for $l = 1000$ and $k = 10$, up to $P = 0.9$ for $l = 250$ and $k = 30$. The probability increases as $k$ and $n$ get larger, but decreases with increasing $l$. For many settings, more than half of all bit columns will have at least two q-grams hashed to them. This will significantly reduce the chances an adversary has to correctly align an integer value with its encoded q-gram. Hence, any re-identification of sensitive values in a database based on a frequency alignment of integer values with q-grams will unlikely be possible, especially if values from more than one attribute are encoded.

## 5 Experiment Evaluation

We used three data set pairs that are extracted from two real voter registration databases from the US states of North Carolina (NC) (See: `http://dl.ncsbe.gov/`) and Michigan (MC) (See: `http://michiganvoters.info`), respectively. We use *first name*, *last name*, *street address*, *city*, and *zipcode* as the set of attributes, $A$, because these are commonly used for record linkage [1,13].

We use two NC data sets (NCL) collected in April 2014 and October 2019, and two MC data sets (MCL) collected in July 2013 and September 2016. Both NCL and MCL contain between 6.2 and 7.6 million records where over 98% are exact matching record pairs. Due the high skewness of exact matches we only used NCL and MCL to evaluate scalability. To evaluate linkage quality we used a data set pair (NCS) with 222,251 and 224,061 records, respectively, and containing 66.6% matches. In this data set the vast majority of records (98.9%) had at least one value change in any of the attributes we selected above.

We set $k = [10, 20, 30]$ in the first hashing step, and $l = [250, 500, 1000]$. We set the q-gram length $q = 2$ and used random hashing [9] in the first hash step to encode q-grams. As we discussed in Sect. 3, we used a pseudo random number generator in the second hash step to encode column bit vectors.

We compared our two-step hash (2SH) encoding approach with two baselines. The first is popular Bloom filter (BF) encoding [8]. Following earlier BF work in PPRL, we set the BF parameters as $l$ =1000 bits, $k = 30$, and $q = 2$ [8,13]. The second baseline is the tabulation hash (TH) encoding proposed by Smith [11]. Following [11], we used 8 tabulation keys each of 64 bits length to generate one bit array of length $l$ =1000 bits to encode the attribute values in a record.

As shown in Fig. 1, to evaluate linkage quality we simulated a three-party PPRL protocol [13]. To allow fair comparison, we used phonetic blocking on all encoding techniques where we used Soundex [1] on *first name*, *last name*, and *city*, and the first three digits of *zipcode* as the blocking keys. Following [8], in classification we set the threshold $t$ ranging from 0.1 to 1.0, in 0.1 steps.

We evaluated the scalability using runtime, and measured linkage quality using precision and recall [3]. Precision is the ratio of the number of true matches correctly classified against the total number of record pairs compared, while recall is the ratio of the number of true matches correctly classified against the total number of true matches across two data sets.

We could not use any privacy attacks on bit vectors [2,14], such as BFs, on our 2SH encoding to evaluate privacy. Hence, we conducted a frequency attack [2] on the integer values generated by our approach. We aligned frequent integer values with frequent q-grams aiming to re-identify encoded attribute values. We assumed the worst case scenario where the LU has gained access to an unencoded database $\mathbf{D}$ of a database owner (DO), and is trying to re-identify the encoded values in integer sets $\mathbf{E}$ of the other DO by using $\mathbf{D}$. However, such an attack is highly unlikely since the DOs do not send $\mathbf{D}$ to any other party.

For implementation we used Python (version 2.7). All experiments were run on a server with 64-bit Intel Xeon (2.4 GHz) CPUs, 128 GBytes of memory, and Ubuntu 14.04. The programs are available from the authors.

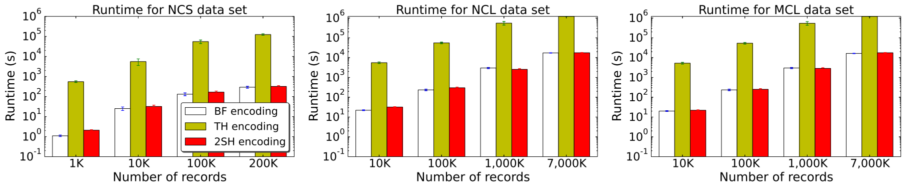

Fig. 3: Average runtime plots of BF [8], TH [11], and our 2SH encoding with different data sets. We stopped TH on NCL and MCL after two weeks.

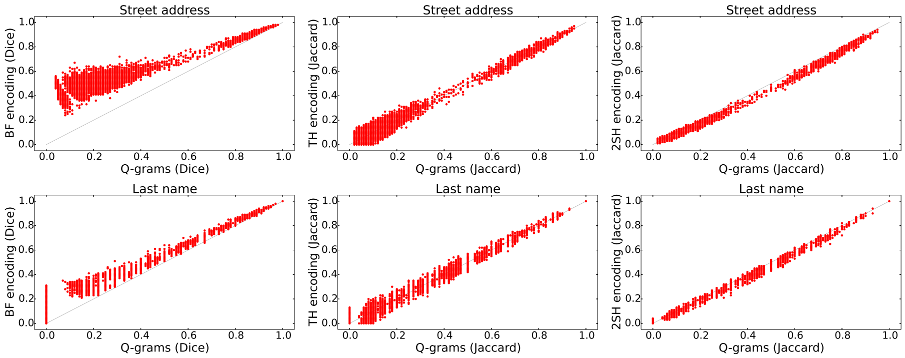

Fig. 4: Similarity plots of BF [8] (left), TH [11] (middle), and 2SH (right) encoding, with different attributes. We used $k$ = 30 and $l$ =1000 for 2SH encoding.

**Results and Discussion**: Figure 3 illustrates the scalability of our encoding approach in terms of average runtime. As can be seen, 2SH scales linearly with the data set size. However, 2SH consumes slightly higher runtime compared to BFs due to the second hashing step of $l$ column bit patterns into integer values. Both 2SH and BF encodings are significantly faster than TH which requires more hash encodings. Furthermore, 2SH consumes around 10% to 20% more memory compared to BF and TH encodings due to the integer set encoding.

Figure 4 shows scatter plots of q-gram based similarities versus corresponding similarities on encodings. For BFs we used Dice similarity [1], while for TH and 2SH we used Jaccard similarity. As can be seen, similarities calculated on BFs are much higher than the corresponding q-gram set similarities especially between sets that only have a few common q-grams. This is due to collisions where different q-grams are hashed to the same BF bit positions. TH results in more accurate similarities, where encoded similarities can be both above and below the q-gram set similarities. Hence, the similarities calculated on BFs and with TH can lead to inaccurate PPRL results, and thus wrong follow-up analysis.

As also shown in Fig. 4, 2SH results in more accurate similarities calculated on encoded integer sets compared to both BF and TH. 2SH unlikely results in false matches which will lead to more precise classification of record pairs [7]. We also note that as $l$ increases the similarity difference between the integer and q-

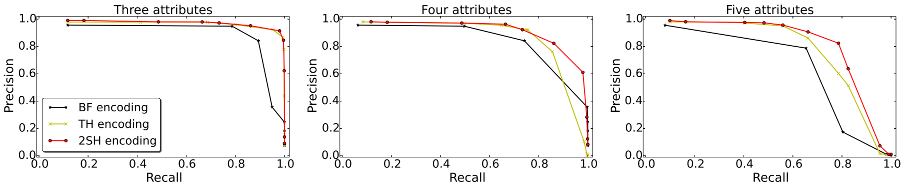

Fig. 5: The linkage quality results of BF [8], TH [11], and our 2SH encoding for the NCS data set. We used $k$ = 30 and $l$ =1000 for our 2SH encoding.

Table 1: The number of correct 1-to-1 attribute value re-identifications based on the alignments of the top $n_q$ frequent q-grams with corresponding integer values for different $k$, $l$, and different number of attributes using the NCL data set. *all* represents frequency alignment of all extracted q-grams from the data set.

| | | Three attributes | Four attributes | Five attributes |
|---|---|---|---|---|
| $l$ | $k$ | $n_q$ = 10 / 100 / *all* | $n_q$ = 10 / 100 / *all* | $n_q$ = 10 / 100 / *all* |
| 250 | 10 | 0 / 0 / 1 | 0 / 0 / 1 | 0 / 0 / 0 |
| | 20 | 0 / 0 / 1 | 0 / 0 / 0 | 0 / 0 / 0 |
| | 30 | 0 / 0 / 0 | 0 / 0 / 0 | 0 / 0 / 0 |
| 500 | 10 | 0 / 1 / 2 | 0 / 1 / 1 | 0 / 0 / 1 |
| | 20 | 0 / 1 / 1 | 0 / 0 / 1 | 0 / 0 / 0 |
| | 30 | 0 / 0 / 1 | 0 / 0 / 0 | 0 / 0 / 0 |
| 1000 | 10 | 1 / 2 / 4 | 0 / 1 / 2 | 0 / 1 / 1 |
| | 20 | 1 / 1 / 2 | 0 / 1 / 1 | 0 / 0 / 1 |
| | 30 | 0 / 1 / 2 | 0 / 1 / 1 | 0 / 0 / 1 |

gram sets gets lower which results in less false negatives. Figure 5 shows linkage quality results in terms of precision and recall. 2SH achieves higher recall and precision compared to both BF and TH. As we encode more attribute values, recall decreases similar to BF and TH encodings. This is because different q-grams are hashed into the same positions leading to different integer values to be generated for similar q-gram sets and resulting in false negatives. As we discussed in Sect. 4, increasing $l$ can reduce the number of false negatives.

Finally, Table 1 shows the number of correct attribute value re-identifications based on the 1-to-1 assignment of integer values to q-grams that only hash individually in a column position. We noted the number of such correct re-identifications increases for smaller numbers of attribute values and with the increase of $l$. This is because the frequencies of q-grams can be correctly identified as less q-grams are mapped to a certain column position in the first hashing step. However, as we hash more attribute values with more hash functions, an adversary will not be able to re-identify attribute values because not enough frequency information is available to identify q-grams that are encoded into integer values. Hence, conducting such a frequency attack upon integer sets that are encoded with q-grams from different attributes will unlikely be successful.

## 6 Conclusion and Future work

We have presented a novel encoding approach that can be used for privacy-preserving record linkage (PPRL). We used a two-step hashing process to encode sensitive values into integer sets. Our analysis showed that our encoding can provide high linkage quality, validated on large real databases, where it can achieve higher quality similarity calculations compared to other encoding techniques used in PPRL. Our approach also provides privacy against frequency attacks on integer sets which prevents the re-identification of encoded sensitive values. As future work we aim to extend the experiments with different hashing techniques in our encoding approach. Another future research avenue is to find the optimal values for the length of bit vectors and the number of hash function used in the encoding process that can maximise both linkage quality and privacy. Furthermore, we plan to adapt existing privacy attacks for our encoding and conduct a privacy comparison with other PPRL techniques.

## References

1. Christen, P.: Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer (2012)
2. Christen, P., Ranbaduge, T., Vatsalan, D., Schnell, R.: Precise and fast cryptanalysis for Bloom filter based privacy-preserving record linkage. IEEE TKDE **31**(11), 2164–2177 (2018)
3. Hand, D.J., Christen, P.: A note on using the F-measure for evaluating record linkage algorithms. Statistics and Computing **28**(3), 539–547 (2018)
4. Johnston, D.: Random Number Generators – Principles and Practices: A Guide for Engineers and Programmers. Walter de Gruyter GmbH & Co (2018)
5. Karapiperis, D., Verykios, V.: An LSH-based blocking approach with a homomorphic matching technique for privacy-preserving record linkage. IEEE TKDE **27**(4), 909–921 (2014)
6. Kuzu, M., Kantarcioglu, M., et al.: A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In: PET. pp. 226–245. Waterloo, Canada (2011)
7. Ruggles, S., Fitch, C.A., Roberts, E.: Historical census record linkage. Annual Review of Sociology **44**, 19–37 (2018)
8. Schnell, R., Bachteler, T., Reiher, J.: Privacy-preserving record linkage using Bloom filters. BMC Med Inform Decis Mak **9** (2009)
9. Schnell, R., Borgs, C.: Randomized response and balanced Bloom filters for privacy preserving record linkage. In: ICDMW. pp. 218–224. Barcelona (2016)
10. Schnell, R.: An efficient privacy-preserving record linkage technique for administrative data and censuses. Statistical Journal of the IAOS **30**(3), 263–270 (2014)
11. Smith, D.: Secure pseudonymisation for privacy-preserving probabilistic record linkage. Journal of Information Security and Applications **34**, 271–279 (2017)
12. Suzuki, K., et al.: Birthday paradox for multi-collisions. In: ICISC. pp. 29–40. Busan, Korea (2006)
13. Vatsalan, D., Sehili, Z., Christen, P., Rahm, E.: Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges. Springer (2017)
14. Vidanage, A., Ranbaduge, T., Christen, P., Schnell, R.: Efficient pattern mining based cryptanalysis for privacy-preserving record linkage. In: IEEE ICDE. pp. 1698–1701. Macau (2019)