Precise and Fast Cryptanalysis for Bloom Filter Based Privacy-Preserving Record Linkage

Peter Christen, Thilina Ranbaduge, Dinusha Vatsalan, and Rainer Schnell

Abstract—Being able to identify records that correspond to the same entity across diverse databases is an increasingly important step in many data analytics projects. Research into privacy-preserving record linkage (PPRL) aims to develop techniques that can link records across databases such that besides the record pairs classified as matches no sensitive information about the entities in these databases is revealed. A popular technique used in PPRL is to encode sensitive values into Bloom filters (bit vectors), which has the advantage of allowing approximate matching using character q-grams. PPRL based on Bloom filter encoding has shown to be accurate and scalable to large databases, and is thus now being used in real-world PPRL systems in Australia, Canada and the UK. However, recent studies have shown that Bloom filters used for PPRL are vulnerable to cryptanalysis attacks that can re-identify some of the sensitive values encoded in these Bloom filters. While previous such attack methods were slow and required knowledge of various encoding parameters, we present a novel efficient attack which exploits how attribute values are encoded into Bloom filters. Our attack method does not require knowledge of the encoding function or its parameter settings used. It is able to correctly re-identify with high precision q-grams that could not have been hashed to certain Bloom filter bit positions, and using these re-identified q-grams it can then re-identify attribute values with high precision. Our method is significantly faster than earlier PPRL cryptanalysis attacks, and in our experimental evaluation it is able to successfully re-identify attribute values from large real-world databases in a few minutes.

Index Terms—Re-identification, frequency analysis, entity resolution, privacy evaluation, privacy attack.

1 INTRODUCTION

There is an increasing need in many data engineering and analytics projects to integrate data from multiple sources to enable more sophisticated analysis, to detect and correct inconsistencies in data, or to enrich individual data sources. Record linkage is the task of identifying records that correspond to the same real-world entities across databases when unique entity identifiers (such as social security numbers) are not available [1]. Instead, approximate matching techniques have to be used to calculate the similarities between records, followed by a classification of the compared record pairs into matches (assumed to refer to the same entity) and non-matches (assumed to refer to different entities). Applications of record linkage range from the health sector and business analytics to national security [1].

In many applications the records to be linked refer to people, for example patients or tax payers, and the attributes required to calculate the similarities between such records include names, addresses, dates of birth, and so on. These quasi-identifiers in many applications cannot be exchanged or shared across organizations due to privacy laws and regulations, or for commercial reasons [2], [3].

Research in the area of privacy-preserving record linkage (PPRL) [4], [5], [6] over the past two decades has aimed to develop techniques that allow the linkage of databases without the need of any private or confidential information to be shared between the organizations involved in a linkage project. PPRL is conducted such that only limited information about the record pairs classified as matches (such as their record identifiers or values from selected non-identifiable attributes) is revealed to the organization(s) that require(s) the linkage results. The techniques used in PPRL must also guarantee that no internal or external attacker (adversary) can compromise any sensitive information about the entities in the databases that are being linked [8], [5].

The general idea of PPRL is to encode (or encrypt) the attribute values required for linkage at the source databases such that approximate similarities between records can still be calculated [4]. Various encoding techniques for PPRL have been developed over the past decades. These can be categorized [5] into secure multi-party computation (SMC), perturbation, and hybrid techniques [7]. SMC based techniques generally employ cryptographic operations and they are provably secure [8]. However, to allow for approximate matching, many SMC based techniques have high computation and communication costs, making them currently not scalable to linking large databases [5].

Perturbation based techniques generally provide a trade-off between linkage quality, efficiency, and privacy. Practical PPRL applications [2], [9], [10] are mostly based on perturbation techniques that provide adequate privacy protection while achieving acceptable linkage quality and performance. It is however vital that practical PPRL applications are secure and cannot be attacked as otherwise sensitive information can potentially be revealed [4], [5].

One popular perturbation based technique for PPRL is Bloom filter (BF) encoding [11]. A BF is a bit vector into which elements of a set are hashed into. As we describe in
Section 3 textual values can be encoded into BFs by converting a string into its set of q-grams (character strings of length q). Techniques to encode numerical values into BFs have also been proposed [12, 13]. The similarity between two BFs can be calculated using a set-based similarity measure [1].

BF based PPRL techniques have the advantage of being efficient and facilitate the linkage of large real-world databases, where approximate matching is crucial due to errors and variations in attribute values [2, 10]. Recent research has however shown that BF based techniques can be vulnerable to cryptanalysis attacks that aim to re-identify the sensitive attribute values encoded in BFs [14, 15, 16, 17, 18]. These attacks use the frequency counts and bit patterns in a set of BFs to map frequent bit patterns to frequent plain-text values. Existing cryptanalysis attacks are however not practical because they assume an attacker has knowledge of certain parameter settings used during BF encoding, and/or they have high computational costs.

In this paper we extend our recently proposed attack method [14] which exploits how q-grams are hashed into BFs. For each bit position, this method identifies a set of possible and a set of not possible q-grams, and then re-identifies attribute values using only the sets of possible q-grams. However, as our experiments show, these sets have low precision (i.e. they contain q-grams that were not hashed to a certain bit position) which leads to low re-identification accuracy. In contrast, our novel attack method uses the sets of not possible q-grams which have higher precision.

Contributions: We contribute an efficient cryptanalysis attack on BF encoding for PPRL which exploits bit patterns in a set of frequent BFs that are frequency aligned with a set of frequent plain-text attribute values. We present a novel approach to identify q-grams that must have been hashed to certain bit positions in a BF (we name these assigned q-grams), and two new methods to refine and expand the sets of possible, not possible, and assigned q-grams. We then propose a novel approach to re-identify attribute values based on the sets of not possible q-grams. We experimentally evaluate our attack method on several real-world databases, confirming its ability to re-identify q-grams at BF bit positions with high precision. Our attack can re-identify sensitive attribute values from databases that contain hundreds of thousands of records in minutes.

Outline: We next provide the required background on PPRL and describe existing attacks on BF based PPRL. In Section 3 we discuss BF encoding for PPRL as well as techniques that aim to make BFs more resilient to attacks. We describe our attack method in Section 4 and analyze its complexity and limitations in Section 5. Using real-world databases, in Section 6 we experimentally show how our attack method is able to re-identify attribute values with high precision, and we provide recommendations on how to use BF encoding securely for practical PPRL. We conclude our work in Section 7 with possible research directions.

2 Background and Related Work

Privacy in record linkage was first investigated in the mid 1990s and since then PPRL has been an active research field. A variety of techniques have been developed to allow linkage using encoded or encrypted values of quasi-identifying attributes across two or more databases [4, 5, 6].

Because real-world data are often dirty, exact matching is not sufficient to achieve high linkage quality in most PPRL applications [5]. This is especially the case when personal identifying attributes, such as names and addresses, are used for linkage because these are prone to variations and typographical errors [1]. Any encoding used for PPRL needs to preserve the original similarities between values to allow approximate matching between encoded values.

PPRL based on SMC techniques encrypt values using, for example, homomorphic encryption [8]. Because these techniques are often computationally expensive [8], recent research in PPRL has developed perturbation based techniques using, for example, noise addition using differential privacy [7], generalization such as k-anonymity, or embedding spaces [5, 6]. These techniques facilitate scalable PPRL at the cost of a trade-off with privacy and linkage quality.

BF encoding is a widely used perturbation technique for PPRL that can support approximate matching by preserving the similarities in the BF space [5, 14, 16]. Schnell et al. [11] introduced an approximate matching approach for string values encoded into BFs, as we describe in Section 3 while Vatsalan and Christen [12] and Karapiperis et al. [13] recently proposed two approaches for encoding numerical attributes. A recent study showed that BF encoding can achieve similar linkage quality compared to traditional record linkage techniques on unencoded values [10].

The efficiency and effectiveness of BFs has sparked wide ranging research using this technique, and first practical applications of BF based PPRL are now being implemented in several countries [2, 9, 10]. However, as we describe next and summarize in Table 5, BFs are prone to cryptanalysis attacks that aim to re-identify encoded values based on frequency information and background knowledge.

A first such attack using a constraint satisfaction solver was studied by Kuzu et al. [16], where plain-text attribute values are mapped to BF bit patterns from an encoded database that meet constraints such as frequency alignments. Kuzu et al. [17] then investigated the accuracy of their attack using two real-world databases. A medical database was used as the encoded database and a public voter registration database as the plain-text database. The experiments showed that the attack is less likely to be accurate in many cases and required significant computational resources with a larger number of frequent names.

Niedermeyer et al. [15] proposed a cryptanalysis attack on BF encodings using filtering and statistical analysis techniques that exploits the linear combination of the values used to generate hash functions [11]. The attack generates a set of possible bit patterns (so called atoms) that are aligned with plain-text values according to their frequencies. The experiments conducted showed that the proposed attack was able to re-identify 934 frequent values from 7,580 surnames before it was stopped. Kroll and Steinmetzer [15] extended this attack for the case where values from several attributes are encoded into one BF. Their attack was able to correctly re-identify 44% of 100,000 attribute values. Both these attack methods, [18] and [15], assume the double hashing (DH) scheme [29] used by Schnell et al. [11] for BF encoding, which we describe in the following section.
We recently proposed an efficient attack method [14] that works independently of the BF encoding method used and does not require any knowledge about the encoding parameter settings applied. Based on frequent attribute values that are aligned with frequent BFs, sets of possible and not possible q-grams are identified for each BF bit position. Using the possible q-gram sets, attribute values are re-identified based on BF bit patterns. While this attack is fast, for many BFs it re-identifies wrong attribute values because the sets of not possible q-grams are not fully used. In this paper we substantially improve this attack as we describe in Section 4.

3 Bloom Filter Encoding and Hardening

BFs were proposed by Bloom in 1970 as an efficient way to represent sets [21]. A BF is a bit vector of length $l$ bits where initially all bits are set to 0. A set of $k$ independent hash functions, $h_1, \ldots, h_l$, each with range $[1, \ldots, l]$, is used to hash each element $s$ of a set $S$ into a BF by setting corresponding bit positions to 1: $b[h_j(s)] = 1$, with $1 \leq j \leq k$.

For a given query element $s_q$, if any of the $k$ hashed positions for $s_q$ is 0 (formally: $\exists j:1 \leq j \leq k : b[h_j(s_q)] = 0$), then $s_q$ cannot be in the set $s$. While the hashing of elements into BFs cannot lead to false negatives [21], false positives are possible due to collisions, as shown in Figure 1.

In PPRL, the (sensitive) values in the attributes used to compare records first need to be converted into sets that can be hashed into BFs, where string values are commonly converted into sets of character q-grams [11]. For each record in a database to be linked, either one BF is generated per attribute used for the linkage (attribute-level BF), or one combined single BF is generated where all linkage attributes are hashed into (record-level BF) [11, 19]. Using record-level BFs will increase privacy, however it will likely lead to reduced linkage quality because only a single similarity can be calculated between two records (compared to one similarity per attribute when attribute-level BFs are used) [5].

The BFs generated by a database owner for each record in their database are then sent to a linkage unit (LU) that calculates the similarities for pairs of BFs (as described below) and classifies these pairs as matches or non-matches [11, 19]. PPRL can also be conducted without a LU by exchanging BF segments among the database owners to distributively calculate the similarities between BFs [22].

A set-based similarity function such as the Jaccard or Dice coefficient [1] can be used to calculate the similarity between two BFs. As shown in Figure 1 for two BFs, $b_1$ and $b_2$, the Dice coefficient is calculated as $sim_D(b_1, b_2) = 2c / (x_1 + x_2)$, where $c$ is the number of bit positions that are set to 1 in both BFs (the common 1-bits), and $x_1$ and $x_2$ are the number of 1-bits in $b_1$ and $b_2$, respectively [5, 11].

The initial proposal of BF encoding for PPRL used a double hashing scheme (DH) [11, 20], where the $k$ bit positions for an element to be hashed are determined by the sum of the integer representation of two independent hash functions. However, DH has a weakness in that it generates a much reduced number of bit patterns that can be exploited by a cryptanalysis attack [15, 18]. As an alternative, random hashing (RH) has recently been proposed [25], where the random seeds for the $k$ hash functions are based on the actual q-grams in the value to be encoded, leading to more diverse bit patterns that could not be successfully attacked by some previous attack methods [15, 18]. However, as our experiments in Section 3 show, our proposed attack can be successful on both the DH and RH encoding methods.

The vulnerability of BFs used in PPRL with regard to cryptanalysis attacks has recently been addressed by the development of BF hardening techniques [23]. A first approach to make BF encoding more secure is to use record-level instead of attribute-level BFs. With record-level BFs much less frequency information is available that could be exploited by a cryptanalysis attack [16, 18].

Another proposed hardening approach for BFs is XOR-folding, where a BF of length $l$ bits is split into two halves of length $l/2$ each, and then bit-wise exclusive OR (XOR) is applied on these two shorter BFs to combine them into one new BF [24]. The XOR operation makes it impossible to reconstruct the actual 0 or 1 bit values in the original BF. A different hardening approach is balancing [23], where a BF of length $l$ bits is concatenated with its negated copy (all bits flipped) followed by a permutation of the 2l bits. As a result, all BFs generated from a database will have exactly half of their bits set to 1 (i.e. have a uniform Hamming weight of $l$) and thus less frequency information is available that could be exploited by an attack. In our experiments in Section 6 we show that our attack can be successful even when XOR-folding or balancing has been applied.

4 Precise and Efficient Q-gram Based Cryptanalysis on Bloom Filters

Our proposed attack consists of four steps as we describe in detail in this section, and illustrate in Figures 2 to 6. Compared to our previous attack [14] (described in Section 3), our new methods substantially improve the quality of the re-identified attribute values, as confirmed in the experimental evaluation in Section 6.

As with existing attacks on BFs for PPRL [14, 15, 16, 17, 18], we assume an attacker has access to a set of BFs, $B$, and their frequencies. However, unlike earlier attacks, our approach does not require any knowledge about the parameters used in the encoding process, such as the actual hash functions and their number $k$, nor the hashing approach (such as DH or RH) used.

We assume $B$ represents a set of BFs that encode sensitive values from one or a few attributes. These are the values we aim to re-identify, i.e. we want to assign plaintext values to BFs. Based on the frequency distribution of the Hamming weights (number of 1-bits) of the BFs in $B$, an attacker can guess which attribute(s) have been encoded.
into these BFs, because different attributes (such as first name, surname, or postcode) have distinctive distributions of Hamming weights [23]. The attacker can sample values from a large public database (such as a telephone directory) and select a set of plain-text values, \( V \), from a single or a few attributes where their corresponding q-gram sets have a frequency distribution that is similar to the distribution of Hamming weights of the BF set \( B \) to be attacked.

As for notation, we use bold letters for BFs, sets and lists (with upper-case bold letters for sets or lists of BFs, sets and lists) and normal type letters for integer or string values. We denote sets with curly and lists with square brackets (where lists have an order while sets do not), the frequency of how often a specific BF \( b_i \in B \) occurs with \( b_i.f \), and for each plain-text value \( v_i \in V \) its frequency with \( v_i.f \).

### 4.1 Q-gram Candidate Set Generation

In this first step of our attack method we identify for each BF bit position \( p \) the sets of q-grams that are possible and not possible, respectively, at that position, as shown in Figure 2 and detailed in Algorithm 1 [14]. Our approach exploits the construction principle of BFs as follows.

**Theorem 1.** Hashing q-grams into Bloom filters: For a given BF, \( b \), and set of q-grams, \( q \), extracted from an attribute value, \( v \), a 1-bit in a certain position, \( p \), in \( b \) implies that at least one q-gram \( q_i \in q \) from \( v \) must hash to \( p \). On the other hand, a 0-bit at a certain position, \( p \), implies no q-gram \( q_i \in q \) from \( v \) could have been hashed to \( p \).

**Proof:** As described in Section 3, the elements \( s \) of a set \( s \) are hashed into a BF of length \( l \) bits using \( k \geq 1 \) hash functions \( h_1, \ldots, h_k \). Assuming a BF is initialized with all bits set to 0, the bit at position \( p \) (with \( 1 \leq p \leq l \)), \( b[p] \), is set to 1 if and only if \( b[p] = 1 \iff (3s \in s : h_j(s) \leq k : h_j(s) = p) \). A bit at position \( p \) can only be 0 if none of the \( k \) hash functions 'hits' position \( p \) for any of the elements in: \( b[p] = 0 \iff (v_s \in s : h_j(v) \leq k : h_j(s) \neq p) \).

Algorithm 1 starts by initializing two empty candidate sets for each BF position \( p \): \( c^+ \) of possible q-grams and \( c^- \) of not possible q-grams. In lines 2 and 3 we identify the BFs in \( B \) and the attribute values in \( V \) that occur at least \( f_m \) times and include them in two lists \( B_F \) and \( V_F \). In line 4 we then sort both these lists according to their frequencies in reverse order (most frequent first), and in line 5 we align a BF \( b_i \) and corresponding attribute value \( v_i \) into the sorted list \( A \) of pairs \( (b_i, v_i) \). We only consider such pairs as long as both \( b_i \) and \( v_i \) have a unique frequency compared to the next, less frequent, \( b_j \) and \( v_j \). This criterion ensures we do not have an attribute value that could correspond to two or more BFs and vice versa, as this would likely lead to an incorrect alignment of a BF with an attribute value and thus incorrect mappings of q-grams to BF bit positions.

Starting in line 6, we iterate over pairs of aligned BFs and attribute values, \( (b_i, v_i) \in A \), and in line 7 we convert \( v_i \) into its set \( q_i \) of character q-grams of length \( q \). Then we loop over all BF bit positions, \( p \), in line 8 and if the bit in BF \( b_i \) at position \( p \) is 1 then in line 10 we add the q-grams in set \( q_i \) to the set \( c^+ \) of possible candidate q-grams at that position because this 1-bit means at least one q-gram from \( q_i \) was hashed to position \( p \) (following Theorem 1). Conversely, if the bit in BF \( b_i \) at position \( p \) is 0 then we add all q-grams in \( q_i \) to \( c^- \) in line 12 because no q-gram from the set \( q_i \) could have been hashed to that position.

**Algorithm 1: Possible and not possible q-gram set generation [14]**

**Input:**
- \( V \): Attribute values and their frequencies from a public database
- \( B \): BFs and their frequencies from the sensitive database
- \( l \): BF length
- \( q \): Length of substrings to extract from attribute values
- \( f_m \): Minimum frequency for BFs and attribute values

**Output:**
- \( C^+ \): List of possible q-gram sets at each BF position
- \( C^- \): List of not possible q-gram sets at each BF position
- \( A \): Aligned tuples of frequent BFs and frequent attribute values

**Proof:** As described in Section 3, the elements \( s \) of a set \( s \) are hashed into a BF of length \( l \) bits using \( k \geq 1 \) hash functions \( h_1, \ldots, h_k \). Assuming a BF is initialized with all bits set to 0, the bit at position \( p \) (with \( 1 \leq p \leq l \)), \( b[p] \), is set to 1 if and only if \( b[p] = 1 \iff (3s \in s : h_j(s) \leq k : h_j(s) = p) \). A bit at position \( p \) can only be 0 if none of the \( k \) hash functions 'hits' position \( p \) for any of the elements in: \( b[p] = 0 \iff (v_s \in s : h_j(v) \leq k : h_j(s) \neq p) \).

Algorithm 1 starts by initializing two empty candidate sets for each BF position \( p \): \( c^+ \) of possible q-grams and \( c^- \) of not possible q-grams. In lines 2 and 3 we identify the BFs in \( B \) and the attribute values in \( V \) that occur at least \( f_m \) times and include them in two lists \( B_F \) and \( V_F \). In line 4 we then sort both these lists according to their frequencies in reverse order (most frequent first), and in line 5 we align a BF \( b_i \) and corresponding attribute value \( v_i \) into the sorted list \( A \) of pairs \( (b_i, v_i) \). We only consider such pairs as long as both \( b_i \) and \( v_i \) have a unique frequency compared to the next, less frequent, \( b_j \) and \( v_j \). This criterion ensures we do not have an attribute value that could correspond to two or more BFs and vice versa, as this would likely lead to an incorrect alignment of a BF with an attribute value and thus incorrect mappings of q-grams to BF bit positions.

Starting in line 6, we iterate over pairs of aligned BFs and attribute values, \( (b_i, v_i) \in A \), and in line 7 we convert \( v_i \) into its set \( q_i \) of character q-grams of length \( q \). Then we loop over all BF bit positions, \( p \), in line 8 and if the bit in BF \( b_i \) at position \( p \) is 1 then in line 10 we add the q-grams in set \( q_i \) to the set \( c^+ \) of possible candidate q-grams at that position because this 1-bit means at least one q-gram from \( q_i \) was hashed to position \( p \) (following Theorem 1). Conversely, if the bit in BF \( b_i \) at position \( p \) is 0 then we add all q-grams in \( q_i \) to \( c^- \) in line 12 because no q-gram from the set \( q_i \) could have been hashed to that position.
### Q-gram Position Assignment

In this section we propose a method that uses the list $C^p$ of possible q-gram sets, as generated in Algorithm 1, to identify q-grams that must have been hashed to a certain bit position (we name these as assigned q-grams). While $C^p$ contains the q-grams that possibly have been hashed to bit position $p$, so far we have not validated if any of them actually has been hashed to this position. Our method identifies q-grams where for a given frequent BF and attribute value pair $(b_i, v_i) \in A$ (as generated in Algorithm 1) there is only one single q-gram that could have generated the 1-bit for $v_i$ at a certain position $p$, as illustrated in Figure 3.

Algorithm 2, which identifies assigned q-grams, starts by initializing one empty set of assigned q-grams, $C^A[p]$, per BF bit position $p$. The loop in line 2 then iterates over the aligned pairs of frequent BFs $b_i$ and frequent attribute values $v_i$ in $A$ as generated by Algorithm 1. For each BF bit position $p$, if the bit $b_i[p]$ is set to 1 then in line 6 we intersect the q-gram set $q_i$ (as generated from $v_i$ in line 3) with the set of possible q-grams at that position, $C^p[p]$. If the result of this intersection is a single q-gram (the test in line 7, where $|\cdot|$ denotes the cardinality of a set), then this q-gram is added to the set of assigned q-grams at position $p$ in line 8. At the end of Algorithm 2 we have the new list of q-gram sets $C^A[p]$ that contain those q-grams that must have been hashed to a certain position $p$.

#### Algorithm 2: Q-gram position assignment

<table>
<thead>
<tr>
<th>Algorithm 2: Q-gram position assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
</tr>
<tr>
<td>- $A$: List of aligned frequent attribute values and BFs (from Algo. 1)</td>
</tr>
<tr>
<td>- $C^p$: List of possible q-grams for BF positions (from Algo. 1)</td>
</tr>
<tr>
<td>- $l$: Length of BFs</td>
</tr>
<tr>
<td>- $q$: Length of substrings to extract from attribute values</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td>- $C^A$: List of assigned q-gram sets at each BF position</td>
</tr>
</tbody>
</table>

1: $C^A[p] = \{\}$, $1 \leq p \leq l$  // Initialize assigned q-gram sets
2: for $(b_i, v_i) \in A$ do:  // Loop over pairs of freq BF and attr value
3:     $q_i = genQGramSet(v_i, q)$  // Convert attr value into q-grams
4:     for $1 \leq p \leq l$ do:  // Loop over BF bit positions
5:         if $(b_i[p] == 1)$ then:
6:             $q_{\text{p}} = q_i \cap C^p[p]$  // Get set of possible q-grams at position
7:                 if $(|q_{\text{p}}| == 1)$ then:  // Only one possible q-gram
8:                     $C^A[p] = C^A[p] \cup q_{\text{p}}$  // Add to set of assigned q-grams
9:         return $C^A$

### 4.3 Q-gram Set Refinement and Expansion

A drawback of the candidate set generation method described in Section 4.1 is that only a small number of frequent attribute values, $v_i$, can accurately be aligned in list $A$ to the corresponding frequent BFs, $b_i$, that encode these values. The reason for this is that the frequencies of the attribute values encoded into BFs are likely somewhat different from the frequencies in a plain-text database that an attacker has access to, as illustrated in the left side of Figure 2.

A frequency based ordering of values in the same attribute from two similar databases will likely lead to the same ordering of the TOP most frequent values. However, for less frequent values the ordering at some point will likely become different. For example, if ‘smith’ is the most frequent surname in a population and it is considerably more frequent than ‘miller’, then it is likely that ‘smith’ is also more frequent than ‘miller’ in two large databases that are samples of this population. However, in one of these databases the surname ‘williams’ might be ranked 20th and ‘rendell’ 21st, while in another database their order is reversed. Note that most existing attacks on BF encoding for PPRL make this assumption of aligning frequent BFs and frequent plain-text values [15, 16, 17, 18].

Therefore, as frequencies become lower and the differences between the respective frequencies of two BFs or two attribute values become smaller, wrong assignments of attribute values to BFs (that encode a different value) can happen. Because the list $A$ in Algorithm 1 likely only includes a small number of attribute values, the number of q-grams in those values is likely only a fraction of all possible q-grams that occur in a domain. In our experiments only around 5% of all possible q-grams for first names and surnames are included in the values $v_i \in A$.

Our first improvement identifies new not possible positions for q-grams that are in the sets of possible q-grams, while the second improvement expands the sets of possible, not possible and assigned q-grams with additional q-grams. Both improvements are based on analyzing the q-gram sets.
and bit patterns between frequent (as included in the list A in Algorithm 1) and not frequent attribute values and BFs, respectively, as detailed in Algorithm 3.

We use the example shown in Figure 4 to illustrate our refinement and expansion methods. For each frequent value \( v_j \in A \) (as returned by Algorithm 1), we identify the not frequent values \( v_j \in V \setminus V_F \) with \( V_F \) the set of all frequent attribute values in \( A \) whose q-gram sets \( q_i \) are either a proper subset of the q-gram set \( q_i \) of the frequent value \( v_j \), or a proper superset of \( q_i \). For example, in Figure 4, the q-gram sets of 'pet' and 'peter' are subsets of the q-gram set of 'peter', while the q-gram sets of 'peters' and 'petersen' are supersets of the q-gram set of 'peter'.

This refinement and expansion process depends upon the availability of sub- and supersets of q-gram sets as generated from attribute values, which is dependent on data characteristics such as the size of alphabets, the length of attribute values, and their frequency distributions. As Table 1 shows, in our experimental data sets such sub- and supersets seem to be quite common even in attributes where all values are of the same length or where attribute values have a large variety. For example, even when all Zipcodes in a data set have the same length, some have a q-gram set that is a sub- or superset of others. When using \( q = 2 \), Zipcode '28725', \( 28787, 27225 \), is a superset of '28728', \( 28787, 7272 \).

Let us denote the set of all not frequent attribute values with \( V_N \), with \( V_N = V \setminus V_F \). For a given frequent attribute value \( v_j \in V_F \), we find the set of all shorter attribute values \( V_i^S \subset V_N \) where their q-gram sets are proper subsets of the q-gram set \( q_i \) of \( v_j \) (with \( q_i = genQGramSet(v_j, q_i) \)), and the set of all longer attribute values \( V_i^L \subset V_N \) where their q-gram sets are proper supersets of \( q_i \). More formally:

\[ V_i^S = \{ v_j \in V_N : genQGramSet(v_j, q_i) \subset q_i \} \]

\[ V_i^L = \{ v_j \in V_N : genQGramSet(v_j, q_i) \supset q_i \} \]

Similarly, for each frequent BF \( b_j \in A \), we can identify candidate BFs in the set of not frequent BFs, \( B_N \) (where \( B_N = B \setminus B_F \) and \( B_F \) is the set of all frequent BFs in \( A \)), that potentially can encode a value \( v_j \in V_i^S \) as those BFs \( b_j \in B_N \) that have less 1-bits than \( b_j \), and all 1-bits of \( b_j \) are also set to 1 in \( b_j \). If a BF \( b_j \) has even a single 1-bit in a position where \( b_j \) has a 0-bit then \( b_j \) cannot encode a value in \( V_i^S \). Similarly, candidate BFs \( b_j \) that could encode a value \( v_j \in V_i^L \) are those BFs that have 1-bits in all the positions where \( b_j \) also has a 1-bit, plus they need to have at least one extra 1-bit (but likely more) at a position where \( b_j \) has a 0-bit. If there is a single position where \( b_j \) has a 0-bit but \( b_j \) has a 1-bit then \( b_j \) cannot encode an attribute value in \( V_i^L \) because this 0-bit in \( b_j \) means the corresponding value \( v_j \) does not contain all q-grams that \( v_i \) contains.

For a given frequent BF \( b_j \in A \), we define \( B_j^S \) and \( B_j^L \), to be the candidate sets of not frequent BFs that can encode shorter and longer values in \( V_i^S \) and \( V_i^L \), respectively:

\[ B_j^S = \{ b_j \in B_N : (b_j \cap b_j) \wedge hw(b_j) > hw(b_j) \} \]

\[ B_j^L = \{ b_j \in B_N : (b_j \cap b_j) \wedge hw(b_j) < hw(b_j) \} \]

The \( \cap \) operator denotes the bit-wise logical AND, and the function \( hw(b) \) counts the number of 1-bits in a BF \( b \), i.e., returns its Hamming weight.

Because of collisions, it might be possible that an extra q-gram that only occurs in a value \( v_j \in V_i^L \) is hashed to the same bit positions as the q-grams in \( v_i \). Given \( k > 1 \) hash functions are used this is highly unlikely. The expected fraction of 1-bits in a BF of length \( l \) after \( n = 1 | q | \) q-grams have each been hashed \( k \) times is:

\[ f_1 = 1 - (1 - 1/l)^kn \]

(see page 111 in [25]). The probability \( P \) for one extra q-gram to be hashed \( k \) times to bit positions that have already been set to 1 by other q-grams is therefore \( P = f_i^k \). For example, for a BF that has 60% 1-bits and \( k = 30 \) hash functions are used, this probability becomes \( P = 2.21 \cdot 10^{-7} \). Even if this would happen, it would only mean that a BF \( b_j \in B_N \) would not be included in \( B_j^L \) however it would not lead to a wrong refinement or expansion of q-gram sets.

If the alignment in Algorithm 1 of frequent attribute values with frequent BFs into the list \( A \) is correct, and if there are shorter \( (V_i^S) \) and longer \( (V_i^L) \) not frequent attribute values for a given frequent value \( v_i \in A \), then the BFs identified in the sets \( B_i^S \) and \( B_i^L \) must encode such shorter and longer attribute values, respectively.

In our refinement and expansion approach detailed in Algorithm 3, in lines 4 to 9, for all frequent BF and attribute value pairs \( (b_i, v_i) \in A \) we first identify all the shorter and longer attribute values, \( V_i^S \) and \( V_i^L \), and the BFs \( B_i^S \) and \( B_i^L \). Note that if there are several shorter or longer values and several corresponding BFs then we do not know which shorter value \( v_j \in V_i^S \) corresponds to which BF \( b_j \in B_i^S \), and which longer value \( v_j \in V_i^L \) corresponds to which BF \( b_j \in B_i^L \). The larger the sets \( B_i^S \), \( B_i^L \), \( V_i^S \), and \( V_i^L \) are, the less accurate the refinement and expansion process will become because an increasing number of q-grams and bit positions will differ between \( v_i \) and the \( v_i \)'s and \( b_i \) and the \( b_i \)'s, respectively. We therefore use a parameter \( m \) to limit the size of these sets in Algorithm 3 (lines 10 and 21).

### Table 1

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Unique values</th>
<th>Subsets</th>
<th>Supersets</th>
</tr>
</thead>
<tbody>
<tr>
<td>First name</td>
<td>22,101</td>
<td>5,321 / 24.08%</td>
<td>9,104 / 41.19%</td>
</tr>
<tr>
<td>Surname</td>
<td>40,089</td>
<td>8,707 / 21.72%</td>
<td>15,459 / 38.56%</td>
</tr>
<tr>
<td>Address</td>
<td>210,154</td>
<td>3,436 / 1.64%</td>
<td>3,551 / 1.69%</td>
</tr>
<tr>
<td>City</td>
<td>735</td>
<td>27 / 3.67%</td>
<td>14 / 1.90%</td>
</tr>
<tr>
<td>Zipcode</td>
<td>824</td>
<td>24 / 2.91%</td>
<td>111 / 13.47%</td>
</tr>
</tbody>
</table>

The number and percentage of attribute values (based on their q-gram sets, with \( q = 2 \)) in the NCVR data set (as described in Section 5), that are a sub- and superset of at least one other attribute value.
Algorithm 3: Q-gram set refinement and expansion

Input:
- \( V \): Attribute values and their frequencies from a public database
- \( B \): BFs and their frequencies from the sensitive database
- \( A \): List of aligned frequent attribute values and BFs (from Alg 1)
- \( l \): Length of BFs
- \( q \): Length of substrings to extract from attribute values
- \( m \): Maximum size of attribute value and BF sets

Output:
- \( C^p \): List of possible q-gram sets at each BF position
- \( C^n \): List of not possible q-gram sets at each BF position
- \( C^d \): List of assigned q-gram sets at each BF position

1. \( C^p_i[p] = \emptyset \), \( C^n_i[p] = \emptyset \), \( C^d_i[p] = \emptyset \), \( 1 \leq p \leq l \)
2. \( V_P = \{ v_i : (v_i,b_i) \in A \} \), \( B_P = \{ b_i : (v_i,b_i) \in A \} \)
3. \( V_N = \{ v_i \in V : v_i \notin V_P \} \), \( B_N = \{ b_i \in B : b_i \notin B_P \} \)
4. for \( (b_i,v_i) \in A \) do
   // Loop over pairs of freq BF and attr value
5. \( q_i = \text{genQGramSet}(v_i,q) \) // Convert attr value into q-grams
6. \( V_S = \{ v_j \in V_N : \text{genQGramSet}(v_j,q) \subseteq q_i \} \)
7. \( V_L = \{ v_j \in V_N : \text{genQGramSet}(v_j,q) \supset q_i \} \)
8. \( B_S = \{ b_j \in B_N : (b_j \otimes b_i) \land \text{hw}(b_i) > \text{hw}(b_j) \} \)
9. \( B_L = \{ b_j \in B_N : (b_j \otimes b_i) = b_i \} \land \text{hw}(b_i) = \text{hw}(b_j) \}

10. if \((0 < |V_S| \leq m) \land (0 < |B_S| \leq m) \) then: // Refinement
11. \( q_a = \cup_{v_j \in V_S} \text{genQGramSet}(v_j,q) \subseteq q_i \)
12. \( q_d = q_a \cup q_i \) // q-grams that only occur in \( v_i \)
13. \( b_j = (\oplus b_v \land b_i) \in B_S \) // Bit-wise OR of BFs
14. if \((q_a \neq \emptyset) \land (b_j \neq b_i) \) then
15. for \( 1 \leq p \leq l \) do
16. \( B_i[p] = 0 \) \land \( b_j[p] = 1 \) then
17. \( C^p_i[p] = C^p_i[p] \cup q_i \) // Add to possible q-grams
18. \( C^n_i[p] = C^n_i[p] \cup q_i \) \land \( \emptyset \) // Add to not possible q-grams
19. if \((v_i[p] = 1) \land (b_j[p] = 0) \) then
20. \( C^d_i[p] = C^d_i[p] \cup q_i \) // Add to assigned q-grams

21. if \((0 < |V_L| \leq m) \land (0 < |B_L| \leq m) \) then: // Expansion
22. \( q_a = \cup_{v_j \in V_L} \text{genQGramSet}(v_j,q) \subseteq q_i \)
23. \( q_d = q_a \cup q_i \) // Q-grams that only occur in all \( v_j \in V_L \)
24. \( b_j = (\oplus b_v \land b_i) \in B_L \)
25. if \((q_a \neq \emptyset) \land (b_j \neq b_i) \) then
26. for \( 1 \leq p \leq l \) do
27. \( B_i[p] = 0 \) \land \( b_j[p] = 0 \) then
28. \( C^p_i[p] = C^p_i[p] \cup q_i \) // Add to possible q-grams
29. \( C^n_i[p] = C^n_i[p] \cup q_i \) // Add to not possible q-grams
30. \( C^d_i[p] = C^d_i[p] \cup q_i \) // Add to assigned q-grams
31. \( b_j[p] = 0 \) \land \( b_i[p] = 0 \) then
32. \( C^p_i[p] = C^p_i[p] \cup q_i \) // Add to possible q-grams
33. \( C^n_i[p] = C^n_i[p] \cup q_i \) // Add to not possible q-grams
34. \( C^d_i[p] = C^d_i[p] \cup q_i \) // Add to assigned q-grams
35. return \( C^p_i, C^n_i, \) and \( C^d_i \)

A small value of \( m \) will lead to a smaller refined and expanded q-gram sets \( C^p, C^n \) and \( C^d \) (as generated by Algorithm 3) that are of higher quality, while increasing \( m \) will lead to larger refined and expanded q-gram sets that however might contain more incorrectly identified q-grams as \( m \) gets larger. We will evaluate the quality of re-identified q-gram sets with different values of \( m \) in Section 4.

4.3.1 Refinement

Independent of the sizes of \( V_S \) and \( B_S \), we observe that a BF bit position that is 0 in all the \( b_j \in B_S \) but 1 in \( b_i \) can only encode a q-gram that occurs in \( v_i \) but in none of the \( v_j \in V_S \). Based on this, in lines 11 and 12 in Algorithm 3 we get the union \( q_i \) of all q-grams from the attribute values \( v_j \in V_S \), and then extract the set of differing q-grams, \( q_d \), as those q-grams that only occur in \( v_i \) but in none of the \( v_j \in V_S \). In line 13 we generate the bit-wise OR \((\oplus)\) of all BFs in \( B_S \) as \( b_o \), because any position that is 0 after this operation (but is 1 in \( b_o \)) can encode only q-grams in \( q_o \).

If \( q_d \) is not empty (i.e. there are differing q-grams) and \( b_o \) differs from \( b_i \), then we loop over all bit positions in line 15 and if for a certain position \( p \) the bit in \( b_o[p] = 0 \) but \( b_i[p] = 1 \) then we know that this position can only encode the differing q-grams in \( q_o \). In line 17 we therefore add the q-grams in \( q_o \) to the set of possible q-grams at that position, \( C^d_i[p] \), and in line 18 we add the q-grams in \( q_o \) that occur in any of the shorter attribute values to the set of not possible q-grams at that position, \( C^n_i[p] \).

A special case to consider is if there is only one shorter attribute value, \( v_j \), and one corresponding BF, \( b_j \), and only one q-gram differs between \( v_i \) and \( v_j \) (line 19). In this case we know that this q-gram must have been hashed to a certain position, and therefore in line 20 we add the q-gram to the set of assigned q-grams at that position, \( C^d_i[p] \).

4.3.2 Expansion

Similar to the refinement approach, independent of the sizes of \( V_L \) and \( B_L \), a BF bit position that is 0 in \( b_i \) and in all \( b_j \in B_L \) must encode a q-gram that occurs in all \( v_j \in V_L \). On the other hand, if none of the \( b_j \in B_L \) have a 1-bit at a given position then none of the q-grams in any of the \( v_j \in V_L \) could have been hashed to this position. Based on this, in lines 22 and 23 in Algorithm 3 we generate the union and intersection of all q-gram sets as \( q_a \) and \( q_o \), respectively, from the \( v_j \in V_L \). In line 24 we identify all q-grams that occur in all longer (not frequent) attribute values but not in the frequent value as \( q_d \). In lines 25 and 26 we calculate the bit-wise OR \((\oplus)\) of all BFs in \( B_L \) as \( b_o \), and the bit-wise AND \((\land)\) of all BFs in \( B_L \) as \( b_o \). The former is required to identify bit positions where no q-gram in \( q_o \) could have been hashed to, while the latter is needed to identify positions where the extra q-grams in \( q_o \) could have been hashed to.

In line 27, if both \( q_d \) is not empty (i.e. there are differing q-grams) and \( b_o \) differs from \( b_i \), then we loop over all bit positions (line 28) and for any position \( p \) where \( b_o[p] = 1 \) and \( b_i[p] = 0 \) in line 30 we add the set of extra q-grams, \( q_o \), that occur in all longer values but not the frequent value \((\land)\) to the set of possible q-grams at that position, \( C^d_i[p] \).

Additionally, if there is only one longer attribute value, \( v_j \), and one BF, \( b_j \), and only one q-gram differs between \( v_i \) and \( v_j \) (line 31), then we know this single q-gram in \( q_o \) must have been hashed to this position and so we add it in line 32 to the set of assigned q-grams, \( C^d_i[p] \), at this position.

As described above, if a BF position \( p \) is 0 in all the BFs \( b_j \in B_L \) and also 0 in the frequent BF \( b_i \), (the test in line 33) then we know that no q-gram in any longer value \( v_j \in V_L \) could have been hashed to this position. In line 34 we therefore add the q-grams in \( q_o \) to the set of not possible q-grams, \( C^n_i[p] \), at that position.

4.3.3 Merging Q-gram Sets

At the end of the refinement and expansion step we have new sets of q-grams that can be merged with the corresponding sets from Algorithms 1 and 2. If we denote the sets of basic possible, not possible and assigned q-grams from Algorithms 1 and 2 with \( C^p, C^n \) and \( C^d \), and the refined and expanded q-gram sets generated by Algorithm 3
with $C^r_p$, $C^N_p$, and $C^N_i$, then for each bit position $1 \leq p \leq l$ the merged q-gram sets are generated as:

Not possible: $C^N_m[p] = C^N_m[p] \cup C^N_i[p]$
Possible: $P_m[p] = (C^r_m[p] \cup C^N_i[p]) \setminus C^N_m[p]$
Assigned: $A_m[p] = (C^r_m[p] \cup C^N_i[p]) \setminus C^N_m[p]$

For each bit position, the merging first takes the union of the not possible q-gram sets, followed by the union of the possible q-gram sets minus the just merged not possible q-gram sets. This ensures any q-gram that was identified to be not possible at a certain position is not in the set of possible q-grams at that position. The sets of assigned q-grams are merged similarly, where no q-gram in an assigned set for a position can occur in the not possible set at that position.

Based on these merged sets, the final step of our attack is to re-identify attribute values, as we describe next.

### 4.4 Attribute Value Re-identification

Given sets of possible, $C^P$, and not possible, $C^N$, q-grams at BF bit positions (as generated by Algorithms 1 to 3), we can now try to re-identify attribute values from a set of plain-text values $V$ and a set of BFs $B$ (the sets used in Algorithms 1 and 3). Our aim is to re-identify attribute values with high precision, i.e. values where we are certain they must have been encoded into a given BF. Note that we currently do not exploit the assigned q-grams in the sets $C^A$ because these are subsets of the possible sets ($A^C \subset B^C$).

In the following subsections we present two re-identification approaches. The first one is our recently presented cryptanalysis attack method [14] that only exploits the sets of possible q-grams, $C^P$. Due to the low precision of the q-gram sets in $C^P$, as shown in our experimental evaluation in Section 6, for many BFs more than one attribute value is re-identified resulting in low re-identification accuracy. In Section 4.4.2 we then propose a new approach that exploits the sets of not possible q-grams, $C^N$, which are of high precision, leading to improved re-identification accuracy.

A requirement of the candidate attribute values in $V$ that we can re-identify is that at least one of their q-grams must occur in either of the sets $C^P$ or $C^N$, respectively. Otherwise there would be no information about possible and not possible BF bit positions that could be used in the re-identification process. In our experiments we found that the majority of attribute values in a domain contain at least one q-gram that occurs in one of the sets $C^P$ or $C^N$.

To generate the set of candidate attribute values which we aim to re-identify (i.e. guess), named $G^P \subset V$ (possible) and $G^N \subset V$ (not possible), we first identify all q-grams in the possible and not possible sets as $Q^P = \{q_j \in C^P[p] : 1 \leq p \leq l\}$ and $Q^N = \{q_j \in C^N[p] : 1 \leq p \leq l\}$. Based on those two sets of q-grams we can extract the possible, $G^P$, and not possible, $G^N$, candidate attribute values we aim to re-identify as $G^P = \{v_i \in V : \exists q_j \in genQGramSet(v_i, q) : q_j \in Q^P\}$ and $G^N = \{v_i \in V : \exists q_j \in genQGramSet(v_i, q) : q_j \in Q^N\}$.

#### 4.4.1 Re-identification using Possible Q-gram Sets

The re-identification step based on the possible q-gram sets, $C^P$, is detailed in Algorithm 4 and illustrated in Figure 5. The idea is that for a 1-bit to occur at a certain position in a BF, at least one q-gram of an attribute value must have been hashed to that position (see Theorem 1). Therefore, for a given candidate attribute value $q_j \in G^P$, and BF $b_i \in B$, if the set of possible q-grams at position $p$, where $b_i[p] = 1$, does not include any q-gram of $q_j$, then $q_j$ could not have been encoded in $b_i$.

The inputs to Algorithm 4 are the same set of BFs, $B$, used in Algorithms 1 and 3, the set of candidate attribute values we aim to re-identify (guess), $G^P$ (generated as described above), the list of possible q-gram sets at bit positions, $C^P$ (as generated in Algorithms 1 or 3), and the BF length, $l$. The algorithm returns a set of re-identified attribute value(s), $G_i \subset G^P$, for each BF $b_i \in B$, collected in the result list $R$.

After initializing $R$, from line 2 onwards Algorithm 4 loops over all BFs $b_i \in B$, and for each $b_i$ it initializes the set of possible candidate attribute values $G_i$ (that potentially have been hashed into this BF) as all values in $G^P$ in line 3. We then loop over all BF bit positions $p$ in line 4, and for any position that has a 1-bit ($b_i[p] = 1$) we retrieve the set of possible q-grams $q_{ij} = C^P[p]$ at that position in line 6.

We now check for each value $q_j \in G_i$, if at least one of the q-grams in $q_{ij}$ occurs in $g_j$ (lines 7 and 8). If a $g_j$ does not contain any q-gram from $q_{ij}$, then it could not have generated the 1-bit at position $p$ in $b_i$, and so in line 9 we
Algorithm 4: Re-identification based on possible q-gram sets

Input:
- B: Set of BFs from the sensitive database (same as in Algo. 1)
- G^P: Set of candidate attribute values to be re-identified
- C^P: List of possible q-gram sets at BF position (from Algo. 1 or 3)
- I: Length of BFs
Output:
- R: List of possible attribute values re-identified for each BF in B

1: \( R = [] \)
2: for \( b_i \) \( \in \) B do:
3: \( G^i = G^P \) // Initialize set of candidate attribute values
4: for \( 1 \leq l \leq I \) do: // Loop over BF bit positions
5: if \( (b_i[p] == 1) \) then:
6: \( q_p = G^P[p] \) // Set of possible q-grams at this position
7: for \( j \in G_i \) do: // Check all candidate attribute values
8: if \( (\forall q \in q_p : q \not\subseteq g_j) \) then: // At least one q-gram must occur
9: \( G_i = G_i \setminus \{g_j\} \) // Value cannot have been hashed into \( b_i \)
10: \( R,append(G_i) \) // Append possible attribute values for BF \( b_i \)
11: return \( R \)

Algorithm 5: Re-identification based on possible q-gram sets

Input:
- B: Set of BFs from the sensitive database (same as in Algo. 1)
- G^N: Set of candidate attribute values to be re-identified
- C^N: List of not possible q-gram sets at BF pos. (from Algo. 1 or 3)
- I: Length of BFs
- q: Length of substrings to extract from attribute values
Output:
- R: List of possible attribute values re-identified for each BF in B

1: \( R = [] \)
2: for \( b_i \) \( \in \) B do:
3: \( B^Q = \{\} \), \( B^C = \{\} \) // A BF with only 0-bits
4: \( Q^i = \{q_j \in C^N[p] : 1 \leq l \leq I \} \) // All q-grams that occur in \( C^N \)
5: \( B^{Q}[q] = b_i \forall q_j \in Q^i \) // A 0-bit BF for each q-gram in \( Q^i \)
6: for \( 1 \leq l \leq I \) do: // Loop over BF bit positions
7: if \( q_j \in C^N[p] \) do:
8: \( B^{Q}[q] = b_i \) // Set positions to 1 where q-gram is not possible
9: for \( g_j \in G^N \) do:
10: \( q_j = genQGramSet(g_j, q) \) // Convert attr value into q-grams
11: if \( (\forall q_k : q_k \not\subseteq g_j) \) then: // All q-grams known
12: \( B^{Q}[q] = b_i \cup B^{Q}[q_j] \) \( q_k \in q \) // Attribute value BF
13: for \( b_i \in B \) do:
14: \( G_i = \{\} \) // Initialize set of candidate attribute values
15: for \( g_j \in G^N \) do:
16: if \( (\forall q_k : q_k \not\subseteq g_j) \) then: // All 0-bit are 0
17: \( G_i = G_i \cup \{g_j\} \) // A re-identified attribute value
18: \( R,append(G_i) \) // Append possible attribute values for BF \( b_i \)
18: return \( R \)

4.4.2 Re-identification using Not Possible Q-gram Sets

As illustrated in Figure 3, for each q-gram there are bit positions where this q-gram cannot have been hashed into based on the sets in \( C^N \). We combine these not possible positions for an attribute value \( g_j \in G^N \) by taking the intersection of the sets of positions for all q-grams in \( g_j \). This results in a BF \( b_g \) which has 1-bits in all positions where a BF that potentially can encode \( g_j \) needs to have a 0-bit. A conjunction of \( b_g \) with any \( b_i \in B \) must therefore result in a BF that only has 0-bits for \( b_i \) to be able to encode \( g_j \).

Algorithm 5 details the steps of this approach, where the inputs are similar to Algorithm 4, except the not possible q-gram sets, \( C^N \), and a corresponding set of candidate attribute values, \( G^N \), are provided. As the algorithm requires q-grams to be generated, the length \( q \) of these q-grams is also needed. The algorithm returns the list \( R \) for each BF \( b_i \in B \) a set of re-identified attribute values, \( G_i \subset G^N \).

In lines 1 and 2, Algorithm 5 initializes the result list, \( R \), the sets of BFs for q-grams, \( B^Q \), and candidate values, \( B^C \), as well as a BF \( b_0 \) of length \( l \) that only contains 0-bits. Next, in line 3, all unique q-grams in \( C^N \) are collected into the set \( Q^N \), and for each q-gram \( q_j \in Q^N \) one BF that only contains 0-bits is created in line 4, collected into the set \( B^Q \).
In the first phase of the algorithm (lines 5 to 7), for each bit position \( p \) and for each q-gram \( g_j \in C^N[p] \) at that position, we set the bit at position \( p \) in the q-gram's corresponding BF \( B^q[g_j] \) to 1, leading to one BF per q-gram with 1-bits in all positions where this q-gram cannot be hashed to.

In the second phase, in lines 8 to 11, for each candidate attribute value \( g_j \in G^N \) that has all its q-grams in the set \( Q^N \) (of q-grams where we know their not possible positions, the test in line 10), we generate a BF for value \( g_j \) in line 11 as the conjunction (bit-wise AND, \( \otimes \)) of the BFs in \( B^q \) for all of the value's q-grams, \( g_k \in q_\varphi_k \), as generated in line 9. We collect all candidate attribute value BFs in the set \( B^G \).

In lines 12 to 17, we loop over all BFs \( b_i \in B \) and for each in line 13 we initialize an empty set of candidate values. In line 14 we then loop over all candidate attribute values \( g_j \in G^N \), and if the conjunction between the BF \( b_i \) and the candidate's BF \( B^G[g_j] \) results in a BF that contains only 0-bits then \( g_j \) could have been encoded into BF \( b_i \) and therefore in line 16 we add \( g_j \) to the set \( G_i \). We then add the candidate set for \( b_i \) to the result list \( R \) in line 17.

5 Complexity Analysis and Limitations

We now discuss the complexity and limitations of our efficient cryptanalysis attack on BFs.

Complexity: To simplify our analysis, we assume \( n = |B| = |V| \) is the number of BFs and attribute values, with \( n_Q \) the average number of q-grams per \( v \in V \), \( k \) the number of hash functions used, and \( l \) the BF length.

In Algorithm 1, the initialization in lines 2 and 3 is of \( O(n) \), sorting in line 4 is of \( O(n \log n) \), and generating \( A \) is \( O(nA) \) assuming \( n_A = |A| = \min(\{|B|_F, |V_F|}) \). The complexity of lines 7 to 12 is \( O(nA nQ l) \) as a \( v_i \) in each \( (v_i, b_i) \in A \) is converted into a q-gram set and all \( l \) bit positions are checked for the corresponding \( b_i \). In lines 14 to 16, assuming \( Q \) is the set of all unique q-grams generated from all values \( v_i \in V \), then on average \( k/l |Q| \) q-grams are hashed into one bit position, leading to a complexity of \( O(l) \) for one set difference and \( O(l l) \) for all \( l \) bit positions.

Algorithm 2 loops over all \( (b_i, v_i) \in A \), converts each \( v_i \) into its q-gram set and then iterates over all \( l \) BF bit positions where a set intersection is conducted for each 1-bit. Assuming that each \( C^P[p] \) contains \( t \) q-grams then the complexity of Algorithm 2 is \( O(nA(nQ + l l)) \).

The initialization in lines 2 and 3 in Algorithm 3 is \( O(nA + n) \) as it requires loops over \( A \), and \( V \) and \( B \), respectively. The first loop in lines 4 to 9 has a complexity of \( O(nA n(nQ + l)) \) because for each \( (v_i, b_i) \in A \) q-gram sets are generated and compared with q-gram sets from all \( v_j \in V_N \) and bit-wise AND operations are performed between BFs. The refinement and expansion phases in lines 18 to 42 have a complexity of \( O(nA(m nQ + l)) \) as they require the union and intersection of small numbers \( m \) of q-gram sets as well as bit-wise AND and OR of BFs of length \( l \) for each tuple \( (v_i, b_i) \in A \).

Algorithm 4 is of \( O(n l t(G^P[p])) \) as it loops over all BFs of length \( l \) in \( B \), and for each checks if any of the candidate values in \( G^P \) can be encoded into a certain BF, where we assume \( t \) is the average number of q-grams in a \( C^P[p] \).

Finally, in Algorithm 5, the initialization in lines 3 and 4, and the loop in lines 5 to 7, are of \( O(l u) \), where \( u = |Q| (1 – k/l) \) is the average number of q-grams that are not hashed to a BF bit position, as they iterate over all \( l \) bit positions and sets of not possible q-grams in \( Q^N \). The second loop (lines 8 to 11) iterates over all candidate values \( g_j \in G^N \) and for each generates a BF (line 11). As we assume an attribute value \( g_j \) in average contains \( n_Q \) q-grams then this loop is of \( O(|G^N| n_Q) \). The loop in lines 12 to 17 is of \( O(n|G^N|) \) as it iterates over all BFs in \( B \) and for each checks if any value \( g_j \in G^N \) could possibly have been encoded in the BF.

Limitations: Our attack method on BFs first assumes an attacker has access to a population database from where the set \( V \) of plain-text attribute values and their frequencies can be extracted. Second, we assume that the BF database \( B \) contains a subset of BFs that occur multiple times, and that the frequency distribution of BFs in \( B \) is similar to the frequency distribution of a single or a subset of attribute values in \( V \). These two assumptions are common to earlier published attacks on BFs [14, 15, 16, 17, 18].

6 Experimental Evaluation

We evaluated our attack method with several data sets, as we describe next together with the experimental setup used. We then discuss the results obtained, and finally provide recommendations on how to securely employ BFs for PPRL.

6.1 Data Sets and Experimental Setup

We conducted experiments using three data sets from two countries, as summarized in Table 2. The first is a pair of North Carolina Voter Registration (named ‘NCVR’) data sets (from: [http://alt.ncsbe.gov/data/) collected in June 2014 (the database to be attacked) and October 2016 (the plain-text database). We extracted two subsets containing 224,073 (2014) and 224,061 (2016) records, respectively. The second pair of data sets (named ‘UKCD’) are census records collected from the years 1851 (the sensitive database) and 1861 (the plain-text database) for the town of Rawtenstall in England. These data sets contain 17,034 (1851) and 22,430 (1861) records. For both data sets we use the First name, Surname, and City (NCVR) or Address (UKCD) attributes, respectively, as well as the concatenation of pairs of these attributes. A third data set was a voter database from Michigan (from: [http://michigangovoters.info]) containing 7,408,330 records that we used as the plain-text database available to an attacker in a cross-state attack experiment.

Following earlier attacks on BFs [14, 15, 16], we set encoding parameters as q-gram length \( q = [2, 3, 4] \), BF length \( l = [500, 1000, 2000] \), and either the double or random hashing methods [25] as described in Section 5. We set the

<table>
<thead>
<tr>
<th>Data set</th>
<th>First name</th>
<th>Surname</th>
<th>City / Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCVR</td>
<td>22,117 / 22,282</td>
<td>40,113 / 40,513</td>
<td>735 / 745</td>
</tr>
<tr>
<td>UKCD</td>
<td>624 / 813</td>
<td>994 / 1,638</td>
<td>414 / 529</td>
</tr>
<tr>
<td>Michigan</td>
<td>195,431</td>
<td>362,907</td>
<td>–</td>
</tr>
</tbody>
</table>

TABLE 2 The number of unique values in the attributes used in the experimental evaluation. For NCVR and UKCD we used two files, where one was the sensitive database encoded into BFs and the other the plain-text database available to the attacker, while in a cross-state experiment NCVR was the encoded and Michigan the plain-text database.
number of hash functions $k = [20, 30]$ and also calculated an optimal number (opt) for $k$ based on $l$ and $q$ such that the false positive rate is minimized [22]. We also applied the BF hardening techniques balancing and XOR-folding [23] as described in Section 3. We set the maximum size of attribute value and BF sets in Algorithm 3 as $m = [2, 5, 10]$ as these values provided good results in a set of initial experiments, and the number of candidate attribute values to re-identify (as discussed in Section 4.4) to $|G^P| = |G^N| = [100, 1000]$. We implemented our attack method in Python 2.7 and conducted experiments on a server with 64-bit 2.4 GHz CPUs, 128 GBytes of memory and running Ubuntu 14.04. The programs are available from the authors.

We present the precision of the possible ($C^P$), assigned ($C^A$), and not possible ($C^N$) $q$-gram sets over all BF bit positions, where for each position we calculate precision as the ratio of $q$-grams in a set in $C^P$, $C^A$ or $C^N$ that were truly hashed (for $C^P$ and $C^A$) or not hashed (for $C^N$) to a position. We do not report recall since each q-gram is hashed $k$ times, and even if not all positions of a q-gram are found (low recall), those identified can provide valuable information to allow re-identification of attribute values.

We evaluate the quality of the re-identification methods from Section 4.4 as re-identification accuracy by counting the number of BFs in the encoded database for which we were able to (1) correctly re-identify their attribute value (labeled 'Correct 1'), (2) correctly re-identify their attribute value but at least one other (incorrect) value was also re-identified (labeled 'Correct many'), or (3) one or several wrong attribute value(s) were re-identified (labeled 'Wrong').

6.2 Results and Discussion

In Figure 7, we show the precision of the $q$-gram sets $C^P$, $C^A$, and $C^N$. As can be seen, the sets of not possible $q$-grams have the highest precision. Even when BF hardening techniques are applied, our attack can correctly identify the positions where $q$-grams could not have been hashed to. This provides high quality information about $q$-gram assignments to bit positions that can be used in the re-identification process. The sets of possible $q$-grams are of lower precision because for many BF bit positions they also contain $q$-grams that in fact were not hashed to those positions. Using the possible sets therefore leads to more multiple re-identified attribute values, as Figure 9 shows.

The sets of assigned $q$-grams are of reasonable precision for certain attributes (but not all) and only when no hardening technique has been applied. One reason that some of these assigned sets have low precision is that any error in the frequency alignment of BFs and attribute values in the list $A$, as described in Section 4.1, leads to wrong assignments. From Figure 7, it is clear that the best way to re-identify attribute values is to use the not possible $q$-gram sets, $C^N$.

Figure 8 shows how different parameters affect the precision of the $q$-gram sets in $C^P$, $C^A$, and $C^N$. Note that...
only the parameter $m$ (maximum size of attribute value and BF sets as used in Algorithm 3) is controllable by the attacker. As can be seen, our attack is robust with regard to most used parameter settings, except for the precision of the assigned q-gram sets, $C^A$, which drops for an optimal number of hash functions, $k = \text{opt}$ and q-grams of length $q = 3$. This indicates that the approach to identify assigned q-grams is data and parameter specific and might not be useful to accurately re-identify encoded values.

The values for $m$ were selected based on the objective of our attack to identify q-grams with high precision. With an increase in $m$ the refinement and expansion steps in Algorithm 3 will lead to lower quality of the identified q-grams and thus lower final re-identification quality. Overall, as can be seen from Figure 8, the sets of not possible q-grams, $C^N$, are of much higher precision compared to the possible and assigned q-grams, $C^P$ and $C^A$, supporting that $C^N$ should be used to re-identify attribute values.

As can be seen in Figure 9 for the larger NCVR data sets the re-identification quality is higher than for the smaller UKCD data sets. This is because a larger number of frequent BFs and attribute values are available that can be correctly aligned in the list $A$. Our attack will be more accurate on larger BF databases as they can be aligned to large population databases such as telephone directories.

An example where our attack fails (as would others) is on addresses for the UKCD data set, where the alignment in the list $A$ is incorrect even for the most frequent address (changing from ‘Newchurch’ in 1851 to ‘Crawshawbooth’ in 1861). This highlights, as observed earlier [17], that an attacker needs to have access to a plain-text database that has a similar frequency distribution as the BF database.

The new re-identification method based on the not possible q-gram sets is able to correctly re-identify single attribute values for more BFs compared to our previous re-identification method based on the possible q-grams sets [14]. As Table 3 shows, our new method is also more precise as the average number of ‘Correct many’ re-identifications is much smaller compared to our previous method. Furthermore, the new re-identification method is also faster and even on the larger NCVR data sets the attack only requires a few seconds, as shown in Table 4.

With regard to the refinement and expansion of q-gram sets (as proposed in Section 4.3), Table 4 shows that for the re-identification method based on possible q-grams the proposed methods generally increase the number of correct single re-identifications while at the same time they reduce the number of multiple re-identifications. For the re-identification method based on not possible q-grams the results are more mixed, with some increases in the number of single re-identifications and large decreases in the number of multiple re-identifications. At least for the first name attribute there is also an increase in the number of wrong re-identifications. We plan to further investigate the performance of the refinement and expansion methods.

To evaluate our attack in a scenario where an attacker does not have access to a plain-text data set that has similar frequency characteristics to the encoded BF database (a more likely scenario in a practical attack given the availability of large population databases in various domains [17]), in Figure 10 we show the precision of q-gram sets using NCVR as the encoded BF database and a voter data set from Michigan as the plain-text database. These two databases are very different with regard to their sizes, and the most frequent first name and surname values are ranked differently. For example, the three most frequent surnames in NCVR are:
Table 3
Average number of multiple re-identified attribute values using either the re-identification approach based on possible or not possible q-gram sets for the NCVR data sets with the number of candidate attribute values, $G^F$ or $G^N$, set to 100 or 1,000, respectively. Results are averaged (with standard deviations shown) over various parameter settings. The method based on not possible q-gram sets leads to much smaller numbers of re-identified attribute values, making the attack much more accurate. Similar results were obtained with the UKCD data sets (not shown due to limited space).

<table>
<thead>
<tr>
<th>Re-identify method / Size G</th>
<th>First name</th>
<th>Surname</th>
<th>City</th>
<th>First name+Surname</th>
<th>First name+City</th>
<th>Surname+City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible, $</td>
<td>G^F</td>
<td>= 100$</td>
<td>10.38 ± 0.9</td>
<td>5.97 ± 2.1</td>
<td>8.19 ± 3.2</td>
<td>18.44 ± 1.5</td>
</tr>
<tr>
<td>Not Possible, $</td>
<td>G^F</td>
<td>= 100$</td>
<td>2.13 ± 0.2</td>
<td>2.19 ± 0.1</td>
<td>1.45 ± 1.0</td>
<td>3.37 ± 0.6</td>
</tr>
<tr>
<td>Possible, $</td>
<td>G^N</td>
<td>= 1,000$</td>
<td>93.84 ± 11.4</td>
<td>43.50 ± 20.6</td>
<td>56.83 ± 20.0</td>
<td>168.12 ± 24.9</td>
</tr>
<tr>
<td>Not Possible, $</td>
<td>G^N</td>
<td>= 1,000$</td>
<td>5.94 ± 0.9</td>
<td>3.61 ± 0.5</td>
<td>3.66 ± 0.4</td>
<td>3.83 ± 0.6</td>
</tr>
</tbody>
</table>

Table 4
Average improvements in the number of re-identified attribute values for the NCVR data sets using the refinement and expansion methods. Smaller improvements were obtained for the UKCD data sets because these are much smaller and contain less values that could match the refinement or expansion requirements described in Section 4.3.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Possible</th>
<th>Not possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corr 1 / Corr M / Wrong</td>
<td>Corr 1 / Corr M / Wrong</td>
<td></td>
</tr>
<tr>
<td>First name</td>
<td>+94.6% / -21.6% / +15.6%</td>
<td>-1.6% / -23.6% / +100.0%</td>
</tr>
<tr>
<td>Surname</td>
<td>+55.9% / -7.1% / -1.2%</td>
<td>+13.3% / -46.2% / -0.2%</td>
</tr>
<tr>
<td>City</td>
<td>+56.0% / -2.5% / -0.2%</td>
<td>+3.6% / -6.9% / -3.7%</td>
</tr>
</tbody>
</table>

Table 5
Comparison of re-identification results with existing BF cryptanalysis attack methods with regard to re-identification quality and run time.

<table>
<thead>
<tr>
<th>Publication</th>
<th>Data set and attr.</th>
<th>Num BF</th>
<th>Corr 1</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kuzu'11 [16]</td>
<td>NCVR first names</td>
<td>3,500</td>
<td>400</td>
<td>1,000 sec</td>
</tr>
<tr>
<td>Kuzu'13 [17]</td>
<td>Patient names</td>
<td>20</td>
<td>4</td>
<td>few sec</td>
</tr>
<tr>
<td>Niedermeyer [18]</td>
<td>German surnames</td>
<td>7,580</td>
<td>934</td>
<td>&gt; week</td>
</tr>
<tr>
<td>Kroll [15]</td>
<td>German surnames and town names</td>
<td>100K</td>
<td>44K</td>
<td>&gt; days</td>
</tr>
<tr>
<td>Christen [14]</td>
<td>NCVR first names</td>
<td>100</td>
<td>10</td>
<td>0.75 sec</td>
</tr>
<tr>
<td>Our approach</td>
<td>NCVR diff. attr.</td>
<td>&gt;200K</td>
<td>&gt;49K</td>
<td>&lt; 10 min</td>
</tr>
</tbody>
</table>

Smith (1.40%), Williams (1.05%), and Jones (0.94%), while in Michigan they are Smith (0.98%), Johnson (0.75%), and Williams (0.61%). These different rankings lead to wrong alignments of plain-text values and BFs in the list A in Algorithm 1 that results in mostly wrong identification of possible and assigned q-grams (as Figure 10 shows) and thus low re-identification accuracy of attribute values. However, the not possible q-gram sets identified are still highly precise because of the commonality of q-grams in these frequent attribute values. As a result, some information about q-grams encoded in the BF database can be learned.

Table 6
Comparison of re-identification results with existing BF cryptanalysis attack methods with regard to re-identification quality and run time.

<table>
<thead>
<tr>
<th>Publication</th>
<th>Data set and attr.</th>
<th>Num BF</th>
<th>Corr 1</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kuzu'11 [16]</td>
<td>NCVR first names</td>
<td>3,500</td>
<td>400</td>
<td>1,000 sec</td>
</tr>
<tr>
<td>Kuzu'13 [17]</td>
<td>Patient names</td>
<td>20</td>
<td>4</td>
<td>few sec</td>
</tr>
<tr>
<td>Niedermeyer [18]</td>
<td>German surnames</td>
<td>7,580</td>
<td>934</td>
<td>&gt; week</td>
</tr>
<tr>
<td>Kroll [15]</td>
<td>German surnames and town names</td>
<td>100K</td>
<td>44K</td>
<td>&gt; days</td>
</tr>
<tr>
<td>Christen [14]</td>
<td>NCVR first names</td>
<td>100</td>
<td>10</td>
<td>0.75 sec</td>
</tr>
<tr>
<td>Our approach</td>
<td>NCVR diff. attr.</td>
<td>&gt;200K</td>
<td>&gt;49K</td>
<td>&lt; 10 min</td>
</tr>
</tbody>
</table>

Therefore, database owners should ensure each BF in their encoded database is unique. This can be achieved by using record-level BF encoding [11], [19], where several attribute values are encoded into one BF. (2) Employ different hash mechanisms for each attribute to further reduce frequency information (so a q-gram will be encoded differently depending upon if it, for example, occurs in a first name or a surname). (3) Apply advanced BF hardening methods [23] to distort any useful patterns that could be exploited even further. Combined, these three recommendations will mean that there are unlikely frequent BFs or even frequent bit patterns that can be exploited by an attack method.

Although database owners need to be aware that there is no guarantee that any encoding is secure against any future attack methods that are based on sophisticated crypanalysis techniques [27], the proposed mitigation methods will make attacks even more harder. However, further extensive experiments are required to properly analyze the strengths and weaknesses of different BF encoding and hardening techniques and their potential vulnerabilities with regard to different attack methods.

7 Conclusions and Future Work
We have presented a novel efficient attack method on BFs that contain encoded sensitive attribute values intended for PPRL. Unlike earlier attacks on BFs for PPRL, our approach only requires an attacker to have access to a public database of attribute values and their frequencies, but no information about the BF encoding used. Our attack can be successful even when some BF hardening techniques have been applied. It is also much faster than earlier attacks, making it feasible for database owners to efficiently validate the security of their large encoded sensitive databases before they are being sent to other parties for conducting PPRL.
We believe our attack is an important component to make BF based PPRL more secure for practical applications.

As future work we aim to improve the refinement and expansion steps by using all possible q-gram subsets, including the assigned q-gram sets, \( C^A \). We also plan to investigate how probabilistic language models \[28\] can be employed to find the most likely candidate attribute value when several candidates have been re-identified for a certain BF. Finally, we plan to explore the risk of re-identification when other hardening techniques, such as Bloom-and-flIP (BLIP) or salting \[19\], have been applied.

**ACKNOWLEDGEMENTS**

The authors would like to thank the Isaac Newton Institute, Cambridge, for support and hospitality during the programme Data Linkage and Anonymisation (EPSRC grant EP/K032208/1). P. Christen was partially supported by the Simons Foundation. This work was also partially funded by the Australian Research Council under grant DP130101801.

**REFERENCES**


**TABLE 6**

<table>
<thead>
<tr>
<th>Re-identify method / Size G</th>
<th>First name</th>
<th>Surname</th>
<th>City</th>
<th>First name + Surname</th>
<th>First name + City</th>
<th>Surname + City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible, (G^2 = 100)</td>
<td>7.66 ± 1.7</td>
<td>12.11 ± 2.1</td>
<td>2.27 ± 1.1</td>
<td>30.29 ± 9.2</td>
<td>17.08 ± 5.0</td>
<td>156.31 ± 136.2</td>
</tr>
<tr>
<td>Not Possible, (G^2 = 100)</td>
<td>2.50 ± 1.1</td>
<td>3.97 ± 1.7</td>
<td>1.95 ± 1.0</td>
<td>7.09 ± 2.3</td>
<td>6.49 ± 2.4</td>
<td>5.71 ± 2.1</td>
</tr>
<tr>
<td>Possible, (G^1 = 1,000)</td>
<td>50.94 ± 17.9</td>
<td>78.92 ± 38.4</td>
<td>4.77 ± 1.9</td>
<td>293.09 ± 116.4</td>
<td>136.16 ± 49.4</td>
<td>1,374.93 ± 1247.6</td>
</tr>
<tr>
<td>Not Possible, (G^1 = 1,000)</td>
<td>2.79 ± 1.1</td>
<td>5.07 ± 2.6</td>
<td>1.98 ± 1.0</td>
<td>9.57 ± 3.5</td>
<td>7.71 ± 3.1</td>
<td>6.38 ± 2.1</td>
</tr>
</tbody>
</table>


**Peter Christen** is a Professor at the Australian National University (ANU) Research School of Computer Science (RSCS). He graduated with a PhD in Computer Science from the University of Basel, Switzerland, in 1999. His research interests are in record linkage and data mining, with a focus on privacy and machine learning aspects of record linkage. He has published over 130 articles in these areas, including in 2012 the monograph “Data Matching” published by Springer.

**Thilina Ranbaduge** is a PhD post-doctoral fellow at the ANU. His research interests are in data mining, and in multi-database and privacy-preserving record linkage. He received his PhD in Computer Science from the ANU in 2018 and completed his PG Dip and BSc (Hons) at the University of Moratuwa, Sri Lanka, in 2013 and 2009, respectively.

**Dinusha Vatsalan** is a Research Scientist at Data61-CSIRO, Australia, and an Honorary Lecturer in the ANU RSCS. She completed her PhD in Computer Science at the ANU in 2014 and her BSc (Hons) at the University of Colombo, Sri Lanka, in 2009. Her research interests are in privacy in data matching and mining, privacy in social media, privacy risk evaluation and prediction, and health informatics.

**Rainer Schnell** is Professor at the University of Duisburg-Essen and holds the Chair in Research Methodology in the Social Sciences. He graduated with a postdoctoral degree (habilitation) in Research Methodology at the University of Mannheim in 1996. He is a Survey Methodologist with a research focus on nonsampling errors, applied sampling, census operations, and privacy preserving record linkage.