Journal of Privacy and Confidentiality (2014) **6**, Number 2, 59–79

Cryptanalysis of Basic Bloom Filters Used for Privacy Preserving Record Linkage

Frank Niedermeyer*, Simone Steinmetzer†, Martin Kroll‡, and Rainer Schnell§

1 Introduction

Linking databases containing information on individual characteristics and behavior is of increasing scientific and commercial interest. In many applications, linking databases has to be done without a unique personal number. Hence, due to privacy concerns, privacy preserving record linkage (PPRL) is used most often. In this context encrypted personal quasi-identifiers such as first names, surnames, as well as date and place of birth are used. These quasi-identifiers are not unique personal identifiers themselves, but personal characteristics, so that their combination could be used to identify identical records in different databases. However, quasi-identifiers usually contain errors; therefore, linking only the subset of records with exact matching identifiers is usually unacceptable. To allow for errors in encrypted identifiers, special techniques such as encrypted phonetic codes are commonly used. For example, many European cancer registries outside of the Scandinavian countries use encrypted Soundex codes as substitutes for cases with nonmatching exact identifiers. Because the information retrieval properties of encrypted phonetic codes are far from perfect, a number of privacy preserving record linkage techniques have been suggested in the last 10 years. An overview of PPRL techniques is given in [22]. One of the most widely used techniques is based on Bloom filters [19].

Our contribution In this paper, we demonstrate a manual attack on a database of surnames encrypted through Bloom filters. The surnames used are considered to be a random sample of a specific population, but in contrast to the assumptions in previous literature [11], we do not suppose that the attacker has access to a population data set from which this random sample is drawn. In addition to the attacked database the adversary knows only a publicly available list of the most common surnames. Thus, we use two completely different data sets for the encryption and the attack. Furthermore, the attack described in this paper uses an entirely different approach than the attack presented in [11]. Instead of considering whole names, the attack is based on subtle filtering of bigrams and analyzing which bigrams could appear in a particular Bloom filter. Thus, we conduct a full cryptanalysis of the fundamental construction principle of basic Bloom filters as used in record linkage applications. In contrast to

*Federal Office for Information Security (BSI), Bonn, Germany mailto:frank.niedermeyer@bsi.bund.de .

†University of Duisburg-Essen, Germany mailto:simone.steinmetzer@uni-due.de.

‡University of Duisburg-Essen, Germany mailto:martin.kroll@uni-due.de.

§University of Duisburg-Essen, Germany mailto:rainer.schnell@uni-due.de.

© 2014 by the authors

the computationally intensive solution of a constraint satisfaction problem as described in [11] and [12], our manual cryptoanalytic attack uses considerably less computational effort. Finally, we are not only interested in the demonstration that some names could be identified correctly, but in recovering as many names as possible from the Bloom filters, limited only by the persistence of the attacker.

2 Background

Bloom filters were introduced by B. H. Bloom in 1970 [2] as an efficient data structure for checking set memberships.
A Bloom filter is defined as a bit array of length m, initially filled with zeros. To store a given set $S = \{s_1, \ldots, s_n\}$ of items in a Bloom filter, each element is mapped with k different hash functions $h_1, \ldots, h_k$, whereupon each hash function maps its input to an integer $h_j(s_i) := x_{i,j}$, where $0 \leq x_{i,j} \leq m-1$, $1 \leq i \leq n$, $1 \leq j \leq k$, and k, m, n positive integers. Afterwards, the bit position in the Bloom filter corresponding to $x_{i,j}$ is set to 1. A bit location can be set to 1 multiple times and it retains this value.

To check whether an item t is contained in S, it is mapped with the same k hash functions. Hence, if the bit positions $h_1(t), \ldots, h_k(t)$ in the Bloom filter are all set to 1, then t is presumably a member of S. Nevertheless, there is a probability for false positive results, which means the check indicates $t \in S$, although this is not the case. False positives occur when the ones on positions $h_1(t), \ldots, h_k(t)$ are caused by different s_i. However, if there exists at least one bit position that is set to 1 by (at least) one of the hash functions $h_1(t), \ldots, h_k(t)$, but for which the Bloom filter of S shows zero, t definitely does not belong to S.[1]

2.1 Application of Bloom Filters in Privacy Preserving Record Linkage

In 2009, the approach for conducting privacy preserving record linkage with Bloom filters was presented in [19]. In the suggested protocol, two data custodians A and B at first agree upon a password. Then they standardized the identifiers, padded them with blanks at the beginning and the end, and split them into substrings of two characters, called bigrams (digrams or 2-grams). Next, each bigram of an identifier is mapped through multiple password-dependent hash functions (keyed-hash message authentication codes (HMACs) [1], like keyed MD5 or SHA-1) to a Bloom filter. By computing the similarity of every two Bloom filters, that means evaluation of the number of coincident bit positions that are set to 1, the similarity of the encoded identifiers can be approximated via a similarity metric such as Jaccard index [6], Dice's [3], or Tanimoto's [18] coefficient. Hence, through calculation of the similarity between two encrypted records, record linkage based on Bloom filters allows for errors in the encrypted data. Because

[1] In the recent past, many variations of Bloom filters have been developed, which are used in a wide range of applications, such as spell checking, encrypted search, and database applications. A short overview is given in [15].

each identifier is handled with a separate Bloom filter, any standard record linkage software can be used if a function for computing similarities of binary vectors is available.

Bloom filter-based record linkage has been used in real-world medical applications, such as in Australia [16], Brazil [13], Germany [21], and Switzerland [10]. A recent study based on healthcare databases with more than 26 million records has shown that privacy preserving record linkage using Bloom filters can be done without difference in linkage quality compared with traditional probabilistic methods using fully unencrypted personal identifiers [16]. However, the widespread use of Bloom filters is hampered by the fact that currently no comprehensive cryptanalysis of Bloom filters in record linkage is available. For example, the British Office for National Statistics considered several methods for linking administrative data within the *Beyond 2011 Programme*, but refrained from all "(...) recent innovations, such as bloom filter encryption (...)" since they "(...) have not been fully explored from an accreditation perspective" [14]. We hope our contribution will be helpful for such accreditation processes.

2.2 Prior Attacks on Bloom Filters

Only two studies addressing attacks on Bloom filters in privacy preserving record linkage have been published so far. Kuzu et al. [11] sampled 20,000 records from the North Carolina voter registration list and encrypted the bigrams of forenames using 15 hash functions and Bloom filters with length equal to 500 bits. They formulated their attack as a constraint satisfaction problem (CSP), which defines a set of constraints on variables. To do this, the variables and their domains were determined by a frequency analysis of the identifiers from the voter registration list and of the Bloom filter encodings. To assign possible forenames to the Bloom filters, Kuzu et al. generated frequency intervals for the forenames in the voting list and for the Bloom filters. Furthermore, they filtered the number of possible bits that could be set for a name, to restrict the number of alternative names that could correspond to a Bloom filter. Thus, they showed under the assumptions that (1) the encrypted records are a random sample of a known resource and (2) the adversary has access to the resource from which the sample is drawn that Bloom filter encodings are vulnerable in principle. More precisely, the 400 most frequent names of approximately $3,500$ different forenames were assigned correctly, which corresponds to 11% of the database. However, these assumptions are rarely given in practical applications. Hence, in their second paper, Kuzu et al. [12] took a more critical look at the practical use of their own attack for real-world data sets, because the distribution of personal identifiers such as those from a medical database is unlikely to generate a random sample. Thus, they investigated the influence of different public resources (voter registration records) on the extent to which real personal identifiers (from the Vanderbilt University Medical Center) could be disclosed, when the latter are not a proper random sample of the first. After some modifications to the attack and restricting the problem to a set of 42 names, the CSP solver "(...) was unable to discover a solution after almost a week of continuous runtime." Restricting the problem to the set of only the 20 most frequent names, the CSP assigned four of those 20 names correctly in a few seconds. Therefore, Kuzu et al. [12] concluded "(...) that the attack

remains feasible in practice, but that it results in significantly lower precision in the set of predictions and longer computation time in the cryptanalysis than suggested in theory."

3 Encrypting Names with Bloom Filters

In this section we introduce some basic notation and describe the encrypting procedure. We start with a definition of the underlying alphabet and the encryption function.

Definition 3.1. Let $\Sigma := \{␣, A, B, \ldots, Z\}$ be an alphabet, where ␣ denotes the padding blank and $m, n \in \mathbb{N}$. We call an element $b \in \Sigma^n$ *n-gram* and *bigram* if $n = 2$. Further, we define a *hash function* as a mapping f from the set of bigrams into the set of nonnegative integers smaller than m:

$$f : \Sigma^2 \longrightarrow \{0, \ldots, m-1\}.$$

A hash function always maps the same bigram to the same integer value. Furthermore, if the hash function is unknown, e.g., if it is dependent on a cryptographic key, the bigram cannot be deduced from its hash value.

In [19] the hash values for k hash functions H_j were derived as proposed in [8] from a linear combination of the hash values of two hash functions g and h by the *double hashing scheme*

$$H_j(b) := g(b) + j \cdot h(b), \quad j \in \{0, \ldots, k-1\}. \tag{1}$$

To gain higher security against cryptographic attacks, g and h should depend on cryptographic keys K_0 and K_1, respectively, where both keys are random bit strings of sufficient length. Furthermore, g and h should be chosen such that it is computationally infeasible to determine the keys from the hash values, even if the bigrams from which the hash values have been calculated are known. This can be realized by an HMAC [1] associated with a strong cryptographic hash function, e.g., SHA-2, such that g and h are defined by

$$g(b) := \mathrm{HMAC}(K_0, b) \bmod m, \quad h(b) := \mathrm{HMAC}(K_1, b) \bmod m.$$

It should be noted, that the choice of g and h has no relevance for our attack. However, the linear combination of g and h is essential for the proposed attack.

Next we define the basic structure of our encryptions, *atoms*.

Definition 3.2. Let $m, n \in \mathbb{N}$ and $b \in \Sigma^2$ be a bigram. We define the *atom* realized by the bigram b as Bloom filter $\mathcal{B}(b)$ of length m given by

$$\mathcal{B}(b) := (p_0, \ldots, p_{m-1}) \in \{0,1\}^m,$$

where

$$p_i := 1 \Leftrightarrow \exists j \in \{0, \ldots, k-1\} : H_j(b) \equiv i \bmod m.$$

Thus, *atoms* denote Bloom filters that are generated from only one bigram. Furthermore, at most k positions in an atom are set to 1, because each bigram is subject to each hash function.

The Bloom filter of a name is the combination of the atoms corresponding to the bigrams of the name, by using the bitwise OR operation:

$$\mathcal{B}(\texttt{NAME}) = \bigvee_{b \in \{\texttt{␣N, NA, AM, ME, E␣}\}} \mathcal{B}(b),$$

where $\bigvee$ is the bitwise OR operation and NAME denotes a standardized name. As an example, we consider the most common German surname: MUELLER.

***Example* 3.1.** The resulting set of bigrams from the surname MUELLER is

$$\{\texttt{␣M, MU, UE, EL, LL, LE, ER, R␣}\}.$$

Hence, the bigram ER mapped with $k = 15$ hash functions could yield a vector as shown in Figure 1.

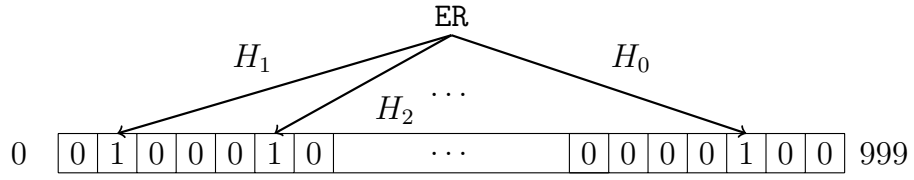

Fig. 1: *Atom*: Bloom filter of the bigram ER

Mapping each bigram with 15 hash functions results in eight atoms. Combining them with the bitwise OR operator yields the Bloom filter for MUELLER as illustrated in Figure 2.

	0000100 . . . 0000010	$\mathcal{B}(\texttt{␣M})$
$\vee$	0001000 . . . 0000100	$\mathcal{B}(\texttt{MU})$
$\vee$	0101010 . . . 1010101	$\mathcal{B}(\texttt{UE})$
$\vee$	0001000 . . . 1000010	$\mathcal{B}(\texttt{EL})$
$\vee$	1000000 . . . 1000000	$\mathcal{B}(\texttt{LL})$
$\vee$	0000100 . . . 0000010	$\mathcal{B}(\texttt{LE})$
$\vee$	0100010 . . . 0000100	$\mathcal{B}(\texttt{ER})$
$\vee$	0101010 . . . 0000001	$\mathcal{B}(\texttt{R␣})$
	1101110 . . . 1010111	$\mathcal{B}(\texttt{MUELLER})$

Fig. 2: Bloom filter of MUELLER, composed of atoms belonging to the underlying bigrams

Initial setting: We consider the common setup of a semi-trusted third party C that conducts the record linkage process between two Bloom filter encrypted databases. This semi-trusted third party does not collude with any of the two data custodians A and B, but tries to infer as much information as possible from the databases. In our example, a data set containing Bloom filters was assumed to be sent to such a third party and we further assume that the third party will try to decrypt the Bloom filters. The adversary has knowledge of the encryption process, but does not know the key.[2]

For the attack, we generated 10,000 Bloom filters built from German surnames with the frequency of each name approximately corresponding to the frequency of surnames in the German population.[3] (Before encryption, the names were standardized (removal of unusual characters and punctuation), as is common practice in record linkage [17].) Finally, the names were padded with blanks so that the alphabet was given by the letters A-Z and an additional blank. For the encryption we used Bloom filters of length $m = 1{,}000$, built from bigrams, which were encrypted with $k = 15$ hash functions.[4]

4 Cryptanalysis of the Bloom Filters

This section provides a detailed description of the deciphering process. At first we reduce the number of Bloom filters, then we investigate which atoms may be contained in each of the remaining Bloom filters. Finally, we assign bigrams to these atoms and reassemble the encrypted surnames from them.

Assumptions: The hash functions g and h are dependent on a cryptographic key, which is unknown to the attacker, but the attacker knows about the double hashing scheme. Our aim is to reconstruct the original surnames. Note that our deciphering process could also be applied in the case of alternative values for the length m of the Bloom filters and the number k of hash functions. Furthermore, it does not depend on the used hash functions themselves.

4.1 Sorting and Counting Bloom Filters

First, we sorted and deduplicated the given Bloom filters, resulting in 7,580 unique ones. The three most frequent Bloom filters occurred 67, 50, and 30 times. Altogether, 934 of the 10,000 Bloom filters existed at least twice. Only this subset was used for further analysis to prevent problems caused by rare names and rare bigrams. In the following, the most frequent 934 Bloom filters will be denoted by $\mathcal{B}_i$ ($i \in \{0, \ldots, 933\}$), where $\mathcal{B}_0$ is the most frequent one.

[2] If the key is not used or if it is known to the attacker, it is possible to encrypt (frequent) names with the known hash functions, to compare these encryptions to the database of Bloom filters and thus to find coincident names. If the used key is unknown to the attacker, the computational effort for systematically testing arbitrary keys becomes unacceptable.

[3] The database of the social security administration, which covers about 39% of the whole German population, was used for sampling names.

[4] The values for m and k resulted from tests based on those described in [19].

4.2 Enumerating Possible Atoms

Atoms are the results of linear combinations of two hash functions g and h. For a specific input, let x and y be the hash values of g and h, respectively. Then it holds that $x, y \in \{0, \ldots, 999\}$ and we define $\mathcal{B}(x, y) = (p_0, \ldots, p_{999})$ as before by

$$p_i := 1 \Leftrightarrow \exists j \in \{0, \ldots, 14\} : x + jy \equiv i \bmod 1000. \tag{2}$$

Hence, for each bigram b, there exist x and y with $\mathcal{B}(b) = \mathcal{B}(x, y)$, even though this representation does not need to be unique. For example, $\mathcal{B}(0, 250)$ and $\mathcal{B}(0, 750)$ both lead to atoms in which the positions $0, 250, 500$, and 750 are set to 1. Because we only knew the given Bloom filters of the surnames, but did not know the hash functions used, we generated every possible atom. As $x, y \in \{0, \ldots, 999\}$ holds, a pairwise combination of x and y resulted in a multiset $\mathcal{M}$ of $1{,}000^2 = 1{,}000{,}000$ possible atoms.

The next step was testing which of those atoms actually occurred in the 934 most frequent Bloom filters.

Definition 4.1. Let $m \in \mathbb{N}$. For a Bloom filter $\mathcal{B}(\cdot) = (p_0, \ldots, p_{m-1})$, we define

$$\mathcal{I}(\mathcal{B}(\cdot)) := \{0 \leq i \leq m - 1 : p_i = 1\}.$$

$\mathcal{I}$ denotes the set of indices corresponding to the positions set to one in $\mathcal{B}(\cdot)$. Then, for two Bloom filters $\mathcal{B}(\cdot)$ and $\mathcal{B}'(\cdot)$, we define a partial relation $\preccurlyeq$ by

$$\mathcal{B}(\cdot) \preccurlyeq \mathcal{B}'(\cdot) :\Longleftrightarrow \mathcal{I}(\mathcal{B}(\cdot)) \subseteq \mathcal{I}(\mathcal{B}'(\cdot)). \tag{3}$$

Testing the existence of a $\mathcal{B}_i$ with $\mathcal{B}(x, y) \preccurlyeq \mathcal{B}_i$ for each of the 1,000,000 possible atoms reduced the size of $\mathcal{M}$ to 3,952 possible atoms. The Bloom filter of each bigram contained in a name corresponding to one of the Bloom filters $\mathcal{B}_i$ is a possible atom. However, not every $\mathcal{B}(x, y)$ that is part of a given Bloom filter originates from a bigram of the corresponding name. For such an atom two cases are possible:

(a) The atom is not realized by a bigram at all;

(b) The atom is realized by a bigram that is not part of the considered name.

We refer to atoms satisfying condition (a) as *phantom atoms*. Because there are 26 letters and the padding blank in our alphabet, at most $27^2 = 729$ true atoms exist. Hence, most of the 3,952 possible atoms are phantom atoms. Let us consider an example.

***Example* 4.1.** Assume $\mathcal{B}(0, 100)$ is a true atom, then in the corresponding Bloom filter the positions $0, 100, 200, 300, 400, 500, 600, 700, 800$, and 900 are set to 1, as shown in Equation (2). For $\mathcal{B}(0, 200)$, the positions $0, 200, 400, 600$, and 800 would be set in the corresponding Bloom filter (Figure 3). Although $\mathcal{B}(0, 200)$ is a possible atom resulting from the relation $\mathcal{B}(0, 200) \preccurlyeq \mathcal{B}(0, 100)$, it could also be a phantom atom.

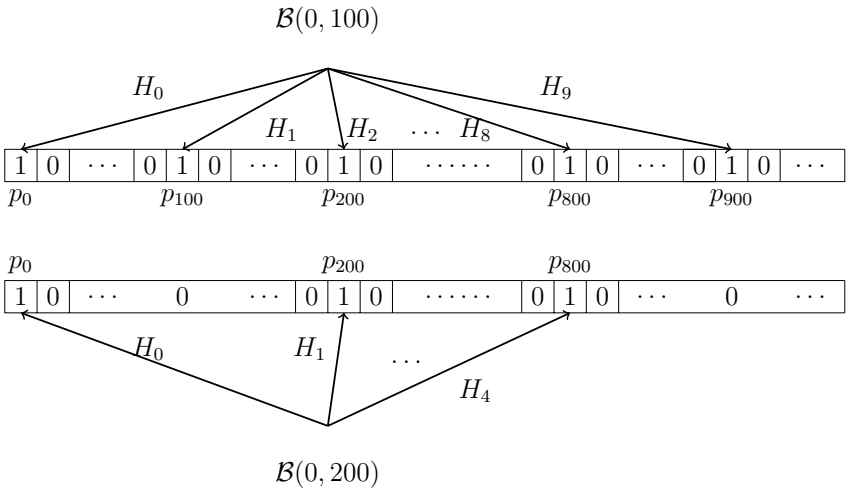

Fig. 3: Comparison of the atoms $\mathcal{B}(0,100)$ and $\mathcal{B}(0,200)$.

4.3 Filtering Phantom Atoms

We first introduce the Hamming weight of a Bloom filter, which simply corresponds to the number of ones in the Bloom filter.

Definition 4.2. For a Bloom filter $\mathcal{B}(\cdot) = (p_0, \ldots, p_{m-1})$ $(m \in \mathbb{N})$, the Hamming weight is defined by

$$\begin{aligned} HW : \{0,1\}^m &\longrightarrow \{0, \ldots, m\} \\ HW(\mathcal{B}(\cdot)) &\longmapsto \sum_{i=0}^{m-1} p_i. \end{aligned} \tag{4}$$

Example 4.1 motivates that the Hamming weight of many phantom atoms is less than the maximum possible weight k of an atom, which is equal to 15 in our case. Therefore, the number of possible atoms with Hamming weight 15 was determined and only these atoms were used for further analyses. This approach resulted in 805 possible atoms[5] and could implicate two problems concerning the filtering of true atoms and the remaining phantom atoms.

Problem 1: Real atoms could be removed erroneously from the set.

This case is unlikely, because a randomly generated atom has Hamming weight 15 with high probability. More precisely, in our specific case the probability of generating an

[5]In our first approach, atoms with Hamming weight $HW(\mathcal{B}(x,y)) < 4$ were removed, reducing the set of possible atoms to 839. The previously mentioned procedure includes this one, and thus is stronger.

atom with Hamming weight 15 is equal to $P = \frac{984}{1000} = 0.984$. Thus, the probability that an arbitrary (but fixed) real atom is filtered is equal to $1 - P = \frac{16}{1000} = 0.016$ (for details, see Lemma A.1 in Appendix A). Consequently, the expected number of erroneously filtered real atoms is bounded from above[6] by $27^2 \cdot \frac{16}{1000} \approx 11.7$.

Later assignments of atoms to bigrams showed that the most frequent atoms like $\mathcal{B}(\texttt{ER})$ or $\mathcal{B}(\texttt{CH})$ were indeed not filtered. However, we found that the atom of the bigram `HA` was filtered by the restriction. Although this is a frequent atom, its filtering had no negative influence on the deciphering process.

Problem 2: There could be further phantom atoms among the residual 805 atoms.

For example, assume $\mathcal{B}(0, 10)$ and $\mathcal{B}(30, 10)$ are real atoms and there exists an i for which $\mathcal{B}(0, 10) \preccurlyeq \mathcal{B}_i$ and $\mathcal{B}(30, 10) \preccurlyeq \mathcal{B}_i$. Then $\mathcal{B}(10, 10) \preccurlyeq \mathcal{B}_i$ and $\mathcal{B}(20, 10) \preccurlyeq \mathcal{B}_i$ holds, although $\mathcal{B}(10, 10)$ and $\mathcal{B}(20, 10)$ do not need to correspond to bigrams and could therefore be phantom atoms. If we suppose that all real atoms except for the maximal 11.7 erroneously filtered ones are contained in the remaining 805 possible atoms, we can estimate the number of remaining phantom atoms by $805 - 729 + 11.7 = 87.7$. Thus, at least 87.7 phantom atoms are expected to be contained in the 805 possible atoms.

Assignment of atoms to bigrams still showed some phantom atoms among the remaining 805 possible atoms.

4.4 Sorting Possible Atoms

For each of the remaining possible atoms $\mathcal{B}(x, y)$, we determined the number of Bloom filters $\mathcal{B}_i$, $i \in \{0, \ldots, 933\}$ containing the respective atom. The atoms were then numbered from a_0 to a_{804} according to their frequency, in which the most frequent candidate atom is indexed by zero. Initially, we assumed that the phantom atoms appear rarely and therefore were assigned to high numbers. It turned out that this is correct to a large extent. Nevertheless, some candidate atoms with low numbers (such as 52, 68, and 80) turned out to be phantom atoms.

4.5 Assignment of Atoms to Bigrams

At first, we identified the most frequent bigrams in different lists of German surnames. In all of the lists, the three most frequent bigrams were `ER`, `R␣`, and `CH` (because of padding of the surnames, we distinguished between beginning letters, i.e., `␣R`, and ending letters, i.e., `R␣`). Next we assigned these particular bigrams to the three most frequent candidate atoms: $a_0 \rightarrow \texttt{ER}$, $a_1 \rightarrow \texttt{R␣}$, and $a_2 \rightarrow \texttt{CH}$.

[6] Only atoms that actually appear in the encrypted names can be removed erroneously from the set. It is unlikely that rare bigrams, such as `QX` or `QZ`, occur in common surnames and in fact they did not appear in our data set as well. Hence, the number of erroneously filtered real atoms is even less than 11.7 in our case.

Because we knew that ␣SCH is a common German prefix combined from the bigrams ␣S, SC, and CH, we determined which candidate atoms generally occur together with atom number two for CH, and hence assigned these ones to ␣S and SC. We proceeded likewise with the frequent combination of bigrams in MANN. We then guessed names from the completed assignments of bigrams to possible atoms, looked up common bigrams from our frequency analysis, and thus found further mappings between frequent bigrams and atoms. Wikipedia listed as the most frequent German surnames the names MUELLER and SCHMIDT. The common bigrams in our first two Bloom filters indicated that these corresponded to MUELLER and SCHMIDT. The Wikipedia list also contained four different spellings of the name MEYER. Thus, we compared several spellings of common German surnames such as MEYER with respect to identical atoms and determined further assignments of candidate atoms to bigrams. A detailed description of this procedure is given in Appendix B.

However, the ranks of frequencies for our list of encrypted names did not correspond exactly to the frequency rank in the Wikipedia list of the most common German surnames. For example, we identified the name NGUYEN as Bloom filter $\mathcal{B}_{73}$, but in the list from Wikipedia it appears at position 815.

Note that these assignments strongly depended on our knowledge of German surnames as well as good guessing. To verify our assumptions, we used a statistical analysis of surnames, based on Hamming weights as described in the following section.

4.6 Statistical Analysis Based on Hamming Weights

Based on names from a German telephone CD, we extracted approximately 700,000 surnames as an additional verification for the correctness of our assignments. Because an adversary does not know the hash functions g and h that were used for the encryption procedure, we chose two arbitrary different hash functions $\tilde{g}$ and $\tilde{h}$, dissimilar from g and h, and computed the string lengths $L(\texttt{NAME})$ as well as the Hamming weight $HW(\mathcal{B}(\texttt{NAME}))$ of each corresponding Bloom filter. Then, for a fixed Hamming weight of some Bloom filter, we counted the string length l of the corresponding names and l showed only a small variance.

For example, there were 5,003 names for which the Hamming weight of the corresponding Bloom filter had a value of 150. The string length of these names varied between $l = 9$ and $l = 15$. The average was 10.17 with variance 0.21. Hence, we concluded that a name with Hamming weight 150 of the corresponding Bloom filter has a string length of $l = 10$.

We then computed the Hamming weights of our most frequent Bloom filters $\mathcal{B}_0, \ldots, \mathcal{B}_{933}$ and found that $\mathcal{B}_0$ and $\mathcal{B}_1$ had Hamming weights 111 and 113, respectively. Based on the mentioned statistics, these Hamming weights correspond to names with string length $l = 7$. Because the most frequent surnames MUELLER and SCHMIDT both consist of seven letters each, we substantiated our assumption and assigned $\mathcal{B}_0 = \mathcal{B}(\texttt{MUELLER})$ and $\mathcal{B}_1 = \mathcal{B}(\texttt{SCHMIDT})$.

As expected, the distribution of the Hamming weights is approximately the same for Bloom filters generated with the hash functions g and h as well as with $\tilde{g}$ and $\tilde{h}$. Overall, the Hamming weight is a good indicator for the length of a name encoded in a Bloom filter, because it directly reflects the length of the encrypted identifier (except for different bigrams that are partially hashed to the same bit positions). However, the Hamming weight depends on the number of different bigrams resulting from the respective identifier. For example, the length of the name `BERGER` could be estimated 5 instead of 6 due to the duplicate bigram `ER`, when padding is not used. However, most names do not contain bigrams that appear more than once and we used padding of the surnames. Thus, we could neglect this fact in our case.

5 Summary of the Attack

In this section we summarize our attack to give an overall impression of the executed steps.

The basis for our attack was a database of 10,000 German surnames with the frequency of each name approximately corresponding to the frequency of surnames in the German population. We first standardized the surnames and padded them with blanks. Then we encrypted each name through a Bloom filter with length equal to $m = 1{,}000$ bits by hashing it with $k = 15$ hash functions as described in Equation (1).

Next, we simulated the action an adversary would take. We assumed that the adversary knows about the encryption procedure with the double hashing scheme, but does not know the key that was entered into the hash functions. For that purpose, we first deduplicated the 10,000 Bloom filters, which resulted in 7,580 unique ones. We then determined the 934 Bloom filters that existed at least twice and used those ones for further analyses.

Since an attacker has to find out which bigrams are encrypted through each of the 934 Bloom filters, we generated Bloom filters with every possible combination of bit positions set to 1, resulting in at most[7] 1,000,000 possibilities. These Bloom filters simulated the possible encryptions of single bigrams and thus the candidate atoms.

Since there exist only 729 bigrams, the attacker has to determine which of the maximal 1,000,000 possible atoms truly are encrypted bigrams. Thus, we first matched the bit positions set to 1 in the possible atoms with the bit positions set to 1 in the Bloom filters, which resulted in 3,952 possible atoms. Since the phantom atoms that are not encrypted bigrams have Hamming weight less than 15 with high probability, or in other words possible atoms that are truly encrypted bigrams have Hamming weight 15 with high probability, we only used possible atoms with Hamming weight 15 for further analyses.

[7] As mentioned in Section 4.2 there are atoms such as $\mathcal{B}(0, 250)$ and $\mathcal{B}(0, 750)$ in which the same positions, in this example $0, 250, 500$, and 750, are set to 1 when using the double hashing scheme. Thus, there exist even less than 1,000,000 candidate atoms in this case.

The next steps consisted in first determining the number of Bloom filters $\mathcal{B}_i$, $i \in \{0, \ldots, 933\}$ containing the remaining 805 possible atoms and then numbering those atoms, as well as the 934 Bloom filters, according to their frequencies in descending order. Furthermore, we identified which of the 805 possible atoms are contained in which of the 934 Bloom filters. Finally, we generated the bigram frequencies from a publicly available list of German surnames and assigned the three most frequent bigrams `ER`, `R␣`, and `CH` to the most frequent possible atoms by hand. We displayed the Bloom filters with the included atoms as shown in Appendix B and followed the procedure described in Section 4.5 to find further assignments of possible atoms to bigrams until we had manually deciphered all 934 Bloom filters.

We want to emphasize that the attack was conducted manually, assisted by statistical calculations. For the generation of possible atoms and the filtering procedure the statistical programming language R was used and needed approximately one day. However, this task could be done faster with a compiled programming language such as C++. The main work, namely the assignment of candidate atoms to bigrams, was done by hand. For these assignments we needed several work days. This time would be approximately the same for other databases of the same size and would increase for larger data sets. As usual in cryptanalysis, the purpose of this paper is to show that the encryption can basically be broken. Thus, no algorithm that automatically deciphered the surnames was implemented, and this is left for future research. Furthermore, we conducted an attack on the bigrams as the building blocks of the Bloom filters, not on the surnames as a whole. Since we conducted a manual attack, there were only two possibilities: We could guess a name from the assigned bigrams or we could not. If we made a wrong atom to bigram assignment, we reset the mapping and compared Bloom filters until we found a correct assignment.

Note, that the attack does not have to depend on bigrams, it can also be conducted on Bloom filters built from n-grams. For example, the 3-grams `SCH` and the 4-grams `MANN` appear frequently in German surnames and thus provide a point of attack. However, since the linkage qualities with bigrams are better than with 3-grams or 4-grams, we concentrated on the more realistic scenario with bigrams.

Furthermore, the attack does not require a deduplicated list of names. In addition, as mentioned in [20], the success of a frequency attack depends on the ratio of the number of hash functions used to the length of the Bloom filter $\frac{k}{m}$. In this paper we considered the values $m = 1{,}000$ and $k = 15$. In the case of higher values for m and/or lower values for k, less bit positions will be set to 1 in a Bloom filter. This implies that during the filtering process less phantom atoms will be found, because there are less collisions of the hash values. Thus, the assignment of candidate atoms to bigrams will be much easier. On the contrary, higher values for k and/or lower values for m result in Bloom filters with more bit positions set to one. Thus, more phantom atoms will be found due to collisions of hash values and the assignment of candidate atoms to bigrams will be much more difficult. However, the more bit positions that are set to 1 in a Bloom filter, the worse the linkage properties and hence there is no real-world application for those Bloom filters. Thus, the attack proposed in this paper is at least feasible as long as the Bloom filters are built with realistic properties.

6 Discussion

In this paper we have demonstrated a successful attack on basic Bloom filters as used in privacy preserving record linkage applications. In contrast to previous research, very little computational effort is needed and only publicly available name frequency lists are used for this form of attack. The attack concentrates on the frequency of n-grams as the building blocks of Bloom filters instead of the frequency of entire names. In our example, after the assignment of possible atoms to bigrams for the most frequent 934 Bloom filters, we can consider the whole database and follow the procedure described in Section 4.5 to find further mappings between bigrams and candidate atoms. We can continue with this strategy until we have uncovered all bigrams, which are contained in the encrypted names and for which the corresponding real atoms were not filtered. With the knowledge of all or almost all of those bigrams, we thus can decipher every name and even unique names. Therefore, the attack described in this paper can be used for the deciphering of a whole database instead of only a small subset of the most frequent names, as in previous research and it is conceivable to develop a fully automatic deciphering algorithm based on the presented procedure.[8]

In summary, we must conclude that basic Bloom filters using only one identifier per Bloom filter are not suited for encrypting sensitive personal data and should not be used for applications requiring strong security guarantees.

However, the enumeration of possible atoms in our analysis (cf. Section 4.2) depends on the special composition of hash functions as described in Equation (1). We believe that modifying the construction scheme of the hash functions offers a higher level of security against the attack described in this paper. Altogether, several options for modifying basic Bloom filters are conceivable:

Independent Hash Functions. The double hashing scheme is based on k hash functions, which are linear combinations of two cryptographic hash functions (cf. Section 3). In contrast, an attacker's background knowledge will be minimal by giving up the double hashing scheme and using k independent hash functions instead. Thereby, the cardinality of the set of possible atoms increases. For example, for the parameters considered in our concrete example (m = 1,000, k = 15), the number of candidate atoms for the double hashing scheme is at most 1,000,000 $= 1 \cdot 10^6$ (see footnote 7), as described in Section 4.2. When permitting arbitrary hash values this number rises to approximately $\binom{1000}{15} = 6.9 \cdot 10^{32}$. Therefore, as a first modification to basic Bloom filters, we strongly suggest the use of independent hash functions for applications. In this case, applying the described attack is not promising, since filtering phantom atoms just by taking the atoms with maximal Hamming weight would result in too many remaining candidate atoms to assign to bigrams directly. To be more precise, an entirely new approach for the detection of possible atoms has to be invented.

[8] More details on the development of a fully automatic deciphering algorithm are given in [9].

Modification of identifiers. Further options for hardening Bloom filters are modifications of the identifiers, such as deletion or sampling of bigrams for long names. Shortening the identifiers will prevent the identification of longer names. On the one hand the bigram frequencies would change, so that applying this attack would imply that the attacker knows the deletion or sampling scheme to assign candidate atoms to bigrams appropriately. On the other hand it would be much more difficult for an adversary to guess the names from the remaining or sampled bigrams. Furthermore, omitting the padding of bigrams makes the identification of starting and ending letters much harder, because an attacker cannot differ between bigrams from the center and the beginning or the end of a name.

Salting. *Salting* describes the process of generating hash values that depend on a record-specific key. In record linkage scenarios, short identifiers such as date or year of birth are obvious natural candidates for such a key. Then, for a single bigram b appearing in two names, the same bit positions are set to 1 in the corresponding Bloom filters only if the keys coincide. If the keys are not equal, it is unlikely that all hash values for the bigram b are the same. Thus, the keys should not contain too many errors so that the Bloom filters remain error tolerant. However, applying the presented attack in this case is also not promising. The bigrams resulting from the name `MUELLER` would be hashed completely differently if the year of birth was used as the key and two persons with this name were for example born in 1959 and 1972. Thus, a new approach for the detection of possible atoms would have be developed.

Inserting random bits. Adding random bits to Bloom filters, as proposed by Schnell et al. [19], should have no serious effect on their similarities, but it will prevent any attack using deterministic constraints only. Furthermore, in an attack as described in this paper, fewer phantom atoms could be filtered, so that altogether more possible atoms appear in each of the most frequent Bloom filters. This makes the assignment of candidate atoms to bigrams much more difficult.

Using a single Bloom filter for all identifiers. The use of a single Bloom filter for storing all identifiers (*Cryptographic Longterm Key, CLK*) was first suggested in 2011, [20]. Because not only bigrams from first names and surnames, but also bigrams from numerical variables such as date of birth and additional identifiers like place of birth are hashed with different hash functions to the same bit array, it becomes more difficult for an attacker to detect repetitive patterns, even in sets of Bloom filters. By increasing the probability that the same pattern of positions in a Bloom filter could be set by different bigrams from different identifiers (this property is not true for Record-level Bloom filters proposed by Durham et al. [4]), attacking a single Bloom filter of the kind used for CLKs will further impede CSP attacks as described in [11]. For example, Kuzu et al. [12] reported that they failed in attacking data structures such as CLKs with their CSP attack. Therefore, we consider the use of the CLK approach as the most promising modification of Bloom filters to prevent attacks.

Record-level Bloom filters. In [4], Durham et al. presented a variation of CLKs, called *Record-level Bloom filters*. In that paper samples of bit positions from the separate Bloom filters for each identifier are drawn. The sampled bits of all identifiers are concatenated and finally permuted. Applying the presented attack on such Bloom filters would not be promising, because they consist of multiple identifiers, like CLKs, and their bit positions are sampled from the initial Bloom filters. Thus, the assignment of possible atoms to bigrams will be much more difficult in this case as well.

Fake injections. The proposed attack strongly depends on comparing the observed frequencies of atoms and expected bigram frequencies. The expected bigram frequencies are highly skewed.[9] Hence, the success of an attack can be reduced if the frequency distribution of bigrams is modified artificially. This could be achieved, for example, by the insertion of random strings that contain rare bigrams. Thus, the overall frequency distribution of hashed bigrams will be closer to a uniform distribution, which makes a correct assignment of candidate atoms to bigrams more difficult. In [7], three such fake injection methods were described in the context of phonetic codes. Neither of the methods is without serious drawbacks. However, it should not be hard to adopt similar approaches to the Bloom filters.

7 Conclusion

We have demonstrated a successful attack on basic Bloom filters using only one identifier per Bloom filter. Therefore, this specific form of encoding identifiers should not be used for applications requiring strong security guarantees. However, we suggested a list of modifications that make attacks as described in this paper more difficult. At least one modification (*salting*) has not been mentioned in the previous PPRL literature. Neither this technique nor the effects of the other modifications have been studied in the context of attacks on privacy preserving record linkage. Because the number of alternatives to Bloom filters in PPRL is quite limited, further research on the cryptographic properties of modified Bloom filters and the effect of different databases on their security is desirable.

[9] In [5] it was shown that the distribution of English surnames is highly skewed and obeys Zipf's law by a discrete Pareto distribution. The same phenomenon is observed for the bigram frequencies.

A Probability of the Event $HW(a) = 15$

We are interested in the probability that a randomly chosen atom generated according to the rule in Equation (2) has a maximal Hamming weight equal to k. A high probability for this event implies a small probability for filtering true atoms in the deciphering process. We assume that the hash values of g and h are independent and uniformly distributed on $\{0, \ldots, m-1\}$.

Lemma A.1. Let $k, m \in \mathbb{N}$ be positive integers. Assume that x and y are independent and identically distributed according to the uniform distribution on $\{0, \ldots, m-1\}$. We define $\mathcal{I}(x, y) := \{x + iy \ mod \ m : i = 0, \ldots, k-1\}$. Then, for the probability of the event

$$A := \{(x, y) : |\mathcal{I}(x, y)| < k\}$$

we have

$$\mathbb{P}(A) = \frac{1}{m} \sum_{\substack{d<k \\ d|m}} \varphi(d)$$

where φ denotes Euler's totient function.

Proof. Because the cardinality of $\mathcal{I}(x, y)$ does not depend on x, it is sufficient to treat the case $x = 0$ without loss of generality. $|\mathcal{I}(0, y)| < k$ holds if and only if there exists $1 \leq d \leq k-1$ with $dy = 0 \ mod \ m$. We choose d minimal with this property; thus, y is an element of order d in the additive group $\mathbb{Z}/m\mathbb{Z}$ with $d \leq k-1$. For a divisor d of m, the number of elements of order d in $\mathbb{Z}/m\mathbb{Z}$ equals $\varphi(d)$. This yields

$$|\{0 \leq y \leq m-1 : |\mathcal{I}(0, y)| < k\}| = \sum_{\substack{d<k \\ d|m}} \varphi(d)$$

which implies the statement of the lemma. □

***Example* A.1.** In the example used in this paper, we considered $m = 1{,}000$ and $k = 15$ as parameters of the Bloom filters. In this case, we have

$$\sum_{\substack{d<k \\ d|m}} \varphi(d) = \sum_{d \in \{1,2,4,5,8,10\}} \varphi(d) = 1 + 1 + 2 + 4 + 4 + 4 = 16.$$

Hence, the probability that a randomly generated atom has Hamming weight $k = 15$ is equal to $P = \frac{984}{1000}$.

B Deciphering Bloom Filters Manually

We computed every step of the deciphering process described in Section 4 in the statistical programming language R. The assignment of candidate atoms to bigrams as described in Section 4.5 is illustrated by the following R output. The following table shows the 10 most frequent Bloom filters and the possible atoms contained in them. For example, the Bloom filter $\mathcal{B}_2$ contains the atoms $a_0, a_1, a_{36}, a_{44}, a_{194}$, and a_{271}.

```
0  |  0   1   9   16  36  40  41  418
1  |  2   3   7   18  76  93  112 170 180 183 293 348
2  |  0   1   36  44  194 271
3  |  0   1   2   7   20  53  66  79  272
4  |  0   1   5   19  34  60  94
5  |  0   1   5   37  40  61  399
6  |  4   6   10  13  17  47  127 156 210 222 229 309 342 381
7  |  0   1   22  25  132 249 415 460
8  |  2   3   7   48  99  141 153
9  |  0   1   2   3   7   15  22  28  109 170 171 180 273
```

Tests on several German surname databases indicate ER as the most frequent bigram, R␣ as the second-most common bigram, and CH as the third-most common bigram. The rank of the frequencies for all other bigrams differed across databases. Substitution of the three most frequent bigrams yields the following output:

```
0  |  ER  R_  9   16  36  40  41  418
1  |  CH  3   7   18  76  93  112 170 180 183 293 348
2  |  ER  R_  36  44  194 271
3  |  ER  R_  CH  7   20  53  66  79  272
4  |  ER  R_  5   19  34  60  94
5  |  ER  R_  5   37  40  61  399
6  |  4   6   10  13  17  47  127 156 210 222 229 309 342 381
7  |  ER  R_  22  25  132 249 415 460
8  |  CH  3   7   48  99  141 153
9  |  ER  R_  CH  3   7   15  22  28  109 170 171 180 273
```

As shown in the table, all records containing CH also contain atom a_7. Statistical analysis (and experience with names) led to the conjecture that atom a_7 is equal to SC. In addition, many of the records contain atom a_3. Because there are many German surnames beginning with the 4-gram ␣SCH, we assigned a_3 to ␣S. These additional assignments produced the following table:

```
0  |  ER  R_  9   16  36  40  41  418
1  |  CH  _S  SC  18  76  93  112 170 180 183 293 348
2  |  ER  R_  36  44  194 271
3  |  ER  R_  CH  SC  20  53  66  79  272
```

```
4  |  ER  R_  5   19  34  60  94
5  |  ER  R_  5   37  40  61  399
6  |  4   6   10  13  17  47  127 156 210 222 229 309 342 381
7  |  ER  R_  22  25  132 249 415 460
8  |  CH  _S  SC  48  99  141 153
9  |  ER  R_  CH  _S  SC  15  22  28  109 170 171 180 273
```

Next, we examined Bloom filters very similar to one of the 10 first Bloom filters. Among these, Bloom filters $\mathcal{B}_2$, $\mathcal{B}_{22}$, $\mathcal{B}_{37}$, and $\mathcal{B}_{44}$ have many candidate atoms in common. Furthermore, very few frequent names have several notations. Among those names is ␣MEYER␣. As a first guess, we supposed that these Bloom filters correspond to different variations of this name.

```
 2  |  ER  R_  36  44  194 271
22  |  ER  R_  15  36  38  44  367
37  |  ER  R_  10  36  264 271 556
44  |  ER  R_  10  36  38  213 269 616
```

All of these Bloom filters have the atom a_{36} in common, which was assigned to the bigram ␣M. We further assumed that the Bloom filter $\mathcal{B}_2$ corresponds to the most common variation of this name, which is ␣MEYER␣. Bloom filters $\mathcal{B}_2$ and $\mathcal{B}_{22}$ both also contain candidate atom a_{44}, which was assumed to be ME (this implies that $\mathcal{B}_{22}$ encrypts the name MEIER). Furthermore, MEIER contains the bigram EI, which is frequent in German surnames. Thus, we assigned $a_{15} \rightarrow$ EI. Bloom filters $\mathcal{B}_{37}$ and $\mathcal{B}_{44}$ should encrypt MAIER and MAYER, yielding $a_{10} \rightarrow$ MA. Furthermore, we got $a_{38} \rightarrow$ IE. In accordance with all previous assignments, candidate atom a_{367} has to be considered as a phantom atom.

```
0  |  ER  R_  9   16  _M  40  41  418
1  |  CH  _S  SC  18  76  93  112 170 180 183 293 348
2  |  ER  R_  _M  ME  194 271
3  |  ER  R_  CH  SC  20  53  66  79  272
4  |  ER  R_  5   19  34  60  94
5  |  ER  R_  5   37  40  61  399
6  |  4   6   MA  13  17  47  127 156 210 222 229 309 342 381
7  |  ER  R_  22  25  132 249 415 460
8  |  CH  _S  SC  48  99  141 153
9  |  ER  R_  CH  _S  SC  EI  22  28  109 170 171 180 273
```

Because atom a_{10} seems to encrypt the bigram MA, we searched for Bloom filters encrypting names ending with ...MANN␣, which is a frequent ending in Germany. These Bloom filters should have at least three more candidate atoms in common (in addition to a_{10}), which should encrypt the bigrams AN, NN, and N␣. Among many others, the following Bloom filters satisfied this condition:

```
 6  |  4   6   MA  13  17  47  127 156 210 222 229 309 342 381
15  |  4   6   MA  17  22  82  140 157 255 412
23  |  4   6   MA  13  17  39  46  277
29  |  ER  4   6   MA  17  ME  113 133 152 172 290 291 551
```

Thus, we assumed that the atoms a_4, a_6, and a_{17} encrypt the bigrams AN, NN, and N␣. However, at this point we were not able to make an exact assignment.

As a next step, we tested a frequent beginning of a name. Because many German surnames start with ␣SCHU... such as SCHUSTER, SCHUMANN, SCHULZ, or SCHUMACHER we suspected some of these names among the following Bloom filters:

```
  8  |  CH  _S  SC  48  99  141 153
 24  |  CH  _S  SC  11  99  102 141 153
220  |  ER  R_  CH  _S  SC  MA  20  79  99  136 157 440
142  |  CH  _S  4   6   SC  MA  17  99  127 134 135 157
```

Omitting the already known bigrams, these Bloom filters have only one possible atom in common, a_{99}, which might encrypt HU.

```
0  |  ER  R_  9   16  _M  40  41  418
1  |  CH  _S  SC  18  76  93  112 170 180 183 293 348
2  |  ER  R_  _M  ME  194 271
3  |  ER  R_  CH  SC  20  53  66  79  272
4  |  ER  R_  5   19  34  60  94
5  |  ER  R_  5   37  40  61  399
6  |  4   6   MA  13  17  47  127 156 210 222 229 309 342 381
7  |  ER  R_  22  25  132 249 415 460
8  |  CH  _S  SC  48  HU  141 153
9  |  ER  R_  CH  _S  SC  EI  22  28  109 170 171 180 273
```

Comparing this table with the list of frequent German surnames led to the conjecture that Bloom filters $\mathcal{B}_8$ and $\mathcal{B}_{24}$ correspond to the names SCHULZ and SCHULZE. Thus, we had $a_{48} \rightarrow$ Z␣. Because SCHOLZ is another frequent surname and very similar to SCHULZ, we searched for a Bloom filter very similar to $\mathcal{B}_8$, which turned out to be Bloom filter $\mathcal{B}_{55}$. So, $\mathcal{B}_{55}$ was likely to be the encryption of SCHOLZ.

```
8   |   CH  _S  SC  Z_  HU  141 153
55  |   CH  _S  SC  47  Z_  50  153
```

Given all previous assignments, the following candidate atoms could be assigned: $a_{141} \rightarrow$ UL, $a_{153} \rightarrow$ LZ.

At this point, the name SCHULZ has been found among the Bloom filters. All of its bigrams could be deciphered. By using similar arguments, all 934 names and their corresponding bigrams could be detected successfully.

References

[1] Bellare, M., Canetti, R., and Krawczyk, H. (1996). Keying hash functions for message authentication. In Koblitz, N. (ed.), *Advances in Cryptology–CRYPTO 96*, vol. 1109 of *LNCS*, 1–15. Berlin: Springer.

[2] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7): 422–426.

[3] Dice, L. R. (1945). Measures of the amount of ecologic association between species. *Ecology*, 26(3): 297–302.

[4] Durham, E. A., Kantarcioglu, M., Xue, Y., Toth, C., Kuzu, M., and Malin, B. (2012). Composite Bloom filters for secure record linkage. *IEEE Transactions on Knowledge and Data Engineering*, 1(1).

[5] Fox, W. R. and Lasker, G. W. (1983). The distribution of surname frequencies. *International Statistical Review*, 51(1): 81–87.

[6] Jaccard, P. (1912). The distribution of the flora in the alpine zone. *The New Phytologist*, 11(2): 37–50.

[7] Karakasidis, A., Verykios, V. S., and Christen, P. (2012). Fake injection strategies for private phonetic matching. In Garcia-Alfaro, J., Navarro-Arribas, G., Cuppens-Boulahia, N., and de Capitani di Vimercati, S. (eds.), *Data Privacy Management and Autonomous Spontaneus Security*, vol. 7122 of *LNCS*, 9–24. Berlin: Springer.

[8] Kirsch, A. and Mitzenmacher, M. (2008). Less hashing, same performance: Building a better Bloom filter. *Random Structures & Algorithms*, 33(2): 187–218.

[9] Kroll, M. and Steinmetzer, S. (2015). Automated cryptanalysis of Bloom filter encryptions of health records. Presentation at 8$^{\text{th}}$ International Conference on Health Informatics 2015 (forthcoming), Lisbon.

[10] Kuehni, C. E., Rueegg, C. S., Michel, G., Rebholz, C. E., Strippoli, M.-P. F., Niggli, F. K., Egger, M., and von der Weid, N. X. (2012). Cohort profile: The swiss childhood cancer survivor study. *International Journal of Epidemiology*, 41(6): 1553–1564.

[11] Kuzu, M., Kantarcioglu, M., Durham, E., and Malin, B. (2011). A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In Fischer-Hübner, S. and Hopper, N. (eds.), *Privacy Enhancing Technologies*, vol. 6794 of *LNCS*, 226–245. Berlin: Springer.

[12] Kuzu, M., Kantarcioglu, M., Durham, E. A., Toth, C., and Malin, B. (2013). A practical approach to achieve private medical record linkage in light of public resources. *Journal of the American Medical Informatics Association*, 20(2): 285–292.

[13] Napoleão Rocha, M. C. (2013). Vigilância dos óbitos registrados com causa básica hanseníase. Master thesis, Universidade de Brasília, Brazil.

[14] Office for National Statistics (2013). Beyond 2011: Matching anonymous data. Methods & Policies M9, ONS, London.

[15] Pal, S. K. and Sardana, P. (2012). Bloom filters & their applications. *International Journal of Computer Applications and Technology*, 1(1): 25–29.

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

[17] Randall, S. M., Ferrante, A. M., Boyd, J. H., and Semmens, J. B. (2013). The effect of data cleaning on record linkage quality. *BMC Medical Informatics and Decision Making*, 13(1): 64–74.

[18] Rogers, D. J. and Tanimoto, T. T. (1960). A computer program for classifying plants. *Science*, 132(3434): 1115–1118.

[19] Schnell, R., Bachteler, T., and Reiher, J. (2009). Privacy-preserving record linkage using Bloom filters. *BMC Medical Informatics and Decision Making*, 9(41): 1–11.

[20] — (2011). A novel error-tolerant anonymous linking code. Working Paper WP-GRLC-2011-02, German Record Linkage Center, Nuremberg.

[21] Schnell, R., Richter, A., and Borgs, C. (2014). Performance of different methods for privacy preserving record linkage with large scale medical data sets. Presentation at International Health Data Linkage Conference, Vancouver.

[22] Vatsalan, D., Christen, P., and Verykios, V. S. (2013). A taxonomy of privacy-preserving record linkage techniques. *Information Systems*, 38(6): 946–969.