TRANSACTIONS ON DATA PRIVACY 12 (2019) 91–115

Inference Attacks on Fuzzy Searchable Encryption Schemes

Daniel Homann*, **Lena Wiese***

*Universität Göttingen, Institut für Informatik, Goldschmidtstraße 7, 37077 Göttingen, Germany

E-mail: homann@cs.uni-goettingen.de, wiese@cs.uni-goettingen.de

Received 21 May 2018; received in revised form 27 October 2018; accepted 2 December 2018

Abstract. Searchable Encryption allows to securely store and search data on remote servers. Fuzzy searchable encryption is an extension of this technique which can find results matching approximately the search term. Low information leakage is a very important property of searchable encryption schemes. We show that the fuzzy searchable encryption schemes of Chuah and Hu, Wang et al., and Hu and Han do not have the claimed security properties. We describe several attacks on these schemes. For these attacks we use techniques as the reconstruction of information from Bloom filters, and the algebraic analysis of encryption schemes. These attacks show the difficulty of constructing secure and efficient fuzzy searchable encryption schemes.

Keywords. Searchable Encryption, Fuzzy Search, Security Analysis

1 Introduction

Searchable Encryption allows to securely store and search data on remote servers. The basic assumption of searchable encryption is that the user wants to save money by storing documents on remote cloud servers. The cloud provider might be curious and monitor its customers. Therefore, the user needs to encrypt sensitive documents to keep them confidential. Standard encryption schemes (e.g. AES) provide confidentiality. However, they do not allow to search encrypted documents efficiently. Searchable encryption solves this problem (see e.g. [3, 7, 9, 11, 15, 16, 19, 24, 25, 35, 36]). It allows a client to search documents stored on the server, while preserving their confidentiality towards the server.

When a user searches for a misspelled word with Google, he will find results which match the intended search term. For example, about a fifth of all Google queries for 'Britney Spears' are misspelled and corrected by the search engine [17]. Such functionality is also desirable when using searchable encryption schemes. This is provided by fuzzy searchable encryption schemes [2, 4, 10, 14, 20, 21, 28, 29, 31, 37–39, 42]. Fuzzy searchable encryption schemes are also known as similarity searchable encryption schemes.

Generally a (fuzzy) searchable encryption scheme works as follows: In a first step, a client generates some encryption keys. With these keys the client encrypts a set of documents. The encrypted documents are sent to the server and stored in a search index. When a search for a certain keyword should be performed, the client generates a trapdoor for this keyword with the help of the secret keys. This trapdoor is then sent to the server, which performes a

search in its index for documents containing the desired keyword. The resulting documents are returned to the client and decrypted.

It is possible to construct searchable encryption schemes which provide nearly perfect security. They do not leak any information about the stored documents except the size of the document collection. However, they require a search runtime and communication bandwidth linear in the size of the document collection [32]. Since cloud servers are used to save cost and improve the runtime, such schemes are of low practical value. To keep runtime and bandwidth requirements low, many searchable encryption schemes accept that a certain amount of information is leaked to the server, the so-called leakage.

To be considered secure, it must be proven that a searchable encryption scheme satisfies a security definition. This security definition (e.g. [11]) then limits the leakage of the scheme. Such bounds on the information leakage allow to evaluate the suitability of a certain scheme for a given application. Hence, it is important that a scheme really satisfies the claimed security definition. In this work we show that three fuzzy searchable encryption schemes do not fulfil the claimed security definitions. Their leakage is much higher than stated by their security definitions.

The searchable encryption schemes investigated in this work mainly use symmetric encryption operations. The only exception is the scheme of Wang et al. which uses asymmetric encryption for result ranking. Other techniques which have been used for encrypted search are fully homomorphic encryption and oblivious RAM (ORAM): Fully homomorphic encryption can be used to build very flexible searchable encryption schemes [5]. However, fully homomorphic operations have high computational cost. Consequently, fully homomorphic encryption based searchable encryption schemes are impractical for many use cases. ORAM is another technique which was proposed for use in searchable encryption. Naveed [32] showed that the use of ORAM alone does not lead to efficient and secure searchable encryption schemes. Nevertheless, ORAM inspired searchable encryption schemes have been proposed [20, 36]. Fully homomorphic encryption or ORAM are not used in the investigated schemes.

In this paper we show that three fuzzy searchable encryption schemes do not meet their security definition. For this aim we give several attacks on these schemes. In detail the contributions of this work are:

1. For the scheme of Chuah and Hu [10] we show that the index leaks a large amount of information: We give algorithms which derive the length of an index entry, the similarity between two index entries and the membership of a given word in the index. Depending on background knowledge and keyword length we can even infer the complete set of keywords contained in the index. All this information should not leak according to the security definition. The main idea of the attacks is to reconstruct information from Bloom filters. We also provide a practical evaluation of our attack based on a realistic email data set.

2. For the scheme of Wang et al. [38] we give four attacks which reconstruct information about the number and similarity of queries and keywords. They exploit the algebraic structure of the underlying encryption scheme. These attacks have an asymptotic advantage of $1/2$ and show that the scheme does not provide the claimed non-adaptive semantic security [11]. Our attacks show that the scheme does not even fulfil a weakened version of the claimed security definition.

3. For the scheme of Hu and Han [21] we construct an adversary that can derive information about stored keywords by querying for a similar keyword. When r is the number

of hash functions, the advantage of this adversary is greater than $1/2 - 1/2r$. To achieve reasonable search quality, the number of hash functions needs to be greater than one. This shows that in any practical configuration the scheme does not achieve the claimed IND-CKA security [16].

The rest of the paper is organized as follows: In the next section we describe related work on security of searchable encryption schemes. In Section 3 we give more information about the investigated schemes and the used data structures. The following three sections are devoted to the three investigated fuzzy searchable encryption schemes. In each of these sections we proceed as follows: First the fuzzy searchable encryption scheme is recapitulated. Then the claimed security definition of the scheme is given. Finally, we give attacks for the scheme and describe why the claimed security definition is not met. In Section 7 we summarise our results.

2 Related Work

Several works investigated the security of non-fuzzy searchable encryption schemes. They usually take one or both of the following roads to argue that a scheme is insecure:

- They show that a scheme's leakage, as given by its security definition, is too high to protect the confidentiality of certain practical datasets [8, 23, 26, 33, 34]. These attacks usually require the adversary to have a certain background knowledge.
- They assume a more powerful adversary than assumed by the scheme's security definition and show that this leads to a high leakage [8, 43].

Both types of attacks are called inference attacks because the adversary infers information about the cleartext datasets. Results on the security of non-fuzzy searchable encryption can be applied to fuzzy searchable encryption because the leakage of fuzzy schemes is usually higher than the leakage of non-fuzzy schemes. In contrast to the above works, in this work we show that the investigated fuzzy searchable encryption schemes do not fulfil their security definition, i.e. their leakage is higher than their security definitions allow. Most of our proposed attacks do not require the adversary to have any background knowledge.

Few works considered the security of fuzzy searchable encryption schemes explicitly. Boldyreva and Chenette [2] showed that a very strict security definition of fuzzy searchable encryption can only be achieved by very space-inefficient schemes. Furthermore, they showed that the scheme of Liu et al. [31] does not fulfil this security definition.

One of the schemes we investigate in this work is the scheme of Wang et al. An attack on this scheme was also described by Lin et al. [30]. Their attack uses a ciphertext-only setting (queries and index entries) and recovers the plaintexts belonging to these ciphertexts. They note that the task of the adversary is related to solving a non-linear optimisation problem (sparse non-negative matrix factorization). By solving the optimisation problem approximately, they give estimates for the reconstructed ciphertext. However, in a realistic setting, with a low number of chosen buckets in a query ($\rho = 5\%$) and a sufficiently large Bloom filter ($L = 1000$), their attack is less effective than randomly guessing decryptions of keywords. Random guessing would give a precision and recall of the bits in the Bloom filter of 5%, which is higher than the precision and recall of their attack in this setting.

Yao, Li and Xiao [41] proposed a chosen-plaintext attack on the encrypted k-nearest neighbour scheme of Wong et al. [40]. This scheme is used in the encrypted search of Wang et al.

However, the existence of this attack on the underlying ranking scheme does not violate the security definition of the searchable encryption scheme.

One key technique for information retrieval used in this work is the extraction of information from Bloom filters. In attacks on non-fuzzy searchable encryption schemes, this technique has been used by Pouliot and Wright [34]. In contrast to their work, statistical background information is not available in our scenario. Furthermore, our reconstruction technique works on Bloom filter encodings of parts of words (bigrams) instead of whole words.

3 Background

The basic setting of (fuzzy) searchable encryption consists of a server and a client. The user wants to store searchable data on the server, which should be encrypted, but still should be searchable. The advantages of offloading data sets to the server are a reduced storage demand on the client devices. Furthermore, the data can be accessed from a variety of client devices (notebook, smartphone, etc) whithout the need of data duplication on all these devices. In the security model of searchable encryption it is assumed, that the user is trustworthy. However, the server might be compromised and act againest the interest of the users. Therefore, the aim of searchable encryption is to prevent data breaches by the server.

3.1 Investigated Fuzzy Searchable Encryption Schemes

In this section we want to give an overview of the three investigated fuzzy searchable encryption schemes. Fuzzy means that the schemes allow to search for a word and find documents which contain the same or a very similar word.

The schemes have in common an honest-but-curious threat model and the ability to perform dynamic updates. An honest-but-curious adversary can monitor operations performed on the server. In contrast to a malicious adversary it is not allowed to deviate from the given protocol. An update operation is either the addition of a new document to the index or the deletion of an existing document from the index. A searchable encryption scheme can always perform updates by completely rebuilding its data structure. Therefore, a scheme is considered dynamic if there is a more efficient way to update the index than completely rebuilding its index.

The schemes differ in the used security definitions, the ability to perform multi-keyword searches, and their encoding of keywords. The schemes use three different security definitions: The game-based IND-CKA security definition of Goh [16], the simulation-based non-adaptive semantic security definition of Curtmola et al. [11] and a third less formal definition. The last definition is not completely precise in its formulation, but sufficiently precise to show that the proposed scheme does not fulfil this definition. The schemes of Wang et al. and Chuah and Hu can perform a multi-keyword search, i.e. they can search for documents most similar to a given set of search terms instead of just searching one word at a time. The schemes also vary in the way they allow to search for similar words. The scheme of Hu and Han uses wildcard characters in search queries to express fuzziness. The others measure the similarity between a search query and a document by the similarity of the corresponding bigram sets. Table 1 shows a comparison of the schemes.

Table 1: Properties of the investigated fuzzy searchable encryption schemes

Scheme	Security definition	Multi-keyword	Dynamic updates	Similarity
Chuah and Hu [10]	Custom	Yes	Yes	Bigram
Wang et al. [38]	Non-adaptive semantic security [11]	Yes	Yes	Bigram
Hu and Han [21]	IND-CKA [16]	No	Yes	Wildcard

3.2 Bloom Filter and Locality-Sensitive Hashing (LSH)

Bloom filters and locality-sensitive hashing are important techniques to understand the investigated fuzzy searchable encryption schemes. They also play an important role in our proposed attacks. Therefore, we briefly revise these two concepts in this section.

A Bloom filter is a probabilistic data structure [1]. It allows for fast membership tests. A Bloom filter consists of a family $\mathcal{H} = (h_i)_{1 \leq i \leq r}$ of hash functions and a bit array $v \in \{0,1\}^L$. The variables v_i are also known as buckets and initialized with zero. The hash functions map from a feature space $\mathcal{F}$ to $\{1, \ldots, L\}$, i.e.

$$h_i : \mathcal{F} \mapsto \{1, \ldots, L\}$$

for all $1 \leq i \leq r$. A feature $f \in \mathcal{F}$ is stored in the vector v by setting:

$$v_j = \begin{cases} 1 & \text{if } \exists 1 \leq i \leq r : h_i(f) = j \\ v_j & \text{otherwise} \end{cases} \qquad \text{for all } 1 \leq j \leq L.$$

A Bloom filter stores multiple features by adding them consecutively. To check whether a feature $f' \in \mathcal{F}$ is contained in a Bloom filter v, the hash values $h_i(f')$ are calculated for $1 \leq i \leq r$. If one of the buckets $v_{h_i(f')}$ is not set to 1, the feature f' is not contained in the Bloom filter. Otherwise, no definite statement about the membership is possible. However, the probability of f' being included in the Bloom filter can be calculated and is usually very high. The concept of Bloom filters was extended to counting Bloom filters. Counting Bloom filters do not only store that a certain bucket was hit, but also how often it was hit, i.e. they use a $v \in \mathbb{N}^L$.

A family of locality-sensitive hash (LSH) functions can be used to encode similarity [22]. Let dist be a similarity metric on the feature space $\mathcal{F}$. Then a family of hash functions $\mathcal{H}$ is called a (r_1, r_2, p_1, p_2)-sensitive LSH family if for any features $x, y \in \mathcal{F}$ and for any $h \in \mathcal{H}$ it holds:

- if $\mathsf{dist}(x, y) \leq r_1$, then $\mathsf{P}(h(x) = h(y)) \geq p_1$
- if $\mathsf{dist}(x, y) \geq r_2$, then $\mathsf{P}(h(x) = h(y)) \leq p_2$.

Assume that a (r_1, r_2, p_1, p_2)-sensitive LSH family $\mathcal{H}$ is given. Then an LSH family $\mathcal{H}'$ of an arbitrary sensitivity (r_1', r_2', p_1', p_2') can be constructed by an AND- and OR-combination of the hash functions of $\mathcal{H}$.

4 Scheme of Chuah and Hu

The scheme of Chuah and Hu [10] is rather complex. Therefore, we will only discuss the aspects of the construction which are relevant for the attacks. The basic server-side data

structure of the scheme is a Bed-tree [44]. This data structure is used to store key-value pairs (v, P). Each unique keyword in the set of stored documents corresponds to exactly one key-value pair in the Bed-tree. The Bed-tree is chosen for performance reasons. For our security analysis it can be understood as a simple key-value store.

The payload P of a key-value pair is used to store information for measuring the similarity of keywords. The structure of P is rather complex. As P is not relevant for our attack, we will not describe it here. The key v of a key-value pair (v, P) is given by the gram counting order [44].

The gram counting order is calculated by splitting the keyword up in bigrams. To denote the whitespace before the first and after the last character a special whitespace character (#) is used when building the bigram sets. The bigram set of the word CASTLE, for example, is given by:

$$\{\#\text{C}, \text{CA}, \text{AS}, \text{ST}, \text{TL}, \text{LE}, \text{E}\#\}.$$

Then these bigrams are inserted in a counting Bloom filter. The used hash functions $\mathcal{H} = \{h_1, \ldots, h_r\}$ do not depend on any secret key. Applying the described counting Bloom filter on a bigram set yields the gram counting order $v \in \mathbb{N}^L$. The vector v is then stored in unencrypted form with the payload P on the server.

For an example of the gram counting order assume $L = 4$, $r = 1$, and let h_1 be given by:

$$h_1(f) = \begin{cases} 1 & \text{if } f = \#\text{C or LE} \\ 2 & \text{if } f = \text{E}\# \text{ or AS or CA} \\ 3 & \text{if } f = \text{ST or TL} \\ 4 & \text{otherwise.} \end{cases}$$

Then the gram counting order of $w = \text{CASTLE}$ is given by: $v = (2, 3, 2, 0)$. For the sake of the example, a small size L of the Bloom filter was chosen here. In a realistic application the Bloom filter size would be higher: Chuah and Hu proposed a Bloom filter size of $L = 32$ and the choice of $r = 2$ hash functions.

They do not claim any security guarantees when search operations are performed (see Section 4.1). Hence, search operations are not relevant for our attacks and, consequently, their description is omitted here.

4.1 Security Definition

According to the authors, their scheme fulfils the following security definition. It should prevent the adversary 'from learning additional information from any dataset and its associated index' in the following cases (based on [6]):

Known-ciphertext model: A ciphertext-only setting where the adversary gets only a copy of the data stored on the server.

Known-background model: A setting where the adversary has additional statistical information (e.g. keyword frequencies) about the stored documents.

This type of security definitions is sometimes called snapshot security, as the adversary has only access to a snapshot of the index data structure. Obviously, the attacker in the first setting is strictly less powerful than in the second setting. Compared to the commonly used security definitions for fuzzy searchable encryption (e.g. [28]) both security definitions are rather weak.

Table 2: Inference attacks on the scheme of Chuah and Hu [10]

Information derived	Section	Applicable	
		Known-ciphertext	Known-background
Length of stored keywords	4.2.1	Yes	Yes
Similarity of stored keywords	4.2.2	Yes	Yes
Membership of a given word in the index	4.2.3	Yes	Yes
Keywords contained in the index	4.2.4	No	Yes
Keywords contained in the index	4.2.5	Yes	Yes

Usually, the security definitions of searchable encryption schemes capture the leakage which is caused by search operations. This is a more realistic security definition because in the cloud setting such information is available to the adversarial server running the scheme. The leakage of search operations consists of trapdoors sent to and corresponding result sets received from the server.

Although the above-stated security definition does not even consider the leakage of search operations, it is still not met by the proposed scheme. This is shown in the next section.

4.2 Attacks

In this section we describe which information the adversary can infer from a given index. Table 2 gives an overview of our attacks.

4.2.1 Length of Keyword

In the known-ciphertext setting the server can deduce the lengths of all stored keywords. Let w be a keyword stored on the server. The gram counting order v of w is available to the server because it is the key in the Bed-tree data structure stored on the server. The gram counting order v leaks the length $|w|$ of w by:

$$|w| = \frac{\sum_{i=1}^{L} v_i}{r} - 1.$$

This holds because each bigram is added r times to the Bloom filter. Therefore, each bigram increases the sum of v by r. A word w of length $|w|$ consists of $|w| - 1$ bigrams. When the bigrams are generated in the scheme of Chuah and Hu, the keyword w is surrounded by $\#$ characters. Therefore, the encoding of a word w of length $|w|$ consists of $|w| + 1$ bigrams.

4.2.2 Similarity

In the known-ciphertext setting the adversary can estimate the similarity of two index entries. Let v^1 and v^2 be the index entries for the keywords w^1 and w^2. Then an estimate for the similarity of the keywords w^1 and w^2 is given by:

$$s(w^1, w^2) := \frac{\sum_{i=1}^{L} |v_i^1 - v_i^2|}{\max_{k \in \{1,2\}} \sum_{i=1}^{L} v_i^k}.$$

The function s takes values in the interval $[0, 1]$. A value of 0 denotes high, 1 low similarity. The definition of s can be explained as follows: The numerator of s counts the similarity of the vectors. This figure is correlated to the number of coinciding bigrams. The denominator normalises this measure by the total number of occuring bigrams.

Furthermore, the gram counting order can be associated with the edit distance of the corresponding words [44]. A lower bound on the edit distance $\mathsf{edit\text{-}dist}(w^1, w^2)$ of the keywords w^1 and w^2 is given by:

$$\mathsf{edit\text{-}dist}(w^1, w^2) \geq \frac{\max\left(\sum_{v_i^1 > v_i^2} v_i^1 - v_i^2, \sum_{v_i^1 < v_i^2} v_i^2 - v_i^1\right)}{rL}.$$

4.2.3 Membership Test

In the known-ciphertext setting the adversary can infer with high probability whether a word w is contained in the index. For this, the adversary calculates the gram counting order v of w. The adversary can do this because the calculation of the gram counting order v does not depend on a secret key. If v is not contained as key in the Bed-tree, the adversary can deduce that w is not contained in any of the stored documents. When v is contained in the Bed-tree, the conclusion is not clear. Due to the probabilistic structure of the counting Bloom filter, it could happen that two different words map to the same gram counting order v. Therefore, it holds: If v is contained in the Bed-tree, it is very probable, but not certain, that w is contained in one of the stored documents.

4.2.4 Content Reconstruction with Background Information

In the known-background model the adversary can deduce for which keyword w a given Bed-tree entry v stands. As background knowledge it requires to know a set (dictionary) that the keywords were chosen from.

To perform the attack the adversary calculates the gram counting order of all keywords of the dictionary and compares the results to v. There might be several words $w_1, \ldots, w_i$ matching v. One of these words is the correct keyword w. The other keywords are words having the same gram counting order as w. So the adversary can only tell that v belongs to one keyword from the set $\{w_1, \ldots, w_i\}$. As this set of keywords would also be returned when users search for w, here the quality of the search for legitimate users directly corresponds to the information derivable by the adversary. To speed up the attack, the adversary computes in a first step the length of the attacked keyword w (see Section 4.2.1). Then it only needs to consider the terms from the dictionary having this length.

To perform the attack, the adversary requires a dictionary of possible keywords. For the problem domain of text search such dictionaries are readily available if the language of the documents is known. These dictionaries might not contain all keywords occurring in the document collection but at least the most common ones. Consequently, background knowledge about the language of the stored documents is sufficient. We will show the practical feasibility of the attack with this low amount of background knowledge in Section 4.3.1.

4.2.5 Content Reconstruction without Background Information

If no background information is available (known-ciphertext model), the attacker can perform a depth-first search for the keyword. Our attack function is given by Algorithm 1. This function computes a candidate set that contains the searched keyword.

Algorithm 1 Calculation of keywords corresponding to a given gram counting order v

function SEARCH(v)
  $len \leftarrow \frac{\sum_{i=1}^{L} v_i}{r} - 1$
  $w \leftarrow \underbrace{* \cdots *}_{len\text{-times}}$
  return SEARCHTREE($v, len, w, 1$)

Algorithm 2 Recursive search for keywords corresponding to a given gram counting order

function SEARCHTREE(v, len, w, pos)
  if $pos = len + 1$ **then**
    return $\{w\}$
  else
    res $\leftarrow \emptyset$
    $w' \leftarrow w$
    for all $c \in \mathbb{A}$ **do**
      $w'_{pos} \leftarrow c$
      $v' \leftarrow$ GRAMCOUNTINGORDER(w')
      if $v_i - v'_i \geq 0 \quad \forall 0 \leq i < L$ **then**
        res $\leftarrow$ res $\cup$ SEARCHTREE($v, len, w', pos + 1$)
    return res

The algorithm requires to know the alphabet $\mathbb{A}$ from which the letters of the keywords are chosen (e.g. lowercase letters). Let $*$ be a special character with $* \notin \mathbb{A}$. The function GRAMCOUNTINGORDER(w) returns the gram counting order of a keyword w. When a $*$ character is contained in a bigram, this bigram is skipped and not included in the counting Bloom filter.

On a high level the proposed algorithm works as follows: To find the set of possible keywords belonging to a given gram counting order v, the function SEARCH(v) (Algorithm 1) is called. This function initializes the parameters of the search for keywords matching v. With these parameters, the recursive, depth-first search function SEARCHTREE (Algorithm 2) is called. Starting at the first letter, it recursively fixes letters of possible keywords. A new keyword w' derived from w is considered for further search if its gram counting order SEARCHTREE returns a candidate set, consisting of words with gram counting order v. This set contains the searched keyword.

In detail SEARCH(v) works as follows: The function calculates the length len of the corresponding keyword (see Section 4.2.1). Then it creates a keyword w of this length with each $*$ character standing for a yet unknown character from the alphabet $\mathbb{A}$. Finally, it calls the recursive function SEARCHTREE and returns its result.

The algorithm SEARCHTREE calculates the possible letters for position pos of a keyword w. As additional arguments it receives the gram counting order v of the searched keyword and its length len. When the end of the word is reached, i.e. no more letters are to be chosen, SEARCHTREE returns the set containing the current word w. At this point it is guaranteed that all $*$ characters of w have been replaced by letters from $\mathbb{A}$. Otherwise, it tests all possible letters for the current position pos. For each tested letter the gram counting order v' of the resulting word w' is calculated. A word w' belongs to the candidate set if its gram counting order v' is in no dimension higher than the gram counting order v of the keyword. This is captured by the condition $v_i - v'_i \geq 0$ for all $0 \leq i < L$. If this condition is fulfilled, the

function fixes the current letter and recursively considers the next position of the word. In the end, a result set of all matching words is returned.

It is guaranteed that the result set is non-empty and contains the keyword belonging to v. But as before the result set might contain other words having the gram counting order v. The worst-case runtime of this algorithm is $\mathcal{O}(|\mathbb{A}|^{|w|})$ for a word w of length $|w|$. Although the runtime of the attack is exponential in the word length, it is feasible for many words because words in many common languages tend to be short. This feasibility of the attack will also be shown in Section 4.3.2.

4.3 Experimental Evaluation

In this section we provide an experimental evaluation for the keyword reconstruction attacks proposed in Section 4.2.4 and Section 4.2.5. By evaluating the known-background attack, we also implicitly evaluated the membership attack (Section 4.2.3) because both attacks use the same idea for information reconstruction.

For the evaluation we assume that the user indexed a large set of mails with the searchable encryption scheme of Chuah and Hu. As email data set we used the complete Enron data set [13, 27]. We processed these $517\,401$ mails as follows: In a first step the header of the mails containing information as timestamp and subject of the mail was removed. Subsequently all lines containing binary data (e.g. mail attachments) were removed as well. The remaining lines constitute the body of the mail. Punctuation marks, parentheses and numbers were removed from the body. Finally, all words were changed to lowercase letters and split in single words. In total the data set contains $424\,943$ different words. The high number of words compared to English language dictionaries is caused by multiple incorrect spellings of one word which are contained in the data set and counted as different words. Furthermore, the list of words contains mail addresses and proper names.

4.3.1 Reconstruction with Background Knowledge

To perform the described attack, the adversary needs some knowledge about the set where the keywords were chosen from. Here, we assume that the attacker only knows that the keywords are English language words. The adversary then can use a freely available list of English language words, in our case the words_alpha list [12]. This list contains 370 099 words. If two (or more) of the words in the list have the same gram counting order, the adversary simply assumes the first word to be the correct cleartext for this gram counting order.

Our implementation takes about a minute on a standard desktop machine to perform the attack against the whole Enron word set. Depending on the parameters of the scheme, our attack can correctly identify up to $65\,626$ different words. Since the attacker's dictionary and the Enron word set have $65\,638$ words in common, this means that all except 12 words from the Enron data set contained in the attacker's dictionary could be recovered. Higher recovery rates could be achieved by using a better dictionary containing more of the words from the target data set. The 12 words which could not be recovered have non-unique bigram sets and therefore an incorrect keyword was chosen by the adversary. Detailed results can be found in Figure 1. We also evaluated the false-positive rate of our attack (Figure 2). For reasonable parameter choices the false-positive rate is very low. Chuah and Hu, for example, proposed the choice of $L = 32$ and $r = 2$ for which our attack has a false-positive rate of about 0.02%. The number of false-positives never falls below 0.018%.

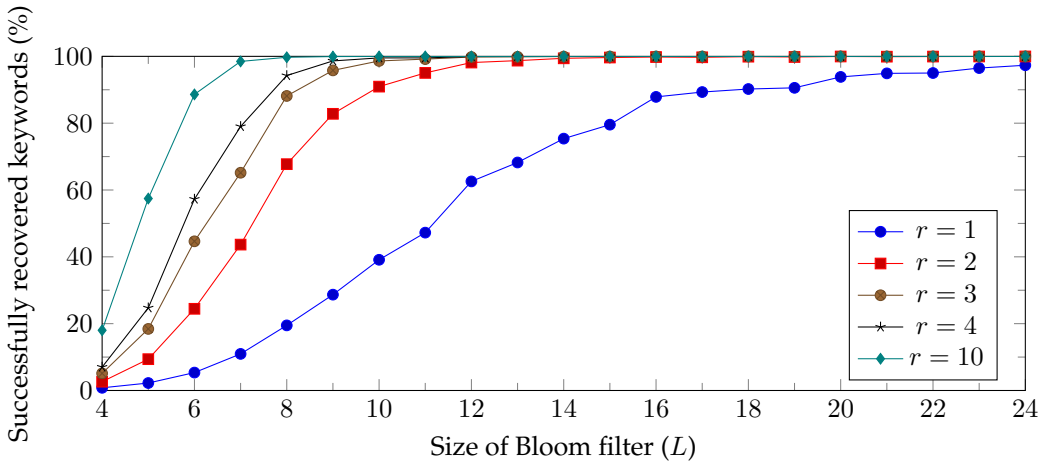

Figure 1: Percentage of words that our known-background attack on the scheme of Chuah and Hu recovers correctly with regard to Bloom filter size (L) and used number of hash functions (r)

This is caused by 78 words from the Enron word set which have the same bigram set as some words from the attackers list, although the words are not equal.

4.3.2 Reconstruction without Background Knowledge

We also implemented the keyword reconstruction technique which does not require background knowledge. In addition to the lowercase letters the keywords in the Enron word set contain some special characters like '@' and '&'. This alphabet $\mathbb{A}$ of all appearing characters was known by the adversary. We tested our attack with the preferred encoding parameters of Chuah and Hu ($L = 32$ and $r = 2$). With one day of computation time on an Intel i7-7700K@4.20GHz CPU we were able to determine 38.3% of the words from the Enron word set correctly. For these words the candidate set had size 1, so the words were directly determined. For an additional 10.8% of the word set the algorithm computed candidate sets with an average size of 3.2 words.

4.4 Mitigation and Discussion

As these attacks show, the scheme leaks lengths, similarity and membership of keywords in the index. Depending on word length and background knowledge, it is even possible to reconstruct many keywords stored in the index. All this information should not be leaked according to the security definition.

Some of the leakage could be prevented by changing the searchable encryption schemes as follows: The leakage of word lengths could be prevented by padding all words to a certain maximal keyword length. The reconstruction of words could be prevented by using keyed hash functions with a secret key instead of the un-keyed hash functions in the Bloom filter construction. However, this would not prevent all leakage. Especially the leakage of the similarity between stored keywords is hard to prevent.

It is important to note that all described attacks only use the leakage induced by the index

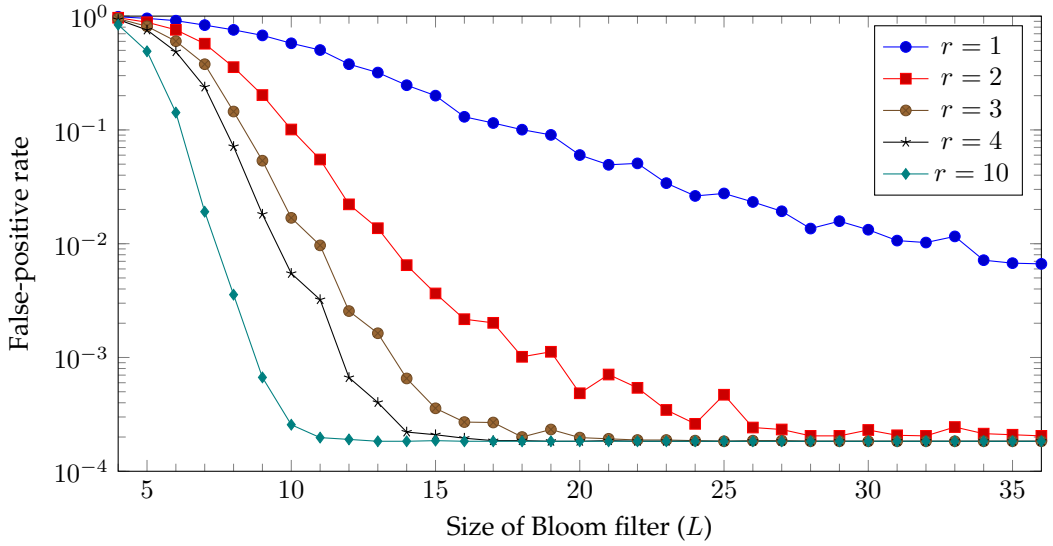

Figure 2: False-positive rate of our known-background attack on the scheme of Chuah and Hu for different Bloom filter sizes L and different numbers of hash functions r

data structure. When search trapdoors are available to an adversary, this causes additional leakage to the adversary. Its impact is not discussed here because the security definition of the scheme does not claim it to be secure when the search leakage is considered.

5 Scheme of Wang et al.

Wang et al. [38] gave two versions of their scheme. The enhanced version is described here. Let $\mathbb{A}$ be the alphabet where the letters of the keywords were chosen from. Given some key length L it requires a family

$$\mathcal{H} \coloneqq \left\{ h_i : \{0,1\}^{|\mathbb{A}|\times|\mathbb{A}|} \to \{1,\ldots,L\} \;\middle|\; 1 \le i \le r \right\}$$

of LSH functions and a keyed pseudo-random function

$$f_{k_i} : \{1,\ldots,L\} \to \{1,\ldots,L\}.$$

The key of the scheme is given by a tuple (M_1, M_2, S, k) where $M_1, M_2 \in \mathbb{R}^{L\times L}$ are invertible matrices, $S = (s_j)_{1\le j\le L} \in \{0,1\}^L$ is a binary vector, and $k = (k_i)_{1\le i\le r}$ is a set of keys for the keyed pseudo-random function f. Although this is not explicitly stated by Wang et al., we assume that the components of the key (M_1, M_2, S, and k) are chosen randomly from the uniform distribution on their respective sets. Any other choice of these parameters would further increase the leakage of the scheme.

The index of the scheme is built document-wise. For each document $D \in \mathbb{D}$ a Bloom filter $I = (i_j)_{1\le j\le L}$ of length L is created. Each keyword w of D is then interpreted as vector $\vec{w}$ in the space $\{0,1\}^{|\mathbb{A}|\times|\mathbb{A}|}$ by the following embedding:

$$\vec{w}_i = \begin{cases} 1 & \text{if the } i\text{-th bigram of } \mathbb{A}\times\mathbb{A} \text{ is contained in } w \\ 0 & \text{otherwise.} \end{cases}$$

Then the functions $f_{k_i} \circ h_i(\vec{w})$ for $1 \leq i \leq r$ are calculated. The corresponding buckets in the Bloom filter I are set to one. This is done for all keywords $w \in D$. Floating-point arrays $U = (u_j)_{1 \leq j \leq L}$ and $V = (v_j)_{1 \leq j \leq L}$ are defined based on I:

$$\begin{aligned} u_j = v_j = i_j, \quad & \text{if } s_j = 1 \\ u_j = \frac{i_j}{2} + t \quad \text{and} \quad v_j = \frac{i_j}{2} - t, \quad & \text{if } s_j = 0 \end{aligned}$$

for some random $t \in \mathbb{R}$. These vectors are encrypted with M_1 and M_2:

$$U_{\text{Enc}} \coloneqq M_1^T U \quad \text{and} \quad V_{\text{Enc}} \coloneqq M_2^T V.$$

Finally, the tuple $(\text{id}(D), U_{\text{Enc}}, V_{\text{Enc}})$ is stored on the server. Here, $\text{id}(D)$ is the identifier of the document D. This is done for all documents D of the document collection $\mathbb{D}$.

The generation of search queries works similar. The keywords are inserted in a Bloom filter Q of length L as explained above. Then arrays $X = (x_j)_{1 \leq j \leq L}$ and $Y = (y_j)_{1 \leq j \leq L}$ are generated from $Q = (q_j)_{1 \leq j \leq L}$ via:

$$\begin{aligned} x_j = y_j = q_j \quad & \text{if } s_j = 0 \\ x_j = \frac{q_j}{2} + t \quad \text{and} \quad y_j = \frac{q_j}{2} - t \quad & \text{if } s_j = 1 \end{aligned} \tag{1}$$

for some random $t \in \mathbb{R}$. The only difference to the generation of index entries is that the cases are switched, i.e. $x_j = y_j = q_j$ is chosen for $s_j = 0$ instead of $s_j = 1$. The query is encrypted by:

$$X_{\text{Enc}} \coloneqq M_1^{-1} X \quad \text{and} \quad Y_{\text{Enc}} \coloneqq M_2^{-1} Y,$$

and the resulting trapdoor $(X_{\text{Enc}}, Y_{\text{Enc}})$ is then sent to the server. The server now calculates for all stored documents $D \in \mathbb{D}$ a score with regard to this trapdoor. The score $s(D, w)$ of a document D with regard to a query w is calculated by:

$$\begin{aligned} s(D, w) &= (U_{\text{Enc}})^T X_{\text{Enc}} + (V_{\text{Enc}})^T Y_{\text{Enc}} \\ &= \left(M_1^T U\right)^T M_1^{-1} Q^1 + \left(M_2^T V\right)^T M_2^{-1} Q^2 = U^T Q^1 + V^T Q^2 = I^T Q. \end{aligned}$$

The above calculation shows that the score calculated from encrypted trapdoors and index entries matches the score of the plaintext query and index entry. Finally, a list of document identifiers with their corresponding scores is returned to the client.

Wang et al. gave two versions of their scheme. We described the enhanced, supposedly more secure, version. The only difference to the basic version is that the basic version does not use the pseudo-random function f. In the basic scheme keywords are inserted in Bloom filters during index and query generation only using the LSH functions. In Section 5.2 we will describe the attacks only for the enhanced version of the scheme. With slight modifications they also apply to the basic version.

5.1 Security Definition

The basic construction should provide non-adaptive semantic security as defined by Curtmola et al. [11]. The enhanced version of the scheme should be able to withstand an even more powerful adversary, which they call an adversary in the known-background model. In addition to the capabilities available to the adversary in the previous setting, this adversary

has access to statistical information about the stored documents, like keyword frequencies. The known-background model lacks a formal definition. Furthermore, the adversary in the known-background model is strictly more powerful than in the non-adaptive semantic security setting. This means all attacks possible in the latter setting are also possible in the former setting. In Section 5.2 we assume that the attacker only has the capabilities of the non-adaptive semantic security setting.

Let us recall (nearly verbatim) the simulation-based definition of non-adaptive semantic security [11].

Definition 1 (Non-adaptive semantic security for searchable encryption)**.**
Let SSE $=$ (GENERATEKEYS, ENCRYPTDOCS, TRAPDOOR, SEARCH, DECRYPTDOC) be a searchable encryption scheme, $L \in \mathbb{N}$ be the security parameter, $\mathcal{A}$ be an adversary, $\mathcal{S}$ be a simulator and consider the following probabilistic experiments:

function $\text{REAL}_{\text{SSE},\mathcal{A}}(L)$
  $K \leftarrow \text{GENERATEKEYS}(1^L)$
  $(\mathcal{H}, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^L)$
  parse $\mathcal{H}$ as $(\mathbb{D}, \mathbb{W})$
  $(I, c) \leftarrow \text{ENCRYPTDOCS}_K(\mathbb{D})$
  for all $1 \leq i \leq q$ **do**
    $t_i \leftarrow \text{TRAPDOOR}_K(\mathbb{W}_i)$
  let $t = (t_1, \ldots, t_q)$
  return $\mathcal{V} = (I, c, t)$ and $st_{\mathcal{A}}$

function $\text{SIM}_{\text{SSE},\mathcal{A},\mathcal{S}}(L)$
  $(\mathcal{H}, st_{\mathcal{A}}) \leftarrow \mathcal{A}(1^L)$
  $\mathcal{V} \leftarrow \mathcal{S}(\mathcal{T}(\mathcal{H}))$
  return $\mathcal{V}$ and $st_{\mathcal{A}}$

We say that SSE is semantically secure if for all polynomial-size adversaries $\mathcal{A}$, there exists a polynomial-size simulator $\mathcal{S}$ such that for all polynomial-size distinguishers $\mathcal{D}$ the advantage of the adversary $\mathcal{A}$

$$\text{Adv}_{\mathcal{A}} = |\text{P}(\mathcal{D}(\text{REAL}_{\text{SSE},\mathcal{A}}(L)) = 1) - \text{P}(\mathcal{D}(\text{SIM}_{\text{SSE},\mathcal{A},\mathcal{S}}(L)) = 1)|$$

is negligible in L. The probabilities are taken over the coins of GENERATEKEYS and ENCRYPTDOCS.

In this definition $\mathbb{D}$ is the set of stored documents and $\mathbb{W}$ the set of performed searches. The generated index is denoted by I and the collection of encrypted documents by c. The internal state of the adversary is given by $st_{\mathcal{A}}$. The history $\mathcal{H}$ describes the state of the searchable encryption scheme, i.e. it consists of the stored documents and the performed search queries. The view $\mathcal{V}$ contains the information that is visible to the adversary. The trace $\mathcal{T}$ describes what should be the maximal amount of information that the adversary is able to infer.

In an informal way Definition 1 says that the server can never know more about the stored data than defined by the trace $\mathcal{T}$. In the security definition of Wang et al. the trace $\mathcal{T}$ is:

$$\mathcal{T}(\mathcal{H}) = \{\mathcal{T}(\mathbb{W}_i) \mid 1 \leq i \leq q\}, \quad \text{where } \mathcal{T}(\mathbb{W}_i) = \{(\text{id}(D), s(D, \mathbb{W}_i)) \mid D \in \mathbb{D}\}.$$

Here, $s(D, \mathbb{W}_i)$ is the calculated similarity score between the query $\mathbb{W}_i$ and the document D.

5.2 Attacks

The attacks described here work for the most secure version of the scheme (enhanced version) assuming only the least powerful attacker capabilities (non-adaptive semantic

Table 3: Inference attacks on the scheme of Wang et al. [38]

		Applicable for trace definition	
Information exploited	Section	Original $\mathcal{T}(\mathcal{H})$	Extended $\mathcal{T}'(\mathcal{H})$
Number of trapdoors	5.2.1	Yes	No
Number of documents	5.2.2	Yes	No
Similarity of trapdoors	5.2.3	Yes	Yes
Similarity of documents	5.2.4	Yes	Yes

security). To show that the scheme does not satisfy its security definition we will define an adversary $\mathcal{A}$ and a distinguisher $\mathcal{D}$ such that every simulator $\mathcal{S}$ will lead to a non-negligible advantage $\text{Adv}_{\mathcal{A}}$. We will furthermore introduce a new extended trace definition and show that even this trace definition will not capture all the leakage of the scheme. Our attacks are summarised in Table 3.

5.2.1 Number of Trapdoors

The adversary outputs a history $\mathcal{H} = (\mathbb{D}, \mathbb{W})$. The set of documents $\mathbb{D}$ is empty, i.e. $\mathbb{D} = \emptyset$. For the set of queries the adversary samples a bit b from the uniform distribution on $\{0, 1\}$. Depending on b it chooses the set of queries as follows:

$$\mathbb{W} = \begin{cases} \emptyset & \text{if } b = 0 \\ (w) & \text{otherwise,} \end{cases}$$

where w is some fixed keyword. The distinguisher $\mathcal{D}$ generates an output based on the view $\mathcal{V} = (I, c, t)$. It outputs:

$$\mathcal{D}(\mathcal{V}) = \begin{cases} 0 & \text{if } |t| = 0 \\ 1 & \text{otherwise.} \end{cases}$$

All histories generated by the adversary contain an empty set of documents $\mathbb{D}$. This means that for all these histories the trace is empty, i.e. $\mathcal{T}(\mathcal{H}) = \emptyset$. Based on this trace, the simulator needs to generate a view $\mathcal{V} = (I, c, t)$. In particular, it needs to decide how many trapdoors it generates. The simulator knows that the number of trapdoors is 0 in one half of the cases, and 1 in the other half of the cases. Since the trace $\mathcal{T}(\mathcal{H})$ is an empty set, it does not contain any information about the number of trapdoors. Therefore, the output of $\mathcal{D}(\text{SIM}_{\text{SSE},\mathcal{A},\mathcal{S}})$ is independent of the initial choice of b by the adversary. When the protocol is executed, the number of trapdoors is visible to the distinguisher as part of the view. Consequently, $\mathcal{D}(\text{REAL}_{\text{SSE},\mathcal{A}})$ always returns the correct bit b. This means the adversarial advantage $\text{Adv}_{\mathcal{A}}$ is at least $1/2$. This advantage is not negligible in L and therefore the scheme does not meet its security definition.

5.2.2 Number of Documents

In the previous attack we exploited the fact that the trace does not contain the number of queries if no documents are stored on the server. Likewise, the trace does not contain the number of stored documents if no queries are performed. Therefore, we could define an

adversary which samples a bit b according to the uniform distribution and chooses

$$\mathbb{D} = \begin{cases} \emptyset & \text{if } b = 0 \\ (D) & \text{otherwise,} \end{cases}$$

where D is some fixed document. It then outputs $\mathcal{H} = (\mathbb{D}, \mathbb{W})$ where $\mathbb{W} = \emptyset$. The distinguisher $\mathcal{D}$ is defined as follows:

$$\mathcal{D}(\mathcal{V}) = \begin{cases} 0 & \text{if } |c| = 0 \\ 1 & \text{otherwise} \end{cases}$$

for a given view $\mathcal{V} = (I, c, t)$. With an analogous argumentation as before it follows that for any simulator holds:

$$\text{Adv}_{\mathcal{A}} \geq \frac{1}{2}.$$

5.2.3 Similarity of Trapdoors

As a consequence of the above attacks one might want to fix the trace definition. The above attacks could be prevented if the simulator is given access to the number of trapdoors and documents. Therefore, we consider the following new definition of the trace $\mathcal{T}'(\mathcal{H})$:

$$\mathcal{T}'(\mathcal{H}) = (\mathcal{T}(\mathcal{H}), |t|, |c|).$$

In this definition, $\mathcal{T}(\mathcal{H})$ is the original trace definition. By leaking more information, schemes having this new trace would be less secure than schemes with the original trace definition. In this section we give another attack on the scheme of Wang et al. This attack is feasible for the original and the new trace definition. This means that the leakage of the scheme is even higher than the leakage contained in $\mathcal{T}'(\mathcal{H})$.

The basic idea of the attack is to investigate the dimension of the space spanned by several keywords. The more similar the keywords are, the smaller is the dimension of this space. In the following we will formalize this intuition for the case $r = 1$. The idea of the attack works as well for $r > 1$. However, the calculation of the adversarial advantage in this setting becomes more complex. Therefore, we do not present it here.

For the attack in the case $r = 1$ the adversary chooses an empty set of documents $\mathbb{D} = \emptyset$. Depending on the value of a variable b sampled from the uniform distribution on $\{0, 1\}$ it chooses the set of queries:

$$\mathbb{W} = \begin{cases} (\underbrace{w_1, \ldots, w_1}_{L-\text{times}}) & \text{if } b = 0 \\ (w_1, \ldots, w_L) & \text{otherwise.} \end{cases}$$

The keywords $w_1, \ldots, w_L$ are chosen such that:

$$h_1(w_i) = i$$

for all $1 \leq i \leq L$. Such a choice of the w_i is possible because the LSH function h_1 is not supposed to be a cryptographic hash function. For the LSH function proposed by Wang et al. a suitable w_i can be found in the (small) space of bigrams $\mathbb{A} \times \mathbb{A}$.

For the L queries generated by the adversary the trapdoors $((X^1_{\text{Enc}}, Y^1_{\text{Enc}}), \ldots, (X^L_{\text{Enc}}, Y^L_{\text{Enc}}))$ are generated. The distinguisher calculates the dimension of the vector space spanned by the X^i_{Enc}:

$$d := \dim(X^1_{\text{Enc}}, \ldots, X^L_{\text{Enc}}). \tag{2}$$

Depending on d it outputs:

$$\mathcal{D}(\mathcal{V}) = \begin{cases} 0 & \text{if } d \leq \lfloor \frac{2}{3} L \rfloor \\ 1 & \text{otherwise.} \end{cases}$$

In the following, we will calculate the adversarial advantage resulting from this definition. If $b = 0$, then L queries for the same keyword w_1 are calculated. The application of the LSH function h_1 and the keyed hash function f_{k_1} sets exactly one bucket in the Bloom filter to 1. All the L repetitions of the keyword w_1 result in this same Bloom filter $Q^1 = \cdots = Q^L$. For the $X^i = (x^i_j)$ generated from this Bloom filter holds:

$$x^1_j = \cdots = x^L_j \quad \text{if } s_j = 0 \quad \text{for all } 1 \leq j \leq L.$$

Let $\mathsf{ones}(S)$ be the number of ones in S. So it follows: $\dim(\langle X^1, \ldots, X^L \rangle) \leq \mathsf{ones}(S)$. If $s_j = 1$, a random $t \in \mathbb{R}$ is added to $x_j/2$. Since the t are sampled independently, it follows that the probability of $X^{k+1} \in \langle X^1, \ldots, X^k \rangle$ for some $k < \mathsf{ones}(S)$ is zero. Therefore, it holds with probability one:

$$\dim(X^1, \ldots, X^L) = \mathsf{ones}(S).$$

Since M_1 is invertible, it is rank-preserving and it holds with probability one:

$$d = \dim(X^1_{\text{Enc}}, \ldots, X^L_{\text{Enc}}) = \dim(M_1^{-1} X^1, \ldots, M_1^{-1} X^L) = \mathsf{ones}(S).$$

S is chosen uniformly from $\{0,1\}^L$. Consequently $\mathsf{ones}(S)$ is distributed according to a binominal distribution:

$$d = \mathsf{ones}(S) \sim \text{Binom}(L, 1/2).$$

By the normal approximation of the binominal distribution, it follows:

$$\mathsf{P}\left(d \leq \left\lfloor \frac{2}{3} L \right\rfloor \,\middle|\, b = 0 \right) \to 1 \text{ for } L \to \infty.$$

If $b = 1$, then the w_i are chosen such that $h_1(w_i)$ are pairwise different. Applying the pseudo-random function f_{k_1} these keywords result in Bloom filters $Q^1, \ldots, Q^L$. Since $r = 1$, these Bloom filters have exactly one bucket set to 1. Due to the pseudo-randomness of f, we get for the dimension $d' = (Q^1, \ldots, Q^L)$ the following probability distribution:

$$\mathsf{P}(d' = k) = \frac{\binom{L}{k} k! \left\{ {L \atop k} \right\}}{L^L}.$$

Here, $\left\{ {L \atop k} \right\}$ denotes the Stirling number of the second kind [18]. The number d'' of different vectors which have $x_j = 1$, where $s_j = 0$ is described by the hypergeometric distribution:

$$\mathsf{P}(d'' = k) = \sum_{i=k}^{L} \mathsf{P}(d' = i) \cdot \text{HyperGeo}(L, i, L - \mathsf{ones}(S), k).$$

In the dimensions, where $s_j = 1$, the X^i span, as before, a $\mathsf{ones}(S)$-dimensional space. Therefore, it follows:

$$d = \dim(X^1_{\text{Enc}}, \ldots, X^L_{\text{Enc}}) = \dim(X^1, \ldots, X^L) = \mathsf{ones}(S) + d''.$$

Hence:

$$\mathsf{P}\left(d \leq \left\lfloor \frac{2}{3}L \right\rfloor \,\middle|\, b = 1\right) = \sum_{k=0}^{\lfloor \frac{2}{3}L \rfloor} \sum_{j=0}^{k} \sum_{i=k-j}^{\min(L,k)} \frac{L!i!\left\{ {L \atop i} \right\}}{2^L L^L (i-k+j)!(k-j)!(L-k)!(k-i)!} \to 0$$

for $L \to \infty$. By the definition of the trace, the simulator does not receive any information about the similarity between the trapdoors. Therefore, it can do nothing better than guessing, whether the adversary has chosen $b = 0$ or $b = 1$. It follows:

$$\mathrm{Adv}_{\mathcal{A}} \to \frac{1}{2} \quad \text{for } L \to \infty.$$

The advantage of the adversary is not only asymptotically perfect. A computation shows that for a key length as low as $L = 8$ the adversary already has an advantage of 0.36. For a key length of $L = 32$ the advantage $\mathrm{Adv}_{\mathcal{A}}$ is greater than 0.48.

5.2.4 Similarity of Documents

The similarity of trapdoors is not captured by the trace definitions (neither the original nor the new one). We exploited this fact for the last attack. Likewise, the similarity of documents is not captured by the trace definition. This can be exploited for an attack as well. In this attack the adversary does not choose any queries. The documents are chosen in the same way as the queries in the previous attack. The attack then works completely analogously to the last attack.

5.3 Discussion

Since Wang et al. provide a detailed security proof for their scheme, one might ask where this proof goes wrong. Within their scheme they use the k-nearest neighbour encryption scheme of Wong et al. [40] for result ranking. Unfortunately, they assume that this encryption provides indistinguishability. The used k-nearest neighbour method is mainly a classic polygraphic substitution cipher, also known as Hill cipher, combined with some random padding. This explains the lack of indistinguishability.

6 Scheme of Hu and Han

The scheme of Hu and Han [21] is a modification of the scheme of Suga, Nishide and Sakurai [37]. As Suga, Nishide and Sakurai used another security definition than Hu and Han, the attack of this chapter does not apply to their scheme. Similarity in the scheme of Hu and Han is expressed by wildcard characters which can stand for an arbitrary number of characters. The index generation as well as the trapdoor generation of the scheme use an encoding which maps a keyword to a set of strings. As the search terms can contain wildcards, they define this mapping for words which may contain wildcards. Wildcards cannot occur during the index generation.

Let $w_1, \ldots, w_{|w|}$ denote the characters of w. Let a be the position of the first wildcard character, or $|w| + 1$ if no such character exists. Let b be the position of the last wildcard

character, or 0 if no such character exists. Let $G_n(w)$ be the set of all n-grams of a string w. Let $G(w) \coloneqq \{G_n(w) \mid 1 \leq n \leq |w|\}$. Furthermore, let $W(w)$ be the set of strings generated when splitting the string w at the positions of the wildcard characters, or w, when no wildcard characters are present in w. Now define sets A, B and C as follows:

$$\begin{aligned} A &= \{i \| w_i \mid 1 \leq i < a\} \\ B &= \{(i - |w| - 1) \| w_i \mid b < i \leq |w|\} \\ C &= \bigcup_{s \in W(w)} \{0 \| x \mid x \in G(s)\}. \end{aligned}$$

Here, $\|$ denotes string concatenation. Each feature w of a stored document is encoded in the following set of subfeatures:

$$T' = A \cup B \cup C.$$

From T' we get the final embedding by adding $l_{\max} - |T|$ additional random strings.

For each different feature exactly one Bloom filter is created. In this Bloom filter all the elements of T are encoded with r hash functions each. The index then consists of pairs of Bloom filters and an encrypted list of the documents containing this keyword. A trapdoor for a feature w is generated by applying the mapping as defined above and generating the corresponding Bloom filter.

We would like to explain this embedding with an example. The sets A, B and C for the keyword CAT are:

$$\begin{aligned} A &= \{1\|\mathrm{C}, 2\|\mathrm{A}, 3\|\mathrm{T}\} \\ B &= \{-3\|\mathrm{C}, -2\|\mathrm{A}, -1\|\mathrm{T}\} \\ C &= \{0\|\mathrm{C}, 0\|\mathrm{A}, 0\|\mathrm{T}, 0\|\mathrm{CA}, 0\|\mathrm{AT}, 0\|\mathrm{CAT}\}. \end{aligned}$$

So we get:

$$T' = \{1\|\mathrm{C}, 2\|\mathrm{A}, 3\|\mathrm{T}, -3\|\mathrm{C}, -2\|\mathrm{A}, -1\|\mathrm{T}, 0\|\mathrm{C}, 0\|\mathrm{A}, 0\|\mathrm{T}, 0\|\mathrm{CA}, 0\|\mathrm{AT}, 0\|\mathrm{CAT}\}.$$

After adding the required number of random dummy entries, this set is encoded in a Bloom filter. We provide another example for the query generation. Assume that the user wants to search for C$*$T. Then the corresponding sets are:

$$\begin{aligned} A &= \{1\|\mathrm{C}\} \\ B &= \{-1\|\mathrm{T}\} \\ C &= \{0\|\mathrm{C}, 0\|\mathrm{T}\}. \end{aligned}$$

This gives us the complete query:

$$T' = \{1\|\mathrm{C}, -1\|\mathrm{T}, 0\|\mathrm{C}, 0\|\mathrm{T}\}.$$

The search result of a query consists of the documents which have the highest number of Bloom filter entries in common with this query.

6.1 Security Definition

The authors claim that their scheme satisfies IND-CKA security for indexes as given by Goh [16]. However, this is not true. We will show here that the given scheme does not satisfy IND-CKA security. For this purpose, let us first recall almost verbatim the security definition of IND-CKA for indexes [16]. It uses a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Definition 2 (IND-CKA security for indexes)**.**

Setup: The challenger $\mathcal{C}$ creates a set S of q words and gives this to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ chooses some subsets from S. This collection of subsets is called S^* and is returned to C. Upon receiving S^*, the challenger $\mathcal{C}$ runs Keygen to generate the master key K, and for each subset in S^*, the challenger $\mathcal{C}$ encodes its contents into an index using BuildIndex. Finally, $\mathcal{C}$ sends all indexes with their associated subsets to the adversary $\mathcal{A}$.

Queries: The adversary $\mathcal{A}$ can query $\mathcal{C}$ on a word x and receives the trapdoor T_x for x. With T_x, the $\mathcal{A}$ can invoke SearchIndex on an index I to determine if $x \in I$.

Challenge: After making some Trapdoor queries, $\mathcal{A}$ decides on a challenge by picking a non-empty subset $V_0 \in S^*$, and generating another non-empty subset V_1 from S such that $|V_0 \setminus V_1| \neq 0$, $|V_1 \setminus V_0| \neq 0$, and the sum of the characters of the words V_0 is equal to that in V_1. Lastly, $\mathcal{A}$ must not have queried $\mathcal{C}$ for the trapdoor of any word in the symmetric difference $V_0 \triangle V_1$. Next, $\mathcal{A}$ gives V_0 and V_1 to $\mathcal{C}$ who chooses $b \in \{0, 1\}$, invokes BuildIndex(V_b,K) to obtain the index I_{V_b} for V_b, and returns I_{V_b} to $\mathcal{A}$. The challenge for $\mathcal{A}$ is to determine b. After the challenge is issued, $\mathcal{A}$ is not allowed to query $\mathcal{C}$ for the trapdoors of any word $V_0 \triangle V_1$.

Response: The adversary $\mathcal{A}$ eventually outputs a bit b', representing its guess for b.

The advantage of $\mathcal{A}$ in winning this game is defined as:

$$\text{Adv}_{\mathcal{A}} = |\mathsf{P}(b = b') - 1/2|,$$

where the probability is taken over $\mathcal{A}$ and $\mathcal{C}$'s coin tosses. We say that an adversary $\mathcal{A}$ (t, ϵ, q)-breaks an index if $\text{Adv}_{\mathcal{A}}$ is at least ϵ after $\mathcal{A}$ takes at most t time and makes q trapdoor queries to the challenger. We say that I is a (t, ϵ, q)-IND-CKA secure index if no adversary can (t, ϵ, q)-break it for any initial choice of S.

Hu and Han claim that their scheme is $(u, \epsilon, v/r)$-secure given that some pseudo-random function f is (u, ϵ, v)-secure. The authors do not state explicitly where they use this pseudo-random function f in their scheme. However, the only functions which could be reasonably modelled as pseudo-random functions are those hash functions, which are used for embedding strings in Bloom filters.

6.2 Attack

The attack described here does not depend on the properties of the hash functions used in the scheme. Our argument would even work when the used pseudo-random functions are perfect (i.e. modelled as random oracles). We will show that there exists an adversary with an advantage of:

$$\text{Adv}_{\mathcal{A}} = \frac{1}{2} - \frac{1}{2r}.$$

To achieve this advantage the adversary $\mathcal{A}$ only needs to perform a single trapdoor query. Furthermore, the adversary has a very moderate runtime.

Let $S = \{\text{A, AB, CD, EF}\}$. Let the adversary $\mathcal{A}$ choose $S^* \coloneqq S$. The challenger $\mathcal{C}$ generates the ciphertexts for all subsets of S^* and gives them to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ queries the challenger $\mathcal{C}$ for the Trapdoor T_A for the word A. This is the only trapdoor query the adversary $\mathcal{A}$ performs. The adversary $\mathcal{A}$ chooses the sets $V_0 = \{AB, EF\}$ and $V_1 = \{CD, EF\}$. This choice is permissible as the following statements hold:

- V_0 and V_1 are non-empty
- $|V_0| = |V_1|$
- $|V_0 \setminus V_1| \neq 0$ and $|V_1 \setminus V_0| \neq 0$
- The adversary $\mathcal{A}$ did not query the challenger $\mathcal{C}$ for any word from the symmetric difference $V_0 \triangle V_1 = \{AB, CD\}$.

Now the adversary $\mathcal{A}$ receives the encrypted index I_{V_b} of V_b and outputs a guess b' for b. As V_0 and V_1 both contain exactly two features, it follows that I_{V_b} contains exactly two Bloom filters. Let $I^i_{V_b}$ for $i \in \{1, 2\}$ denote these Bloom filters. The adversary $\mathcal{A}$ knows T_A. If many of the hashes from T_A are contained in I_{V_b} the adversary $\mathcal{A}$ can deduce that b was probably 0, otherwise b was probably 1. More concrete the adversary $\mathcal{A}$ outputs:

$$b' = \begin{cases} 0 & \text{if exists } i \in \{1, 2\} \text{ such that } \left|T_A \cap I^i_{V_b}\right| \geq 2r \\ 1 & \text{otherwise.} \end{cases}$$

The advantage of the adversary employing this strategy can be calculated as follows: It holds $|T_A| = l_{\max} > 3r$ as the set consists of r hashes belonging to each of the strings $\{1\|A, -1\|A, 0\|A\}$, and additionally random padding values. The index $I^0_{V_0}$ or $I^1_{V_0}$ contains the hashes belonging to $\{1\|A, 0\|A\}$. Therefore, it holds $|T_A \cap I_{V_0}| \geq 2r$.

Furthermore, $\left|I^i_{V_1}\right| = l_{\max} > 7r$ for $i \in \{1, 2\}$. As the hashed strings in $I^i_{V_1}$ for $i \in \{1, 2\}$ do not coincide with the strings hashed in T_A, the probability of each of these strings to be in the set of hashes of T_A is $\frac{1}{t}$ where t is the length of the Bloom filter. So it holds for $i \in \{1, 2\}$:

$$\mathsf{P}\left(\left|T_A \cap I^i_{V_1}\right| \geq 2r\right) = \sum_{n=2r}^{l_{\max} \cdot r} \binom{l_{\max} \cdot r}{n} \left(\frac{1}{t}\right)^n \left(\frac{t-1}{t}\right)^{l_{\max} \cdot r - n}.$$

By Markov's inequality applied on the binominal distribution it follows for $i \in \{1, 2\}$:

$$\mathsf{P}\left(\left|T_A \cap I^i_{V_1}\right| \geq 2r\right) \leq \frac{r \cdot l_{\max}}{2rt}.$$

It follows:

$$\mathsf{P}\left(\left|T_A \cap I^i_{V_1}\right| \geq 2r \text{ for some } i \in \{1, 2\}\right) < \frac{r \cdot l_{\max}}{rt}.$$

If $|T_A \cap I_{V_b}| < 2r$ it can be deduced with certainty that $b = 1$. So it holds for the advantage of the adversary $\mathcal{A}$ as defined above:

$$\mathrm{Adv}_{\mathcal{A}} = \left|\mathsf{P}(b = b') - \frac{1}{2}\right| > \frac{1}{2} - \frac{r \cdot l_{\max}}{2rt}. \tag{3}$$

If the searchable encryption scheme does not generate too much false-positive search results, the number of entries in the Bloom filter is much smaller than its size. Therefore, it follows that $r \cdot l_{\max} < t$. Plugging this in (3) yields:

$$\mathrm{Adv}_{\mathcal{A}} > \frac{1}{2} - \frac{1}{2r}.$$

It follows that the proposed scheme is insecure in the IND-CKA security model for any number of hash functions r which is larger than 1.

The edge case $r = 1$ is not relevant for practical applications because the low number of calculated hash values will lead to many false-positives. According to Hu and Han the use of $r = 1$ would lead to the (unacceptably high) false-positive rate of at least 50%. Furthermore, using different estimations we could still get, depending on the values of t and $l_{\max}$, a non-negligible advantage of the adversary $\mathcal{A}$ in the case $r = 1$.

7 Conclusion

For the practical application of fuzzy searchable encryption, a well-defined information leakage to the server is very important. We showed that the information leakage of three schemes is higher than it was claimed to be. Two of the schemes, those of Chuah and Hu [10] and Hu and Han [21], exhibited such a high leakage that they are of no practical use. The high leakage of these schemes is caused by Bloom filters which are used to protect confidentiality. The information leakage of the scheme of Wang et al. [38] is also higher than claimed. However, this information leakage still might be small enough for certain practical use cases.

Two of the examined schemes [21, 38] used security definitions invented and normally used for non-fuzzy searchable encryption. Fuzzy searchable encryption schemes inherently need to process more information about keywords than non-fuzzy searchable encryption schemes. Therefore, it would be surprising if a fuzzy scheme leaked the same amount of information as a non-fuzzy scheme.

In addition to the attacked fuzzy searchable encryption schemes, several more schemes have been proposed. These schemes have different security definitions and performance properties. It is still an open question which security definition for fuzzy searchable encryption is best suited for which practical use case. Therefore, at the moment it has to be evaluated on a case by case basis whether a scheme with suitable security properties and performance exists for a certain use case.

Furthermore, more research on security definitions is required. It is still an open question which asymptotic performance can be reached for a given security definition, and vice versa. In our opinion, it is promising to look for security definitions explicitly chosen for fuzzy searchable encryption. Based on Curtmola et al. [11], some custom security definitions for fuzzy search have been proposed [20, 28]. In those security definitions the search result quality directly correlates with the leakage of the schemes. It would be useful to find security definitions for fuzzy searchable encryption not showing this correlation.

Acknowledgment

This work was funded by the DFG under grant number Wi 4086/2-2.

References

[1] Burton H. Bloom. 'Space/Time Trade-offs in Hash Coding with Allowable Errors'. In: *Commun. ACM* 13.7 (1970), pp. 422–426. DOI: 10.1145/362686.362692.

[2] Alexandra Boldyreva and Nathan Chenette. 'Efficient Fuzzy Search on Encrypted Data'. In: *FSE 2014*. Vol. 8540. LNCS. Springer, 2014, pp. 613–633. DOI: 10.1007/978-3-662-46706-0_31.

[3] Christoph Bösch et al. 'A Survey of Provably Secure Searchable Encryption'. In: *ACM CSUR* 47.2 (2014), 18:1–18:51. DOI: 10.1145/2636328.

[4] Christoph Bösch et al. 'Conjunctive Wildcard Search over Encrypted Data'. In: *Secure Data Management 2011*. Vol. 6933. LNCS. Springer, 2011, pp. 114–127. DOI: 10.1007/978-3-642-23556-6_8.

[5] Michael Brenner, Henning Perl and Matthew Smith. 'Practical Applications of Homomorphic Encryption'. In: *SECRYPT 2012*. SciTePress, 2012, pp. 5–14.

[6] Ning Cao et al. 'Privacy-preserving multi-keyword ranked search over encrypted cloud data'. In: *INFOCOM 2011*. IEEE, 2011, pp. 829–837. DOI: 10.1109/INFCOM.2011.5935306.

[7] David Cash et al. 'Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation'. In: *NDSS 2014*. The Internet Society, 2014. DOI: 10.14722/ndss.2014.23264.

[8] David Cash et al. 'Leakage-Abuse Attacks Against Searchable Encryption'. In: *CCS 2015*. ACM, 2015, pp. 668–679. DOI: 10.1145/2810103.2813700.

[9] Yan-Cheng Chang and Michael Mitzenmacher. 'Privacy Preserving Keyword Searches on Remote Encrypted Data'. In: *Applied Cryptography and Network Security 2005*. Vol. 3531. LNCS. 2005, pp. 442–455. DOI: 10.1007/11496137_30.

[10] M. Chuah and W. Hu. 'Privacy-Aware BedTree Based Solution for Fuzzy Multi-keyword Search over Encrypted Data'. In: *ICDCS 2011 Workshops*. IEEE, 2011, pp. 273–281. DOI: 10.1109/ICDCSW.2011.11.

[11] Reza Curtmola et al. 'Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions'. In: *CCS 2006*. ACM, 2006, pp. 79–88. DOI: 10.1145/1180405.1180417.

[12] Dwyl Ltd. *english-words*. Commit: be341d3. GitHub repository. 2017. URL: https://github.com/dwyl/english-words (visited on 14/10/2018).

[13] William W. Cohen. *Enron email dataset*. May 2015. URL: https://www.cs.cmu.edu/~./enron/ (visited on 14/10/2018).

[14] Zhangjie Fu et al. 'Enabling Personalized Search over Encrypted Outsourced Data with Efficiency Improvement'. In: *Trans. Parallel Distrib. Syst* 27.9 (2016), pp. 2546–2559. DOI: 10.1109/TPDS.2015.2506573.

[15] Benjamin Fuller et al. 'SoK: Cryptographically Protected Database Search'. In: *SP 2017*. IEEE, 2017, pp. 172–191. DOI: 10.1109/SP.2017.10.

[16] Eu-Jin Goh. 'Secure Indexes'. In: *IACR Cryptology ePrint Archive* (2003). Report 2003/216.

[17] Google. *Misspellings of the name Britney Spears*. 2010. URL: https://www.google.com/jobs/britney.html. Available via: https://web.archive.org/web/20100208221904/https://www.google.com/jobs/britney.html (visited on 14/10/2018).

[18] Ronald L. Graham, Donald E. Knuth and Oren Patashnik. *Concrete mathematics – a foundation for computer science*. 2nd ed. Addison-Wesley, 1994.

[19] Florian Hahn and Florian Kerschbaum. 'Searchable Encryption with Secure and Efficient Updates'. In: *CCS 2014*. ACM, 2014, pp. 310–320. DOI: 10.1145/2660267.2660297.

[20] Daniel Homann, Christian Göge and Lena Wiese. 'Dynamic Similarity Search over Encrypted Data with Low Leakage'. In: *Security and Trust Management 2017*. Vol. 10547. LNCS. Springer, 2017, pp. 19–35. DOI: 10.1007/978-3-319-68063-7_2.

[21] Changhui Hu and Lidong Han. 'Efficient wildcard search over encrypted data'. In: *Int. J. Inf. Sec.* 15.5 (2016), pp. 539–547. DOI: 10.1007/s10207-015-0302-0.

[22] Piotr Indyk and Rajeev Motwani. 'Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality'. In: *STOC 1998*. ACM, 1998, pp. 604–613. DOI: 10.1145/276698.276876.

[23] Mohammad Saiful Islam, Mehmet Kuzu and Murat Kantarcioglu. 'Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation'. In: *NDSS 2012*. The Internet Society, 2012.

[24] Seny Kamara and Charalampos Papamanthou. 'Parallel and Dynamic Searchable Symmetric Encryption'. In: *FC 2013*. Vol. 7859. LNCS. Springer, 2013, pp. 258–274. DOI: 10.1007/978-3-642-39884-1_22.

[25] Seny Kamara, Charalampos Papamanthou and Tom Roeder. 'Dynamic searchable symmetric encryption'. In: *CCS 2012*. ACM, 2012, pp. 965–976. DOI: 10.1145/2382196.2382298.

[26] Georgios Kellaris et al. 'Generic Attacks on Secure Outsourced Databases'. In: *CCS 2016*. ACM, 2016, pp. 1329–1340. DOI: 10.1145/2976749.2978386.

[27] Bryan Klimt and Yiming Yang. 'Introducing the Enron Corpus'. In: *Conference on Email and Anti-Spam 2004*. 2004.

[28] Mehmet Kuzu, Mohammad Saiful Islam and Murat Kantarcioglu. 'Efficient Similarity Search over Encrypted Data'. In: *ICDE 2012*. IEEE, 2012, pp. 1156–1167. DOI: 10.1109/ICDE.2012.23.

[29] Jin Li et al. 'Fuzzy Keyword Search over Encrypted Data in Cloud Computing'. In: *INFOCOM 2010*. IEEE, 2010, pp. 441–445. DOI: 10.1109/INFCOM.2010.5462196.

[30] Weipeng Lin et al. 'Revisiting Security Risks of Asymmetric Scalar Product Preserving Encryption and Its Variants'. In: *ICDCS 2017*. IEEE, 2017, pp. 1116–1125. DOI: 10.1109/ICDCS.2017.20.

[31] Chang Liu et al. 'Fuzzy keyword search on encrypted cloud storage data with small index'. In: *Cloud Computing and Intelligence Systems 2011*. IEEE, 2011, pp. 269–273. DOI: 10.1109/CCIS.2011.6045073.

[32] Muhammad Naveed. 'The Fallacy of Composition of Oblivious RAM and Searchable Encryption'. In: *IACR Cryptology ePrint Archive* (2015). Report 2015/668.

[33] Muhammad Naveed, Seny Kamara and Charles V. Wright. 'Inference Attacks on Property-Preserving Encrypted Databases'. In: *CCS 2015*. ACM, 2015, pp. 644–655. DOI: 10.1145/2810103.2813651.

[34] David Pouliot and Charles V. Wright. 'The Shadow Nemesis: Inference Attacks on Efficiently Deployable, Efficiently Searchable Encryption'. In: *CCS 2016*. ACM, 2016, pp. 1341–1352. DOI: 10.1145/2976749.2978401.

[35] Dawn Xiaodong Song, David A. Wagner and Adrian Perrig. 'Practical Techniques for Searches on Encrypted Data'. In: *SP 2000*. IEEE, 2000, pp. 44–55. DOI: 10.1109/SECPRI.2000.848445.

[36] Emil Stefanov, Charalampos Papamanthou and Elaine Shi. 'Practical Dynamic Searchable Encryption with Small Leakage'. In: *NDSS 2014*. The Internet Society, 2014. DOI: 10.14722/ndss.2014.23298.

[37] Takanori Suga, Takashi Nishide and Kouichi Sakurai. 'Secure Keyword Search Using Bloom Filter with Specified Character Positions'. In: *ProvSec 2012*. Vol. 7496. LNCS. Springer, 2012, pp. 235–252. DOI: 10.1007/978-3-642-33272-2_15.

[38] Bing Wang et al. 'Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud'. In: *INFOCOM 2014*. IEEE, 2014, pp. 2112–2120. DOI: 10.1109/INFOCOM.2014.6848153.

[39] Cong Wang et al. 'Achieving usable and privacy-assured similarity search over outsourced cloud data'. In: *INFOCOM 2012*. IEEE, 2012, pp. 451–459. DOI: 10.1109/INFCOM.2012.6195784.

[40] Wai Kit Wong et al. 'Secure kNN computation on encrypted databases'. In: *SIGMOD 2009*. ACM, 2009, pp. 139–152. DOI: 10.1145/1559845.1559862.

[41] Bin Yao, Feifei Li and Xiaokui Xiao. 'Secure nearest neighbor revisited'. In: *ICDE 2013*. IEEE, 2013, pp. 733–744. DOI: 10.1109/ICDE.2013.6544870.

[42] Xingliang Yuan et al. 'Enabling Privacy-Assured Similarity Retrieval over Millions of Encrypted Records'. In: *ESORICS 2015*. Vol. 9327. LNCS. Springer, 2015, pp. 40–60. DOI: 10.1007/978-3-319-24177-7_3.

[43] Yupeng Zhang, Jonathan Katz and Charalampos Papamanthou. 'All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption'. In: *USENIX Security 2016*. USENIX, 2016, pp. 707–720.

[44] Zhenjie Zhang et al. 'Bed-tree: an all-purpose index structure for string similarity search based on edit distance'. In: *SIGMOD 2010*. ACM, 2010, pp. 915–926. DOI: 10.1145/1807167.1807266.