

1-1-2022

## Privacy preserving scheme for document similarity detection

AYAD ABDULSADA

SALAH AL-DARRAJI

DHAFER HONI

Follow this and additional works at: <https://journals.tubitak.gov.tr/elektrik>



Part of the [Computer Engineering Commons](#), [Computer Sciences Commons](#), and the [Electrical and Computer Engineering Commons](#)


---

### Recommended Citation

ABDULSADA, AYAD; AL-DARRAJI, SALAH; and HONI, DHAFER (2022) "Privacy preserving scheme for document similarity detection," *Turkish Journal of Electrical Engineering and Computer Sciences*: Vol. 30: No. 3, Article 10. <https://doi.org/10.55730/1300-0632.3801>  
Available at: <https://journals.tubitak.gov.tr/elektrik/vol30/iss3/10>

This Article is brought to you for free and open access by TÜBİTAK Academic Journals. It has been accepted for inclusion in Turkish Journal of Electrical Engineering and Computer Sciences by an authorized editor of TÜBİTAK Academic Journals. For more information, please contact [academic.publications@tubitak.gov.tr](mailto:academic.publications@tubitak.gov.tr).

## Privacy preserving scheme for document similarity detection

Ayad I. Abdulsada\*, Salah Al-Darraji, Dhafer G. Honi

Computer Science Department, College of Education for Pure Sciences, University of Basrah, Basrah, Iraq

Received: 05.04.2021

Accepted/Published Online: 03.09.2021

Final Version: 21.03.2022

**Abstract:** The problem of detecting similar documents plays an essential role for many real-world applications, such as copyright protection and plagiarism detection. To protect data privacy, the new version of such a problem becomes more challenging, where the matched documents are distributed among two or more parties and their privacy should be preserved. In this paper, we propose new privacy-preserving document similarity detection schemes by utilizing the locality-sensitive hashing technique, which can handle the misspelled mistakes. Furthermore, the keywords' occurrences of a given document are integrated into its underlying representation to support a better ranking for the returned results. We introduced a new security definition, which hides the exact similarity scores towards the querying party. Extensive experiments on real-world data illustrate that our proposed schemes are efficient and accurate.

**Key words:** Document similarity, local sensitive hashing, multiparty computing, privacy preserving

### 1. Introduction

Detecting similar documents [1] is a hot topic that received a lot of interest. Particularly, the exponential growth of digital data requires efficient file system management. Such a system can be improved when similar documents can be grouped and indexed together. Similarly, the efficiency of web crawlers can be improved by eliminating all pages that are similar to their indexed versions [2]. Furthermore, document similarity detection constitutes a basic technology for detecting plagiarism and double submission in academic institutes.

Typically, document files are decentralized among several parties. So, evaluating document similarity becomes a hard process, and even harder when data privacy should be preserved. In this paper, we focus on this distributed setting and show how to evaluate document similarity without violating data privacy.

To see the importance of our work, consider the following examples. Health care offices need to find similar attributes in the medical diagnosis information of their patients' records. To do so, a secure protocol is required to perform such a task. Otherwise, patients' privacy could be compromised severely. In the same matter, intelligence agencies may be asked to identify suspected persons who have similar attributes under different names in their collections. Disclosing persons' records to the other parties also involves privacy risks. To handle such an issue, the participating agencies run a secure protocol that identifies the closed records for the provided query, and only the most similar records are revealed. Another application, to the setting of our problem, is detecting the double submission of scientific articles to multiple venues simultaneously. For example, to check whether the submitted article violates the submission policy, the conference committees want jointly to detect similar articles without revealing their confidential contents.

\*Correspondence: ayad.abdulsada@uobasrah.edu.iq

To solve the mentioned problems, the privacy-preserving similar document detection (PPSDD) technique is utilized, which is a subbranch of the general secure two-party computation problem [3]. In its overall setting, PPSDD considers the execution of a secret computing protocol among two parties, each one holding a confidential textual collection. Both parties want to identify any similar documents within their particular collections. The original version of this problem is used commonly to detect duplicate files [4] in the web environment or the distributed setting. To preserve data privacy, no party wants to disclose its private textual files to the other party.

Documents are represented in the existing solutions of PPSDD in three different ways: vectors, sets, and multisets. Vector-based schemes [5–9] generate a binary vector for each textual document, where the inner product or the Jaccard similarity measures are adopted to capture similarity scores between two vectors. However, such schemes incur high computational and communication costs as the size of vectors increase. Furthermore, existing schemes require the two involved parties to identify a global vocabulary of either keyword [5, 6] or  $N$ -gram sets [7].

Set-based schemes, such as [10], extract from each document a descriptive set. A secure intersection protocol is required to identify the intersection size of two private sets without breaking data privacy. However, such schemes do not consider keyword frequencies and thus show poor relevance ranking.

Documents under multiset-based schemes are transformed either into long binary vectors or descriptive sets [11], which also inherit the disadvantages of those schemes.

In this paper, we describe a new method based on local sensitive hashing LSH [12]. LSH is a hashing technique that hashes similar items into the same buckets (i.e hash values) with a higher probability than the dissimilar items. The keyword set of a given textual document is translated into bigram vectors. LSH technique is utilized to hash keyword vectors into a constant size vector, where hash values are used to determine the vector locations that store keyword frequencies. This allows us to (i) evaluate the similarity over small vectors rather than large ones, and (ii) render the similarity calculation to execute a secure dot product protocol over two private vectors. To further improve the security, we enhance the basic scheme to protect the similarity scores of the matched documents. In fact, our enhanced scheme returns only the index of the best-matched document, without revealing the remaining similarity scores. This is the first PPSDD scheme that achieves this goal.

To overcome the deficiencies of the previous schemes, we can summarize the design goals of our schemes as follows:

1. Efficiency: Our proposed schemes employ a new document transforming method that customizes the vector size according to the requirement of our schemes without identifying a shared vocabulary.
2. Accuracy improvement: The proposed schemes are designed to ensure better satisfaction for the relevant documents to the provided query by integrating keyword occurrences into the generated vectors.
3. Privacy-preserving: Our schemes are designed to perform PPSDD between two parties without compromising the privacy of the underlying documents. The privacy requirements are described in the following:
  - Collection privacy: The contents of documents that are to be matched should be protected from the querying party.

- Query privacy: The underlying query information including keywords and their occurrences should not be revealed to the other party.
- Similarity scores: Our enhanced scheme is designed to prevent the querying party from identifying the similarity scores of the matched documents.

Our main contributions can be summarized as follows:

1. In contrast to existing vector-based schemes [5–7], which generate long vectors, our schemes utilize the LSH technique to release efficient PPSDD with changeable size vector regardless of the number of keywords within the document. Users can customize vector sizes according to their preferences.
2. A new method is designed to represent documents effectively, where misspelled keywords can still generate a vector that is highly similar to the correct one.
3. Our schemes enhance the security of some previous schemes [5–7], where the need for sharing the global vocabulary is eliminated.
4. Document similarity calculation is efficient, where a secure inner product protocol is sufficient to compare two numerical vectors.
5. A real-world data set is used to investigate the effectiveness and efficiency of our proposed schemes.

The remainder of this paper is organized as follows. Section 2 surveys previous work on PPSDD schemes. Section 3 demonstrates the basic methods and the cryptographic tools utilized during our schemes. Section 4 provides a formal description of the PPSDD problem along with its security definition. Section 5 shows, in detail, the basic steps of our basic scheme, Section 6 presents the enhanced scheme that returns the index of the best matching document. Section 7 introduces the security proof of our proposed schemes. Section 8 reports the results of our experiments and Section 9 concludes our work.

## 2. Related work

Related works can be divided into the following topics.

### 2.1. Vector space-based PPSDD schemes

The first solution for the PPSDD problem is presented in [5], where each document is transformed into a numerical vector, where its size is determined by the total number of vocabulary keywords. To generate correct vectors, the two participant parties need to run a secure protocol to determine the vocabulary, which includes the common keywords of their collections. Document vector stores at the entry of each keyword its corresponding normalized term frequency. The similarity of two vectors is evaluated using a cosine similarity measure, which requires a secure inner product protocol to maintain data privacy.

The performance of [5] is further improved in [6], where document vectors are grouped into  $k$  clusters using clustering algorithms. Once providing a document query, its nearest cluster is identified, and only the similarity scores of  $n_k$  documents within that cluster are computed. The performance of [5, 6] degrades dramatically as vocabulary space increases. Vector space-based schemes have the disadvantage of missing the local similarities, such as the overlapped segments of texts. To handle such a problem, Jiang et al. [7] utilized the  $N$ -gram approach, which has the advantage of detecting local similarities. The  $N$ -gram set of a given document includes

all of its substrings of size  $N$ . Specifically, they used a 3-gram set. This scheme reveals the global  $N$ -gram set to the other parties, which is a security risk since one party can utilize this leak to check the existence of a given word in the collection of the other party. BuyrukBILEN and Bakiras [8] reduce the communication and computational costs of the previously mentioned schemes using SimHash [13] method, where a short binary vector (usually of 64 bits) is generated for each document, and a secure Hamming distance protocol is developed to measure the similarity between two binary vectors. Furthermore, such a work introduced a new security definition for their enhanced scheme, where only the binary scores are returned to the querying party. The study in [9] has utilized the oblivious garbled Bloom filter protocol [14] to capture the intersection of two vectors. Under such a work, each document is represented as a garbled Bloom filter, which is a fixed size vector. However, oblivious transfer-based solutions [15] incur a high computational cost as the number of documents increases. Recently, Schoppmann et al. [21] have proposed a secure scheme for classifying collaborated documents, where a k-nearest neighbor classifier is applied to compute the secure similarity of textual documents.

## 2.2. Set-based PPSDD schemes

Blundo et al. [10] developed an efficient yet secure scheme to evaluate the similarity of two sets. The intersection cardinality of two sets is evaluated privately using the protocol of De Cristafaro et al. [16]. The authors applied such a scheme to detect similar documents, where a Q 3-gram set is generated for each document. The size of this set is reduced by MinHash method [1] into a fixed number  $v$  of elements. Jaccard similarity of two sets  $S_1$  and  $S_2$  is approximated as  $JC(S_1, S_2) = \frac{|S_1 \cap S_2|}{v}$ .

## 2.3. Multiset-based schemes

The set-based PPSDD schemes are only suitable for traditional sets, where frequencies of the  $N$ -gram elements are ignored. However, ignoring such valuable information sacrifices the ranking functionality. So, the new version of PPSDD has shifted towards multiset-based schemes, where the occurrence of each element is preserved. Forman and Samanthula [11] proposed a secure yet efficient scheme to compute the intersection cardinality of two multisets, which is considered as a subproblem of PPSDD. They utilized Bloom filter to handle multiset data. To do so, documents are transformed into several sets with different frequency levels as follows. Set one includes all keywords that appear at least one time, set two includes all keywords that appear at least two times, etc. Each set is hashed using some predefined hash functions into its corresponding Bloom filter [17]. Bloom filters for different frequency levels are concatenated together for producing a long multiset Bloom filter. The number of intersected bits among two Bloom filters and the size of each one are utilized to approximate the intersection size of their respective documents. Unfortunately, this scheme represents each document as a long vector to consider the term frequencies. On the other hand, our proposed schemes introduced a new method for aggregating document terms and their corresponding frequencies into a short vector.

However, all of the abovementioned schemes, except the enhanced scheme of [8], are vulnerable to reveal the actual similarity scores of the compared documents for the queried party. Specifically, our enhanced scheme hides such information and returns only the index of the most matched document. Furthermore, our work customizes the size of document vectors via a tuning parameter, which provides a better balance between system complexity and accuracy results according to the users' preferences. A comparison of our schemes with existing works is given in Table .

**Table .** Comparison with existing schemes. Rep. is the underlying document representation method. Leakage is the information leaked to  $P_2$  party. Matching comm. is the size of trapdoor sent from  $P_2$ .  $n$  is the number of documents in the collection of  $P_1$ .  $\Sigma$  is the global vocabulary of keywords in collections  $P_1$  and  $P_2$ .  $\Delta$  is the N-gram set of  $P_1$ .  $\zeta$  is the size of SimHash vector, which is much smaller than  $|\Sigma|$  and  $|\Delta|$ .  $\alpha_1, \dots, \alpha_n$  are the actual similarity scores, while  $\beta_1, \dots, \beta_n$  are binary similarity scores.  $\hat{t}$  is a user defined threshold.  $v$  is the size of MinHash set, where  $v < |\Delta|$ .  $\gamma$  is the size of vectors for our proposed schemes, where  $\gamma < |\Sigma|$ .  $c$  is the number of frequency levels.

Scheme	Rep.	Leakage	Doc. size	Matching time/Comm.
Term vector (TV) [5]	Numerical vector	$\alpha_1, \dots, \alpha_n$	$ \Sigma $	$O(n), O( \Sigma )$
Enhanced term vector (ETV) [6]	Numerical vector	$\alpha_1, \dots, \alpha_k, \alpha_1, \dots, \alpha_{n_k}$	$ \Sigma $	$O(kn_k), O( \Sigma )$
N-gram [7]	Binary vector	$\alpha_1, \dots, \alpha_n, \Delta$	$ \Delta $	$O(n), O( \Delta )$
Basic SimHash [8]	Binary vector	$\alpha_1, \dots, \alpha_n$	$\zeta$	$O(n), O(\zeta)$
Enhanced SimHash [8]	Binary vector	$\beta_1, \dots, \beta_n$	$\zeta$	$O(n(\hat{t} + 1)), O(n(\hat{t} + 1))$
EsPRESSo [10]	Set	$\alpha_1, \dots, \alpha_n$	$v$	$O(n), O(v)$
Multiset [11]	Multiset	$\alpha_1, \dots, \alpha_n$	$c.\gamma$	$O(n), O(c.\gamma)$
Our basic scheme	Numerical vector	$\alpha_1, \dots, \alpha_n$	$\gamma$	$O(n), O(\gamma)$
Our Enhanced scheme	Numerical vector	$\max_{ind}\{\alpha_i : \forall i = 1, \dots, n\}$	$\gamma$	$O(n), O(\gamma)$

### 3. Background

This section introduces a brief description of the local sensitive hashing method and the basic cryptographic tools adopted during our designs.

#### 3.1. Local sensitive hashing LSH

LSH is a hashing technology used for solving the problem of nearest neighbor search, where high-dimensional items are hashed into low-dimensional items without compromising their relative distance. Interestingly, unlike traditional hashing functions, LSH hashes similar items into the same buckets with high probability. Given the hash family  $\mathcal{H}$ , the distance metric  $dist$ , two probabilities  $p_1$  and  $p_2$  s.t.  $p_1 > p_2$ , and two distances  $r_1$  and  $r_2$  s.t.  $r_1 < r_2$  then the hash family  $\mathcal{H}$  is  $(r_1, r_2, p_1, p_2)$ -sensitive if for any two points  $x, y$ , and any  $h \in \mathcal{H}$  it satisfies:

$$if \ dist(x, y) \leq r_1 : Pr[h(x) = h(y)] \geq p_1 \tag{1}$$

$$if \ dist(x, y) \geq r_2 : Pr[h(x) = h(y)] \leq p_2 \tag{2}$$

where  $dist(x, y)$  is the distance measure between  $x$  and  $y$ .

The  $(r_1, r_2, p_1, p_2)$ -sensitive hash family  $\mathcal{H}$  can be amplified into a new  $(r_1, r_2, 1 - (1 - p_1^r)^d, 1 - (1 - p_2^r)^d)$ -sensitive hash family  $\mathcal{G}$ . First, a new  $(r_1, r_2, p_1^r, p_2^r)$ -sensitive hash family  $\mathcal{F}$  is generated, where its hash functions  $f \in \mathcal{F}$  are constructed by randomly choosing  $r$  different hash functions  $h_1, h_2, \dots, h_r$ , where  $f(x) = [h_1(x), \dots, h_r(x)]$ . Under such a construction  $f(x) = f(y)$  if  $h_i(x) = h_i(y)$  for  $i = 1, \dots, r$ . Next, the family  $\mathcal{F}$  is amplified into the  $(r_1, r_2, 1 - (1 - p_1^r)^d, 1 - (1 - p_2^r)^d)$ -sensitive hash family  $\mathcal{G}$ . Each hash function  $g \in \mathcal{G}$  is

defined by randomly selecting  $d$  hash functions  $f_1, \dots, f_d$  from  $\mathcal{F}$ , where  $g$  is defined as  $g(x) = \text{mix}(f_1, \dots, f_d) \bmod \gamma$ , where  $\gamma$  is a user defined parameter used to tune the number of buckets. Under such a construction,  $g(x) = g(y)$  if  $f_i(x) = f_i(y)$  for at least one value of  $i = 1, \dots, d$ . The proper selection of  $r$  and  $d$  pushes  $p_1$  and  $p_2$  to be close to 0 and 1, respectively.

**Stable distributions:** Hashing families can be constructed according to the distance metric *dist*. In this work, we adopted Euclidean metric distance. For this metric, the hash function  $h_{(a,b)}() : \mathcal{R}^t \rightarrow \mathcal{N}$  can be defined as  $h_{(a,b)}(x) = \lfloor \frac{ax+b}{W} \rfloor$ , where  $a$  is a  $t$  dimensional vector, whose elements are chosen randomly from a stable distribution [18] (Gaussian distribution is used here),  $b$  is a real random number belongs to  $[0, \dots, W]$ , and  $W$  is a fixed integer that determines the number of bins.

### 3.2. Cryptography tools

Below we describe briefly the basic cryptographic tools utilized in our schemes.

#### 3.2.1. Homomorphic encryption

Homomorphic encryption allows anyone to perform certain arithmetic operation *op* on the encryption of plaintexts  $m_1$  and  $m_2$ , by evaluating an efficient operation on their corresponding ciphertexts  $c_1$  and  $c_2$  without knowing the private key. The popular RSA encryption scheme supports multiplicative homomorphic operation (i.e. *op* is multiplication operation). When the encryption scheme supports any *op*, we refer to it as fully homomorphic, that was designed first by Gentry [19] and improved later by many works [20, 22]. See [23] for more information.

In this paper, we are interested in additively homomorphic encryption schemes. Paillier encryption [24] is a well-known public key cryptographic scheme that implements additive homomorphic and constant multiplication homomorphic properties. Let  $E(\cdot)$  denote the encryption operation,  $E(\cdot)^{-1}$  denote the multiplicative inverse, the homomorphic properties of this scheme can be illustrated as follows:

$$E(m_1 + m_2) = E(m_1).E(m_2)$$

$$E(m_1 - m_2) = E(m_1).E(m_2)^{-1}$$

$$E(m_1 m_2) = E(m_1)^{m_2}.$$

Note that, Pillier's scheme is semantically secure, i.e. there is no probabilistic polynomial time (PPT) adversary that can distinguish its ciphertext from truly random numbers. Its security is based on the decisional composite residuosity assumption. The ciphertext of the message  $m_1$  is referred as  $[[m_1]]$ , which can also be refreshed using a new random value  $r \in \mathbb{Z}_N^*$  as  $[[m_1]].r^N$ .

#### 3.2.2. Secure multiparty computation

In secure multiparty computation [3], several parties compute jointly a function of their private inputs without violating the privacy of their inputs. Yao [25] proposed the garbled circuit technique to solve the millionaires problem in the case of two-party. Using this technique any function can be evaluated privately by using the Boolean circuit description of that function. In particular, the truth table of each logic gate in the circuit is encrypted and garbled by one party and outsourced to the other party. The latter evaluates the circuit by using inputs from the garbling party. Unfortunately, this technique is not efficient when the circuit has a

large number of inputs and logical gates, since each input requires the execution of an oblivious transfer (OT) protocol [15], which consumes more computation overhead. On the other hand, the size of the circuit determines the communication cost. Yao's technique was extended later [26] to multiparty computation. Over the past years, many improvements have been developed to enhance the efficiency of garbled circuits [27, 28]. Despite these improvements, garbled circuits are still impractical solutions for multiparty computations. Participant parties can be corrupted by adversaries of two types.

1. **Semihonest adversaries:** In this type, the corrupted parties follow precisely the protocol steps. However, the adversary utilizes the messages received to the party to learn further private information.
2. **Malicious adversaries:** In this type, the active adversary asks the corrupted parties to deviate from the protocol steps. Therefore, the correctness of the results should be ensured.

### 3.2.3. Secure comparison protocols

Secure comparison protocols are interactive cryptographic tools that allow two parties with private inputs  $x, y$  to determine whether  $x > y$  or not while preserving their respective inputs. Such protocols constitute building blocks for several real-world applications such as secure classification [29]. Veugen [30] designed an efficient protocol to compare two encrypted integers  $[[x]], [[y]]$  s.t.  $[[0]] < [[x]]$  and  $[[y]] \leq [[2^\ell]]$  without decryption. Such protocol, in turn, employs DGK comparison protocol [31] to compare two private unencrypted integers.

The basic idea of this solution is to compute  $[[z]] = [[2^\ell]] - [[x]] + [[y]]$  of  $(\ell + 1)$ -bit and inspect its most significant bit  $z_\ell$ . If it is 1, then  $[[x]] \geq [[y]]$ , otherwise  $[[x]] < [[y]]$ . Thus it is enough to compute the bit  $z_\ell$  for deciding whether  $[[x]] \geq [[y]]$  or not. Note that  $z_\ell$  can be calculated by dividing  $[[z]]$  by  $2^\ell$ .

Veugen's protocol assumes two parties:  $P_1$  and  $P_2$ . The inputs of  $P_1$  are two encrypted numbers  $[[x]]$  and  $[[y]]$ , its output is  $[[\delta]] = ([[x]] < [[y]]) = [[1 - z_\ell]]$ . The security definition of this protocol prevents  $P_2$  from knowing the original values  $x, y$ , and prevents  $P_1$  from knowing the comparison result  $\delta$ . Figure 1 shows the basic steps of Veugens [30] protocol.

## 4. Problem description and security definition

### 4.1. Problem description

Our proposed schemes considers two parties:  $P_1$ , which holds a collection of  $n$  textual documents  $\mathcal{D} = \{d_1, d_2, \dots, d_n\}$ . Each document  $d_i \in \mathcal{D}$  is described as a set of keywords and their corresponding occurrences  $\{(w_i, t f_i)\}$ .  $P_2$  holds a collection of  $m$  textual documents  $\mathcal{Q} = \{q_1, q_2, \dots, q_m\}$ , which are described in the same way. At each time,  $P_2$  chooses one of its documents  $q_i$  and wants to evaluate its similarity scores against all documents of  $P_1$ 's collection in a privacy-preserving manner.

### 4.2. Security definition

In this work, we designed two schemes with different security levels. Our basic scheme has a security level that is identical to the security supported by all previous PPSDD schemes, that is:  $P_2$  knows all the similarity scores  $\{\alpha_i : \forall i = 1, \dots, n\}$  between its query  $q$  and the collection  $\mathcal{D}$  of  $P_1$ , while  $P_1$  knows nothing. The security of our enhanced scheme supports a stronger level for  $P_1$ , where the only index of the maximum score is returned to  $P_2$ . Formally speaking,  $P_2$  knows  $\max_{ind}\{\alpha_i : \forall i = 1, \dots, n\}$ . Like all the existing PPSDD schemes, we assume that both parties are semihonest.



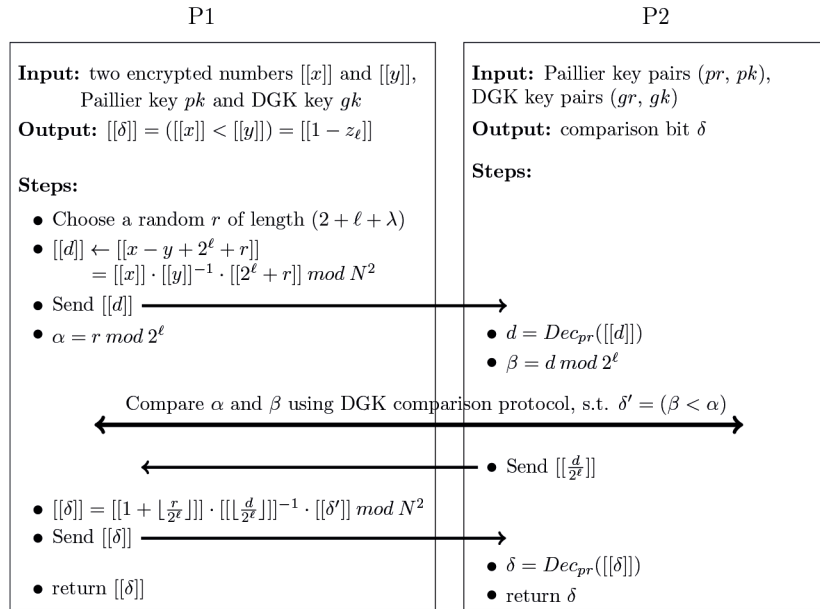
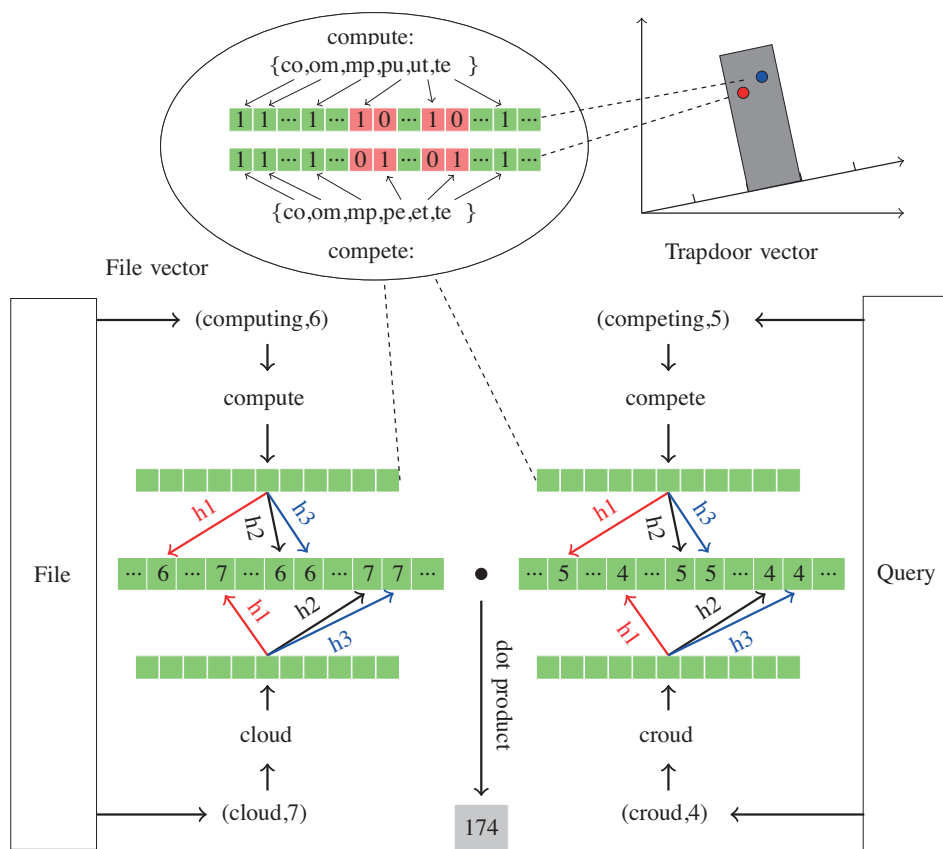


Figure 1. Comparing two encrypted integers.

### 5. Basic scheme

In this section, we describe our basic scheme that discloses similarity scores for each one of  $P_1$ 's documents to  $P_2$ . Our scheme generates a constant vector  $V_i$  per document  $d_i$ . The vector  $V_i$  represents all the keywords of  $d_i$  and their corresponding occurrences as a numerical vector of size  $\gamma$ . To allow for similarity matching, we first transform each keyword  $w_j$  into a bigram vector and then utilize LSH hash functions to hash this vector for determining the locations of  $V_i$  that store the keyword frequency  $tf_j$  corresponding to  $w_j$ . The basic operations are demonstrated in Figure 2 and described below:

1. Data Preprocessing: Before generating document vectors, each party executes a preprocessing step to describe each of its documents as a set of keywords and their corresponding occurrences  $\{(w_j, tf_j)\}$ . This step includes some operations borrowed from the information retrieval field [32], such as lower case conversion, stop word elimination, and stemming. Stemming operation converts each keyword to its basic stem. For example, all the keywords *compute*, *computing*, and *computation* are all converted to the word *compute*. Such operation ensures the best matching between the provided query and the stemmed documents.
2. Bigram vector generation: Recall that LSH functions hash the close vectors to the same integer value with high probability. Thus, we need to transform string keywords into vectors, such that LSH functions can use them. A bigram set is extracted from a given keyword, which includes all the contiguous sequences of 2 characters that appeared in that keyword. For instance, the bigram set of the keyword *cryptography* is  $\{cr, ry, yp, pt, to, og, gr, ra, ap, ph, hy\}$ . Typically, such a set requires a vector of  $26^2$ -bits to be represented. If the bigram element exists in the bigram set, and then its corresponding entry in the binary vector is set to 1. Otherwise, it is set to 0. Such a representation can recover the misspelled keywords and still be able to generate keyword vectors that are very similar to the vector of the correct keyword. The bigram vector is the key idea that enables the direct application of the LSH technique.



**Figure 2.** The proposed basic scheme for PPSDD, query encryption is omitted for better presentation.

3. Document vector construction: In this paper, we utilize LSH functions to construct document vectors. LSH functions will hash similar items into the same buckets with high probability. The use of LSH functions in constructing the per document based vectors is the key to support similarity matching. Therefore, documents and queries now are described as numerical vectors instead of string keywords. We use  $k$  hash functions  $\{g_1, \dots, g_k\}$  form the hash family  $\mathcal{G}$  as explained in subsection 3.1. Each document vector  $V_i$  is initialized as  $\gamma$  zero elements  $\langle v_0, \dots, v_\gamma \rangle$ . When document keywords are transformed into their respective bigram vectors, they are ready to be hashed into the document vector. Each keyword  $w_j$  is hashed using the  $k$  hash functions to get  $k$  locations  $\{l_1, \dots, l_k\}$ . Then we set keyword frequency  $tf_j$  to the locations  $v_{l_1}, v_{l_2}, \dots, v_{l_k}$ . When different keywords are mapped into the same locations, the average value of their term frequencies is stored.
4. Documents matching: As Figure 2 illustrates, the final document vector is a constant numerical vector that describes all of the string keywords appearing in that document. Since query and document vectors are constructed in the same manner, a similarity score between them can be evaluated. If a given document contains the same keyword(s) of the query, then their corresponding vectors will include closed frequencies at the same locations, thus the inner product between them will be high. So, we can use this measure to evaluate the similarity between the two documents.
5. Query encryption: The document vector hides the underlying keyword set and its size. This is because

each element of that vector is calculated by multiplying the bigram vector for each keyword with a set of random vectors and then mapping the results to a set of bucket numbers. However, the document vector is constructed in a deterministic way. Therefore, the same collection of  $b$  hash functions are used for generating all document vectors. Deterministic construction enables the adversary party to learn if the same keywords are included in the provided query by utilizing a keywords-chosen attack. To protect the query vector, we encrypt it before its revealing. However, since the other party is asked to perform a similarity matching, a homomorphic encryption method is adopted. Specifically, Paillier encryption scheme is used to encrypt individually all the elements of the query vector. Hence, the other party can calculate the encrypted similarity scores using a secure dot product between the encrypted query and each one of its document vectors. Note that the dot product is just the addition of the partial multiplications of two numerical vectors.

To protect its trapdoor,  $P_2$  encrypts every element of its query vector  $QV$  and sends  $\gamma$  ciphertexts to  $P_1$ .  $P_1$  is not able to decrypt these ciphertexts but it can blindly compute the required multiplication and addition operations. Particularly, for each textual document  $d_i$  in its collection,  $P_1$  initializes the ciphertext of the similarity score to 0. Next, it proceeds over the  $\gamma$  elements of the corresponding document vector  $V_i$ . At each iteration, it simply utilizes the multiplication property of Paillier cryptosystem to multiply two elements, then it adds the result to the encrypted score. After the computation of all documents,  $P_2$  returns the encrypted scores to  $P_1$ . Finally,  $P_1$  employs its private key to get the exact matching scores and determine the best matching documents to  $q$ . Construction 1 illustrates the details of our basic scheme.

**Discussion:** Our proposed method for transforming document keywords uses the bigram vector. Such a method has the ability to handle the following mistakes:

1. Misspelled letters: For example, the keywords `secure` and `secude` are transformed into  $\{\text{se, ec, cu, ur, re}\}$  and  $\{\text{se, ec, cu, ud, de}\}$ , respectively, where they differs only in two bigrams.
2. Anagram keywords: In this case, two different keywords are formulated by the same letters but with different orders. For example, our method can distinguish between the keywords `dessert` and `stressed`, since they have completely different bigram sets.
3. Keywords in the same root: Our scheme can distinguish between `computing` and `compute`, since it utilizes stemming operation during the preprocessing step.

Notice that if document  $d_i$  is seen in the retrieved results  $R$  of the search query  $q$ , then we have five cases: **Case1:** when the exact keywords of  $q$  are a subset of the keywords of  $d_i$ , then  $d_i$  would appear in  $R$ , since we use the same LSH functions  $g_1, \dots, g_k$  during the index construction step of both  $d_i$  and  $q$ . Therefore, the corresponding indexes of  $d_i$  and  $q$  will share 1 at the same positions leading to a maximum dot product, and hence  $d_i$  will be included in  $R$ . **Case2:** when keyword  $w'$  of  $q$  is misspelled and differs slightly from keyword  $w$  of  $d_i$  (i.e.  $d(w, w') \leq r_1$ ), and  $g_i(w) = g_i(w'), \forall i = 1, \dots, k$ , then the dot product will reach also the maximum value. **Case3:** when  $d(w', w) \leq r_1$  but  $g_j(w') \neq g_j(w)$ , then the inner product will be affected. Recall that each  $g_j$  function is designed using  $d$  internal functions  $f_1, \dots, f_d$ , thus the probability of occurring the event  $g_j(w') \neq g_j(w)$  will decrease when  $d$  increases. **Case4:** when keyword  $w'$  of  $d_i$  differs significantly from keyword  $w$  of  $d_i$  (i.e.  $d(w', w) > r_1$ ), then there is a low probability that  $w$  and  $w'$  will be hashed to the same locations. Thus, the inner product will be low. **Case5:** when  $d(w', w) > r_2$  but their corresponding

**Algorithm 1:** Basic scheme.

**Input :**  $\lambda$ : security parameter,  $\mathcal{D} = \{d_1, \dots, d_n\}$  document collection of  $P_1$ ,  $q$  document query of  $P_2$ .

**Output:**  $P_2$  gets  $\alpha_1, \alpha_2, \dots, \alpha_n$  similarity scores.

**Setup.**

$P_2$ : Uses  $\lambda$  to generate the key pair  $(pk, pr)$  for Paillier cryptosystem, chooses  $k$  random hash functions  $\{g_1, g_2, \dots, g_k\}$  form the sensitive hash family  $\mathcal{G}$ , and selects  $\gamma$ : the size of the resulted document vectors. Shares  $(pk, g_1, g_2, \dots, g_k, \gamma)$  with  $P_1$ .

**Document embedding.**

$P_1$ :

```

for  $i \leftarrow 1$  to  $n$  do
   $\{(w_1, tf_1), (w_2, tf_2), (w_z, tf_z)\} \leftarrow \text{Preprocessing}(d_i)$ 
  for  $j \leftarrow 1$  to  $z$  do
     $Bv_j = \text{Bigram\_vector}(w_j)$ ;
  end for
   $V_i = \text{LSH}(Bv_1, \dots, Bv_z, \gamma, g_1, \dots, g_k, tf_1, \dots, tf_z)$ ;
end for

```

**Trapdoor generation.**

$P_2$ :

```

 $\{(w_1, tf_1), (w_2, tf_2), (w_z, tf_z)\} \leftarrow \text{Preprocessing}(q)$ ;
for  $j \leftarrow 1$  to  $z$  do
   $Bv_j = \text{Bigram\_vector}(w_j)$ ;
end for
 $QV = \text{LSH}(Bv_1, \dots, Bv_z, \gamma, g_1, \dots, g_k, tf_1, \dots, tf_z)$ ;
for  $j \leftarrow 1$  to  $\gamma$  do
   $c_j = \text{Enc}_{pk}(QV_j)$ ;
end for
Sends  $\langle c_1, \dots, c_\gamma \rangle$  to  $P_1$ .

```

**Secure matching.**

$P_1$ :

```

for  $i \leftarrow 1$  to  $n$  do
   $[[\alpha_i]] = \text{Enc}_{pk}(0)$ 
  for  $j \leftarrow 1$  to  $\gamma$  do
     $[[\alpha_i]] = [[\alpha_i]] \cdot (c_j^{V_{ij}})$ ;
  end for
end for

```

**end for**

Sends  $[[\alpha_1]], [[\alpha_2]], \dots, [[\alpha_n]]$  to  $P_2$ .

$P_2$ : uses  $pr$  to decrypt  $[[\alpha_1]], [[\alpha_2]], \dots, [[\alpha_n]]$ .

hashes are matched. This case is defined as false positive (FP) and it is introduced in our schemes due to two reasons: the locality sensitive hashing functions (introduces  $p_2^k$  FP) and the vector representation (introduces  $(1 - \frac{1}{\gamma})^{ks}$  FP, where  $s$  is the number of inserted keywords,  $k$  is the number of LSH functions,  $\gamma$  is the size of produced vector for each document).

**6. Enhanced scheme**

The basic scheme allows  $P_2$  to learn the exact similarity scores for every document  $d_i$  in  $P_1$ 's collection. The security definition of such a scheme is equivalent to the definition of all the current PPSDD schemes. However,

providing  $P_2$  with this information may allow it to build some fake queries that reveal more information about  $P_1$ 's collection. Even though such an attack is not trivial with document vectors, it is still important to hide the actual similarity scores for preventing any potential attacks. For this reason, in this section, we present our enhanced scheme that protects similarity scores and returns only the index of the best matched document. This is the first PPSDD scheme in the literature that supports this strong security level. Particularly, the two involved parties  $P_1$  and  $P_2$  agree on a similarity threshold  $\tau$ , which determines whether the provided query is relevant to a given document or not. Documents are considered relevant only when their inner products are greater than  $\tau$ . Our enhanced scheme returns only the index  $ind$  of the best matching document, where  $\alpha_{ind} > \tau$ . Observing that larger threshold implies returning the most similar document. Construction 2 illustrates the details of the secure matching of the enhanced scheme.

To achieve this goal, we utilized the protocol of [33], where it preserves a linear number of comparisons. First,  $P_1$  applies a random permutation  $\pi$  on the encrypted similarity scores to prevent  $P_2$  from learning the order of such values. Then both parties start to compare the first encrypted score  $[[\alpha_i]]$  with the encryption of  $\tau$  using Veugen's protocol (see subsection 3.2.3). The result is that  $P_2$  knows the index  $ind$  of the two compared values. They proceed to compare  $[[\alpha_{ind}]]$  with  $[[\alpha_2]]$ . The iteration of such a process over all encrypted scores allows  $P_2$  to know the permuted value of  $ind$ . Finally,  $P_1$  performs  $\pi^{-1}(ind)$  to get the right result. Note,  $P_1$  still be able to learn the minimum of every two compared scores. To handle such a security issue,  $P_2$  utilizes the refresh property of Paillier cryptosystem to reencrypt the minimum value after each comparison.

## 7. Security analysis

Our schemes are designed under the setting of secure two-party computing and their security is defined according to the model of semihonest parties (see subsection 4.2). We need to show that such schemes perform private computation. To do so, we use the simulation approach [34], which is a popular approach for proving the security of two-party computing protocols. The key idea of this approach is illustrated as follows: Let the view of any party be the received messages during the execution of the protocol. We say that the protocol is secure if, for each party, there exists a simulator (a PPT algorithm) that can simulate efficiently the messages it receives from other parties, given only the inputs and outputs of that party. This means that if the view of a given party can be simulated only from its input and output, then no additional information is leaked from the received messages.

Formally speaking, a two-party protocol  $\pi$  evaluates securely a deterministic polynomial function  $\text{Func}$  for two parties with the security parameter  $\lambda$ . The function  $\text{Func}$  receives two inputs:  $x \in \{0, 1\}^*$  from the first party and  $y \in \{0, 1\}^*$  from the second party, and generates two outputs:  $\text{Func}_1$  for the first party and  $\text{Func}_2 \in \{0, 1\}^*$  for the second party.

The notation  $\text{View}_1(x, y)$  (resp.,  $\text{View}_2(x, y)$ ) is used to describe the view of the first party (resp., the second party) during the execution of  $\pi$ , where  $\text{View}_i(\lambda, x, y) = (1^\lambda, z, \text{mes}_1^i, \dots, \text{mes}_t^i, r^i)$  where  $z \in (x, y)$ ,  $\text{mes}_j^i, \forall j = 1, \dots, t$  are the  $t$  messages that party  $i$  receives, and  $r^i$  is the random tape of party  $i$ .

We say that  $\pi$  computes  $\text{Func}$  securely in the presence of static semihonest adversaries if there exists PPT algorithms  $\mathcal{S}_1$  and  $\mathcal{S}_2$  such that

$$\begin{aligned} \{\mathcal{S}_1(x, \text{Func}_1(x, y))\}_{x,y} &\equiv_c \{\text{View}_1(x, y)\}_{x,y} \\ \{\mathcal{S}_2(y, \text{Func}_1(x, y))\}_{x,y} &\equiv_c \{\text{View}_2(x, y)\}_{x,y} \end{aligned}$$

**Algorithm 2:** Secure matching of the enhanced scheme.

---

**Secure matching.**

$P_1$ :

**for**  $i \leftarrow 1$  **to**  $n$  **do**

$[[\alpha_i]] = \text{Enc}_{pk}(0)$ ;

**for**  $j \leftarrow 1$  **to**  $\gamma$  **do**

$[[\alpha_i]] = [[\alpha_i]] \cdot (c_j^{V_{ij}})$ ;

**end for**

**end for**

Pick a random permutation  $\pi$  over  $\{1, \dots, n\}$ ;

Set  $[[max]] \leftarrow \text{Enc}_{pk}(\tau)$ ;

**for**  $i \leftarrow 1$  **to**  $n$  **do**

$[[\delta]] \leftarrow \text{Secure\_Comparison}([[ \alpha_{\pi(i)} ]], [[max]])$ ;

Choose two random integers  $\rho_i, \varrho_i \in (0, 2^\ell)$ .

Set  $[[max']] \leftarrow [[max]] \cdot [[\rho_i]]$ ;

Set  $[[\omega]] \leftarrow [[\alpha_{\pi(i)}]] \cdot [[\varrho_i]]$

Send  $[[\omega]]$  and  $[[max']]$  to  $P_2$

$P_2$ : **if**  $\delta = 1$  **then**

$ind \leftarrow i$

$[[v]] \leftarrow \text{Refresh}([[ \omega ]])$

**end if**

**else**

$[[v]] \leftarrow \text{Refresh}([[max']])$

**end if**

Send  $[[v]]$  to  $P_1$

$P_1$ : Compute  $[[max]] \leftarrow [[v]] \cdot ([[1]]^{-1})^{\rho_i} \cdot [[\delta_i]]^{-\varrho_i}$

**end for**

$P_2$ : Send  $ind$  to  $P_1$

$P_1$ : Set  $ind \leftarrow \pi^{-1}(ind)$ , and return  $ind$  to  $P_2$

---

The notation  $\equiv_c$  means that for all PPT adversaries, the two sides are computationally indistinguishable. For simplicity,  $\lambda$  will be omitted from views and security proof. We apply this approach to prove our basic scheme. In our basic scheme, the input of  $P_2$  is a numerical vector  $QV$  of size  $\gamma$  and its output is  $n$  similarity scores:  $\alpha_1, \dots, \alpha_n$ . The view of  $P_2$  is  $\text{View}_2 = \{QV, \gamma, [[\alpha_1]], [[\alpha_2]], \dots, [[\alpha_n]], pk, pr, rc\}$ , where  $rc$  is the random tape for Paillier encryptions. The simulator  $\mathcal{S}_2$  knows the public key and the output of  $P_2$ . Thus,  $\mathcal{S}_2$  can simulate the real view of  $P_2$  by utilizing the inputs  $([[\alpha_1]], [[\alpha_2]], \dots, [[\alpha_n]], pk, pr)$  as follows:

1. Uses  $pk$  to encrypt  $\alpha_1, \alpha_2, \dots, \alpha_n$  to obtain  $[[\hat{\alpha}_1]], [[\hat{\alpha}_2]], \dots, [[\hat{\alpha}_n]]$
2. Generate  $\hat{rc}$  as a  $\gamma$  random tape for Paillier encryptions
3. Returns  $\{QV, \gamma, [[\hat{\alpha}_1]], [[\hat{\alpha}_2]], \dots, [[\hat{\alpha}_n]], pk, pr, \hat{rc}\}$

Notice that  $[[\hat{\alpha}_1]], [[\hat{\alpha}_2]], \dots, [[\hat{\alpha}_n]]$  and  $[[\alpha_1]], [[\alpha_2]], \dots, [[\alpha_n]]$  are indistinguishable since Paillier is semantically secure cryptosystem.

The inputs of  $P_1$  are numerical vectors:  $V_1, \dots, V_n$  of size  $\gamma$  and its output is nothing. The real view of  $P_1$  is  $\text{View}_1 = \{V_1, \dots, V_n, pk, \gamma, c_1, \dots, c_\gamma, rc\}$ . The simulator  $\mathcal{S}_1$  uses  $(pk, \gamma, c_1, \dots, c_\gamma)$  to simulate  $\text{View}_1$  as follows:

1. Encrypts 0 by Paillier  $\gamma$  times:  $\hat{c}_1, \hat{c}_2, \dots, \hat{c}_\gamma$
2. Generate  $\hat{rc}$  as a  $\hat{\gamma}$  random tape for Paillier encryptions
3. Returns  $\{V_1, \dots, V_n, pk, \gamma, \hat{c}_1, \dots, \hat{c}_\gamma, rc\}$

The enhanced scheme utilizes a subprotocol to identify the index of the document that has the best matching. The security of such a protocol is explained in [33].

## 8. Experimental results

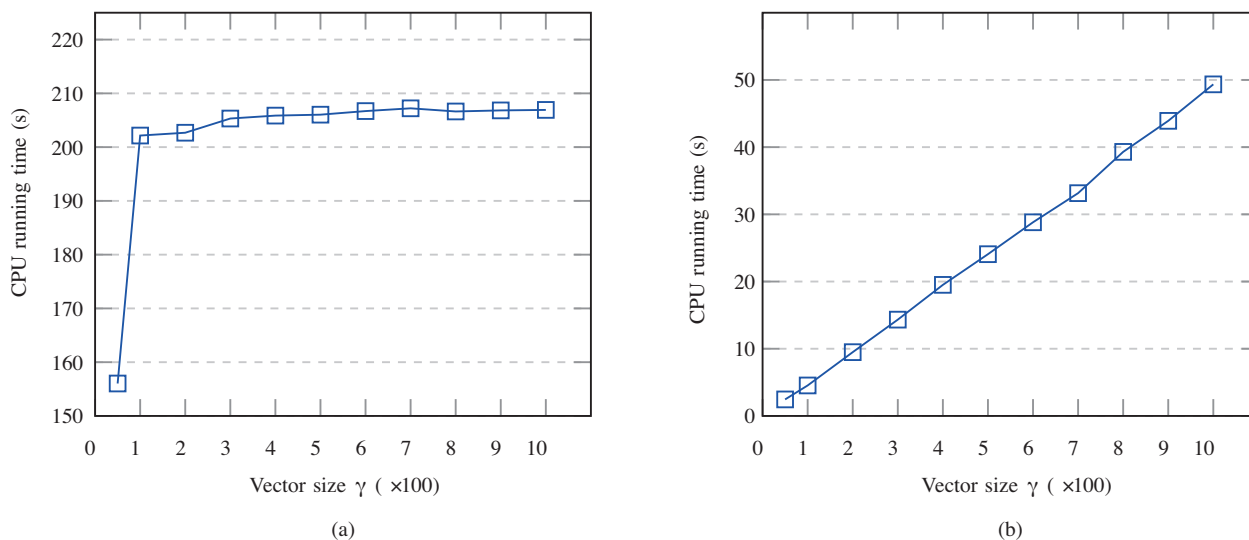
In this section, we estimate the performance of our proposed schemes in terms of matching accuracy and indexing efficiency. A real-data collection from Wikipedia articles is used to test our two schemes. In fact, 100 main pages are selected from various topics. For each topic, we picked 10 of its history versions. Thus, the total number of textual documents in the collection is 1000. First, the documents are preprocessed by: stop words removal, lower case conversion, and stemming with Porter algorithm [35]. The total number of keywords in the entire collection is 4733, and the average number of keywords per document is 102. We set  $r = 8$ , and  $d = 30$  for LSH, and set  $\gamma = 700$  for document vector size. Our schemes were implemented using a machine of 2.7 GHz Intel corei7 CPU, with a 64-bits Windows 7 operating system and 8 GB of RAM. We used JAVA programming language. Big numbers are handled using BigInteger Java library. Security parameter of  $\lambda = 1024$ -bits is used for Paillier and DGK cryptosystems. We used  $\ell = 22$ -bits to represent the compared integer values. We compared our scheme with the following state-of-the-art schemes: The term vector (TV) [5], enhanced term vector (ETV) [6] (we used  $k$ -means algorithm for document clustering and set the number of clusters  $k = 5$ ), and SimHash [8].

### 8.1. Efficiency investigation

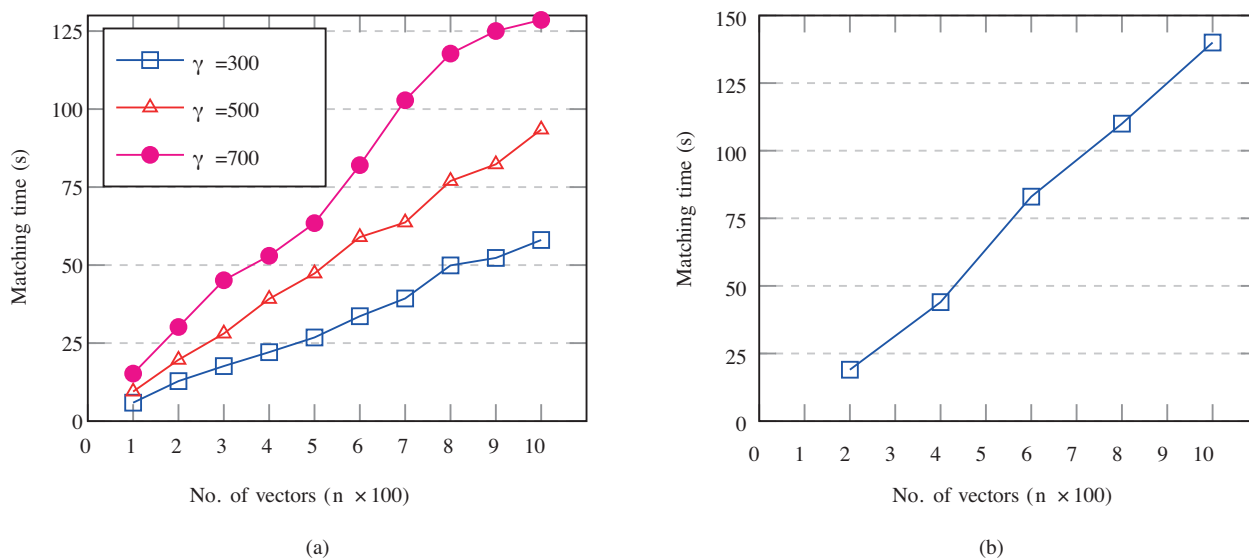
We investigate the efficiency of our proposed schemes in terms of time and communication costs.

1. Document vector construction cost: Note that document vector construction involves three basic operations: data preprocessing, bigram vector generation, and document vector construction. Figure 3a illustrates the CPU running time for generating document vectors for the entire collection with variable vector sizes  $\gamma$ . As expected the running time grows as the vector size increases.
2. Trapdoor generation cost: Trapdoor is generated in the same way as for document vectors. The exception is that it includes an encryption step to protect the privacy of its underlying document. Figure 3b reports the CPU running time for encrypting a single document trapdoor with variable sizes.
3. Matching time: We measure the matching time for our basic scheme, which compares the provided trapdoor through secure dot product operation against the whole documents of the other party. Figure 4a reports the matching time costs for variable document size values  $\gamma$ . Notice that longer document vectors require immediately, as expected, more computational time.

We also measure the matching cost for our enhanced scheme, which requires matching the provided trapdoor against all documents and then selecting the index of the most matched one securely. For this experiment, we set the parameters to their default values. Figure 4b shows the CPU execution time of our enhanced scheme. Notice that this scheme incurs more matching time, mainly due to the additional cryptographic operations involved during the maximum index selection step.



**Figure 3.** Generation and encryption time. (a) Document vector generation time. (b) Trapdoor encryption time.



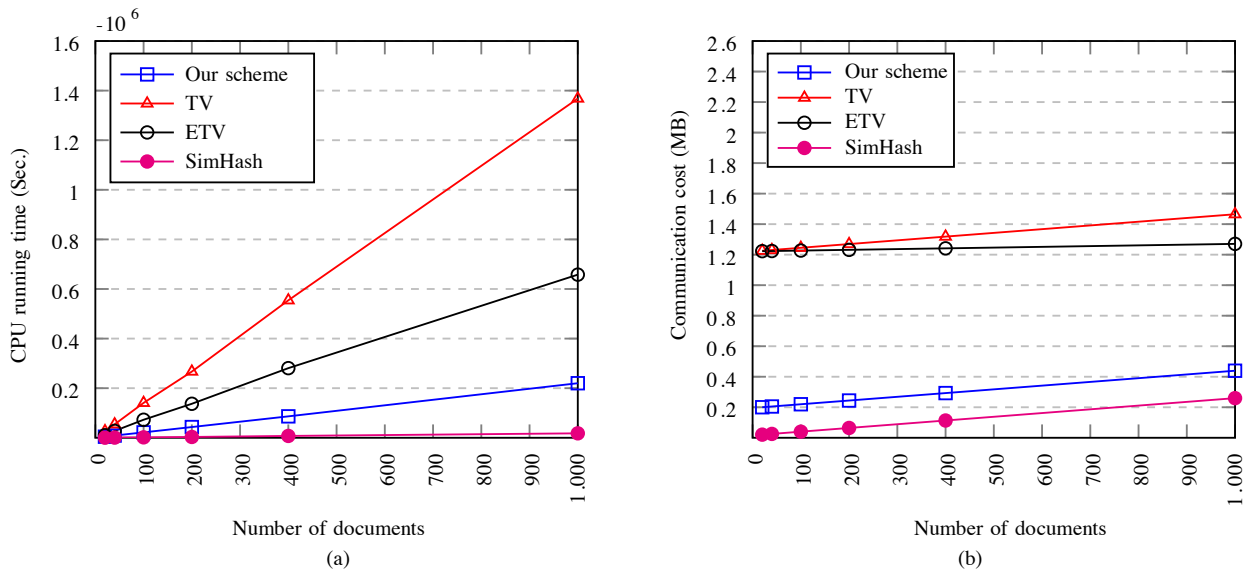
**Figure 4.** Document matching time. (a) Our basic scheme. (b) Our enhanced scheme.

4. Computation cost comparison: In this experiment, we compare the time cost of our scheme against previous schemes. Specifically, we measure the average CPU time of the cryptographic protocols for the 100 unique document queries. Each run includes the time needed to describe the document query, encrypt it, and match it against all documents  $n$ . We set the size  $\gamma$  of generated vectors in our scheme to 800, the size  $|\Sigma|$  of TV scheme to 5000 (since it has the best execution time), and the size  $\zeta$  of SimHash binary vector to 64. Figure 5a reports the CPU time required to compare one document against a collection of variable size  $n$ . SimHash incurs the lowest computation cost among all competitors, mainly due to its short vectors. TV demands a very high computational cost since it requires the encryption and matching of longer numerical vectors. ETV requires less computational cost than TV since it uses clustering method



for improving matching efficiency. Our method is significantly faster than the TV and ETV because it involves the computation of shorter vectors.

5. Communication cost: Figure 5b shows the communication cost for various schemes. Obviously, SimHash outperforms both competitor methods considerably, incurring a communication cost that is at least 18 times smaller under all settings. For example, to compare one document against a collection of  $n = 1000$  documents, requires 0.26 MB of data communication for SimHash, 0.44 MB for our basic scheme, 1.27 MB for ETV, and 1.46 MB for TV.



**Figure 5.** Resources comparison. (a) CPU running time. (b) Communication cost.

## 8.2. Effectiveness evaluation

Effectiveness is investigated in terms of document retrieval accuracy for the proposed document representation method. Particularly, we employed the precision metric to observe the accuracy of the results. Notice that our scenario requires sending back the top- $K$  relevant documents instead of returning all the relevant documents. Thus, our concern is on precision rather than recall. Let  $F$  denotes the set of relevant documents for the provided query, and  $R$  denotes the set of retrieved documents, then precision is equal to  $\frac{|R \cap F|}{|R|}$ . Our experiments use every article in the collection as a query, and relevant documents are those documents in the same topic as the query. The size of  $R$  is controlled by retrieving only the top- $K$  documents, where  $K$  ranges from 5 to 15.

1. Effect of LSH hash functions: In the following experiment, we test the effect of the number of hash functions  $b$  on the result accuracy for variable  $\gamma$  values. Figure 6a depicts that the average precision increases as  $b$  increases. This is because more LSH functions lead to more variety in generated document vectors, and so few nonrelevant documents in the collection are returned, which leads to high precision. The retrieval accuracy of the proposed scheme is higher than 40% for all settings.

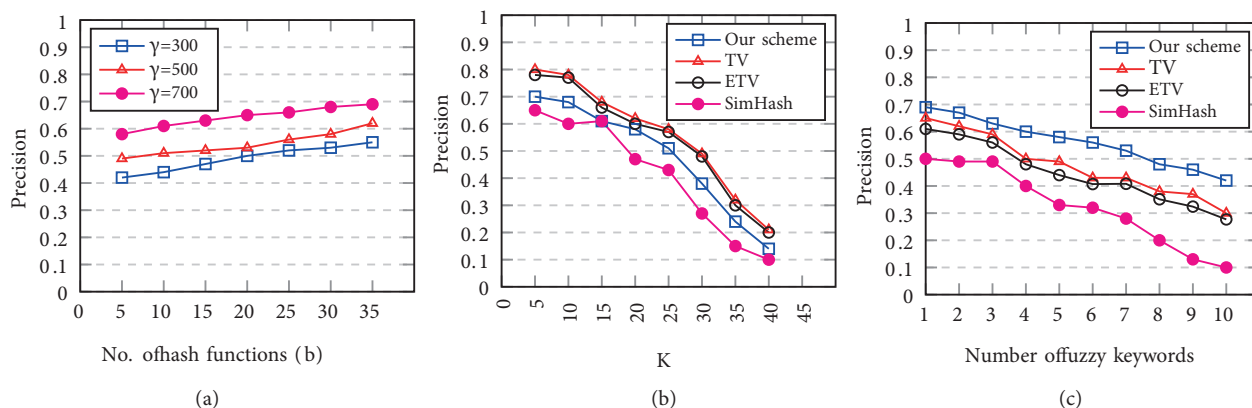


Figure 6. Precision comparison.

2. Document retrieval comparison: We compare the document retrieval accuracy of our proposed scheme against some of the previous schemes. The goal is to test the underlying document description method of our scheme. The retrieval accuracy of the involved schemes is evaluated for variable  $K$  values. The values of  $\gamma$  and  $b$  parameters of our scheme are set to 700 and 35, respectively. For SimHash scheme the size of the vector is set to 64. Figure 6b shows the average retrieval accuracy for various schemes. In all schemes, the accuracy of results decreases when  $K$  increases. As expected, the TV scheme has the best overall accuracy since it uses the standard method, in the literature of information retrieval, for describing documents. Notice that using the clustering method reduces slightly the precision of ETV. Our scheme sacrifices some accuracy for better computation and communication overheads, but it performs very well for  $K$  values lower than 20. SimHash shows the worst accuracy since it utilizes significantly shorter binary vectors to describe documents.
3. Fuzzy matching: One important feature for comparing the underlying description methods of various schemes is to test their ability to recover the misspelled (fuzzy) keywords in the document query. To generate fuzzy document queries, we randomly pick several unique keywords and slightly modify them to get the fuzzy keywords. For each keyword, we replace all its occurrences, in the query document, with the corresponding fuzzy keyword. Figure 6c shows the precision of the fuzzy matching for different schemes with a variable number of fuzzy keywords. In this experiment, we set  $K$  to 5. For all schemes, the precision decreases as the number of fuzzy keywords, in the query, increases. This is because the nonrelevant retrieved documents will be accumulated when the number of fuzzy keywords grows. Clearly, the precision of our proposed scheme outperforms the other schemes, since it uses a better method for the keyword transformation, which is based on the bigram set. On the other hand, TV, ETV, and SimHash schemes could not tolerate fuzzy keywords well, since they use high-level representation methods that deal with whole keywords instead of bigram sets, making them unsuitable to distinguish fuzzy keywords.

To prove the effectiveness improvement of our proposed scheme in terms of handling the fuzzy keywords we utilized statistical significance testing, which is a popular method to estimate how well an improvement in a new system reflects a real improvement between two systems, as opposed to fake improvement because of the selection of samples. Specifically, we used Student's paired t-test as an instance of significance test since it is used extensively by IR researchers[36]. We consider the TV scheme as the baseline scheme and set the significance level to 0.05, which is the default value. During this test, we ran 25 queries for the

two schemes. At each scheme, we compute 10 average precision (AV) values (one value for each number of misspelled keywords). The two schemes are compared in two flavors: one-tailed test (we got 0.0.039) and two-tailed test (we got 0.000014). Notice that the results of both tests are lower than the significance level, so we have an actual improvement in the performance of our proposed scheme.

## 9. Conclusion

In this paper, we address the challenging scenario of matching documents from two distinct parties without violating their privacy. Our proposed schemes utilize LSH hashing functions to construct numerical vectors that provide an efficient solution for handling the misspelling mistakes efficiently. In addition, a secure inner product method is developed to calculate similarity scores. Moreover, our proposed schemes consider the keyword occurrences during results ranking. We proposed two schemes with different security levels. The practical value of our schemes is demonstrated through an extensive experimental evaluation using a real-world collection.

As part of future work, we plan to investigate another keyword transforming method to handle more misspelling mistakes like adding or removing letters from the original keywords. We also try to use semantic document representation to catch the semantic similarity between textual documents. Another direction for future work would be to shift our work into the malicious security model, where the involved parties are no longer assumed to follow the steps specified by the protocol. Specifically, in this case, one party would check the computations of the other party.

## References

- [1] Marjai P, Lehotay-Kéry P, Kiss A. Document similarity for error prediction. *Journal of Information and Telecommunication* 2021; 5 (9):1-4. doi: 10.1080/24751839.2021.1893496
- [2] Fröbe M, Bevendorff J, Gienapp L, Völske M, Stein B et al. CopyCat: Near-Duplicates within and between the ClueWeb and the Common Crawl. In: *Proceedings of the 44th International ACM Conference on Research and Development in Information Retrieval (SIGIR 2021)*; New York, NY, USA; 2021. pp. 2398–2404.
- [3] Ma R, Li Y, Li C, Wan F, Hu H et al. Secure multiparty computation for privacy-preserving drug discovery. *Bioinformatics* 2020; 36 (9):2872-2880. doi: 10.1093/bioinformatics/btaa038
- [4] Abid A, Ali W, Farooq MS, Farooq U, Khan NS et al. Semi-automatic classification and duplicate detection from human loss news corpus. *IEEE Access* 2020; 8: 97737-97747. doi: 10.1109/ACCESS.2020.2995789
- [5] Jiang W, Murugesan M, Clifton C, Si L. Similar document detection with limited information disclosure. In: *IEEE 24th International Conference on Data Engineering*; Cancun, Mexico; 2008. pp. 735-743.
- [6] Murugesan M, Jiang W, Clifton C, Si L, Vaidya J. Efficient privacy-preserving similar document detection. *The International Journal on Very Large Databases* 2010; 19 (4):457-475. doi: 10.1007/s00778-009-0175-9
- [7] Jiang W, Samanthula B. N-gram based secure similar document detection. In: *IFIP Annual Conference on Data and Applications Security and Privacy*; Berlin, Heidelberg; 2011. pp. 239-246.
- [8] Buyrukbilen S, Bakiras S. Secure similar document detection with simhash. In: *Workshop on Secure Data Management*; Trento, Italy; 2013. pp. 61-75.
- [9] Yu X, Chen X, Shi J, Shen L, Wang D. Efficient and scalable privacy-preserving similar document detection. In: *IEEE Global Communications Conference*; Singapore; 2017. pp. 1-7.
- [10] Samanthula B, Jiang W. Secure multiset intersection cardinality and its application to jaccard coefficient. *IEEE Transactions on Dependable and Secure Computing* 2016; 13 (5):591-604. doi: 10.1109/TDSC.2015.2415482

- [11] Forman S, Samanthula B. Secure similar document detection: Optimized computation using the jaccard coefficient. In: IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing,(HPSC) and IEEE International Conference on Intelligent Data and Security (IDS); Omaha, NE, USA; 2018. pp. 1-4.
- [12] Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing; Dallas Texas, USA; 1998. pp. 604-613.
- [13] Charikar M. Similarity estimation techniques from rounding algorithms. In: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing; Montreal Quebec, Canada; 2002. pp. 380-388.
- [14] Dong C, Chen L, Wen Z. When private set intersection meets big data: an efficient and scalable protocol. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security; New York, NY, USA; 2013. pp. 789-800.
- [15] Canetti R, Sarkar P, Wang X. Efficient and round-optimal oblivious transfer and commitment with adaptive security. In: International Conference on the Theory and Application of Cryptology and Information Security; Daejeon, South Korea; 2020. pp. 277-308.
- [16] De Cristofaro E, Gasti P, Tsudik G. Fast and private computation of cardinality of set intersection and union. In: International Conference on Cryptology and Network Security; Berlin, Heidelberg; 2012. pp. 218-231.
- [17] Bloom B. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 1970; 13 (7): 422-426. doi: 10.1145/362686.362692
- [18] Datar M, Immorlica N, Indyk P, Mirrokni V. Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the twentieth annual symposium on Computational geometry; New York, NY, USA; 2004. pp. 253-262.
- [19] Gentry C. Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing; New York, NY, USA; 2009. pp. 169-178.
- [20] Chillotti I, Gama N, Georgieva M, Izabachene M. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: International conference on the theory and application of cryptology and information security; Berlin, Heidelberg; 2016. pp. 3-33.
- [21] Schoppmann P, Vogelsang L, Gascón A, Balle B. Secure and Scalable Document Similarity on Distributed Databases: Differential Privacy to the Rescue. *Proceedings on Privacy Enhancing Technologies* 2020; 2020 (2):209-229. doi: 10.2478/popets-2020-0024
- [22] Brakerski Z, Gentry C, Vaikuntanathan V. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory* 2014; 6 (3):1-36. doi: 10.1145/2090236.2090262
- [23] Martins P, Sousa L, Mariano A. A survey on fully homomorphic encryption: An engineering perspective. *ACM Computing Surveys* 2017; 50 (6):1-33. doi: 10.1145/3124441
- [24] Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques; Berlin, Heidelberg; 1999. pp. 223-238.
- [25] Yao A.C.C. How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science; Toronto, ON, Canada; 1986. pp. 162-167.
- [26] Goldreich O, Micali S, Wigderson A. How to play any mental game, or a completeness theorem for protocols with honest majority. In: 19th Annual ACM Symposium on Theory of Computing; New York, NA, USA; 1987. pp. 218-229.
- [27] Aly A, Orsini E, Rotaru D, Smart NP, Wood T. Zaphod: Efficiently combining LSSS and garbled circuits in SCALE. In: Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography; London, UK; 2019. pp. 33-44.
- [28] Fan X, Ganesh C, Kolesnikov V. Hashing garbled circuits for free. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques; Paris, France; 2017. pp. 456-485.

- [29] Li P, Li T, Yao Z, Tang C, Li J. Privacy-preserving outsourcing of image feature extraction in cloud computing. *Soft Computing* 2017; 21 (15):4349-4359. doi: 10.1007/s00500-016-2066-5
- [30] Veugen T. Improving the DGK comparison protocol. In: 2012 IEEE International Workshop on Information Forensics and Security (WIFS); Costa Adeje, Spain; 2012. pp. 49-54.
- [31] Damgård I, Geisler M, Krøigaard M. Efficient and secure comparison for on-line auctions. In: Australasian conference on information security and privacy; Berlin, Heidelberg; 2007. pp. 416-430.
- [32] Manning C, Raghavan P, Schütze H. *Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press, 2008.
- [33] Bost R, Popa R, Tu S, Goldwasser S. Machine learning classification over encrypted data. In: The Network and Distributed System Security Symposium; San Diego, CA, USA; 2015. pp.1-14.
- [34] Lindell Y, Pinkas B. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality* 2009; 1 (1):59–98. doi: 10.29012/jpc.v1i1.566
- [35] Porter M. An algorithm for suffix stripping. *Program: electronic library and information systems* 2006; 40 (3):211-218. doi: 10.1108/00330330610681286
- [36] Urbano Merino J, De Lima HA, Hanjalic A. Statistical Significance Testing in Information Retrieval: An Empirical Analysis of Type I, Type II and Type III Errors. In: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19); Paris, France; 2019. pp. 505–514.