Secure Similar Document Detection with Simhash

Sahin Buyrukbilen[1] and Spiridon Bakiras[2]

[1] The Graduate Center
City University of New York
sbuyrukbilen@gc.cuny.edu
[2] John Jay College
City University of New York
sbakiras@jjay.cuny.edu

Abstract. Similar document detection is a well-studied problem with important application domains, such as plagiarism detection, document archiving, and patent/copyright protection. Recently, the research focus has shifted towards the privacy-preserving version of the problem, in which two parties want to identify similar documents within their respective datasets. These methods apply to scenarios such as patent protection or intelligence collaboration, where the contents of the documents at both parties should be kept secret. Nevertheless, existing protocols on secure similar document detection suffer from high computational and/or communication costs, which renders them impractical for large datasets. In this work, we introduce a solution based on *simhash* document fingerprints, which essentially reduce the problem to a secure XOR computation between two bit vectors. Our experimental results demonstrate that the proposed method improves the computational and communication costs by at least one order of magnitude compared to the current state-of-the-art protocol. Moreover, it achieves a high level of precision and recall.

1 Introduction

Similar document detection is an important problem in computing, and has attracted a lot of research interest since its introduction by Manber [10]. Specifically, with digital data production growing exponentially, efficient file system management has become crucial. Detecting similar files facilitates better indexing, and provides efficient access to the file system. Furthermore, it protects against security breaches by identifying file versions that are changed by a virus or a hacker. Similarly, web search engines periodically crawl the entire web to collect individual pages for indexing [11]. When a web page is already present in the index, its newer version may differ only in terms of a dynamic advertisement or a visitor counter and may, thus, be ignored. Therefore, detecting similar pages is of paramount importance for designing efficient web crawlers. Finally, plagiarism detection and copyright protection are two other major applications that are built upon similar document detection.

While plaintext similar document detection is extremely important, it is not sufficient for secure and private operations over sensitive data. In many cases, owners of sensitive data may be forced to share their datasets with the government or other entities, in order to comply with existing regulations. For example, health care companies may be asked to provide data to monitor certain diseases reported in their databases. This may be accomplished by identifying similar attribute patterns in patient diagnosis information from different entities. Obviously, such pattern searches cannot be performed without secure protocols, since they may lead to severe privacy violations for the individuals included in the various databases.

Data sharing for intelligence operations also involves risks when disclosing classified information to other parties. A person of interest may have records at several intelligence agencies under different names with similar attributes. To identify similar records, the participating agencies may only wish to disclose the existence of records akin to the query. Detecting violations of the academic double submission policy is another problem with similar restrictions. For example, a conference's organization committee may want to know whether the articles submitted to their conference are concurrently submitted to other publication venues. Since research articles are considered confidential until published, their contents cannot be revealed unless a similar article is found in another venue.

Secure similar document detection (SSDD) leverages secure two-party computation protocols, in order to solve the above problems that arise due to the distributed ownership of the data. In particular, SSDD involves two parties, each holding their own private dataset. Neither party wants to share their data in plaintext format, but they both agree to identify any similar documents within their respective databases. The objective is to compute the similarity scores between every pair of documents without revealing any additional information about their contents. In existing work, document similarity is computed with either the inner product of public key encrypted vectors [7, 12, 8] or with secure set intersection cardinality methods based on N-grams [1]. However, the computational cost of inner product based similarity is very high, due to numerous public key operations. On the other hand, N-gram based methods are more computationally efficient, but they incur a high communication cost as the number of documents increases.

In this study, we present a novel method based on *simhash* document fingerprints[3]. Simhash is essentially a dimensionality reduction technique that encodes all the document terms and their frequencies into a fixed-size bit vector (typically 64 bits). Unlike classical hashing algorithms that produce uniformly random digests, the simhash digests of two similar documents will only differ in a few bit positions [6]. This enables us to (i) evaluate the similarity over a fairly small data structure rather than large vectors, and (ii) reduce the similarity calculation to a secure XOR computation between two bit vectors. To further improve the privacy preserving properties of our approach, we modify the basic method to hide the similarity scores of the compared documents. In particular, the enhanced

[3] We follow the simhash definition of Charikar [2].

version of our protocol returns all the document pairs whose similarity is above a user-defined threshold, while maintaining the exact scores secret. This is the *first* protocol in the literature that provides this functionality. Our experimental results demonstrate that the proposed methods improve the computational and communication costs by at least one order of magnitude compared to the current state-of-the-art protocol. Moreover, they achieve a high level of precision and recall.

The remainder of the paper is organized as follows. Section 2 describes the various primitives utilized in our methods and summarizes previous work on secure similar document detection. Section 3 presents the formal definition of the SSDD query and describes the underlying threat model and security. Section 4 introduces the details of our basic protocol and Section 5 presents the enhanced version that hides the exact similarity scores. Section 6 illustrates the results of our experiments and Section 7 concludes our work.

2 Background

Section 2.1 introduces the cryptographic primitives utilized in our methods and Section 2.2 describes the simhash algorithm. Section 2.3 surveys the related work on secure similar document detection.

2.1 Cryptographic primitives

Homomorphic encryption. Homomorphism in encryption allows one to evaluate arithmetic operations, such as multiplication and addition, over plaintext values by manipulating their corresponding ciphertexts. Most public key encryption schemes in the literature are *partially* homomorphic, i.e., they allow the evaluation of only one type of operation (either addition or multiplication).

In our work, we utilize ElGamal's *additively* homomorphic encryption scheme [5, 3]. The scheme incorporates key generation, encryption, and decryption algorithms, as shown in Figure 1. The homomorphic properties of this cryptosystem are as follows (where $E(\cdot)$ denotes encryption):

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

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

$$E(m_1 k) = E(m_1)^k$$

Note that, ElGamal's scheme is *semantically* secure, i.e., it is infeasible to derive any information about a plaintext, given its ciphertext and the public key that was used to encrypt it. Its security is based on the decisional Diffie-Hellman assumption. Also note that the decryption process involves a discrete logarithm computation. If the encrypted values are not too large (which is the case in our protocol) it is possible to precompute all possible results and use them as a lookup table to speed up the decryption process.

ElGamal cryptosystem

Key generation

1. Instantiate a cyclic group G of prime order p, with generator g (G, g, and p are public knowledge)
2. Choose a *private* key x, uniformly at random from $\mathbb{Z}_p^*$
3. Publish the *public* key $h = g^x$

Encryption

1. Let m be the private message
2. Choose r, uniformly at random from $\mathbb{Z}_p^*$
3. Compute ciphertext $(c_1, c_2) = (g^r, h^{r+m})$

Decryption

1. Compute $h^m = c_2 \cdot (c_1^x)^{-1}$
2. Solve the discrete logarithm to retrieve m

Fig. 1. The ElGamal cryptosystem

Secure two-party computation. A secure two-party computation protocol [9] allows two parties, Alice and Bob, to jointly compute a function based on their inputs, while maintaining their inputs secret (i.e., they only learn the function output). Yao's *garbled circuit* technique [14] is a generic two-party computation protocol that can evaluate securely any function, given its Boolean circuit representation. Nevertheless, Yao's technique is efficient only for relatively simple functions, i.e., when the number of input wires and logic gates is small. In particular, every input wire (for one of the parties) necessitates the execution of an Oblivious Transfer (OT) [13] protocol that is computationally expensive, while the total number of gates affects the overall communication and circuit construction/evaluation costs.

Besides Yao's generic protocol, researchers have also devised application dependent protocols that typically leverage the properties of additively homomorphic encryption. As an example, consider the *secure inner product* computation that is used extensively in previous work [12, 8]. For simplicity, assume that Alice holds vector $\langle a_1, a_2 \rangle$ and Bob holds vector $\langle b_1, b_2 \rangle$. The objective is for Alice to securely compute $S = a_1 b_1 + a_2 b_2$. Initially, Alice encrypts her input with her public key and sends $E(a_1), E(a_2)$ to Bob. Next, Bob utilizes the properties of homomorphic encryption to produce $E(S) = E(a_1)^{b_1} E(a_2)^{b_2}$. Finally, Alice decrypts the result and learns the value of S.

2.2 Simhash

Simhash maps a high dimensional feature vector into a fixed-size bit string [2]. However, unlike classical hashing algorithms, simhash produces fingerprints that have a large number of matching bits when the underlying documents are similar. Computing the simhash fingerprint from a text document is a fairly simple

process. First, one has to extract all the document terms along with their weights (e.g., how many times they appear in the document). Then, a vector of l counters $\langle c_0, c_1, \ldots, c_{l-1} \rangle$ is initialized, where l is the size of the simhash fingerprint (e.g., 64 bits). Each of the document's terms is then hashed with a standard hashing algorithm, such as SHA-1. If the bit at position i ($i \in \{0, 1, \ldots, l-1\}$) in the resulting SHA-1 digest is 0, c_i is decremented by the weight of that term; otherwise, c_i is incremented by the same weight. When all document terms are processed, the simhash fingerprint is constructed as follows: for all $i \in \{0, 1, \ldots, l-1\}$, if $c_i > 0$, set the corresponding bit to 1; otherwise, set the bit to 0.

2.3 Related Work

The problem of secure similar document detection was first introduced by Jiang et al. [7]. In their approach, Alice and Bob first run a secure protocol to identify the common terms that appear in both datasets (dictionary). Then, similarity is computed with the *cosine* of the angle between two document term vectors. The cosine computation requires a secure inner product protocol, identical to the one described in Section 2.1. Specifically, for Alice to compare a single document against Bob's database, she first uses her public key to encrypt the weights of every term in the dictionary (if the term does not exist in Alice's document, its weight is 0). After Bob receives the encrypted vector, he uses his plaintext term vectors to blindly compute the encryptions of the inner products for all documents. Finally, Alice decrypts the results and computes the similarity between her document and each document in Bob's database. This protocol is computationally expensive, because of numerous public key operations at both parties. Furthermore, its performance degrades as the size of the dictionary space increases. For example, the similarity search between two document sets, each containing 500 documents, takes about a week to complete [7].

The authors of [7] extend their work in [12] with two optimizations. First, to reduce the number of modular multiplications at Bob, they ignore every ciphertext in Alice's vector where the corresponding plaintext value at Bob is zero. Second, to reduce the number of document comparisons, each party applies (in a pre-processing step) a k-means clustering algorithm on their documents. The idea is to initially compare only the cluster representatives and measure their similarity. If that similarity value is above a certain threshold, then the documents in both clusters are compared in a pairwise manner. Nevertheless, the drawback of clustering is that it is sensitive to the value of k. If k does not accurately reflect the underlying document similarities, it may result in a significant loss in query precision and recall.

Jiang and Samanthula [8] propose the use of N-grams in their SSDD protocol. An N-gram representation of a document consists of all the document's substrings of size N (after removing all punctuation marks and whitespaces). In general, N-grams are considered a better document representation method than term vectors, because they are language independent, more sensitive to local similarity, simple, and less sensitive to document modifications [8]. Specifically, Jiang and Samanthula utilize 3-gram sets and define the similarity between two

documents as the Jaccard index of their 3-gram sets. Prior to protocol execution, both parties create the 3-gram sets of their documents and Bob discloses his *global* 3-gram set to Alice. To compare a pair of documents, Alice and Bob create the binary vectors of the corresponding 3-gram sets with respect to Bob's global 3-gram set (let A be Alice's vector and B be Bob's vector). Next, the two parties invoke a secure two-party computation protocol to compute $|A \cap B|$ in an additively split form. Finally, they run a secure division protocol to compute the Jaccard index $J = \frac{|A \cap B|}{|A \cup B|}$. Unfortunately, the above protocol is not secure [1], because Bob has to reveal his global 3-gram set to Alice. By utilizing this information, Alice can easily check whether a word appears in Bob's global collection, which is an obvious security breach.

Blundo et al. [1] introduce EsPRESSo, a protocol for privacy-preserving evaluation of sample set similarity. It is based on the private set intersection cardinality (PSI-CA) protocol of De Cristafaro et al. [4]. The authors show that one possible application of EsPRESSo is similar document detection and propose a solution based on 3-grams. To compare two documents, Alice and Bob first create the 3-gram sets of their respective documents. Next, Alice hashes her 3-grams and raises the resulting digests to a random number R_a (let's call this set A). She then sends A to Bob who, in turn, raises these values to his random number R_b and randomly permutes the set. He also hashes his 3-gram set members and raises the hash values to R_b (let's call this set B). Bob then sends both sets back to Alice. Alice removes R_a from A and computes the cardinality of the intersection between A and B ($|A \cap B|$). From this value, she computes the Jaccard index as $J = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$.

The limitation of the basic EsPRESSo protocol is that its performance depends on the total number of 3-grams that appear in the compared documents. To this end, the authors of [1] introduce an optimization based on the MinHash technique. In particular, instead of incorporating every available 3-gram in the corresponding 3-gram sets (A and B), Alice and Bob agree on k distinct hash functions $(H_1, H_2, \ldots, H_k)$ to produce sets of size k, independent of the total number of 3-grams. Specifically, for $i \in \{1, 2, \ldots, k\}$, each party hashes all their 3-grams with the H_i hash function and select the digest with the minimum value as a representative in their respective set. Once sets A and B are constructed, the EsPRESSo protocol is invoked to compute the Jaccard index between the two documents. The MinHash approximation reduces considerably the computational and communication costs and is currently the state-of-the-art protocol in secure similar document detection.

3 Problem Definition

Bob (the server) holds a collection of N documents $\mathbb{D} = \{D_1, D_2, \ldots, D_N\}$. Each document $D_i \in \mathbb{D}$ is represented as a set of pairs $\langle w_i, f_i \rangle$, where w_i is a term appearing in the document and f_i is its frequency (i.e., the number of times it appears in the document). Alice (the client) holds a single document q that is represented in a similar fashion. Alice wants to know which documents in Bob's

collection $\mathbb{D}$ are similar to q. Note that, if Alice herself holds a collection of M documents, the query is simply evaluated M distinct times.

In this work we propose two protocols with different privacy guarantees. The security of the basic protocol (Simhash, Section 4) is identical to the security provided by all existing SSDD protocols:

- For all $i \in \{1, 2, \ldots, N\}$, Alice learns the similarity score between q and D_i.
- Bob learns nothing.

On the other hand, the enhanced version of our protocol (Simhash*, Section 5) provides some additional security to the server (Bob):

- For all $i \in \{1, 2, \ldots, N\}$, Alice learns whether D_i's similarity score is above a user-defined threshold t (boolean value). The exact score remains secret.
- Bob learns nothing.

We assume that both parties could behave in an adversarial manner. Their goal is to derive any additional information other than the existence of similar documents and their similarity scores. For example, they could be interested in the contents of the other party's documents, statistical information about the terms in the other party's document collection, etc. Finally, we assume that both parties run in polynomial time and are "semi-honest," i.e., they will follow the protocol correctly, but will try to gain any advantage by analyzing the information exchanged during the protocol execution.

4 Basic protocol

In this section we introduce our basic protocol that reveals the exact similarity score for each one of Bob's documents to Alice. Section 4.1 presents the protocol and Section 4.2 outlines its security proof.

4.1 The Simhash protocol

Prior to protocol execution, each party runs a preprocessing step to generate the simhash fingerprints of their documents. The preprocessing includes lower case conversion, stop word removal, and stemming. In the end, each document is reduced to a set of terms and their corresponding frequencies. The simhash fingerprints are then created according to the algorithm described in Section 2.2. In what follows, we use a to denote Alice's simhash (from document q) and b_i ($i \in \{1, 2, \ldots, N\}$) to denote the simhash of document D_i in Bob's database. Recall that all fingerprints are binary vectors of size $l = 64$ bits.

Similarity based on simhash fingerprints is defined as the number of *non-matching* bits between the two bit vectors. In other words, a similarity score of 0 indicates two possibly identical documents, while larger scores characterize less similar documents. Consequently, it suffices to securely compute (i) the bitwise XOR of the two vectors and (ii) the summation of all bits in the resulting XOR

Simhash

Input: Alice has a simhash fingerprint a
Bob has N simhash fingerprints $\{b_1, b_2, \ldots, b_N\}$
Output: Alice gets N similarity scores $\{\sigma_1, \sigma_2, \ldots, \sigma_N\}$

Alice
1: Alice sends to Bob $E(a[0]), E(a[1]), \ldots, E(a[l-1])$;

Bob
2: **for** $(i = 1; i \leq N; i++)$ **do**
3: Set $\sigma_i \leftarrow 0$ and compute $E(\sigma_i)$;
4: **for** $(j = 0; j < l; j++)$ **do**
5: **if** $(b_i[j] == 0)$ **then**
6: $E(\sigma_i) \leftarrow E(\sigma_i)E(a[j])$;
7: **else**
8: $E(\sigma_i) \leftarrow E(\sigma_i)E(1)E(a[j])^{-1}$;
9: **end if**
10: **end for**
11: **end for**
12: Bob sends to Alice $E(\sigma_1), E(\sigma_2), \ldots, E(\sigma_N)$;

Alice
13: Alice decrypts all ciphertexts and retrieves $\sigma_1, \sigma_2, \ldots, \sigma_N$;

Fig. 2. The Simhash protocol

vector. Figure 2 shows the detailed protocol, where $E(\cdot)$ denotes encryption with Alice's ElGamal public key (which is known to Bob).

First (line 1), Alice encrypts every bit of her fingerprint a and sends l ciphertexts to Bob. Bob cannot decrypt these ciphertexts but is still able to blindly perform the required XOR and addition operations. In particular, for every document D_i in his database, Bob initializes the encryption of the similarity score to $E(\sigma_i) = E(0)$ (line 3). Next, he iterates over the l bits of the corresponding fingerprint b_i. If the bit at a certain position j is 0, then the result of the XOR operation is equal to $a[j]$ and Bob simply adds the value to the encrypted score (line 6). Otherwise, the result of the XOR operation is $(1 - a[j])$ which is also added to $E(\sigma_i)$ in a similar fashion (line 8). After all documents are processed, Bob sends the encrypted results to Alice (line 12). Finally, Alice uses her private key to decrypt the scores and identify the most similar documents to q (line 13).

4.2 Security

In this section we prove the security of the Simhash protocol for semi-honest adversaries, following the simulation paradigm [9]. In particular, we will show that, for each party, we can simulate the distribution of the messages that the party receives, given only the party's input and output in this protocol. This

is a sufficient requirement for security because, if we can simulate each party's view from only their respective input and output, then the messages themselves cannot reveal any additional information.

Alice's input consists of a bit vector a and her output is $\{\sigma_1, \sigma_2, \ldots, \sigma_N\}$. The only messages that Alice receives from Bob are the encryptions of the N similarity scores. The simulator knows Alice's public key and it also knows her output. Therefore, it can simply generate the encryptions of the corresponding scores.

In Bob's case, the input is N bit vectors and there is no output. In the beginning of the protocol, Bob receives l encryptions from Alice. Here, the simulator can simply generate l encryptions of zero. Given the assumption that the underlying encryption scheme is semantically secure, Bob cannot distinguish these ciphertexts from the ones that are produced by Alice's real input.

5 Enhanced protocol

The basic Simhash protocol has the same security definition as all existing SSDD protocols in the literature. That is, Alice learns the similarity score for every document D_i in Bob's database. Nevertheless, making all this information available to Alice may allow her to construct some "malicious" queries that reveal whether a certain term (or 3-gram) exists in Bob's database. Consider the EsPRESSo protocol as an example. Alice's query may consist of a number of fake 3-grams (i.e., 3-grams that could not appear in Bob's documents) plus a real one that Alice wants to test against Bob's database. After completing the protocol execution, Alice can infer that the 3-gram is present in Bob's database if at least one of the similarity scores is non-zero. This attack is not as trivial to perform with the simhash or MinHash techniques, but it is still possible for sophisticated adversaries to devise similar attacks.

To this end, in this section, we introduce Simhash*, an enhanced version of the basic Simhash protocol that maintains the similarity scores secret. This is the first SSDD protocol in the literature that provides this functionality. In particular, Alice and Bob agree on a similarity threshold t and the protocol returns, for each document D_i, a boolean value θ_i that indicates whether $\sigma_i \leq t$. The detailed protocol is shown in Figure 3.

The first steps of the protocol (lines 1–10) are identical to Simhash, i.e., Bob blindly computes the encryptions of all N similarity scores. However, instead of sending these ciphertexts to Alice, Bob computes, for each $D_i \in \mathbb{D}$, the encryptions of $r_j \cdot (\sigma_i - j)$ where $j \in \{0, 1, \ldots, t\}$ (lines 12–13). Specifically, r_j is a uniformly random value that masks the actual similarity score (σ_i) when it is not equal to j. On the other hand, if σ_i is equal to j, then the computed value is an encryption of 0. Next, Bob uses a random permutation π_i for each set of $(t+1)$ ciphertexts corresponding to document D_i, and eventually sends a total of $(t+1) \cdot N$ ciphertexts back to Alice (line 16). The different permutations are required in order to prevent Alice from inferring the value of j (i.e., similarity score) that produces the encryption of 0. Finally, Alice concludes that document

Simhash*

Input: Alice has a simhash fingerprint a
Bob has N simhash fingerprints $\{b_1, b_2, \ldots, b_N\}$
Output: Alice gets N binary values $\{\theta_1, \theta_2, \ldots, \theta_N\}$

Alice
1: Alice sends to Bob $E(a[0]), E(a[1]), \ldots, E(a[l-1])$;

Bob
2: **for** $(i = 1; i \leq N; i++)$ **do**
3: Set $\sigma_i \leftarrow 0$ and compute $E(\sigma_i)$;
4: **for** $(j = 0; j < l; j++)$ **do**
5: **if** $(b_i[j] == 0)$ **then**
6: $E(\sigma_i) \leftarrow E(\sigma_i)E(a[j])$;
7: **else**
8: $E(\sigma_i) \leftarrow E(\sigma_i)E(1)E(a[j])^{-1}$;
9: **end if**
10: **end for**
11: **for** $(j = 0; j \leq t; j++)$ **do**
12: Choose r_j, uniformly at random from $\mathbb{Z}_p^*$;
13: $E(x_{ij}) \leftarrow [E(\sigma_i)E(j)^{-1}]^{r_j}$;
14: **end for**
15: **end for**
16: Bob sends to Alice $\{E(x_{ij})\}, \forall i \in \{1, 2, \ldots, N\}, j \in \pi_i(\{0, 1, \ldots, t\})$;

Alice
17: Alice decrypts all ciphertexts and retrieves $\{x_{ij}\}$;
18: **for** $(i = 1; i \leq N; i++)$ **do**
19: **for** $(j = 0; j \leq t; j++)$ **do**
20: **if** $(x_{ij} == 0)$ **then**
21: **break**;
22: **end if**
23: **end for**
24: **if** $(j > t)$ **then**
25: $\theta_i \leftarrow$ **false**;
26: **else**
27: $\theta_i \leftarrow$ **true**;
28: **end if**
29: **end for**

Fig. 3. The Simhash* protocol

D_i's similarity score is within the predetermined threshold t, if and only if one of the $(t+1)$ ciphertexts corresponding to D_i decrypts to 0 (lines 19–28).

The security proof of the Simhash* protocol is trivial and follows the proof outlined in Section 4.2. In particular, only Alice's case is different, since (i) her output is N boolean values $\{\theta_1, \theta_2, \ldots, \theta_N\}$ and (ii) she receives $(t+1) \cdot N$

ciphertexts from Bob. Nevertheless, the simulator knows Alice's output and also knows how the protocol operates. Therefore, for all documents D_i where θ_i is *true*, the simulator generates t random encryptions plus one encryption of 0. On the other hand, for documents where θ_i is *false*, the simulator generates $(t+1)$ random encryptions.

6 Experimental evaluation

In this section we experimentally compare the performance of our methods against existing SSDD protocols. Section 6.1 describes the experimental setup and Section 6.2 illustrates the results of our experiments.

6.1 Setup

We compare our protocols against the work of Murugesan et al. [12] that utilizes cosine similarity (labeled as "Cosine" in our results), and EsPRESSo (both the basic protocol and the MinHash optimization) [1] that is based on 3-grams. We implemented all protocols in C++ and leveraged the GMP[4] library for handling large numbers. To ensure a fair comparison, we set the bit length of p (the order of the cyclic group G[5] in Figure 1) to 160 bits, and the bit length of the RSA modulus in Paillier's cryptosystem to 1024 bits. This results in similar security levels for the underlying cryptographic protocols. We ran both the client and the server applications on a 2.4 GHz Intel Core i5 CPU. The performance metrics that we tested include the CPU time, the communication cost, and the precision/recall of the document retrieval process.

The document corpus is a collection of Wikipedia[6] articles. In particular, we selected 103 main articles from diverse topics and, for each article, we also selected a number (around 10) of its previous versions from the history pages of this topic. The total number of documents in the corpus is 1152. For Simhash and Cosine, we applied lower case conversion, stop word removal, and stemming, in order to derive the document terms along with their frequencies. For the EsPRESSo protocols, we extracted the 3-grams as explained in [1]. The total number of terms in the documents is 152,571 and the total number of 3-grams is 10,392.

6.2 Results

In the first set of experiments we investigate the document retrieval performance of the various protocols. The objective is to compare the underlying document representation methods: term vectors, simhash, and 3-grams. The experiments were performed as follows. We run 103 queries, where the query documents were

[4] `http://gmplib.org`
[5] Note that the EsPRESSo protocols are also implemented on top of group G.
[6] `http://en.wikipedia.org`

selected to be the most recent versions of the 103 unique articles. Using different threshold values, we observed the precision and soundness of the retrieved documents (recall that we know in advance the "correct" results, since the different versions of each article are very similar to each other). For our methods we used the threshold values $\{2, 3, 4, 5, 6\}$, while for the rest of the protocols we used the values $\{0.6, 0.7, 0.8, 0.9, 0.99\}$. Observe that, for Simhash, larger threshold values imply less similar documents, whereas for the other methods the opposite is true.

We used the *precision* and *recall* as the performance metrics for the document retrieval process. Precision is defined as:

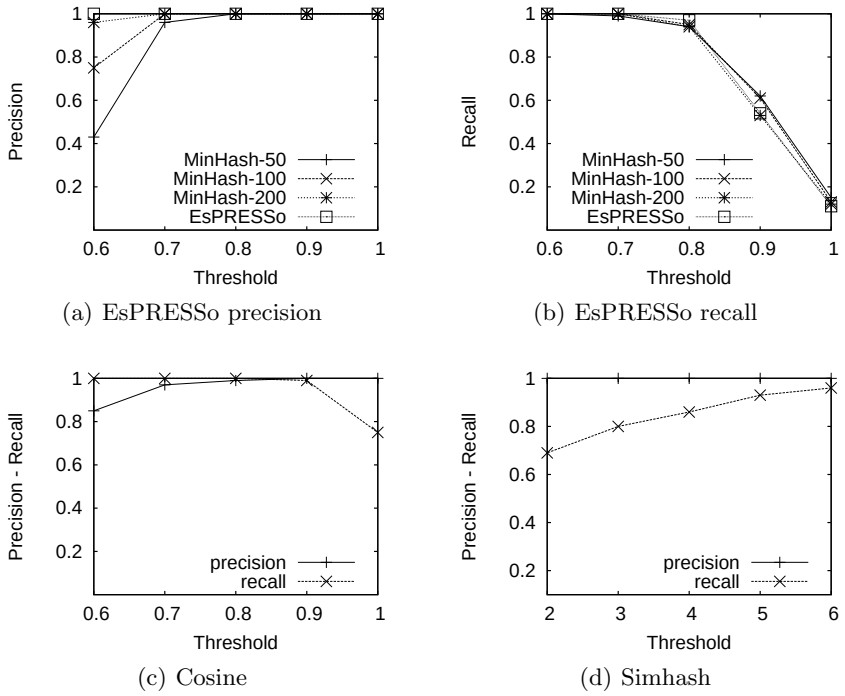
$$precision = \frac{|R \cap V|}{|R|}$$

where R is the set of retrieved documents and V is the total number of documents that satisfy the query. In other words, precision is equal to the fraction of retrieved documents that belong to the result set. Recall, on the other hand, indicates the fraction of the result set that is retrieved by the query and is computed as:

$$recall = \frac{|R \cap V|}{|V|}$$

Figures 4(a) and 4(b) show the precision and recall curves for the various EsPRESSo protocols. As expected, the basic protocol has the best overall performance and maintains a precision of 1.0 for all threshold values. The MinHash approximations sacrifice some precision for better running times, but they all perform very well for threshold values larger than 0.7. In terms of recall, all EsPRESSo variants are very sensitive to the underlying threshold value, experiencing a large drop when the threshold is larger than 0.8. The Cosine method has a very stable performance, as shown in Figure 4(c). In particular, both the precision and recall values remain over 0.75 under all settings. Finally, Simhash exhibits excellent query precision for all threshold values (Figure 4(d)). Furthermore, the query recall raises steadily with increasing threshold values and, when the threshold is 6, Simhash retrieves over 96% of the relevant documents.

In the next experiment we measure the computational cost of the various methods. We select MinHash-50 (i.e., MinHash with $k = 50$ hash functions) to represent the EsPRESSo family of protocols, since it has the best performance in terms of CPU time. The experiments were performed as follows. We run the cryptographic protocols for the 103 unique queries and measured the total CPU time, excluding the initial query encryption time (which is performed only once, independent of the database size N). From this value we determined the average time needed to compare a pair of documents. Using this measurement, Figure 5 depicts the CPU time required to compare one document against a database of size N, where $N \in \{100, 500, 1000, 3000, 5000\}$ (the curves also include the query encryption time). Simhash is by far the best protocol among all competitors and it is one order of magnitude faster than MinHash-50. Cosine incurs a very high computational cost, mainly due to the query encryption step that involves tens



(a) EsPRESSo precision (b) EsPRESSo recall

(c) Cosine (d) Simhash

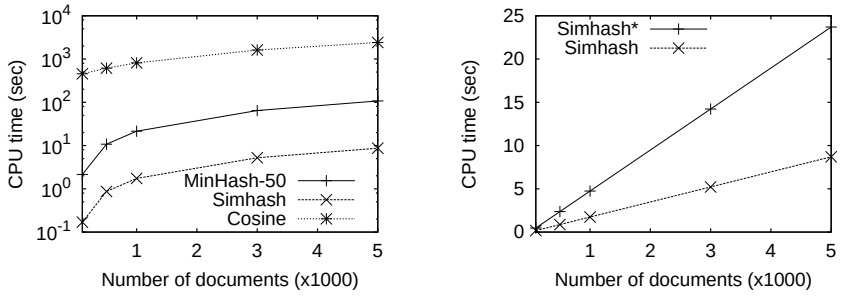
Fig. 4. Precision and recall

of thousands of public key encryptions. MinHash-50 is significantly slower than Simhash, because it involves numerous (expensive) modular exponentiations for every document in the server's database.

Figure 5(b) shows the CPU overhead of the Simhash* protocol, where the similarity threshold is set to $t = 6$. The additional cost is due to the $(t + 1)$ modular exponentiations that are required to hide a document's similarity score. However, Simhash* is considerably faster than MinHash-50, incurring 23.7 sec of CPU time to compare 5000 documents, as opposed to 107.5 sec for MinHash-50.

Figure 6(a) illustrates the communication cost for Simhash, MinHash-50, and Cosine. Clearly, Simhash outperforms significantly both competitor methods, incurring a communication cost that is at least 18 times smaller under all settings. For example, to compare one document against a database of size $N = 5000$, requires 1.24 MB of data communication for Simhash, 35.29 MB for MinHash-50, and 38.47 MB for Cosine. The drawback of MinHash-50 is that it has to send 50 ciphertexts plus 50 SHA-1 hashes for every document in the database. On the other hand, the overhead for Cosine lies exclusively on the transmission of the encrypted term vector, which is why it seems to remain unaffected by the database size N.

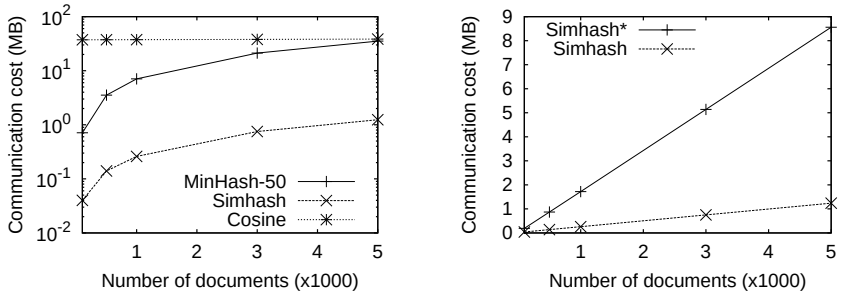
Finally, Figure 6(b) shows the communication overhead for the Simhash* protocol. In this experiment, the threshold t is set to 6, which necessitates the



(a) Simhash, MinHash, Cosine (b) Simhash, Simhash*

Fig. 5. CPU time

transmission of 7 ciphertexts for every document in the database. As a result, the communication cost of Simhash* is around 7 times larger than the cost of the basic Simhash protocol. Nevertheless, it is still significantly lower than the competitor methods, requiring just 8.56 MB of data for $N = 5000$ documents.



(a) Simhash, MinHash, Cosine (b) Simhash, Simhash*

Fig. 6. Communication cost

7 Conclusions

Secure similar document detection (SSDD) is a new and important research area with numerous application domains, such as patent protection, intelligence collaboration, etc. In these scenarios, two parties want to identify similar documents within their databases, while maintaining their contents secret. Nevertheless, existing SSDD protocols are very expensive in terms of both computational and communication cost, which limits their scalability with respect to the number of documents. To this end, we introduce a novel solution based on simhash document fingerprints that is both simple and robust. In addition, we

propose an enhanced version of our protocol that, unlike existing work, hides the similarity scores of the compared documents from the client. Through rigorous experimentation, we show that our methods improve the computational and communication costs by at least one order of magnitude compared to the current state-of-the-art protocol. Furthermore, they perform very well in terms of query precision and recall.

Acknowledgments

This research has been funded by the NSF CAREER Award IIS-0845262.

References

1. Blundo, C., Cristofaro, E.D., Gasti, P.: Espresso: Efficient privacy-preserving evaluation of sample set similarity. In: DPM/SETOP. pp. 89–103 (2012)
2. Charikar, M.: Similarity estimation techniques from rounding algorithms. In: STOC. pp. 380–388 (2002)
3. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. European Transactions on Telecommunications 8(5), 481–490 (1997)
4. Cristofaro, E.D., Gasti, P., Tsudik, G.: Fast and private computation of set intersection cardinality. IACR Cryptology ePrint Archive 2011, 141 (2011)
5. ElGamal, T.: A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
6. Huang, L., Wang, L., Li, X.: Achieving both high precision and high recall in near-duplicate detection. In: CIKM. pp. 63–72 (2008)
7. Jiang, W., Murugesan, M., Clifton, C., Si, L.: Similar document detection with limited information disclosure. In: ICDE. pp. 735–743 (2008)
8. Jiang, W., Samanthula, B.K.: N-gram based secure similar document detection. In: DBSec. pp. 239–246 (2011)
9. Lindell, Y., Pinkas, B.: Secure multiparty computation for privacy-preserving data mining. Journal of Privacy and Confidentiality 1(1), 59–98 (2009)
10. Manber, U.: Finding similar files in a large file system. In: USENIX Winter. pp. 1–10 (1994)
11. Manku, G.S., Jain, A., Sarma, A.D.: Detecting near-duplicates for web crawling. In: WWW. pp. 141–150 (2007)
12. Murugesan, M., Jiang, W., Clifton, C., Si, L., Vaidya, J.: Efficient privacy-preserving similar document detection. VLDB J. 19(4), 457–475 (2010)
13. Naor, M., Pinkas, B.: Computationally secure oblivious transfer. Journal of Cryptology 18(1), 1–35 (2005)
14. Yao, A.C.C.: How to generate and exchange secrets. In: FOCS. pp. 162–167 (1986)