Digital Investigation 29 (2019) S113–S123

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2019 USA — Proceedings of the Nineteenth Annual DFRWS USA

FbHash: A New Similarity Hashing Scheme for Digital Forensics

Donghoon Chang [a], Mohona Ghosh [b], Somitra Kumar Sanadhya [c], Monika Singh [a, d, *], Douglas R. White [d]

[a] Indraprastha Institute of Information Technology, Delhi (IIIT-D), Delhi, India
[b] Department of Information Technology at Indira Gandhi Delhi Technical University of Women, Delhi, India
[c] Department of Computer Science and Engineering, IIT Ropar, India
[d] National Institute of Standards and Technology (NIST), USA

ARTICLE INFO

Article history:

Keywords:
Data fingerprinting
Similarity digests
Fuzzy hashing
TF-IDF
Cosine-similarity

ABSTRACT

With the rapid growth of the World Wide Web and Internet of Things, a huge amount of digital data is being produced every day. Digital forensics investigators face an uphill task when they have to manually screen through and examine tons of such data during a crime investigation. To expedite this process, several automated techniques have been proposed and are being used in practice. Among which tools based on *Approximate Matching algorithms* have gained prominence, e.g., `ssdeep`, `sdhash`, `mvHash` etc. These tools produce hash signatures for all the files to be examined, compute a similarity score and then compare it with a known reference set to filter out known good as well as bad files. In this way, exact as well as similar matches can be screened out. However, all of these schemes have been shown to be prone to active adversary attack, whereby an attacker, by making feasible changes in the content of the file, intelligently modifies the final hash signature produced to evade detection. Thus, an alternate hashing scheme is required which can resist this attack. In this work, we propose a new Approximate Matching scheme termed as - `FbHash`. We show that our scheme is secure against active attacks and detects similarity with 98% accuracy. We also provide a detailed comparative analysis with other existing schemes and show that our scheme has a 28% higher accuracy rate than other schemes for uncompressed file format (e.g., text files) and 50% higher accuracy rate for compressed file format (e.g., docx etc.). Our proposed scheme is able to correlate a fragment as small as 1% to the source file with 100% detection rate and able to detect commonality as small as 1% between two documents with appropriate similarity score. Further, our scheme also produces the least false negatives in comparison to other schemes.

1. Introduction

In today's era of ubiquitous computing, an exponential growth in digital data production (stored in computer hard-disks, external hard-drives, pen-drives, mobile storage, flash drives, tablets, smart chips etc.) is being witnessed. Hence, on a crime scene, a digital forensic investigator may be confronted with several terabytes of digital data, which is too enormous to be analyzed manually. For efficient utilization of time and resources, the major requirement of today's forensic investigation process is to have the capability to extract potentially relevant data from all the data collected at a crime scene. This much smaller but most useful data can then be examined manually.

The filtering process used in extracting the data typically uses fast hashing based algorithms. Large files are passed through a hash function to produce a hash output called a digital fingerprint. The fingerprints of the case files are then matched with a known reference dataset, the most popular being the NIST reference data set (NIST, 2008) to extract unknown files. The filtering process can be performed by either *Blacklisting* or by *Whitelisting*. **Blacklisting** is the process of filtering data by matching them with the set of Known-to-be-bad files (as determined by the investigator). The resultant files after this process are the ones which an investigator needs to examine closely. On the other hand **Whitelisting** is the process of filtering by matching the files with a set of already Known-to-be-good files. The files passing this process need not be examined by the investigator.

Traditional (cryptographic) hash function based matching

* Corresponding author. Indraprastha Institute of Information Technology, Delhi (IIIT-D), Delhi, India.
E-mail address: monikas@iiitd.ac.in (M. Singh).

https://doi.org/10.1016/j.diin.2019.04.006
1742-2876/Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

suffers from a limitation that this kind of filtering only indicates an exact copy of another file. This is due to the fact that even a single bit change in the file content produces a completely unrelated and random-looking hash output whereas the requirement in practical scenarios is often to find *similar* files.

'Approximate Matching' is a generic technique for finding similarity among given files, typically by assigning a *'similarity score'*. This technique is used currently by digital forensic investigators. An approximate matching technique can be characterized into one of the following categories: *Bytewise Matching*, *Syntactic Matching*, and *Semantic Matching* (Breitinger et al., Roussev). Bytewise Matching measures the similarity of the digital object at the byte level without considering the internal structure of the data object. These techniques are known as fuzzy hashing or similarity hashing. Syntactic Matching defines similarity based on the internal structure of the data object. On the other hand, semantic Matching measures similarity based on the contextual attributes of the digital objects. It is also known as Perceptual Hashing or Robust Hashing.

ssdeep (Kornblum, 2006), sdhash (Roussev, 2010), mrsh (Breitinger et al., 2012a) and mvHash (Breitinger et al., 2013a) are some of the most prominent and commonly used approximate matching schemes. All of these schemes measure similarity at byte-level. Several studies (Baier et al., 2011) (Breitinger et al., 2012b) (Chang et al., 2015) (Chang et al., 2016) have shown that these schemes do not withstand an attack by active adversaries, i.e., attackers who can perform some minor but smart modifications to the file content (active manipulation) which causes predictable changes in the hash fingerprint of the content and thus allows bypassing the filtering process. This defeats the very purpose of these algorithms.

Our Contributions. We present a new approximate matching scheme which is secure against active attacks. We term our scheme as FbHash -Frequency Based Hashing. The idea of FbHash is based on the TF-IDF (Term Frequency - Inverse Document Frequency) concept of information retrieval (Ramoset al., 2003). TF-IDF is a statistical measure used to evaluate the importance of a word to a document in a collection or corpus. FbHash uses this notion to identify important fragments (features) of a document. A file fragment's contribution to the final similarity score is based on its importance or relevance as per this measure.

We also provide a comprehensive comparative analysis of FbHash with other prominent approximate matching approaches i.e.ssdeep and sdhash. We show that FbHash detects similarity with 28% higher accuracy for uncompressed file formats (i.e., text files) and around 50% higher accuracy for compressed file formats (i.e., docx). We also show that our proposed scheme is able to correlate a fragment as small as 1% to its source file with 100% detection rate and able to detect commonality as small as 1% between two documents with correct appropriate (low) similarity score and 100% detection rate. Further, our scheme also produces the least false negatives in comparison to other schemes.

We also observe that measuring similarity only at the byte-level does not allow a good match for compressed file format documents. Hence, we present two versions of our tool:

- FbHash-B. This version of our tool measures similarity at the byte-level. We show in section 7 that it can detect similarity with 98% accuracy for uncompressed file formats.
- FbHash-S. This version performs Syntactic matching and uses information about the internal structure of a document in order to measure similarity. This is recommended for compressed file formats.

Finally, we also provide security analysis of our scheme and show that FbHash is resistant against active adversary attacks.

The rest of the paper is organized as follows: We discuss related literature in Section 2. In Section 3, we present our scheme FbHash and its variant FbHash-B that works for uncompressed file formats. In section 4, we show how our scheme generates the final hash to calculate a similarity score. This is followed by Section 5, in which we present our FbHash-S scheme for finding similarity in compressed file formats. Section 6 presents the security analysis of FbHash followed by comparative analysis with other existing schemes in Section 7. Finally, we conclude our work in Section 8.

2. Related work

The first Approximate Matching approach for digital forensics was proposed by Nicolas Harbour (2002) in year 2002 called dcfldd. dcfldd is a block-based hashing scheme. In this scheme, each file is split into fixed size blocks and hash output is generated for each block. The final digest is a concatenation of all the block hashes. Later, an improvement upon dcfldd was proposed by Kornblum (2006) and was named "Context Triggered Piecewise Hashing" (CTPH). The CTPH scheme is based on an email detection algorithm called spamsum, proposed by Andrew et al (Tridgell, 2002). Instead of hashing fixed size blocks, CTPH divides the data in variable size blocks and then each block is hashed using a (non-cryptographic) hash function called FNV hash. This scheme was shown to detect similar files more accurately compared to block wise hashing schemes. The tool which implements CTPH is known as ssdeep, which is also the commonly referred name for the hashing scheme itself. Breitinger et al. in (Breitinger et al., 2012b) presented a thorough analysis of ssdeep and showed that ssdeep does not withstand an active adversary from evading blacklisting.

Roussev et al. (Roussev, 2010) proposed a new scheme called sdhash in the year 2010. The main idea in the sdhash scheme is to generate the final hash using only statistically improbable features of the document. Detailed security and implementation analysis of sdhash is presented in (Breitinger et al., 2012b) by Breitinger et al. This work uncovered several implementation and security issues and showed that it is possible to beat the similarity score by tampering a given file without changing the perceptual behavior of this file (e.g., image files look almost the same despite the tampering). The claims of (Breitinger et al., 2012b) were again verified by Chang et al. in (Chang et al., 2015). This work also presented an attack by which an adversary may mislead the investigator with multiple similar files. Furthermore, Roussev (Roussev (2011) has shown that sdhash outperforms ssdeep in terms of both accuracy and scalability.

Two new schemes known as bbHash (Breitinger and Baier, 2012) and mrsh-v2 (Breitinger et al., 2013b) were proposed by Breitinger et al. in the year 2012. However, because of the high runtime complexity bbHash is not practically useable. Breitinger et al. proposed another scheme in year 2013 called mvHash-B similarity preserving hashing (Breitinger et al., 2013a). The scheme works in three phases: first it compresses the input data using majority voting, then performs run-length encoding and then finally stores the fingerprint into Bloom filters. The 'B' in mvHash-B denotes the bloom filter representation of the similarity digest. In terms of performance, mvHash-B is one of the most efficient schemes among all the existing schemes with the lowest run-time complexity and small digest size. A thorough analysis of mvHash-B is presented by Chang et al. (2016). The paper uncovers the weakness of the mvHash-B scheme and shows that mvHash-B does not withstand an active adversary against blacklisting and also proposes an improvement to mvHash-B design to alleviate the weakness.

3. Construction of `FbHash` (`FbHash-B`) similarity hashing scheme

To facilitate better understanding of our scheme, we first define some important terms and notations that are used throughout the paper.

3.1. Notation and terminology

- Chunk: Sequence of k consecutive bytes of a document.
- ch_i^D : represents the ith chunk of a document D.
- Chunk Frequency: Number of times a chunk ch_i appears in a document D. Represented as $\mathrm{chf}_{ch_i}^D$.
- Document Frequency: The number of documents that contain chunk ch. Represented as df_{ch}.
- N: denotes the total number of documents in the document corpus.
- $\mathrm{RollingHash}(\mathrm{ch}_i)$: Rolling hash value of ch_i
- $\mathrm{ch}-\mathrm{wght}_{ch_i}$: Chunk weight of ch_i
- $\text{doc-wghtdoc}-\mathrm{wght}_{ch_i}$: Document weight of ch_i.
- $\mathrm{W}_{ch_i}^D$: denotes the chunk-Score of ch_i in document D.

3.2. Design of `FbHash-B`

Our scheme adopts the TF-IDF weighing method (Ramoset al., 2003) to find similar documents. The working of our scheme `FbHash-B` is divided into the following three steps:

3.2.1. Chunk frequency calculation

In this step, we first divide our document into certain blocks of bytes. We term each block as a *chunk*. The aim is to then calculate the number of times each chunk appears in the given document, i.e., calculate *chunk frequency*.

1. Let $\mathrm{D}=\mathrm{B}_0^D, \mathrm{B}_1^D, \mathrm{B}_2^D, \ldots, \mathrm{B}_{l-1}^D$ be a l byte long document, where B_i^D indicates the i^{th} byte of the document D. A chunk is a sequence of k consecutive bytes of D, where

$$\begin{aligned}
\mathrm{ch}_0^D &= \mathrm{B}_0^D, \mathrm{B}_1^D, \mathrm{B}_2^D, \ldots\ldots, \mathrm{B}_{k-2}^D,\ \mathrm{B}_{k-1}^D\\
\mathrm{ch}_1^D &= \mathrm{B}_1^D, \mathrm{B}_2^D, \mathrm{B}_3^D, \ldots\ldots, \mathrm{B}_{k-1}^D,\ \mathrm{B}_{k}^D\\
\mathrm{ch}_2^D &= \mathrm{B}_2^D, \mathrm{B}_3^D, \mathrm{B}_4^D, \ldots\ldots, \mathrm{B}_{k}^D,\ \mathrm{B}_{k+1}^D\\
&\vdots\\
\mathrm{ch}_i^D &= \mathrm{B}_i^D, \mathrm{B}_{i+1}^D, \mathrm{B}_{i+2}^D, \ldots\ldots, \mathrm{B}_{i+k-2}^D,\ \mathrm{B}_{i+k-1}^D\\
&\vdots\\
\mathrm{ch}_{l-k}^D &= \mathrm{B}_{l-k}^D, \mathrm{B}_{l-k+1}^D, \mathrm{B}_{l-k+2}^D, \ldots\ldots, \mathrm{B}_{l-2}^D,\ \mathrm{B}_{l-1}^D
\end{aligned}$$

2. To compute the frequency of each of the identified chunks in the document, rolling hash technique is used. A rolling hash is a non-cryptographic hash function which allows the rapid computation of hash of each of the consecutive chunks. The fast computation of the rolling hash is due to the fact that the hash computation of a chunk utilizes the hash of the previous chunk, with which the current chunk shares most of the data bytes.

   In our construction, we use the Rabin Karp rolling hash function (Broder et al., 1993), which calculates the hash value with a very simple function using multiplications and additions as shown below:

$$\mathrm{RollingHash}(ch_i) = B_i^D a^{k-1} + B_{i+1}^D a^{k-2} + B_{i+2}^D a^{k-3} + \ldots. + B_{i+k-1}^D a^0 \ \text{ modulus } \ n$$

$$\mathrm{RollingHash}(ch_{i+1}) = a{*}\mathrm{RollingHash}(ch_i) - B_i^D a^k + B_{i+k}^D \ \text{ modulus } \ n$$

where a is a constant, k is the chunk size, and n is a large prime number.

In our implementation, the value of $\mathrm{RollingHash}(\mathrm{ch}_i)$ is an unsigned 64-bit number, i.e., the rolling hash value lies between 0 to $(2^{64}-1)$. the byte value B_i and the constant a range between 0 and 255. This in turn puts a limitation on k as the value of k must satisfy the following relation:

$$B_i^D{*}a^{k-1} \le 2^{64}-1$$

As the maximum values of B_i and a can be 255, thus,

$$255{*}255^{k-1} \le 2^{64}-1.$$

The maximum value of k which satisfies the above equation is 7 as shown below

$$2^{64}-1 > 255{*}(255)^6 \approx 2^{56}.$$

Hence, we choose $k=7$.

3. Once the rolling hash value of a chunk is calculated, the frequency of each chunk will be computed by the number of times a rolling hash value appears. We make this observation by storing the rolling hash values in a hash table as follows:
   - Index of the hash table is the rolling hash value of a chunk
   - Value of the hash table is the number of times that rolling hash value (i.e., the chunk) appears in a document.
4. To guarantee that each unique chunk gets a unique rolling hash value, i.e., no collision happens, the value of n is taken as a prime number greater than 2^{56} (since, $256{*}256^{k-1}=2^{56}$)
5. Based on chunk frequency, a chunk weight will be assigned to each chunk, using the following formula:

$$\mathrm{ch}-\mathrm{wght}_{ch} = 1+\log_{10}\left(chf_{ch}^d\right)$$

Thus, the higher chunk frequency, the higher weight and vice-versa.[1]

3.2.2. Document frequency calculation

Document frequency of a chunk is the number of documents containing that chunk. The aim of this step is to identify the important chunks of the given document that can help identify it. Usually, the chunks that occur too frequently in a document have little relevance with respect to identifying the document. On the other hand, the less frequent chunks of a document are more important and relevant. Thus, there is a need to weigh up the effects of less frequently occurring chunks.

1. In order to calculate *Document Frequency*, a data-set of N document files has been taken (in our implementation $\mathrm{N}=1000$). The document Frequency of a chunk ch is referred as df_{ch}. Document frequency is being calculated as follows:
- Identify chunks of each document in the data-set.
- Calculate rolling hash of each chunk.

[1] The chunk frequencies are normalized.

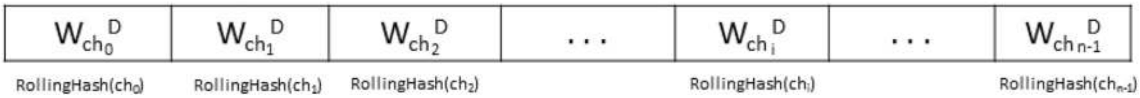

Fig. 1. FbHash digest.

- Create a hash-table where the index of the hash table indicates the rolling hash value of chunks and the value of the hash table indicates the document frequency of the corresponding chunk.
- Every unique chunk of each document will increase the value of the hash-table indexed by it's rolling hash by 1.

2. Based on the document frequency, a *document weight* will be assigned to each chunk indicating the informativeness or uniqueness of the chunk. We denote it as doc-wght$_{ch}$ and doc-wght$_{ch}$ is calculated as follows:

if df$_{ch}$>0: doc-wght$_{ch}$ $= \log_{10}(1000/\mathrm{df}_{ch})$.[2]

otherwise: doc-wght$_{ch}$ $= 1$

3.2.3. Digest generation

1. Once we have the chunk-weight and document-weight of each chunk in document *D*, a Chunk-Score (denoted is as $W^{D}_{ch_i}$) is then calculated as follows:

$$W^{D}_{ch_i} = \mathrm{ch} - \mathrm{wght}^{D}_{ch_i} * \mathrm{doc} - \mathrm{wght}_{ch_i}$$

This chunk score will be utilized to calculate similarity between two documents as shown later.

2. Now, the final FbHash digest of document D can be represented as a n element long vector where the index of vector represents the RollingHash(ch$_i$) and the value of the corresponding element is $W^{D}_{ch_i}$ as shown in Fig. 1. The index of the vector is represented by RollingHash(ch$_i$) because this value uniquely identifies ch$_i$. This is because, since we form 7-byte chunks, the total number of unique chunks can not be more than n.

For ease of explanation in this paper, we represent the FbHash digest as shown below.

$$\mathrm{digest(D)} = W^{D}_{ch_0},\ W^{D}_{ch_1},\ \ W^{D}_{ch_2}, \ldots, W^{D}_{ch_{n-1}}$$

The time complexity of FbHash-B digest generation is calculated as follows: The total chunks in a given document is $l-k$ where, l is the length of the document in bytes as mentioned at the start. Thus, computations of chunk frequencies in Section 3.2.1 will be done in $l-k$ steps. Steps involving document frequency calculation and assigning of document weights in Section 3.2.2 will be done offline and incurs no complexity in the online stage. Again, in Section 3.2.3, calculating the chunk score will be done in $l-k$ steps. Since the complexities of all the steps will be added, the overall complexity of FbHash-B digest generation will be $O(l)$, where, l is the length of the document in bytes.

4. Digest comparison and similarity score calculation

This section explains digest comparison and similarity score calculation of two documents. Let D_1 and D_2 be two documents and FbHash vector digest of D_1 and D_2 is as follows.

$$\mathrm{digest(D_1)} = W^{D_1}_{ch_0},\ W^{D_1}_{ch_1},\ W^{D_1}_{ch_2}, \ldots, W^{D_1}_{ch_{n-1}}$$

$$\mathrm{digest(D_2)} = W^{D_2}_{ch_0},\ W^{D_2}_{ch_1},\ \ W^{D_2}_{ch_2}, \ldots, W^{D_2}_{ch_{n-1}}$$

The similarity score between D_1 and D_2 is calculated using cosine similarity (Salton and Buckley, 1988) as follows:

$$Similarity(D_1, D_2) = \frac{\sum_{i=0}^{n-1} W^{D_1}_{ch_i} * W^{D_2}_{ch_i}}{\sqrt{\sum_{i=0}^{n-1} W^{D_1 2}_{ch_i}} * \sqrt{\sum_{i=0}^{n-1} W^{D_2 2}_{ch_i}}} * 100$$

Final similarity score ranges between 0 and 100. where, 100 indicates the files are exactly the same whereas score of 0 indicates no similarity.

5. Design of **FbHash-S**

The purpose of Fbhash-S is to find similarity in compressed file format documents, e.g., docx, pptx, pdf etc. During our experiments, we observed that similarity detection at the byte-level does not work for compressed documents. For example, if there are two docx files that have 90% similar content after compression, at the byte level there won't be any similarity with high probability. Thus, applying FbHash-B does not deliver good results.

The idea of FbHash-S is to use internal structure information of a document and perform syntactic matching to find similarity. Using the internal structure information of the document, FbHash-S first extracts the uncompressed content of the document. For example, in case of docx files, it will extract the text content (available in xml files stored in word folder) and images (stored in media folder under word folder). In our implementation we have used Apache POI package to extract text and images from the docx files. Then all the four steps, i.e., 1) *Chunk frequency calculation* 2) *Document Frequency Calculation* 3) *Digest Generation* 4) *Digest Comparison and Similarity Score Calculation* are peformed similarly as FbHash-B but individually on the text content and the images. Then the final score is the average of the score generated by text content and images. The run-time complexity of FbHash-S is higher than FbHash-B due to the additional step of content extraction.

6. Security comparison of **FbHash** with other schemes against active adversary attacks

ssdeep, sdhash, mrsh, mvhash are some of the most popular and prominent approximate matching schemes. Several papers have shown that these algorithms are not secure against active adversary attacks. In the subsequent part, we discuss attacks on each of the above mentioned schemes and explain why FbHash is not prone to these attacks:

ssdeep (Kornblum, 2006): The paper Security Aspects of Piecewise Hashing in Computer Forensics (Baier et al., 2011) by Baier and Breitinger shows an anti-blacklisting attack by performing intentional modification. ssdeep divides the input document into variable sized non overlapping blocks and then computes a

[2] 1000 denotes the total documents considered in the dataset.

cryptographic hash (e.g., md5) of each block, which then contribute to the final ssdeep hash digest. The blocks are generated based on some trigger points. The way ssdeep works, irrespective of the file size, the file will always be split into 64 blocks of variable size. Thus the final hash signature will also consist of 64 bytes only. Also, for ssdeep to detect a similar file to a known blacklisted file, the two hash signatures should have at least a common 7-byte substring in both.

- To evade detection, an attacker thus makes sure that such a common 7-byte substring is never found by making minor modifications in the malcious file's content. For example, in one of the attack scenarios, the attacker changes one byte in only the 7^{th} block, 14^{th} block, 21^{st} block (multiples of 7 blocks) and so on while preserving the trigger point locations to change the hash signature. In the other attack scenario, the attacker finds few global trigger sequences that will always create a trigger irrespective of the file size. Insertion of such global trigger sequences will lead to different blocks creation, which will change the hash signature completely and thus will help evading detection. The advantage of such attacks is that by making very small changes in the content, the hash signature can be changed significantly.
  However, in our case such attacks won't work. This is so because making small changes in the content will lead to creation of only few new chunks, having very low chunk frequency and thus low chunk score, preserving most of the high scoring chunks. In our similarity calculation, the low scoring chunks (i.e., the less relevant chunks) do not contribute much in the actual similarity comparison and the file will still be detected as similar to a known file with very high probability. In order to change the hash signature, the attacker will have to modify the majority of the high scoring chunks. In FbHash, as each chunk differs from its neighboring chunk by only one byte (the rest of the bytes are overlapping), in order to highly influence the final score, each chunk needs to be modified. Since the chunk size is 7-bytes only, in order to impact similarity score every 7th byte has to be modified. This will alter the content of the original document data significantly and the attacker's aim to make feasible changes will be defeated and thus of no use.

Sdhash: Breitinger et al. in their work titled - “Security and Implementation Analysis of the Similarity Digest sdhash” (Breitinger et al., 2012b) state that given a file, it is easily possible to tamper a given file to bring down the similarity score to approximately 28. Another paper titled - “A collision attack on sdhash similarity hashing” (Chang et al., 2015) by Chang et al. shows an anti-forensics mechanism that allows someone to generate multiple dissimilar files corresponding to a particular file with 100% sdhash similarity, which can confuse the filtering process. Both of the attacks are possible because the entire content of a file doesn't contribute to the final hash generation. Only some of the selected chunks participates in the final hash generation.

In our scheme, each and every byte of the document contributes to the final score (by formation of a new chunk) and their influence on the final score depends on their importance to the document. Hence, any modification will impact the final score. Further, to bring the similarity score really low or close to zero, almost every chunk has to be modified, which as discussed earlier will alter the content of the document significantly and make it altogether a different file.

mvhash-v The paper titled - “Security Analysis of MVhash-B Similarity Hashing” (Chang et al., 2016) shows that it is possible for an attacker to fool the algorithm by causing the similarity score to be close to zero even when the objects are very similar. The proposed attack is possible because mvhash compresses the input document using Run-length encoding (RLE). This gives the attacker freedom to bring the similarity score down with very few modifications.

No such compression is performed in FbHash. Every byte contributes to the final score calculation and hence our scheme is resistant to the attack.

7. Comparative evaluation of FbHash

In this section, we present a comparative analysis of Fbhash with the two most prominent approximate matching algorithms, (i.e., ssdeep and sdhash) on two test-cases: **Fragment Detection** and **Single-common-block correlation**. We chose these two algorithms for comparison as their reference standard implementation codes are available online and they are the most popular algorithms used by the forensics community. Section 7.1 describes the results of the Fragment Detection test, and the results of the Single-common-block correlation test are shown in Section 7.2.

7.1. Fragment detection

This test aims to identify the tool's ability to correlate a fragment (small part of a file) to its source file. We present a comparison between ssdeep, sdhash and FbHash performance. Fragments are generated in two ways - Sequential Fragments and Random Fragments, in a similar way as shown in (Breitinger et al., 2013c).

- **Sequential Fragments**: Create the fragment from the beginning of the file. For example, for a 1000 byte long file, a 1% fragment of a file is the first 10 bytes of the source file.
- **Random Fragments:** Generate the fragment from a randomly chosen position in the file. For example, for a 1000 byte long file, if the randomly chosen position is ‘*r*’, then the 1% long fragment is the next 10 bytes from *r*.

We perform the test on ‘Text data-set' and ‘Docx data-set' described in sections 7.1.1 and 7.1.2 respectively.

7.1.1. Text data-set result

Experimental Setup: The test is performed on a data-set of 960 fragment files (480 sequential fragments and 480 randomly generated fragments), generated from 20 variable size text files (5 KB to 1 MB taken from T5 corpus (Roussev)). Each text file generates 24 sequential fragment files and 24 random fragment files of the following sizes: 95%, 90%, 85%, 80%, 75%, 70%, 65%, 60%, 55%, 50%, 45%, 40%, 35%, 30%, 25%, 20%, 15%, 10%, 5%, 4%, 3%, 2%, 1%, $<1\%$. The total number of comparisons performed by each scheme for text files is thus 19 200.

Results

- The graph in Fig. 2 represents the results of ssdeep, sdhash and FbHash on Text data-sets.
  i. *X-axis* represents the different fragment sizes.
  ii. The *first Y-axis(left)* represents the *Match Percentage. Match percentage* or correlation detection rate is defined as the percentage of those test samples where, the tools are able to detect similarity by giving a valid match score (in other words, the number of times on a scale of 100, the tool is able to correlate fragments to their original source files for a given particular fragment size). This is illustrated in the form of lines in the graph. For example, in Fig. 2, for the fragment size 50%, ssdeep (represented by blue line) detects similarity between a fragment and its source file for 90% of the total

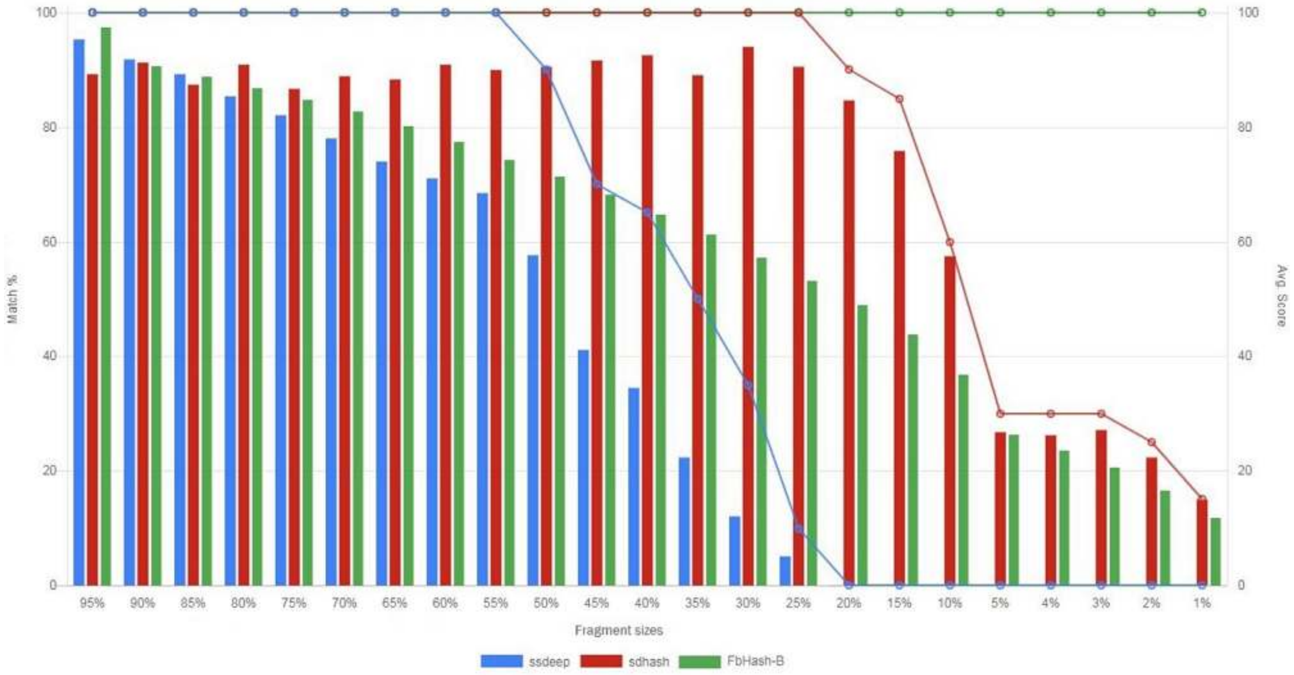

Fig. 2. Fragment detection test results on text data-set.

samples tested but fails for the remaining 10%. On the other hand, for the fragment size 50%, sdhash (red line) and FbHash-B (green line) are able to correlate the fragments to their original source files for all the samples tested, i.e., 100% correlation detection rate (due to overlap between the red and green line, only the red line is visible). Since the test is performed for the fragments as small as 1% of the file, hence any similarity score greater than 1 is being considered as a valid match in these experiments.

iii. The *second (right) Y-axis* represents the average similarity score calculated by the ssdeep, sdhash and FbHash-B between a fragment considered as one document and the original source file as the other document for a given fragment size. Bars in the graph illustrate the average score. For example, in Fig. 2 for 95% long fragments, we calculated the similarity score between each file and its 95% long fragment. The blue bar represents ssdeep, the red bar represents sdhash and FbHash is represented by the green bar. The averge similarity scores generated by ssdeep, sdhash and Fbhash for 95% fragments are 95, 89 and 97 (out of total of 100) respectively.

From Fig. 2, it can be seen that all the three tools show a 100% correlation detection rate for fragment sizes ≥ 55% (due to overlap only the horizontal blue line is visible). ssdeep can correlate a fragment to its source file if it is 50% or more of the source file with high correlation detection percentage, i.e., ≥ 90% of the times of the total samples tested. However, it cannot identify similarity for 20% or smaller fragment size. sdhash can detect similarity for fragment size of 15% or more with high percentage, i.e., ≥ 85% of the times for the total samples tested. However, its correlation detection rate drops to 60% or less as the fragment size decreases beyond 10% or less. On the other hand, the correlation detection rate for FbHash is 100% for all the fragment sizes, i.e., all the fragments that were tested were successfully correlated to their original source files even when the fragment size was as low as 1% as represented by the horizontal green line.

If we look at the right y-axis of Fig. 2, it can be seen that in case of sdhash, the relationship between the similarity scores predicted by the tool and actual similarity of the fragment to its source file is not consistent. For example, for fragment size 30%, the similarity score given by sdhash is comparatively higher than that given for fragment size 95% whereas it should be the reverse. This shows that sdhash similaritiy scores do not reflect the actual similarity. On the other hand, this relationship is correctly reflected by FbHash. It can be seen that as the fragment sizes decrease from 95% to 1%, the average score given by FbHash also decreases. This holds true for ssdeep as well up to fragment size ≥ 25%. However, beyond that ssdeep, cannot identify the similarity which is not the case for FbHash. FbHash shows the correct relationship even for fragments as small as 1%–5% of the file.

- **F-score:** We calculate the F-score in order to calculate the accuracy of the ssdeep, sdhash and FbHash-B. The F-score is a generic measure to test the accuracy of a tool that considers both the precision and recall values of the tool while computing the final score. The precision parameter signifies how many similar files were predicted similar by the tool and recall indicates how many similar files predicted by the tool were actually similar. Precision, Recall and F-score are calculated as follows:

$$F - score = 2*\frac{precision*recall}{precision + recall}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

where, *TP* refers to true positive, *TN* refers to true negative, *FP* refers to false positive and *FN* refers to False negative results generated by the tool. Let f1 and f2 be two given files and the similarity score

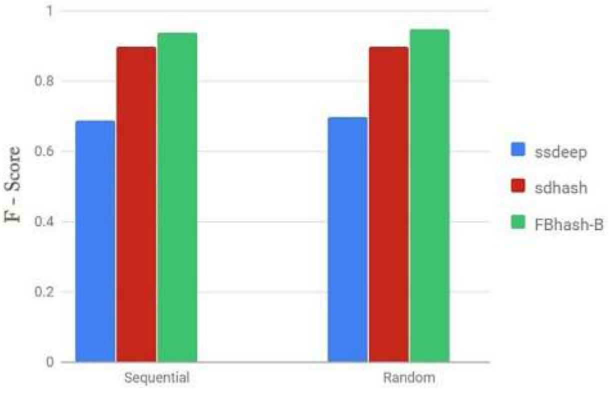

Fig. 3. Figure shows the F-score comparison for Fragment Identification test on Text Data-Set. The value of t is taken to be 22 for all three schemes.

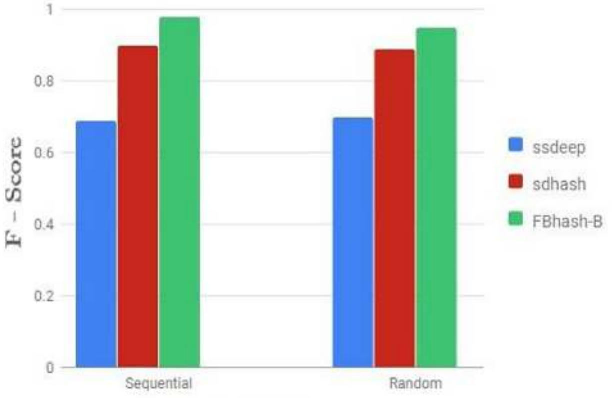

Fig. 4. Figure shows the F-score comparison for Fragment Identification test on Text Data-Set. The value of t is taken to be 22 for `ssdeep` and `sdhash` and 16 for `FbHash`.

generated by an approximate matching tool be represented as AM(f1,f2) which ranges between 0 and 100. Let t be a threshold value, defined later in this section. Since we have generated the data-set with known similarity, thus, we know the actual similarity in the files which we call as ground similarity represented by GS(f1,f2) (ranges between 0 and 100 where 0 indicates no similarity and 100 indicated f1 and f2 are identical). Any value of GS(f1,f2) > 0 indicates that f1 and f2 shares some similarity. The result of a tool is considered TP, TN, FP and FN according to the following conditions:

TP: if $\mathrm{GS} \geq 1$ and $\mathrm{AM}(f1, f2) \geq t$
TN: if $\mathrm{GS} < 1$ and $\mathrm{AM}(f1, f2) < t$
FP: if $\mathrm{GS} < 1$ and $\mathrm{AM}(f1, f2) \geq t$
FN: if $\mathrm{GS} \geq 1$ and $\mathrm{AM}(f1, f2) < t$

`t` represents the threshold value of the similarity score generated by a tool. It is considered that a tool has found a match if the similarity score generated by the tool is greater than or equal to t (i.e. AM(f1,f2)≥ t is a match). The paper (Roussev, 2010) claims that the threshold score of up to 22 yields near-perfect detection for `sdhash`. Since no such value is suggested for `ssdeep`, the value of t is taken to be 22 for all three schemes in order to compare the results. We observed that for t = 16 we get the best detection rate for `FbHash`. Thus we have shown F-score results of `FbHash` for both t = 22 and t = 16 shown in Figs. 3 and 4 respectively. Table 1 shows the TP, TN, FP, FN, precision, recall and F-score value generated by the experiment. A total of 9200 comparisons are performed for each sequential fragment and random fragment test case.

As the results show, all the three schemes have 0 False Positive Rate (FPR), however `ssdeep` has the highest false Negative Rate (FNR) and `FbHash` has the minimum FNR. Figs. 3 and 4 show that `FbHash-B` detects similarity with the highest accuracy of 98% with suggested threshold (16) and 95% with threshold 22, whereas the accuracy of `ssdeep` is 69% and `sdhash` is 89%.

7.1.2. *Docx data-set results*

We also test `ssdeep`, `sdhash` and `FbHash` for docx data-set. Following are the details of the experiment.

Experimental Setup: The test is performed on the data-set of 960 fragment files (480 sequential fragments and 480 randomly generated fragments), generated by 20 variable size docx files. Each docx file generates 24 sequential fragment file and 24 random fragment files of the following sizes: 95%, 90%, 85%, 80%, 75%, 70%, 65%, 60%, 55%, 50%, 45%, 40%, 35%, 30%, 25%, 20%, 15%, 10%, 5%, 4%, 3%, 2%, 1%, <1%. The fragments are generated only by segmenting (cutting) *content* of docx files into pieces. Total number of comparisons performed by each scheme for docx files is 19 200.

Results

- Fig. 5 shows the average similarity score and match percentage of `ssdeep`, `sdhash` and `FbHash-B` on docx Data-Set. The results obtained by all three algorithms are imprecise (inaccurate). As shown in Fig. 5, `sdhash` can detect similarity for all fragment sizes with higher match percentage compared to `ssdeep`. `FbHash-B` on the other hand is able to correlate even the smallest fragment with 100% detection rate (green horizontal line). However, the average matching scores of all the three algorithms do not reflect the actual similarity as fragment sizes decrease from 95% to 1%. Thus, none of these algorithms is useful.

Table 1
Fragment Identification test-case F-Score calculation for Text-Data set. Total number of comparisons performed for each sequential and Random fragments is 9200.

	ssdeep (t = 22)		sdhash (t = 22)		FbHash-B (t = 22)		FbHash-B (t = 16)	
	Sequential	Random	Sequential	Random	Sequential	Random	Sequential	Random
True Positive (TP)	244	246	373	373	408	419	438	442
True Negative (TN)	8740	8740	8740	8740	8740	8740	8740	8740
False Positive (FP)	0	0	0	0	0	0	0	0
False Negative (FN)	216	214	87	87	52	41	22	18
False positive rate (FPR)	0	0	0	0	0	0	0	0
False negative rate (FNR)	0.0234	0.023 26	0.009 404	0.0094	0.0056	0.0044	0.0023	0.0019
Precision	1	1	1	1	1	1	1	1
Recall	0.5304	0.5347	0.8108	0.8108	0.8869	0.9108	0.9521	0.9608
F-score	0.6931	0.6968	0.8955	0.8955	0.9400	0.9533	0.9755	0.9789

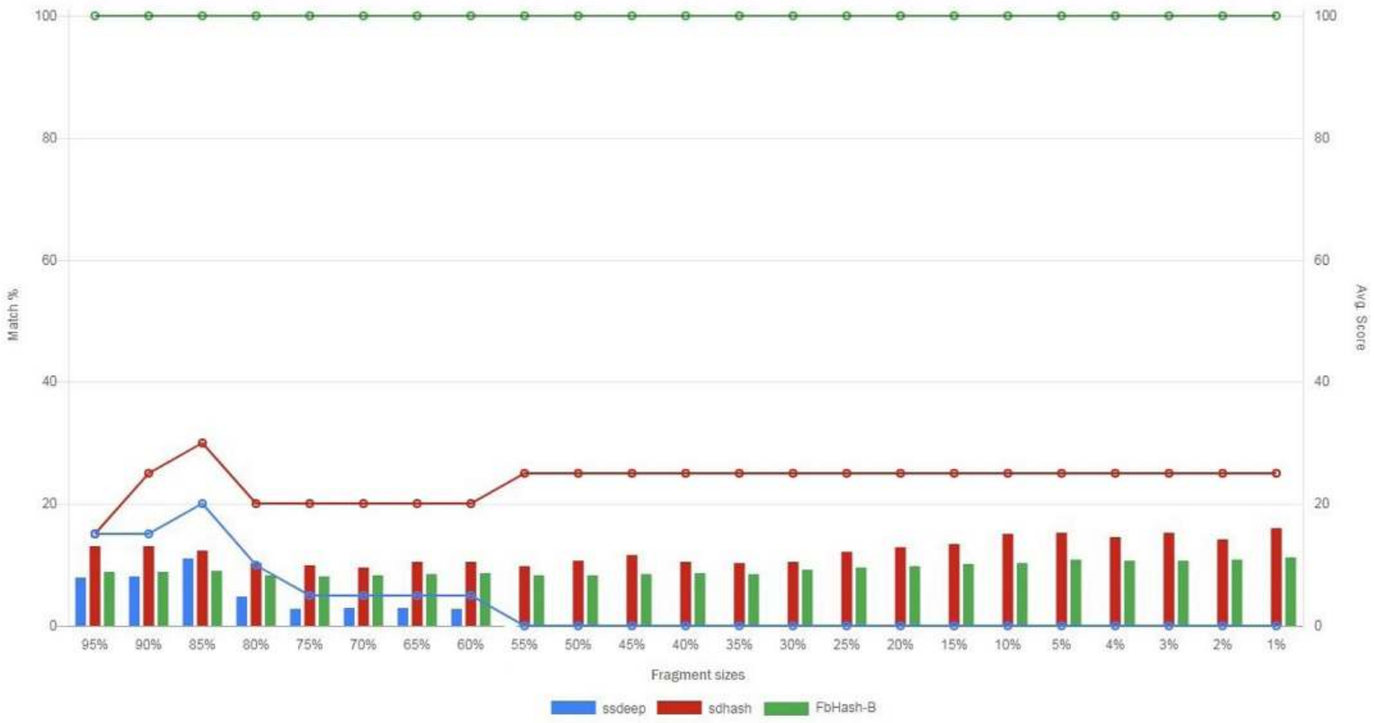

Fig. 5. Fragment detection test results on docx data-set.

The reason behind this is that docx is a compressed file structure due to which any modification in the content of a file changes the final compressed file completely with high percentage. Hence, at the byte level, the two different fragments or versions of a docx file are completely different. Since both `ssdeep` and `sdhash` work at byte level, the resultant similarity score is completely inaccurate. Hence, we state that byte-level matching is not sufficient to find similarity of compressed file structures. Fig. 6 presents a comparison between the results obtained by `ssdeep`, `sdhash`, `FbHash-B` and `FbHash-S`. As Fig. 6 shows, the results obtained by `FbHash-S` are accurate in terms of similarity score and its relationship to the actual similarity of a

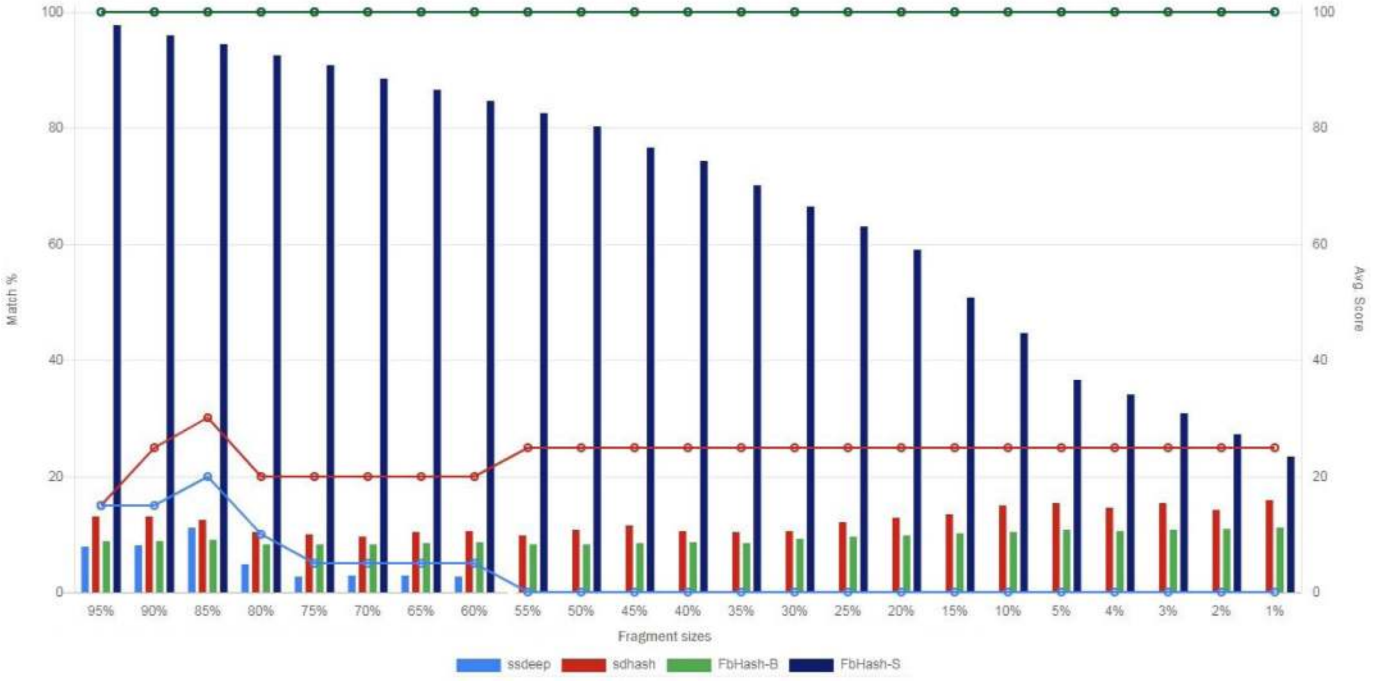

Fig. 6. `FbHash-S` fragment detection test results on docx data-set.

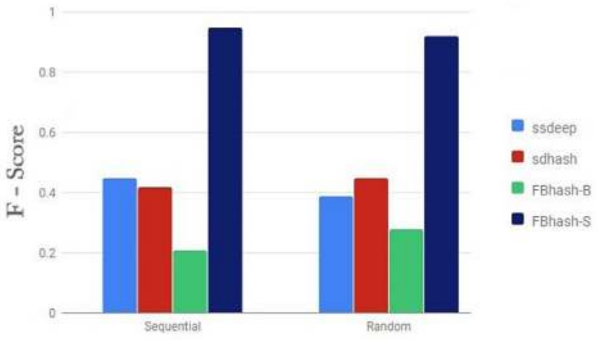

Fig. 7. Fragment Detection Test F-score comparison on Docx Data-set.

fragment to its source file, i.e., as the fragment sizes decrease, the scores also decrease.

- **F-score:** F-score is calculated similarly as explained in Section 7.1.1. Fig. 7 shows the comparison between the F-score of `ssdeep`, `sdhash`, `FbHash-B` and `FbHash-S`. It shows that `FbHash-S` outperforms `ssdeep`, `sdhash` and `FbHash-B` by 50%, 53% and 73% respectively higher accuracy for sequential fragments and by 53%, 47% and 64% respectively higher accuracy for random fragments. `FbHash-S` achieves accuracy of 95% and 92% for sequential and random fragments respectively.

7.2. *Single-common-block file correlation*

This test was first proposed in paper [Roussev, 2011] by Vassil Roussev. It aims to identify the ability of a tool to correlate the related documents, i.e., those which share a common single block of data. For this test-case as well we generated the ground truth data-set with known similarity. To generate the data-set, T5 corpus [NIST, 2008] is being used. the data-set is generated following the steps given below.

- 3 files of the same size are taken from the T5 corpus.
- The following 10 different sized fragments of the first file is created: 100%, 66.66%, 42.86%, 25%, 11.11%, 5.2%, 4.1%, 3.09%, 2.04%, 1.01%.
- Each fragment will be inserted in randomly chosen positions in the second and third file one by one. This will result in the creation of 10 pairs of the second and third file with shared common block of 50%, 40%, 30%, 20%, 10%, 5%, 4%, 3%, 2%, 1% respectively.
- Take another triplet of files and repeat from step 1 to step 3 for various file sizes.

We perform the test on 'Text data-set' and 'Docx data-set', and the results of the tests are shown in Section 7.2.1 and Section 7.2.2 respectively.

7.2.1. *Text Data-set result*

Experimental Setup: The test is performed on a data-set of 280 document pairs with a shared single common block, generated from 60 variable sized text files (5 KB to 10 MB).

Results: The graph in Fig. 8 represents the results of `ssdeep`, `sdhash` and `FbHash-B` on Text data-sets. The results show that `ssdeep` (blue line) can detect the similarity for ≥ 30% single-common-block similarity (commonality) with high percentage (≥ 75%). `sdhash` (red line) can detect similarity up to 3% single-common-block size with high percentage (≥ 93%) whereas `FbHash-B` (green line) can detect similarity up to 1% single-common-block size with high percentage (≥ 93%). `sdhash` and `FbHash-B` both perform well in this case and the similarity score generated by both the tools are very close to the actual similarity.

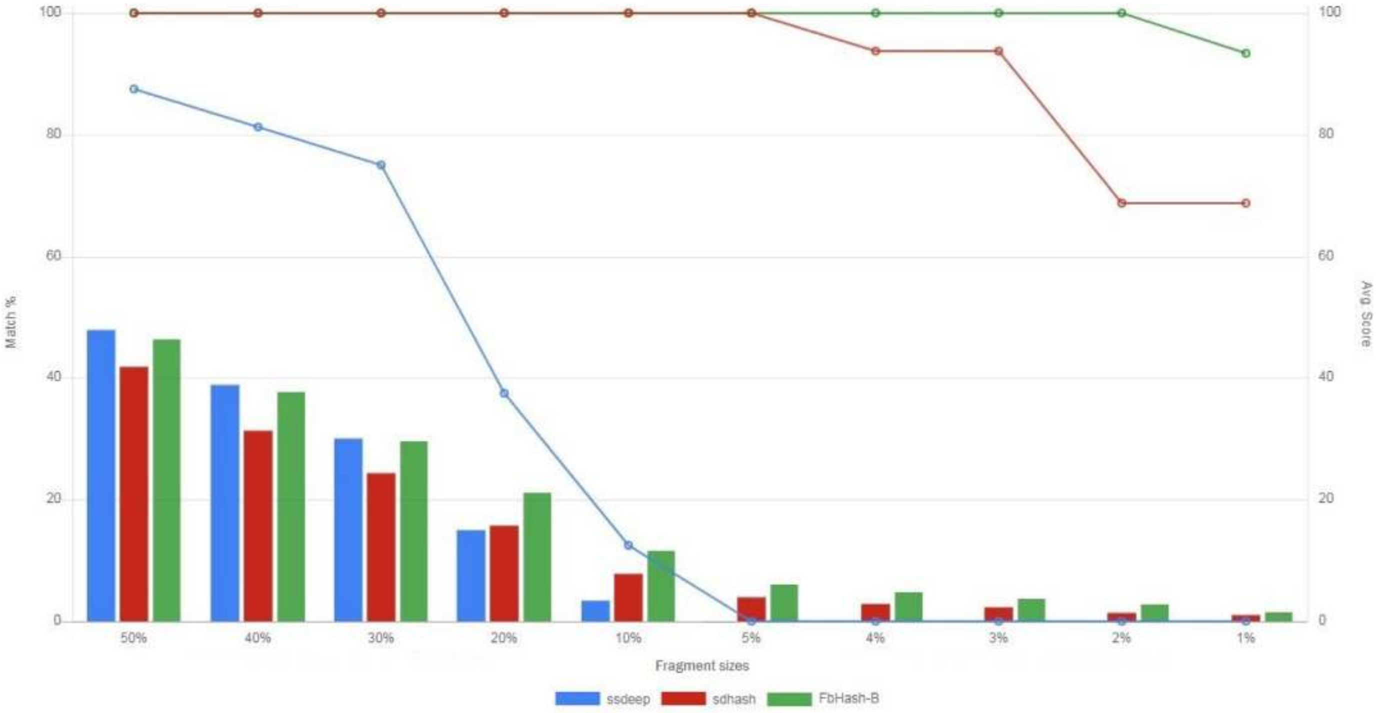

Fig. 8. Single-common-block file correlation Results for Text-Data Set.

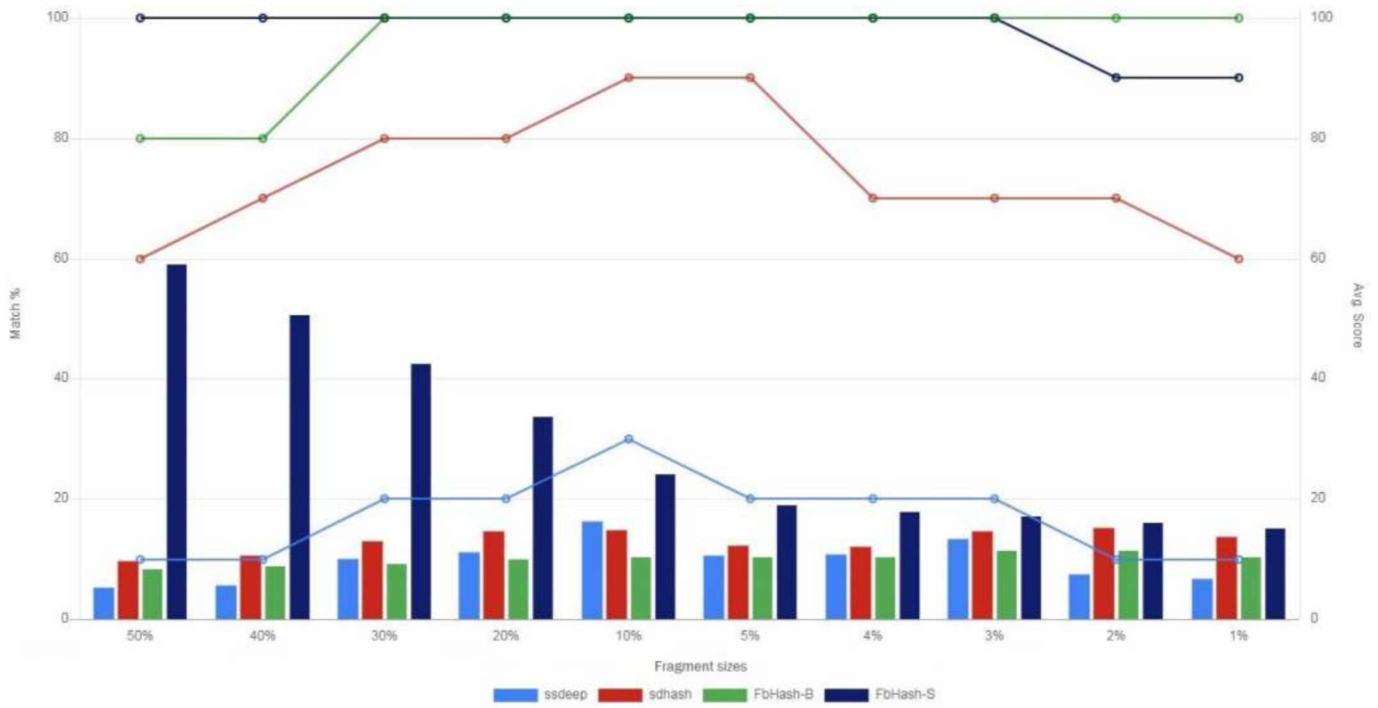

Fig. 9. Single-common-block file correlation Results for Docx data-set.

7.2.2. *Docx data-set results*

This subsection presents the results of the 'Single-Common Block Detection' test for docx data-set. Following are the details of the experiment.

Experimental Setup: The test is performed on a data-set of 280 document pairs with a shared single common block, generated from 60 variable size docx files (5 KB to 1 MB).

Results: Fig. 9 shows the average similarity score and match percentage of `ssdeep`, `sdhash`, `FbHash-B` and `FbHash-S` on Docx Data-Set. The correlation detection rate or the match percentage of `ssdeep`, `sdhash` and `FBHash-B` fluctuates with common block size and is not consistent. On the other hand, the correlation detection rate of `FBHash-S` is consistent and very high (≥ 90%) for all block sizes from 50% to 1%.

In terms of average matching score, it can be seen that results obtained by `ssdeep`, `sdhash` and `FBHash-B` are imprecise and do not reflect the actual similarity, since docx is a compressed file structure. On the other hand, results obtained by `FbHash-S` are more accurate and consistent.

8. Conclusions and future work

This work presents the first approximate matching scheme which is secure against active manipulations. The proposed scheme is able to correlate a fragment as small as 1% to the source file and able to detect commonlaity as small as 1% between two documents with correct/appropriate (low) similarity score. We experimentally demonstrate that the proposed approach provides 98% accuracy in some test cases. We intend to analyze the runtime performance of our tool on various data-sets as a future work. We also plan to explore the capabilities of our tool on a few other test cases such as embedded object identification, related document detection, etc. with different types of data objects (e.g., pdf, xml etc).

9. NIST disclaimer

The views and opinions expressed herein do not necessarily state or reflect those of NIST. Certain commercial entities, equipment, or materials may be identified in this document to illustrate a point or concept. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

Acknowledgments

We thank the anonymous reviewers for their helpful suggestions and feedback. Special thanks to Barbara Guttman, who provided helpful and insightful suggestions during the course of this work.

References

Baier, H., Breitinger, F., 2011. Security aspects of piecewise hashing in computer forensics. In: Morgenstern, H., Ehlert, R., Frings, S., Göbel, O., Günther, D., Kiltz, S., Nedon, J., Schadt, D. (Eds.), Sixth International Conference on IT Security Incident Management and IT Forensics, IMF 2011. IEEE Computer Society, Stuttgart, Germany, pp. 21–36. https://doi.org/10.1109/IMF.2011.16. May 10-12, 2011. https://doi.org/10.1109/IMF.2011.16.

Breitinger, F., Baier, H., 2012. A fuzzy hashing approach based on random sequences and hamming distance. In: Proceedings of the Conference on Digital Forensics. Security and Law, pp. 89–100.

Breitinger, F., Baier, H., 2012. Similarity preserving hashing: eligible properties and a new algorithm mrsh-v2. In: Rogers, M.K., Seigfried-Spellar, K.C. (Eds.), Digital Forensics and Cyber Crime - 4th International Conference, ICDF2C 2012, Lafayette, IN, USA, October 25-26, 2012, Revised Selected Papers, Vol. 114 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, pp. 167–182. https://doi.org/10.1007/978-3-642-39891-9_11. https://doi.org/10.1007/978-3-642-39891-9_11.

Breitinger, F., Baier, H., Beckingham, J., 2012. Security and implementation analysis of the similarity digest sdhash. In: First International Baltic Conference on Network Security & Forensics. NeSeFo).

Breitinger, F., Astebol, K.P., Baier, H., Busch, C., 2013. mvhash-b - a new approach for similarity preserving hashing. In: Seventh International Conference on IT

Security Incident Management and IT Forensics, IMF 2013. Nuremberg, Germany, pp. 33–44. March 12-14, 2013.

Breitinger, F., Baier, H., 2013. Similarity preserving hashing: eligible properties and a new algorithm mrsh-v2. In: Rogers, M., Seigfried-Spellar, K.C. (Eds.), Digital Forensics and Cyber Crime. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 167–182.

Breitinger, F., Stivaktakis, G., Baier, H., 2013. Frash: a framework to test algorithms of similarity hashing. Digit. Invest. 10, S50–S58. https://doi.org/10.1016/j.diin.2013.06.006. https://doi.org/10.1016/j.diin.2013.06.006.

Breitinger, F., Guttman, B., McCarrin, M., Roussev, V.. Approximate matching: definition and terminology. https://www.nist.gov/software-quality-group/national-software-reference-library-nsrl.

Broder, A.Z., 1993. Some applications of rabin's fingerprinting method. In: Capocelli, R., De Santis, A., Vaccaro, U. (Eds.), Sequences II. Springer New York, New York, NY, pp. 143–152.

Chang, D., Sanadhya, S.K., Singh, M., Verma, R., 2015. A collision attack on sdhash similarity hashing. In: Proceedings of 10th Intl. Conference on Systematic Approaches to Digital Forensic Engineering, pp. 36–46.

Chang, D., Sanadhya, S.K., Singh, M., 2016. Security analysis of mvhash-b similarity hashing. The Journal of Digital Forensics, Security and Law: JDFSL 11 (2), 21.

Harbour, N., 2002. Dcfldd. defense computer forensics lab, Net 5 (5.2), 5 article 1).

Kornblum, J.D., 2006. Identifying almost identical files using context triggered piecewise hashing. Digit. Invest. 3 (Suppl. 1), 91–97. https://doi.org/10.1016/j.diin.2006.06.015. https://doi.org/10.1016/j.diin.2006.06.015.

NIST, 2008. National Software reference Library [Online; accessed. https://www.nist.gov/software-quality-group/national-software-reference-library-nsrl. (Accessed 28 October 2018).

Ramos, J., et al., 2003. Using tf-idf to determine word relevance in document queries. In: Proceedings of the First Instructional Conference on Machine Learning, vol. 242, pp. 133–142.

Roussev, V.. The t5 corpus. http://roussev.net/t5/t5.html.

Roussev, V., 2010. Data fingerprinting with similarity digests. In: IFIP International Conference on Digital Forensics. Springer, pp. 207–226.

Roussev, V., 2011. An evaluation of forensic similarity hashes. Digit. Invest. 8, S34–S41. https://doi.org/10.1016/j.diin.2011.05.005. https://doi.org/10.1016/j.diin.2011.05.005.

Salton, G., Buckley, C., 1988. Term-weighting approaches in automatic text retrieval. In: Information processing and management, pp. 513–523.

Tridgell, A., 2002. Spamsum readme. https://www.samba.org/ftp/unpacked/junkcode/spamsum/README.

Donghoon Chang is currently an associate professor (CSE, Math) at IIIT Delhi, India. Before joining IIIT Delhi he worked as a guest researcher at NIST, USA. Dr. Chang received his Ph.D. degree in Information Management and Security from Korea University, Korea. His research interest includes Cryptanalysis, Biometric Security, Cyber Security.

Mohona Ghosh is currently working as an Assistant Professor in Department of Information Technology at Indira Gandhi Delhi Technical University of Women, Delhi. Prior to that she worked as Assistant Professor in CSE department at IIITDM Jabalpur. Dr. Mohona has completed her masters and Ph.D. from IIIT Delhi in Information Security. She did her Postdoctoral from NTU, Singapore. Her research interests include Symmetric Key Cryptography and its associated cryptanalysis and cyber forensics.

Somitra Kumar Sanadhya received a PhD from ISI Kolkata in 2009, M.Tech. from JNU, New Delhi in 2002 and B.Tech. from IIT Delhi in 1994. He is currently an Associate Professor in the Department of Computer Science and Engineering at IIT Ropar. His research interests include design and cryptanalysis of cryptographic primitives, their hardware implementations, and side-channel attacks.

Monika Singh is a PhD student at the Indraprastha Institute of Information Technology, Delhi. She is currently working as a Guest Researcher at National Institute of Standards and Technology (NIST)'s Software and Systems Division. Singh's research interests include digital forensics, information security, privacy, and cryptography.

Douglas White leads the National Software Reference Library project for the National Institute of Standards and Technology. He has over 20 years of experience with distributed systems, distributed databases and telecommunication protocols, real time biomonitoring, real time video processing, system administration and network monitoring. He holds both a B.A and M.S. in computer science from Hood College. He has given lectures for the American Academy of Forensic Sciences, the Federal Law Enforcement Training Center, the High Technology Crime Investigation Association, the Digital Forensic Research Workshop and numerous other digital forensic conferences.