available at www.sciencedirect.com

ScienceDirect

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

ELSEVIER

Digital Investigation

An evaluation of forensic similarity hashes

Vassil Roussev

Department of Computer Science, University of New Orleans, New Orleans, LA 70148, USA

Keywords:
Digital forensics
Similarity hash
Similarity digest
Sdhash
Ssdeep

A B S T R A C T

The fast growth of the average size of digital forensic targets demands new automated means to quickly, accurately and reliably correlate digital artifacts. Such tools need to offer more flexibility than the routine known-file filtering based on crypto hashes. Currently, there are two tools for which NIST has produced reference hash sets–*ssdeep* and *sdhash*. The former provides a fixed-sized *fuzzy* hash based on random polynomials, whereas the latter produces a variable-length *similarity digest* based on statistically-identified features packed into Bloom filters.

This study provides a baseline evaluation of the capabilities of these tools both in a controlled environment and on real-world data. The results show that the similarity digest approach significantly outperforms in terms of recall and precision in all tested scenarios and demonstrates robust and scalable behavior.

1. Introduction

The standard forensic practice of using file-, block-, volume-level cryptographic (md5/sha1) hashes to perform *known-file filtering* is increasingly running against the realities in the field, which require a more robust and flexible approach. In particular, crypto hashes *fail* in several important scenarios, which steadily erode their overall effectiveness as filters:

1) **Identification of embedded/trace evidence**. Given a piece of data, such as a JPEG, an investigator needs to able to search for (traces of) its existence inside another document, archive, disk image, or network trace.
2) **Identification of code versions**. Modern software is dynamically patched and upgraded on a daily basis; it is an infeasible to maintain crypto-hash inventory of all the files for every single version.
3) **Identification of related documents**. Many documents undergo changes/transformations as they are updated. It is often necessary to be able to identify and trace the versions across multiple evidence sources.
4) **Correlation of memory and disk sources**. An investigator needs to be able to correlate memory captures and disk images. The run-time layout and content of an executable/document are different from the on-disk representation so conventional hashes fail; however, identifiable commonality is clearly present.
5) **Correlation of network and disk sources**. Transmitted files are fragmented and interleaved. Currently, correlation requires time-consuming packet flow reconstruction and protocol parsing to extract transmitted files before any hash filtering can be applied.

The challenge for the next generation of forensic hashing tools is to address the above scenarios without incurring a prohibitive performance penalty. In other words, such tools must be capable of reliably determining similarity and correlation on various scales ranging from network packets and disk blocks to TB-class disk images.

Since 2006, a couple of tools–*ssdeep* (Kornblum, 2006) and *sdhash* (Roussev, 2010)–that attempt to address (at least some of) the above requirements have emerged and have reached

E-mail address: vassil@cs.uno.edu.

1742-2876/$ – see front matter © 2011 V. Roussev. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2011.05.005

a development stage that is mature enough for NIST to start generating reference hash sets for its RDS corpus (http://www.nsrl.nist.gov). Up to this point, there has been no systematic effort to quantify the behavior and performance of these tools and to understand their strengths and weaknesses. The purpose of this work is to quantify the performance of these tools and provide insight into their capabilities.

In general, similarity hashes work at the byte-stream representation and do not attempt to parse or interpret the data in any way. As such, there are only capable of detecting commonality among of the binary representations of digital artifacts. This raises two major questions that need to be studied:

- What kind of byte-stream correlations do these tools *actually* detect?
- How do detected correlations relate to human-perceived correlations between the same artificacts?

The former question allows us to understand what are the basic capabilities of the tools, whereas the latter examines how these capabilities relate to real-world situations. Note that both studies are needed in order to paint a complete picture: byte-level correlations do not necessarily map to syntactic/semantic correlations of interest and vice versa. Such understanding is critical for practitioners in the field; as well, it allows researchers to continually improve the tools and align them with real needs.

To answer these questions, we present two evaluation studies. The first one is a *controlled* one in which all data is generated from pseudo-random streams and the ground truth is known precisely. The second study is based on *real data* set of 4457 files in which we manually evaluate the correlation results produced by the tools.

2. Background: *ssdeep* and *sdhash*

The overall goal of a similarity tool is to be a drop-in replacement for the crypto hashes being used in forensic file practice for file filtering. Hash-based filtering involves hashing two data objects and comparing the results. Crypto hashes (by design) can only give simple yes/no answers, whereas a similarity tool provides a probabilistic answer–a number between 0 and 100. Note that the correct interpretation of the result is *not* as an estimate of percentage of overlap between the two objects. Rather, it is closer to a confidence level that the two objects have non-trivial commonality between them. Thus, one would expect a higher number to yield lower false positive rates.

In the following sections, we restrict ourselves to a high-level summary of the design and operation of the tools being evaluated. The lower-level details can be found in the referenced publications.

2.1. *Fuzzy hashes:* ssdeep

The roots of Kornblum's *ssdeep* (Kornblum, 2006) tool can be traced back to Rabin's seminal work on data fingerprinting with random polynomials (Rabin, 1981), which spurned a long line of research in the area of information retrieval. In the interest of brevity, we omit a detailed overview of the area (relevant discussion can be found in Roussev (2010)) and focus on the specific algorithm used by *ssdeep*.

The tool produces *context triggered piecewise hashes*, commonly referred to as *fuzzy hashes*. The idea is relatively simple: a) break up the file into pieces using the result of a rolling hash function; b) use another hash function to produce a (small) hash for each piece; and c) concatenate the results to produce the hash signature for the whole file.

Intuitively, we expect files that have common content would exhibit some level of similarity in their signatures, while unrelated ones would not. A key design decision here is how to break up the file into pieces such that signatures remain relatively resilient in the face of minor changes. Choosing, for example, to hash every 4K disk block of a file does *not* fit this requirement–the insertion of a single character at the beginning of the file would change all block hashes rendering our measurement too fragile.

The rolling hash function uses a small *context* of a few bytes to produce a pseudo-random value h_r. On every iteration, the context slides forward by one byte and if $h_r \equiv \bmod b-1, b$, where $b = 2^k$, then the context is used as a breaking point. Note that the rolling hash a very cheap to compute and that the parameter b is chosen such that the expected length of the signature does not exceed 80 characters. (If the signature exceeds the limit, b is recalibrated and the whole calculation is redone from the beginning). A traditional, non-cryptographic hash is used to produce a 6-bit hash for each piece in between breaking points. The results are concatenated and base64-encoded in the output.

To compare file signatures, the tool treats them as regular strings and employs an edit distance measure borrowed from early spam filters to determine a level of correlation. In essence, it expects that, for related files, it would take relatively few editing operations (insert/delete/change/swap) to transform one signature into the other.

2.2. *Similarity digests:* sdhash

sdhash (Roussev, 2010) takes an altogether different approach to produce, store, and compare its similarity hashes known as similarity digests (*sdhash* = **s** imilarity **d**igest **hash)**.

In short, *sdhash* tries to find from every neighborhood the features (64-byte sequences) that have the lowest empirical probability of being encountered by chance. Each of the selected features is hashed and placed into a Bloom filter (Bloom, 1970) (a probabilistic set representation). When a filter reaches its capacity, a new filter is created until all the features are accommodated. Thus, a *similarity digest* consists of a sequence of Bloom filters and its length is about 2–3% of the length of the input. The latter figure is based on the current implementation where a filter is 256 bytes long with 128 elements and represents (on average) about 7–8 KB of source data (higher densities are also practical).

Bloom filters have predictable probabilistic properties (Broder and Mitzenmatcher, 2002), which allow for two filters to be directly compared using a Hamming distance-based measure $D(\cdot)$. The result gives an estimate of the fraction of features that the two filters have in common that are *not* due to chance. To compare two digests, for each of the filters in the first digest, the maximum match among the filters of the second is found. The resulting matches are then averaged.

Formally, the similarity distance $SD(F, G)$ for digests $F = f_1 f_2 \ldots f_n$ and $G = g_1 g_2 \ldots g_m$, $n \leq m$, is defined as:

$$SD(F, G) = \frac{1}{N}\sum_{i=1}^{n} \max_{j=1..m} D\left(f_i, g_j\right)$$

It is notable that the empirical probability of encountering a 64-byte feature can neither be directly estimated nor could such observation be practically stored and looked up. Therefore, the tool computes a normalized Shannon entropy measure and places features into 1000 classes of equivalence. The statistics are collected using this approximation.

Computing the entropy has the added advantage of allowing some filtering of non-characteristic features to take place. The rationale here is based on the observation (Roussev, 2009) that real-world data formats contain non-trivial amounts of data that is either very low on information content (repetitive blocks), or is format-specific, not object specific (e.g., common header information). Such phenomena lower the effectiveness of Rabin-style fingerprinting by raising the false positive rate.

3. Controlled study

As discussed earlier, the purpose of the study is to evaluate the tools using simulated artifacts in which we precisely control the level of commonality between compared objects and know the ground truth. It would be extremely difficult to achieve such precision using real-world artifacts. The main advantage of this "lab" setup is that it gives us an upper bound on the capabilities of the tools under ideal conditions.

3.1. *Experimental approach*

In all the experiments, we use the same basic setup. We define a target as a piece of data obtained from a (pseudo-) random number generator. Thus, we can safely assume that no two independently generated targets have any content in common. An *embedded object* is a smaller piece of randomly (and independently) generated data that we embed in our target(s) with the explicit purpose of creating commonality. For example, if we take two 1 MB targets T_1 and T_2 and embed a 64 KB object O in each of them, we know that T_1 and T_2 have exactly 64 KB in common.

To produce statistically significant results we need to randomize both the content of the targets/objects and the location of the embedded objects. For all experiments, we use 25 iterations of 40 placement runs for a total of 1000 observations. For each iteration, we generate fresh target(s) and embedded object(s). For each placement run, we randomly embed the object(s) as per our scenario and run the tools accordingly.

We define similarity detection as successful if the tool produces a positive number. From preliminary work, we know that neither tool produces false positives for the uncorrelated targets we generated so the true positive rate is the main metric for the experiment. We define *reliable detection* as the ability to detect correlation on at least 95% of the runs (950 positive observations).

All experiments were performed using *ssdeep* 2.6 and *sdhash* 1.1, which were the current versions at the time of the experiments.

3.2. *Evaluation scenarios*

Based on our initial list of requirements, we considered the following three scenarios:

- **Embedded object detection**. This test aims to directly evaluate the ability of the tools to satisfy our first requirement. That is, we measure the ability of the tool correlate known embedded content and the whole object. For example, whether a given JPEG is embedded in any of set of files/disks.
- **Single-common-block file correlation**. This test simulates a situation where two files have a single common object. This corresponds to requirements 2) and 3) where the files share content. For example, documents sharing an image/ header/footer, or pieces of software sharing library code.
- **Multiple-common-blocks file correlation**. This test speaks to requirements 4) and 5) where commonality might be significant but fragmented. To get a sense of relative performance, we use levels of commonality that we know from the previous test to be well within the tools' capabilities, but split into smaller pieces. Generally, this is a more difficult task and demands more precision from the tools.

3.3. *Embedded object detection*

To evaluate the effectiveness of embedded/trace object detection, we gauge the sensitivity of the tool's correlation capabilities based on the size of the embedded object and the target:

Given a data object O of fixed size (file, disk block) embedded in a target T (file, disk image), what is the largest T for which O and T can be reliably correlated?

Ideally, a tool's performance should be *scalable* in that its detection rates should depend on the size of the embedded object, and *not* the size of the target. Formally, we define reliable detection/correlation as true positive rate exceeding 95% (i.e., at least 95% of the individual runs return a value greater than zero). By design, the false positive rate is zero in all cases.

Table 1 summarizes the experimental results; for each object size: it gives the maximum file size for which at least 95% true positive rate is achieved, for five different object sizes. In searching for the limits, we have incremented file size by 1/8 the size of the embedded object. This means, for example, that for object size of 64 KB, for *ssdeep*, the detection works reliably at 192 KB but fails our criterion at 200 KB.

For *sdhash*, the results given are for the maximum value **tested**. Based on the observed performance and its correlation algorithm, we expect that it would continue to reliably detect all embeds (of the given sizes) regardless of the size of the target. For *ssdeep*, it appears that the algorithm works reliably

Table 1 – Embedded object detection (higher is better).

Object Size (KB)	Max Target Size (KB): 95% Detection	
	ssdeep	sdhash
64	192	65,536
128	384	131,072
256	768	262,144
512	1,536	524,288
1,024	3,072	1,048,576

as long as the embedded object is no smaller than 1/3 of the size of the target.

To put the numbers in context, it is useful to consider some typical file sizes. We calculated average file sizes for six popular file types using a 50% sample (463,248 files) from the NPS GovDocs (Garfinkel et al., 2009) corpus. The results are given in Table 2

Based in the numbers, we can conclude that ssdeep would not, for example, be able to detect an average embedded JPEG (143 KB) inside an average compound document (doc/pdf/ppt/xls) which is 516-1,982 KB in size.

3.4. *Single-common-block file correlation*

Given targets T_1 and T_2 that share a common data object O, what is the smallest O for which the similarity tool reliably correlates the two targets?

Table 3 summarizes the results; for each target size (both targets are of equal size), it gives the minimum embedded object file size for which at least 95% true positive rate is achieved for file correlation.

It is evident that *sdhash* exhibits a predictable sub-linear growth trend for the minimum embedded O, whereas the growth for *ssdeep* fluctuates significantly. We have no reliable insight as why the latter is the case; our best guess is that the limit on the signature's length plays at least some role, and there are some difficult to predict effects from the editing distance metric.

3.5. *Multiple-common-blocks file correlation*

Given targets T_1 and T_2 that share multiple common data objects, what is the probability that the similarity tool will correlate the two targets?

We simulate the embedding of a total amount of common data that is 50% of the target's size but is split into four and eight (non-overlapping) pieces, respectively. Each piece is then independepently and randomly placed into the targets. For example, for the 256 KB case we consider the embedding (at random) of 4 pieces of 32 KB and 8 pieces of 16 KB. Table 4 shows the fraction of runs that produce results greater than zero.

Table 2 – File size statics from GovDocs corpus.

Average File Size in KB					
doc	jpg	pdf	ppt	txt	xls
575	143	516	1,982	1,067	1,118

Table 3 – Single-common-block file correlation (lower is better).

Target Size (KB)	Min Object Size (KB): 95% Detection	
	ssdeep	sdhash
256	80	16
512	160	24
1,024	320	32
2,048	512	48
4,096	624	96

The average *sdhash* score was consistently 17–18 for all 4-piece cases and 13–14 for the 8-piece ones; ssdeep scores dropped from 25 to 35 straight to zero with no readings in between.

3.6. *Observations*

Our controlled experiments lead us to several essential observations:

- The correlation capabilities of *ssdeep* appear crucially dependent on a large, continuous block of common data. On average, a contiguous chunk of 1/4 to 1/3 of the size of the files is necessary to guarantee detection for scenarios 1 and 2. Scenario 3 demonstrates that fragmentation of the common data significantly impedes detection and makes it unreliable.
- The correlation capabilitied of *sdhash* appear considerably less dependent on a large common block and tend to be correlated with the known level of commonality.
- At the margin of detection capability, *ssdeep* non-zero scores tends to drop off from the 25–35 range straight to zero; we were virtually unable to produce scores in the 1–20 range for any of the experiments.
- At the margin, *sdhash* scores approach zero gradually, giving a meaningful signal that detection capability is approaching its limits; this supports filtering and prioritization of the results.

4. Real data study

The main purpose of this study is to run the similarity tools under realistic conditions and evaluate their utility and

Table 4 – Multiple-common-blocks correlation probability (higher is better, 1.0 is optimal).

Targets	Common	4 pieces		8 pieces	
(KB)	(KB)	ssdeep	sdhash	ssdeep	sdhash
256	128	0.35	1.00	0.04	1.00
512	256	0.48	1.00	0.04	1.00
1,024	512	0.39	1.00	0.07	1.00
2,048	1,024	0.59	1.00	0.17	1.00
4,096	2,048	0.96	1.00	0.54	1.00

effectiveness in scouting out related files. Simply put, we run the tools and then manually examine the results and decide whether each identified correlation is real, or not.

One inherent impediment of this type of study (and a major reason we undertook the controlled study) is that it is infeasible to establish the ground truth on any set of non-trivial size. Specifically, we cannot afford to *manually* examine all of the almost 10 million unique pairs of files in our sample and classify them as positives/negatives. Nonetheless, the evaluation scenario is representative of the real-world in which examiners need to perform a similar classification task with no prior knowledge. Further, this approach allows an apples-to-apples comparison of the relative performance of the tools.

4.1. *Experimental approach*

We used a sample of the publicly available GovDocs corpus (http://domex.nps.edu/corp/files/govdocs1/). These are real files obtained by spidering US Government websites and are free of copyright restrictions.

For our test set, which we refer to as **t5**, we used all files between 4 KB and 16.4 MB in directories 000–004. Files below 4 KB (about 300) were eliminated since they contained a lot of web server error messages and corrupted files. A few very large files were also eliminated to avoid skewing the results (*ssdeep* needs file sizes to be relatively close to each other before it can compare them). We ended up with 4457 files (1.8 GB); the actual file sample used is available for download at http://roussev.net/t5/t5-corpus.zip. Table 5 provides a quick summary by file type:

The size of the sample was based on some preliminary runs and estimates of the number of file correlations that the experimenter would have to manually review. The sample was chosen from a sequence of neighboring directories since we expect that to result in groups of genuinely correlated files to be included. (Generally, files that are closer in numbering to each other are more likely to have been retrieved from the same server.)

4.2. *Evaluation procedure*

The sole criterion we use for determining true correlation is *syntactic commonality*. Neither tool performs any kind of semantic analysis; therefore, any correlation needs to be visible rather quickly. For web pages, this almost always means a common template (header/footer/navigation); for office documents, and across different file types it means common images, or large blobs of common text. Our evaluation process was blind in that a common list of pairs was generated from both tools and was examined without knowing where a candidate pairing originated.

Table 5 – T5 corpus file frequencies by type.

doc	gif	html	jpg	pdf	ppt	text	xls
533	67	1093	362	1073	368	711	250

Before we proceed with the actual evaluation, we need to understand the meaning of the tools' output. They both produce a number between 0 and 100 with the implication that a lower number means lower confidence level. Therefore, we tried to objectively establish a threshold value (point of diminishing returns) below which we should ignore any positive results as the rise in false positives starts to overwhelm the rise in true positives.

4.2.1. ssdeep *threshold*

Since no prior data was available, we studied the distribution of scores for both true and false positive in an effort to discover a pattern. Fig. 1 shows all scores obtained. It is apparent that *ssdeep's* similarity metric does not show any obvious threshold point and that false positives are scattered over a wide range.

It is also evident that there is just a single score left of 24. This is entirely consistent with the behavior we observed during the controlled experiment so we feel confident this is not a function of the underlying data.

Another observation is the "comb" shape of the histogram—some 1/3 of the possible score values are entirely "missing". This is not a problem, per se, as it appears to be a function of the fact that, in the final output computation, a number between 0 and 64 is scaled to the 0 to 100 range.

4.2.2. sdhash *thresholds*

The *sdhash* tool exhibits altogether different behavior as shown on Fig. 2. As we approach the vicinity of zero, the number of scores ramps up exponentially. To keep the manual examination to a reasonable level, we decided not to consider any scores *below five*.

Based on prior work (Roussev, 2009), which suggests that a threshold value around 20–21 is a good choice, we examined all file pairs for which the *sdhash* score was 12 and above. We found that 21 is, indeed, a good choice for the threshold as illustrated by Fig. 3. Further, we noticed that for text files (including *html*), the threshold can be dropped all the way down to *five* with no appreciable growth in false positives (Fig. 4).

To summarize, for ssdeep, we use no threshold value; for *sdhash* we use a threshold of *five* if one of the files is a text file,

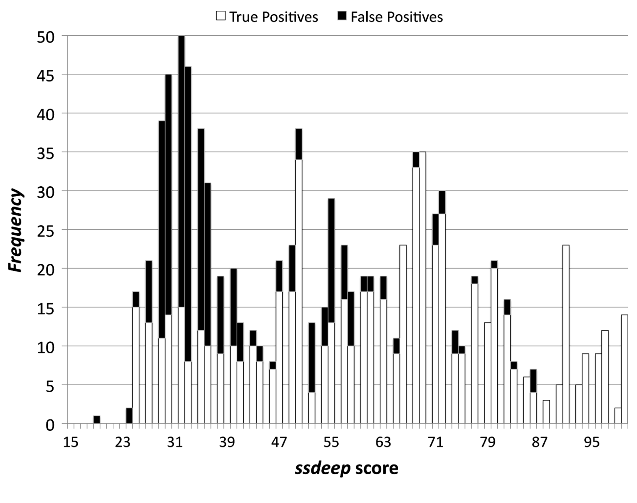

Fig. 1 – Distribution of *ssdeep* scores.

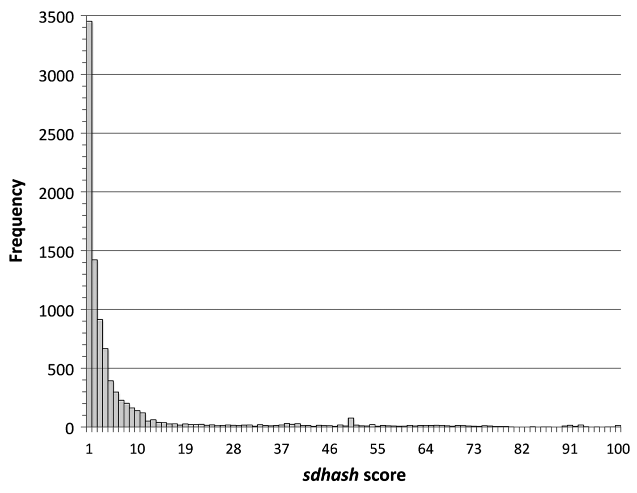

Fig. 2 – Distribution of *all sdhash* scores.

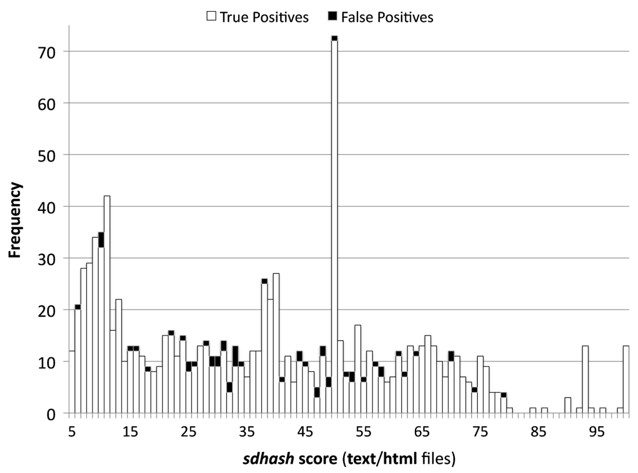

Fig. 4 – Distribution of *text/html sdhash* scores.

and 21, otherwise. (If a score is below the threshold, we count it as if it were zero.) We should also note that the difference in thresholds makes some intuitive sense—unlike text, compound formats tend to have common data that is attributable to the file format, such as headers, fonts, and metadata, that is not file-specific. Thus, such files have ‘built-in’ commonality that is likely not significant from an investigative perspective.

4.3. *Results and observations*

Using the threshold values in the previous section, we manually reviewed a total of 1699 unique file pairs for syntactic correlations (commonality). Table 6 provides some brief statistics. The “common” numbers refer to correlations that have produced the same (positive/negative) result in both tools.

The main results are summarized in Table 7, which gives the number of true and false file correlations for each of the tools, broken by file type. The ‘total’ column gives the total number of known true positive correlations identified by either tool. The vast majority of known correlations (98.65%) are between files of the same type. The next to last row (‘Mixed’) summarizes the cross-type correlations (both true and false ones). The last row (‘All’) is the sum of all numbers by type less the cross-type totals (the latter correlations are counted twice— one for each type).

As an example, the first row should be interpreted as follows: a total of 53 correlations involving at least one *doc* file were identified by manual examination; *ssdeep* has successfully identified 40 of these (true positives) and has incorrectly pointed at another 31 pairs (false positives); *sdhash* has identified 51 pairs at the cost of 7 false positives.

Based on the above numbers, we can calculate the standard information retrieval metrics of *recall* and *precision*. The one caveat mentioned earlier is that we do not know the ground truth and cannot reason about absolute measures of true/false negatives. Therefore, we have to make the assumption that the only positives are the ones discovered by either of the tools and that, if a correlation is not discovered by a tool, then it is non-existent. This works for our purposes as we are trying to describe the performance of the tools *relative* to each other, rather than in absolute terms w.r.t. ground truth.

Recall rates measure the proportion of actual positives which are correctly identified as such. In our case, this translates into the fraction of true positives identified by *one* method relative to the true positives identified by *either* method (Fig. 5). In the extreme, if one method shows a ratio of

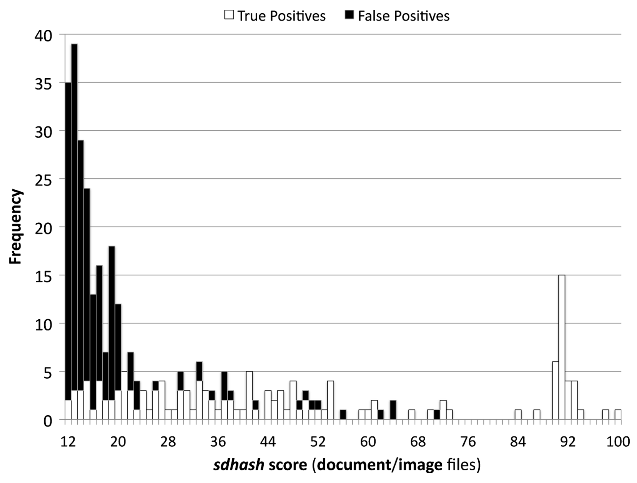

Fig. 3 – Distribution of *document/image sdhash* scores.

Table 6 – Evaluation statistics.

Total correlations examined	1,699
Total unique files examined	820
True positives	1,203
Common true positives	588
False positives	496
Common false positives	65

Table 7 – *ssdeep* vs. *sdhash* : True, false, and total known positives.

Set	*ssdeep*		*sdhash*		Total
	TP	FP	TP	FP	
doc	40	31	51	7	53
html	550	47	985	52	1043
jpg	0	23	0	2	0
pdf	39	28	45	25	46
ppt	15	6	25	0	25
text	9	0	23	0	27
xls	2	199	11	3	11
Mixed	2	24	16	18	58
All	653	310	1124	71	1189

one, it means that it completely subsumes the results of the other (Fig. 6).

It is clear that sdhash consistently outperforms *ssdeep* across the board with overall recall rates of 95% and 55%, respectively. Another way to look at this data is that running both tools offers minimal increase (6%) in additional discoveries (from 1124 to 1189) over running *sdhash* alone. At the same time, running both tools leads to an 82% jump in identified correlations (653–1189) over running ssdeep alone.

Precision is a complimentary measure and can be thought of measuring the cost of achieving the recall rate in terms of false positives. Precision is defined as the ratio of true positives for the method relative to *all* outcomes (true and false) for the method. An ideal method would have no false positives and would yield a perfect precision rate of one.

sdhash consistently outforms ssdeep across all categories, and with overall precision rates of 94% and 68%, respectively. One particularly sore point for *ssdeep* appear to be MS Office documents with *xls* files providing an extreme example of how wrong things can go. Upon close examination, the *xls* files triggering the false positives are a group of small, 20–30 KB files that are otherwise unrelated. We believe that these files contained a lot of sparse data that confuses *ssdeep*; *sdhash* considers such data uncharacteristic and leave it out in picking its signature features.

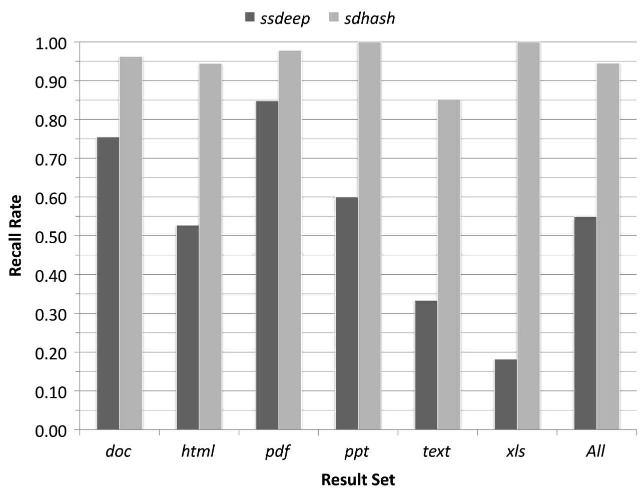

Fig. 5 – *ssdeep* vs. *sdhash* : Recall rates.

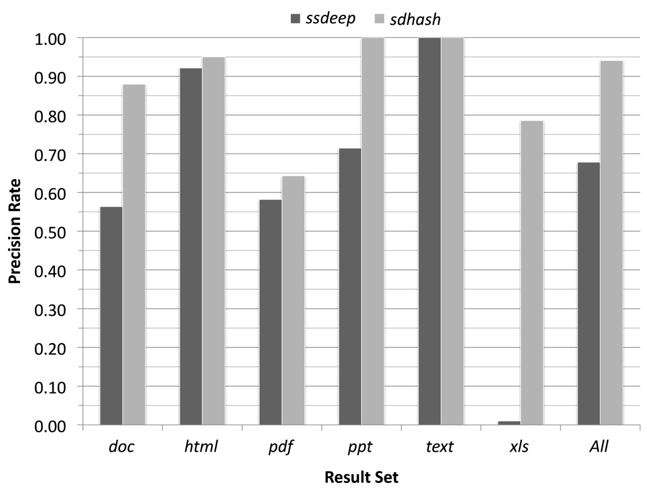

Fig. 6 – *ssdeep* vs. *sdhash* : Precision rates.

It is worth mentioning that MS Office results relate only to the original binary formats (doc/xls/ppt) and *should not* be extrapolated to the newer (OOXML) standard based on compressed XML (docx/xlsx/pptx). (The latter were not included in the study simply because they were not part of the GovDocs corpus.) Of the presented formats, OOXML files most resemble *pdf* but additional work needs to be done to understand the performance on OOXML.

5. Conclusion

ssdeep In all cases, the correlation algorithm is highly dependent on the presence of a large, continuous chunk of common data. The fixed-size hash does not scale well and the suitable range of target parameters is relatively narrow. Once outside the comfort zone, similarity scores and correlation ability dropped rapidly. There was no easily recognizable relationship between the known level of similarity and the similarity score making filtering and prioritization problematic.

On balance, *ssdeep* shows considerable limitations in its ability to satisfy most of the requirements. In our view, such shortcomings are a combination of methodological and design choices.

sdhash In all cases, the tool demonstrated accuracy and scalability with respect to target sizes, on average, outperforming ssdeep by a wide margin. Average similarity scores are highly correlated with the level of data commonality, allowing for filtering and sorting. Scores decreased gracefully as the detection limits were approached. It appears that its filtering mechanism gives it a significant precision advantage when dealing with MS Office documents. That can be explained by the fact that such files tend to have large blobs of low-entropy data that confuse methods that do not filter.

On balance, *sdhash* demonstrated the potential to address all five of the design requirements although further evaluation is necessary to confirm that in working with real network and RAM captures.

REFERENCES

Bloom B. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 1970;13(7):422–6. doi:10.1145/362686.362692.

Broder A, Mitzenmatcher M. Network applications of bloom filters: a survey. In: Annual Allerton conference on communication, control, and computing, Urbana-champaign, Illinois, USA; October 2002.

Garfinkel S, Farrell P, Roussev V, Dinolt G. Bringing science to digital forensics with standardized forensic corpora. In: Proceedings of the digital forensic research conference (DFRWS); 2009. p. S2–11.

Kornblum J. Identifying almost identical files using context triggered piecewise hashing. Digital Investigation 2006;vol. 3(S1):S91–7.

Rabin M. Fingerprinting by random polynomials, technical report TR1581, center for research in computing technology. Cambridge, Massachusetts: Harvard University; 1981.

Roussev V. Building a better similarity trap with statistically improbable features. In: Proceedings of the 42nd Hawaii international conference on system sciences. Waikoloa Village Resort, Hawaii, HI: IEEE; Jan 2009.

Roussev V. Data fingerprinting with similarity digests. In: Chow K-P, Shenoi S, editors. Advances in digital forensics VI, IFIP AICT, 337; 2010. p. 207–25.