Digital Investigation 11 (2014) S18–S26

Contents lists available at ScienceDirect

Digital Investigation

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

Practical use of Approximate Hash Based Matching in digital investigations

Petter Christian Bjelland*, Katrin Franke, André Årnes [1]

Norwegian Information Security Laboratory (NISlab), Gjøvik University College, Norway

Keywords:
Digital forensics
Approximate Matching
Evidence analysis
Data discovery
Malware forensics

A B S T R A C T

Approximate Hash Based Matching (AHBM), also known as *Fuzzy Hashing*, is used to identify complex and unstructured data that has a certain amount of byte-level similarity. Common use cases include the identification of updated versions of documents and fragments recovered from memory or deleted files. Though several algorithms exist, there has not yet been an extensive focus on its practical use in digital investigations. The paper addresses the research question: *How can AHBM be applied in digital investigations?* It focuses on common scenarios in which AHBM can be applied, as well as the potential significance of its results. First, an assessment of AHBM for digital investigations with respect to existing algorithms and requirements for efficiency and precision is given. Then follows a description of scenarios in which it can be applied. The paper presents three modes of operation for Approximate Matching, namely *searching*, *streaming* and *clustering*. Each of the modes are tested in practical experiments. The results show that AHBM has great potential for helping investigators discover information based on data similarity. Three open source tools were implemented during the research leading up to this paper: *Autopsy AHBM* enables AHBM in an existing digital investigation framework, *sddiff* helps understanding AHBM results through visualization, and *makecluster* improves analysis of graphs generated from large datasets by storing each disjunct cluster separately.

Introduction

The focus of this study has been to assist investigators in law enforcement to organize and analyze digital evidence using *Approximate Matching*. The term Approximate Matching refers to the technique of detecting data that are in some way similar. Though tools for performing Approximate Matching of raw data have been known for some time, they are still not integrated in popular digital investigation tools. Approximate Matching has been a standalone capability only used under special circumstances and not as part of standard investigation practices. Why AHBM is not yet in widespread use is difficult to determine, however, two reasons may be dominant: *No integration with existing digital investigation tools*, and *limited knowledge of the potential gains of using it*.

There are three types of Approximate Matching: perceptual, content and hash based matching. While the two latter focus on identifying data that is similar from the perspective of a computer, perceptual matching identifies data that is similar from the perspective of a human. Whereas perceptual matching is well suited for comparing pictures and videos, content and hash based matching algorithms are designed to match binary data, such as documents, executables, memory dumps and network traffic. Hash based matching groups chunks of data and compare them with chunks in other files. Content based matching

* Corresponding author.
E-mail addresses: petter.bjelland@hig.no (P.C. Bjelland), katrin.franke@hig.no (K. Franke), andre.arnes@hig.no (A. Årnes).
[1] The author A. Årnes is also associated with Telenor Group.

http://dx.doi.org/10.1016/j.diin.2014.03.003

computes the exact difference between two files, often using techniques such as Hamming distance (Hamming, 1950) and Levenshtein distance (Levenshtein, 1966). All these types of Approximate Matching may be relevant for digital investigations.

Related to this study is the work by Vassil Roussev and Candice Quates on hash based matching using empirical models for identifying, representing and matching chunks of data with their tool *sdhash* (Roussev, 2010, 2011). They also present an evaluation and comparison of existing AHBM tools in Roussev (2011). Most other published work on AHBM techniques focuses on the technical aspects of how hash based similarity can be measured. An exception is the forensic investigation of the M57 dataset (DigitalCorpora.org, 2009) by Roussev and Quates (2012). The authors describe how *sdhash* can be used to analyze large amounts of data, with a particular focus on reducing the amount of data subjected to human analysis. They present three scenarios in which Approximate Matching can be applied: Detecting the presence of contraband, detecting unauthorized copying of internal data, and detecting unauthorized exfiltration of data. The study focuses on evidence detection and what questions an analyst may ask using AHBM techniques. In contrast, this study focuses less on the inner workings of any particular tool, and instead attempts to define a general modus operandi for Approximate Matching when applied to digital investigations.

This research also has resemblance to cross-evidence correlation techniques different from AHBM, such as large scale data triage (Garfinkel, 2013) and malware identification (Flaglien et al., 2011).

For the purpose of the experiments in this paper, *sdhash* was used as the approximate hash based matching tool, rather than *ssdeep*. This is because *sdhash* yields the most robust and accurate results, as shown in Breitinger et al. (2013) and Roussev (2011). However, as *ssdeep* is more efficient when matching files without specialized hardware (Breitinger et al., 2013) (*sdhash* comparison efficiency depends on whether the POPCNT CPU instruction is available or not), it may be the preferred tool in many situations.

A system for comparing AHBM algorithms, named FRASH[2] was proposed by Breitinger et al. (2013). This work is important for the introduction of Approximate Matching as an integrated part of digital investigations, but current efforts have been limited to reviewing the types of similarities that may be discovered. In order for these techniques to become widely applied in digital forensics, there is a need to explore when and how to approximately match available evidence to discover new information.

This paper addresses these issues by describing the common scenarios where AHBM may add unique information to an investigation. Through practical experiments, the relevance of using the tools in the defined scenarios are reviewed and discussed.

During the research leading up to this paper, three tools were implemented and made available in order to help performing AHBM and analyzing its results. First, a module for performing AHBM in the digital investigation framework Autopsy (Carrier, 2012) was implemented.[3] The module, called *Autopsy AHBM*, allows an investigator to easily perform AHBM searching and streaming during disk image analysis. Second, a tool called *sddiff*[4] was implemented to help understand similarity through visualization. Finally, *makecluster*[5] was implemented to enable analysis of individual clusters by storing each connected cluster in separate files. The tool makes it easier to analyze graphs generated from large datasets, both in terms of computational complexity and visualization.

Fig. 1. Two semantically and perceptually similar files. The two files are not syntactically similar. Color to the left and grayscale to the right. (Photography captured by Petter Chr. Bjelland, 2013).

In the following, Section What is similarity? addresses the philosophical question: What is similarity? Section Approximate Matching describes in detail the various types of Approximate Matching. Section Modes of Approximate Hash Based Matching describes the different modes in which AHBM can be used, and what knowledge and insight may be achieved using modes. Then, Section Practical scenarios with AHBM describes scenarios for what types of data may be analyzed using these modes. Finally, the last two sections complete the paper with subjects for further research and conclusions.

What is similarity?

A word frequently used when discussing Approximate Matching is *similarity*. Investigators may use Approximate Matching to discover the presence of files *similar* to something we already know. So what is similarity and how do we measure it? There are essentially two different ways in which two files can be *similar*: syntactic and semantic. Syntactic similarity is from the perspective of a computer, and semantic similarity is from the perspective of a human.

Two documents are semantically identical if they communicate the same information. For example, a Microsoft PowerPoint presentation is semantically identical to an exported PDF document containing the same pages. Their cryptographic hashes[6] will not be identical, though we can still argue that the documents are the same. A similar concept applies to media, like pictures and videos.

[2] FRASH: A framework to test algorithms of similarity hashing.

[3] https://github.com/pcbje/autopsy-ahbm.

[4] https://github.com/pcbje/sddiff.

[5] https://github.com/pcbje/makecluster.

[6] A cryptographic hash is a digital fingerprint, making it possible to determine whether to pieces of data are exactly the same.

Most people would consider two pictures to be identical even though they are stored in two different formats. The meaning of two documents can also be identical even though the contents are presented in two different formats, e.g., the same set of numbers represented as a table or as list.

Semantically similar documents do not need to be similarly represented on a hard drive, but when presented they will appear similar to a human. Syntactically identical documents are represented identically on a hard drive and $h(A) == h(B)$, where A and B are the two documents, and h is a cryptographic hash function.[7] An example of two semantically similar pictures that are not syntactically similar is given in Fig. 1.

Approximate Matching

In order to understand how Approximate Matching can be applied during digital investigations, there is a need to review some theory on the types of Approximate Matching.

Perceptual similarity

Perceptual hashing aims at detecting objects that are similar from the perspective of a human. Typical object types include media such as pictures and videos. In the world of perceptual hashing, the process of Approximate Matching is often referred to as *media authentication* (Zauner, 2010). To authenticate an object is to determine whether or not it is the same as another object. To define what constitutes two similar, or authenticated, objects, Zauner (2010) describes two different actions that can be performed on a media: *modification* and *manipulation*. Modification is an action performed on the object that should not impact the authentication of the object, whereas manipulation is another type of action that impacts later authentication.

An example of a modification may be to adjust the color balance in a picture, and an example of manipulation is covering large portions of a picture with a filled rectangle, making it perceptually hard to recognize the original picture. An example of modification (two perceptually similar pictures) is given in Fig. 1.

A technology available for doing measuring perceptual similarity in pictures is Microsoft's PhotoDNA. A brief introduction to the technology is provided in Microsoft (2009). Facebook is using PhotoDNA to detect unwanted pictures (Microsoft, 2011).

A common use case for perceptual hashing is content searching. Detecting copyright infringement by scanning pictures on the Internet was mentioned as an example by Meixner and Uhl (2006). Being able to detect semantically similar pictures may in turn help investigators determine the source of the original picture, by examining how the picture has been spreading.

[7] Cryptographic hash functions are one-way, non-injective functions that maps an arbitrary size input to a fixed size output. Given a hash function h and input x, it is easy to compute $h(x) = s$ and computationally infeasible to determine x, given s (Buchmann, 2004).

Content similarity

In addition to hash-based matching, there are other techniques for matching objects that are similar in their binary representation without the use of cryptographic hashes. This section reviews two of them: Hamming distance and Levenshtein distance.

The Hamming distance algorithm (Hamming, 1950) compares two data streams by computing the minimum amount of substitutions needed to transform one stream into the other. The algorithm compares two streams element by element, where both elements have the same position in their respective stream. If the two elements differ, the distance between the two streams is incremented by one. However, if an element is inserted or removed to either stream, the comparison process will not be able to recover and their distance will become large. The Levenshtein distance algorithm (Levenshtein, 1966), also referred to as Min-Edit distance, measures the minimum amount of insert/edit/delete operations required to transform one stream into the other. A significant difference from Hamming distance is that Levenshtein distance is able to handle inserts and deletions in either stream. The algorithm needs to remember all previously computed distances and are thus not memory efficient, making it less suitable for comparing large streams of data.

Both algorithms are important for numerous other algorithms, including AHBM algorithms like *sdhash* and *ssdeep*.

Hash based similarity

Hash based matching measures the syntactical similarity between two files, not by interpreting the perceptual similarity, but by evaluating byte level commonalities in data. Due to the fact that two pictures can look identical but have different encoding and therefore be very different on byte level, AHBM is not suited for the task of measuring similarity between pictures. The benefit of measuring similarity on byte level is that it enables measurement of unknown content types and therefore allows Approximate Matching of complex and unstructured data such as documents, memory images and network packets.

Although encoding makes hash based matching unsuited for measuring similarity in images and videos, it can still provide great value when analyzing fragments from memory or deleted files. A fragment of a picture recovered from memory or unallocated space may be identical to the corresponding fragment in a picture found elsewhere. The same concept applies when looking for traces of known objects in a memory or disk image. An investigator may for example suspect that a certain software has been installed and later deleted from an hard drive. Approximate hash based algorithms may then be used to create a reference set from this software and match the disk image against this reference set. The reference set will usually contain executables, libraries and other resource used by the target software.

The most significant algorithms for computing hash-based similarity are currently *sdhash* (Roussev, 2010) and *ssdeep* (Kornblum, 2006). Other algorithms have been

proposed, but none have yet gained as much scrutiny in the digital forensics community. Comparisons between the tools are given in Roussev (2011) and Breitinger et al. (2013). Both algorithms look for statistical improbable chunks of bytes within the files. These improbable chunks are called features and are used to compute the similarity between two files. A highly simplified figure of hash based matching algorithms are shown in Fig. 2. In this figure, *FileA* has five features and *FileB* has four features. As *FileB* has the least features, and three of its four features are *identical* to features in *FileA*, the two files do therefore have similarity score (3/4)*100 = 75. Common output of Approximate Matching tools are similarity scores in a range from 1 to 100.

AHBM algorithms may detect any type of byte level similarity, however it may be useful to put these types into groups like alternative versions of the same document and embedded objects. Alternative versions of a file are for example two revisions of the same Microsoft Word document, or a log file captured at two different points in time. The files communicate the same information, and the differences between them may add new knowledge about their purpose to the investigator. For example is the possibility to detect attempts of fraud based on suspicious changes in spreadsheet documents.

The hash based similarity caused by embedded objects can be elements such as a company logo in PDF documents. The investigator may also be looking for embedded objects identical or similar to a flash object extracted from a malicious PDF file.

Modes of Approximate Hash Based Matching

This chapter describes the different modes in which AHBM can be used. There are three such modes: *searching*, *streaming* and *clustering*. After describing these modes in detail, the next chapter will describe scenarios where these modes are used.

Searching

A common approach to analyzing digital evidence is searching. With search, the investigator has some lead to follow up on. Typical types of leads are e-mail addresses, telephone numbers, credit card numbers and phrases such as names, organizations and places. When searching with AHBM, files are used as leads rather than text phrases. A conceptual overview of the searching process is illustrated in Fig. 3.

For this kind of search to be efficient, it is necessary that $N \ll M$, where N is the number of leads and M is the number of documents in the search space.

The results from searching can be interpreted as alternative locations of the lead. In a large investigation, such alternative locations may be in network traffic, memory dumps, shared folders and disk images. Table 1 illustrates potential results and interpretations of searching with AHBM with, e.g., a PDF document as lead.

Streaming

A definition of streaming is to pass continuous data to a function. An example of such functions is intrusion detection or prevention systems, where the functions match flowing data against a rule set describing malicious activity. Streaming matches large number of input documents against a smaller search space, and may therefore be considered an opposite of searching. The search space is in this mode referred to as the reference set. A conceptual overview of the streaming process with AHBM is illustrated in Fig. 4.

An AHBM algorithm is a potential streaming function. The streaming context may be network traffic, monitoring of new and modified files in an enterprise network, or the contents of a disk image. If an investigator suspects that some content may reside on a hard drive, it is easy to generate approximate hash based signatures for such files and stream evidence through AHBM.

For such streaming to be efficient, the reference set should be small compared to the amounts of data flowing through the streaming function, i.e., $N \gg M$.

Clustering

Matching a dataset against itself is called clustering. Such matching enables detection of direct and indirect similarities among elements in the data. Clustering is different from the two other modes in that the input and the search space is the same. A conceptual overview of the clustering process with AHBM is illustrated in Fig. 5.

The goal of clustering is to identify groups in data where the members in each group are more similar to each other

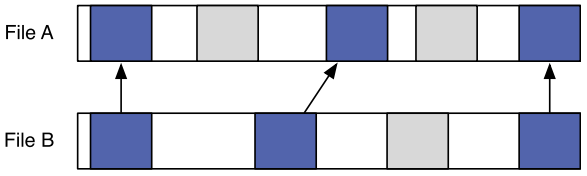

Fig. 2. A highly simplified approximate hash based matching of two files. A box represents the position of an extracted feature. Blue fill indicates that both files contain the exact same feature. Gray fill indicates that the given feature is only present in one of the files. Every feature is a fixed size chunk of bytes. The amount of identical features are used to compute the similarity between the two files. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

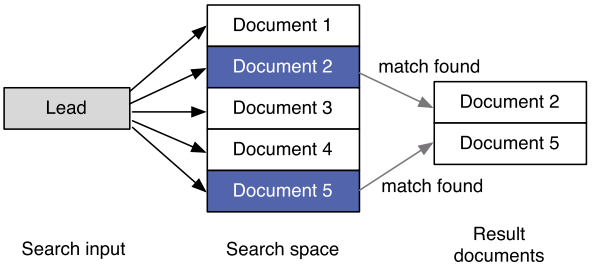

Fig. 3. Conceptual diagram of searching with AHBM. The search document is matched against every document in the search space using, e.g., *sdhash*, and the results are extracted and presented to the user.

Table 1
Example results and interpretations from searching with Approximate Matching and a PDF document as search input.

Match type	Similarity score	Possible interpretation
Network traffic	100	The search input has been uploaded to the Internet.
File in shared folder	67	The match is an alternative version of the search input.
Disk image	100	The search input was present on the employee's computer.
Disk image	32	Fragments of the search input may have been discovered in unallocated space.
Working memory	100	The search input was open on the employee's computer when the memory capture was performed.

than members in other groups. A clustering function is used to measure these distances and for digital evidence AHBM may be used as a such function. The closeness scores are computed for each document pair and similar files are added to each others edge list.

Indirect similarity is when $A \approx B$ and $B \approx C$, but $A \not\approx C$, where $\approx$ represents an approximate match. Such similarities may represent various types of relations between A and C. E.g., in malicious PDF samples, A may share objects with B, but not with C and B may share a different set of objects with C, potentially indicating that A and C has the same origin.

Note that clustering is usually far more computationally expensive than searching and streaming. Both searching and streaming require N^*M comparisons and clustering requires N^*N comparisons. If $N == M$, then both searching and streaming are as expensive as clustering, however, with streaming and searching it is assumed that either $N \gg M$ or $N \ll M$, as discussed earlier.

Practical scenarios with AHBM

Although tools like *sdhash* may provide great value to an investigation, it may also be necessary to visualize matches to better understand the results. For the purpose of this study, a tool called *sddiff* was implemented and used to visualize the position of similarity between two files to better understand the significance of their similarity. The tool uses a feature extraction algorithm based on *sdhash*, though the identified features are not inserted into Bloom filters[8] like *sdhash* do. Instead, *sddiff* records the position of each feature and then compares positions among the two files. If the same feature is present in both files, a blue, vertical bar is written to the output. The position of the vertical bar along the *X*-axis is determined by the position of the similar feature in the larger of the two files. The results of comparing these two files are printed as a picture, where the leftmost and rightmost pixel represents the beginning and the end of the largest file, respectively. See Fig. 7 for an example.

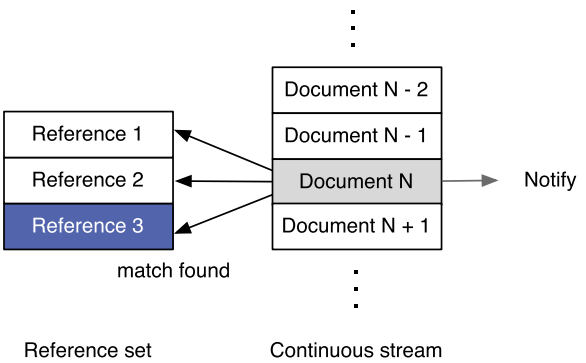

Fig. 4. Conceptual diagram of streaming with AHBM. Documents that pass through the matching function are matched against the reference set and extracted if a similarity is found.

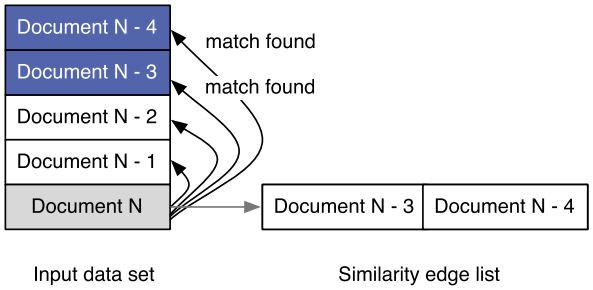

Fig. 5. Conceptual diagram of clustering with AHBM. Each input document is matched against all previous inputs. The results are stored as an edge list.

Search – alternative versions detection

The following experiment will describe how AHBM can be used to search through large datasets. The dataset used in this experiment consist of 50,000 emails from the Enron dataset (Klimt and Yang, 2004). These emails were located on the computers of 17 different users and were organized into 508 distinct folders. For the purpose of this experiment, eight randomly chosen emails found on the computer of nine different users are selected as leads. These emails are assumed to be emails found by some other means, e.g., by keyword search, and are matched against the other 50,000 emails in the dataset using similarity threshold 25[9]. The asked question was *do we have any other similar emails to this one?* The majority of the resulting matches fell into one of these three manually defined categories:

- *Reply*: When the number of matches where either email is a follow up on another email.
- *Similar conversation*: When emails with different subjects and content, sent to and from the same set of email addresses.
- *Different header*: When identical emails found in different folders.

[8] Bloom filters are probabilistic functions to determine whether an element may belong in a collection (Bloom, 1970).

[9] The similarity threshold 25 was chosen because lower thresholds yielded e-mails having only the header in common, i.e., similar recipient and folder. This was considered to be a false positive.

Table 2
Results from searching against the Enron dataset using eight randomly chosen emails as search input.

ID	Reply	Similar conv.	Diff. header	Total
1	0	0	2	2
2	17	2	0	19
3	6	0	0	6
4	0	0	5	5
5	0	0	3	3
6	0	2	2	4
7	5	0	0	5
8	8	1	1	10
Total	36	5	13	54

Detailed results from the experiments are listed in Table 2. The results from analyzing the Enron dataset in this manner depend on the selected seed. Though only a single seed was considered in this experiment, the results do illustrate the concept.

As Table 2 shows, the largest portions of the results were some sorts of a reply. In many cases, there are multiple participants spawning new conversations with new participants or with a subset of the existing participants. In these cases, no single email file contains all the available information related to the input email, and an analysis of several files is therefore required. In addition, there are several duplicate emails in the dataset. It is quite common for the same email to reside in several folders on a single computers, causing a relatively large portion of the matches to fall under the *different header* category. There were also several cases where a single reply to the search input was found in several folders. In these cases the matches were considered to be a reply. From a search input yielding six matches in the reply-category, Fig. 6 was manually generated.

The gray boxes in Fig. 6 represent content not present in the lead document. The ability to easily discover the complete flow of email correspondence in a large, unstructured dataset is a valuable feature when searching. Such emails may otherwise be difficult to discover.

Streaming – file transfer detection

This section describes a simple experiment where AHBM is used as a streaming function for matching a larger stream of data against a small set of references. In this scenario we are trying to determine whether cryptographic software was downloaded by analyzing recorded network traffic. The downloadable installer, as well as the unpacked files from the installer is used as a reference, while a network packet capture file (PCAP) is used as the streamed input. It is important to note that streaming in this context does not refer to a continuous flow of bits, but rather a flow of recorded network traffic sent to the matching function at some defined interval or threshold.

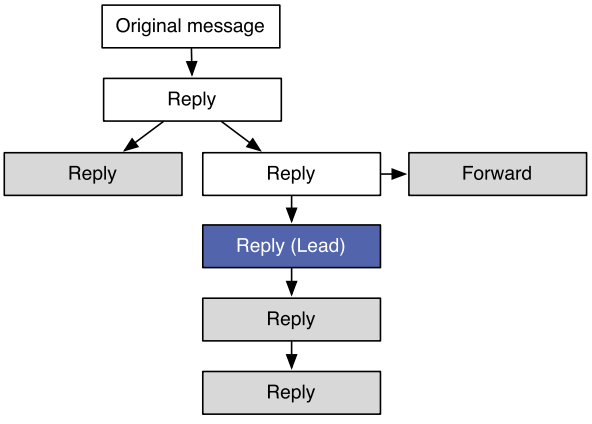

Fig. 6. A figure showing how search with AHBM can discover alternative follow ups on the same e-mail. The lead was the input to the search query.

Table 3
Results when matching the PCAP-file against a set of re-source files related to a cryptographic tool.

Role	Filetype	Similarity score
Installer	Executable	31
Installed tool	Executable	1
User guide	PDF	2
Tool formatter	Executable	2

Such capture and storage of network traffic is an aspect of organizational forensic readiness that enables post-incident investigation. A description of how network traffic may be recorded in a forensic context is presented in Garfinkel (2002). Finally, large scale network traffic storage and analysis is gaining attention with big data technologies like *packetpig* (PacketLoop, 2013) for, e.g., intrusion detection purposes.

In this experiment, a virtual machine is generating real network traffic that is captured by a network sniffing tool running on the host machine. The generated traffic is a mix of web surfing and a download of the installer for a cryptographic tool. The goal is to determine if the PCAP-files contains the packets transferring any of the reference files, including the installer. The used PCAP-file is 15 Megabytes large.

Matching the PCAP-files against the reference set yields four results, where the type of the matched files and their similarity score is shown in Table 3 below.

As Table 3 shows, there was a significant match on the tool installer, indicating that the tool in fact was downloaded during the time frame the PCAP-file was generated. The ability to prove that the installer is in fact present within the capture network traffic may be sufficient for the investigation objective. One way to increase the reliability of the findings is to illustrate where in the network traffic the matches reside. Fig. 7 below is an *sddiff* visualization of the similarity between the PCAP-file and the installer file.

The narrow blue bars to the left and right of the wide bar in Fig. 7 are false positive similarities, while the wide blue bar indicates the position of the packets transferring the cryptographic tool in the PCAP-file. There are two major benefits of analyzing network traffic in this manner over using conventional cryptographic hashes like MD5 and SHA1. First, we do not have to preprocess the data to match against several simultaneous sessions. Second, we can do

Fig. 7. Position of similarity between PCAP-file and installer file in reference set indicating that the file was transferred in the middle of the traffic recording session.

partial file transfers, i.e., where files are not completely transferred, or not transferred in a single session.

Clustering – organizing an unknown input set

This final section of the practical scenarios of AHBM usage will describe the clustering mode. A cluster is in this context a group of at least two binary files where none of the members have any similarity score to a member of another cluster. As part of the research leading up to this paper, a tool to easily generate these clusters was implemented. The tool, called *makecluster*, takes a pipe-separated file where each line is on the format *fileA|fileB|similarity score* and generates one GraphML[10] file for each disjunct cluster of similar files. The input format is compatible with the standard *sdhash* output format. Splitting the results from AHBM clustering like this makes it easier to process large datasets, as each graph file only contains nodes that are directly or indirectly connected. As it is easy to label each cluster with, e.g., node count and edge count, it becomes easier for the analyst to sort and navigate the generated clusters. Using this technique may save substantial amounts of computational resources when some kind of measurements on the graph as the number of nodes are reduced. It is also easier for humans to understand small, rather than large graphs.

The data source for this experiment is a Contagio[11] malware set containing approximately 10,000 malicious PDF documents. The dataset is publicly available, but password protected.

All ten thousand documents were matched against each other using *sdhash* and an edge was added between two documents if they had a similarity score similar to, or above 10^{12}. The matching yielded 296 clusters. The largest cluster contained almost 800 documents and 300,000 edges. Most of the clusters were dense, meaning that most of the documents has a significant similarity score with each other, as shown in Fig. 8.

Fig. 9 represents a cluster with more internal differences than Fig. 8. The clusters are visualized using Gephi.[13] Analyzing these clusters, it may be possible to determine the source of a document, e.g., the adversary behind some malware. However, a deeper analysis is needed in order to conclude on these matters.

Clustering – a closer look

Considering the cluster in Fig. 9, a closer look was taken at the three documents *A*, *B* and *C*. The three documents were chosen because of their topology and that they represent outliers within the cluster. Some of their metadata are listed in Table 4 and similarity scores are listed in Table 5.

[10] http://graphml.graphdrawing.org.
[11] http://contagiodump.blogspot.com.
[12] Similarity threshold 10 was chosen to allow for weaker connections within the clusters than we allowed in the search experiment.
[13] http://gephi.org.

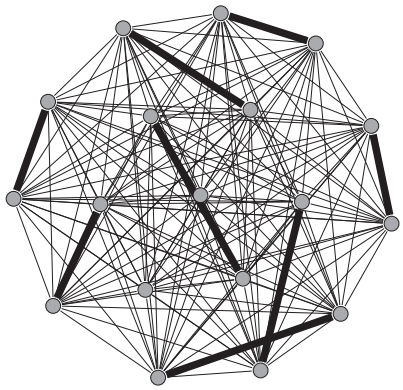

Fig. 8. Cluster of similar malicious PDF documents. The thicker the edge, the higher the similarity score among the two documents.

Running *sddiff* on the documents gives the similarity diagrams in Figs. 10–12.

The pictures above suggest that all documents share the same header and footer. In addition, document *A* shares two regions with *B* and one region with *C*. A region may consist of one or more PDF objects. *B* and *C* do not share any objects other than the header and footer. To further analyze the similarity, we need to extract the relevant regions. PDFStreamDumper (Zimmer, 2010) is used to decompress and extract data stream objects from the documents. When manually comparing objects from the different documents with each other, a pattern emerges:

- The documents contain identical JavaScript block headers.
- The documents contain JavaScript code of same format, but using different variable and function obfuscations.
- The documents contain identical, relatively large, blobs of repetitive data.

The JavaScript blocks found in these documents are too small for *sdhash* to compare individually. However, based

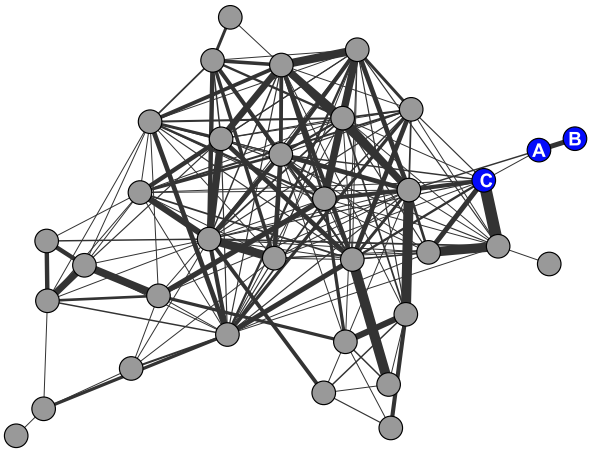

Fig. 9. Cluster of similar malicious PDF documents. The thicker the edge, the higher the similarity score among the two documents. The three nodes *A*, *B* and *C* are selected for further analysis.

Table 4
Excerpt of metadata for PDF documents in *A*, *B* and *C*.

ID	sha1sum [0:30]	Size
A	c4d4fc7d8a69f74856d64809f596e3…	7522 bytes
B	0933c9020001624d192a0adcb2615…	7530 bytes
C	65e37c684c47526f4109b6f7ae3982…	7517 bytes

Table 5
Similarity scores among documents *A*, *B* and *C*.

Document 1	Document 2	Similarity score
A	*B*	027
A	*C*	010
B	*C*	001

Fig. 10. *sddiff* on *A* and *B*. Blue bars represent position and size of similar data streams. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

on the overall similarity and additional analysis of common structure and PDF stream objects, it is assumed that all three documents have the same origin.

AHBM tools may, as the results indicate, be able to detect malware by comparing unknown documents to known malware. Malware has numerous ways to hide their nature, e.g., through mutation and packers. A deeper study on how AHBM can be used to detect and classify malware is a subject for further research. However, the possibility of automatically organizing documents into clusters ought to greatly improve analysis when faced with an unknown dataset.

Further work

Of the more appealing subjects for further research is a thorough study of how AHBM can be applied when analyzing malware. The impact various techniques for masquerading malware have on Approximate Matching technique is not reviewed. Such analysis should both consider malicious documents, such as PDFs and Office macros, as well as malicious executables. An interesting research question is whether or not it is possible to determine the origin of an unknown malware based on its binary representation.

Fig. 11. *sddiff* on *A* and *C*.

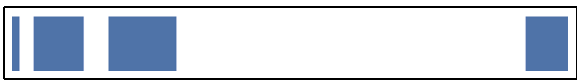

Fig. 12. *sddiff* on *B* and *C*.

Another subject for further research is to extends the work done by Roussev and Quates (2012), assessing the possibilities and challenges when applying AHBM on multi-terabyte, potentially petabyte scale datasets. Further, if Approximate Matching is to be presented as evidence in a court of law, there is a need to discuss and evaluate the necessary confidence of these tools with regards to false positives and false negatives in a legal context. There has also been limited focus on how forensic soundness can be ensured during digital investigations using Approximate Matching.

Finally, as Approximate Matching still is a computationally expensive operation, and investigators usually are in a hurry, considerable attention should be given to how the matching may be distributed across multiple computers. Current approaches involves using graphical processing units (GPU), however, existing technologies within distributed computing should enable lower latency on matching and thereby also data exploration of large datasets.

Conclusions

The primary findings in this study have been the definition of three modes in AHBM: *searching*, *streaming* and *clustering*. The experiments have shown that the Approximate Matching technique is well suited for the task to discover additional information based on syntactical similarity. Tests on the Enron dataset revealed that it is possible to discover alternative conversations by matching an email with the dataset. Experiments using *sdhash* as a streaming function showed that the technique is also a good fit for monitoring continuous streams of data, such as network traffic, for traces of interesting data. Finally, experiments with clustering a dataset consisting of malicious PDF documents showed that Approximate Matching techniques has a potential for organizing and classifying unknown data. A deeper analysis revealed that *sdhash* does not detect similarity among obfuscated JavaScript files, but it may still detect similarities among malicious documents based on other parts of the documents.

Regarding the use of AHBM to search for emails in the Enron dataset, it is in order to question whether the benefits justify the additional computational complexity compared to using, e.g., string search. Simple string search may yield the same results as Approximate Matching, but it also puts a larger cognitive burden on the analyst, who without Approximate Matching has to determine what parts of an email to search for. This reduces effectiveness and also increases the risk of human error during the investigation.

Finally, the implemented tool *sddiff* may be helpful to better evaluate the significance of similarity between two files, however it does ignore potentially important metadata such as timestamps, filenames and sizes. It should therefore be used in addition to existing file analysis techniques.

References

Bloom BH. Space/time trade-offs in hash coding with allowable errors. Commun ACM 1970;13(7):422–6.

Breitinger F, Stivaktakis G, Baier H. Frash: a framework to test algorithms of similarity hashing. Digit Investig 2013;10:S50–8.
Buchmann J. Introduction to cryptography. Springer; 2004.
Carrier B. Sleuthkit autopsy home page http://www.sleuthkit.org/autopsy/; 2012 [Accessed 20.02.14].
DigitalCorpora.org. 57 patents scenario disk images http://digitalcorpora.org/corpora/scenarios/m57-patents-scenario; 2009 [Accessed: 20.02.14].
Flaglien A, Franke K, Årnes A. Identifying malware using cross-evidence correlation. In: Advances in Digital Forensics VII. Springer; 2011. pp. 169–82.
Garfinkel S. Network forensics: tapping the internet; 2002.
Garfinkel SL. Digital media triage with bulk data analysis and *bulk_extractor*. Comput Secur 2013;32:56–72.
Hamming RW. Error detecting and error correcting codes. Bell Syst Tech J 1950;29(2):147–60.
Klimt B, Yang Y. Introducing the Enron Corpus. In: CEAS; 2004.
Kornblum J. Identifying almost identical files using context triggered piecewise hashing. Digit Investig 2006;3:91–7.
Levenshtein VI. Binary codes capable of correcting deletions, insertions and reversals. In: Soviet Physics Doklady, vol. 10; 1966. p. 707.
Meixner A, Uhl A. Robustness and security of a wavelet-based CBIR hashing algorithm. In: Proceedings of the 8th workshop on Multimedia and security. ACM; 2006. pp. 140–5.
Microsoft. PhotoDNA fact sheet http://www.microsoft.com/en-us/news/presskits/photodna/docs/PhotoDNAFS.doc; 2009 [Retrieved 20.02.14].
Microsoft. 500 million friends against child exploitation http://blogs.technet.com/b/microsoft_blog/archive/2011/05/19/500-million-friends-against-child-exploitation.aspx; 2011 [Accessed 20.02.14].
PacketLoop. Packetpig project page https://github.com/packetloop/packetpig; 2013 [Accessed 20.02.14].
Roussev V. Data fingerprinting with similarity digests. In: Advances in Digital Forensics VI. Springer; 2010. pp. 207–26.
Roussev V. An evaluation of forensic similarity hashes. Digit Investig 2011;8:S34–41.
Roussev V, Quates C. Content triage with similarity digests: the m57 case study. Digit Investig 2012;9:S60–8.
Zauner C. Implementation and benchmarking of perceptual image hash functions [Master's thesis]. Hagenberg Campus: Upper Austria University of Applied Sciences; 2010. p. 43.
Zimmer D. PDFStreamDumper project page http://sandsprite.com/blogs/index.php?uid=7&pid=57; 2010 [Accessed 23.11.14].