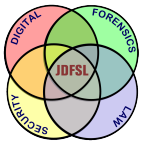

THE JOURNAL OF
DIGITAL FORENSICS, SECURITY AND LAW

Journal of Digital Forensics, Security and Law

Volume 11 | Number 2 Article 4

2016

Bytewise Approximate Matching: The Good, The Bad, and The Unknown

Vikram S. Harichandran
University of New Haven

Frank Breitinger
University of New Haven

Ibrahim Baggili
University of New Haven

BYTEWISE APPROXIMATE MATCHING: THE GOOD, THE BAD, AND THE UNKNOWN

Vikram S. Harichandran, Frank Breitinger and Ibrahim Baggili
Cyber Forensics Research and Education Group (UNHcFREG)
Tagliatela College of Engineering
University of New Haven, West Haven CT, 06516, United States
e-Mail: {vhari2, FBreitinger, IBaggili}@newhaven.edu

ABSTRACT

Hash functions are established and well-known in digital forensics, where they are commonly used for proving integrity and file identification (i.e., hash all files on a seized device and compare the fingerprints against a reference database). However, with respect to the latter operation, an active adversary can easily overcome this approach because traditional hashes are designed to be sensitive to altering an input; output will significantly change if a single bit is flipped. Therefore, researchers developed approximate matching, which is a rather new, less prominent area but was conceived as a more robust counterpart to traditional hashing. Since the conception of approximate matching, the community has constructed numerous algorithms, extensions, and additional applications for this technology, and are still working on novel concepts to improve the status quo. In this survey article, we conduct a high-level review of the existing literature from a non-technical perspective and summarize the existing body of knowledge in approximate matching, with special focus on bytewise algorithms. Our contribution allows researchers and practitioners to receive an overview of the state of the art of approximate matching so that they may understand the capabilities and challenges of the field. Simply, we present the terminology, use cases, classification, requirements, testing methods, algorithms, applications, and a list of primary and secondary literature.

Keywords: Approximate matching, Fuzzy hashing, Similarity hashing, Bytewise, Survey, Review, ssdeep, sdhash, mrsh-v2.

1. INTRODUCTION

It is no secret that the number of networked devices in the world continues to increase alongside the complexity of cyber crimes, size of storage devices, and amount of data. We are beyond the point where investigators can manually analyze all cases. These advances have also been complemented with an increase in processing power. Speaking in numbers, while 80-200 GB HDDs, 2-4 GB of RAM memory, and dual core were the quasi standards for machines in 2011, nowadays they are 512 GB SSDs, 8-16 GB RAM, and multicore architectures; (external) storage devices may have several terabytes of storage. Furthermore, if privately owned computing resources are insufficient for a task,

one may shift it to the cloud. In short, practitioners need tools and techniques that are capable of automatically handling large amounts of data since time in investigations is of the essence.

A common forensic process to support practitioners is *known file filtering*, which aims at reducing the amount of data an investigator has to manually examine. The process is quite simple: (1) compute the hashes for all files on a target device (2) compare the hashes to a reference database. Based on the signatures in the database, files are whitelisted (filtered out / known-good files, e.g., files of the operating system) or blacklisted (filtered in / known-bad files, e.g., known illicit content). This straightforward procedure is commonly implemented using cryptographic hash functions like MD5 (Rivest, 1992) or an algorithm from the SHA family (FIPS, 1995; Bertoni, Daemen, Peeters, & Assche, 2008).

While cryptographic hashes are well-established and tested, they have one downside – they can only identify bitwise identical objects. This means changing a single bit of the input will result in a totally different hash value. Subsequently, the community worked on a counterpart for (cryptographic) hashing algorithms that allows similarity identification – *approximate matching*. Although this is a practically useful concept, a recent survey by Harichandran, Breitinger, Baggili, and Marrington (2016) with 99 participants showed that only 12% of the forensic experts polled use this technology on regular basis. Detailed results are provided in Table 1.

Table 1. Answers to the survey question: Have you ever used approximate matching/similarity hashing algorithms?

Answer	**in %**
Yes, I use them on a regular basis.	12.50
Yes, a few times.	34.38
No, they are too slow for practical use.	7.29
No, they are unnecessary for my purposes.	31.25
No, I am unaware of what it is.	14.58

Contribution. In this paper we aim to address the almost 15 % that have never heard of approximate matching by providing them with a comprehensive literature survey, and the 31 % (unnecessary for my purposes) by illustrating a multitude of applications for approximate matching. Accordingly, we address the following key points:

- Terminology, use cases, classification, requirements, and testing.
- High-level description of existing algorithms including strengths and weaknesses.
- Secondary literature that enhances / assesses existing approaches.
- New applications that employ approximate matching, e.g., file carving and data leakage prevention.
- Current limitations and challenges, and possible future trends.

Since it is low-level (is directly concerned with the structure of everything digital), and may be the most impacting / implemented type due to its usage for automation, we focus on *bytewise* approximate matching.

Differentiation from previous work. When writing this article, there were three articles similar to this survey. The first was the SP 800-168 from the National Institute for Standards and Technology (NIST, Breitinger, Guttman, McCarrin, Roussev, and White (2014)). While this article provides an overview of the terminology, uses

cases, and testing, it does not include any algorithm concepts, applications, or critical discussion. Moreover, a reader is not provided with a long list of references. Martínez, Álvarez, and Encinas (2014) is a purely technical paper and focused on the full details of the algorithms and their implementations. Thirdly, the dissertation from Breitinger (2014) contained almost all of these topics but is extremely lengthy. Ergo, our intention during writing was to make this publication the primary source for researchers / practitioners to grasp a cursory bird's-eye view of bytewise approximate matching.

We summarized the most important elements of these works in a condensed and direct manner to increase the awareness of approximate matching. Extra texts are also shared for each algorithm in Sec. 4.

Structure. The remainder of this paper is organized as follows: Sec. 2 provides the historical background of approximate matching. Concepts are outlined, including use cases, types, requirements (this subsection describes the core principles of algorithm design), and testing. Then, after traversing 8 of the most popular algorithms in Sec. 4 we mention newly explored prospects in Sec. 5. Limitations and challenges precedes a brief listing of future areas of research in Sec. 7.

2. HISTORY

Many of the approximate matching algorithms designed to solve modern-day problems in digital forensics rely fundamentally on the ability to represent objects as sets of features, thereby reducing the similarity problem to the well-defined domain of set operations (Leskovec, Rajaraman, & Ullman, 2014). This approach has roots in the work of the early Swiss 20^{th} century biologist Paul Jaccard, who suggested expressing the similarity between two finite sets as the ratio of the size of their intersection over the size of their union (Jaccard, 1901, 1912): if A and B are sets, then the Jaccard index J is defined as $J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$. It has been widely adopted as a method for quantifying similarity and is still used mainly within computer linguistics for plagiarism detection.

Nearly a century later, Broder (1997) proposed using the Jaccard index as part of his algorithm for identifying similar documents. Broder suggested a distinction between two commonly used types of similarity: 'roughly the same' (resemblance) and 'roughly contained inside' (containment). While he recommended using the Jaccard index for resemblance, he introduced a variation to approximate containment which "indicates that A is roughly contained within B": $c(A,B) = \frac{|A \bigcap B|}{|A|}$. Additionally, Broder described the `MinHash` algorithm, an efficient method for estimating these similarities (Broder, Charikar, Frieze, & Mitzenmacher, 1998).

On the other hand, Manber (1994) presented `sif`, an implementation used to correlate text files. "Files are considered similar if they have a significant number of common pieces, even if they are very different otherwise." Due to the complexity of comparing strings directly, he utilized Rabin fingerprinting to hash and compare substrings (Rabin, 1981).

A first step towards approximate matching as we use it today was `dcfldd`[1] by Harbour in 2002, which was an extension for the well-known disk dump tool `dd`. His tool divided the input into chunks of fixed length and hashed each chunk. Therefore, Harbour's approach is also called *block-based hashing*. While this approach works perfectly for flipped bits, it has a limited capacity to detect similarity in strings where

[1] `http://dcfldd.sourceforge.net` (last accessed Feb 4^{th}, 2016).

the deletion or insertion of bits creates a shift that changes the hashes of all blocks that follow. Theoretically, the shift of even a single bit at the beginning of a file could cause nearly identical objects to appear to have nothing in common, much like the naive (traditional) file hashing approach.

Although this weakness can present a problem for file-to-file comparison, it may be acceptable in some scenarios. For example, if the goal is to determine which parts of a disk image might have been changed during a cyber attack, Harbour's technique remains useful. Likewise, in the case of an analyst scanning for blacklisted material across a drive or a collection of drives, the loss of a few block matches may be a worthwhile trade-off for gains in speed and simplicity, particularly because a single block is often sufficient evidence to demonstrate the presence of an artifact, or at least to warrant closer inspection.

Several efforts have been made to further leverage this technique for detecting similar material by matching fixed length file fragments. Collange, Dandass, Daumas, and Defour (2009) coined the term "hash-based carving" to describe this method of scanning for blacklisted material, since it can be used to extract content without aid from the file system, provided the targets are known beforehand.

Key (2013)'s File Block Hash Map Analysis (FBHMA) EnScript and Simson Garfinkel's tool frag_find (S. L. Garfinkel, 2009) provided practical implementations that automated the process for forensic examiners, though searches were limited to a few files at a time. S. Garfinkel, Nelson, White, and Roussev (2010) described the implementation and evaluation of *frag_find* in detail, noting a particular difficulty in storing and searching billions of hashes at practical speeds. S. Garfinkel and McCarrin (2014) later succeeded in scanning a drive image for matches of 4096-byte blocks across a set of nearly one million blacklisted files stored in the custom-built database, *hashdb*.

3. CONCEPTS

While approximate matching (a.k.a. fuzzy hashing or similarity hashing) started to gain popularity in the field of digital forensics in 2006, it was not until 2014 that the National Institute for Standards and Technologies (NIST) developed standard definitions, publishing *Approximate Matching: Definition and Terminology* (NIST SP 800-168, Breitinger, Guttman, et al. (2014)). Subsections below briefly summarize this work's principles.

The 'Purpose and Scope' section of the NIST document defines approximate matching as follows: "Approximate matching is a promising technology designed to identify similarities between two digital artifacts. It is used to find objects that resemble each other or to find objects that are contained in another object."

3.1 Use cases

In investigative cases, approximate matching is used to filter known-good or known-bad files while using a reference approximate matching hashed data set, either on static data or data in transit over a network. The primary use cases for approximate matching are presented below:

- Similarity detection correlates related documents, e.g., different versions of a Word document.
- Cross correlation correlates documents that share a common object, e.g., a DOC and a PPT document including the same image.

- Embedded object detection identifies an object inside a document, e.g., an image inside a memory dump.
- Fragment detection identifies the presence of traces/fragments of a known artifact, e.g., identify the presence of a file in a network stream based on individual packets.

A lecture at DFRWS USA 2015 decided to break down uses into 6 categories instead, from the perspective of the bytestreams matching: R is identical to T, R contains T, R & T share identical substrings, R is similar to T, R approximately contains T, and R & T share similar substrings (where R and T are two sequences) (Ren & Cheng, 2015).

Notwithstanding, approximate matching may not be appropriate when used to whitelist artifacts since such content can be quite similar to benign content, e.g., an SSH server with a backdoor would look analogous to a regular entry point (Baier, 2015).

3.2 Types

Regardless of the use cases, approximate matching can be implemented at different abstractions. Usually we distinguish between the following three abstraction categories:

Bytewise: matching operates on the byte level and uses only the byte sequences as input only (also known as *fuzzy hashing* and *similarity hashing*).

Syntactic: matching also works on the byte level but may use internal structure information, e.g., one may ignore the TCP header information of a packet that is parsed.

Semantic: matching works on the content-visual layer and therefore closely resembles human behavior (also called *perceptual hashing* and *robust hashing*), e.g., the similarity of the content of a JPG and a PNG image where the image file types / byte streams are different, but the picture is the same.

Furthermore, there are 4 cardinal categories of algorithms (see Sec. 4 for the inner workings) (Martínez et al., 2014):

- Context-Triggered Piecewise Hashing (CTPH).
- Block-Based hashing (BBH).
- Statistically-Improbable Features (SIF).
- Block-Based Rebuilding (BBR).

3.3 Requirements

There are multiple ways to interpret substring matching. For example, “ababa” and “cdcdc” might be considered similar because they both have five characters ranging over two alternating values, or they might be treated as dissimilar because they have no common characters. Thus, algorithms must define the lowest common denominator of its interpretation - a *feature* - including how they are derived from an input. When two features are compared the outcome is binary, match or no match.

A *feature set* refers to a set of distinct features found in the entire bytestream of a file or file fragment. An algorithm may choose to only include some features in this set and must outline the method/criteria for inclusion. Feature sets are then used to produce a match/similarity score, representing the amount of similarity between sets of target files rationally (an increasing monotonic function).

Bytewise algorithms have two main functions. A *feature extraction function* identifies and extracts features from objects to convert them to a compressed version for

comparison. Then, a *similarity function* performs the comparison between these compressed versions to output a normalized match score. This comparison usually involves string formulas such as *Hamming distance* and *Levenshtein distance*; Martínez, Álvarez, Encinas, and Ávila (2015) and Li et al. (2015) have proposed new algorithms for specific uses. Normalized scores may be calculated by weighing the number of matching features against the total number of features for both objects (for resemblance), or by ignoring unmatched features in the container object (if concerned with containment).

In addition to the above, these traits must be satisfied to be considered a valid approximate matching algorithm, according to NIST:

Compression: actual storage of features is usually implemented as a one-way hash known as a *similarity digest*, *signature*, or *fingerprint*); length is shorter than the original feature/input itself.

Similarity preservation: similar inputs should result in similar digests.

Self-evaluation: authors should state the confidence level for the circumstances/-parameters used to produce the match score and what the scale is (e.g., 0 = no features matched, 1 = all exact match).

Time complexity/runtime efficiency: speed should be stated via theoretical complexity in O-notation as well as the runtime speed; for bytewise algorithms it is preferable to know the isolated speeds of the feature extraction and similarity functions.

3.4 Testing bytewise approximate matching

Testing algorithms is an important task, so researchers set out to create a test environment for bytewise approximate matching. The first step was taken by Breitinger, Stivaktakis, and Baier (2013), called FRamework to test Algorithms of Similarity Hashing (FRASH).

It tested *efficiency*, *sensitivity and robustness*, and *precision and recall*. This last category can be divided further into synthetic data vs. real world data. While synthetic data provides the perfect ground truth (further described below), it does not coincide with the real world, and vise versa.

Synthetic data test results were published (Breitinger, Stivaktakis, & Roussev, 2013) in addition to real world data (Breitinger & Roussev, 2014). The complete results are too complex to be presented in this article but can be found in chapter 6 in Breitinger (2014). In the following subsections we briefly summarize how approximate matching algorithms can be evaluated and FRASH's results. The main findings were:

- `sdhash` and `mrsh-v2` outperform other algorithms.
- `mrsh-v2` is faster and shows better compression than `sdhash`.
- `sdhash` obtains slightly better precision and recall rates than `mrsh-v2`.

Therefore, the final decision for selecting an algorithm depends on the use case.

Efficiency. As with cryptographic hash functions, compression and runtime efficiency are important, but approximate matching algorithms involve additional concerns; several do not output fixed length digests. Thus, researchers usually report compression ratio, $cr = \frac{\text{digest length}}{\text{input length}}$.

The community distinguishes between the following for runtime:

Generation efficiency: time needed to process an input and output the similarity digest.

Comparison efficiency: summarizes the theoretical complexity (in O-notation) to compare digests against an existing data set / database; again, often stated in units of time for implementations for bytewise approximate matching.

Space efficiency: calculated by dividing digest length by input length.

Sensitivity and robustness. Sensitivity refers to the granularity at which an algorithm can detect similarity, i.e., how minute the feature is. At some threshold making a feature too fine causes almost all objects to appear common, however, and therefore the algorithm designer must strike a balanced sensitivity to optimize utility and time efficiency.

Robustness is a metric of how effective an algorithm can be in the midst of noise and plain transformations such as fragmentation and insertion/deletion into the target byte sequence.

As outlined by FRASH, these attributes are tested by creating manual mutations of the target fragments/files:

Alignment robustness: inserts blocks of various sizes at the beginning of an input; this should simulate scenarios like growing log files or emails.

Fragment detection: identifies the smallest fragment of a byte sequence that still matches by cutting it; this feature is important for network traffic analysis (see Sec. 5.2) and hash-based file carving (see Sec. 5.1).

Single-common block correlation: analyzes the minimum amount of correlation between two files, e.g., two word documents that share an common paragraph.

White noise resistance: is a probability-driven test that introduces (uniform) random changes into a byte sequence (via insertion, deletion, & substitution); a viable scenario is source code where a developer renamed a variable.

Precision and recall on synthetic data. Precision can be thought of as a measure of false positives (possibility of counting objects as similar that in actuality are not) while recall refers to the false negatives (omitting objects that should be tallied as similar). These attributes are an indication of an algorithm's reliability.

In order to quantify them, the initial step is to analyze synthetic data. First, random byte sequences (the FRASH paper used Linux `/dev/urandom`) are generated. Next, mutations are created through methods like those mentioned in the previous subsection. Finally, the comparison is executed and the results are analyzed.

Precision and recall on real world data. Testing on real world data is a bit more complex because there is no definition for similarity and no ground truth (publicly available data sets for testing that involves explanations of what the expected similarity is between different files). To define the ground truth, the community developed *approximate longest common substring* (aLCS) which estimates the longest common substrings of two files. According to this, two inputs are declared as similar, if their aLCS is sufficient (e.g., 1 % of the total input length, or at least 2 KiB).

3.5 Security

Most approximate matching algorithms currently implement few-to-zero security features to guard against active, real-time attacks. One staple is inherently built into approximate matching algorithms: hash functions are one-way, even for similarity, and

therefore prevent reverse engineering the original input sequence of a fragment / file. The few other tolerances exhibited by the algorithms are stated in their individual subsections under Sec. 4.

However, we posit that for most uses of approximate matching, security features are not essential. As pointed out by Baier (2015) these algorithms are most likely to be used for blacklisting. Why would an active adversary want to create files that match a blacklist of static (not in transit) data? Researchers must find an answer to how easy it is to avoid matching files. Maybe in the future we should classify security for approximate matching algorithms by the minimum amount of changes that are necessary between two files in order to produce a non-match. A question that needs fresh exploration, though, is what practices criminals can use to bypass certain (types of) algorithms, use cases, and applications; a rigorous analysis of this has not been performed partially due to missing standards / ground truth.

3.6 Extending existing concepts

One of the major challenges that comes with approximate matching is related to the nearest neighbor problem, i.e., how to identify the similarity digests that are similar to a given one. More precisely, let's assume a database containing n entries. Most algorithms require an 'against-all' comparison which equals a complexity of $O(n)$.

Winter, Schneider, and Yannikos (2013) presented an approach to diminish this complexity for `ssdeep` named F2S2. Generally speaking, instead of storing the complete Base64 encoded similarity digest in the database, they stored n-grams using hashtables. In order to lookup single digests they first looked for the n-grams which reduced the overall amount of comparisons. For the final decision, the `ssdeep` comparison function was used. As a result, they reduced the comparison time of 195,186 files against a database containing 8,334,077 records from 442 h to 13 min (boosted by a factor of about 2000), a 'practical speed'.

However, this approach works only for Base64 and hence for none of the other approaches like `sdhash` or `mrsh-v2`. Therefore, Breitinger, Baier, and White (2014) presented a concept that could speed up the process via Bloom filter-based approaches. They suggested using one single huge Bloom filter to store all feature hashes, which results in a complexity of $\sim O(1)$. Their approach overcomes the drawback of comparing digests against digests but loses precision. That is, it allows for only yes or no decisions: yes means there is a similar file in the set; no equates to none of the files being similar above the chosen threshold. It does not allow for the returning of the matched file(s).

Consequently, the authors presented an enhancement which simply uses multiple large Bloom filters to generate a tree structure that results in a complexity of $O(\log(n))$ (Breitinger, Rathgeb, & Baier, 2014). But these are only assumptions – while there is a working prototype for the first approach, the latter concept only exists in theory.

3.7 Distinction from locality-sensitive hashing (LSH)

It's critical to note that sometimes people confuse *Locality-Sensitive Hashing* (LSH) (e.g., Rajaraman and Ullman (2012)) with approximate matching. Therefore, we included this section. LSH is a general mechanism for nearest neighbor search and data clustering where the performance strongly relies on the used hashing method. Two pop-

ular algorithms are `MinHash` (Broder, 1997) and `SimHash`[2] (Charikar, 2002).

This does not necessarily coincide with the idea of approximate matching. Specifically, while LSH aims at mapping similar objects into the same bucket, approximate matching outputs a similarity digest that is comparable.

We would like to note here that the following section mainly focuses on bytewise approximate matching.

4. INTRODUCTION TO ALGORITHMS

As previously mentioned, approximate matching started to gain attention in 2006 with the concept of context triggered piecewise hashing and its first implementation, `ssdeep` (Kornblum, 2006). In the following years, new algorithms were proposed and published.

We will introduce the eight known approximate matching algorithms. While the first three algorithms are still extended and relevant, the last four algorithms are less promising from a digital forensics perspective for various reasons, e.g., precision and recall rates, runtime efficiency and detection capabilities. The last algorithm (TLSH) is more related to LSH than approximate matching and is included for completeness.

This section is a high-level summary of the current algorithms. Throughout each subsection references are cited for deeper reading.

4.1 ssdeep

CTPH is the technique used by `ssdeep` and was presented by Kornblum (2006). Roughly speaking, it is a modified version of the spam detection algorithm from Tridgell (2002–2009) generalized to cope with any digital object.

In CTPH the approach is to identify trigger points to divide a given input into chunks/blocks. This breakup is performed using a rolling hash that slides through the input, adds bytes to the current context (think of it as a buffer), creates a pseudo-random value, and removes them from the context after a set number of bytes are completed. The context is then used as a trigger – whenever a specified sequence is created the current context is hashed by the non-cryptographic FNV-hash function (Fowler, Noll, & Vo, 1994–2012). To create the similarity digest, the FNV-chunk-hashes are reduced to 6 bits, converted into a Base64 character and concatenated; this is done continuously as the trigger outputs FNV hashes.

At the time of this article, `ssdeep` was still an active project with version 2.13 and is freely available online[3]. Over the years, several extensions and performance improvements have been published that mostly focus on the efficiency of the implementation (Chen & Wang, 2008; Seo, Lim, Choi, Chang, & Lee, 2009; Breitinger & Baier, 2012b). However, a security analysis conducted by Baier and Breitinger (2011) showed that CTPH cannot withstand an active attack.

4.2 sdhash

Similarity digest hashing was published four years later by Roussev, Richard, and Marziale (2008); Roussev (2010) and is also still active. The SIF algorithm extracts statistically improbable features that are determined by Shannon entropy (not the ones with the highest / lowest entropy but the

[2]Note, SimHash is a common term and is used several times literature. Accordingly, it is also used twice in this article. Besides this section it is also used in Sec. 4.6 where it describes an approach from Sadowski and Levin (2007).

[3]`http://ssdeep.sourceforge.net` (last accessed Feb 4th, 2016).

ones that seem unique, Roussev (2009)). In `sdhash`, a feature is a byte sequence of 64 bytes that is then compressed by hashing it with SHA-1. Finally, the author developed a way to insert the hashes into a Bloom filter[4] (Bloom, 1970).

The original version was extended several times, now supporting GPU usage for calculation and a block-based hashing mode (Roussev, 2012). The current version (3.4) is available online[5].

A comparison between `ssdeep` and `sdhash` showed that the latter algorithm outperforms its predecessor (Roussev, 2011). In addition, a security analysis showed that `sdhash` is much more robust and difficult to overcome (Breitinger & Baier, 2012c).

4.3 mrsh-v2

This algorithm was published by Breitinger and Baier (2013) and is a combination of `ssdeep` and `sdhash`[6]. Like the aforementioned implementations, `mrsh-v2` is still supported[7]. The algorithm uses the feature identification procedure from `ssdeep`, then hashes the feature using the non-cryptographic FNV (Fowler et al., 1994–2012) and proceeds like `sdhash`, consequently overcoming the weaknesses of `ssdeep` and becoming faster than `sdhash`. The precision and recall rates are slightly worse than `sdhash`.

4.4 bbHash

Building block hashing is a completely different approach and is based on the concept of eigenfaces (biometrics) and de-duplication (data compression). Contrary to expectation, its type (see Sec. 3.2) is not eponymous, but rather BBR. The main difference is that this approach utilizes an external reference point – the building blocks.

A set of 16 building blocks (random byte sequences) is used to optimize representation of a given file. In order to find this representation the algorithm calculates the *Hamming distance*, which is time consuming and slow for practical usage (e.g., it takes about two minutes to process a 10 MB file) (Breitinger & Baier, 2012a).

4.5 mvHash-B

Majority vote hashing, another BBR type, was published by Åstebøl (2012); Breitinger, Åstebøl, Baier, and Busch (2013). It transforms any byte-sequence into long runs of `0x00`s and `0xFF`s by considering the neighboring bytes of a specific byte. If the neighborhood consists of mainly 1s, the byte is set to `0xFF`, otherwise to `0x00`. Next, these runs are encoded by Run Length Encoding (RLE). Although this proceeding is very fast, it requires a specific configuration for each file type.

4.6 SimHash

`SimHash` was presented by Sadowski and Levin (2007) and embodies the notion of counting the occurrences of certain predefined binary strings called "Tags" within an input. In their BBR implementation, the authors used 16 8-bit Tags, i.e., a possible Tag could have been 00110101. Subsequently, the tool parses an input bit by bit, searching for each Tag. The total number of matches is stored in a *sum table*. A hash key is computed as a function of the sum table entries that form linear combinations. Lastly, all information (including file name, path, and size) is stored in a database.

To identify similarities, a second tool named `SimFash` is used to query the

[4] A Bloom filter is a space efficient data structure to represent a set. Bloom filters will not be discussed in this article but more details can be found online.

[5] `http://sdhash.org` (last accessed Feb 4th, 2016).

[6] It was also inspired by multi-resolution similarity hashing (Roussev, III, & Marziale, 2007).

[7] `http://www.fbreitinger.de` (last accessed Feb 4th, 2016).

database. The hash keys are used as a first filter to identify all possible matches. Next, the sum tables are compared and a match is found if the distance is within a specified tolerance.

The authors clearly state that "two files are similar if only a small percentage of their raw bit patterns are different. ... [Thus,] the focus of SimHash has been on resemblance detection" (Sadowski & Levin, 2007).

4.7 saHash

Another SIF type, `saHash` uses Levenshtein distance to derive similarity between two byte sequences. The output is a lower bound for the Levenshtein distance between two inputs. Akin to `SimHash` (Sec. 4.6), `saHash` allows for the detection of only near duplicates (up to several hundred Levensthein operations).

A unique characteristic of this approach is its definition of similarity. While all other approaches output a number between 0 and 1 (not a percentage value), `saHash` actually returns the lower bound of Levenshtein operations (Ziroff, 2012; Breitinger, Ziroff, Lange, & Baier, 2014) to convert one file into another.

4.8 TLSH

TLSH belongs to the category of locality-sensitive hashes, published by Oliver, Cheng, and Chen (2013), and is open source[8]. It processes an input byte sequence using a sliding window to populate an array of bucket counts, and determines the quartile points of the bucket counts. A fixed length digest is constructed which consists of two parts: (i) a header based on the quartile points, the length of the input, and a checksum; (ii) a body consisting of a sequence of bit pairs, which depends on each bucket's value in relation to the quartile points. The distance between two digest headers is determined by the difference in file lengths and quartile ratios. Meanwhile, the bodies are contrasted via their approximate Hamming distance. Summing these together produces the TLSH similarity score.

According to the authors, the precision and recall rates are robust across a range of file types. Additional experiments (Oliver, Forman, and Cheng (2014)) showed that TLSH can detect strings which have been manipulated with adversarial intentions[9]. TLSH is also effective in detecting embedded objects depending on the level of object manipulation. Despite these advantages, it is less powerful than `sdhash` and `mrsh-v2` for cross correlation.

5. APPLICATIONS

Originally, approximate matching was designed to support the digital investigation process via the use cases stated in Sec. 3.1; search for target file(s)/fragments or reduce the volume of data needing investigation. Recently, tools such as EnCase, X-Ways Forensics, and Forensic Toolkit (FTK) have incorporated similar object detection technologies (Breitinger, 2014). Researchers have now identified additional working areas where these techniques or tools can have practical impact, e.g., for file carving (see Sec. 5.1), data leakage prevention (see Sec. 5.2 and 5.4) and Iris recognition (see Sec. 5.5).

5.1 Automatic data reduction and hash-based file carving

As sifting through data has become cumbersome, pre-processing schemes have risen.

[8] https://github.com/trendmicro/tlsh (last accessed Feb 4th, 2016).

[9] Tolerance of manipulation was one of the design considerations for TLSH.

Extracting data in bulk is arguably the most sought after application of approximate matching. One perspective that should be fruitful is hash-based file carving.

This alienated area of work was presented by S. Garfinkel and McCarrin (2014). In their paper, the authors combined techniques from file carving and approximate matching to search on "media for complete files and file fragments with sector hashing and hashdb." Instead of focusing on the complete file and comparing it against a database, the authors use individual data blocks. They utilized a special database named `hashdb` (Allen, 2015) to obtain high throughput.

The evaluation proved their strategy works, although they had to solve the problem of non-probative blocks that emerged "from common data structures in office documents and multimedia files." To filter out such artifacts, the authors presented several 'tests' that alleviated the problem.

5.2 Network traffic analysis

Gupta (2013); Breitinger and Baggili (2014) demonstrated preliminary results when using approximate matching on network traffic for data leakage prevention. The question was (since approximate matching can be used for fragment detection) whether network packets could be matched back to their original files.

Design was similar to its traditional counterpart: create a database of known-object signatures (most likely files) and identify these objects, but instead of analyzing a hard drive the researchers used a network stream (single packets). This work illustrated approximate matching's utility in data leakage prevention, a formerly untouched application.

Beginning with modifying the original `mrsh-v2` algorithm to handle the small size of 1460 bytes per packet, the authors showed that this method works robustly on random data (true positive rate 99.6 %, true negative rate 100.0 %) having a throughput of 650 Mbit/s on a regular workstation.

Regardless, they faced several unsolved problems for real world data. One obstacle was that many files share the same structural information (e.g., file header information; this is equivalent to the non-probative blocks problem from the previous subsection) which led to false positive rates of around 10^{-5} – too high for network traffic analysis.

5.3 Malware

Innately, similarity hashing is ideal for grouping things together, but it was not until 2015 that it was rigorously tested when applied to malware clustering (Li et al., 2015). Faruki, Laxmi, Bharmal, Gaur, and Ganmoor (2015) developed AndroSimilar, a syntactical detection algorithm for Android Malware that falls into the SIF category. While Zhou, Zhou, Jiang, and Ning (2012)'s DroidMoSS, a CTPH algorithm, was developed to also detect mobile malware, a comparison between the two could not be performed due to unavailable code.

Polymorphic malware families are on the rise, often hosted on servers that automatically alter inconsequential segments of a file (before sending across a network) to bypass cryptographic detection tactics (Security, 2013). An intriguing paper by Payer et al. (2014) expands on ways criminals can circumvent the use of similarity-based matching for spotting malicious binary code, which often diversifies itself during recompilation. Thus, it is imperative that malware receives more attention.

5.4 Data leakage prevention

With respect to data leakage prevention, approximate matching may also be utilized for

printer data inspection, e.g., MyDLP[10] and Symantec (2010) Data Leakage Prevention. If a document is protected, the software can discard the print (implemented by MyDLP). A similar experiment was also run by our research group[11] which created a virtual secure printer that analyzed a sent document before forwarding it to an actual printer.

Note, according to Comodo Group Inc. (2013), these software solutions often call their technology partial document matching, unstructured data matching, intelligent content matching, or statistical document matching, all synonyms for approximate matching.

5.5 Biometrics

Biometrics is another independent domain employing approximate matching with promising results (Rathgeb, Breitinger, & Busch, 2013; Rathgeb, Breitinger, Busch, & Baier, 2013). In their work, the authors demonstrated the feasibility of using techniques from approximate matching for biometric template protection, data compression and efficient identification. According to Breitinger (2014), there are three improvements:

- "Template protection: the successive mapping of parts of a binary biometric template to Bloom filters represents an irreversible transformation achieving alignment-free protected biometric templates."

- "Biometric data compression: the proposed Bloom filter-based transformation can be parameterized to obtain a desired template size, operating a trade-off between compression and biometric performance."

- "Efficient identification: a compact alignment-free representation of iris-codes enables a computationally efficient biometric identification reducing the overall response time of the system."

6. LIMITATIONS AND CHALLENGES

Bytewise approximate matching has some intrinsic limitations. First and foremost, it cannot pick up similarity at a higher level of abstraction, such as semantically. For instance, it cannot meaningfully match two image files that have the same semantic picture but are different file types / formats due to different binary encoding. However, placing it in tandem with other approaches will still help; Neuner, Mulazzani, Schrittwieser, and Weippl (2015) include it as a critical component of the digital forensic process. Doubly crucial when it comes to applications like malware that employ databases is lookup time. Winter et al. (2013) outlined a faster approach to conduct similarity searching using a database but this method will not work effectively for all approximate matching algorithms. We will not belabor the point since this was discussed in Sec. 3.6.

Evidently, the first challenge to confront is awareness and adoption of approximate matching, a possible indication that more research needs to be conducted to understand the needs for the community. As we outlined in the introduction, 15 % of professionals are unaware of approximate matching – an unacceptable number. Conversely, 7 % criticize that algorithms are too slow for practical use. On the other hand, inquiry needs to be made into why 35 % have used approximate matching only a few times. Our hope

[10] https://www.mydlp.com (last accessed Feb 4th, 2016).

[11] The project was done by Kyle Anthony, a member of the UNH Cyber Forensics Research & Education Group

is that having the NIST SP 800-168 definition, a technical classification of algorithms (Martínez et al., 2014), and this more universal outline will improve awareness and adoption.

Yet another major obstacle is the lack of a standard definition of similarity (Baier, 2015). As addressed by S. Garfinkel and McCarrin (2014), not all kinds of byte level similarity are equally valuable as there are some artifacts (e.g., structural information, headers, footers, etc.) that are less important or lead to false positives. Hence, we need a filtering mechanism to prioritize matches. One possibility could be to extract the main elements (like text or images) and compare those, meaning including a pre-processing step before the comparison.

Aside from initial efforts to test approximate matching algorithms, there are currently no accepted standards and reliable testing frameworks. FRASH is not easy to implement, a deterrent to practitioners. This is one reason this paper avoids giving absolute comparisons between all the (types of) algorithms. More bothersome is the lack of an accepted ground truth for real world data that would support implementation assessments like whether algorithms scale effectively. The ground truth should embody the four use cases (see Sec. 3.1) and algorithm types, even if different data sets are required for each one. Once this is done, the algorithms will be directly comparable (e.g., embedded object detection: A better than B better than C; speed: B faster than A faster than C). We posit that it is critical for practitioner efficiency to know which algorithms solve which potential problems. At the moment the National Software Reference Library (NSRL[12]) has built the most prominent software database and is piecing together a test corpus (its usefulness was demonstrated in Rowe (2012)).

[12] http://www.nsrl.nist.gov (last accessed Feb 4th, 2016).

7. FUTURE FOCUS

Future research should, in accompaniment to prior comments, pursue areas that branch out from digital forensics, even if completely detached. Below we name some possible applications that could use enhancement, and areas that approximate matching may be able to be enhanced by (this is not an exhaustive list):

- Bioinformatics: This field already uses exact matching methods, albeit using bytewise or semantic approximate matching alongside today's methods could conceivably increase efficiency.
- Text mining: Identifying patterns in structured data to gain high-quality, semantic value.
- Templates and layouts: Semantically identify document layout and separate content from template automatically.
- Deep / machine learning: Automated forms of learning need to be able to process information fast, store it efficiently space-wise, and be able to differentiate similarities and differences; semantic hashing is a known aspect of deep learning and ergo might be able to strengthen approximate matching.
- Source code governance: Manage shared code better, especially for open source software.
- Spam filtering and anti-plagiarism: These have already been looked at but might be behooved by deeper scrutiny.

Ultimately, approximate matching is an alienated domain and its increased adoption

will strongly support other domains. Researchers looking to improve on the slowness of indexing and searching may also benefit to look into other domains such as compression and programming. Once more people are aware of approximate matching we might identify more fields where the technology would be relevant.

8. ACKNOWLEDGMENTS

We would like to thank Michael McCarrin for reviewing and editing some paragraphs, especially those that are related to his research. Additionally, we would like to thank Jonathan Olivier for providing us with the summary for TLSH.

REFERENCES

Allen, B. (2015). Hashdb. *https://github.com/simsong/hashdb*.

Åstebøl, K. P. (2012). *mvhash - a new approach for fuzzy hashing* (Unpublished master's thesis). Gjøvik University College.

Baier, H. (2015). Towards automated preprocessing of bulk data in digital forensic investigations using hash functions. *it-Information Technology*, *57*(6), 347–356.

Baier, H., & Breitinger, F. (2011, May). Security Aspects of Piecewise Hashing in Computer Forensics. *IT Security Incident Management & IT Forensics (IMF)*, 21–36. doi: 10.1109/IMF.2011.16

Bertoni, G., Daemen, J., Peeters, M., & Assche, G. V. (2008, October). *Keccak specifications* (Tech. Rep.). STMicroelectronics and NXP Semiconductors.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*, 422–426.

Breitinger, F. (2014). *On the utility of bytewise approximate matching in computer science with a special focus on digital forensics investigations* (Doctoral dissertation, Technical University Darmstadt). Retrieved from http://tuprints.ulb.tu-darmstadt.de/4055/

Breitinger, F., Åstebøl, K. P., Baier, H., & Busch, C. (2013, March). mvhash-b - a new approach for similarity preserving hashing. In *It security incident management and it forensics (imf), 2013 seventh international conference on* (p. 33-44). doi: 10.1109/IMF.2013.18

Breitinger, F., & Baggili, I. (2014, September). File detection on network traffic using approximate matching. *Journal of Digital Forensics, Security and Law (JDFSL)*, *9*(2), 23–36. Retrieved from http://ojs.jdfsl.org/index.php/jdfsl/article/view/261

Breitinger, F., & Baier, H. (2012a, May). A Fuzzy Hashing Approach based on Random Sequences and Hamming Distance. In *7th annual Conference on Digital Forensics, Security and Law (ADFSL)* (pp. 89–100).

Breitinger, F., & Baier, H. (2012b). Performance issues about context-triggered piecewise hashing. In P. Gladyshev & M. Rogers (Eds.), *Digital forensics and cyber crime* (Vol. 88, p. 141-155). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-35515-8_12 doi: 10.1007/978-3-642-35515-8_12

Breitinger, F., & Baier, H. (2012c). Properties of a similarity preserving hash function and their realization in

sdhash. In *Information security for south africa (issa), 2012* (p. 1-8). doi: 10.1109/ISSA.2012.6320445

Breitinger, F., & Baier, H. (2013). Similarity preserving hashing: Eligible properties and a new algorithm mrsh-v2. In M. Rogers & K. Seigfried-Spellar (Eds.), *Digital forensics and cyber crime* (Vol. 114, pp. 167–182). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-39891-9_11 doi: 10.1007/978-3-642-39891-9_11

Breitinger, F., Baier, H., & White, D. (2014, May). On the database lookup problem of approximate matching. *Digital Investigation*, *11, Supplement 1*(0), S1 - S9. Retrieved from http://www.sciencedirect.com/science/article/pii/S1742287614000061 (Proceedings of the First Annual DFRWS Europe) doi: http://dx.doi.org/10.1016/j.diin.2014.03.001

Breitinger, F., Guttman, B., McCarrin, M., Roussev, V., & White, D. (2014, May). *Approximate matching: Definition and terminology* (Special Publication 800-168). National Institute of Standards and Technologies. Retrieved from http://dx.doi.org/10.6028/NIST.SP.800-168

Breitinger, F., Rathgeb, C., & Baier, H. (2014, September). An efficient similarity digests database lookup - A logarithmic divide & conquer approach. *Journal of Digital Forensics, Security and Law (JDFSL)*, *9*(2), 155–166. Retrieved from http://ojs.jdfsl.org/index.php/jdfsl/article/view/276

Breitinger, F., & Roussev, V. (2014, May). Automated evaluation of approximate matching algorithms on real data. *Digital Investigation*, *11, Supplement 1*(0), S10 - S17. Retrieved from http://www.sciencedirect.com/science/article/pii/S1742287614000073 (Proceedings of the First Annual DFRWS Europe) doi: http://dx.doi.org/10.1016/j.diin.2014.03.002

Breitinger, F., Stivaktakis, G., & Baier, H. (2013, August). FRASH: A framework to test algorithms of similarity hashing. In *13th Digital Forensics Research Conference (DFRWS'13)*. Monterey.

Breitinger, F., Stivaktakis, G., & Roussev, V. (2013, Sept). Evaluating detection error trade-offs for bytewise approximate matching algorithms. *5th ICST Conference on Digital Forensics & Cyber Crime (ICDF2C)*.

Breitinger, F., Ziroff, G., Lange, S., & Baier, H. (2014, January). sahash: Similarity hashing based on levenshtein distance. In *Tenth annual ifip wg 11.9 international conference on digital forensics (ifip wg11.9)*.

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Compression and complexity of sequences (sequences'97)* (pp. 21–29). IEEE Computer Society.

Broder, A. Z., Charikar, M., Frieze, A. M., & Mitzenmacher, M. (1998). Min-wise Independent Permutations. *Journal of Computer and System Sciences*, *60*, 327-336.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual acm symposium on theory of computing* (pp. 380–388).

Chen, L., & Wang, G. (2008). An efficient

piecewise hashing method for computer forensics. In *Knowledge discovery and data mining, 2008. wkdd 2008. first international workshop on* (pp. 635–638). doi: 10.1109/WKDD.2008.80

Collange, S., Dandass, Y. S., Daumas, M., & Defour, D. (2009). Using graphics processors for parallelizing hash-based data carving. In *System sciences, 2009. hicss'09. 42nd hawaii international conference on* (pp. 1–10).

Comodo Group Inc. (2013). *PDM (Partial Document Matching).* https://www.mydlp.com/partial-document-matching-how-it-works/.

Faruki, P., Laxmi, V., Bharmal, A., Gaur, M., & Ganmoor, V. (2015). Androsimilar: Robust signature for detecting variants of android malware. *Journal of Information Security and Applications*, *22*, 66–80.

FIPS, P. (1995). 180-1. secure hash standard. *National Institute of Standards and Technology*, *17*, 45.

Fowler, G., Noll, L. C., & Vo, P. (1994–2012). *Fnv hash.* http://www.isthe.com/chongo/tech/comp/fnv/index.html.

Garfinkel, S., & McCarrin, M. (2014). Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb. *digital investigation*, *12*.

Garfinkel, S., Nelson, A., White, D., & Roussev, V. (2010). Using purpose-built functions and block hashes to enable small block and sub-file forensics. *digital investigation*, *7*, S13–S23.

Garfinkel, S. L. (2009, March). *Announcing frag_find: finding file fragments in disk images using sector hashing.* Retrieved from http://tech.groups.yahoo.com/group/linux_forensics/message/3063

Gupta, V. (2013). *File fragment detection on network traffic using similarity hashing* (Unpublished master's thesis). Denmark Technical University.

Harichandran, V. S., Breitinger, F., Baggili, I., & Marrington, A. (2016). A cyber forensics needs analysis survey: Revisiting the domain's needs a decade later. *Computers & Security*, *57*, 1–13.

Jaccard, P. (1901). Distribution de la flore alpine dans le bassin des drouces et dans quelques regions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, *37*, 241–272.

Jaccard, P. (1912, February). The distribution of the flora in the alpine zone. *New Phytologist*, *11*, 37–50.

Key, S. (2013). *File block hash map analysis.* Retrieved from https://www.guidancesoftware.com/appcentral/

Kornblum, J. (2006, September). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, *3*, 91–97. Retrieved from http://dx.doi.org/10.1016/j.diin.2006.06.015 doi: 10.1016/j.diin.2006.06.015

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Mining of massive datasets.* Cambridge University Press.

Li, Y., Sundaramurthy, S. C., Bardas, A. G., Ou, X., Caragea, D., Hu, X., & Jang, J. (2015). Experimental study of fuzzy hashing in malware clustering analysis. In *8th workshop on cyber security experimentation and test (cset 15).*

Manber, U. (1994). Finding similar files in a large file system. In *Proceedings of*

the usenix winter 1994 technical conference on usenix winter 1994 technical conference (pp. 1–10).

Martínez, V. G., Álvarez, F. H., & Encinas, L. H. (2014). State of the art in similarity preserving hashing functions. *worldcomp-proceedings.com*.

Martínez, V. G., Álvarez, F. H., Encinas, L. H., & Ávila, C. S. (2015). A new edit distance for fuzzy hashing applications. In *Proceedings of the international conference on security and management (sam)* (p. 326).

Neuner, S., Mulazzani, M., Schrittwieser, S., & Weippl, E. (2015). Gradually improving the forensic process. In *Availability, reliability and security (ares), 2015 10th international conference on* (pp. 404–410).

Oliver, J., Cheng, C., & Chen, Y. (2013). Tlsh–a locality sensitive hash. In *4th cybercrime and trustworthy computing workshop (ctc)* (pp. 7–13).

Oliver, J., Forman, S., & Cheng, C. (2014). Using randomization to attack similarity digests. In *Applications and techniques in information security* (pp. 199–210). Springer.

Payer, M., Crane, S., Larsen, P., Brunthaler, S., Wartell, R., & Franz, M. (2014). Similarity-based matching meets malware diversity. *arXiv preprint arXiv:1409.7760*.

Rabin, M. O. (1981). *Fingerprinting by random polynomials* (Tech. Rep. No. TR1581). Cambridge, Massachusetts: Center for Research in Computing Technology, Harvard University.

Rajaraman, A., & Ullman, J. D. (2012). *Mining of massive datasets*. Cambridge: Cambridge University Press.

Rathgeb, C., Breitinger, F., & Busch, C. (2013, June). Alignment-free cancelable iris biometric templates based on adaptive bloom filters. In *Biometrics (icb), international conference on* (p. 1-8). doi: 10.1109/ICB.2013.6612976

Rathgeb, C., Breitinger, F., Busch, C., & Baier, H. (2013). On the application of bloom filters to iris biometrics. *IET Biometrics*.

Ren, L., & Cheng, R. (2015, August). *Bytewise approximate matching, searching and clustering.* DFRWS USA.

Rivest, R. (1992). The MD5 Message-Digest Algorithm.

Roussev, V. (2009). Building a better similarity trap with statistically improbable features. In *System sciences, 2009. hicss '09. 42nd hawaii international conference on* (pp. 1–10). doi: 10.1109/HICSS.2009.97

Roussev, V. (2010). Data fingerprinting with similarity digests. In K.-P. Chow & S. Shenoi (Eds.), *Advances in digital forensics vi* (Vol. 337, pp. 207–226). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-15506-2_15 doi: 10.1007/978-3-642-15506-2_15

Roussev, V. (2011, August). An evaluation of forensic similarity hashes. *Digital Investigation*, *8*, 34–41. Retrieved from http://dx.doi.org/10.1016/j.diin.2011.05.005 doi: 10.1016/j.diin.2011.05.005

Roussev, V. (2012). Managing terabyte-scale investigations with similarity digests. In G. Peterson & S. Shenoi (Eds.), *Advances in digital forensics viii* (Vol. 383, pp. 19–34). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-33962-2_2 doi: 10.1007/978-3-642-33962-2_2

Roussev, V., III, G. G. R., & Marziale, L.

(2007, September). Multi-resolution similarity hashing. *Digital Investigation*, *4*, 105–113. doi: 10.1016/j.diin.2007.06.011

Roussev, V., Richard, I., Golden, & Marziale, L. (2008). Class-aware similarity hashing for data classification. In I. Ray & S. Shenoi (Eds.), *Advances in digital forensics iv* (Vol. 285, pp. 101–113). Springer US. Retrieved from http://dx.doi.org/10.1007/978-0-387-84927-0_9 doi: 10.1007/978-0-387-84927-0_9

Rowe, N. C. (2012). Testing the national software reference library. *Digital Investigation*, *9*, S131–S138.

Sadowski, C., & Levin, G. (2007, December). *Simhash: Hash-based similarity detection.* http://simhash.googlecode.com/svn/trunk/paper/SimHashWithBib.pdf.

Security, H. N. (2013). *The rise of sophisticated malware.* Retrieved from http://www.net-security.org/malware_news.php?id=2543

Seo, K., Lim, K., Choi, J., Chang, K., & Lee, S. (2009). Detecting similar files based on hash and statistical analysis for digital forensic investigation. In *Computer science and its applications, 2009. csa '09. 2nd international conference on* (p. 1-6). doi: 10.1109/CSA.2009.5404198

Symantec. (2010). *Machine learning sets new standard for data loss prevention: Describe, fingerprint, learn* (Tech. Rep.). Symantec Corporation.

Tridgell, A. (2002–2009). *spamsum.* http://www.samba.org/ftp/unpacked/junkcode/spamsum/. Retrieved from http://samba.org/ftp/unpacked/junkcode/spamsum/README (last accessed Feb 4th, 2016)

Winter, C., Schneider, M., & Yannikos, Y. (2013). F2s2: Fast forensic similarity search through indexing piecewise hashsignatures. *Digital Investigation*, *10*(4), 361–371. doi: http://dx.doi.org/10.1016/j.diin.2013.08.003

Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second acm conference on data and application security and privacy* (pp. 317–326).

Ziroff, G. (2012, February). *Approaches to similarity-preserving hashing, Bachelor's thesis, Hochschule Darmstadt.*

