Towards a Process Model for Hash Functions in Digital Forensics

Frank Breitinger[1](✉), Huajian Liu[2], Christian Winter[2], Harald Baier[1], Alexey Rybalchenko[1], and Martin Steinebach[2]

[1] da/sec - Biometrics and Internet Security Research Group,
Hochschule Darmstadt, Darmstadt, Germany
{frank.breitinger,harald.baier}@cased.de, alexryba@yandex.ru
[2] Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany
{huajian.liu,christian.winter,martin.steinebach}@sit.fraunhofer.de

Abstract. Handling forensic investigations gets more and more difficult as the amount of data one has to analyze is increasing continuously. A common approach for automated file identification are hash functions. The proceeding is quite simple: a tool hashes all files of a seized device and compares them against a database. Depending on the database, this allows to discard non-relevant (whitelisting) or detect suspicious files (blacklisting).

One can distinguish three kinds of algorithms: (cryptographic) hash functions, bytewise approximate matching and semantic approximate matching (a.k.a perceptual hashing) where the main difference is the operation level. The latter one operates on the semantic level while both other approaches consider the byte-level. Hence, investigators have three different approaches at hand to analyze a device.

First, this paper gives a comprehensive overview of existing approaches for bytewise and semantic approximate matching (for semantic we focus on images functions). Second, we compare implementations and summarize the strengths and weaknesses of all approaches. Third, we show how to integrate these functions based on a sample use case into one existing process model, the computer forensics field triage process model.

Keywords: Digital forensics · Hashing · Similarity hashing · Robust hashing · Perceptual hashing · Approximate matching · Process model

1 Introduction

One of the biggest challenges in computer crime is coping with the huge amounts of data – the trend is that everything goes digital. For instance, books, photos, letters and long-playing records (LPs) turned into ebooks, digital photos, email and mp3. In addition, we have smartphones providing access to wireless Internet virtually everywhere.

To handle all this, the forensic community developed investigation models to assist law enforcement [1] which mainly describe where investigators should

P. Gladyshev et al. (Eds.): ICDF2C 2013, LNICST 132, pp. 170–186, 2014.
DOI: 10.1007/978-3-319-14289-0_12

start. For instance, in 2006 Rogers presented the computer forensics field triage process model (CFFTPM) which is promoted to be "an on-site or filed approach for providing the identification, analysis and interpretation of digital evidence in a short time frame" [2]. While this model precisely describes how to approach a computer crime, the author states that steps could be very time consuming due to the amount of data. Hence, it is important to reduce the amount of data to be inspected manually by automatically distinguishing between relevant and non-relevant files.

A common technology for automated file identification are hash functions. The proceeding is quite simple: calculate hashes for all files and compare these fingerprints against a reference database (e.g., NRSL [3] from NIST). Depending on the underlying database, known files are either filtered out (no further consideration) or highlighted as suspicious.

Currently, mostly cryptographic hash functions (e.g., SHA-1 [4]) are applied which are very efficient in exact duplicate detection but fail in similar file detection. However, investigators might also be interested in similarity, e.g., detect the correlation between an original image and its thumbnail, which could be solved using approximate matching.

The contribution of this paper is tripartite. Firstly, we give an overview of existing approximate matching algorithms. The second part is a brief comparison of algorithms. Besides a comparison of the same group (bytewise and semantic), we also present a sketchy comparison across groups to clearly demonstrate benefits and drawbacks. The third part of the paper shows a sample use case of how to integrate hashing and approximate matching into existing investigation models wherefore we focus on CFFTPM.

The rest of the paper is organized as follows: First, we give a summary of exiting bytewise and semantic approximate matching algorithms in Sect. 2. Next is an explanation of our test methodology in Sect. 3 followed by the experimental results and assessment in Sect. 4. Out of the assessment, we discuss a possible usage based on a sample use case in Sect. 5. Section 6 concludes the paper.

2 Hashing and Approximate Matching

Hashing has a long tradition in computer sciences and various fields of application. The impact of applying cryptographic hash functions in forensics was first analyzed by White [5] and later by Baier and Dichtelmüller [6]. While White propagates an identification rate up to 85 %, Baier and Dichtelmüller only obtained rates between 15 % and 52 %. This low detection rates result from changing files which happens during updates. Besides, it is very likely that word/excel documents, logfiles or source code changes over the time.

Hence, it is necessary to have approximate matching which is able to detect similarity between objects. In general, one distinguishes between bytewise[1] and semantic approximate matching[2]. While the former one operates on the byte

[1] Well-known synonyms are fuzzy hashing and similarity hashing.

[2] Well-known synonyms are perceptual hashing and robust hashing.

level and thereby follows the view of a computer, the latter one works on a perceptual level and tries to imitate the perspective of a human observer. Of course, operating on a semantic level requires a separate algorithm for each media type, i.e., there need to be algorithms for images, videos, audio, text documents etc.

2.1 Bytewise Approximate Matching

Bytewise approximate matching is a rather new area and probably had it breakthrough in 2006 with a tool called ssdeep. However, it has been proven to be useful for similar inputs detection (e.g., different versions of a file), embedded objects detection (e.g., a jpg within a Word document) or fragment detection (e.g., network packages).

To the best of our knowledge, there are currently seven different algorithms. We ignore the following algorithms:

- bbHash [7] is very slow as it takes about 2 min to process a 10 MiB file.
- mvHash-B [8] needs a specific configuration for each file type.
- SimHash [9] and MinHash [10] can handle near duplicates only.

The remaining three algorithms are briefly explained in the following:

ssdeep. In 2006 Kornblum presented context triggered piecewise hashing (abbreviated CTPH) which is based on the spam detection algorithm from Tridgell [11]. The implementation is freely available and currently in version ssdeep 2.9[3].

The overall idea of ssdeep is quite simple. CTPH identifies trigger points to divide a given byte sequence into chunks. In order to generate a final fingerprint, all chunks are hashed using FNV [12] and concatenated. To represent the fingerprint of a chunk CTPH only takes the least significant 6 bits of the FNV hash resulting in a Base64 character.

sdhash. Four years later Roussev suggested a completely different algorithm named *similarity digest hashing* which resulted in the tool sdhash[4] [13]. Instead of dividing an input into chunks the algorithm extracts statistically improbable features by using the Shannon entropy whereby a feature is a byte sequence of 64 bytes. All features are hashed by SHA-1 [4] and inserted into a Bloom filter [14]. Hence, files are similar if they share identical features.

mrsh-v2. In 2012 Breitinger & Baier proposed a new algorithm [15] that is based on MRS hash [16] and CTPH. Equal to CTPH the algorithm divides an input into chunks and hashes each chunk. In contrast to ssdeep, there are two main modifications. Firstly, we removed the condition of a maximum fingerprint length of 64 characters. Secondly, mrsh-v2 uses now Bloom filters instead of Base64 characters.

[3] http://ssdeep.sourceforge.net; visited 2013-Aug-20.

[4] http://roussev.net/sdhash/sdhash.html; visited 2013-Aug-20.

2.2 Semantic Approximate Matching

Semantic approximate matching can be performed for any media type, but we restrict our analysis to algorithms for images because a main application of semantic approximate matching in digital forensics is the detection of child pornographic images.

Semantic approximate image matching originates from *content-based image retrieval* (CBIR). This term dates back to 1992 [17] while research in the field has an even longer tradition. CBIR systems evaluate the similarity of images based on descriptors for color, texture and shape [18]. A standardized set of image features for CBIR applications has been defined in the MPEG-7 standard [19]. However, the calculation of multiple image features is quite time consuming. The INACT software, which is based on MPEG-7 descriptors, and which has been developed for supporting forensic investigations, requires already 10 s for processing a medium resolution image (640 × 320 pixels) [20]. This is far too slow for a usage under real-world conditions.

The analysis of maybe hundreds of thousands images in an investigation target and up to millions of images in the reference database requires very fast methods for digest calculation and comparison. Hence we focus on image features with the potential for high efficiency:

Histograms. Color histograms, lightness histograms etc. are very basic image features with a long tradition [21]. They just count how many pixels correspond to each value of the observed attribute. Robustness and compactness of the information can be increased by extracting features from the histogram like its first three moments [22], Haar wavelet coefficients (MPEG-7), or range selections [23]. However, the extent of images considered as similar in histogram-based matching approaches is more than just different versions of the same image, as the histograms do not consider any spacial information. Hence such approaches are not well suited for recognizing known images. They are more appropriate for finding images from similar scenes and for clustering images according to the depicted scene.

Low-Frequency Coefficients. While high-frequency parts of images get easily disturbed or lost due to rescaling or lossy compression, low-frequency parts are quite robust. Hence low-frequency Fourier coefficients, DCT coefficients [24], wavelet coefficients [25], etc. can be used as robust image features. The same idea can be used for deriving a key-dependent robust digest by replacing the low-frequency basis functions of the aforementioned transformations with "random smooth patterns generated from a secret key" [24]. Typically, images are scaled to a fixed, low resolution before coefficient calculation for reasons of efficiency.

Block Bitmaps. Robust features can be obtained by dividing an image into a small, fixed number of blocks and calculating one feature bit per block. The most simple version of this approach scales the image down such that it has one pixel per block and sets the bit according to whether the lightness of the pixel is above or below the median lightness [26]. An improved variant called rHash considers the median of each quadrant of the image separately and incorporates a flipping mechanism for robustness against mirroring [27]. Another approach derives an edge map from the image. Such a map can be obtained for example by thresholding the gradient magnitude calculated with the Sobel operator [28,29]. However, more sophisticated edge detection algorithms should be avoided to keep the computing time low.

Projection-Based. This class of approaches has been inspired by the Radon transform, which calculates angle-dependent, one-dimensional projections of the image by integrating the image along straight lines parallel to each projection direction. The hashing algorithm RASH calculates the integral along one radial line for each direction [30]. The proposed improvement RADISH replaces the integral by the variance of the luminance of the pixels on the line [31]. Furthermore, the low-frequency DCT coefficients of the previously calculated angle-dependent function can be used as compact, robust digest of an image [32].

Interest points are another kind of image features. Such points are corners and other prominent points in the image, and various kinds of perceptual hashing based on interest points have been proposed [33,34]. Each interest point can be attached with descriptors of the neighborhood of that point [35,36]. However, the calculation of interest points is computationally expensive – similar to sophisticated edge detection. Lv and Wang report an average processing time of 3.91 s for an image with their default size of 256 × 342 pixels [36].

For the evaluation in this paper we selected 4 algorithms which are potentially suitable for investigating huge amounts of images: DCT based hash [24], Marr-Hildreth filter based hash [33], radial variance based hash [32] and block mean based hash rHash [37]. A similar evaluation of the mentioned first 3 algorithms and a proof-of-concept implementation of the block bitmap approach based on [26] has been done by Zauner et al. [38,39]. In contrast to their evaluation on a relatively small number of high-resolution images, we will show results for larger collections of images and different resolutions.

3 Test Methodology

In order to grade the aforementioned approaches, we need criteria. These criteria are mainly borrowed from existing literature (e.g., [39,40]).

Firstly, we focus on the general efficiency properties of approximate matching algorithms like compression, runtime efficiency and fingerprint comparison in Sect. 3.1. These properties are derived from traditional/cryptographic hash functions and play an important role. The tests for bytewise and semantic approximate matching are explained in Sects. 3.2 and 3.3, respectively.

3.1 Efficiency

The efficiency tests analyze the two main aspects of approximate matching algorithms named compression and ease of computation. It is composed of the following three sub-tests:

Compression: The output of each algorithm is either of a fixed length or variable. In the latter case we simply compare the input against the output size and present a ratio.

Runtime efficiency: Runtime describes the time needed to process an input. Simply put, the time for generate a similarity digest.

Fingerprint comparison: Once similarity digests are created, they are usually compared against a set. To estimate the performance for large scale scenarios, we discuss the complexity of indexing/ordering them.

3.2 Bytewise Approximate Matching

Bytewise approximate matching is especially helpful when analyzing similar files, file fragments and embedded objects. Compared to semantic approximate matching, it is file type independent and therefore also applicable for multiple, different or unknown file types.

In order to classify existing algorithms, there is a framework called FRASH [40] which tests algorithms by the following sensitivity and robustness tests:

Single-common-block correlation (sensitivity) calculates the smallest object that two files need to have in common for which the algorithm reliably correlates two targets. An example is comparing two similar documents.

Fragment detection (sensitivity) quantifies the smallest fragment for which the similarity tool reliably correlates the fragment and the original file. Examples are network packet analysis or RAM analysis.

Alignment robustness analyzes the impact of inserting byte sequences at the beginning of an input by correlating the size of the change to changes in the comparison output. Examples may be logfiles, source code files, office documents or emails.

Random noise resistance analyzes the impact of random edits on the correlation capabilities of the algorithm. An example may be source code files where the name of a variable is changed.

3.3 Semantic Approximate Matching

Semantic approximate matching has two essential properties: robustness and discriminability. Robustness refers to the ability to resist content-preserving processing and distortions, while discriminability is the ability to differentiate contents, i.e. to avoid collisions [41].

Content-preserving processing includes the manipulations that only modify the digital representation of the image content and that apply insignificant perceptual changes on the image content. To evaluate the robustness the following manipulations are applied:

- mirroring: flipped horizontally
- resizing: 61 % downscaling
- cropping: remove outer 10–15%
- rotation: 90 degree clockwise
- blurring: Gaussian filter with 20px radius
- color modification: red and blue plus 100
- compression: JPEG with 5 % quality
- stretching: horizontally 20 % downscaling.

The discriminability can be measured by the false positive rate (FPR) and the false negative rate (FNR). FPR refers to the probability that different contents result in similar hash values, i.e. non-relevant contents are identified as relevant, while FNR denotes the possibility that the same or similar contents produce significantly different digests, i.e. relevant contents are missed in the identifying process. For investigating huge amount of images, low FPR is of essential importance, which must be kept as close as possible to zero in order to reduce the amount of data for the manual inspection followed [37].

4 Experimental Results and Assessment

Our assessment is based on the cryptographic hash function SHA-1 and the bytewise approximate matching algorithms `ssdeep`, `sdhash` and `mrsh-v2`. On the semantic approximate matching side, we run DCT based hash (`dct`), Marr-Hildreth operator based hash (`mh`), `radial` variance based hash and block mean value based `rHash`. The pHash C library[5] offers implementation of the first three functions. The implementation of `rHash` is based on the improved block mean value based hash algorithm in [37].

4.1 Infrastructure

All tests were performed on a conventional business notebook having an Intel Core 2 Duo T9400 CPU clocked at 2.53 GHz with 4 GB RAM.

We used 3 different test sets of images for our experiments: TS_{2000} is a set of 2197 low resolution images (400×266 pixels) having a total size of 53.3 MB; TS_{1500} is a set of 1500 medium resolution images (approximately 1000×800 pixels) having a total size of 603 MB; TS_{1000} is a set of 998 high resolution images (3000×2250 pixels) having a total size of 719 MB.

To define the runtime efficiency we measured the real time which is wall clock time - time from start to finish of the call. This is all elapsed time including time slices used by other processes and time the process spends blocked (for example if it is waiting for I/O to complete).

4.2 Efficiency

Compression. Actually, a fixed length fingerprint is a basic property of hash functions. However, approximate matching only partly fulfills this issue, i.e. `ssdeep` has a maximum length of 108 bytes (but might be shorter) and `sdhash` has proportional length between 1.6 % and 2.6 % (depending on the mode). All perceptual algorithms and SHA-1 have a fixed length output as shown in Table 1.

[5] http://phash.org; visited 2013-Aug-20.

Table 1. Fingerprint length for different algorithms (compression).

dct	mh	radial	rHash	ssdeep	sdhash	mrsh-v2	SHA-1
64 bit	576 bit	320 bit	256 bit	∼600 bit	1.6–2.6 %	∼1 %	160 bit

Runtime Efficiency. The assessment of the runtime efficiency is based on all test sets. The time for building a hash database out of a test set and the time for checking a test set against a database are tested respectively. For hash database building, the original images in each test set are used; for hash checking, the images (except ssdeep) in each set are downscaled by 25 % and compressed by JPEG 20 %.

Table 2. Time for processing test sets (in seconds).

Test Set	TS_{2000}		TS_{1500}		TS_{1000}	
Algorithm	Build	Check	Build	Check	Build	Check
mh	471.39	514.96	436.21	396.52	1015.06	647.06
radial	32.26	114.53	179.87	111.12	799.23	450.44
dct	54.69	30.17	281.24	132.92	1601.42	854.64
rHash	18.37	10.88	92.33	41.55	415.08	220.41
sdhash	6.86	5.90	47.81	30.09	55.49	97.17
ssdeep	5.57	5.79	41.43	48.33	47.16	52.02
mrsh-v2	1.36	6.17	3.83	35.37	4.56	102.83
SHA-1	0.84	0.72	2.35	0.84	2.81	1.12

As shown in Table 2, the cryptographic hash function is the fastest for all test sets, followed by bytewise approximate matching. For all test sets, SHA-1 takes less than three seconds to build hash databases and around one second to check the attacked images. For TS_{2000}, SHA-1 is about 5–8 times faster than ssdeep and sdhash, and for medium and high resolution images, SHA-1 is orders of magnitudes faster than others.

Regarding bytewise approximate matching only, mrsh-v2 is far the fastest for generating. However, ssdeep becomes approximately 6–10x faster when checking the attacked images instead of the original ones. This is because the hash database generated by ssdeep features a file size comparison which significantly speeds up the hash checking process when comparing against files of different sizes.

Among the four semantic algorithms, rHash has the best runtime for images of any resolution. The DCT based hash is very fast for low resolution images but becomes the slowest one while hashing high resolution images, where it takes about 4 times longer than rHash. Comparing with other perceptual hashes, mh is not efficient for low resolution images, but its speed is comparable with radial when coming to large images. The checking process for radial is much slower than the building process, which indicates that peak of cross correlation is not so efficient as hamming distance for hash comparison.

Fingerprint Comparison. While designing the algorithms all developers pay attention on simple hash values and thus they could be compared easily. For instance, the hash comparison time for all the perceptual hashing functions stay below 4 µs, with the exception of radial variance based function, which needs an average of 16 µs for the comparison.

However, the focus of this subsection is the behavior of querying large scale databases on a theoretical level (it is not based on own empirical tests).

Let n be the amount of fingerprints in the database. Cryptographic hash values are stored within hash tables or binary trees and hence their lookup complexity is $O(1)$ or $O(\log n)$, respectively. Considering approximate matching, it was not possible to sort or index the fingerprints for a while. Recently, both algorithms were extended and now have a possibility for indexing. In case of `ssdeep` the authors propagate an improvement of a factor of almost 2000 which is 'practical speed'. For instance, they decrease the time for verifying 195,186 files against a database with 8,334,077 entries from 364 h to 13 min [42]. With respect to `sdhash` we could not find any numbers describing the improvement.

The fingerprint comparison for perceptual hashes is similar to approximate matching – it is not trivial to sort or order them. In the worst case a lookup is an all-against-all (bruteforce) comparison and thus a complexity of $O(n)$. First experimental results using locally sensitive hashing and binary trees showed that minor improvements are possible [43]. However, this is ongoing work and final results are unclear.

To conclude, besides cryptographic hash functions only `ssdeep` offers a possibility to reduce the comparison against large databases down to a practical time.

4.3 Bytewise Approximate Matching

As mentioned in Sect. 2.1, our evaluation focuses on `ssdeep`, `sdhash` and `mrsh-v2`. The former two approaches were already deeply analyzed [40,44] with the main result that `sdhash` outperforms `ssdeep` in all points beside compression.

For instance, assuming a practical relevant file size of 2 MiB, `sdhash` correlates them if they share a common subsequence of 170 bytes while `ssdeep` needs 368 bytes. Regarding fragment detection, `ssdeep` can only reliably detect similarity down to 50 % (in 94 % of all cases) whereas `sdhash` works nearly perfect down to 20 % pieces (in 97 % of all cases). The alignment test which inserts up to 400 % of the original file size only beaks down `ssdeep` that worked acceptable for inserts smaller than 100 % (detection rate of 73 %). The last test described in [40] is random-noise-resistance. While `ssdeep` only works acceptable for less than 0.026 % changes, `sdhash` allows to change over 1.100 % of all bytes and still has a higher detection rate. `mrsh-v2` showed very similar results than `sdhash` and thus we won't give more details.

Due to the fact that we now have at least two algorithms nearly on the same level, we need further comparison criteria. An important aspect is always the false positive and false negative rate.

4.4 Semantic Approximate Matching

To test the robustness of semantic approximate matching, ten images are randomly selected out of TS_{1000} and compose a new test set TS_{10}. Next, we applied the manipulations listed in Sect. 3.3. For easier comparison, the matching scores of different algorithms are represented by a normalized score varying from 0 to 100.

The score of each algorithm after resizing and blurring is always above 95 except `mh` which produces scores between 90 and 95. The color modification yields similar scores but both `mh` and `dct` produce slightly lower scores (90–95) than the others (95–100).

The results of the remaining five manipulations vary enormously and are presented in Fig. 1. For rotation and mirroring, the scores of all algorithms are around 50 except that `rHash` performs very well for mirroring. Cropping is a challenging manipulation, where `radial` performs best, followed by `dct`, and `mh` and `rHash` are not robust. Both `radial` and `rHash` are very robust to compression, where `dct` and `mh` are inferior. Regarding stretching all algorithms deliver scores around 98 except radial which is only at 55.

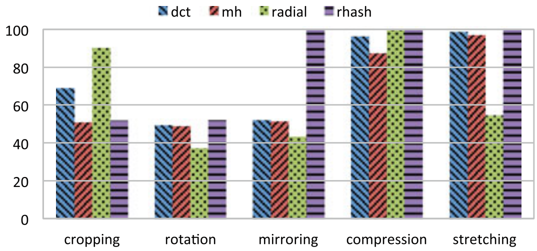


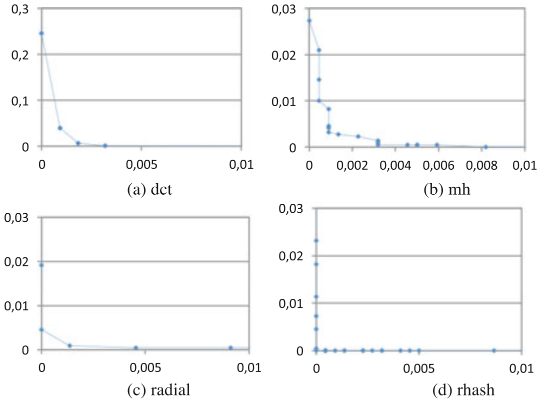
Fig. 1. Average scores for five most influencing perceptual changes on TS_{10}.

As mentioned in Sect. 3.3, the false positive rate (FPR) and the false negative rate (FNR) can be used to measure the discriminability. Hence, we further evaluate the recall precision of different algorithms using TS_{2000} as the known image set and a new test set consisting of 2197 other similar images, called TS_{2000}U, as the unknown image set. The 4394 images in TS_{1000} and TS_{2000}U contain similar scenes of cheerleaders.

First, each algorithm builds a hash database out of TS_{2000}. Then, all images in the known image set, TS_{2000}, are downscaled by 25 % followed by JPEG compression with a quality factor of 20. Finally, digest matching is performed on the modified TS_{2000} and TS_{2000}U respectively.

The results are plotted in Fig. 2. The x-axis denotes FPR and the y-axis FNR. All algorithms obtain fairly good results except `dct`. Among the four algorithms, only `rHash` achieves zero FPR together with zero FNR. Under the requirement of zero FPR, `dct` has the worst result, whose FNR reaches as high as 0.245, while `mh` and `radial` obtain a FNR of 0.027 and 0.0045. For `dct` algorithm, the best tradeoff is to achieve a FPR of 0.0009 with a FNR of 0.0396.

To conclude, all algorithms show good robustness in case of format conversion, blurring, color modification, resizing, compression and stretching (except `radial`), but are not robust against rotation, cropping (except radial) and mirroring (except `rHash`). Furthermore, under combined manipulation of downscaling and compression, all algorithms (except `dct`) achieve good discriminability between known images and unknown images.



(a) dct

(b) mh

(c) radial

(d) rhash

Fig. 2. FNR/FPR of perceptual hashes on TS2000.

Byte Level Changes for Semantic Approximate Matching. Here we briefly analyzed the behavior of semantic approximate matching for byte level changes which corrupt the files. More precisely, we did the following byte level modifications:

- broken data: randomly manipulates 10 bytes all in the file body.
- broken/missing header: deletes the first 128 bytes of the image file.
- missing content: deletes 128 bytes in the middle of the image file.
- missing end: deletes the last 128 bytes of the image file.
- inserting data: inserts 128 bytes from a random image file in the middle.

Real life scenarios are where these manipulations could happen are: transmitting errors, defect hard disk sectors, ram analysis or deleted, fragmented files.

'Missing end' does not influence the score at all–all algorithms output a score of 100. Considering 'missing content' and 'broken data' the scores were still high at round about 90. The lowest scores were returned by 'inserting data' lying between 72 and 82. In all cases the algorithms warn about corrupt JPG data. Regarding 'broken/header' all algorithms failed to produce meaningful results, either by aborting, crashing or delivering out of range errors.

4.5 Summary of Experimental Results

All algorithms have a good to very good compression. Most of them produce a fixed length output or have a upper limit expect sdhash and mrsh-v2 with a proportional length.

As shown in Table 2 the crypto hash SHA-1 is the fastest algorithms followed by the bytewise approximate matching algorithms. Regarding perceptual hashing, there are huge differences in the processing time where rHash is far the fastest.

Considering the fingerprint comparison, it is obvious that the lookup complexity with $O(1)$ for crypto hashes is best followed by ssdeep. Currently it is unclear what improvement can be obtained for sdhash indexing. For now, the worst fingerprint comparison is for perceptual hashing.

The sensitivity & robustness is hard to decide. On the one hand we have the semantic approximate matching algorithms which are very robust against domain specific attacks. However, they do not allow fragment detection (e.g., the header is missing) or embedded object detection (e.g., JPG in a Word document) which are the benefits of bytewise approximate matching. In addition, semantic approximate matching is file domain bound, each domain needs its own algorithm, e.g., images, movies or music.

5 Sample Use Case: Analyzing a USB Hard Disk Drive

In this section we present a reasonable utilization of the three different hash function families on base of the use case *allegation of production and ownership of child abuse pictures*. During a house search the police and IT forensic special agents find besides different computers and DVDs a USB hard disk drive of size 40 GiB (presumably an old backup device).

Following Marcus Roger's process model CFFTPM "time is of the essence" and the search for evidence should start at the crime scene [2]. His key argument is that accused persons tend to be more cooperative within the first hours of an investigation.

During the planning phase of the CFFTPM the forensic investigator chooses hardware (e.g., a forensic workstation equipped with a hardware write blocker for USB devices) and software (implementation of at least one hash function of each family together with respective databases of incriminated pictures) to examine a device onsite for pictures of child abuse. The identification software is configured to run silently in the background and notify the investigator, if potential evidence is found or if the software terminates. An overview of our sample process model for the use case at hand is given in Fig. 3.

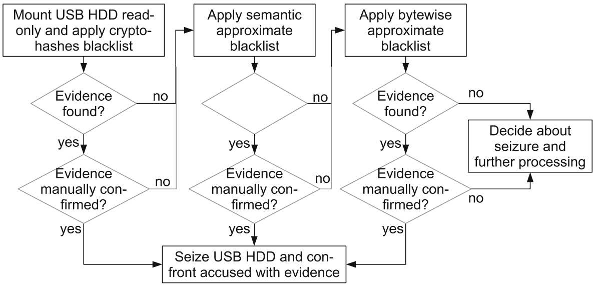

Mount USB HDD read-only and apply crypto-hashes blacklist
Evidence found?
no
yes
Evidence manually con-firmed?
no
yes
Apply semantic approximate blacklist
no
yes
Evidence manually con-firmed?
no
yes
Apply bytewise approximate blacklist
Evidence found?
no
yes
Evidence manually con-firmed?
no
yes
Decide about seizure and further processing
Seize USB HDD and con-front accused with evidence

Fig. 3. Process model in case of onsite search for pictures of child abuse.

In our sample use case the investigator decides that an analysis of a 40 GiB volume is feasible at scene. He mounts the USB HDD read-only into the file system of his forensic workstation and starts the automatic identification of evidence.

Due to their superior efficiency with respect to runtime, compression and fingerprint comparison (see Sect. 4.2), the identification software first applies a blacklist of *crypto-hashes* (e.g., PERKEO[6]) to all files on the USB HDD. If there is a match the identification software notifies the investigator, who manually inspects the potential evidence. If it turns out to be a picture of child abuse, he seizes the USB HDD and informs the police to confront the accused person with the evidence.

If the blacklist crypto-hashes do not yield a trace or if the alert turns out to be a false-positive, the identification software turns to *semantic approximate hashing*. We favor semantic approximate matching, because we expect a higher recall in this specific use case. However, this claim has to be confirmed. The investigator and the software operate analogously in case of a true-positive and false-positive, respectively.

Finally, if after the second step no evidence is found, the software performs file carving on the USB HDD and applies *bytewise approximate matching* to all extracted files/fragments. Please note that in contrast to semantic approximate matching, the final bytewise approximate matching may find fragments or embedded pictures of non-image data files (e.g., pdf, doc). If after all no evidence is found, the investigator decides about a seizure of the device and the further processing, e.g., in the lab.

6 Conclusion and Future Work

In this paper we analyzed the impact of different hashing technologies for forensic investigations. We discussed the three families of crypto hashes, semantic approximate matching algorithms and bytewise approximate hashing functions. We showed that all approaches have different strengths and weaknesses and hence all families are of relevance.

[6] http://perkeo.com; visited 2013-Aug-20.

Semantic approximate hashing functions have proven to be most powerful in the area of content identification. Compared to cryptographic hashing or approximate matching, they offer significantly higher detection quality in the areas of image (or other media) copyright violations or illegal material such as child pornography. However, they are bound to their file domain and it is thus necessary to run perceptual hashing for multiple domains, e.g., images and movies – additional processing time. In addition, these approaches are by default slower than there bitwise opponents.

The key strength of approximate matching is that is able to detect embedded objects, e.g., detect a JPG within a Word document. In addition, it allows fragment detection, which is especially important when dealing with network traffic or defect file systems, e.g., one may analyze the hard disk on the sector or block level.

We identified benefits of cryptographic hash functions, too. The algorithms are superior to their competitors with respect to efficiency (runtime, compression, fingerprint comparison). They are the most recognized in court (yet) and the US NIST provides a comprehensive database containing approximately 100 million hash values. Finally they do not err, i.e., their security properties allow to identify equal files with nearly 100 % probability, which is very important for whitelisting.

We finally presented a sample order of applying the hash function families within a sample use case of investigating a USB HDD at crime scene. However, an actual process model to optimize the operation of hash functions and its validation is still missing. Our next step is to identify typical use cases and propose a reasonable order of application of hash functions (both inside a family and between different families), respectively. Then we validate our proceeding with respect to efficiency and sensitivity and try to abstract from the use cases to a more general model.

Finally we think that it is also necessary to consider the defendants where we have two possibilities. On the one hand the defendant is the ‘normal user’ and not very familiar with the personal computers. Thus, the files reside somewhere unencrypted on the device. Maybe they are processed with a tool to all have the same size. On the other hand the defendant is an ‘expert’ and files might be encrypted. Hence, investigators can try to find fragments in the RAM[7] or in unallocated HDD sectors.

Acknowledgments. This work is supported by CASED (Center for Advanced Security Research Darmstadt).

References

1. Pollitt, M.M.: An ad hoc review of digital forensic models. In: Second International Workshop on Systematic Approaches to Digital Forensic Engineering, SADFE 2007, pp. 43–54 (2007)

[7] Live response.

2. Rogers, M.K., Goldman, J., Mislan, R., Wedge, T., Debrota, S.: Computer forensics field triage process model. In: Conference on Digital Forensics, Security and Law, pp. 27–40 (2006)
3. NIST: National Software Reference Library, May 2012. http://www.nsrl.nist.gov
4. NIST: Secure Hash Standard. National Institute of Standards and Technologies, FIPS PUB 180–1 (1995)
5. White, D.: Hashing of file blocks: When exact matches are not useful. Presentation at American Academy of Forensic Sciences (AAFS) (2008)
6. Baier, H., Dichtelmueller, C.: Datenreduktion mittels kryptographischer Hashfunktionen in der IT-Forensik: Nur ein Mythos? In: DACH Security 2012, pp. 278–287, September 2012
7. Breitinger, F., Baier, H.: A Fuzzy Hashing Approach based on Random Sequences and Hamming Distance. In: ADFSL Conference on Digital Forensics, Security and Law, pp. 89–101, May 2012
8. Breitinger, F., Åstebøl, K.P., Baier, H., Busch, C.: mvhash-b - a new approach for similarity preserving hashing. IT Security Incident Management & IT Forensics (IMF), vol. 7, March 2013
9. Sadowski, C., Levin, G.: Simhash: Hash-based similarity detection, December 2007. http://simhash.googlecode.com/svn/trunk/paper/SimHashWithBib.pdf
10. Broder, A.Z.: On the resemblance and containment of documents. In: Compression and Complexity of Sequences (SEQUENCES'97), pp. 21–29. IEEE Computer Society (1997)
11. Tridgell, A.: Spamsum, Readme (2002). http://samba.org/ftp/unpacked/junkcode/spamsum/README
12. Noll, L.C.: Fowler/Noll/Vo (FNV) Hash (2001). http://www.isthe.com/chongo/tech/comp/fnv/index.html
13. Roussev, V.: Data fingerprinting with similarity digests. Int. Fed. Inf. Process. **337**(2010), 207–226 (2010)
14. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**, 422–426 (1970)
15. Breitinger, F., Baier, H.: Similarity Preserving Hashing: Eligible Properties and a new Algorithm MRSH-v2. In: 4th ICST Conference on Digital Forensics & Cyber Crime (ICDF2C), October 2012
16. Roussev, V., Richard, G.G., Marziale, L.: Multi-resolution similarity hashing. Digital Forensic Research Workshop (DFRWS), pp. 105–113 (2007)
17. Kato, T.: Database architecture for content-based image retrieval. In: Image Storage and Retrieval Systems. Proc. SPIE, IS&T, SPIE Electronic Imaging, San Jose. California, 9–14 February, vol. 1662, pp. 112–123, April 1992
18. Eakins, J., Graham, M.: Content-based image retrieval. University of Northumbria at Newcastle, JTAP report 39, October 1999
19. MPEG: Information technology - multimedia content description interface - part 3: Visual. ISO/IEC, Technical Report 15938–3 (2002)
20. Grega, M., Bryk, D., Napora, M.: INACT–INDECT advanced image cataloguing tool. Multimedia Tools and Applications, July 2012
21. Swain, M.J., Ballard, D.H.: Color indexing. Int. J. Comput. Vis. **7**(1), 11–32 (1991)
22. Stricker, M., Orengo, M.: Similarity of color images. In: Storage and Retrieval for Image and Video Databases III. Proc. SPIE, IS&T, SPIE Electronic Imaging, San Jose, California, 5–10 February, vol. 2420, pp. 381–392, March 1995

23. Xiang, S., Kim, H.J.: Histogram-based image hashing for searching content-preserving copies. In: Shi, Y.Q., Emmanuel, S., Kankanhalli, M.S., Chang, S.-F., Radhakrishnan, R., Ma, F., Zhao, L. (eds.) Transactions on DHMS VI. LNCS, vol. 6730, pp. 83–108. Springer, Heidelberg (2011)
24. Fridrich, J.: Robust bit extraction from images. In: IEEE International Conference on Multimedia Computing and Systems, vol. 2, pp. 536–540. IEEE Computer Society (1999)
25. Venkatesan, R., Koon, S.-M., Jakubowski, M.H., Moulin, P.: Robust image hashing. In: 2000 International Conference on Image Processing, vol. 3, pp. 664–666. IEEE (2000)
26. Yang, B., Gu, F., Niu, X.: Block mean value based image perceptual hashing. In: Intelligent Information Hiding and Multimedia Multimedia Signal Processing. IEEE Computer Society (2006)
27. Steinebach, M.: Robust hashing for efficient forensic analysis of image sets. In: Gladyshev, P., Rogers, M.K. (eds.) ICDF2C 2011. LNICST, vol. 88, pp. 180–187. Springer, Heidelberg (2012)
28. Queluz, M.P.: Towards robust, content based techniques for image authentication. In: Multimedia Signal Processing, pp. 297–302. IEEE (1998)
29. Xie, L., Arce, G.R.: A class of authentication digital watermarks for secure multimedia communication. IEEE Trans. Image Process. **10**(11), 1754–1764 (2001)
30. Lefèbvre, F., Macq, B., Legat, J.-D.: Rash: radon soft hash algorithm. In: EUSIPCO'2002, vol. 1. TéSA, pp. 299–302 (2002)
31. Stanaert, F.-X., Lefèbvre, F., Rouvroy, G., Macq, B., Quisquater, J.-J., Legat, J.-D.: Practical evaluation of a radial soft hash algorithm. In: ITCC, vol. 2, pp. 89–94. IEEE Computer Society (2005)
32. De Roover, C., De Vleeschouwer, C., Lefèbvre, F., Macq, B.: Robust image hashing based on radial variance of pixels. In: ICIP, vol. 3, pp. 77–80. IEEE (2005)
33. Bhattacharjee, S., Kutter, M.: Compression tolerant image authentication. In: 1998 International Conference on Image Processing, vol. 1, pp. 435–439. IEEE Computer Society (1998)
34. Monga, V., Evans, B.L.: Perceptual image hashing via feature points: performance evaluation and tradeoffs. IEEE Trans. Image Process. **15**(11), 3453–3466 (2006)
35. Lowe, D.G.: Object recognition from local scale-invariant features. In: International Conference on Computer Vision, no. 2, pp. 1150–1157. IEEE Computer Society (1999)
36. Lv, X., Wang, Z.J.: Perceptual image hashing based on shape contexts and local feature points. IEEE Trans. Inf. Foren. Sec. **7**(3), 1081–1093 (2012)
37. Steinebach, M., Liu, H., Yannikos, Y.: Forbild: Efficient robust image hashing. In: SPIE 8303. Security, and Forensics, Media Watermarking (2012)
38. Zauner, C.: Implementation and benchmarking of perceptual image hash functions, Master's thesis, University of Applied Sciences Upper Austria, July 2010
39. Zauner, C., Steinebach, M., Hermann, E.: Rihamark: perceptual image hash benchmarking. In: Media Watermarking, Security, and Forensics III. Proc. SPIE, IS&T/SPIE Electronic Imaging, San Francisco, California, 23–27 January, vol. 7880, pp. 7880 0X-1-15, Feb 2011. http://dx.doi.org/10.1117/12.876617
40. Breitinger, F., Stivaktakis, G., Baier, H.: FRASH: a framework to test algorithms of similarity hashing. In: 13th Digital Forensics Research Conference (DFRWS'13), Monterey, August 2013
41. Weng, L., Preneel, B.: From image hashing to video hashing. In: Boll, S., Tian, Q., Zhang, L., Zhang, Z., Chen, Y.-P.P. (eds.) MMM 2010. LNCS, vol. 5916, pp. 662–668. Springer, Heidelberg (2010)

42. Winter, C., Schneider, M., Yannikos, Y.: F2S2: fast forensic similarity search through indexing piecewise hash signatures. http://www.anwendertag-forensik.de/content/dam/anwendertag-forensik/de/documents/2012/Vortrag_Winter.pdf
43. Giraldo Triana, O.A.: Fast similarity search for robust image hashes, Bachelor Thesis, Technische Universität Darmstadt (2012)
44. Roussev, V.: An evaluation of forensic similarity hashes. In: Digital Forensic Research Workshop, vol. 8, pp. 34–41 (2011)