https://doi.org/10.2352/ISSN.2470-1173.2017.7.MWSF-321

Benefits of combining forensic image creation and file carving

York Yannikos, Martin Steinebach, Michael Rettig
Fraunhofer SIT, Darmstadt, Germany

{firstname.lastname}@sit.fraunhofer.de

Abstract

Typical tasks in a forensic investigation are data acquisition, checksum calculation, file recovery, or content identification. These tasks can be performed mostly without user interaction but are still time-consuming, especially when a large amount of data has to be processed. Individual tasks (or sub-tasks they have in common) often do not perform efficiently and the corresponding implementations could be improved.

In this paper we present stream carving, an approach to speed up tasks that are typically performed in a forensic investigation. By identifying and combining similar or identical subtasks and parallelizing most data processing, we are able to decrease the overall processing time significantly. We implemented a stream carving tool that is able to copy, recover, and identify known visual content. The general idea behind stream carving can help developing forensic multi-purpose tools that run several tasks very efficiently.

Introduction

Today a typical digital forensics case involves large amounts of data to be inspected and analyzed. Therefore, a forensic investigator must rely on analysis tools that provide a reasonable trade-off between speed and accuracy. Tasks that are often part of a forensic investigation are creating a forensically sound copy of storage devices, calculating checksums, and recovering deleted files. However, these tasks are often done ineffectively or inefficiently, costing too much time or human resources. One reason for this is that sub-tasks that the individual tasks have in common can be very similar or identical but are performed separately in each task. In the worst case, this results in doing exactly the same work multiple times. This is a waste of resources where efficiency is really important.

Another important aspect in forensic investigations is the handling of multimedia files: If deleted, these files are often difficult to recover (e.g. large collections of illegal video files), especially if they are fragmented. Also, the number of individual files is often too big to process manually (e.g. vast image collections). Methods based on robust hashing (e.g. [22, 25, 26]) are known to be of significant help here, allowing to identify multimedia files even after modifications. However, these methods are not applied until a forensic disk image has been created. This means again that first a time-consuming copy process takes place that is then followed by another slow carving, robust hashing, and look-up process.

Contribution

In this paper we present an optimized workflow for forensic investigations based on creation and analysis of disk images. We introduce stream carving, an architecture to process forensics tasks more efficiently by identifying and minimizing redundant work in typical forensic tasks. With stream carving we utilize well-known forensic methods but orchestrate them in a novel and improved way. Using this architecture we implemented a tool that is capable to perform data acquisition, checksum calculation, file carving, and robust hashing. We evaluated our approach and found that we were able to complete all tasks almost 4 times as fast as compared to standard approaches.

In previous works we focused on the optimization of multimedia file carving with respect to carving recall [23] and the successful retrieval of as many media files as possible [24]. This work is a realization of a concept we introduced in [18] where the basic idea of stream carving is described but an actual implementation and evaluation is not presented.

The remainder of this paper is organized as follows: Sect. *Multimedia File Carving* gives a brief overview about different existing file recovery approaches for multimedia data like pictures or videos. In Sect. *Content Identification* we describe techniques used for content identification, namely cryptography hashing and robust hashing. In Sect. *Stream Carving* we explain the stream carving approach and give an overview about how our stream carving tool works. Sect. *Evaluation* covers the evaluation of the approach. In Sect. *Related Work* we give an overview about similar work and conclude in Sect. *Conclusion*.

Multimedia File Carving

Multimedia file carving describes the process of recovering deleted multimedia data from a storage device by analyzing it as one large binary data stream. Basic carving techniques use header and footer signatures of different multimedia file formats to recognize the beginning or end of a file in the binary data stream. If the header signature of a specific file format is found, e.g. from a JPEG file, the carver searches for the corresponding footer signature. If the search is successful, the data between both signatures is assumed to be valid JPEG data and can be recovered, i. e. stored as a file on a recovery device.

More advanced carving techniques comprise the verification of the file structure of different multimedia file formats as proposed by Garfinkel in [5]. In the same work, Garfinkel proposed *bifragment-gap carving*, a technique to validate file contents with one gap in between. Memon and Pal proposed using greedy heuristics to reassemble multi-fragmented images that works well for bitmap images [10]. In later works, the authors proposed sequential hypothesis testing to find fragmentation points of fragmented image files [12]. With this approach the authors were able to successfully recover most of the fragmented JPEG files of two testsets from DFRWS 2006 and 2007.

Examples of open source file carvers that support basic file carving are *foremost* [2] and *scalpel* [13] as well as *photorec* [6] that also supports advanced file carving techniques especially for multimedia data. Another very effective file carver for fragmented image recovery is the commercial *Adroit Photo Forensics* that is based on the mentioned works of Memon and Pal.

Content Identification

Automatic identification of known content is an important task in most digital forensic investigations: Whitelists, i. e. lists that contain known harmless content, can help to drastically reduce the amount of data a forensic investigator has to inspect, whereas blacklists, i. e. lists that contain known harmful content like child pornography, can even help to end an investigation in shorter time, e. g. if blacklisted content was found that leads to an indictment against the suspect.

One of the main problems in content identification are error rates: High false-negative rates, i. e. content that is reported to be unknown although actually being black- or whitelisted, may result in overlooked evidence data and more data that has to be inspected manually. High false-positive rates, i. e. content that is reported to be black- or whitelisted although actually being unknown, also result in more data to inspect and, in the worst case, may lead to wrong conclusions.

Cryptographic Hashing

Cryptographic hashing is a standard technique to identify known content: It is very reliable in identifying data that is *binary-identical* to known content but cannot be used to identify data that is very similar but not identical. This leads to a high false-negative rate especially when analyzing visual content like images or videos. In digital forensics cryptographic hashing is used for creating checksums of evidence data and for black- and whitelisting. Widely supported cryptographic hash functions are MD5, SHA-1, or the SHA-2 family.

For several cryptographic hash functions weaknesses were found regarding collision resistance [20, 19]. However, these were not relevant for content identification where resistance against preimage attacks is required, i. e. it must be hard to find corresponding content for a given hash or other content that gives the same hash. Thus even MD5, nowadays considered broken, is still a reasonable choice as checksum for content identification.

Robust Hashing

Compared to cryptographic hashing, robust hashing is a very different approach as it is able to recognize similarity based on human perception: If for instance a human finds that two images are very similar to each other, e. g. a picture with the same content but stored in two different file formats (and therefore not being binary-identical), then a robust image hash would be able to also recognize both pictures as being similar. Fig. 1 shows an example of robust image hashes for two similar pictures.

There exist many different robust hashing approaches for all kinds of visually or auditory perceivable data. Current approaches mainly differ in robustness against different kind of attacks, e. g. Roover's image hash which is robust against geometrical operations like scaling or rotation [3]. Other approaches are based on random noise similarity [4] or histograms [21]. With *pHash* there is also a set of robust hash functions available for development

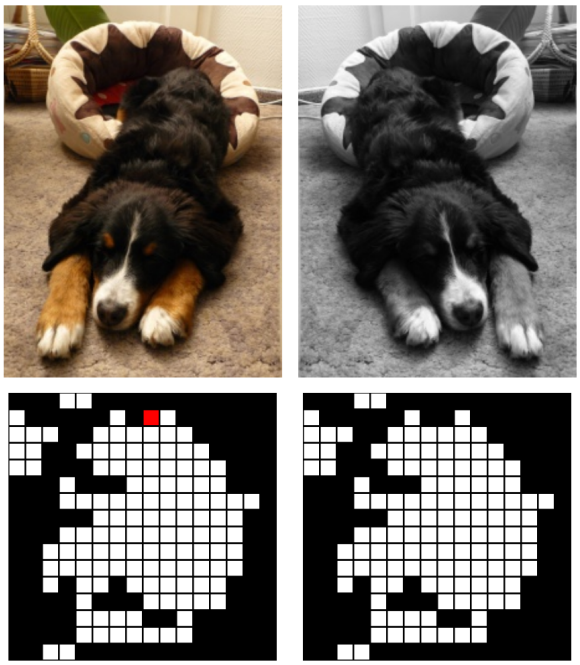


Figure 1. *Visual representation of robust image hashes for two pictures (hash difference is marked in red)*

[1]. However, most existing algorithms do not provide robustness especially against cropping attacks.

In [15] we proposed *rHash*, an efficient robust hash for image files with low error rates, based on a block mean value based perceptual image hash by Yang et al. [22]. We improved the robust hash in several ways with respect to processing speed, error rates, and robustness e. g. against mirroring. The *rHash* algorithm to generate a robust hash for an image file is basically the following (cf. [16]):

- Convert the image to grey scale and normalize the original image into a preset size.
- Let n denote the bit length (e. g. 256 bit) of the final hash value. Divide the pixels of the image I into non-overlapped blocks $I_1, I_2, \ldots, I_n$.
- Calculate the mean of the pixel values of each block, i. e. calculate the mean value sequence $M_1, M_2, \ldots, M_n$ from the corresponding block sequence. Finally obtain the median value M_d of the mean value sequence.
- Normalize the mean value sequence into a binary form and obtain the hash value

$$h(i) = \begin{cases} 0 & \text{if } M_i < M_d \\ 1 & \text{otherwise} \end{cases}$$

An example of a 256-bit *rHash* generation using a sample image is given in Fig. 2.

Since the initial *rHash* approach was vulnerable to cropping attacks, we first used face recognition to create individual robust hashes for any faces found in images [17]. Later we could improve cropping resistance by using image segmentation in order to create one robust hash for each segment of an image [16].

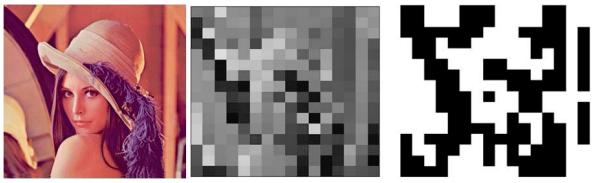


Figure 2. *Original image (left), down-scaled and converted to gray scale (middle), visual representation of the final rHash (right) (cf. [14])*

Stream Carving

A forensic investigation typically includes the previously described techniques for file recovery and content identification. Our goal was to use these techniques in a more efficient way than just applying them consecutively. Therefore, we identified the following 4 tasks of a typical forensic investigation that we wanted to focus on in order to make the whole process less time-consuming:

1. Creation of a forensically sound copy of the input data
2. Checksum calculation of the input data using cryptographic hashes
3. File recovery within the input data
4. Robust hashing of recovered image files and video frames

All 4 tasks need to process (read) the complete input data which is typically done separately by each task. Since reading the same data multiple times is very time-consuming, we created the stream carving architecture in such a way that the input data is read only once and most processing is done in RAM. By that we could significantly decrease the total time required to complete all tasks because reading from a storage device is much more time-consuming than e. g. reading from RAM. The general architecture of the stream carver is shown in Fig. 3.

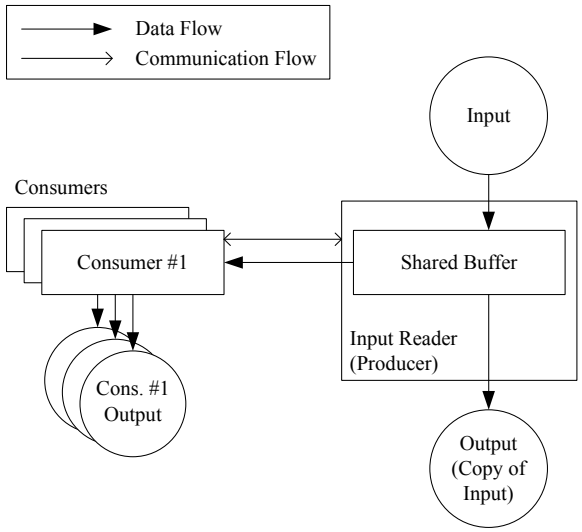


Figure 3. *Stream carving architecture with multithreading (one producer, multiple consumers)*

For the first task we implemented an input data reader with a shared buffer, running in its own core thread (producer). Each time the shared buffer has been filled with a new chunk of input data, the input reader sends notifications to any other threads that need to process the data (consumer). The consumers then process the data currently in the buffer in parallel and notify the producer again when they are finished. The producer then continues to fill the buffer with the next chunk of input data. When no input data is left the producer notifies the consumers again so that they can finalize their individual data processing task.

For the other tasks we implemented consumers that wait for a notification from the producer, each running in a separate thread. The first consumer calculates a checksum of all data that going through the shared buffer (task 2). The second consumer applies image file carving with header verification (similar to [5]) on the data (task 3). Each time the carver recognizes a known image file header signature in the buffer, it creates an in-memory copy of all subsequent data until it finds a corresponding footer signature or a threshold is met. The copied data is then safely stored as a recovered (complete or incomplete) image file.

Additionally, the image file carving consumer puts each recovered image file into a queue and notifies the third consumer: This is a queue manager that is responsible to distribute all image files in the queue to different consumers which perform robust hashing (task 4). For this the queue manager checks the status of the queue in regular intervals and after a notification from the file carving consumer. Each time the number of image files in the queue passes a specific threshold (currently 10), the queue manager creates an additional thread for robust hashing. When the number of image files sinks below that threshold again, the queue manager stops any robust hashing consumers that are currently idle. The resulting robust hashes make a fast identification of visual content possible that is identical or similar to known (black- or whitelisted) content. An overview about the queue management and its interaction with the robust hashing consumers is shown in Fig. 4.

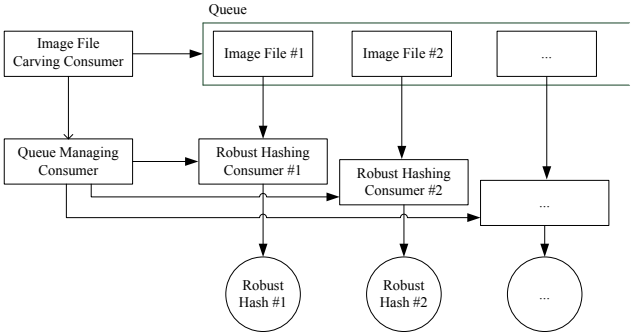


Figure 4. *Queue management and dequeuing with robust hashing consumers*

Overall the stream carver is using a maximum of 13 threads in parallel: 1 core thread, 3 threads for checksum calculation (1 for each cryptographic hash), 1 for file carving, 1 for queue management, and up to 7 for dequeuing/robust hashing. The maximum number of threads for robust hashing can be configured. We also tested the impact of different maximum numbers of robust hashing consumers in the next section.

Evaluation

In order to evaluate the performance of our stream carving tool we run several tests with fixed test data and measured the time required to successfully complete the test. We compared

our approach to consecutively processing each task mentioned in Sect. *Stream Carving*. Additionally, we compared our approach against a more advanced process where tasks 1 and 2 are handled by *dc3dd* [8], a modified version of *dd* that uses parallel data processing (see Sect. *Related Work*).

We executed our tests using Ubuntu 14.10 on a desktop computer with an Intel Core i5 CPU (4 cores with 3.1 GHz each, no hyper-threading), 4 GB DDR3 system memory, and a Samsung 840 EVO solid-state drive. As test data we decided to use a partition representing a typical SDHC memory card used for digital photography that is filled with a reasonable amount of pictures. Therefore we created a 16 GB partition image and completely filled it with random data. Then we formatted the partition image with FAT32 using 8 KiB cluster size (default for that partition size). After these initialization steps we stored 1000 non-fragmented and complete JPEG files in various sizes on the formatted partition. We did not use fragmented or corrupt JPEG files because we did not focus on robust JPEG file recovery in the first place.

We defined the following three test routines where we combined the four tasks data acquisition, checksum calculation, data recovery, and content identification in different ways:

Sequential In the first test routine *dd* is used for data acquisition, the standard tools *md5sum*, *sha1sum*, *sha256sum* are used for calculating the corresponding checksums, *photorec* is used for JPEG file recovery, and *rHash* for calculating a robust hash. All six tools are run in consecutive order using the following Bash code:

```
1  # flush file system cache
2  echo 3 | sudo tee /proc/sys/vm/drop_caches
3  # copy $input to $output
4  dd if=$input of=$output bs=$buffersize
5  # calculate md5, sha1, sha256 of $input
6  md5sum $input
7  sha1sum $input
8  sha256sum $input
9  # recover all JPEG files within $input data
10 photorec /d recovered_files /cmd $input
       ↪ fileopt,everything,disable,jpg,enable
       ↪ ,wholespace,search
11 # build robust hashes of all
12 # recovered JPEG files
13 rhash recovered_files/
```

Listing 1. Sequential *test routine in Bash code*

Parallel+Sequential The second test routine uses *dc3dd* which efficiently combines data acquisition and checksum calculation in one step. *photorec* is again used for JPEG file recovery, and *rHash* is used for robust hash calculation. The three tools are run in consecutive order. Therefore, we took the code from List. 1 and replaced lines 3–8 with the *dc3dd* call:

```
1  # flush file system cache
2  echo 3 | sudo tee /proc/sys/vm/drop_caches
3  # copy $input to $output and calculate
4  # checksums on the fly
5  dc3dd if=$input of=$output bufsz=$buffersize
       ↪ hash=md5 hash=sha1 hash=sha256
6  # recover all JPEG files within $input data
7  photorec /d recovered_files /cmd $input
       ↪ fileopt,everything,disable,jpg,enable
       ↪ ,wholespace,search
8  # build robust hashes of all
9  # recovered JPEG files
10 rhash recovered_files/
```

Listing 2. Parallel+Sequential *test routine in Bash code*

Stream Carving The third routine runs our stream carver that processes all tasks as mentioned in Sect. *Stream Carving* in parallel. From List. 2 we replaced lines 3–10 with the call to our stream carver:

```
1  # flush file system cache
2  echo 3 | sudo tee /proc/sys/vm/drop_caches
3  # copy $input to $output, calculate
4  # checksums, recover JPEG files and
5  # build robust hashes on the fly
6  streamcarver -i $input -o $output -b
       ↪ $buffersize -md5 -sha1 -sha256 -jpg -
       ↪ rhash
```

Listing 3. Stream Carving *test routine in Bash code*

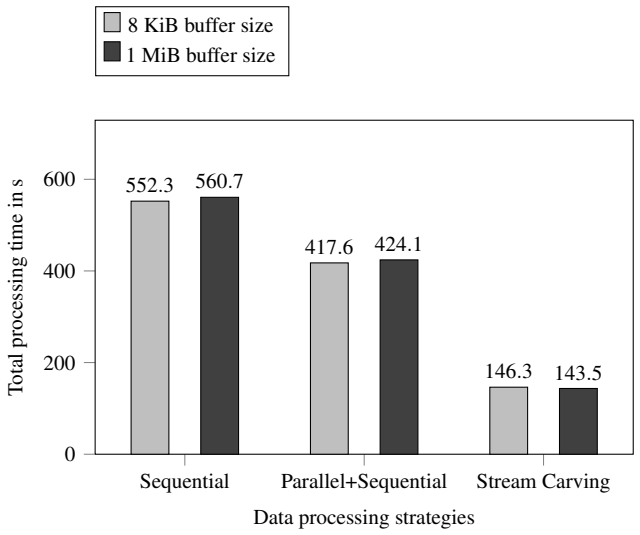
To measure a potential impact of the buffer size used for reading the input data, we ran all 3 test routines using a buffer size of 8 KiB and 1 MiB for *dd*, *dc3dd*, and the stream carver.

We used *dumbbench* [11] for measuring the required time for running each test routine. Each routine was run at least 20 times until we reached our target precision, i. e. the uncertainty on our measurements was $\leq 10\%$. Additionally, we discarded any detected outliers. Both *photorec* and our stream carver were able to successfully recover the complete dataset of 1000 JPEG files. We verified the integrity of each recovered file using SHA-256 checksums.

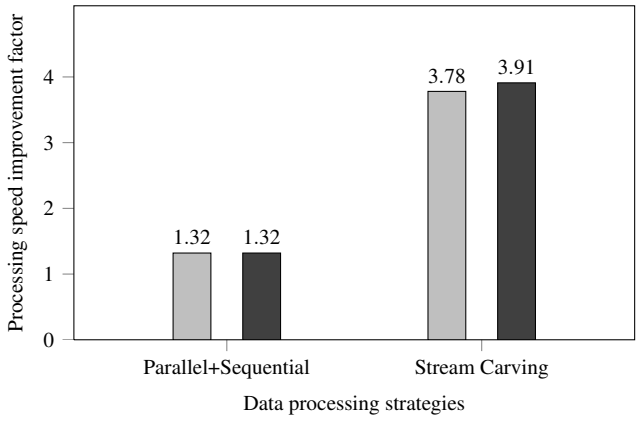
Fig. 5(a) shows the measured time to finish the test routines using the different buffer sizes. We can see that the combination of Parallel+Sequential processing as well as Stream Carving are significantly faster than running all tasks consecutively. In Fig. 5(b) the speed improvements over Sequential processing are given. While Parallel+Sequential processing runs 1.32 times as fast as Sequential processing, Stream Carving runs 3.78 up to 3.91 times as fast depending on the used buffer size. Directly comparing to Parallel+Sequential processing, Stream Carving is still 2.85 up to 2.96 times as fast. The overall uncertainty of all measurements was $< 0.5\%$.

For Sequential processing as well as for Parallel+Sequential processing we saw that both *dd* and *dc3dd* seem to take no benefit from larger buffer sizes, both perform slightly slower when using buffers with a size of 1 MiB instead of 8 KiB. In contrast to this, Stream Carving slightly benefits from using larger buffers.

We also measured the impact of the maximum number of consumers that are created to process the images in the robust hashing queue. For this we used our 16 GB test image again and run dumbbench with the stream carver using an 8 KiB buffer size and different configurations for the maximum numbers of consumers. In Fig. 6 the total processing time is shown in relation to the number of consumers. We can see that using only 1 consumer impacts the total processing time significantly (303 seconds). A maximum number of 5 robust hashing consumers shows the best results with a total processing time of 146 seconds (2.07 times as fast as using 1 consumer).



(a) Total processing time in seconds for each buffer size



(b) Processing speed improvements compared to sequential processing

Figure 5. *Total time needed to process the 16 GB test image for each test routine using different buffer sizes (a) and processing speed improvement of the test routines compared to sequential processing (b)*

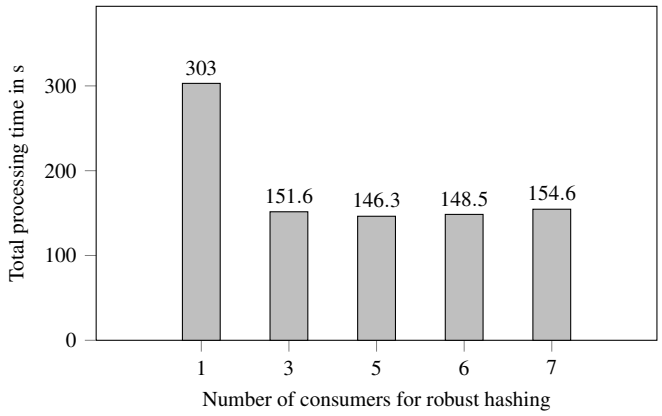


Figure 6. *Total time needed to process the 16 GB test image with stream carving using different maximum numbers of consumers for robust hashing*

Related Work

A few other forensic tools also implement parallel data processing for one or more tasks described in Sect. *Stream Carving*. Two examples are *dc3dd* [8] that we also used in our evaluation, and the very similar *dcfldd* [7]. Both are based on *dd* and use parallel processing for simultaneously copying and calculating different cryptographic hashes of the input data. While *dcfldd* is quite old and has not been updated since 2006, *dc3dd* is available as a patch for the latest *dd* version that includes several bugfixes not included in *dcfldd*.

In [9] the authors modified the *scalpel* file carver to perform multithreaded file carving using an NVIDIA G80 GPU. The authors were able to substantially speed up the file carving process, especially in experiments where they spawned a very large number of threads (1 thread per byte of the input buffer, 10 MB buffer size) that simultaneously searched header and footer signatures in the input buffer.

While many approaches focus on improving processing speed for single forensic tasks such as file carving (*scalpel*), data acquisition (*dc3dd*), or robust hashing (*rHash*), the overall process of a typical forensic investigation is a combination of many of these tasks (see Sect. *Stream Carving*). With stream carving we focused on how to combine optimized single-purpose approaches to build an efficient multi-purpose tool that can significantly speed up the overall investigation process.

We did not look into how commercial or proprietary multi-purpose tools like EnCase Forensics or FTK handle the different tasks in detail because there is typically no technical documentation about internal data processing available.

Conclusion

We proposed stream carving, an architecture that helps to speed-up a typical digital forensic investigation where storage devices have to be inspected and analyzed. Our approach takes common tasks such as data acquisition or image file recovery and combines them in an efficient way that reduces cost-intensive operations like multiple reads from a storage device. We implemented a stream carving tool that is capable to create forensically sound copies of input data, i. e. a seized hard disk, supports calculation of multiple checksums, and can perform file recovery and robust hashing of recovered files. This can help investigators to get data acquisition of seized material efficiently done and also getting a first impression of whether or not known visual content can be found in the data.

Our evaluation shows that minimizing redundant work and parallel computing are important aspects for speeding up forensic investigations. Many forensic tools still work only single-threaded and therefore make not use of typically available multi-core CPUs. Also tools often provide only one single functionality such that investigators have to use multiple tools to get all required tasks done. That makes processing speed improvements for the whole investigation process even more difficult. The difference to known solutions like *photorec*, *scalpel* or *foremost* is that they may use parallel processing when it comes to linear jobs to be executed on a number of items, but they do not utilize the performance gain when carving and copying is done within one process.

In the future we plan to implement support for carving additional multimedia file formats, a fast lookup of robust hashes in

hash databases, and a more robust handling of problems like file fragmentation. We also plan to do more extensive testing with various different hardware and larger as well as smaller datasets.

Acknowledgment

This work was supported by the German Federal Ministry of Education and Research (BMBF) as well as the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP.

References

[1] Aetilius, Inc. pHash - The open source perceptual hash library.

[2] Air Force Office of Special Investigations and Center for Information Systems Security Studies and Research. Foremost, 2001.

[3] Cedric De Roover, Christophe De Vleeschouwer, Frédéric Lefèbvre, and Benoit Macq. Robust video hashing based on radial projections of key frames. *IEEE Transactions on Signal Processing*, 53(10):4020–4037, 2005.

[4] Jiri Fridrich and Miroslav Goljan. Robust hash functions for digital watermarking. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, pages 178–183. IEEE Computer Society, 2000.

[5] Simson L. Garfinkel. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, 4:2–12, 2007.

[6] Christophe Grenier. Photorec, 2007. Available at: `http://www.cgsecurity.org/wiki/PhotoRec`.

[7] Nick Harbour. dcfldd, 2002. Available at: `http://dcfldd.sourceforge.net/`.

[8] Jesse Kornblum. dc3dd, 2008. Available at: `http://sourceforge.net/projects/dc3dd/`.

[9] Lodovico Marziale, Golden G Richard, and Vassil Roussev. Massive threading: Using gpus to increase the performance of digital forensics tools. *Digital Investigation*, 4:73–81, 2007.

[10] Nasir Memon and Anindrabatha Pal. Automated reassembly of file fragmented images using greedy algorithms. *IEEE transactions on image processing*, 15(2):385–393, February 2006.

[11] Steffen Müller. dumbbench (perl module), 2010. Available at: `http://search.cpan.org/~smueller/Dumbbench-0.10/lib/Dumbbench.pm`.

[12] Anandabrata Pal, Husrev T. Sencar, and Nasir Memon. Detecting file fragmentation point using sequential hypothesis testing. *Digital Investigation*, 5(Suppl.):S2–S13, 2008.

[13] Golgen G. Richard III and Vassil Roussev. Scalpel: A frugal, high performance file carver. In *Proceedings of the 2005 digital forensics research workshop (DFRWS 2005)*, 2005.

[14] Martin Steinebach. Robust hashing for efficient forensic analysis of image sets. In *Proceedings of the 3rd ICST International Conference on Digital Forensics and Cyber Crime*. Springer, 2011.

[15] Martin Steinebach, Huajian Liu, and York Yannikos. ForBild: Efficient robust image hashing. In Nasir D. Memon, Adnan M. Alattar, and Edward J. Delp III, editors, *Media Watermarking, Security, and Forensics 2012*, volume 8303 of *Proc. SPIE*, pages 8303 0O–1–8, 2012.

[16] Martin Steinebach, Huajian Liu, and York Yannikos. Efficient cropping-resistant robust image hashing. In *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, pages 579–585, Sept 2014.

[17] Martin Steinebach, Huajian Liu, and York Yannikos. FaceHash: Face Detection and Robust Hashing. In Pavel Gladyshev, Andrew Marrington, and Ibrahim Baggili, editors, *Digital Forensics and Cyber Crime*, volume 132 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 102–115. Springer International Publishing, 2014.

[18] Martin Steinebach, York Yannikos, Sascha Zmudzinski, and Christian Winter. Advanced multimedia file carving. *Handbook of Digital Forensics of Multimedia Data and Devices*, pages 219–269, 2015.

[19] Xiaoyun Wang, Yiqun Yin, and Hongbo Yu. Finding collisions in the full sha-1. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer Berlin / Heidelberg, 2005.

[20] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 19–35. Springer, 2005.

[21] Shijun Xiang, Hyoung-Joong Kim, and Jiwu Huang. Histogram-based image hashing scheme robust against geometric deformations. In *Proceedings of the 9th ACM Workshop on Multimedia & Security*, pages 121–128. ACM, 2007.

[22] Bian Yang, Fan Gu, and Xiamu Niu. Block mean value based image perceptual hashing. In *International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, pages 167–172, December 2006.

[23] York Yannikos, Muhammad-Nadeem Ashraf, Martin Steinebach, and Christian Winter. Automating video file carving and content identification. In *Ninth IFIP WG 11.9 International Conference on Digital Forensics*, pages 195–212. Springer, 2013.

[24] York Yannikos, Jonathan Schlüßler, Martin Steinebach, Christian Winter, and Kalman Graffi. Hash-Based File Content Identification Using Distributed Systems. In *Ninth IFIP WG 11.9 International Conference on Digital Forensics*, pages 119–134. Springer, 2013.

[25] Christoph Zauner, Martin Steinebach, and Eckehard Hermann. Rihamark: perceptual image hash benchmarking. In *Media Watermarking, Security, and Forensics III*, volume 7880 of *Proc. SPIE*, pages 7880 0X–1–15. International Society for Optics and Photonics, 2011.

[26] Xuebing Zhou, Martin Schmucker, and Christopher L Brown. Video perceptual hashing using interframe similarity. In *GI Sicherheit*, pages 107–110, 2006.

Author Biography

York Yannikos received a Diplom (equiv. MS) in computer science from the University of Rostock, Germany (2008). Since then he has worked in the Multimedia Security and IT Forensics division at Fraunhofer SIT in Darmstadt, Germany. His work has focused on the improvement of standard processes and tools in digital forensics.

Dr. Martin Steinebach is the manager of the Media Security and IT Forensics division at Fraunhofer SIT. From 2003 to 2007 he was the manager of the Media Security in IT division at Fraunhofer IPSI. He studied computer science at the Technical University of Darmstadt and finished his diploma thesis on copyright protection for digital audio in 1999. In 2003 he received his PhD from the Technical University of Darmstadt for his work on digital audio watermarking.

Michael Rettig received his MS in computer science from the Hochschule Darmstadt (2016). He has worked in the Multimedia Security and IT Forensics division at Fraunhofer SIT since 2013. His work has focused on multimedia forensics.