Digital Investigation 10 (2013) S50–S58

Contents lists available at SciVerse ScienceDirect

Digital Investigation

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

FRASH: A framework to test algorithms of similarity hashing

Frank Breitinger*,[1], Georgios Stivaktakis [1], Harald Baier [1]

da/sec – Biometrics and Internet Security Research Group, Hochschule Darmstadt, Haardtring 100, 64295 Darmstadt, Germany

Keywords:
Digital forensics
Similarity hashing
Test framework
ssdeep
sdhash

A B S T R A C T

Automated input identification is a very challenging, but also important task. Within computer forensics this reduces the amount of data an investigator has to look at by hand. Besides identifying exact duplicates, which is mostly solved using cryptographic hash functions, it is necessary to cope with similar inputs (e.g., different versions of a file), embedded objects (e.g., a JPG within a Word document), and fragments (e.g., network packets), too. Over the recent years a couple of different similarity hashing algorithms were published. However, due to the absence of a definition and a test framework, it is hardly possible to evaluate and compare these approaches to establish them in the community.

The paper at hand aims at providing an assessment methodology and a sample implementation called *FRASH:* a framework to test algorithms of similarity hashing. First, we describe common use cases of a similarity hashing algorithm to motivate our two test classes *efficiency* and *sensitivity & robustness*. Next, our open and freely available framework is briefly described. Finally, we apply FRASH to the well-known similarity hashing approaches `ssdeep` and `sdhash` to show their strengths and weaknesses.

1. Introduction

The handling of terabytes of data is a major challenge in today's IT forensic investigations. It is important to *automatically* reduce the amount of data that needs to be inspected manually by either removing non-relevant objects like operating system files or marking suspect files like company secrets or child pornography.

Identifying exact duplicates is often solved using cryptographic hash functions. However, it is also helpful to have more flexible and robust algorithms that allow *similarity detection* (e.g., different versions of a file), *embedded object detection* (e.g., JPG in a Word document), *fragment detection* (e.g., analyzing a device on the byte level or network packages) or *clustering files* (e.g., e-mails and Word documents with similar content).

As a consequence the community came up with similarity hashing, which either operates on the *byte level* or on the *semantic level* (e.g., to decide about the similar perception of pictures). Both levels feature their respective strengths and weaknesses. For instance, in the former case an active adversary can circumvent detection by changing the format of a multimedia file or zip it. However, byte level approaches offer fragment and embedded object detection.

In the following we focus on byte level similarity and thus two inputs are equal/similar if they share common byte sequences. This topic has become more and more visible in the community, e.g., Garfinkel (2010) addresses this as one of the candidates to solve the signature metrics abstraction problem.

In general establishing a new algorithm requires a thorough assessment by the community on base of well-known criteria. For instance, the US National Institute of Standards and Technology (NIST) governed the process to standardize

* Corresponding author.
E-mail addresses: frank.breitinger@cased.de (F. Breitinger), georgios.stivaktakis@cased.de (G. Stivaktakis), harald.baier@cased.de (H. Baier).
[1] URL: dasec.h-da.de (Frank Breitinger, Georgios Stivaktakis and Harald Baier).

1742-2876/$ – see front matter © 2013 Frank Breitinger, Georgios Stivaktakis and Harald Baier. Published by Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.diin.2013.06.006

the new symmetric block cipher AES (Nechvatal et al., 2000) or the cryptographic hash function SHA-3 Keccak (Bertoni et al., 2009). Hence, similarity hashing will only be accepted by both the scientific community and practitioners if an assessment methodology and a test framework are available (Garfinkel, 2010; Dewald and Freiling, 2012).

Our contribution within this paper is to provide a test framework, which evaluates existing similarity hashing algorithms. We call it FRASH: a *FR*amework to test *A*lgorithms of *S*imilarity *H*ashing. FRASH is open source and freely available online.[2] On the one hand FRASH is inspired by previous work on 'eligible properties' of similarity hashing algorithms (Breitinger and Baier, 2012d). On the other hand we analyzed multiple papers and how the authors evaluate and compare similarity hashing, e.g., Roussev (2011); Sadowski and Levin (2007); Tridgell (2002–2009). The result of our analysis yields several test cases, which we group in two classes: *efficiency* and *sensitivity & robustness*.

The first class measures the runtime efficiency and the compression rate of the algorithms. Efficiency is important for practical reasons as the computation and storage amount must meet practical needs. The second class addresses sensitivity & robustness issues like random-noise-resistance, alignment robustness, fragment detection, and file correlation. FRASH assesses a similarity hashing algorithm and uncovers its strengths and weaknesses in normal operation and when under attack, respectively.

Currently ssdeep and sdhash are the best-known algorithms. We therefore make use of FRASH to assess them. Our results show that sdhash is superior to ssdeep in all categories except for compression.

The rest of the paper is organized as follows: In Section 2 we discuss the state of the art and relevant literature. In addition, we explain multiple similarity hashing functions and their usage in digital forensics. The scope of FRASH is explained in Section 3. Section 4 provides details about the implementation itself, which is followed by some experimental results in Section 5. Finally, we conclude the paper and point to future work in Section 6.

2. Background

Nowadays a popular use case of cryptographic hash functions within computer forensics is detecting known inputs. The proceeding is quite simple: hash all files on a storage medium and compare the hashes to a reference database. In case of a match, the investigator is convinced that the referred file actually is on the device. The most famous database is the *National Software Reference Library* (NSRL, NIST Information Technology Laboratory (2003–2013)) with its Reference Data Set (RDS).[3] However, due to their security requirements crypto hashes only allow yes-or-no decisions, whereas similarity hashing comparison outputs a match score between 0 and 100.

Identifying similarity has a long history and may start with the Jaccard index suggested by Jaccard (1901) that calculates the similarity of two finite sets A and B by $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. A common application of the Jaccard index is plagiarism detection. Two strings are decomposed (e.g., by spaces or by n-grams) into tokens, which are the elements of the respective sets A and B. Then $J(A,B)$ is used to identify the similarity of the two input strings. However, the sets have to be kept in memory to compute $J(A,B)$, which may be very space consuming. For instance, assume that we split the long byte sequences into 4-g. In the worst case we then have to keep $2^{4\cdot 8} = 2^{32}$ different 4-g in memory, i.e., 16 GiB.

Manber (1994) presented the sif tool to quantify similarities among text files. "Files are considered similar if they have a significant number of common pieces, even if they are very different otherwise." Manber uses a set of *anchors*, which are short character sequences. In order to test for similarity sif searches for anchors and considers the neighborhood, e.g., the next 50 characters. As comparing strings directly is time consuming Manber integrated Rabin fingerprinting (Rabin, 1981) to hash the substrings. Then it is possible to compare numeric values. The main problem is that training data is needed in order to identify reasonable anchors. As a consequence text files of different languages may not be comparable as they do not contain anchors.

In recent years similarity hashing has become more and more popular and thus new approaches were published. All approaches share two commonalities as they consist of

- a generation function that outputs a fingerprint/hash value/digest and
- a comparison function that measures the similarity of two fingerprints.

For the remainder of this paper we use the terms *similarity hashing* and *comparison function*, respectively.

Although FRASH will only be applied to ssdeep and sdhash, the following subsections briefly describe published similarity hashing algorithms and explain, where the algorithms succeed and where they fail in normal operation and when under attack. We decided to mention all algorithms for two reasons. First, this paper should give a rough overview of existing algorithms and how they proceed. Second, the papers describing the algorithms contain valuable test information and point to the authors' concerns. Readers familiar with the existing approaches may skip the remainder of Section 2.

2.1. Context triggered piecewise hashing

Similar to sif, Kornblum (2006) presented an algorithm known as context triggered piecewise hashing (abbreviated CTPH) that is based on the spam detection algorithm of Tridgell (2002–2009). The implementation is freely available and currently in version ssdeep 2.9[4].

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 (Noll, 1994–2012) and concatenated.

[2] https://www.dasec.h-da.de/staff/breitinger-frank/#downloads; (last accessed 2013–04–11).

[3] http://www.nsrl.nist.gov; (last accessed 2013–04–11).

[4] http://ssdeep.sourceforge.net; (last accessed 2013–04–11).