FRASH: A framework to test algorithms of similarity hashing

DFRWS'13 – F. Breitinger, G. Stivaktakis & H. Baier
Frank Breitinger

- Bachelor Degree at University of applied sciences Mannheim in March 2009.

- Master Degree at University of applied sciences Darmstadt in February 2011.
  - Bytewise approximate matching.

- PhD Research Student at CASED since March 2011.

- Current working topics:
  - Testing, comparing and improving existing approaches.
  - Finishing my thesis.
Motivation

- Handling terabytes of data is a challenge in today's IT forensic investigation.
  - Needle in the Haystack.

How to minimize the haystack or enlarge the needle?
Towards a solution

- Automatically identify files.
  - Highlight suspect files (e.g., company secrets or child pornography) or
  - Remove non-relevant objects (e.g., OS files) from further investigation

- Identifying exact duplicates is often solved using cryptographic hash functions.
  - National Software Reference Library (NSRL).

- However, it is also helpful to have more flexible and robust algorithms that allow similarity detection.
  - E.g., different versions of files.

→ Approximate matching (a.k.a. similarity hashing).
Problem

- Establishing a new algorithm requires a thorough assessment by the community on base of well-known criteria.
  - E.g., NIST governed the processes to standardize AES and SHA-3.

- Approximate matching will only be accepted by both the scientific community and practitioners if an assessment methodology and a test framework are available.
What do we expect from tools?

An approach should solve at least one of these “tasks”:

- **Document similarity detection.**
  - Identify related documents, e.g., different versions of a Word document.

- **Embedded object detection.**
  - Identify a given object inside a container, e.g., a JPG within a Word document.

- **Fragment detection.**
  - Identify an original input based on a fragment, e.g., analyzing a device on the byte level or cropped pictures.

- **Clustering files.**
  - Group files that share similar content, e.g., a Word document and an e-mail.
Distinction of approaches

- Semantic approximate matching.
  - Uses contextual attributes of the digital object, and operates at a level close to human perception.

- Syntactic approximate matching.
  - Uses internal structures present in digital objects, e.g., byte structure of network packets.

- Bytewise approximate matching.
  - Matching relies only on the sequences of bits which make up a digital object.
  - Our focus in the following.
Tools for bytewise approximate matching

Most prominent Tools:

- **ssdeep (Jesse Kornblum, 2006)**
  - Divide input in chunks based on the rolling hash. Concatenate chunk hashes to get a final similarity digest (fingerprint).

- **sdhash (Vassil Roussev, 2010)**
  - Extract statistically improbable features, hash them and put them into a Bloom filter which is the similarity digest.

Further approaches:

- bbHash, mvHash-B, mrsh-v2.
What should we test?
Efficiency [1/2]

- **Runtime efficiency (ease of computation).**
  - Fundamental properties of algorithms.
  - Due to large amount of data it is obvious that algorithms have to be fast.
  - Time that the algorithm needs to process the input (reading file from device and generating the similarity digest).
  - FRASH includes SHA-1 as a benchmark.

- **Compression.**
  - Traditional hash functions output a fixed length fingerprint, which is in contrast to approximate matching, where we often have a variable length.
  - Short fingerprints are desirable.
  - Compression measures the ratio between input and output.

\[
\text{compression} = \frac{\text{output length}}{\text{input length}} \cdot 100
\]
Efficiency [2/2]

- **Fingerprint comparison.**
  - An approach is only useful if it has a fast comparison function.
  - Time may vary due to different fingerprint length and comparison algorithms (e.g., Hamming distance of sdhash vs Levenshtein of ssdeep).
  - Fingerprint comparison measures the time of an all-against-all comparison of fingerprints (excludes the fingerprint generation).
Sensitivity & robustness [1/4]

- Single-common-block correlation.
  - Simulates a situation where two files have a single common object”. Considering two files f1 and f2 that are completely different, but share a common object O, “what is the smallest O for which the similarity tool reliably correlates the two targets?” (Roussev, 2011).

- Test procedure:
  - Create two random files f1 and f2 of size $X \in \{512 \text{ KB, } 2048 \text{ KB, } 8192 \text{ KB}\}$ and a common block O of size $X/2$.
  - O overwrites f1 and f2 at different and randomly chosen offsets.
  - If score > 0, reduce O by 16 KB and restart.
  - Test stops when match score = 0.
Sensitivity & robustness [2/4]

- **Fragment detection.**
  - Considering a file, what is the smallest piece/fragment, for which the similarity tool reliably correlates the fragment and the original file? Fragment detection identifies the minimum correlation between an input and a fragment.

- **Test procedure:**
  - Cut X% of the original input length and generates the match score. Default X = 5; max cuts: 100/X - 1.
  - In case the algorithm still identifies similarity, FRASH does a further reduction in 1% steps until only 1% of the input is left.
  - Two different modes:
    - 1. Random cutting: randomly cut at the beginning or at the end.
    - 2. End side cutting: only cut blocks at the end.

- **Alignment robustness.**
  - Analyzes the impact of inserting byte sequences at the beginning of an input whereby we add fixed and percentage blocks.

- **Test procedure:**
  - Test consists of two parameters, the maximum size $M$ and the size of a step $s$.
  - Insert sequentially a block of size $s$ at the beginning and stops after $n$ steps when $n \cdot s \geq M$.

- Two different modes:
  - Fixed blocks: $M = 64$ KB; $s = 4$ KB.
    We decided for a step size of 4 KB as this is the typical sector size.
  - Percentage blocks: $M = 100\%; \ s = 10\%$.
    We decided for a step size of 10\% in order to analyze the impact of large changes. Especially logfiles may grow very rapidly.
Sensitivity & robustness [4/4]

- Random-noise-resistance.
  - Randomly driven test trying to produce false negatives.
  - E.g. a few changes all over the input are sufficient to obtain a non-match for ssdeep.

- Test procedure:
  - What is the maximum number of changes if the match score $s$ is equal or above $X$, i.e., $s \geq X$ where $X = \{90, 80, \ldots, 0\}$.
  - Randomly change bytes all over the input.
    - Edit operations: deletion, insertion, and substitution.
General information about FRASH

- Implemented in Ruby 2.0 and currently supports sdhash and ssdeep.

- Unix environment is necessary to run the framework.
  - Find command is used.

- FRASH is a command-line tool.

  $ frash [-v] [-r] [-t] PATH

  - -v: verbose – prints more details
  - -t: set the test scope: efficiency, single_common_block, fragment, alignment, random-noise.
  - -r: reads path recursively
Integrating new algorithms

- **Requirements for algorithms:**
  - Accept a directory and a file as input.
  - Print fingerprint to standard output, e.g., Base64 encoded.
  - The implementation needs to support an all-against-all comparison.

- **Integration:**
  - Create a wrapper: Copy the wrapper template and modify it.
    - E.g., which flag is used for all-against-all comparison.
Experimental results

- **Tools:**
  - ssdeep 2.9 and sdhash 3.2.

- **Test-corpus:**
  - T5 (4457 files, total 1.78 GB).
  - Types: jpg, gif, doc, xls, ppt, html, pdf and txt.

- **Remark:**
  - Test results are very comprehensive therefore this presentation only contains a rough summary.
Efficiency test - runtime

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Total</th>
<th>Fingerprint comparison</th>
<th>( \frac{\text{algorithm}}{\text{SHA-1}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1sum</td>
<td>0.0013s</td>
<td>5.632s</td>
<td>-</td>
<td>1.00</td>
</tr>
<tr>
<td>ssdeep -s</td>
<td>0.0089s</td>
<td>39.789s</td>
<td>18.217s</td>
<td>7.06</td>
</tr>
<tr>
<td>sdhash</td>
<td>0.0167s</td>
<td>74.278s</td>
<td>346.730s</td>
<td>13.19</td>
</tr>
<tr>
<td>sdhash -p4</td>
<td>0.0066s</td>
<td>29.382s</td>
<td>346.902s</td>
<td>5.22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Avg. hash length</th>
<th>Avg. ratio</th>
<th>Digest file size</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1sum</td>
<td>20 B</td>
<td>0.00466 %</td>
<td>311 KB</td>
</tr>
<tr>
<td>ssdeep -s</td>
<td>57 B</td>
<td>0.01329 %</td>
<td>483 KB</td>
</tr>
<tr>
<td>sdhash</td>
<td>10.6 KB</td>
<td>2.52033 %</td>
<td>61.2 MB</td>
</tr>
</tbody>
</table>

**Conclusion**

- sdhash is slower than ssdeep but outperforms it when it is parallelized.
- ssdeep shows a better compression.
S&R – Single-common-block correlation

- File size of 2048 KB.

<table>
<thead>
<tr>
<th>score</th>
<th>≥ 40</th>
<th>≥ 30</th>
<th>≥ 25</th>
<th>≥ 20</th>
<th>≥ 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. block size (KB)</td>
<td>605</td>
<td>384</td>
<td>368</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Avg. block size (%)</td>
<td>29.53</td>
<td>18.75</td>
<td>17.97</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Matches</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sdhash</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. block size (KB)</td>
<td>912</td>
<td>720</td>
<td>604</td>
<td>480</td>
<td>170</td>
</tr>
<tr>
<td>Avg. block size (%)</td>
<td>44.53</td>
<td>35.16</td>
<td>29.49</td>
<td>23.44</td>
<td>8.28</td>
</tr>
<tr>
<td>Matches</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Conclusion:
  - sdhash is able to detect smaller, common blocks.
S&R – Fragment detection

- Random cutting.

<table>
<thead>
<tr>
<th>fragment size</th>
<th>50%</th>
<th>30%</th>
<th>25%</th>
<th>20%</th>
<th>5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssdeep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. score</td>
<td>65.86</td>
<td>50.90</td>
<td>47.62</td>
<td>44.98</td>
<td>26.00</td>
</tr>
<tr>
<td>Matches (%)</td>
<td>94.64</td>
<td>38.64</td>
<td>20.75</td>
<td>8.86</td>
<td>0.04</td>
</tr>
<tr>
<td>Std. deviation</td>
<td>10.09</td>
<td>10.29</td>
<td>11.34</td>
<td>13.08</td>
<td>1.00</td>
</tr>
<tr>
<td>sdhash</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. score</td>
<td>69.49</td>
<td>70.63</td>
<td>71.18</td>
<td>71.91</td>
<td>76.16</td>
</tr>
<tr>
<td>Matches (%)</td>
<td>100</td>
<td>99.46</td>
<td>98.86</td>
<td>97.33</td>
<td>75.59</td>
</tr>
<tr>
<td>Std. deviation</td>
<td>22.45</td>
<td>23.17</td>
<td>23.27</td>
<td>23.22</td>
<td>22.72</td>
</tr>
</tbody>
</table>

- Conclusion:
  - ssdeep detect file fragments between 50% and 25%; high precision until 45% pieces then ‘matches’ reduces rapidly.
  - sdhash also identifies 5%-fragments in over 75% of all cases.
S&R – Alignment robustness

- Fixed blocks...and percentage.

<table>
<thead>
<tr>
<th>Added block</th>
<th>1 KB</th>
<th>4 KB</th>
<th>16 KB</th>
<th>32 KB</th>
<th>64 KB</th>
<th>400%</th>
<th>29.00</th>
<th>0.06</th>
<th>2.94</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ssdeep</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. score</td>
<td>96.56</td>
<td>91.25</td>
<td>82.66</td>
<td>79.33</td>
<td>76.47</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matches (%)</td>
<td>100</td>
<td>99.69</td>
<td>87.91</td>
<td>74.29</td>
<td>59.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std. deviation</td>
<td>3.79</td>
<td>10.51</td>
<td>16.27</td>
<td>17.84</td>
<td>18.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>sdhash</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg. score</td>
<td>84.11</td>
<td>51.47</td>
<td>64.37</td>
<td>52.68</td>
<td>78.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matches (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Std. deviation</td>
<td>10.57</td>
<td>21.04</td>
<td>17.01</td>
<td>21.05</td>
<td>15.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Conclusion:
  - sdhash detects all (100% matches) but score is alternating.
  - ssdeep runs into trouble the larger the inserted blocks.
## S&R – Random-noise resistance

<table>
<thead>
<tr>
<th></th>
<th>score</th>
<th>≥ 80</th>
<th>≥ 60</th>
<th>≥ 50</th>
<th>≥ 30</th>
<th>≥ 20</th>
<th>≥ 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssdeep</td>
<td>Avg. changes</td>
<td>14.65</td>
<td>43.89</td>
<td>85.17</td>
<td>160.00</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Avg. changes (%)</td>
<td>0.009%</td>
<td>0.026%</td>
<td>0.050%</td>
<td>0.094%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Matches</td>
<td>71</td>
<td>54</td>
<td>29</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>sdhash</td>
<td>Avg. changes</td>
<td>211.67</td>
<td>514.62</td>
<td>729.36</td>
<td>1116.24</td>
<td>1483.54</td>
<td>1860.83</td>
</tr>
<tr>
<td></td>
<td>Avg. changes (%)</td>
<td>0.1216%</td>
<td>0.304%</td>
<td>0.431%</td>
<td>0.660%</td>
<td>0.877%</td>
<td>1.100%</td>
</tr>
<tr>
<td></td>
<td>Matches</td>
<td>78</td>
<td>80</td>
<td>78</td>
<td>85</td>
<td>82</td>
<td>84</td>
</tr>
</tbody>
</table>

### Conclusion:
- ssdeep is vulnerable against noise, e.g., only 29 matches for 85 changes.
- sdhash is very robust, e.g., detects files with >10 while 1% of the bytes changed.
Take home messages

- To establish approximate matching, we need to test algorithms.
  - This shows strengths and weaknesses of approaches.

- An automatic testing is now possible.
  - No dedicated tests are needed anymore (e.g., Vassil 2011).

- FRASH provides a first set of tests.
  - Classes: efficiency AND sensitivity & robustness.

Open issues / future work:
- Integrate further algorithms.
- Do we need further tests / test-classes?
- How to obtain precision & recall rates? (See panel discussion at 1:45pm)
Thank you! - Questions?

- Contact:
  - da/sec – biometrics and internet-security research group darmstadt
  - Email: frank.breitinger@cased.de
  - Web: https://www.dasec.h-da.de/staff/breitinger-frank/
    - FRASH download.