A METHOD FOR PERFORMANCE VERIFICATION OF FREEWARE HASH UTITLIES USING MATLAB

by

SEAN ROBERT JACOBSON

B.S., University of Colorado Denver, 2011

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Recording Arts

2013

This thesis for the Master of Science degree by

Sean Robert Jacobson

has been approved for the

Recording Arts Program

by

Catalin Grigoras, Chair

Jeff Smith

Leslie M Gaston Bird

Date: 7/11/2013

Jacobson, Sean Robert, M.S., Recording Arts

A Method for Performance Verification of Freeware HASH Utilities Using MATLAB

Thesis directed by Associate Professor Catalin Grigoras

ABSTRACT

HASH values are the result of a cryptographic HASH function that can be used for security or file authentication. Due to the rise of file exchange on the internet this value has become even more important when working with digital evidence. In recent years freeware programs have become available online for both commercial and personal use to check for this value. However, there has not yet been a set way to verify these programs and to check that the results they are giving is in fact accurate. This thesis proposes a series of tests that allows for a user or agency to check the program and see if it is indeed accurate.

The form and content of this abstract are approved. I recommend its publication.

Approved: Catalin Grigroas

ACKNOWLEDEMENTS

Firstly I want to say thank you to my mom and dad, Teri and David Jacobson. You put me through school and kept encouraging me. This thesis wouldn't be possible without your support. Secondly to my teachers Jeff Smith and Catalin Grigoras. For giving me great advice on where to find information, putting up with frustrations, and being there when I needed help, thanks. Leslie Gaston Bird a long time teacher for being on my committee and reviewing my multiple drafts. And to my co-workers at the Bookworm, for putting up with me during the final process and my stressful moments. Thanks so much for everything!

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER I

INTRODUCTION

In the emerging field of media forensics digital files are an important part of evidence analysis. When working with a file such as video or an image file it is important for a forensic examiner to know that the file that they are working with is the file that was originally collected at the crime scene. In order to verify evidence integrity upon seizure and throughout processing, HASH values should be calculated and compared.

In today's society the internet has opened up avenues to free exchange of information. This has also caused some problems with knowing whether or not the file received is the original file or if there have been alterations made to it. The HASH value is a tool used to determine whether or not the file's contents is the same or if it is different. There are many different kinds of programs that allow for a person to check the HASH value, some of them have been authenticated by the forensic community and some are freeware programs. In this paper eight different freeware programs are compared to two forensically sound programs to determine whether or not they function the same.

What are HASH Values?

A HASH value is a product of cryptographic HASH function used to authenticate a digital file [1]. The HASH function will take information from the digital file and run through an algorithm to generate the value. In digital media each digital file should have a unique HASH value. No other digital file should have the same HASH value unless it is a clone. [2]

A HASH value should be unique to the file as it currently is. If a person were to make alterations to the file the value would also be changed. In digital forensics the HASH value has come to be more important with the rise in file exchange sites and download utilities on the internet. Below is an example of this using a program called WinHex to check for the HASH value. In Figure 1-1 and Figure 1-2 an example of this is given. A text file called Test File was created and placed in WinHex to get its HASH value. Figure 1-1 is the original file and Figure 1-2 is its copy.

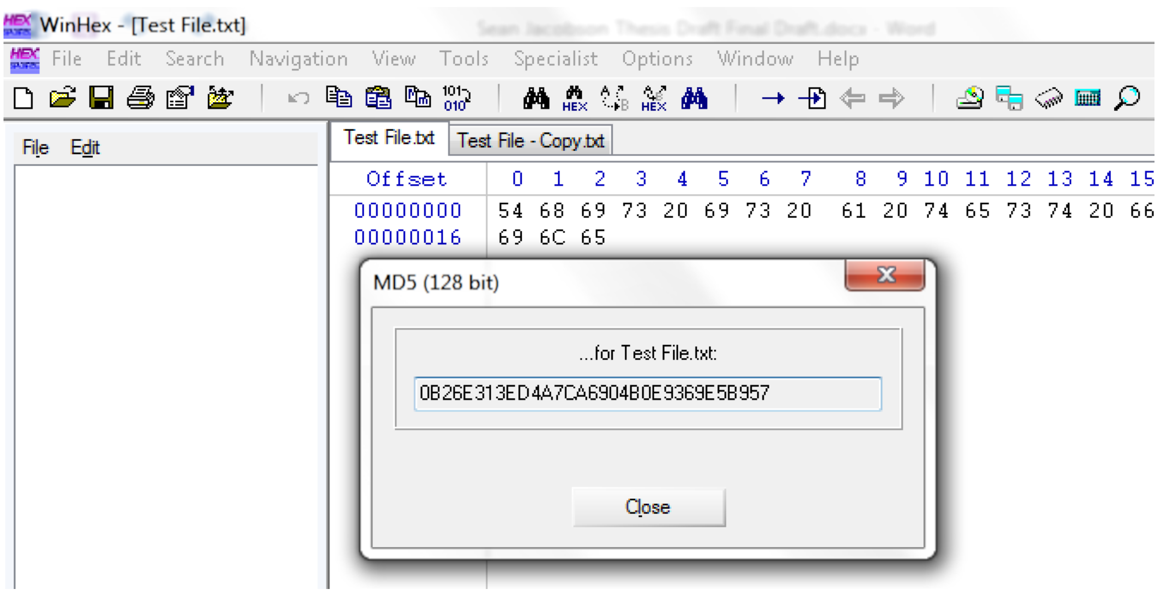


Figure 1-1: The Original File and its HASH.

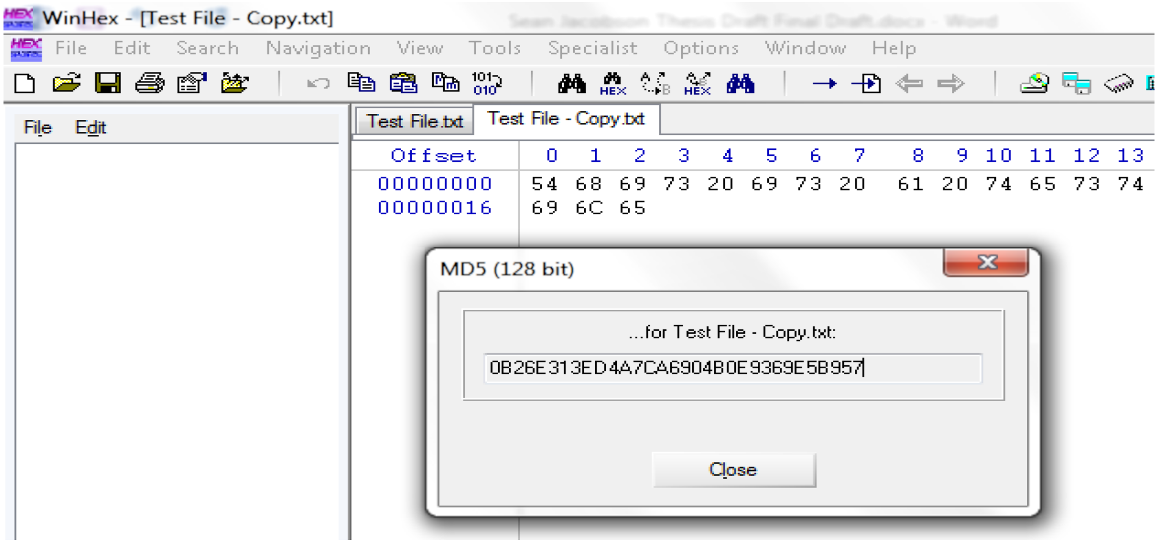


Figure 1-2: The Copy of the Orignal File and its HASH.

In Figure 1-3 and Figure 1-4 one of the file's contents have been changed. This has caused the HASH value of that file to have also changed.

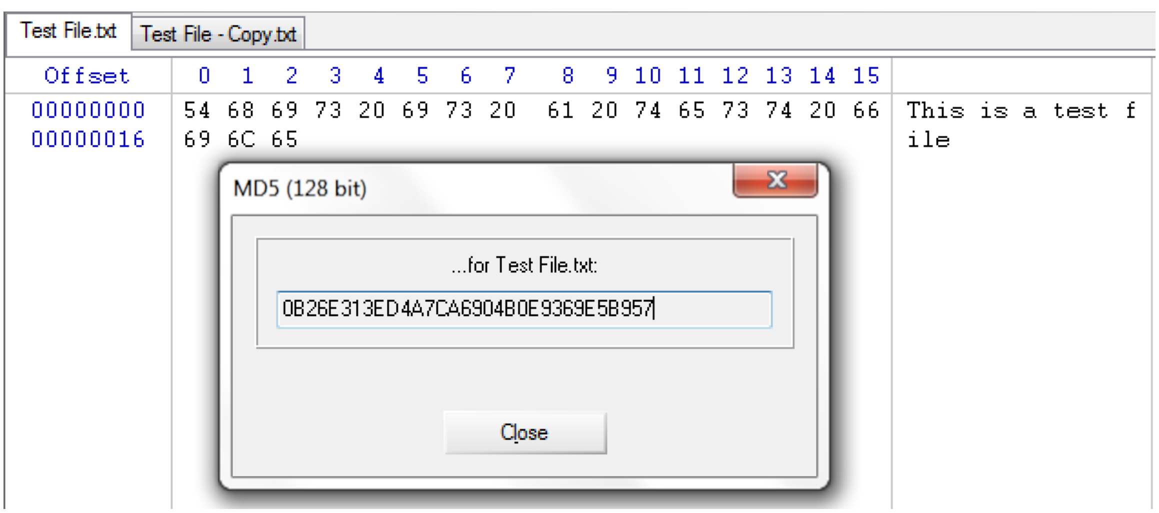


Figure 1-3: The Original File and its HASH.

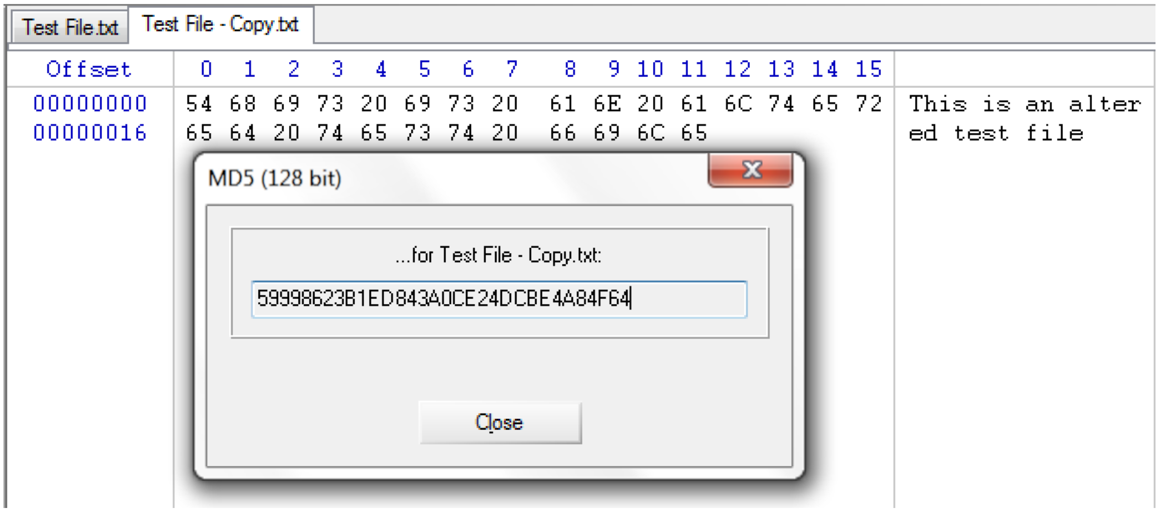


Figure 1-4: The Copied File Altered.

When working with digital information it is important for an examiner to know that the file they are working with is the original. A HASH value, when checked, gives the examiner the assurance that this is indeed a bit-for-bit copy and accurately reflects the original evidence. The tables in Chapter two are the results of testing for errors in multiple HASH checker programs.

Common HASH Algorithms

There are many different kinds of HASH algorithms used today. Some of the most common have been CRC32, Adler32, MD5, SHA1, and WHIRLPOOL.

The cyclic redundancy check (CRC) was first invented in 1961 by W. Wesley Peterson, an American mathematician and computer scientist. The CRC operates by working on multiple blocks of data at one time. However, the CRC is not a cryptographic function. It is a linear function. This means that there is a series of steps that the function needs to go through in order to find the errors. The following is an example of a CRC algorithm [3]:

$$F(X) = Xn\text{-}kG(X) + R(X) = Q(X)P(X),$$

The Adler32 algorithm was developed by Mark Adler in 1995. Here the algorithm calculates two 16-bit checksums and then links them together to form a 32-bit result. The problem with this is that if the data to be analyzed (called a message) is too small than errors will occur [4].

The Message-Digest-Algorithm 5 (MD5) is a cryptographic function designed by Professor Ronal Rivest in 1991 as a stronger version, more secure version of MD4. The algorithm breaks down the message into 512-bits and processes it through the following function [5]:

$$Hi{+}1 = f(Hi,Mi),\ 0 \leq i \leq t - 1.$$

The Secure HASH Algorithm (SHA-1) is another cryptographic function developed by the National Institute of Science and Technology (NIST) as a processing standard. The algorithm has been widely used in government and industry security checks. The algorithm also processes messages in 512-bit blocks. The following is an example of the SHA-1 algorithm [6]:

$$mi = (m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16}) << 1.$$

WHIRLPOOL is one of the first freeware algorithms in the market today. Designed by Paulo S. L. M. Barreto, a cryptographer from Brazil, it is a 512-bit hashing function that works with messages less than 2^{256} in length. Figure 1-5 is a diagram of the algorithm [7]:

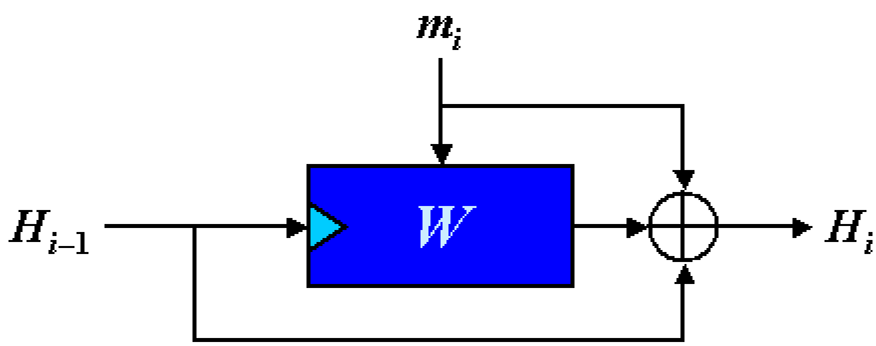


Figure 1-5: WHIRLPOOL Algorithm Diagram.
Source: http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html

Problems with HASH Values

Some of the most commonly used HASH values are the MD5 and SHA-1. However, it is known that neither HASH algorithm is infallible. In 1993 collisions were found that caused some concern with the MD5's algorithm [8]. Collision means that two different documents when put through the HASH's algorithm will come up with the same value. Thus new algorithms have been created to improve upon security. Newer values like SHA-2, WHIRLPOOL and Tiger-192 have not caused problems just yet in testing.

A famous example of HASH collision is "The Story of Alice and her Boss," created by Magnus Daum and Stefan Lucks as a way of illustrating how two files can have the same HASH value. Alice is an intern with Caesar and brings a letter of recommendation for Caesar to sign [9]. The HASH value is shown in Figure 1-6.

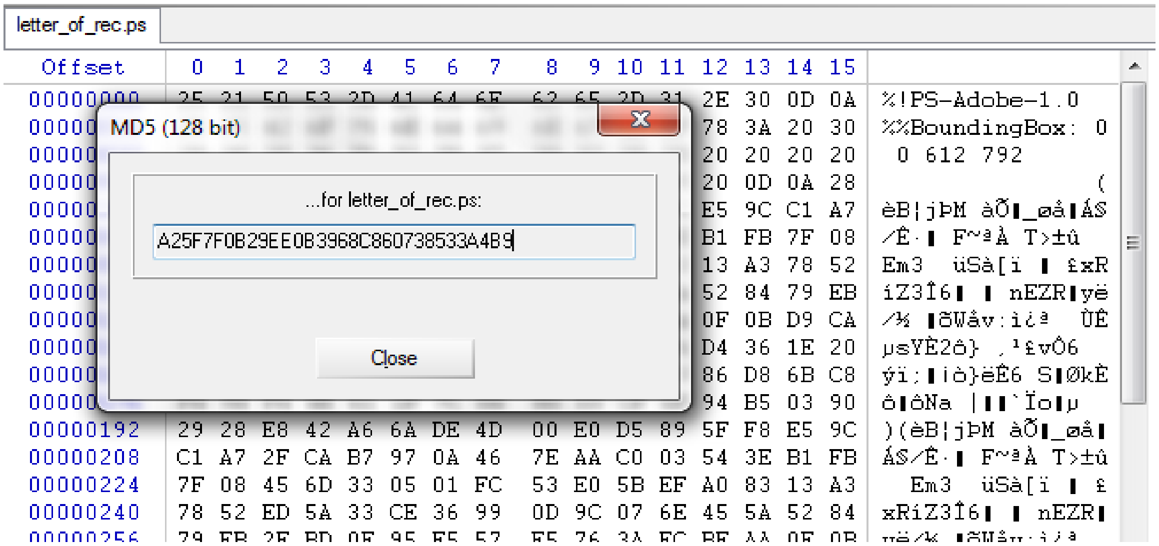


Figure 1-6: The Letter of Recommendation.

In the scenario Alice also wants to have a security clearance and sets up an algorithm so that two documents will have the same MD5. The altered document for security is shown below in Figure 1-7.

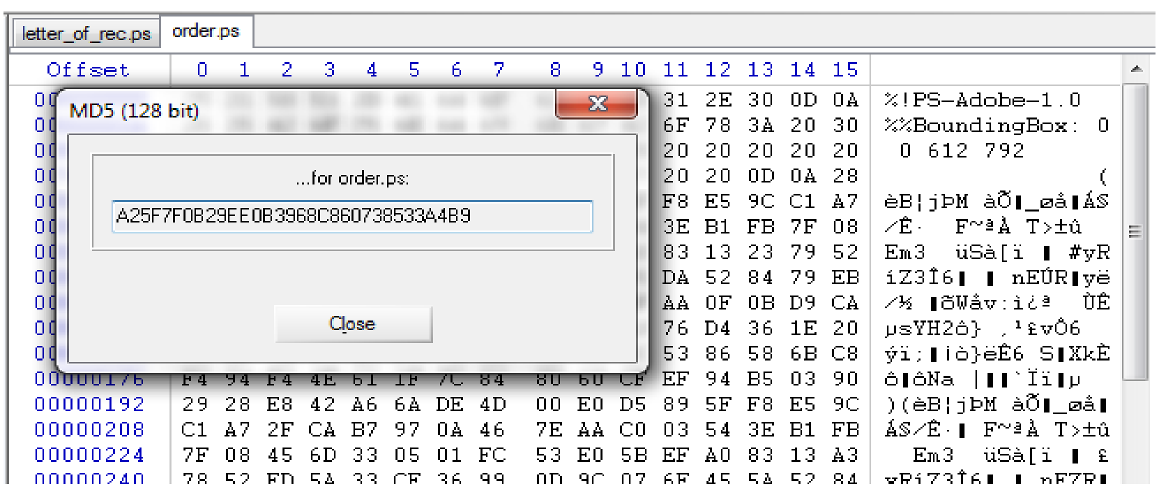


Figure 1-7: The Altered Document.

How could this happen? Two files are not supposed to have the same HASH value right?

In theory a HASH algorithm is supposed to be collision proof. Two cryptographers, Bert von Boer and Antoon Bosselaers, published a paper in 1994 explaining an algorithm that could find collisions within the MD5 algorithm. In the paper they talk about how the search algorithm can search for collisions in the MD5 algorithm by going through four different rounds [9]. After its publication a group of scientists took it a step further.

Xiaoyun Wang and Hongbo Yu are both scientists at Shandong University China. In 2005 they published a paper about the different ways to create collisions and break HASH functions such as MD5. Their results showed how easily the algorithm could be broken by putting a file through different test rounds and generating the result [5]. Wang and Yu went on to publish another paper about collision attacks against SHA-1 with Yiqun Lisa Yin later in 2005 [6].

NIST and NIJ work with HASH

There has been significant work done with HASH values aside from testing for collisions. The National Institute of Justice (NIJ) began work with the National Software Reference Library (NSRL) to create a program that would help a computer forensic examiner [10]. When working on a case with digital files a computer forensic specialist must determine what files are the most important for analysis. The new Reference Data Set (RDS) contains software profiles that can help an examiner find these files. In this dataset a file is given a profile and a HASH value unique to the dataset allowing for faster results. This is an ongoing project and is continually updated [11].

Because of the need for security, there is a need for “collision proof” algorithms. The National Institute of Science and Technology (NIST) had a competition in 2007 for the generation of the SHA algorithm, SHA-3. Sixty-four submissions were received for the first round and were narrowed down to five finalists with the help of public opinion. The hope was that after the selection the algorithm would be able to be collision-proof for at least twenty years. The finalists were placed through a series of tests that would test for strength and compatibility. The algorithms not only had to be able to handle large messages, but shorter ones as well [12]. In 2012 the algorithm Keccak, designed by Guido Bertoni et al., was chosen as the new SHA-3 algorithm.

When using a HASH algorithm through a software program it is important for the user to know that they are getting accurate results. There is some good forensically sound software, like WinHex, that is used by law enforcement when analyzing cases. Unfortunately, this software is often limited to only law enforcement use. Freeware software allows for personal and professional use. This paper proposes a way to test some of these programs for validity.

CHAPTER II

PROPOSED FRAMEWORK AND EXPERIMENTAL PROCESS

In the forensic sciences, it necessary to employ techniques and tools that are known to generate repeatable and reproducible results. This means that the testing procedure must be able to generate the same results when done by another person. During testing a series is run more than once in order to make sure that accuracy is maintained without bias to allow for reliability and consistency.

The following is a proposal when testing different freeware HASH programs for validity and accuracy.

In this series of tests ten different kinds of file types were used. The files were chosen based on commonality and what can be easily found in a personal computer. For each file type one hundred files were created. The files were numbered from one to one hundred. Example: Book001.xlsx, this is the first file for the Excel Documents. The file types made were:

- Excel Documents
- Word Documents
- JPEG Image Files
- Notepad Text File
- PDF Document
- MP3 Audio File
- WAVE Audio Fie
- WMA File

- AVI Video File
- QuickTime Video File

Once each file was created ten different kinds of HASH checker utilities were used to check both the MD5 and SHA-1 HASH values. Eight of the programs were freeware programs that can be found and downloaded from the internet and two programs were forensically sound programs commonly used by forensic examiners. The reason for this was to provide the ground truth when comparing HASH values to each other. These values will be called root values. The programs used were:

- FTK, forensically sound program
- WinHex, forensically sound program
- Advanced Hash Calculator (AHC)
- Arpoon
- Febooti Hash-CRC
- HashTab
- HashGenerator
- MD5-SHA1 Hash Utility
- IgorWare Hasher
- SFVNinja

Each of the different file types were put into a program one at a time and then run three times to for later comparison. The value was copied and placed into an Excel spreadsheet. Three columns were labeled Test 1, Test 2, and Test 3 for each of the different runs.

	A	B	C	D
1	File Name	Test 1	Test 2	Test 3
2	Book001	a2363b2057eeb8f5f33a2150ee380f14	a2363b2057eeb8f5f33a2150ee380f14	a2363b2057eeb8f5f33a2150ee380f14
3	Book002	3c5fb96eed88d02aa6c800351a5fd32e	3c5fb96eed88d02aa6c800351a5fd32e	3c5fb96eed88d02aa6c800351a5fd32e
4	Book003	0294a1546a21f54368f58b0e25153925	0294a1546a21f54368f58b0e25153925	0294a1546a21f54368f58b0e25153925
5	Book004	61dc6135505a045c3f496e93546b9db1	61dc6135505a045c3f496e93546b9db1	61dc6135505a045c3f496e93546b9db1
6	Book005	8e60cea5d634646bbd69794275084d13	8e60cea5d634646bbd69794275084d13	8e60cea5d634646bbd69794275084d13
7	Book006	1fa78edadab9c018ae42ade44e3f6246	1fa78edadab9c018ae42ade44e3f6246	1fa78edadab9c018ae42ade44e3f6246
8	Book007	281d1ac905c116d497eeb1ae0e6872f7	281d1ac905c116d497eeb1ae0e6872f7	281d1ac905c116d497eeb1ae0e6872f7
9	Book008	ef49e68d7f1da886dcdfcf6837931867	ef49e68d7f1da886dcdfcf6837931867	ef49e68d7f1da886dcdfcf6837931867
10	Book009	2921c6466bb35e4005b0dfaf9b17861a	2921c6466bb35e4005b0dfaf9b17861a	2921c6466bb35e4005b0dfaf9b17861a
11	[illegible]	[illegible]	[illegible]	[illegible]

Figure 2-1: Excel Spreadsheet Example.

After all the HASH values were gathered and placed into Excel MATLab was used to compare each of the value to ensure that they were the same. A script created by Catalin Grigoras was used for this. The full script can be found in the Appendix.

In MATLab each freeware value was compared to the two root values, first FTK then WinHex. The Excel documents were loaded in MATLab and the script was run. First the three columns were compared to see if the values matched. If they did a 1 would appear in the row and a message "All the HASH values match" would appear.

```
EDU>> SeanThesis01
All the c11 HASH values match.
All the c12 HASH values match.
All the c13 HASH values match.
All the c21 HASH values match.
All the c22 HASH values match.
All the c23 HASH values match.
All the HASH values match.
fx EDU>>
```

Figure 2-2: All HASH Values Match.

```
c1 =

     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1
```

Figure 2-3: All HASH Values Match, columns.

If the values did not match, a 0 would appear in the work and a message "Check the HASH differences" would appear.

```
EDU>> SeanThesis01
All the c11 HASH values match.
All the c12 HASH values match.
All the c13 HASH values match.
All the c21 HASH values match.
All the c22 HASH values match.
All the c23 HASH values match.
Check the HASH differences.
fx EDU>>
```

Figure 2-4: HASH Value Differences.

```
c1 =

     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

Figure 2-5: HASH Value Differences, column.

If a difference was detected the file would be run again through that HASH program and then run through MATLab again. If a value of 0 appeared again then the file was run through the program once more as well as MATLab. If the value was again 0 then it would be assumed that there was something wrong within the freeware HASH checker and it would fail the test.

Once all three columns had matching values the freeware file values were compared to the root value. If the values all matched a value of 1 was given and a message “All the HASH values match” would appear. If the values did not match a value of 0 was given and a message “Check the HASH differences would appear.” If a difference was detected the file would be run again through that HASH program and then run through MATLab again. If a value of 0 appeared again then the file was run through the program once more as well as MATLab. If the value was again 0 then it would be assumed that there was something wrong within the freeware HASH checker. Results were recorded in an Excel spreadsheet. A green X was used to indicate that all files matched and a red O was used to indicate where files did not match despite being tested three different times.

Table 1: Excel Document MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 2: Excel Document SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 3: JPEG Image File MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 4: JPEG Image File SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 5: PDF File MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 6: PDF File SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 7: Text File MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 8: Text File SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 9: Word Document MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 10: Word Document SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 11: WAVE File MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 12: WAVE File SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 13: MP3 File MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 14: MP3 File SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 15: WMA File MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 16: WMA File SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 17: AVI Video File MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 18: AVI Video File SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 19: QuickTime Video File MD5 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

Table 20: QuickTime Video File SHA-1 Results

Checker Name	FTK	WinHex
FTK	X	X
WinHex	X	X
AHC	X	X
Arpoon	X	X
Febooti Hash-CRC	X	X
HashTab	X	X
HashGenerator	X	X
MD5 SHA1	X	X
IgorWare Hasher	X	X
SFVNinja	X	X

CHAPTER III

CONCLUSION

As seen in the previous tables there were no errors when gathering the HASH values from these freeware programs. However, this represents only a fraction of the different kinds of programs available online for download off of the internet. It is recommended that new programs are tested in a similar way, if not the same way, before they are considered to be verified. Testing other HASH algorithms, such as WHIRLPOOL, can help in the verification process. As a reminder MD5 and SHA-1 used together helps to safeguard against errors when safeguards are checked.

Freeware software is something that can be used to verify digital evidence from a working case. Proper protocol for handling digital files should still be followed depending on the Standard Operating Procedure (SOP) setup by a department. It is still recommended to double check the HASH values with a forensically based program like FTK, but when gathering evidence at a crime scene freeware programs can be used.

This thesis has proposed a way of testing freeware software for errors, but not for collisions. Future tests can be done with the dataset to see if collisions occur when the HASH of each file is compared to the other.

BIBLIOGRAPHY

1. *What is a Hash Function?* 2012, EMC Corporation. 7 Jul 2013 <http://www.rsa.com/rsalabs/node.asp?id=2176>

2. Lu, Wenjun, Avinash L. Varna and Min Wu. "Forensic Hash for Multimedia Information." *Media Forensics and Secutiry II* 7541 (2010).

3. Peterson, William Wesley, and Daniel T. Brown. "Cyclic codes for error detection." *Proceedings of the IRE* 49.1, 1961: 228-235.

4. Maxino, Theresa. "Revisiting Fletcher and Adler Checksums." (2006).

5. Wang, Xiayon, and HongBo Yu. "How to Break MD5 and Other Hash Functions." *Lecture Notes in Computer Science* 3494 (2005): 19-35

6. Wang, Xiaoyun, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1." *Advances in Cryptology–CRYPTO 2005*. Springer Berlin Heidelberg, 2005.

7. *The WHIRLPOOL Hash Function.* Ed. Paulo S. L. M. Barreto. 2008. 7 Jul 2013 <http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html>

8. den Boer, Bert, and Antoon Bosselaers. "Collisions for the compression function of MD5." *Lecture Notes in Computer Science* 765 (1994): 293-304

9. Daum, Magnus and Stefan Lucks. *Hash Collisions (The Poisoned Message Attack): "The Story of Alice and her Boss".* 26 May 2013 <http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/>

10. National Institute of Standards and Technology (NIST), and United States of America. "Forensic Examination of Digital Evidence: A Guide for Law Enforcement." 2004

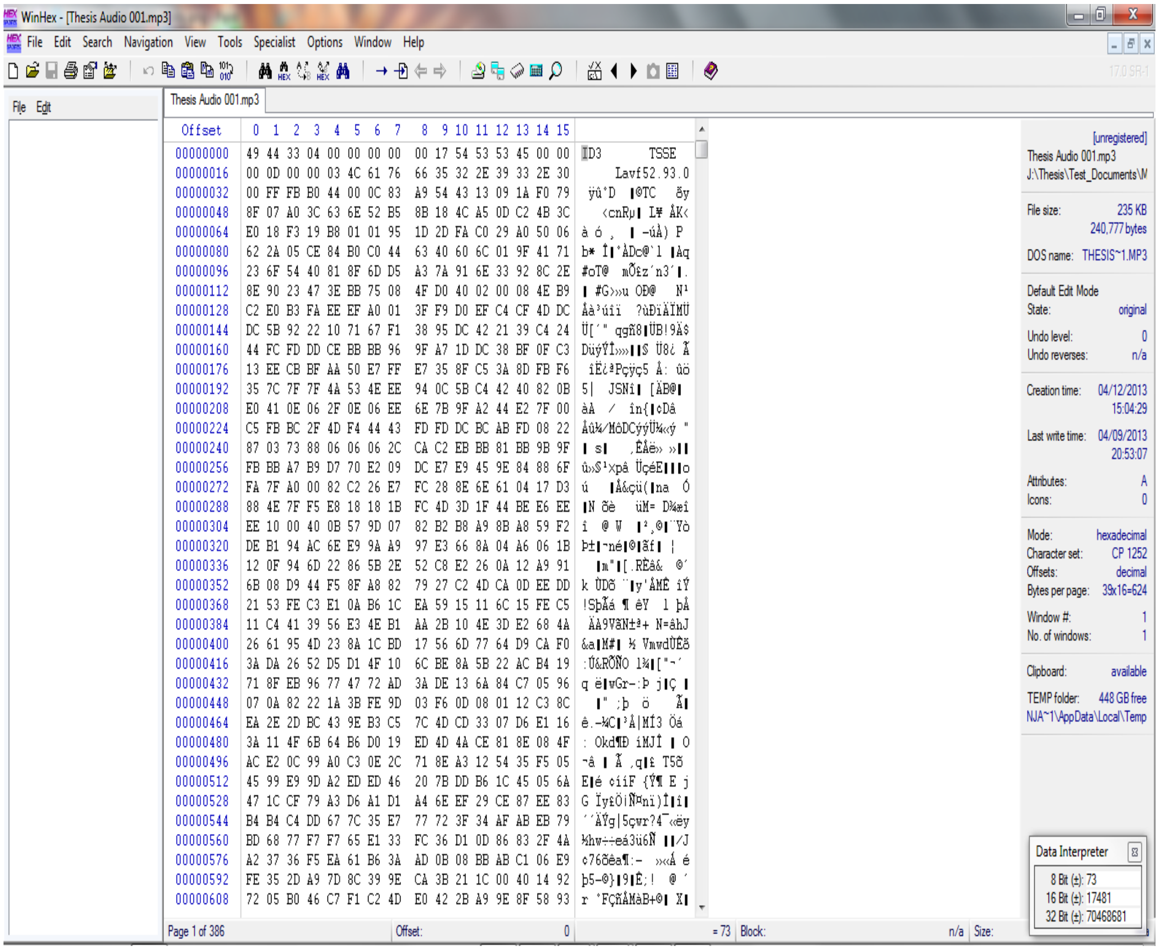
11. *National Software Reference Library.* 20 Aug. 2003. National Institute of Software and Technology. 7 Jul. 2013. <http://www.nsrl.nist.gov/>

12. Turan, Meltem Sonmez et al. *Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition*. NIST Interagency Report 7764, 2011.

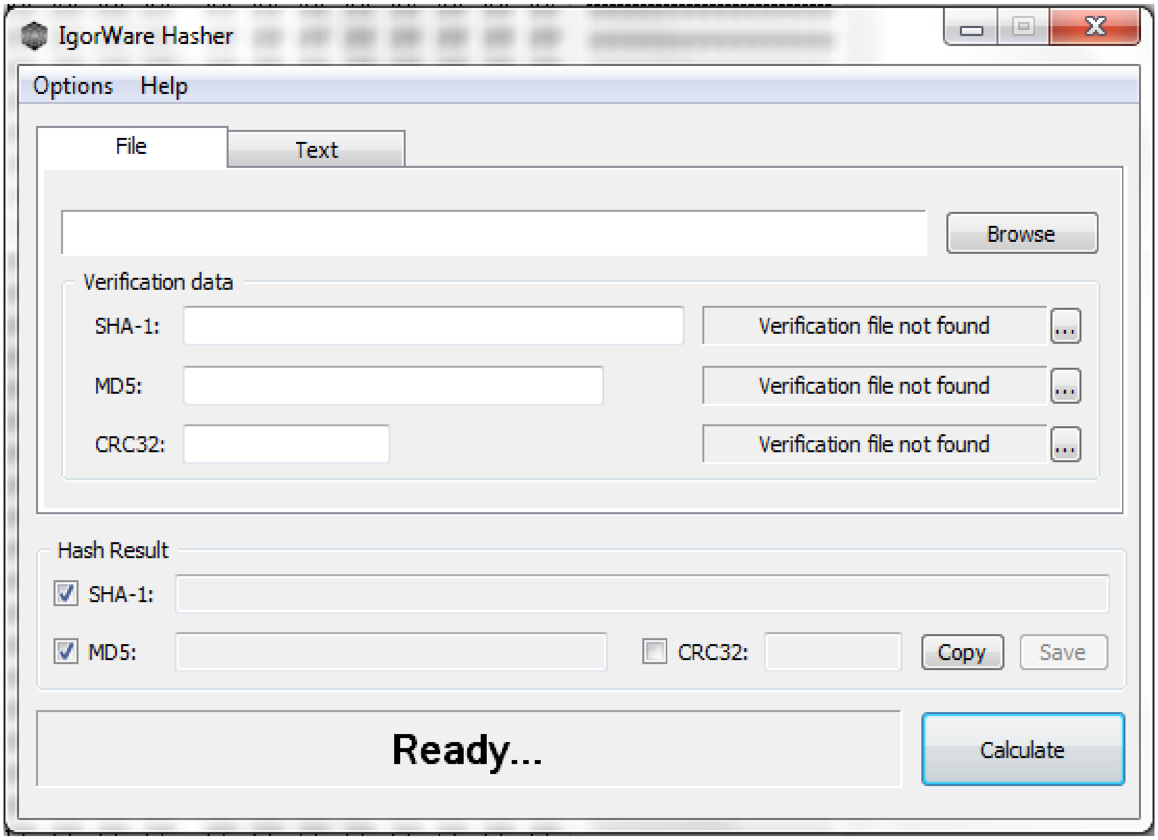
APPENDIX

HASH Programs

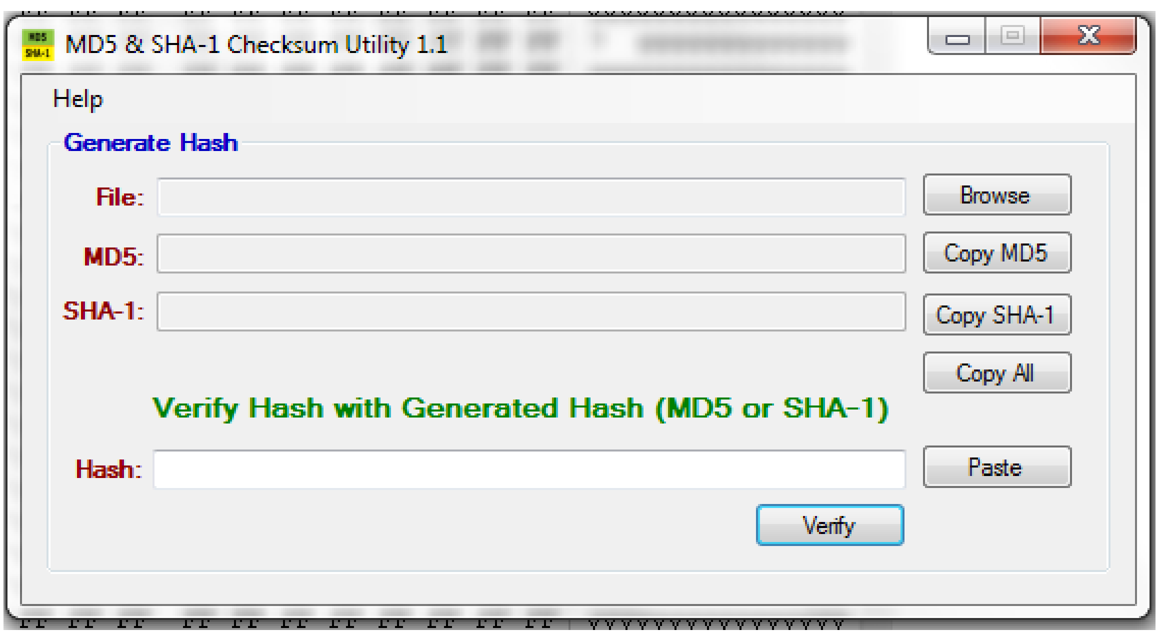
WinHex



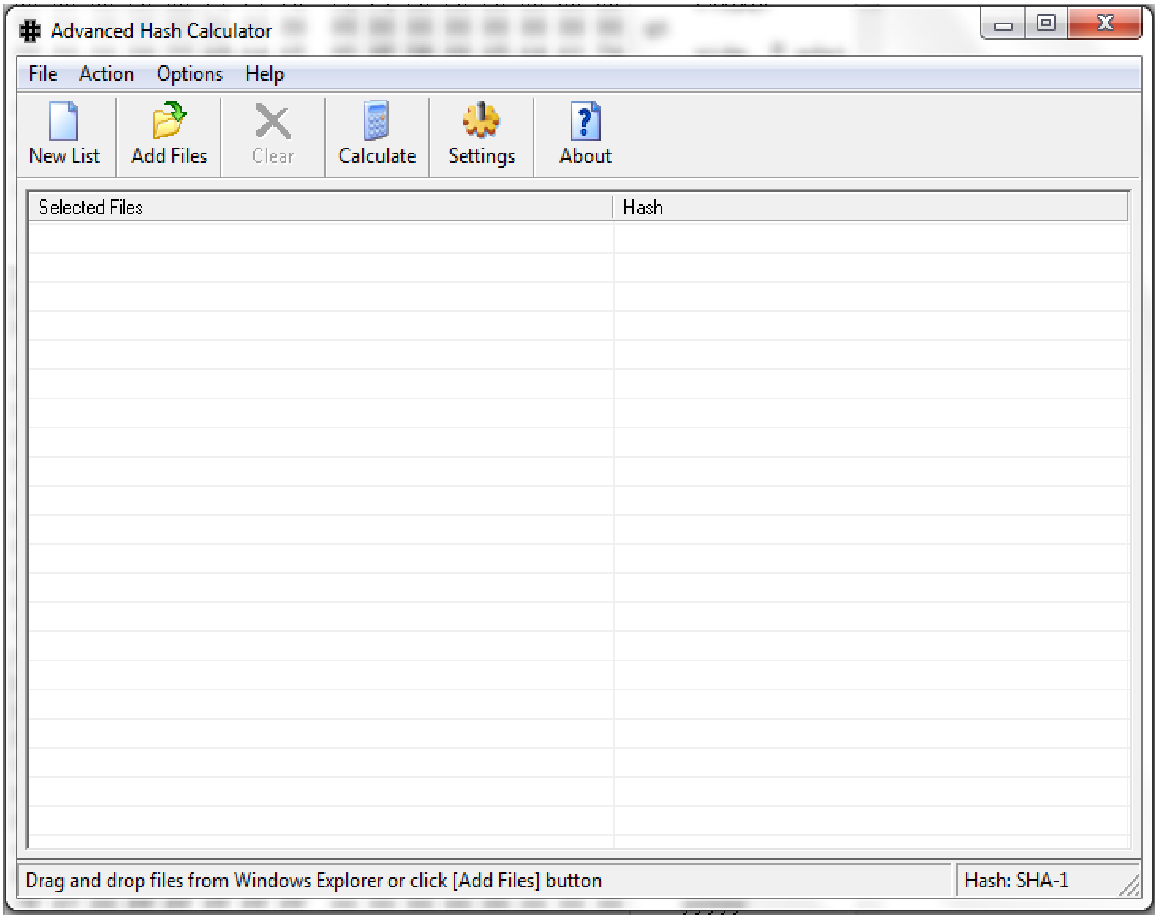
IgorWare Hasher



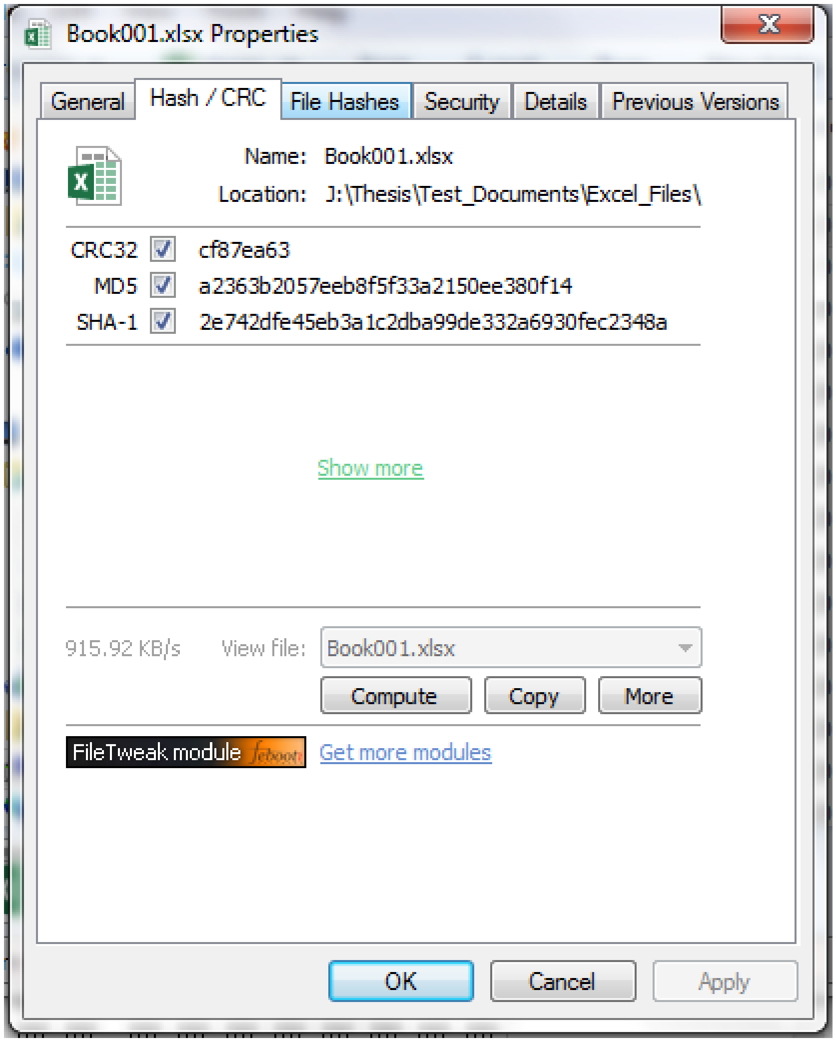
MD5-SHA1 Utility



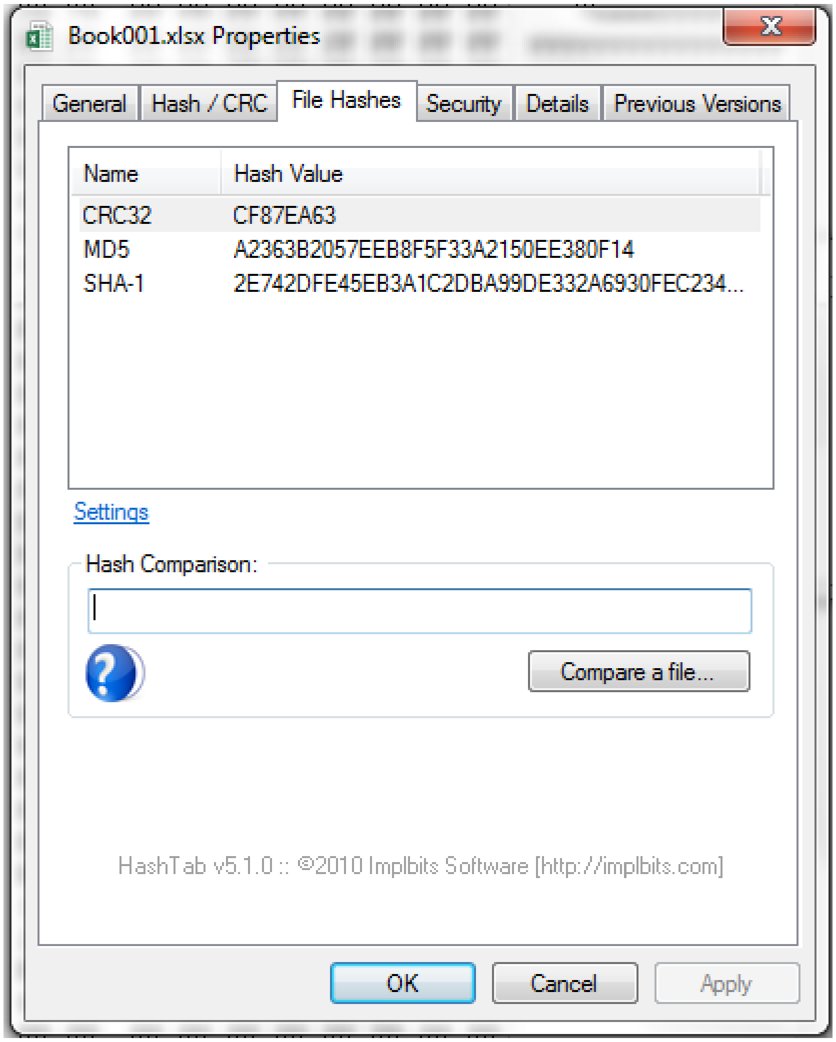
Advanced Hash Calculator



Febooti HASH-CRC



HashTab



Book001.xlsx Properties
General
Hash / CRC
File Hashes
Security
Details
Previous Versions
Name
Hash Value
CRC32
CF87EA63
MD5
A2363B2057EEB8F5F33A2150EE380F14
SHA-1
2E742DFE45EB3A1C2DBA99DE332A6930FEC234...
Settings
Hash Comparison:
Compare a file...
HashTab v5.1.0 :: ©2010 Implbits Software [http://implbits.com]
OK
Cancel
Apply

Hash Generator

MATLab Script

```
clear

% cd directory
cd J:\Thesis\Test_Documents\
%cd J:\Thesis\Test_Documents\Excel_Files\Excel_Program_HASHes\SHA1
% cd

%Get root file
[ExcelRoot, Root] = uigetfile ('*.xlsx', 'Select Root Excel File');
addpath (Root);

%Get comparison file
[ExcelCompare, Compare] = uigetfile('*.xlsx', 'Select Comparison Excel
File');
addpath (Compare)

% read the Excel files

%[sean01,txt01]=xlsread('Checksums.xlt');
%[sean02,txt02]=xlsread('HASH_CRC.xlt');

%[sean01,txt01]=xlsread('Excel File MD5 HASH FTK.xls');
%[sean02,txt02]=xlsread('Excel File MD5 HASH WinHex.xls');

%MD5
%[sean01,txt01]=xlsread('Excel File MD5 WinHex');
%[sean02,txt02]=xlsread(ExcelCompare);

%SHA1
[sean01,txt01]=xlsread(ExcelRoot);
[sean02,txt02]=xlsread(ExcelCompare);

% compute the tables lines and columns numbers
[a01,b01]=size(txt01);
[a02,b02]=size(txt01);

% compare the HASH values of the Test1...Test3 columns
% txt01
for k1=2:a01
    c11(k1-1)=strcmpi(txt01(k1,2),txt01(k1,3));
    c12(k1-1)=strcmpi(txt01(k1,2),txt01(k1,4));
    c13(k1-1)=strcmpi(txt01(k1,3),txt01(k1,4));
end

if prod(c11(:))==1
    disp('All the c11 HASH values match.')
elseif prod(c11(:))==0
    disp('Check the c11 HASH differences.')
end

if prod(c12(:))==1
    disp('All the c12 HASH values match.')
```

```
elseif prod(c12(:))==0
    disp('Check the c12 HASH differences.')
end

if prod(c13(:))==1
    disp('All the c13 HASH values match.')
elseif prod(c13(:))==0
    disp('Check the c13 HASH differences.')
end

% txt02
for k1=2:a02
    c21(k1-1)=strcmpi(txt02(k1,2),txt02(k1,3));
    c22(k1-1)=strcmpi(txt02(k1,2),txt02(k1,4));
    c23(k1-1)=strcmpi(txt02(k1,3),txt02(k1,4));
end

if prod(c21(:))==1
    disp('All the c21 HASH values match.')
elseif prod(c21(:))==0
    disp('Check the c21 HASH differences.')
end

if prod(c22(:))==1
    disp('All the c22 HASH values match.')
elseif prod(c22(:))==0
    disp('Check the c22 HASH differences.')
end

if prod(c23(:))==1
    disp('All the c23 HASH values match.')
elseif prod(c23(:))==0
    disp('Check the c23 HASH differences.')
end

% compare the HASH values of two different Excel files
for k1=2:min(a01,a02)
    for k2=2:min(b01,b02)
        c1(k1-1,k2-1)=strcmpi(txt01(k1,k2),txt02(k1,k2));
    end
end

if prod(c1(:))==1
    disp('All the HASH values match.')
elseif prod(c1(:))==0
    disp('Check the HASH differences.')
end
```