

How do hashing algorithms in specific
SHA256, SHA512 and MD5 affect
Number of Passwords decrypted and
Computational Power required in a
given period?

Subject: Computer Science

Topic: Encryption and Hashing Algorithms

Word Count: 3944

Table of Contents

1. Introduction	4
1.1 Hash Functions	4
1.2 Properties and Features of Cryptographic Hash Functions.....	5
1.3 Importance of Cryptographic Hash Functions	6
2. Methodology	7
2.1 DevOps	7
2.2 Hashcat.....	8
2.3 Attack Modes	9
2.3.1 Dictionary Attack.....	9
2.3.2 Other Attacks Modes	10
2.4 Testing Parameters.....	10
2.4.1 Independent Variables.....	10
2.4.2 Dependent Variables	15
3. Testing	17
3.1. Probable (12645) Password List	19
3.1.1. Period: 120 Seconds	20
3.1.2. Period: 300 Seconds	23
3.1.3 Analysing Data.....	25
3.2 John (3106) Password List.....	25
3.2.1 Period: 120 Seconds	26
3.2.2 Period: 300 Seconds	28
3.2.3 Analysing Data.....	29

4. Analysis	30
5. Conclusion.....	33
6. Bibliography	34
7. Appendix.....	37
7.1 Custom Programs	37
7.2 Extra Attack Modes.....	40
7.3 Data Sets	43
7.4 Raw Data	44
7.5 Other References.....	45

1.Introduction

In today's world, a lot of our time is spent on the internet using web services where we create and use passwords every day. For the user, a password is the first line of defence between their private data and its unwanted exposure. This protection has to be maintained by protecting and securing the password itself. Passwords have come to be protected by cryptographic hash functions¹, and the security of one's password and therefore the safety of one's data depends on these hash functions. For this reason, the research question " How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Time taken and Computational Power required to decrypt passwords?" will be investigated, to understand which hashing algorithm is most optimal for securing passwords.

1.1 Hash Functions

What exactly is a hash function? A hash function¹ is a function that can map data of any size to data of a predetermined size. The values outputted by hash functions are called hashes and are usually represented in hexadecimal (base 16). These functions have a variety of applications and use all over the field of computer science, but arguably the most important use is in the area of encryption. In encryption, an extension of hash functions called cryptographic hash functions¹ is used. These functions are unique as they are mathematically designed to be one-way operations, as in it is infeasible to take the output hash and retrieve the input data.

¹ Paar, Christof; Pelzl, Jan (2009). "11: Hash Functions". *Understanding Cryptography, A Textbook for Students and Practitioners*. Springer. Archived from the original on 2012-12-08.

1.2 Properties and Features of Cryptographic Hash Functions

A cryptographic hash function is a subset of hash function that has unique features and properties that make them suitable for cryptography². These functions are mathematically designed to be one-way functions.

An ideal cryptographic hash function has five distinct features³; first, it is deterministic, i.e. It will always produce the exact output for a specific input. Second, any given message can be computed into a hash reasonably quickly. Third, any change however insignificant it may seem will create a hash so different from the original hash, that they seem entirely unrelated. Fourth, it is infeasible to retrieve the input from an output hash except for trying all possible combinations. Fifth, it is impossible that two different data have the same hash value.

These functions must be able to withstand all known types of cryptanalytic attack. The following properties are used to determine the security level of a given hash function.

Pre-image resistance³: For any given hash h it should be challenging to find a message m such that $h = \text{hash}(m)$. This property relates to the one-way nature of hashing algorithms.

² Sharma, R. (2018). *Cryptographic Hash Functions*. [online] Investopedia. Available at: <https://www.investopedia.com/news/cryptographic-hash-functions/> [Accessed 21 June. 2018].

³ Rjařsko, M. (2008). Properties of Cryptographic Hash Functions. *Comenius University in Bratislava Faculty of Mathematics, Physics and Informatics Department of Computer Science*, pp.11-32.

Collision resistance³: Given any two different messages m_1 and m_2 it should be tough to find $\text{hash}(m_1) = \text{hash}(m_2)$. Such a pair of hashes would be called a cryptographic hash collision. Secure cryptographic hash functions are expected to have strong collision resistance.

1.3 Importance of Cryptographic Hash Functions

Modern computer science would not be able to function without the use of the cryptographic hash function. Their unique set of features and properties enable computer scientists to use them in a multitude of different and unique ways, some of these ways include verifying the integrity of files or message, password validation and storage, pseudo-random generation⁴.

As these functions have become integral to a variety of fields in computer science, it is essential to understand which algorithms are designed for what purpose and which hashing algorithm needs to be used to for a given task. This paper will try addressing this question for the specific field of password storage.

⁴ Halevi, S. (2009). *Cryptographic Hash Functions and their many applications*.

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

2. Methodology

2.1 DevOps

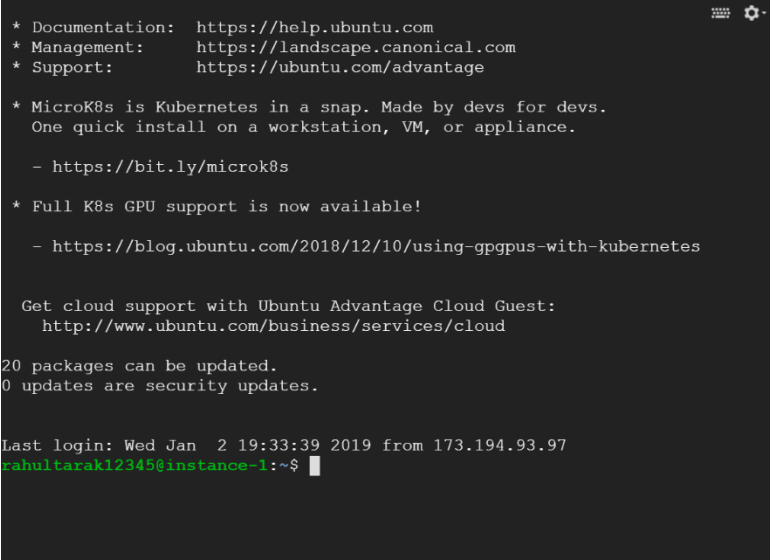
As stated above, cryptographic hash algorithms are designed such that it is infeasible to crack² them, which makes the process of comparing hashing algorithms complicated. While there are a few approaches that have come up over the years, they all require a lot of computational power and time³. While computational power used to be somewhat limited and very expensive, these days with cloud services like AWS and Google Cloud, they are much more accessible and relatively cheap.

For this investigation, a virtual machine (VM) instance has been set up on google cloud platform's compute engine⁵. This VM instance has been configured with quad-core server grade CPU, 24 Gbs of Memory and most importantly an Nvidia Tesla K80, a server grade graphics card with up to 8.73 teraflops⁶ of power and 24 Gigabytes of Video Memory. The instance has been configured with Ubuntu 17.10, as it is more optimal for the given task.

⁵ Google Cloud. (2018). *Compute Engine - IaaS | Compute Engine | Google Cloud*. [online] Available at: <https://cloud.google.com/compute/> [Accessed 3 Jun. 2018].

⁶ NVIDIA. (2018). *NVIDIA Tesla Product Literature*. [online] Available at: <https://www.nvidia.com/en-us/data-center/tesla-k80/> [Accessed 11 Aug. 2018].

The VM instance is accessed using a protocol called SSH (Secure Shell)⁷, which allows users to login and access other systems remote. This protocol allows the investigation to be conducted on the cloud server without any hinderances.



```
* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

* MicroK8s is Kubernetes in a snap. Made by devs for devs.
  One quick install on a workstation, VM, or appliance.
  - https://bit.ly/microk8s

* Full K8s GPU support is now available!
  - https://blog.ubuntu.com/2018/12/10/using-gpgpus-with-kubernetes

Get cloud support with Ubuntu Advantage Cloud Guest:
  http://www.ubuntu.com/business/services/cloud

20 packages can be updated.
0 updates are security updates.

Last login: Wed Jan  2 19:33:39 2019 from 173.194.93.97
rahultarak12345@instance-1:~$
```

Figure 1 : SSH Client Window

2.2 Hashcat⁸

Hashcat proclaims itself to be the world's fastest password recovery tool. Hashcat can crack hashes from various hashing algorithms including MD5, SHA256, SHA512 and many more.

A version of hashcat, an open source password recovery tool, called Cuda-hashcat was installed on the virtual machine (VM). The Cuda (Compute Unified Device Architecture)⁹ version was chosen as it can make full of Tesla K80 by using the graphics card to compute hashes. Hashcat as a software offers a variety of options and features perfect for this project.

⁷ Ssh.com. (2018). *SSH Protocol – Secure Remote Login and File Transfer | SSH.COM*. [online] Available at: <https://www.ssh.com/ssh/protocol/> [Accessed 21 Aug. 2018].

⁸ Hashcat.net. (2018). *hashcat - advanced password recovery*. [online] Available at: <https://hashcat.net/hashcat/> [Accessed 17 May 2018].

⁹ Nvidia.in. (2018). *CUDA Parallel Computing Platform for Developers | NVIDIA*. [online] Available at: <https://www.nvidia.in/object/cuda-parallel-computing-in.html> [Accessed 17 Sep. 2018].

2.3 Attack Modes¹⁰

There are many approaches to attack hashes to break them; each attack mode has its advantages and disadvantages. Hashcat supports five different attack modes

```
- [ Attack Modes ] -

# | Mode
---+-----
0 | Straight
1 | Combination
3 | Brute-force
6 | Hybrid Wordlist + Mask
7 | Hybrid Mask + Wordlist
```

Figure 2 : HashCat Attack Modes

(each attack mode is represented with corresponding attack mode number in hashcat). This investigation will only be using Dictionary Attack and hence not discussing the other modes.

```
hashcat -a 0 hashFile.hash dictionary.txt10
```

2.3.1 Dictionary Attack¹¹

First is a dictionary attack (called straight in hashcat) or attack mode 0, the above command represents this mode. This mode hashes a premade dictionary of words and comparing with the set of hashes. In this primary state, this mode is not very good at cracking passwords, but this can be improved by using a rule set.

```
hashcat -a 0 hashFile.hash -r testRules.rule10
```

¹⁰ Hashcat.net. (2018). hashcat [hashcat wiki]. [online] Available at: <https://hashcat.net/wiki/doku.php?id=hashcat> [Accessed 11 Jul. 2018].

¹¹ Hashcat.net. (2018). *dictionary_attack* [hashcat wiki]. [online] Available at: https://hashcat.net/wiki/doku.php?id=dictionary_attack [Accessed 12 Jun. 2018].

Above code snippet represents how to use the rules set. This rule set makes small modifications to the words in the dictionary, like changing o to 0.

2.3.2 Other Attacks Modes

Hashcat also has four more attack modes which will be described in the appendix

2.4 Testing Parameters

The investigation will be conducted on three hashing algorithms, SHA-256, SHA-512 and MD5. These are a few popular hashing algorithms used by many companies for storing and securing passwords.

For each algorithm, the test will be conducted with the below independent and dependent variables.

2.4.1 Independent Variables

2.4.1.1 Time Provided

All Passwords can theoretically be cracked given correct enough time¹², but some passwords would take so long to be cracked that it is infeasible and unpractical to try to crack them.

Hence, all hashing algorithms will be compared within the same periods

¹² Paar, Christof; Pelzl, Jan (2009). "11: Hash Functions". *Understanding Cryptography, A Textbook for Students and Practitioners*. Springer. Archived from the original on 2012-12-08.

Period given to attempt to crack the passwords is the most significant independent variable.

This investigation will use two periods 120 seconds and 300 seconds.

2.4.1.2 Entropy and Data Sets

Password entropy is basically how random a password is; it is based on a variety of factors mainly the length of the password and the character used in the password.¹³

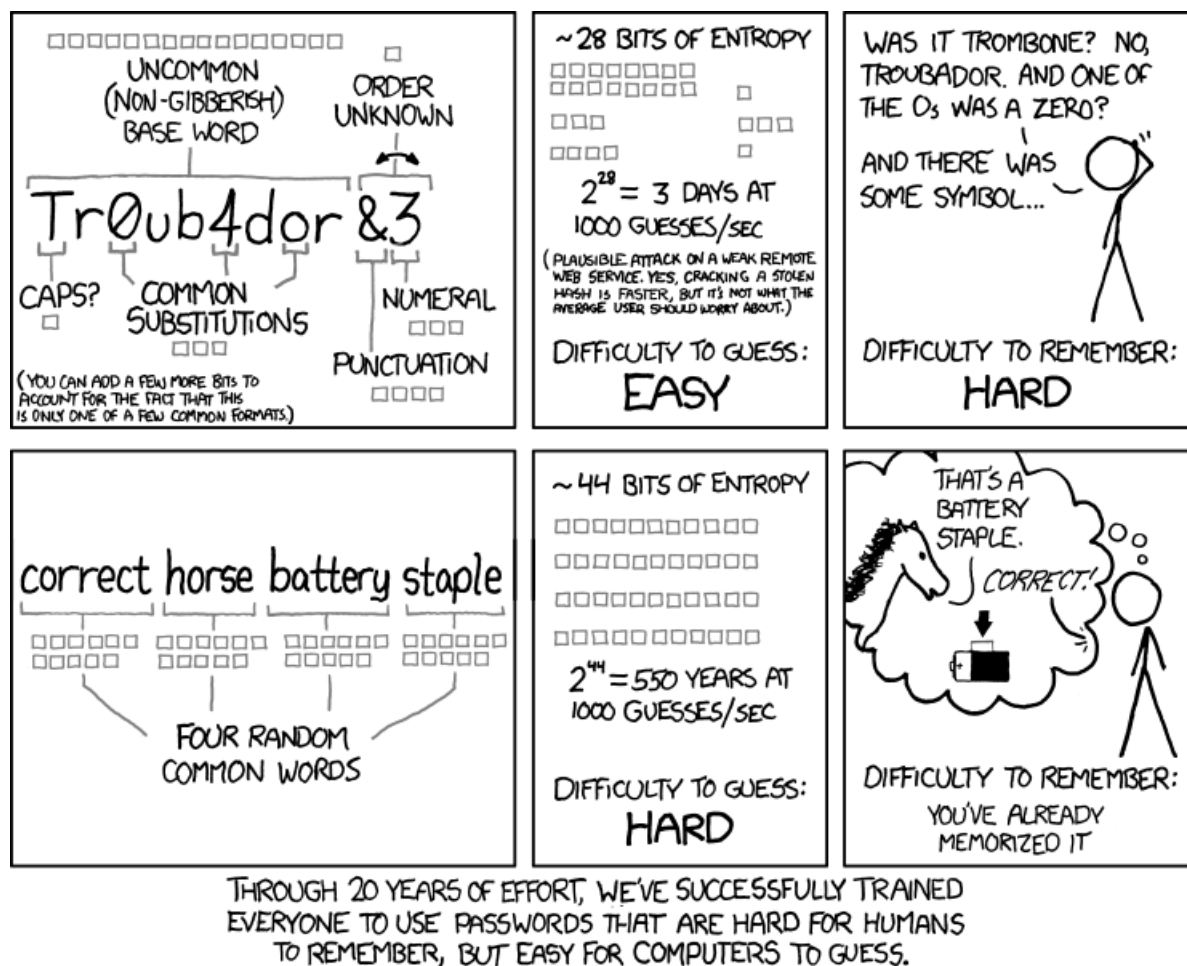


Figure 3: Xkcd Password Strength

¹⁴The above image is a visual representation of entropy and passwords strength made by an online creator called xkcd¹⁴.

¹³ WhatIs.com. (2018). *What is password entropy? - Definition from WhatIs.com*. [online] Available at: <https://whatIs.techtarget.com/definition/password-entropy> [Accessed 21 Nov. 2018].

¹⁴ Xkcd.com. (2018). *xkcd: Password Strength*. [online] Available at: <https://xkcd.com/936/> [Accessed 21 Nov. 2018].

This investigation will be using different data sets with many different types of passwords with varying password entropy. Hence, it is essential to understand the average password entropy of the data set, to be able to distinguish the variance in results.

To calculate password entropy, this investigation will be using Dropbox's low budget password strength estimation tool called zxcvbn¹⁵. The python library for this tool python-zxcvbn¹⁶ will be used with a program that can be found in the appendix to calculate the average entropy of a data set.

```
result.score      # Integer from 0-4 (useful for implementing a strength bar)

0 # too guessable: risky password. (guesses < 10^3)

1 # very guessable: protection from throttled online attacks. (guesses < 10^6)

2 # somewhat guessable: protection from unthrottled online attacks. (guesses <
10^8)

3 # safely unguessable: moderate protection from offline slow-hash scenario.
(guesses < 10^10)

4 # very unguessable: strong protection from offline slow-hash scenario. (guesses
>= 10^10)
```

Figure 4: Dropbox Zxcvbn Documentation on Password Entropy¹⁷

¹⁵ Lowe Wheeler, D. (2016). zxcvbn: Low-Budget Password Strength Estimation. *This paper is included in the Proceedings of the 25th USENIX Security Symposium*. [online] Available at: https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_wheeler.pdf [Accessed 10 Nov. 2018].

¹⁶ GitHub. (2018). *dwolfhub/zxcvbn-python*. [online] Available at: <https://github.com/dwolfhub/zxcvbn-python> [Accessed 12 Nov. 2018].

¹⁷ GitHub. (2018). *dropbox/zxcvbn*. [online] Available at: <https://github.com/dropbox/zxcvbn> [Accessed 13 Nov. 2018].

Above described is the range of scores provided by zxcvbn, each password within the dataset will be assigned a score using zxcvbn, and then all the scores will be averaged out to get the mean entropy of the set. It is expected that this mean entropy is on the lower side as many passwords will have a score of only 0 or 1.

The python program below uses zxcvbn to calculate the average entropy of the dataset

```

from zxcvbn import zxcvbn
results = []
fileInput = input("Enter File Path : ")
unencrypted = open(fileInput, "r")
words = []
for elem in unencrypted:
    elem = elem.split("\n")
    words.append(elem[0])
unencrypted.close()
for elem in words:
    result = zxcvbn(elem)
    results.append(result)
scores = results['score']
import statistics
mean = statistics.mean(scores)

```

Figure 5: Self-Generated Code Snippet to Calculate Average Entropy of Dataset

2.4.1.3 Other Hashcat Properties¹⁸

Apart, from the attack modes, hashcat also has a long list of other properties and features which can impact the time taken to crack passwords. A full list of hashcat features can be found in the Appendix.

-t, --markov-threshold	Num	Threshold X when to stop accepting new markov-chains	-t 50
--runtime	Num	Abort session after X seconds of runtime	--runtime=10
--session	Str	Define specific session name	--session=mysession
--restore		Restore session from --session	
--restore-disable		Do not write restore file	
--restore-file-path	File	Specific path to restore file	--restore-file-path=x.restore

Figure 6: Hashcat Extra Options

18

The above option `-t` allows hashcat to use Markov chains to predict the next character in the password thus making the password cracking process faster and more efficient. This command will be used in the investigation.

Another option `-w` sets the workload profile of hashcat, which impacts the time taken to break the hash but also impacts the temperature of the system.

```
- [ Workload Profiles ] -
```

#	Performance	Runtime	Power Consumption	Desktop Impact
1	Low	2 ms	Low	Minimal
2	Default	12 ms	Economic	Noticeable
3	High	96 ms	High	Unresponsive
4	Nightmare	480 ms	Insane	Headless

Figure 7: Hashcat Work Profiles

18

Above are all the workload profile hashcat profiles.

¹⁸ Hashcat.net. (2018). hashcat [hashcat wiki]. [online] Available at: <https://hashcat.net/wiki/doku.php?id=hashcat> [Accessed 11 Jul. 2018].

Another option `--potfile-disable` prevents hashcat from storing all the passwords already decrypted into a file. This option is essential to ensure that the test results for all algorithms are not skewed by previous attempts to crack the passwords.

2.4.2 Dependent Variables

2.4.2.1 Number of Passwords Cracked

The primary dependent variable is the number of hashes cracked in a given period.

This will vary drastically with each hashing algorithm, attack modes and other independent variables.

The more the number of passwords cracked in a given period the weaker the hashing algorithm is as keep passwords safe.

2.4.2.2 Computational Power

Another essential dependent variable is the hashing speed which is an indication of the computational power required. As hashing speed goes down, the computational power required increases.

Each hashing algorithm has a base hashing speed which depends on its quality and complexity which will be established using hashcat's inbuilt benchmark tool.

Computational Power can also be determined by the temperature of the system

Using the fact that Temperature is directly proportional to computational power and Speed of hashing given in Nano Seconds is proportional to computational power.

$$\text{Computational Power} \propto \frac{\text{Temperature}}{\text{Speed of Hashing per Nano Second}}$$

Speed is taken in Nanoseconds as a Nanosecond is defined as Time to execute one machine cycle by a 1 GHz microprocessor¹⁹

Using k a constant of proportionality to change the proportionality sign:

$$\text{Computational Power} = k \times \frac{\text{Temperature}}{\text{Speed of Hashing per Nano Second}}, \text{ where } k \text{ is a constant of propotionality}$$

$$\text{Therefore } \frac{\text{Computational Power}(CP)}{k} = \frac{\text{Temperature in Celius}}{\text{Speed of Hashing per Nano Second}}$$

¹⁹ "The Performance Equation." N.p., 2018. Web. 16 Nov. 2018.

3. Testing

While hashcat is great for cracking the password, the data collected while hashcat is executing needs to be collected and stored in a .CSV file. For this task, a python program was used to read the output from hashcat and store it in the . CSV file.

This program doing this task can be found below and in the appendix.

```
def is_number(num):
    try:
        float(num)
        return True
    except ValueError:
        return False

attack = input("Attack : ")
shFile = input("Location of shFile : ")
command = 'sh ' + shFile
attack = '../Data/' + attack + '.csv'

import pexpect

child = pexpect.spawn(command) # Spawns child application
child.sendline('s')
print("Python output")
child.timeout = 5000
output = child.read()
output = output.decode('utf-8')
print(str(output))

decoded_string = bytes(output, "utf-8").decode("unicode_escape")
newLines = []

for line in decoded_string.split('\n'):
    newLine = [x.strip() for x in line.split('\t')]
    fixedNewLine = []
    for elem in newLine:
        if elem != '':
            for i in elem.split(' '):
                if is_number(i):
                    fixedNewLine.append(i)
    newLines.append(fixedNewLine)
```

Figure 8: Self-Generated Python Code Snippet

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

All test will be conducted under the dictionary attack mode or `-a 0`.

All dictionary attacks will be using the `../rockyou.txt` dictionary/word list which a leaked distortionary of over 14 million actual passwords and therefore considered one of the best dictionaries for password cracking

These attack along with the rockyou dictionary will be using the rule set created by a developer with pseudonym Dive `-r ./rules/dive.rule`

Apart from the dictionary and rules, these attacks will also be using the below options:

- Markov Chains Threshold after which hashcat stop accepting new Markov-chains `-t 32`
- Maximum Workload Profile of 4: `-w 4`
- A potfile usually stores already cracked passwords, and hence it is disabled `--potfile-disable`
- Quiet Mode which prevents unnecessary outputs `--quiet`
- Status and Status timer which allow the status to update at a given interval which allows the python program to read them `--status-timer=1 -status`
- Machine Readable Output, which makes the output more accessible for a program to parse and store in a CSV file `--machine-readable`

The final command executed for the dictionary attacks is shown below where (pathToHashFile) is the relative path from hashcat to where the hash file is stored and

--runtime=(GivenRuntime) represents the execution time of the program in seconds.

```
hashcat -t 32 -a 0 (pathToHashFile) ../rockyou.txt -r
./rules/dive.rule -w 4 --potfile-disable --quiet --status-timer=1 --
status --runtime=(GivenRuntime) --machine-readable
```

There will be a total of four different tests per hashing algorithm using the dictionary attack which involved two different password lists and two different periods.

For both these password lists, all three-hashing algorithm will execute the above command for both 120s and 300s.

All raw data files can be found in the appendix.

3.1. Probable (12645) Password List

This is a password list acquired from an online security testing repository called SecList²⁰; this dataset contains 12645 unique passwords ranging in password strength score from 0 to a score of 3. This dataset got a mean password strength score of **0.6671411625148**.

²⁰ GitHub. (2018). *danielmiessler/SecLists*. [online] Available at: <https://github.com/danielmiessler/SecLists> [Accessed 15 Aug. 2018].

3.1.1. Period: 120 Seconds

Hashing Algorithm	Passwords Found	Average Number of Hashes per Nano Second (Speed)	Average Temperature in Celsius
MD5	4205/12645	1.357028164	45.95
SHA256	3880/12645	0.421523743	45.875
SHA512	3677/12645	0.158746873	46.64957265

Table 1: Raw Data Probable Period 120 Seconds

Using the above data, $\frac{\text{Computational Power}}{k}$ can be calculated using the formula

$$\frac{\text{Computational Power}(CP)}{k} = \frac{\text{Temperature in Celsius}}{\text{Speed of Hashing per Nano Second}}$$

Therefore, the below values can be calculated

Hashing Algorithm	Cp/k
MD5	33.95585144
SHA256	108.1860335
SHA512	291.8746364

Table 2: Calculated Computational Power

From the above data, we can calculate the computational power of each algorithm proportional to each other.

$$\frac{\text{Computational Power}_{SHA256}}{k} / \frac{\text{Computational Power}_{MD5}}{k}$$

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

Where both k values cancel and we get Computational Power of SHA256 proportional to Computational Power of MD5

Similarly, the Computational Power of SHA512 proportional to MD5 and SHA512 proportional to SHA256 can be calculated

Percentage of Extra Passwords Cracked

$$= \frac{\text{Difference of Passwords Cracked Between Algorithm}}{\text{Total Number of Passwords}} \times 100$$

Hashing Algorithms	Relative Cp	Percentage of Extra Passwords cracked
SHA256/MD5	3.186079246	$(325/12645) \times 100$
SHA512/MD5	8.595709546	$(528/12645) \times 100$
SHA512/SHA256	2.697895716	$(203/12645) \times 100$

Table 3: Relative Computational Power

This data can be interpreted as SHA256 needs 310% more computational power when compared to MD5, while SHA512 needs 859% more computational power when compared to MD5. Finally, SHA512 needs 269% more computational power when compared to SHA256.

Furthermore, the relationship between computational power and the difference in the number of passwords cracked can be examined.

Graphing Relative Cp on X-Axis and Difference on Y-Axis

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

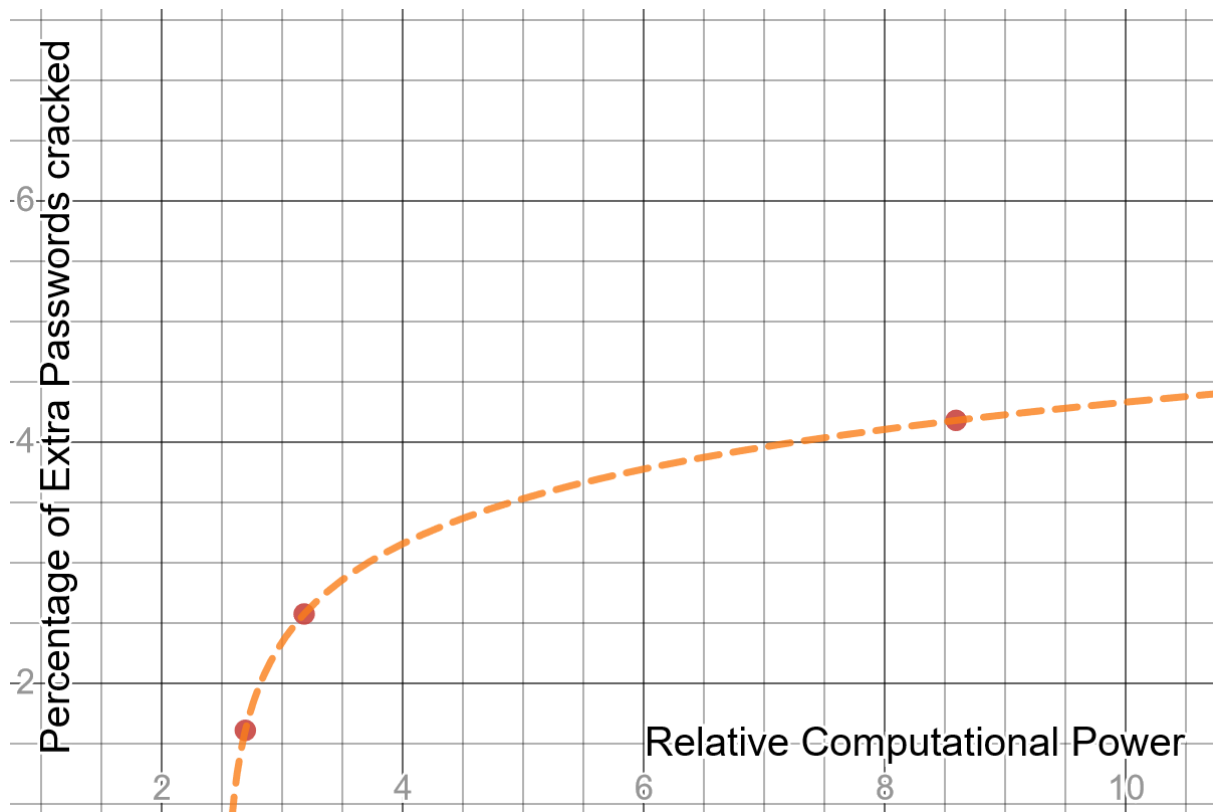


Figure 9: Plotted Relationship between Relative Computational Power Vs Percentage of Extra Passwords Cracked

Using regression, a logarithmic relationship is established between Relative Cp and Difference in Passwords Cracked where

$$y \sim a \log(x - b) + c,$$

where $a = 1.66962$, $b = 2.86755$, $c = 2.86755$, with a R^2 score of 1

3.1.2. Period: 300 Seconds

While the data is different for 300 second period, the analysis will follow the same method as for the 120 second period, and hence all the steps will not be repeated.

Hashing Algorithm	Passwords Found	Average Number of Hashes per Nano Second (Speed)	Average Temperature in Celsius
MD5	4680/12645	1.283145992	73.49236641
SHA256	4219/12645	0.398616051	73.69202899
SHA512	3817/12645	0.152001525	73.73381295

Table 4: Raw Data Probable Period 300 Seconds

Using the same method used for 120 second period Cp/k is calculated

Hashing Algorithm	Cp/k
MD5	57.29701177
SHA256	184.8956681
SHA512	485.139793

Table 5: Calculated Computational Power

From this data, Relative Computational Power can be calculated

Hashing Algorithms	Relative Cp	Percentage of Extra Passwords cracked
SHA256/MD5	3.226968778	$(461/12645) \times 100$
SHA512/MD5	8.467104618	$(863/12645) \times 100$
SHA512/SHA256	2.623857	$(402/12645) \times 100$

Table 6: Relative Computational Power

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

This data can be interpreted as SHA256 needs 322% more computational power when compared to MD5, while SHA512 needs 846% more computational power when compared to MD5. Finally, SHA512 needs 262% more computational power when compared to SHA2256.

Similar to the 120s second period, the relationship between Relative Cp and Difference can be examined by graphing Relative Cp on the X-Axis and Difference on the Y-Axis

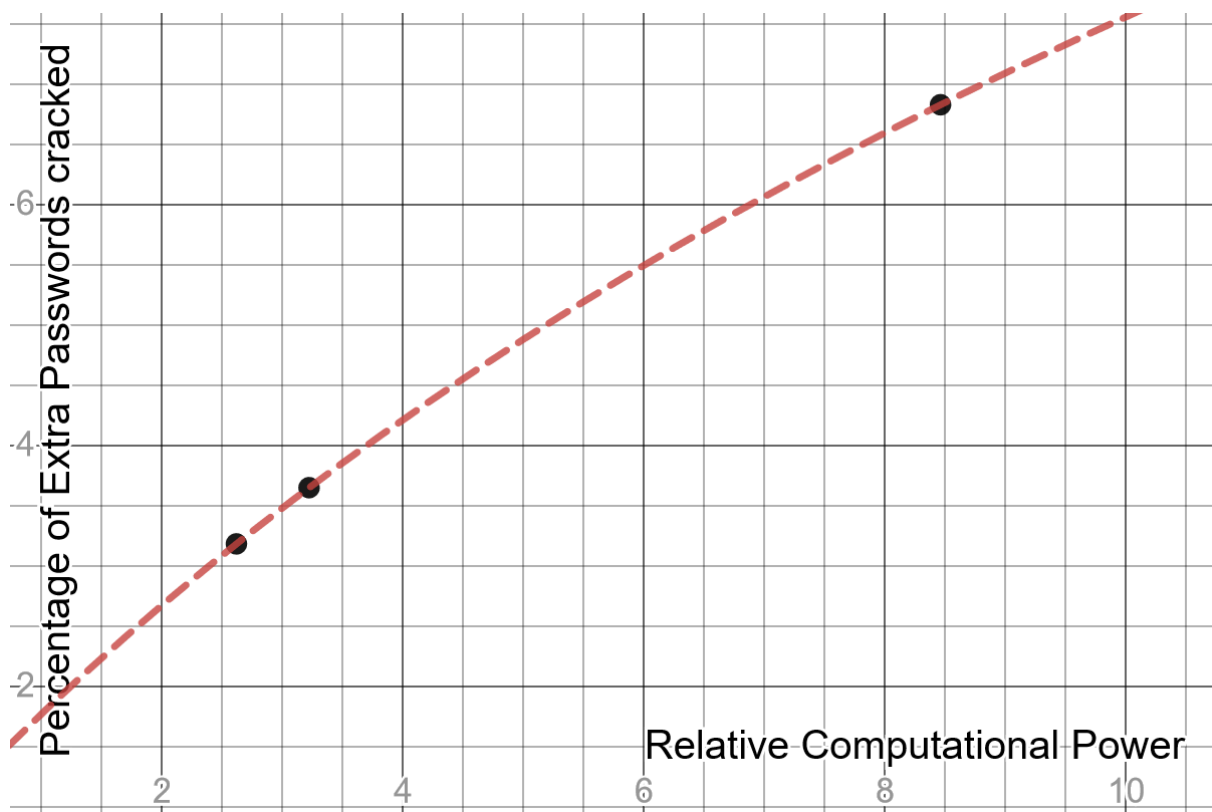


Figure 10: Plotted Relationship between Relative Computational Power Vs Percentage of Extra Passwords Cracked

Similar to the 120s period, a logarithmic relationship is found

$$y \sim a \log(x - b) + c,$$

where $a = 17.7716$, $b = -7.05409$, $c = -14.3399$, with a R^2 score of 1

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

3.1.3 Analysing Data

The data from the above two tests shows promising results, establishing logarithmic relationships, showcasing trends of relative computational power between hashing algorithms.

Hashing Algorithm	Number of Passwords cracked in 120s	Number of Passwords cracked in 300s
MD5	4205	4680
SHA256	3880	4219
SHA512	3677	3817

Table 7: Number of Passwords Cracked Probable

From the above data, the difference in the number of passwords cracked between each algorithm is explicit, MD5 has the most passwords cracked in either period proving to be the weakest of the set, thus far. While SHA512 has the least passwords cracked proving to be superior to the other hashing algorithms.

3.2 John (3106) Password List

This password list contains 3106 unique passwords, with password strengths scores ranging from 0 to 3. This list has a mean password strength score of **0.437862202189311**.

While this is a different password list and hence will produce different results the method of analysis will be the same as it was for the Probable List and hence will not be repeated in this section.

3.2.1 Period: 120 Seconds

Hashing Algorithm	Passwords Found	Average Number of Hashes per Nano Second (Speed)	Average Temperature in Celsius
MD5	1409/3106	1.372335188	45.74257426
SHA256	1245/3106	0.422704219	46.20175439
SHA512	1142/3106	0.159295941	46.28070175

Table 8: Raw Data John 120 Seconds

Cp/k Is calculated using above data:

Hashing Algorithm	Cp/k
MD5	33.4415779
SHA256	109.4119483
SHA512	290.648032

Table 9: Calculated Computational Power

Now, Relative Cp is calculated

Hashing Algorithms	Relative Cp	Percentage of Extra Passwords cracked
SHA256/MD5	3.271734026	$(164/3106) \times 100$
SHA512/MD5	8.691217649	$(267/3106) \times 100$
SHA512/SHA256	2.656456051	$(103/3106) \times 100$

Table 10: Relative Computational Power

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

Similar to the data from Probable Password List, a relationship between Relative Cp and Difference can be identified by graphing Relative Cp on X-Axis and Difference on Y-Axis

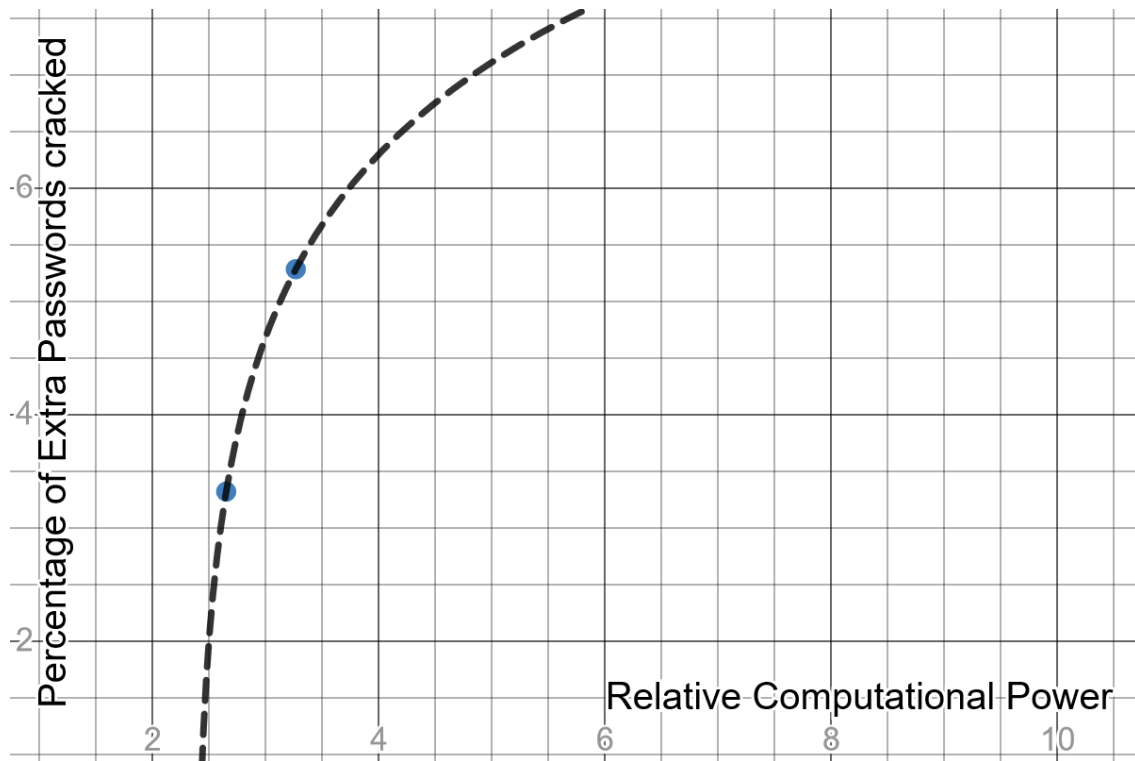


Figure 11: Plotted Relationship between Relative Computational Power Vs Percentage of Extra Passwords Cracked

Here also a logarithmic relationship is found between Relative Cp and Difference

$$y \sim a \log(x - b) + c,$$

where $a = 3.91427$, $b = 2.37356$, $c = 5.46267$, with a R^2 score of 1

3.2.2 Period: 300 Seconds

Hashing Algorithm	Passwords Found	Average Number of Hashes per Nano Second (Speed)	Average Temperature in Celsius
MD5	1576/3106	1.295241411	73.70930233
SHA256	1390/3106	0.39981739	73.54151625
SHA512	1234/3106	0.152377785	73.725

Table 11: Raw Data John 300 Seconds

Cp/k for 300 Second Period:

Hashing Algorithm	Cp/k
MD5	56.94170959
SHA256	183.9617015
SHA512	483.8616971

Table 12: Calculated Computational Power

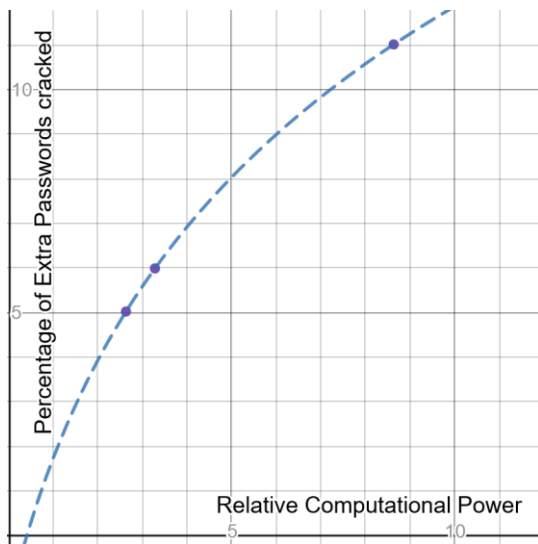
Relative Cp for 300 Seconds:

Hashing Algorithms	Relative Cp	Percentage of Extra Passwords cracked
SHA256/MD5	3.230702113	$(186/3106) \times 100$
SHA512/MD5	8.497491568	$(342/3106) \times 100$
SHA512/SHA256	2.630230604	$(156/3106) \times 100$

Table 13: Relative Computational Power

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

Similar to the last three tests, a relationship between Relative Cp and Difference can be examined by graphing Relative Cp on X-Axis and Difference on Y-Axis



The above relationship is also a logarithmic relationship:

$$y \sim a \log(x - b) + c,$$

$$\text{where } a = 15.6893, b = -1.63063,$$

$$c = -4.85603, \text{ with a } R^2 \text{ score of } 1$$

Figure 12: Plotted Relationship between Relative Computational Power Vs Percentage of Extra Passwords Cracked

3.2.3 Analysing Data

The data from the above two test follow a similar trend to the data from the Probable tests which proves consistency in the data. Hence making the results more reliable.

Hashing Algorithm	Number of Passwords cracked in 120s	Number of Passwords cracked in 300s
MD5	1409	1576
SHA256	1245	1390
SHA512	1142	1234

Table 14: Number of Passwords Cracked John

The above data follows a trend to the data from the Probable test where MD5 has the most passwords cracked in either period, furthering the hypothesis that MD5 is the

weakest of the hashing algorithms being tested. While SHA512 is proving the hypothesis that it is the superior hashing algorithm by having the least hashes cracked in either period.

4. Analysis

Using all the data from the above four tests conducted, conclusions on the effect of SHA512, SHA256 and MD5 on Number of Passwords cracked and Computational Power required were drawn.

Hashing Algorithm	Percentage of John's Password List Completed in 120s	Percentage of John's Password List Completed in 300s	Percentage of Probable Password List Completed in 120s	Percentage of Probable Password List Completed in 300s
MD5	45.3638%	50.7405%	33.2542%	37.0106%
SHA256	40.0837%	44.7520%	30.0514%	33.3649%
SHA512	36.7675%	39.7295%	29.0786%	30.1858%

Table 15: Compiled Data from All the above tests

From both the number of Passwords cracked, it is clear without a doubt that MD5 as a hashing algorithm is the worst of the three. While SHA256 is between SHA512 and MD5 performing significantly better than MD5 but significantly worse than SHA512, finally, SHA512 performs the best of all three with even allow up to 11% less password to be cracked in the same period as MD5.

The difference between the percentage of dataset cracked in the same periods can be explained once the difference in average password strength of both sets and their sizes are factored. Probable data set is a more complex data set with average password strength of **0.667141162514828** while John data set has an average password strength of **0.437862202189311**. The probable dataset is also almost four times as big as the John data set. These are two factors that can help explain the up to 13% difference in the completion of data sets between the John and Probable data sets.

Hashing Algorithms	Average Relative Cp	Average Percentage Increase in Required Computation
SHA512-SHA256	2.64624516	264.624516%
SHA256-MD5	3.241375216	324.1375216%
SHA512-MD5	8.596904312	859.6904312%

Table 16: Average Relative Computational Power

Now looking at relative computational power, again it is clear MD5 requires the least computational power and hence the weakest of the three hashing algorithms. While SHA256 again is in the middle required around 300% more computational power than MD5 but around 250% less than SHA512. Finally, SHA512 demands the most computational power which makes it the most secure out of the three hashing algorithms, demanding on average 860% more computational power when compared to MD5.

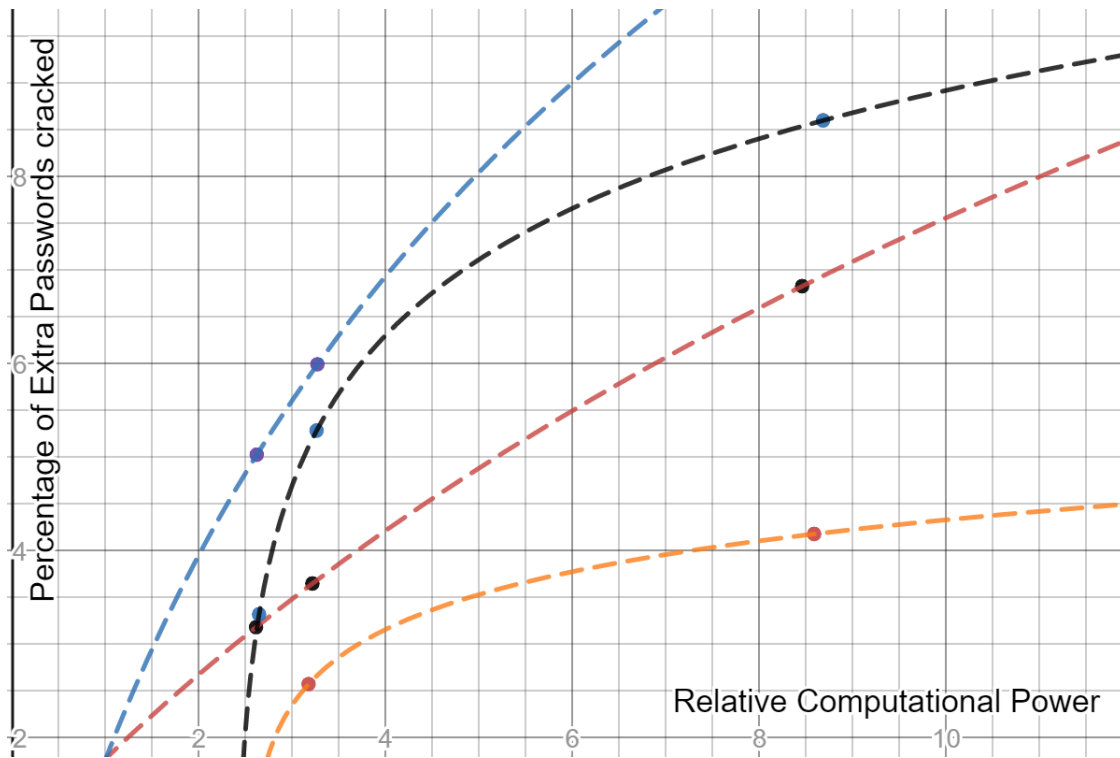


Figure 13: Plotted Relationships from all the above-conducted Tests

The above-established relationships between relative computational power and percentage of extra passwords cracked showcases how the percentage of new password cracks increases with the increase in relative computational power. Which means for two hashing algorithms A and B, if relative computational power between B and A increases then more passwords hashed with algorithm A will be cracked when compared to password hashed with algorithm B.

While using the data from this investigation, a clear function between Relative Computational Power and Extra Percentage of Password Cracked cannot be established it is clear that they follow a proportional relationship.

$$\text{Relative Computational Power} \propto \text{Extra Percentage of Password Cracked}$$

5. Conclusion

The investigation is able to present the variance in the number of passwords cracked and computational power required for cracking the same password sets with different hashing algorithms.

Although the data showcases SHA-512 as the most secure of the tested algorithms, which was the initially hypothesised, not a wide enough range of tests were conducted to say that SHA-512 will always be better and more secure than the other algorithms. Furthermore, the investigation is able to showcase the effects of different hashing algorithms on the overall safety of applications, highlighting the importance of choosing an appropriate hashing algorithm. Taking the investigation forward, more algorithms and more test could be conducted to gain a more accurate result.

6. Bibliography

1. Paar, Christof; Pelzl, Jan (2009). "11: Hash Functions". *Understanding Cryptography, A Textbook for Students and Practitioners*. Springer. Archived from the original on 2012-12-08.
2. Sharma, R. (2018). *Cryptographic Hash Functions*. [online] Investopedia. Available at: <https://www.investopedia.com/news/cryptographic-hash-functions/> [Accessed 21 June. 2018].
3. Rjaňsko, M. (2008). Properties of Cryptographic Hash Functions. *Comenius University in Bratislava Faculty of Mathematics, Physics and Informatics Department of Computer Science*, pp.11-32.
4. Halevi, S. (2009). *Cryptographic Hash Functions and their many applications*.
5. Google Cloud. (2018). *Compute Engine - IaaS | Compute Engine | Google Cloud*. [online] Available at <https://cloud.google.com/compute/> [Accessed 3 Jun. 2018].
6. NVIDIA. (2018). *NVIDIA Tesla Product Literature*. [online] Available at: <https://www.nvidia.com/en-us/data-center/tesla-k80/> [Accessed 11 Aug. 2018].

7. Ssh.com. (2018). *SSH Protocol – Secure Remote Login and File Transfer | SSH.COM*. [online] Available at: <https://www.ssh.com/ssh/protocol/> [Accessed 21 Aug. 2018].
8. Hashcat.net. (2018). *hashcat - advanced password recovery*. [online] Available at: <https://hashcat.net/hashcat/> [Accessed 17 May 2018].
9. Nvidia.in. (2018). *CUDA Parallel Computing Platform for Developers | NVIDIA*. [online] Available at: <https://www.nvidia.in/object/cuda-parallel-computing-in.html> [Accessed 17 Sep. 2018].
10. Hashcat.net. (2018). *hashcat [hashcat wiki]*. [online] Available at: <https://hashcat.net/wiki/doku.php?id=hashcat> [Accessed 11 Jul. 2018].
11. Hashcat.net. (2018). *dictionary_attack [hashcat wiki]*. [online] Available at: https://hashcat.net/wiki/doku.php?id=dictionary_attack [Accessed 12 Jun. 2018].
12. Paar, Christof; Pelzl, Jan (2009). "11: Hash Functions". *Understanding Cryptography, A Textbook for Students and Practitioners*. Springer. Archived from the original on 2012-12-08.
13. WhatIs.com. (2018). *What is password entropy? - Definition from WhatIs.com*. [online] Available at: <https://whatis.techtarget.com/definition/password-entropy> [Accessed 21 Nov. 2018].

14. Xkcd.com. (2018). *xkcd: Password Strength*. [online] Available at:
<https://xkcd.com/936/> [Accessed 21 Nov. 2018].
15. Lowe Wheeler, D. (2016). *zxcvbn: Low-Budget Password Strength Estimation. This paper is included in the Proceedings of the 25th USENIX Security Symposium*. [online] Available at:
https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_wheeler.pdf [Accessed 10 Nov. 2018].
16. GitHub. (2018). *dwolfhub/zxcvbn-python*. [online] Available at:
<https://github.com/dwolfhub/zxcvbn-python> [Accessed 12 Nov. 2018].
17. GitHub. (2018). *dropbox/zxcvbn*. [online] Available at:
<https://github.com/dropbox/zxcvbn> [Accessed 13 Nov. 2018].
18. Hashcat.net. (2018). *hashcat [hashcat wiki]*. [online] Available at:
<https://hashcat.net/wiki/doku.php?id=hashcat> [Accessed 11 Jul. 2018].
19. "The Performance Equation." N.p., 2018. Web. 16 Nov. 2018.
20. GitHub. (2018). *danielmiessler/SecLists*. [online] Available at:
<https://github.com/danielmiessler/SecLists> [Accessed 15 Aug. 2018].

7. Appendix

All documents and programs related to the extended essay can be found on a repository on GitHub <https://github.com/CryogenicPlanet/Extended-Essay>

7.1 Custom Programs

All the custom programs used for this investigation can be found here.

7.1.1 Testing Program

This was the program described in the testing section used to execute and save the data

```
def is_number(num):  
    try:  
        float(num)  
        return True  
    except ValueError:  
        return False  
  
attack = input("Attack : ")  
shFile = input("Location of shFile : ")  
command = 'sh ' + shFile  
attack = '../Data/' + attack + '.csv'  
  
import pexpect  
  
child = pexpect.spawn(command) # Spawns child application  
child.sendline('s')
```

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

```

print("Python output")
child.timeout = 5000
output = child.read()
output = output.decode('utf-8')
print(str(output))

decoded_string = bytes(output, "utf-8").decode("unicode_escape")

newLines = []
for line in decoded_string.split('\n'):
    newLine = [x.strip() for x in line.split('\t')]
    fixedNewLine = []
    for elem in newLine:
        if elem != '':
            for i in elem.split(' '):
                if is_number(i):
                    fixedNewLine.append(i)
    newLines.append(fixedNewLine)

with open(attack, 'w') as output_file:
    for elem in newLines:
        if elem:
            print(','.join(elem), file=output_file)

```

7.1.2 Mean Entropy Calculator

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

```

from zxcvbn import zxcvbn
results = []
fileInput = input("Enter File Path : ")
unencrypted = open(fileInput, "r")
words = []
for elem in unencrypted:
    elem = elem.split("\n")
    words.append(elem[0])
unencrypted.close()
for elem in words:
    result = zxcvbn(elem)
    results.append(result)
scores = results['score']
import statistics
mean = statistics.mean(scores)

```

7.1.3 Hasher

```

from passlib.hash import md5_crypt , sha1_crypt, sha256_crypt, sha512_crypt
import hashlib

fileInput = input("Enter File Path : ")
fileOutput = input("Enter Output File Path : ")
algo = input("Choose Algo: 1. MD5 2.SHA1 3. SHA256 4. SHA512 : ")
unencrypted = open(fileInput, "r")
encrypted = open(fileOutput, "w")
while True:
    line = unencrypted.readline()

```

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

```

if not line:
    break

if algo == "1":
    hash_object = hashlib.md5(line.encode()).hexdigest()
elif algo == "2":
    hash_object = hashlib.sha1(line.encode()).hexdigest()
elif algo == "3":
    hash_object = hashlib.sha256(line.encode()).hexdigest()
elif algo == "4":
    hash_object = hashlib.sha512(line.encode()).hexdigest()

encrypted.write(hash_object)
encrypted.write("\n")
encrypted.close()
unencrypted.close()

```

7.2 Extra Attack Modes

7.2.1 Brute Force Attack

The third attack mode is brute-force; this mode tries all possible combinations from a given character set. The number of characters tried and which character set can be determined while using the attack mode.

The below command uses a brute force attack for passwords of length seven with all lowercase characters.

```
hashcat -a 3 hashFile.hash ?l?l?l?l?l?l?l?l
```


Built-in charsets

- ?l = abcdefghijklmnopqrstuvwxyz
- ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
- ?d = 0123456789
- ?h = 0123456789abcdef
- ?H = 0123456789ABCDEF
- ?s = «space»!"#\$%&'()*+,-./:;<=>?@[\\]^_`{|}~
- ?a = ?l?u?d?s
- ?b = 0x00 - 0xff

Hashcat comes with the built-in with these character sets.

Using the above characters sets, the brute force mode can be used to guess passwords. The number of character sets chosen the size of the password guessed. This also determines the time taken to guess the password, as the time taken for brute force increases exponentially per character.

A brute force attack can be modelled using the function x^y where x is the number of characters in your character set and y is the length of the mask.

For example, a password of length seven with all lower case characters would be $26^7 = 8031810176$ which is still manageable, while a password of length 10 with both lowercase and uppercase characters would be $52^{10} = 1.4455511 * 10^{17}$ which would take significantly longer.

This becomes a significant problem as the mask length, and character set gets larger as the time taken grows exponentially.

This is why it is not practical to brute force passwords with more than eight characters. This method is also entirely inefficient concerning time and computational power. However, a brute-force attack is an essential part of hybrid attacks which will be used in this investigation.

7.1.2 Hybrid Attacks

Hashcat offers two hybrid attack modes, modes 6 and 7. These modes use a combination of both dictionary attacks and brute force attacks.

The commands below represent the two different hybrid attack modes in hashcat

```
hashcat -a 6 hashFile.hash dictionary.txt ?1?1?1?1
```

```
hashcat -a 7 hashFile.hash ?1?1?1?1 dictionary.txt
```

As these hybrid modes combine both dictionary and brute force attack modes, they are quite efficient at cracking passwords. These hybrid attack modes take the advantages of both attack modes through this attack mode could be more intensive on the graphics card.

This will be one of the main attack modes used for this investigation.

7.3 Data Sets

7.3.1 John Data Set

Only the first few words are showcased here; the full dataset can be found

<https://github.com/CryogenicPlanet/Extended-Essay/blob/master/EE/Hasher/john.txt>

```
3107 lines (3106 sloc) | 21.4 KB
Raw Blame History
1 12345
2 abc123
3 password
4 computer
5 123456
6 tigger
7 1234
8 a1b2c3
9 qwerty
10 123
11 xxx
12 money
13 test
14 carmen
15 mickey
16 secret
17 summer
18 internet
19 service
20 canada
21 hello
22 ranger
23 shadow
24 baseball
25 donald
26 harley
27 hockey
28 letmein
29 maggie
30 mike
31 mustang
32 snoopy
33 buster
34 dragon
35 jordan
```

7.3.2 Probable Data Set

The entire dataset can be found at <https://github.com/CryogenicPlanet/Extended-Essay/blob/master/EE/Hasher/12k.txt>

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

```

12646 lines (12645 sloc) | 97.9 KB
Raw Blame History
1 123456
2 password
3 123456789
4 12345678
5 12345
6 qwerty
7 123123
8 111111
9 abc123
10 1234567
11 dragon
12 1q2w3e4r
13 sunshine
14 654321
15 master
16 1234
17 football
18 1234567890
19 000000
20 computer
21 666666
22 superman
23 michael
24 internet
25 iloveyou
26 daniel
27 1qaz2wsx
28 monkey
29 shadow
30 jessica

```

7.4 Raw Data

Sample of the data use for this project will be showcased below, and all the data can be found <https://github.com/CryogenicPlanet/Extended-Essay/tree/master/EE/Data>

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
1	Time	Status	SPEED	Speed Nano S	Unit Time	EXEC_RUN CURKU	PROGRESS PROGRESS	REHASH	Total	RECSALT	Total	TEMP	REJECTED	Cp(nano)	Average Cp								
2	1.189	2	1629470	1.62947	1	267.5402	0	8.75E+08	1.42E+12	0	3106	0	36	2278978	22.09307	33.441578							
3	2.37711	2	1449778	1.449778	1	294.2904	0	2.62E+09	1.42E+12	1099	3106	0	1	37	2278978	25.52115	Average Speed						
4	3.56522	2	1449564	1.449564	1	296.875	0	4.36E+09	1.42E+12	1099	3106	0	1	37	2278978	25.52492	1.3723352						
5	4.75333	2	1436865	1.436865	1	300.6286	0	6.11E+09	1.42E+12	1099	3106	0	1	38	2278978	26.44647	Average Temp						
6	5.94144	2	1444506	1.444506	1	299.6279	0	7.85E+09	1.42E+12	1099	3106	0	1	39	2278978	26.99885	45.742574						
7	7.12955	2	1455312	1.455312	1	297.7735	0	9.6E+09	1.42E+12	1099	3106	0	1	39	2278978	26.79838							
8	8.31766	2	1472446	1.472446	1	294.5518	0	1.13E+10	1.42E+12	1099	3106	0	1	39	2278978	26.48654							
9	9.50577	2	1467217	1.467217	1	295.8023	0	1.31E+10	1.42E+12	1099	3106	0	1	40	2278978	27.2625							
10	10.69388	2	1463846	1.463846	1	296.6355	0	1.48E+10	1.42E+12	1099	3106	0	1	40	2278978	27.32528							
11	11.88199	2	1455005	1.455005	1	298.5662	0	1.66E+10	1.42E+12	1099	3106	0	1	41	2278978	28.1786							
12	13.0701	2	1449310	1.44931	1	299.8399	0	1.83E+10	1.42E+12	1099	3106	0	1	41	2278978	28.28932							
13	14.25821	2	1442168	1.442168	1	301.4113	0	2.01E+10	1.42E+12	1205	3106	0	1	42	2278978	29.12282							
14	15.44632	2	1438792	1.438792	1	302.1044	0	2.18E+10	1.42E+12	1205	3106	0	1	42	2278978	29.19115							
15	16.63443	2	1434087	1.434087	1	303.1619	0	2.36E+10	1.42E+12	1205	3106	0	1	42	2278978	29.28693							
16	17.82254	2	1430444	1.430444	1	303.9895	0	2.53E+10	1.42E+12	1205	3106	0	1	43	2278978	30.0606							
17	19.01065	2	1429144	1.429144	1	304.314	0	2.7E+10	1.42E+12	1205	3106	0	1	43	2278978	30.08794							
18	20.19876	2	1425486	1.425486	1	305.1402	0	2.88E+10	1.42E+12	1205	3106	0	1	43	2278978	30.16515							
19	21.38687	2	1422837	1.422837	1	305.7415	0	3.05E+10	1.42E+12	1211	3106	0	1	43	2278978	30.22131							
20	22.57498	2	1421021	1.421021	1	306.1665	0	3.23E+10	1.42E+12	1211	3106	0	1	44	2278978	30.96365							
21	23.76309	2	1418969	1.418969	1	306.6398	0	3.4E+10	1.42E+12	1211	3106	0	1	44	2278978	31.00843							
22	24.9512	2	1417246	1.417246	1	307.0407	0	3.58E+10	1.42E+12	1211	3106	0	1	44	2278978	31.04613							
23	26.13931	2	1414292	1.414292	1	307.7102	0	3.75E+10	1.42E+12	1211	3106	0	1	44	2278978	31.11097							
24	27.32742	2	1412246	1.412246	1	308.1803	0	3.93E+10	1.42E+12	1211	3106	0	1	44	2278978	31.15605							
25	28.51553	2	1411260	1.41126	1	308.4053	0	4.1E+10	1.42E+12	1223	3106	0	1	45	2278978	31.8864							
26	29.70364	2	1408655	1.408655	1	308.9964	0	4.28E+10	1.42E+12	1223	3106	0	1	45	2278978	31.94537							
27	30.89175	2	1405600	1.4056	1	309.6869	0	4.45E+10	1.42E+12	1223	3106	0	1	45	2278978	32.0148							
28	32.07986	2	1403154	1.403154	1	310.2406	0	4.58E+10	1.42E+12	1223	3106	0	1	45	2278978	32.07061							
29	33.26797	2	1401286	1.401286	1	310.6701	0	4.75E+10	1.42E+12	1223	3106	0	1	45	2278978	32.11336							
30	34.45608	2	1398458	1.398458	1	311.3098	0	4.89E+10	1.42E+12	1223	3106	0	1	45	2278978	32.1783							
31	35.64419	2	1396282	1.396282	1	311.8063	0	5.02E+10	1.42E+12	1223	3106	0	1	45	2278978	32.22845							

How do hashing algorithms in specific SHA256, SHA512 and MD5 affect Number of Passwords decrypted and Computational Power required in a given period?

7.5 Other References

1. "Combinator_Attack [Hashcat Wiki]". *Hashcat.Net*, 2018,
https://hashcat.net/wiki/doku.php?id=combinator_attack. Accessed 27 Aug 2018.
2. "Cracking Hash Algorithm". *Stack Overflow*, 2018,
<https://stackoverflow.com/questions/19484292/cracking-hash-algorithm>.
Accessed 17 May 2018.
3. "Dictionary_Attack [Hashcat Wiki]". *Hashcat.Net*, 2018,
https://hashcat.net/wiki/doku.php?id=dictionary_attack. Accessed 27 Aug 2018.
4. "Hashcat [Hashcat Wiki]". *Hashcat.Net*, 2018,
<https://hashcat.net/wiki/doku.php?id=hashcat>. Accessed 27 Aug 2018.
5. "How To Crack A Password Given Its Hash And Its Salt Using A More Efficient Method Than Brute Force?". *Information Security Stack Exchange*, 2018, <https://security.stackexchange.com/questions/153061/how-to-crack-a-password-given-its-hash-and-its-salt-using-a-more-efficient-metho>. Accessed 17 May 2018.
6. "Hybrid_Attack [Hashcat Wiki]". *Hashcat.Net*, 2018,
https://hashcat.net/wiki/doku.php?id=hybrid_attack. Accessed 27 Aug 2018.

7. "Mask_Attack [Hashcat Wiki]". *Hashcat.Net*, 2018,
https://hashcat.net/wiki/doku.php?id=mask_attack. Accessed 27 Aug 2018.
8. NIST. *Recommendation For Applications Using Approved Hash Algorithms*.
NIST, 2018, pp. 1-25.
9. NIST. *Recommendation For Random Number Generation Using Deterministic
Random Bit Generators*. NIST, 2018, pp. 1-110.
10. NIST. *SHA-3 Derived Functions*. NIST, 2018, pp. 1-32.
11. "NVIDIA Tesla K80 GPU Accelerator". *NVIDIA*, 2018,
<https://www.nvidia.com/en-us/data-center/tesla-k80/>. Accessed 27 Aug 2018.
12. "What Is Hashing? - Definition From Whatis.Com". *Searchsqlserver*, 2018,
<https://searchsqlserver.techtarget.com/definition/hashing>. Accessed 27 Aug
2018.
13. "What Is Password Entropy? - Definition From Whatis.Com". *Whatis.Com*,
2018, <https://whatis.techtarget.com/definition/password-entropy>. Accessed 27
Aug 2018.
14. Pound, Dr.Mike. *Password Cracking - Computerphile*. Computerphile, 2016.
15. Buldas, A. (2011). "Series of mini-lectures about cryptographic hash
functions". Archived from the original on 2012-12-06.

16. "Compute Engine - IaaS | Compute Engine | Google Cloud". Google Cloud, 2018. Online. Internet. 3 Jun. 2018. . Available: <https://cloud.google.com/compute/>.
17. "CUDA Parallel Computing Platform for Developers | NVIDIA". Nvidia.in, 2018. Online. Internet. 17 Sep. 2018. . Available: <https://www.nvidia.in/object/cuda-parallel-computing-in.html>.
18. "dictionary_attack [hashcat wiki]". Hashcat.net, 2018. Online. Internet. 12 Jun. 2018. Available: https://hashcat.net/wiki/doku.php?id=dictionary_attack.
19. "Dropbox/zxcvbn". GitHub, 2018. Online. Internet. 13 Nov. 2018. . Available: <https://github.com/dropbox/zxcvbn>.
20. "dwoolfhub/zxcvbn-python". GitHub, 2018. Online. Internet. 12 Nov. 2018. . Available: <https://github.com/dwoolfhub/zxcvbn-python>.
21. Halevi, Shai. "Cryptographic Hash Functions and their many applications". Presentation, USENIX Security – August 2009, 2009.
22. "hashcat [hashcat wiki]". Hashcat.net, 2018. Online. Internet. 11 Jul. 2018. . Available: <https://hashcat.net/wiki/doku.php?id=hashcat>.
23. "hashcat - advanced password recovery". Hashcat.net, 2018. Online. Internet. 17 May 2018. Available: <https://hashcat.net/hashcat/>.

24. Lowe Wheeler, Daniel. "zxcvbn: Low-Budget Password Strength Estimation".

This paper is included in the Proceedings of the 25th USENIX Security Symposium (2016). Online. Internet. 10 Nov. 2018. . Available:

https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_wheeler.pdf.

25. "NVIDIA Tesla Product Literature". NVIDIA, 2018. Online. Internet. 11 Aug.

2018. . Available: <https://www.nvidia.com/en-us/data-center/tesla-k80/>.

26. Rjařsko, Michal. "Properties of Cryptographic Hash Functions". Comenius

University in Bratislava Faculty of Mathematics, Physics and Informatics

Department of Computer Science (2008): 11-32. Online. Internet. 10 Jul.

2018. .

27. Sharma, Rakesh. "Cryptographic Hash Functions". Investopedia, 2018.

Online. Internet. 21 Nov. 2018. . Available:

<https://www.investopedia.com/news/cryptographic-hash-functions/>.

28. "SSH Protocol – Secure Remote Login and File Transfer | SSH.COM".

Ssh.com, 2018. Online. Internet. 21 Aug. 2018. Available:

<https://www.ssh.com/ssh/protocol/>.

29. Tarak, Rahul. "Cryogenicplanet/Extended-Essay." GitHub. N.p., 2018. Web.

21 Nov. 2018.