Password Cracker Report
CI510

Contents

Introduction .... page(s) 2

Requirements .... page(s) 2

Planning .... page(s) 3 - 5

Implementation .... page(s) 6 - 11

Evaluation .... page(s) 12

Bibliography .... page(s) 12

Introduction

In order to create the program, I have decided to go with Ruby, a scripting language.
I chose this type of language as they are quick to prototype programs, it can always be converted to a compiled language afterwards.

I have been provided with a dictionary file containing a list of commonly used words, from this file we will also create a list of hashed passwords to crack in order to test the program.
There are multiple methods we could implement such as:

- Bruteforce
- Dictionary
- Rainbow
- Mask

Requirements

The program has what looks to be 3 inputs as part of the main loop.
Firstly it takes the method type and then a dictionary file or bruteforce parameters followed by a list of hashed passwords to crack.

For my program I’m going to include the brute-force and dictionary attack methods to begin with.
Overall the program needs to:

- Take user input.
- Have a method to hash strings.
- Iterate files.
- Compare values between files.
- Iterate order of characters array

Planning

Attack Methods

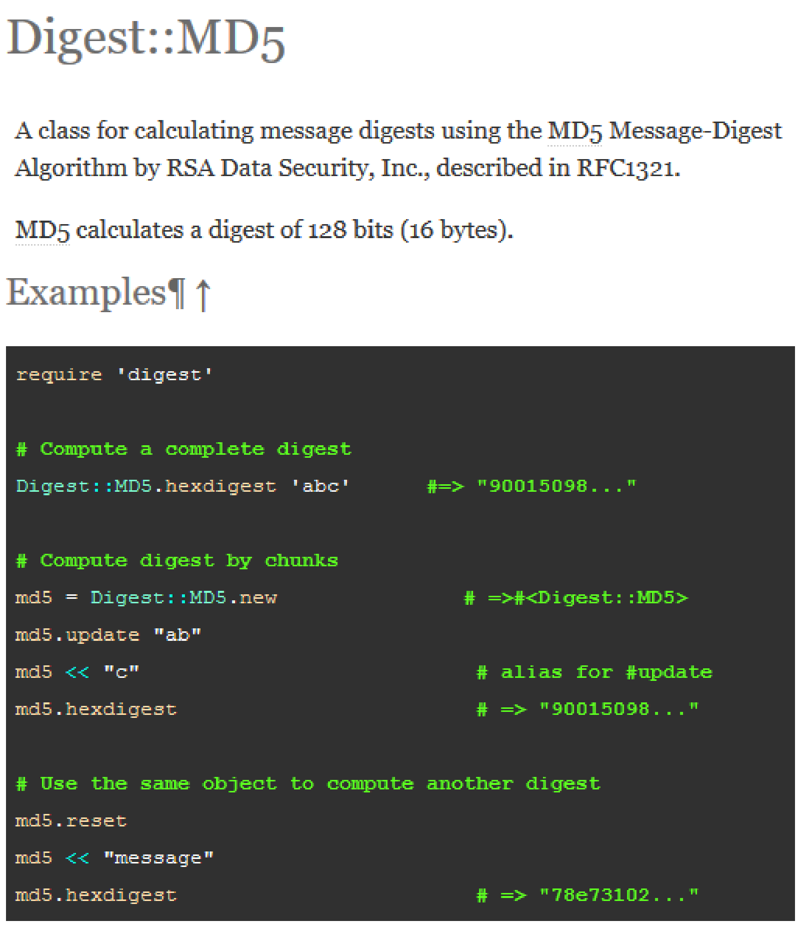
If the method type is bruteforce I need to take the users input on the length of the password then iterate through every possible combination of characters, hash each iteration and compare to the hashed password file to look for matches.
To create the character set that needs to be iterated through I'll take all the characters used from the dictionary file, the main program will contain an array of all these characters.

If the method type is a dictionary attack I need to I need to iterate the dictionary file, hash the words then iterate the hashed password file to look for matches.

Dependencies

In order to hash values in Ruby I'm going to need to utilise the gem called 'digest' which contains the method 'hexdigest' which will allow me to hash strings from the dictionary file.

After some research I found some examples of the gem in use from (Britt and Neurogami, 2020).



Digest::MD5

A class for calculating message digests using the MD5 Message-Digest Algorithm by RSA Data Security, Inc., described in RFC1321.

MD5 calculates a digest of 128 bits (16 bytes).

Examples¶ ↑

```
require 'digest'

# Compute a complete digest
Digest::MD5.hexdigest 'abc'      #=> "90015098..."

# Compute digest by chunks
md5 = Digest::MD5.new               # =>#<Digest::MD5>
md5.update "ab"
md5 << "c"                           # alias for #update
md5.hexdigest                        # => "90015098..."

# Use the same object to compute another digest
md5.reset
md5 << "message"
md5.hexdigest                        # => "78e73102..."
```

Functions

Main
Firstly the main function must handle the arguments given by the user within the command line interface and any error handling in regards to providing the necessary information in order for the program to function.

Hash
This function will simply return a MD5 hash value of any string given to it.

Bruteforce
Will contain an array of all possible characters, this could be changed to include the full ascii character set, for now we will simply take all characters used in the dictionary file.
All possible combinations of the characters will be iterated through with a length limit given by the user.

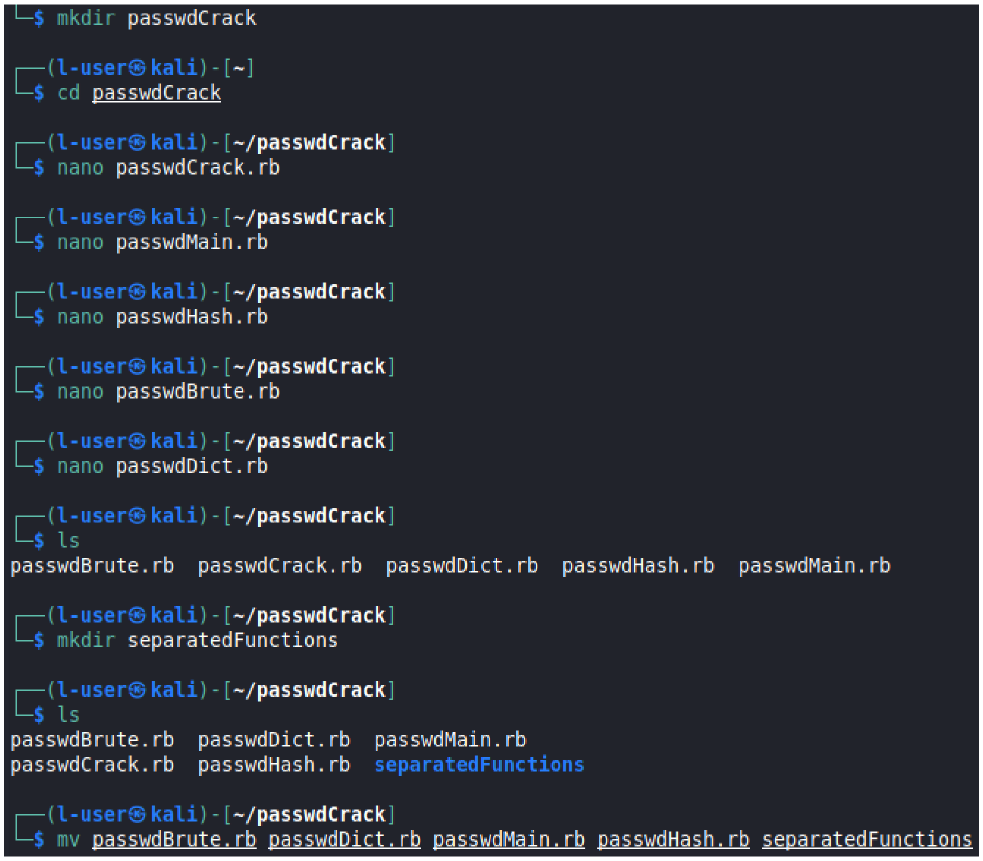
Dictionary
Needs to iterate the dictionary file given, hash each word and compare to the password file.

Read File
Want to make it separate from the dictionary function to make the program more expandable and reduce redundant code.
This will help if I want to expand the program with more cracking methods.

Implementation & Debugging

I decided to start with writing the “main” and “hash” function, to test the code I used a linux virtual machine (debian based distro) so I could run the program via the terminal.

To set up a development area I created a dedicated directory and a separate nested directory that contains the functions isolated from each other for demo purposes.

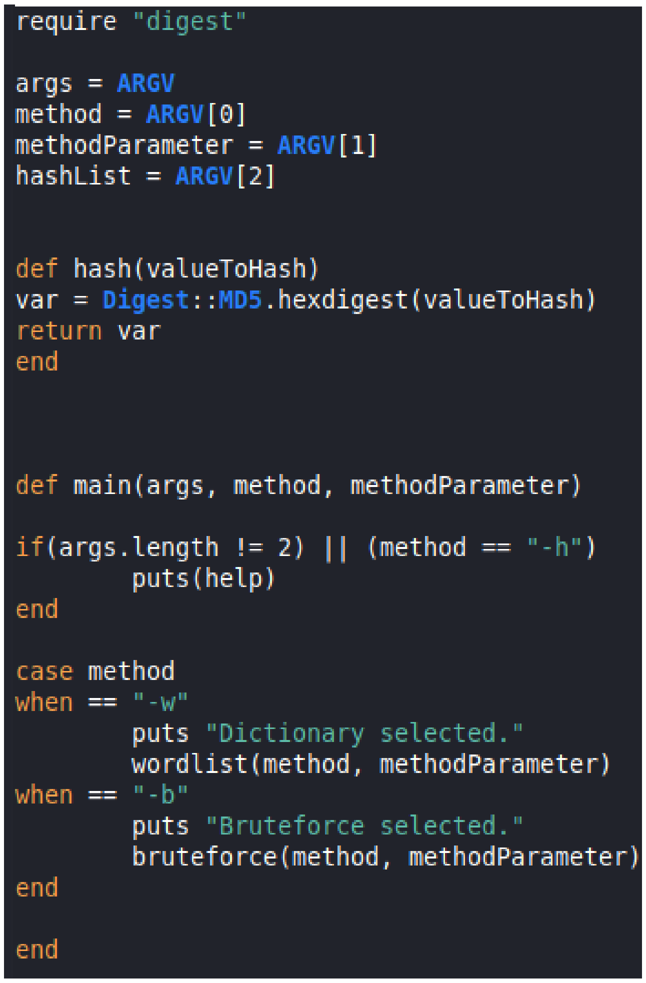


```
└─$ mkdir passwdCrack

┌──(l-user㉿kali)-[~]
└─$ cd passwdCrack

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ nano passwdCrack.rb

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ nano passwdMain.rb

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ nano passwdHash.rb

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ nano passwdBrute.rb

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ nano passwdDict.rb

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ ls
passwdBrute.rb  passwdCrack.rb  passwdDict.rb  passwdHash.rb  passwdMain.rb

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ mkdir separatedFunctions

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ ls
passwdBrute.rb  passwdDict.rb  passwdMain.rb
passwdCrack.rb  passwdHash.rb  separatedFunctions

┌──(l-user㉿kali)-[~/passwdCrack]
└─$ mv passwdBrute.rb passwdDict.rb passwdMain.rb passwdHash.rb separatedFunctions
```

After creating the directory structure I implemented the hash and main functions along with requiring the digest gem.

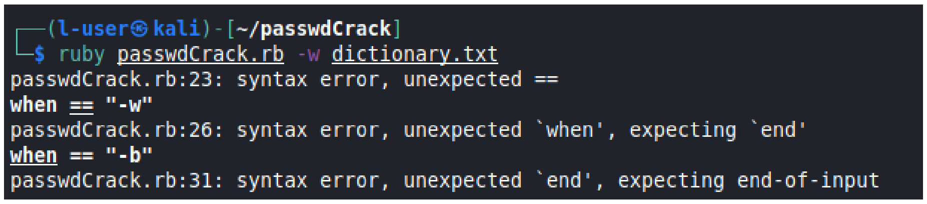
I took the idea to define each element of ARGV from (Flatiron, 2013), a blog on Ruby's ARGV usage.
This structure improves the readability and ease of coding.

I may move the second element of the "if" statement within the main function to the case statement as it may make more sense located there.



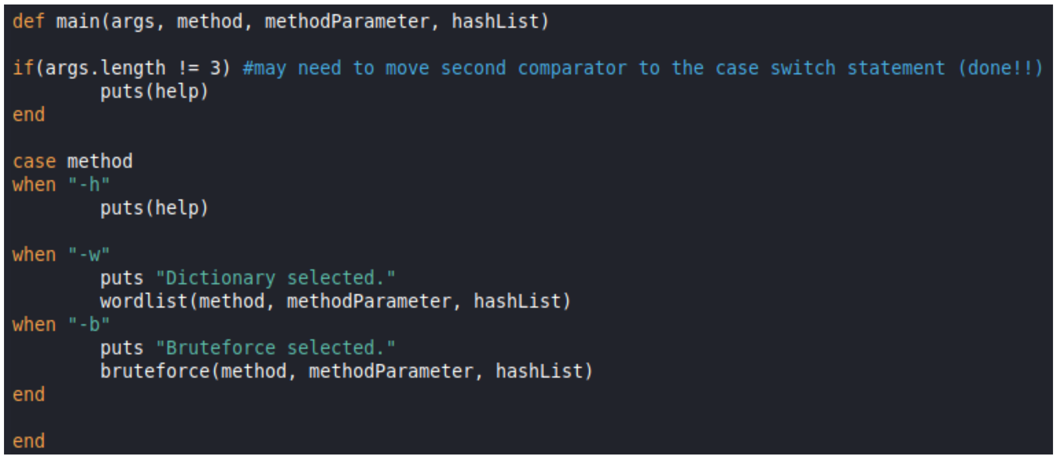
```
require "digest"

args = ARGV
method = ARGV[0]
methodParameter = ARGV[1]
hashList = ARGV[2]


def hash(valueToHash)
var = Digest::MD5.hexdigest(valueToHash)
return var
end



def main(args, method, methodParameter)

if(args.length != 2) || (method == "-h")
        puts(help)
end

case method
when == "-w"
        puts "Dictionary selected."
        wordlist(method, methodParameter)
when == "-b"
        puts "Bruteforce selected."
        bruteforce(method, methodParameter)
end

end
```

When attempting to run this code I ran into several errors which were fairly readable, as a result I was able to debug them fairly easily.



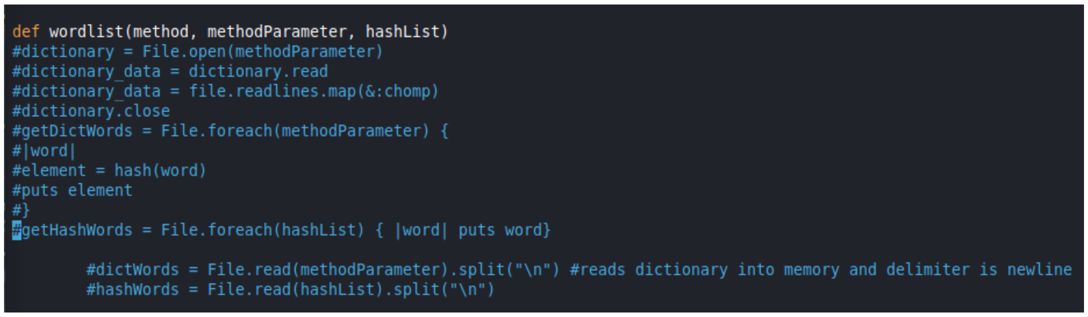
```
┌──(l-user㉿kali)-[~/passwdCrack]
└─$ ruby passwdCrack.rb -w dictionary.txt
passwdCrack.rb:23: syntax error, unexpected ==
when == "-w"
passwdCrack.rb:26: syntax error, unexpected `when', expecting `end'
when == "-b"
passwdCrack.rb:31: syntax error, unexpected `end', expecting end-of-input
```

The code below has been debugged and I moved around some of the logic.



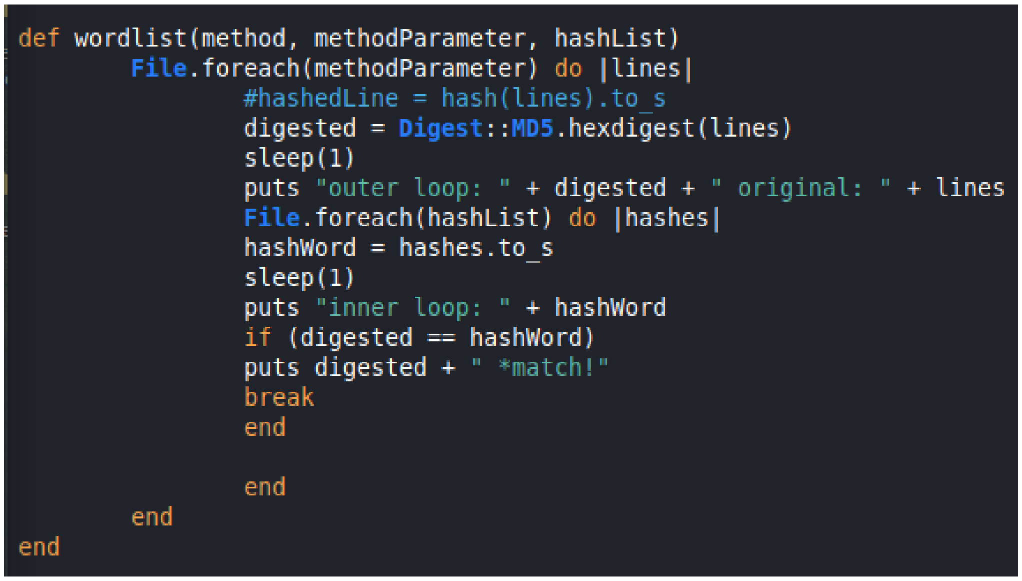
```
def main(args, method, methodParameter, hashList)

if(args.length != 3) #may need to move second comparator to the case switch statement (done!!)
        puts(help)
end

case method
when "-h"
        puts(help)

when "-w"
        puts "Dictionary selected."
        wordlist(method, methodParameter, hashList)
when "-b"
        puts "Bruteforce selected."
        bruteforce(method, methodParameter, hashList)
end

end
```

Moving onto creating the cracking functions, I began with the dictionary method.
This takes the 3 parameters “method”, “methodParameter” and “hashList”; I may remove the first parameter “method” as it seems redundant since the method has already been selected. Initially I had utilised some of the commented out code, however after doing some research I discovered that the file methods are not scalable and will cause the program to crash if the files are too large.



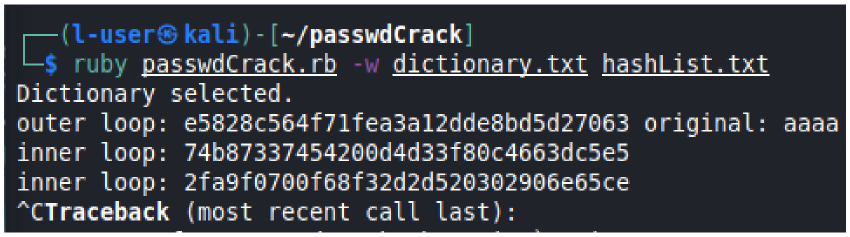
```
def wordlist(method, methodParameter, hashList)
#dictionary = File.open(methodParameter)
#dictionary_data = dictionary.read
#dictionary_data = file.readlines.map(&:chomp)
#dictionary.close
#getDictWords = File.foreach(methodParameter) {
#|word|
#element = hash(word)
#puts element
#}
#getHashWords = File.foreach(hashList) { |word| puts word}

        #dictWords = File.read(methodParameter).split("\n") #reads dictionary into memory and delimiter is newline
        #hashWords = File.read(hashList).split("\n")
```

After removing the commented out code I developed some better code with the help of (smnirven and DJ., 2010) where they mention the “foreach” method which will read the file line by line as opposed to loading the whole file into memory at once.
For this to work I have nested one loop within the other so each word we want to try in our dictionary is compared to every hash in the hash file.



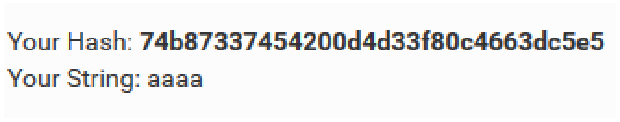
```
def wordlist(method, methodParameter, hashList)
        File.foreach(methodParameter) do |lines|
                #hashedLine = hash(lines).to_s
                digested = Digest::MD5.hexdigest(lines)
                sleep(1)
                puts "outer loop: " + digested + " original: " + lines
                File.foreach(hashList) do |hashes|
                hashWord = hashes.to_s
                sleep(1)
                puts "inner loop: " + hashWord
                if (digested == hashWord)
                puts digested + " *match!"
                break
                end

                end
        end
end
```

When testing this function I was not receiving hash matches, to debug this I printed the original string (unhashed) after the hash value to check I was getting the correct sum.

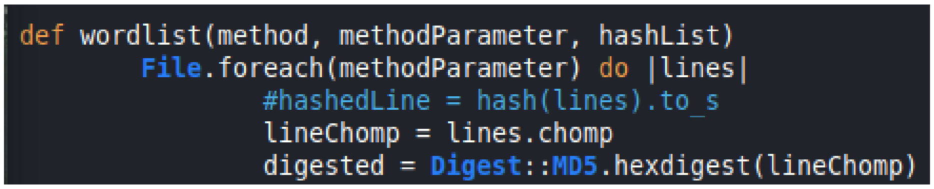


```
┌──(l-user㉿kali)-[~/passwdCrack]
└─$ ruby passwdCrack.rb -w dictionary.txt hashList.txt
Dictionary selected.
outer loop: e5828c564f71fea3a12dde8bd5d27063 original: aaaa
inner loop: 74b87337454200d4d33f80c4663dc5e5
inner loop: 2fa9f0700f68f32d2d520302906e65ce
^CTraceback (most recent call last):
```

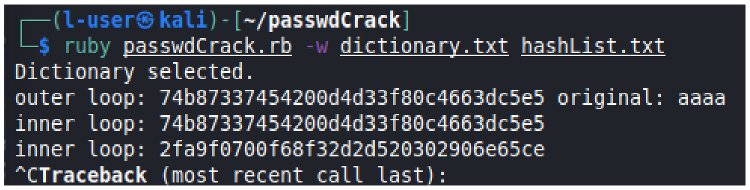
To create the hash file I had used the site: https://www.md5hashgenerator.com/.
In order to test the program worked as intended I copied the first dictionary word “aaaa” and hashed it.

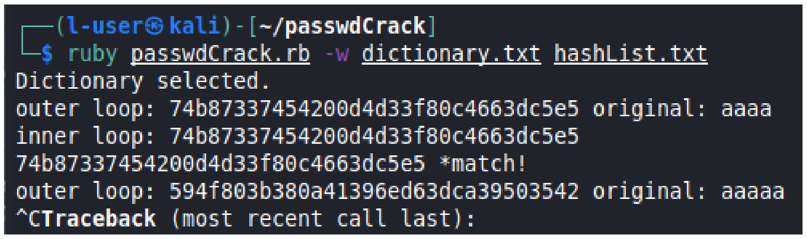

Your Hash: **74b87337454200d4d33f80c4663dc5e5**
Your String: aaaa

Here you can see that this hash value of “aaaa” does not match the outer loop hash value in the screenshot above in my terminal.
To get a better look at what was going on and for general debugging purposes I removed the hash function from the picture and just implemented the hash function code directly into the dictionary function under the variable name “digested”.
My theory is that there might be something interfering with the result such as a line break within the file.
Ruby has a method called “chomp” which can handle this so I tested my theory by implementing it into this function.


```
def wordlist(method, methodParameter, hashList)
        File.foreach(methodParameter) do |lines|
                #hashedLine = hash(lines).to_s
                lineChomp = lines.chomp
                digested = Digest::MD5.hexdigest(lineChomp)
```

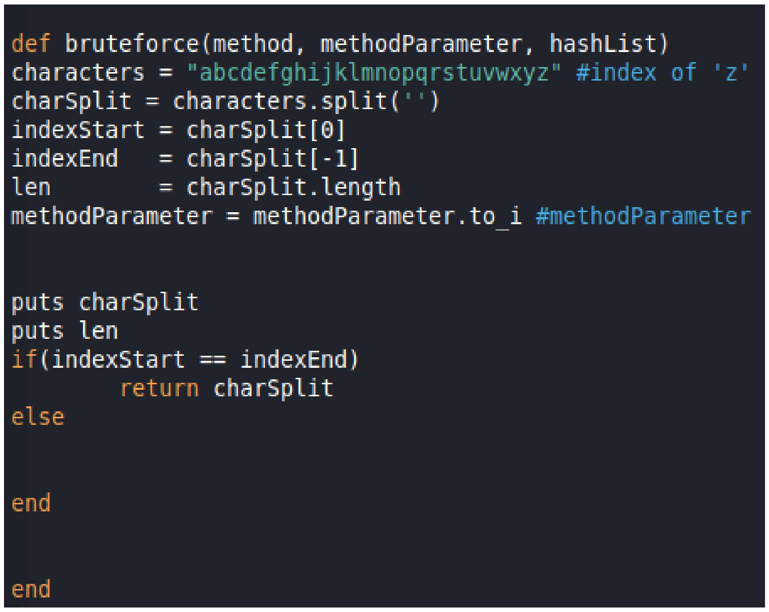
I added the variable “lineChomp” and fed this to the digest gem, the output of these changes can be seen below.
Here we can see that the outer loop and inner loop have the same hash value showing that my theory that a line break was interfering with the result is correct, however the main issue still persists, the program is not notifying us that there are matches, after rereading my code I found this to be a silly mistake as I had not “chomped” the hash only the dictionary word.
After correcting this I was getting matches.


```
┌──(l-user㉿kali)-[~/passwdCrack]
└─$ ruby passwdCrack.rb -w dictionary.txt hashList.txt
Dictionary selected.
outer loop: 74b87337454200d4d33f80c4663dc5e5 original: aaaa
inner loop: 74b87337454200d4d33f80c4663dc5e5
inner loop: 2fa9f0700f68f32d2d520302906e65ce
^CTraceback (most recent call last):
```

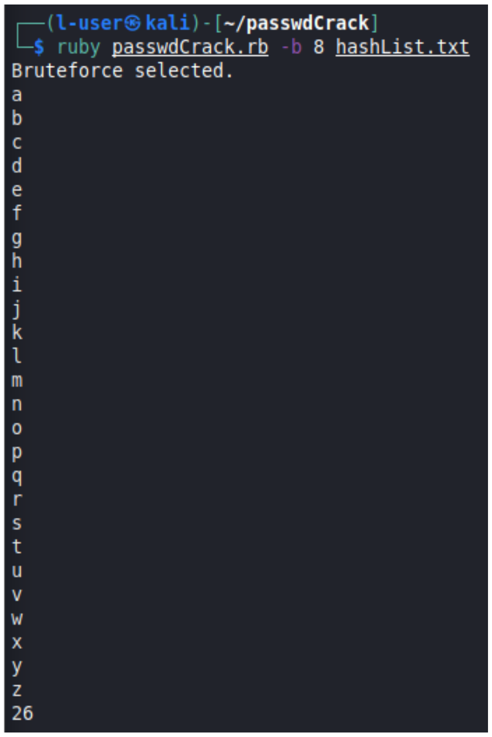

```
┌──(l-user㉿kali)-[~/passwdCrack]
└─$ ruby passwdCrack.rb -w dictionary.txt hashList.txt
Dictionary selected.
outer loop: 74b87337454200d4d33f80c4663dc5e5 original: aaaa
inner loop: 74b87337454200d4d33f80c4663dc5e5
74b87337454200d4d33f80c4663dc5e5 *match!
outer loop: 594f803b380a41396ed63dca39503542 original: aaaaa
^CTraceback (most recent call last):
```

With this function working I added back in the call to the hash function to keep everything separate which didn't effect the results.
Moving onto the bruteforce function, to set it up I looked at the dictionary file and saw that only lowercase letters are being used so I defined all lowercase letters as the "characterset" for the bruteforce method.

I utilised the same design as the main method by defining each position in the string to increase readability, it may affect performance as I'm creating more variables however it is easier to see what is going on.



```
def bruteforce(method, methodParameter, hashList)
characters = "abcdefghijklmnopqrstuvwxyz" #index of 'z'
charSplit = characters.split('')
indexStart = charSplit[0]
indexEnd   = charSplit[-1]
len        = charSplit.length
methodParameter = methodParameter.to_i #methodParameter


puts charSplit
puts len
if(indexStart == indexEnd)
        return charSplit
else


end


end
```



```
┌──(l-user㉿kali)-[~/passwdCrack]
└─$ ruby passwdCrack.rb -b 8 hashList.txt
Bruteforce selected.
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
26
```

To test this function I printed out the "charSplit" and "len" variables which gave me the correct results.
I now needed to iterate through each of these elements up to a length of digits given by the user which is inputted under the "methodParameter" variable.

After researching this thoroughly I could not find a reasonable and working implementation to get this to work.

Evaluation

I'm pleased with the results for the dictionary method and my ability to diagnose and debug the problems, however I wish I had put more time towards working on the bruteforce method as I did not predict how much work would be required to get this function to work.

Bibliography

Britt, J. and Neurogami (2020). *Class: Digest::MD5 (Ruby 2.5.1)*. [online] ruby-doc.org. Available at: https://ruby-doc.org/stdlib-2.5.1/libdoc/digest/rdoc/Digest/MD5.html [Accessed 3 Apr. 2021].

Flatiron (2013). *A Short Explanation of ARGV*. [online] flatironschool.com. Available at: https://flatironschool.com/blog/a-short-explanation-of-argv [Accessed 9 Apr. 2021].

smnirven and DJ. (2010). *ruby - Count the number of lines in a file without reading entire file into memory?* [online] Stack Overflow. Available at: https://stackoverflow.com/questions/2650517/count-the-number-of-lines-in-a-file-without-reading-entire-file-into-memory/2652196 [Accessed 9 Apr. 2021].

Other sources used but not referenced:

Kruczek, J. and Wibowo, A. (n.d.). *Bundler: gemfile*. [online] bundler.io. Available at: https://bundler.io/man/gemfile.5.html [Accessed 8 Apr. 2021].

Echessa, J. (2018). *Generating Random Numbers in Ruby*. [online] AppSignal Blog. Available at: https://blog.appsignal.com/2018/07/31/generating-random-numbers-in-ruby.html [Accessed 8 Apr. 2021].