

## **MID SEM ASSIGNMENT**

**In the partial fulfilment of B. tech -III-year course  
requirement of  
Subject: Cryptography**



**Submitted To:**

**Dr. Purnendu Shekar Pandey**

**School of Engineering & Technology**

**BML Munjal University**

**Submitted by:**

**Name: P. Mohana uma sushmanth**

**ID NO: 1800248C203**

**Section: CSE – 4**

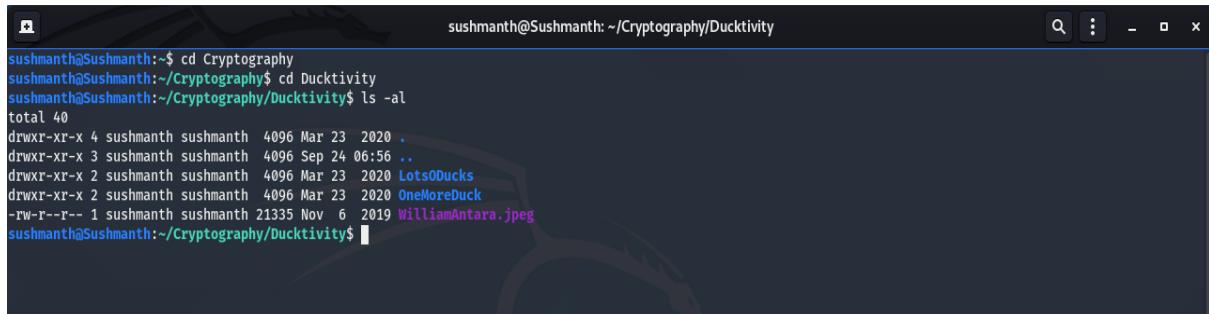
## PART – 1

### Exercise 1 – A NEEDLE IN A HAYSTACK

In this exercise, we are going to use hash value to identify any file containing the same data as the source file.

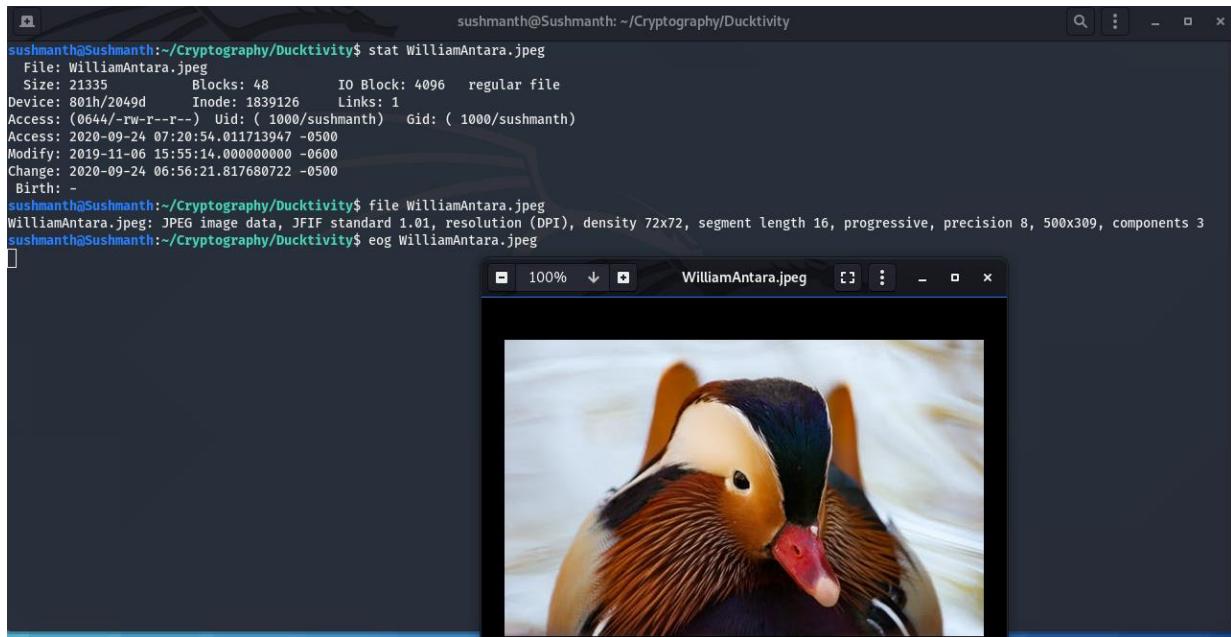
**Hash Value:** A hash value is a numeric value of a fixed length that uniquely identifies data. Hash values represent large amounts of data as much smaller numeric values, so they are used with digital signatures.

Now, lets do the exercise as per the given commands. First thing is using terminal, navigate to the Ducktivity folder and list the contents



```
sushmanth@Sushmanth:~/Cryptography/Ducktivity
sushmanth@Sushmanth:~/Cryptography$ cd Ducktivity
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ ls -al
total 40
drwxr-xr-x 4 sushmanth sushmanth 4096 Mar 23 2020 .
drwxr-xr-x 3 sushmanth sushmanth 4096 Sep 24 06:56 ..
drwxr-xr-x 2 sushmanth sushmanth 4096 Mar 23 2020 LotsOfDucks
drwxr-xr-x 2 sushmanth sushmanth 4096 Mar 23 2020 OneMoreDuck
-rw-r--r-- 1 sushmanth sushmanth 21335 Nov 6 2019 WilliamAntara.jpeg
sushmanth@Sushmanth:~/Cryptography/Ducktivity$
```

Examine the WilliamAntara.jpeg file



```
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ stat WilliamAntara.jpeg
  File: WilliamAntara.jpeg
  Size: 21335          Blocks: 48          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 1839126      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/sushmanth)  Gid: ( 1000/sushmanth)
Access: 2020-09-24 07:20:54.011713947 -0500
Modify: 2019-11-06 15:55:14.000000000 -0600
Change: 2020-09-24 06:56:21.817680722 -0500
 Birth: -
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ file WilliamAntara.jpeg
WilliamAntara.jpeg: JPEG image data, JFIF standard 1.01, resolution (DPI), density 72x72, segment length 16, progressive, precision 8, 500x309, components 3
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ eog WilliamAntara.jpeg
```

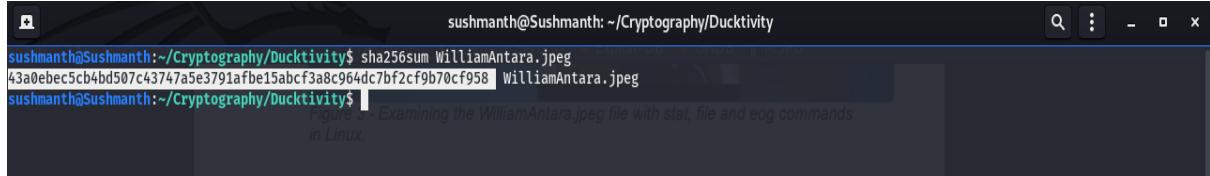


stat command gives the file system information

file command gives the content details

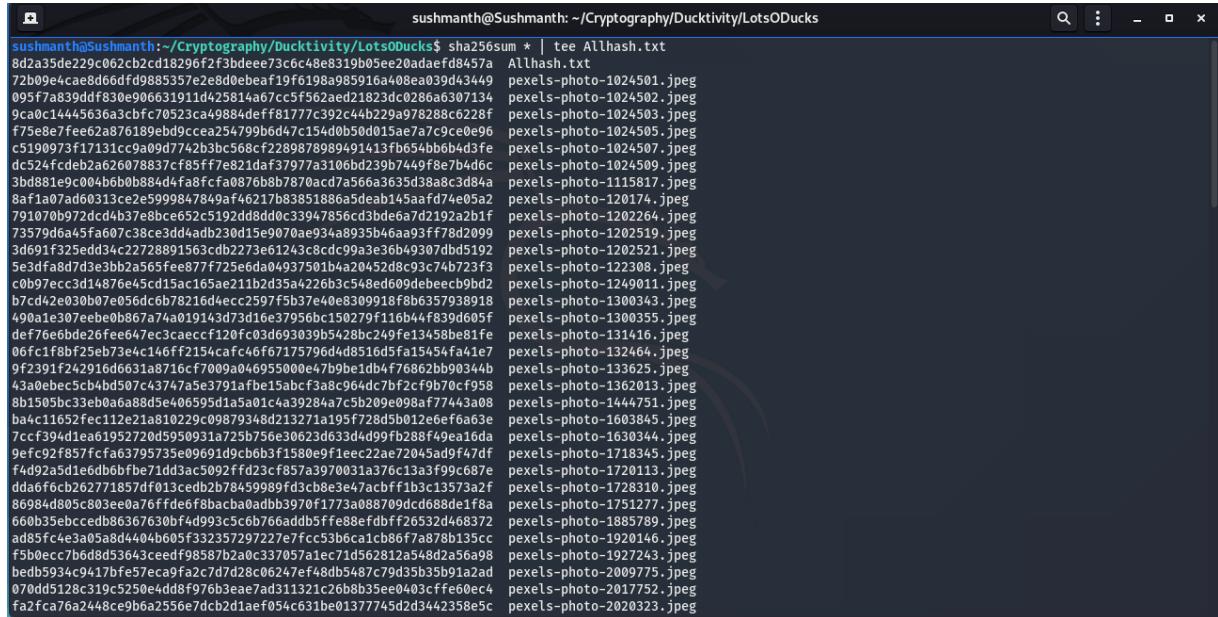
We can examine the picture directly by using Eye of GNOME (eog) picture viewer

Now, we will find the 256-bit hash value of WilliamAntara.jpeg by using sha256sum command.



```
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ sha256sum WilliamAntara.jpeg  
43a0ebec5cb4bd507c43747a5e3791afbe15abcf3a8c964dc7bf2cf9b70cf958 WilliamAntara.jpeg
```

Copy this hash value. And we have to find out hash values for all the files and store them in Allhash.txt file.



```
sushmanth@Sushmanth:~/Cryptography/Ducktivity/LotsODucks$ sha256sum * | tee Allhash.txt  
8d2a35de29c062cbcd18296f2f3bdeee73c6c48e8319b50e20daedf847a Allhash.txt  
72b09e4cae8d66fd9885357e2e8d0ebeaf19f6198a985916a408ea039d43449 pexels-photo-1024501.jpeg  
095f7a839ddf830e90663191d0425814a67cc5f562aed21823dc0286a6307134 pexels-photo-1024502.jpeg  
9ca0c144563ea3cbfc70523ca49884def8f1777c392c4b229a978288c6228f pexels-photo-1024503.jpeg  
f75e8e7fee62a876189ebd9ceaa54799b6d47c154d0b50d015ae7a7c9ce0e96 pexels-photo-1024505.jpeg  
c5190073f17131c9a09d724b2bc568cfc2298978999491413fb654bb6b43fe pexels-photo-1024507.jpeg  
dc524fcdebe2a626078d7f8e82idaef37977a3106bd239b7449f8e7b4dc6 pexels-photo-1024509.jpeg  
3bd881e9c004b6b00884d4fa8fcfa0876bb7870acd7m66a635d38a8c3d84a pexels-photo-1115817.jpeg  
8af8a07ad60313ce2e5999847849a4f6217b38351886a531ba145aafd74e052a pexels-photo-120174.jpeg  
791070b972dc4b37e8bce652c5192dd8dd0c33947856cd3dbea7d2192ab1f pexels-photo-1202264.jpeg  
73579d6a45fa07c38ce3ddcadb230d15e90703e93a4935b6aa93ff78d2099 pexels-photo-1202519.jpeg  
3d691f325edd34c2278891563cd2273e61243c8cd99a3e36b49307dbd5192 pexels-photo-1202521.jpeg  
5e3df4a8d7d3e3bb2a565cd15ac165aa21b2d35a4226b3c548ed609debeecb9hd2 pexels-photo-122308.jpeg  
c0b97ecc3d14876e45cd15ac165aa21b2d35a4226b3c548ed609debeecb9hd2 pexels-photo-1249011.jpeg  
b7cd42e030b07e056dc6b78216deccc2597f5b37e40e8309918fb86357938918 pexels-photo-1300343.jpeg  
490a1e307eebe0b867a74a019143d73d16e37956c150279f116b44f839d605f pexels-photo-1300355.jpeg  
def76e6bde26fee647ec3caeff120f203d693039b5428b249fe13458be81fe pexels-photo-131416.jpeg  
06fc1f8fb25eb73e4c146f2154caf46f67175796d4d8516d5fa154f4a1e7 pexels-photo-132464.jpeg  
9f2391f242916d6631a8716cf7009a046955000e47b9be1d4f76862bb90344b pexels-photo-133625.jpeg  
43a0bec5cb4bd507c43747a5e3791afbe15abcf3a8c964dc7bf2cf9b70cf958 pexels-photo-1362013.jpeg  
8b1505bc33eb0a688d5e406595d1a50a1c4a39284a7c5b209e098af77443a08 pexels-photo-1444751.jpeg  
ba4c11652fec112e21a810229c09879348d213271a195f728d5b012e6ef6a03e pexels-photo-1603845.jpeg  
7ccf394de1ea61952720d5950931a725b756e30623d633d4d99fb288f49e16da pexels-photo-1630344.jpeg  
9efc92f8577c3e0300e0467f8e1ec22ae72045ad9f47df pexels-photo-1718345.jpeg  
f4d92a5d1e6d6b6fb7e17dd3ac5092ffd23c8f857a3970031a376c13a3f99c687e pexels-photo-1720113.jpeg  
ddaf6fc6b262771857dfo13cedb2b7845989f3dc8e3e47acbf1b3c13573a2f pexels-photo-1728310.jpeg  
80984d805c803ee0a76fd6f8bachada0db3970f1773a088709cd688de1f8a pexels-photo-1751277.jpeg  
660b35ebccedb86307630bf4d993c5c6b76addbfff8eefdbf26532d468372 pexels-photo-1885789.jpeg  
ad85fc4e3a05a8d4404b605f332357297227e7fccc53b6ca1cb8ef7a878b135cc pexels-photo-1920146.jpeg  
f5b0ecc7b6d8d53643ceef98587b2a0c337057a1ec71d56281a548d2a56a98 pexels-photo-1927243.jpeg  
bed05934c9417bfe57eca9fa2c7d7d28c06247ef48db5487c9d35b35b91a2ad pexels-photo-2009775.jpeg  
070dd5128c319c5250e4dd8f976b3eae7ad311321c2608b35ee0403cff60ec4 pexels-photo-2017752.jpeg  
fa2fc7a76a2448ce9b6a2556e7dbc2da1ef054c631be01377745d2d3442358e5 pexels-photo-2020323.jpeg
```

Now we will compare the hash value of WilliamAntara.jpeg with all the hash values stored in Allhash.txt



```
sushmanth@Sushmanth:~/Cryptography/Ducktivity/LotsODucks$ cat Allhash.txt | grep "43a0ebec5cb4bd507c43747a5e3791afbe15abcf3a8c964dc7bf2cf9b70cf958"  
43a0ebec5cb4bd507c43747a5e3791afbe15abcf3a8c964dc7bf2cf9b70cf958 pexels-photo-1362013.jpeg
```

pexels-photo-1362013.jpeg has same hash value of WilliamAntara.jpeg.

Question: Assume for a second that you know the content of WilliamAntara.jpeg is Malicious and you know the hash of WilliamAntara.jpeg. If you identified another file on the drive that had the same hash as WilliamAntara.jpeg, can you assume that the other file is also malicious?

Answer: Yes, it is malicious. If the hash value is same, then data contain in file is also same.

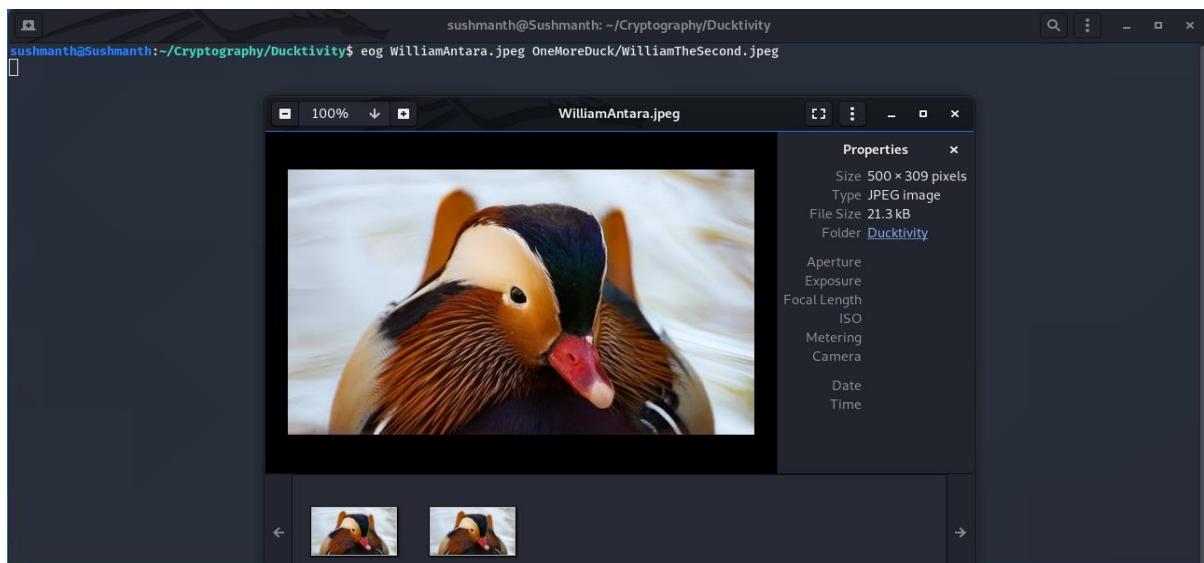
## Exercise 2 – DUCK DODGERS

Run stat command for both WilliamAntara.jpeg and WilliamTheSecond.jpeg.

WilliamTheSecond.jpeg is in OneMoreDuck directory.

```
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ stat WilliamAntara.jpeg
  File: WilliamAntara.jpeg
  Size: 21335        Blocks: 48          IO Block: 4096   regular file
Device: 801h/2049d    Inode: 1839126      Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/sushmanth)  Gid: ( 1000/sushmanth)
Access: 2020-10-01 01:13:33.607124820 -0500
Modify: 2019-11-06 15:55:14.000000000 -0600
Change: 2020-09-24 06:56:21.817680722 -0500
 Birth: -
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ stat OneMoreDuck/WilliamTheSecond.jpeg
  File: OneMoreDuck/WilliamTheSecond.jpeg
  Size: 21335        Blocks: 48          IO Block: 4096   regular file
Device: 801h/2049d    Inode: 1839125      Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/sushmanth)  Gid: ( 1000/sushmanth)
Access: 2019-11-06 16:32:12.000000000 -0600
Modify: 2019-11-06 16:32:12.000000000 -0600
Change: 2020-09-24 06:56:21.817680722 -0500
 Birth: -
sushmanth@Sushmanth:~/Cryptography/Ducktivity$
```

Open both WilliamAntara.jpeg and WilliamTheSecond.jpeg using eog

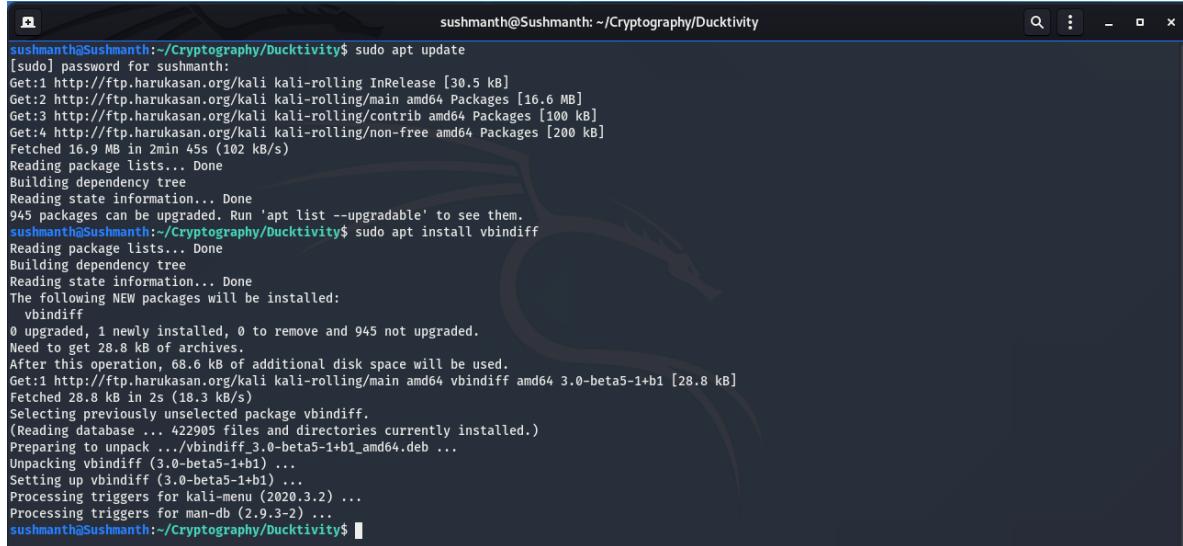


Both files look similar.

Now, After finding the hash values for both files. We observed both are different.

```
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ sha256sum WilliamAntara.jpeg
43a0ebec5cb4bd507c43747a5e3791afbe15abcf3a8c964dc7bf2cf9b70cf958  WilliamAntara.jpeg
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ sha256sum OneMoreDuck/WilliamTheSecond.jpeg
d7a88d2ca3c9822fe50ad517629c120ec77753161286408272517d1e1235463a  OneMoreDuck/WilliamTheSecond.jpeg
sushmanth@Sushmanth:~/Cryptography/Ducktivity$
```

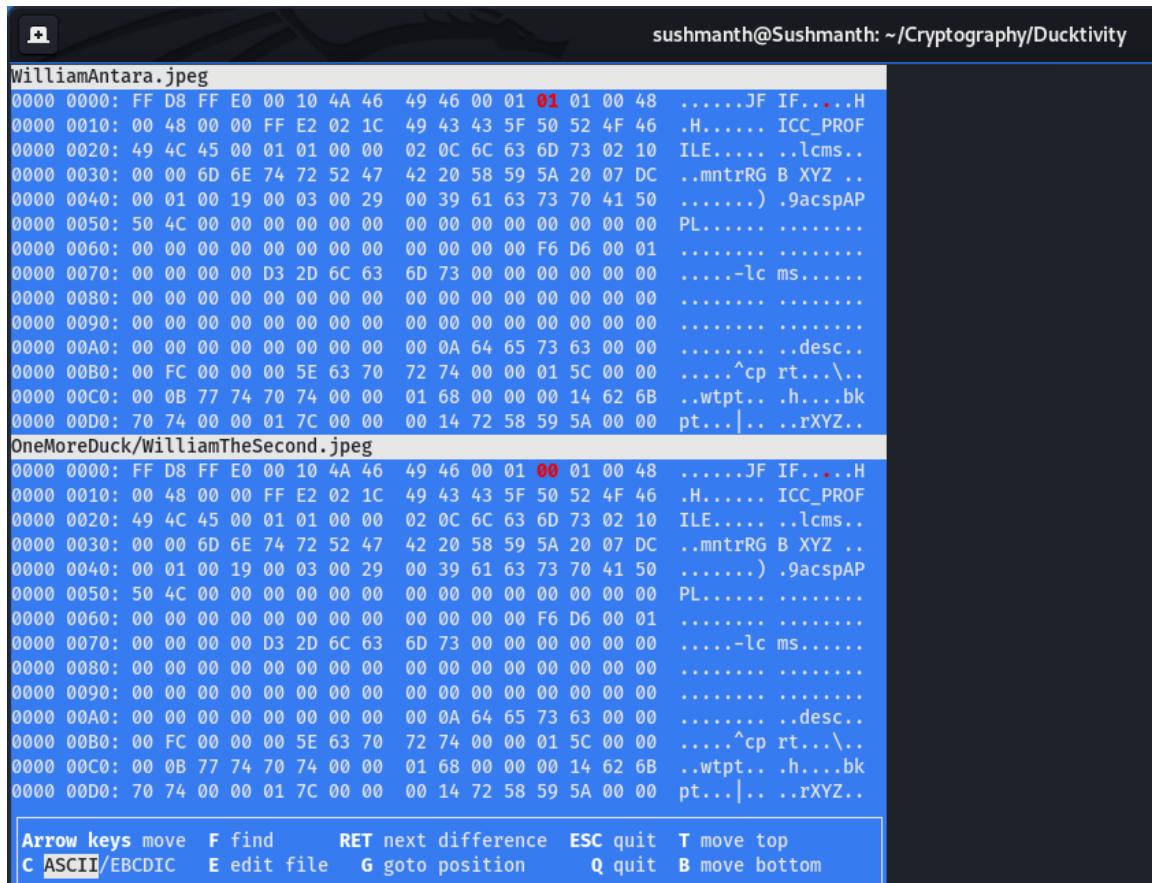
To find out reason behind it, we have to install Visual Binary Diff tool. It shows the file in hexadecimal and ASCII. It also says the difference between them by highlighting it.



```
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ sudo apt update
[sudo] password for sushmanth:
Get:1 http://ftp.harukasan.org/kali kali-rolling InRelease [30.5 kB]
Get:2 http://ftp.harukasan.org/kali kali-rolling/main amd64 Packages [16.6 MB]
Get:3 http://ftp.harukasan.org/kali kali-rolling/contrib amd64 Packages [100 kB]
Get:4 http://ftp.harukasan.org/kali kali-rolling/non-free amd64 Packages [200 kB]
Fetched 16.9 MB in 2min 45s (102 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
945 packages can be upgraded. Run 'apt list --upgradable' to see them.
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ sudo apt install vbindiff
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  vbindiff
0 upgraded, 1 newly installed, 0 to remove and 945 not upgraded.
Need to get 28.8 kB of archives.
After this operation, 68.6 kB of additional disk space will be used.
Get:1 http://ftp.harukasan.org/kali kali-rolling/main amd64 vbindiff amd64 3.0-beta5-1+b1 [28.8 kB]
Fetched 28.8 kB in 2s (18.3 kB/s)
Selecting previously unselected package vbindiff.
(Reading database ... 422905 files and directories currently installed.)
Preparing to unpack .../vbindiff_3.0-beta5-1+b1_amd64.deb ...
Unpacking vbindiff (3.0-beta5-1+b1) ...
Setting up vbindiff (3.0-beta5-1+b1) ...
Processing triggers for kali-menu (2020.3.2) ...
Processing triggers for man-db (2.9.3-2) ...
sushmanth@Sushmanth:~/Cryptography/Ducktivity$
```

By using this command –

```
vbindiff WilliamAntara.jpeg OneMoreDuck/WilliamTheSecond.jpeg
```



```
sushmanth@Sushmanth:~/Cryptography/Ducktivity
```

File	Address	Hex	ASCII	Description
WilliamAntara.jpeg	0000 0000	FF D8 FF E0 00 10 4A 46	.....JF IF....H	
	0010	00 48 00 00 FF E2 02 1C	.H..... ICC_PROF	
	0020	49 4C 45 00 01 01 00 00	..lcms..	
	0030	00 00 6D 6E 74 72 52 47	..mntrRG B XYZ ..	
	0040	00 01 00 19 00 03 00 29	.....) .9acspAP	
	0050	50 4C 00 00 00 00 00 00	PL.....	
	0060	00 00 00 00 00 00 00 00	.....	
	0070	00 00 00 00 D3 2D 6C 63	.....-lc ms.....	
	0080	00 00 00 00 00 00 00 00	.....	
	0090	00 00 00 00 00 00 00 00	.....	
OneMoreDuck/WilliamTheSecond.jpeg	00A0	00 0A 64 65 73 63 00 00	..... .desc..	
	00B0	00 FC 00 00 00 5E 63 70	.....^cp rt...\\..	
	00C0	00 0B 77 74 70 74 00 00	..wtpt.. .h....bk	
	00D0	70 74 00 00 01 7C 00 00	pt... ... .rXYZ..	
	00E0	00 00 00 00 00 00 00 00	.....	
	00F0	00 00 00 00 00 00 00 00	.....	
	0100	00 00 00 00 00 00 00 00	.....	
	0110	00 00 00 00 00 00 00 00	.....	
	0120	00 00 00 00 00 00 00 00	.....	
	0130	00 00 00 00 00 00 00 00	.....	

```
Arrow keys move F find      RET next difference  ESC quit  T move top
C ASCII/EBCDIC  E edit file  G goto position    Q quit  B move bottom
```

There is only one-bit difference. It is highlighted in red colour.

```
sushmanth@Sushmanth: ~/Cryptography/Ducktivity
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ file WilliamAntara.jpeg
WilliamAntara.jpeg: JPEG image data, JFIF standard 1.01, resolution (DPI), density 72x72, segment length 16, progressive, precision 8, 500x309, components 3
sushmanth@Sushmanth:~/Cryptography/Ducktivity$ file OneMoreDuck/WilliamTheSecond.jpeg
OneMoreDuck/WilliamTheSecond.jpeg: JPEG image data, JFIF standard 1.00, resolution (DPI), density 72x72, segment length 16, progressive, precision 8, 500x309, components 3
sushmanth@Sushmanth:~/Cryptography/Ducktivity$
```

After examining both files, we got to know that there is a difference in JFIF standard. That is why there is change in hash values. This technique would work well for detecting if a program has been altered. It would also detect malware that are known. The hash values do not help to identify new or altered malicious files.

**Question:** How is it possible that a file with the exact same size and content has a different hash value? To understand the cause of the different hash values, you need to examine the two files further.

Answer: The above image explains us that there is only one-bit change when compared to images data format. We already know that the only one bit can change entirely the hash value. So, the two images have different hash values.

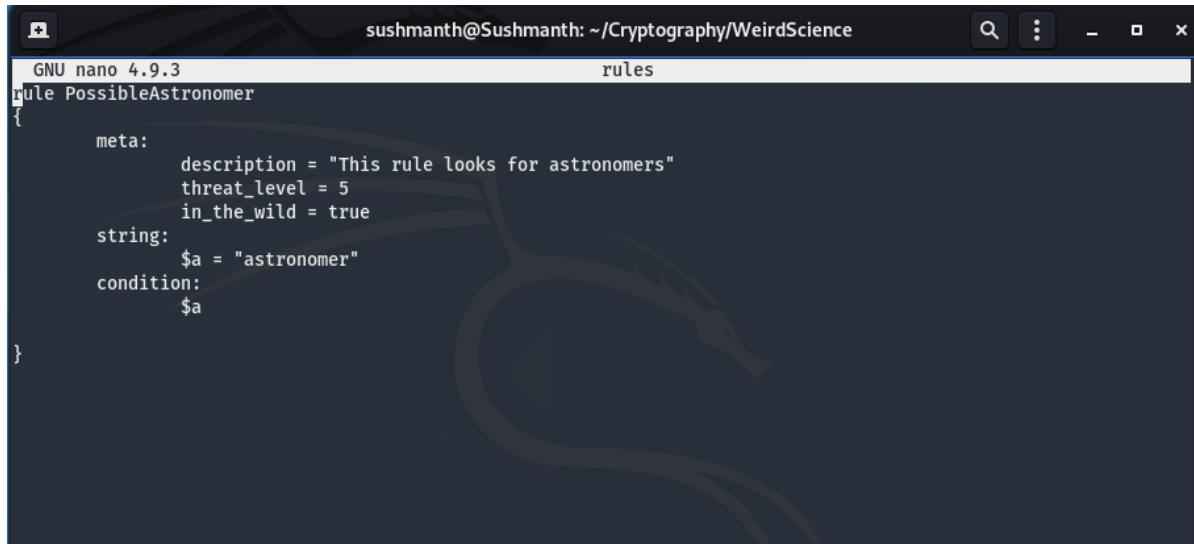
## **Exercise 3 - ASTRONOMERON THE LOOSE**

In this exercise, we are going to use Yara tool. Yara tool is used to identify malware samples. Here focus is to identify part of file which is similar to malicious content. Version of my Yara tool is 4.0.2

A screenshot of a terminal window titled "sushmanth@Sushmanth: ~/Cryptography/Ducktivity". The command "yara --version" is run, displaying the output "4.0.2". The terminal has a dark blue background with white text.

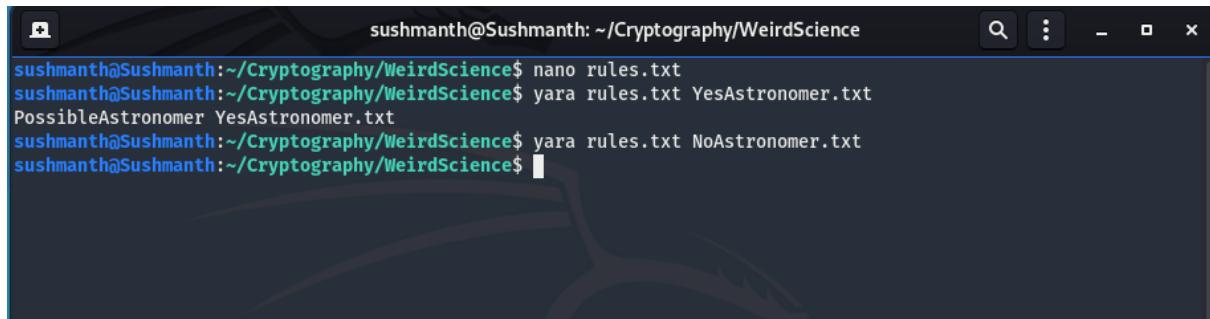
Now, we need to identify the word “Astronomer” in the “YesAstronomer” file

This is “YesAstronomer” file. And the rule for find out the Astronomer word is written in rules.txt file shown below.



```
sushmanth@Sushmanth: ~/Cryptography/WeirdScience
GNU nano 4.9.3                                         rules
rule PossibleAstronomer
{
    meta:
        description = "This rule looks for astronomers"
        threat_level = 5
        in_the_wild = true
    string:
        $a = "astronomer"
    condition:
        $a
```

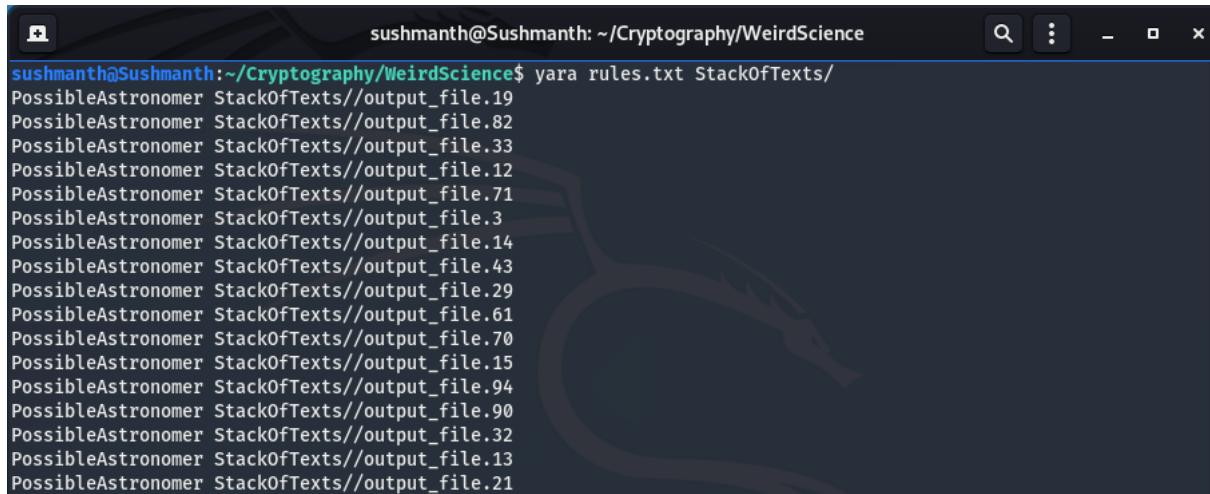
Now, Run the yara command for both files YesAstronomer.txt and NoAstronomer.txt for checking whether Astronomer word is present or not.



```
sushmanth@Sushmanth:~/Cryptography/WeirdScience$ nano rules.txt
sushmanth@Sushmanth:~/Cryptography/WeirdScience$ yara rules.txt YesAstronomer.txt
PossibleAstronomer YesAstronomer.txt
sushmanth@Sushmanth:~/Cryptography/WeirdScience$ yara rules.txt NoAstronomer.txt
sushmanth@Sushmanth:~/Cryptography/WeirdScience$
```

The word Astronomer is in “YesAstronomer.txt” and not in “NoAstronomer.txt”.

Now, Run the yara command on all files in the directory StackOfTexts with same rule by using the command “yara rules.txt StackOfTexts/”

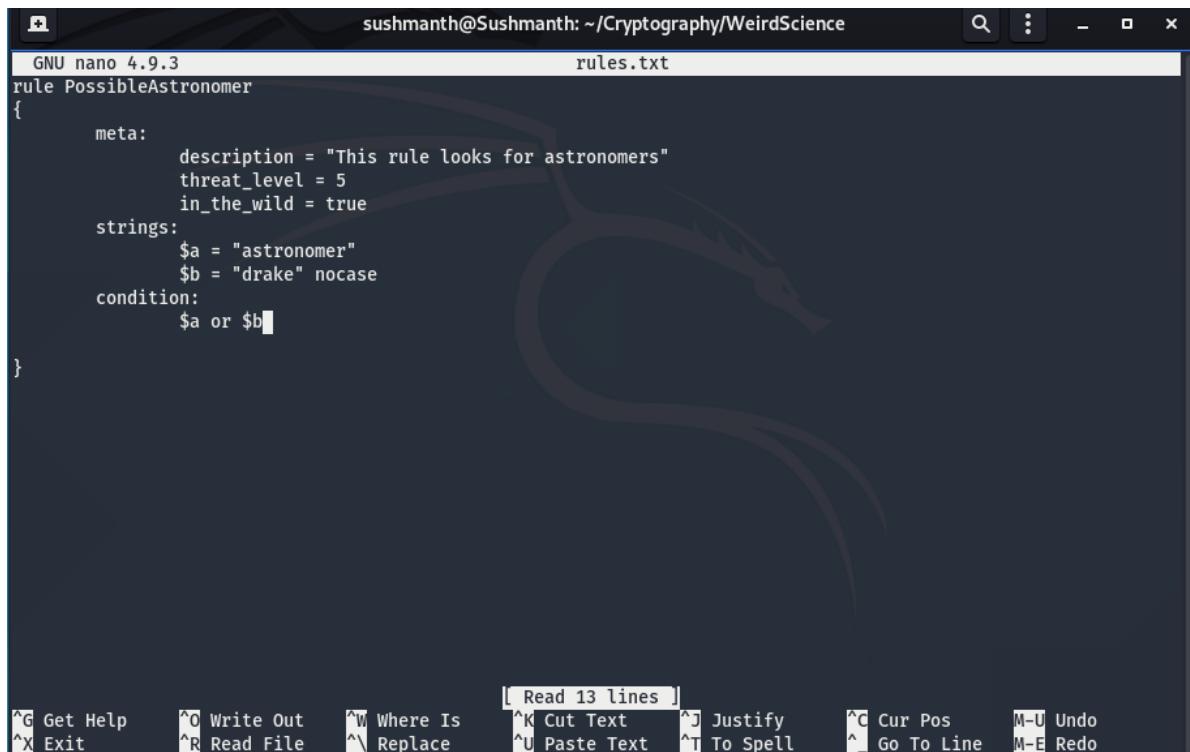


```
sushmanth@Sushmanth:~/Cryptography/WeirdScience$ yara rules.txt StackOfTexts/
PossibleAstronomer StackOfTexts//output_file.19
PossibleAstronomer StackOfTexts//output_file.82
PossibleAstronomer StackOfTexts//output_file.33
PossibleAstronomer StackOfTexts//output_file.12
PossibleAstronomer StackOfTexts//output_file.71
PossibleAstronomer StackOfTexts//output_file.3
PossibleAstronomer StackOfTexts//output_file.14
PossibleAstronomer StackOfTexts//output_file.43
PossibleAstronomer StackOfTexts//output_file.29
PossibleAstronomer StackOfTexts//output_file.61
PossibleAstronomer StackOfTexts//output_file.70
PossibleAstronomer StackOfTexts//output_file.15
PossibleAstronomer StackOfTexts//output_file.94
PossibleAstronomer StackOfTexts//output_file.90
PossibleAstronomer StackOfTexts//output_file.32
PossibleAstronomer StackOfTexts//output_file.13
PossibleAstronomer StackOfTexts//output_file.21
```

Run another command which tells the count (Number of files detected).

```
sushmanth@sushmanth:~/Cryptography/WeirdScience$ yara rules.txt StackOfTexts/ | wc -l
17
sushmanth@sushmanth:~/Cryptography/WeirdScience$
```

Now, we are making some changes in the rule (rules.txt). Add another word “drake” (Case sensitive)



```
sushmanth@sushmanth: ~/Cryptography/WeirdScience
GNU nano 4.9.3                                         rules.txt
rule PossibleAstronomer
{
    meta:
        description = "This rule looks for astronomers"
        threat_level = 5
        in_the_wild = true
    strings:
        $a = "astronomer"
        $b = "drake" nocase
    condition:
        $a or $b
}
```

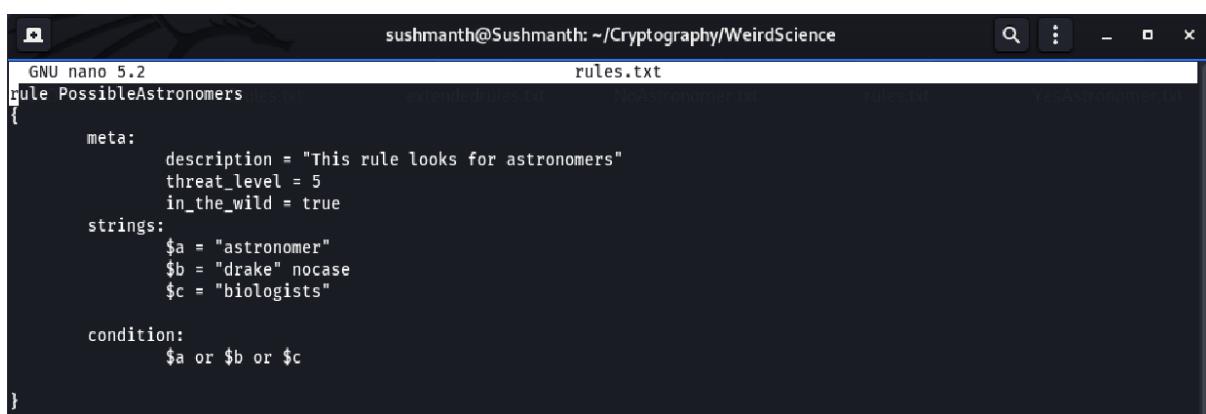
[ Read 13 lines ]

**^G** Get Help **^O** Write Out **^W** Where Is **^K** Cut Text **^J** Justify **^C** Cur Pos  
**^X** Exit **^R** Read File **^A** Replace **^U** Paste Text **^T** To Spell **^G** Go To Line **M-U** Undo  
**M-E** Redo

Now run the yara command to see the detected files.

```
sushmanth@sushmanth:~/Cryptography/WeirdScience$ yara rules.txt StackOfTexts/ | wc -l
24
sushmanth@sushmanth:~/Cryptography/WeirdScience$
```

There are total 24 files detected for the above rule. Now add the word “biologists” in the rule and run the command.



```
sushmanth@sushmanth: ~/Cryptography/WeirdScience
GNU nano 5.2                                         rules.txt
rule PossibleAstronomers
{
    meta:
        description = "This rule looks for astronomers"
        threat_level = 5
        in_the_wild = true
    strings:
        $a = "astronomer"
        $b = "drake" nocase
        $c = "biologists"
    condition:
        $a or $b or $c
}
```

The result is same as before. After running Yara command we observed that the result is 24 which tell that there is no change in the number of files found.

```
sushmanth@sushmanth:~/Cryptography/WeirdScience$ yara rules.txt StackOfTexts/ | wc -l
24
sushmanth@sushmanth:~/Cryptography/WeirdScience$
```

#### Exercise 4 – LATIN, THE LANGUAGE OF DISSENT

Here we are going to make new rules for different strings. Below figure shows the list of strings and the conditions.

Classification	Threat Level	Words	Notes
PossibleAstronomers	5	astronomer, drake	Rule already created in previous exercise.
AstronomicalContent	10	star, galaxy, orion, vega	Match any of these words regardless of case.
LatinThinkers	7	vitae, dicta, exercitationem, nostrum, voluptate, Dorkus Malorkus	Only match in files that are larger than 600 bytes.
Scientists	20	Archimedes, Newton, Euclid, Einstein, Hawking	Only match if present with AstronomicalContent rule
PossibleScienceContent	50	gravity, intelligence, brain, energy	Only match if files are less than 500 bytes AND NOT already matched by AstronomicalContent

Copy the existing rules.txt file to an extendedrules.txt

```
sushmanth@sushmanth:~/Cryptography/WeirdScience$ cp rules.txt extendedrules.txt
sushmanth@sushmanth:~/Cryptography/WeirdScience$
```

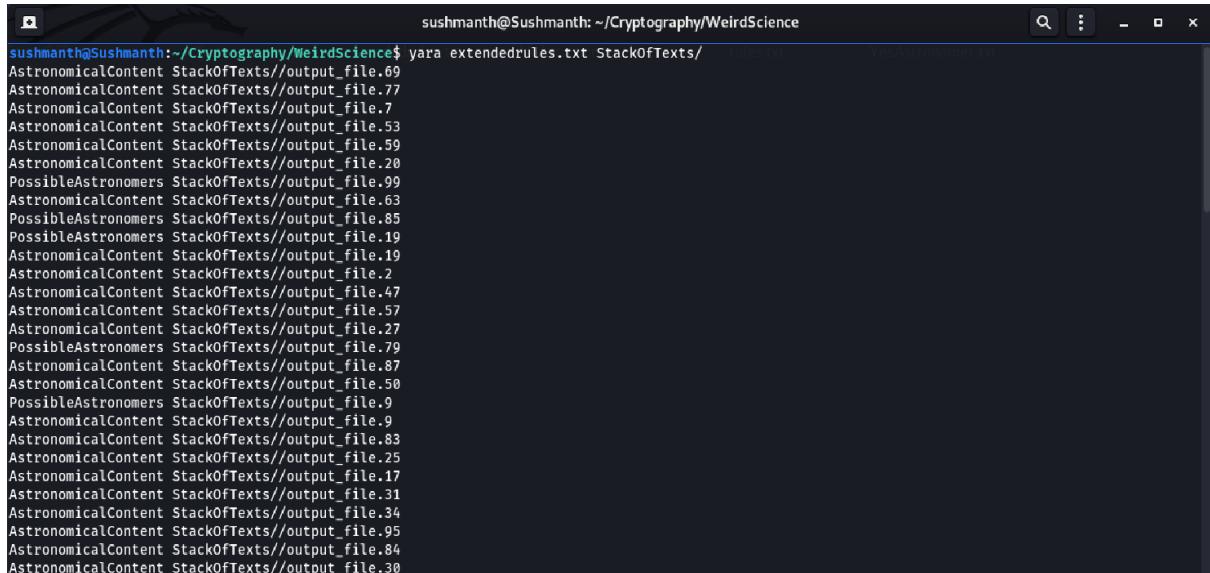
Now edit the extendedrules.txt file by add another rule. That is AstronomicalContent.

```
rules.txt
~/Cryptography/WeirdScience

1 rule PossibleAstronomers
2 {
3     meta:
4         description = "This rule looks for astronomers"
5         threat_level = 5
6         in_the_wild = true
7     strings:
8         $a = "astronomer"
9         $b = "drake" nocase
10
11    condition:
12        $a or $b
13 }
14 }
15 rule AstronomicalContent
16 {
17     meta:
18         description = "for AstronomicalContent"
19         threat_level = 10
20         in_the_wild = true
21     strings:
22         $a = "star" nocase
23         $b = "galaxy" nocase
24         $c = "orion" nocase
25         $d = "vega" nocase
26     condition:
27        $a or $b or $c or $d
28 }
29 }
```

Now, Test the new rule in extendedrules.txt and verify that it's detecting PossibleAstronomers and AstronomicalContent by using following command:

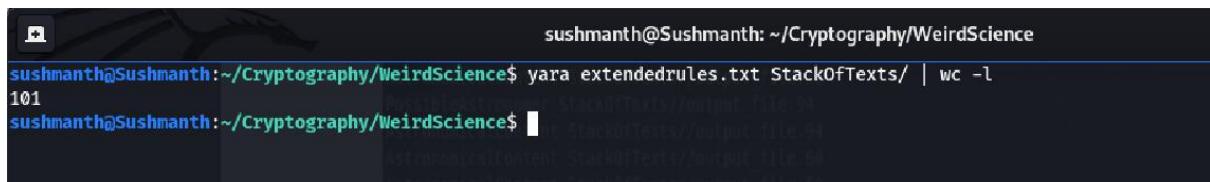
```
yara extendedrules.txt StackOfTexts/
```



A screenshot of a terminal window titled "sushmanth@Sushmanth: ~/Cryptography/WeirdScience". The command "yara extendedrules.txt StackOfTexts/" is run, and the output shows numerous matches across many files in the directory, including "AstronomicalContent" and "PossibleAstronomers" rules.

```
sushmanth@Sushmanth:~/Cryptography/WeirdScience$ yara extendedrules.txt StackOfTexts/
AstronomicalContent StackOfTexts//output_file.69
AstronomicalContent StackOfTexts//output_file.77
AstronomicalContent StackOfTexts//output_file.7
AstronomicalContent StackOfTexts//output_file.53
AstronomicalContent StackOfTexts//output_file.59
AstronomicalContent StackOfTexts//output_file.20
PossibleAstronomers StackOfTexts//output_file.99
AstronomicalContent StackOfTexts//output_file.63
PossibleAstronomers StackOfTexts//output_file.85
PossibleAstronomers StackOfTexts//output_file.19
AstronomicalContent StackOfTexts//output_file.19
AstronomicalContent StackOfTexts//output_file.2
AstronomicalContent StackOfTexts//output_file.47
AstronomicalContent StackOfTexts//output_file.57
AstronomicalContent StackOfTexts//output_file.27
PossibleAstronomers StackOfTexts//output_file.79
AstronomicalContent StackOfTexts//output_file.87
AstronomicalContent StackOfTexts//output_file.58
PossibleAstronomers StackOfTexts//output_file.9
AstronomicalContent StackOfTexts//output_file.9
AstronomicalContent StackOfTexts//output_file.83
AstronomicalContent StackOfTexts//output_file.25
AstronomicalContent StackOfTexts//output_file.17
AstronomicalContent StackOfTexts//output_file.31
AstronomicalContent StackOfTexts//output_file.34
AstronomicalContent StackOfTexts//output_file.95
AstronomicalContent StackOfTexts//output_file.84
AstronomicalContent StackOfTexts//output_file.30
```

Now see the number of files detected.



A screenshot of a terminal window titled "sushmanth@Sushmanth: ~/Cryptography/WeirdScience". The command "yara extendedrules.txt StackOfTexts/ | wc -l" is run, resulting in the output "101".

```
sushmanth@Sushmanth:~/Cryptography/WeirdScience$ yara extendedrules.txt StackOfTexts/ | wc -l
101
sushmanth@Sushmanth:~/Cryptography/WeirdScience$
```

Now we have written all rules for the conditions given in above table.

Rule for LatinThinkers:

```
rule LatinThinkers
{
    meta:
        description = "for LatinThinkers"
        threat_level = 7
        in_the_wild = true
    strings:
        $a = "vitae"
        $b = "dicta"
        $c = "exercitationem"
        $d = "nostrum"
        $e = "voluptate"
        $f = "Dorkus Malorkus"

    condition:
        any of them and (filesize>600)
}
```

### Rule for Scientists:

```
rule Scientists
{
    meta:
        description = "for Scientists"
        threat_level = 20
        in_the_wild = true
    strings:
        $a = "Archimedes"
        $b = "Newton"
        $c = "Euclid"
        $d = "Einstein"
        $e = "Hawking"
    condition:
        any of them and AstronomicalContent
}
```

### Rule for PossibleScienceContent:

```
rule PossibleScienceContent
{
    meta:
        description = "for PossibleScienceContent"
        threat_level = 50
        in_the_wild = true
    strings:
        $a = "gravity"
        $b = "intelligence"
        $c = "brain"
        $d = "energy"
    condition:
        any of them and not AstronomicalContent and (filesize<500)
}
```

Now, we have included these all rules in extendedrules.txt file and run the yara command to verify the number of files detected.

```
sushmanth@Sushmanth:~/Cryptography/WeirdScience$ yara extendedrules.txt StackOfTexts/
PossibleScienceContent StackOfTexts//output_file.41
AstronomicalContent StackOfTexts//output_file.69
AstronomicalContent StackOfTexts//output_file.77
AstronomicalContent StackOfTexts//output_file.7
LatinThinkers StackOfTexts//output_file.7
LatinThinkers StackOfTexts//output_file.55
AstronomicalContent StackOfTexts//output_file.53
AstronomicalContent StackOfTexts//output_file.59
LatinThinkers StackOfTexts//output_file.59
AstronomicalContent StackOfTexts//output_file.20
Scientists StackOfTexts//output_file.20
PossibleAstronomers StackOfTexts//output_file.99
LatinThinkers StackOfTexts//output_file.99
AstronomicalContent StackOfTexts//output_file.63
LatinThinkers StackOfTexts//output_file.63
PossibleAstronomers StackOfTexts//output_file.85
LatinThinkers StackOfTexts//output_file.85
PossibleAstronomers StackOfTexts//output_file.19
AstronomicalContent StackOfTexts//output_file.19
LatinThinkers StackOfTexts//output_file.19
AstronomicalContent StackOfTexts//output_file.2
LatinThinkers StackOfTexts//output_file.2
AstronomicalContent StackOfTexts//output_file.47
PossibleAstronomers StackOfTexts//output_file.82
AstronomicalContent StackOfTexts//output_file.82
LatinThinkers StackOfTexts//output_file.82
AstronomicalContent StackOfTexts//output_file.82
AstronomicalContent StackOfTexts//output_file.57
AstronomicalContent StackOfTexts//output_file.27
```

Now see the count (Number of files detected).

```
sushmanth@Sushmanth:~/Cryptography/WeirdScience$ yara extendedrules.txt StackOfTexts/ | wc -l
156
sushmanth@Sushmanth:~/Cryptography/WeirdScience$
```

## Exercise 5 – UNDER ATTACK

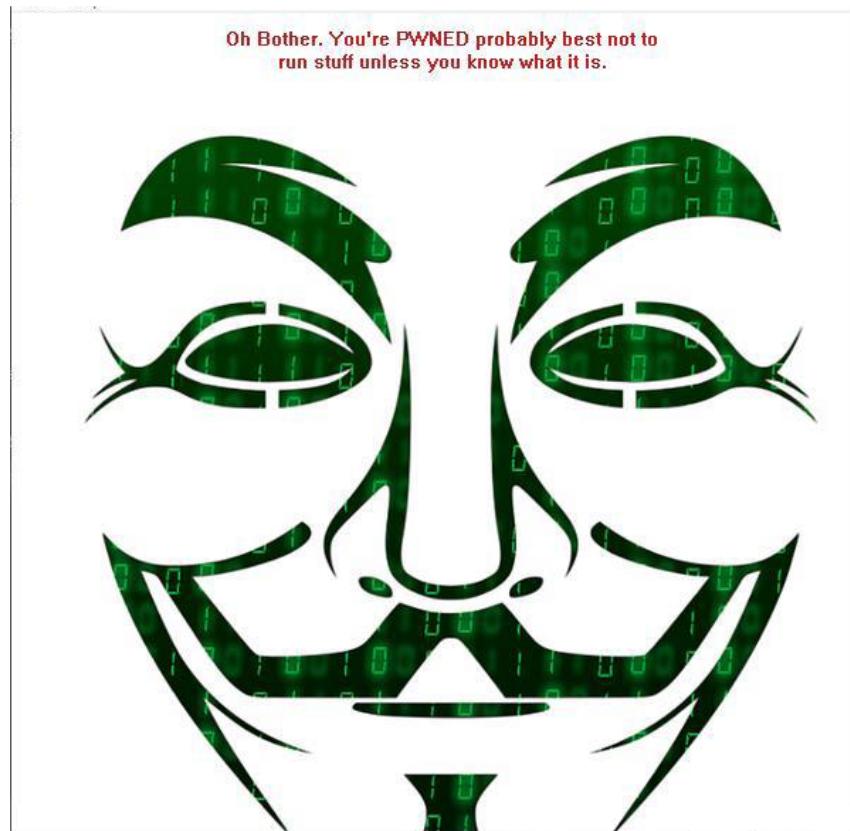
In this exercise, we are trying to detect malicious software which is in the form “.exe”. We have the text file containing certain rules to detect malicious software. Run the command given below:

```
yara -r virusdefs_v1801.txt Executables/
```

```
sushmanth@Sushmanth: ~/Cryptography/NoPwnZone
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ ls
Executables  virusdefs_v1801.txt  virusdefs_v2023.txt
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ yara -r virusdefs_v1801.txt Executables/
RansomwareA Executables//KnownMalware/RansomwareSample.exe
MalwareB Executables//ToTest/SimpleApp.exe
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ [REDACTED]

Within the NoPwnZone folder you have a malware definitions file v1801.
Run YARA to check all of our executables for malware.
```

When we run that command, we only get two .exe files. There is a .exe file named “Great App” shows a scary face. It is a malicious software which did not show up in the above picture because “virusdefs\_v1801.txt” doesn’t contain the required data or the malicious content to identify the “Great App”.



We need to update our rules file. Even in our pc the Anti-virus gets updated regularly so that it does not allow malicious software to run in our pc. We already have the updated rules file named as “virusdefs\_v2023.txt”. Now run the yara command given below to check for malicious software.

```
yara -r virusdefs_v2023.txt Executables/
```

```
sushmanth@Sushmanth: ~/Cryptography/NoPwnZone
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ ls
Executables  virusdefs_v1801.txt  virusdefs_v2023.txt
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ yara -r virusdefs_v2023.txt Executables/
MalwareB Executables//ToTest/SimpleApp.exe
MalwareC_ScaryFace Executables//ToTest/GreatApp.exe
RansomwareA Executables//KnownMalware/RansomwareSample.exe
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ Within the NoPwnZone folder you have a malware definitions file v1801.
Run YARA to check all of our executables for malware.
```

Now we find the GreatApp.exe file in the updated one which tell us that by updating our rules we are able to detect the “Great App” file.

## Exercise 6 – LET’S GO THREAT HUNTING

In this exercise, we are going to find malware specimens which are using same public encryption key. For this, first we have entered rader2 command prompt and go to KnownMalware directory and run the following command “radare2 RansomwareSample.exe”. Now RansomwareSample.exe loaded with the current address at a prompt. Running “?” command shows us the help options.

```
sushmanth@Sushmanth:~/Cryptography/NoPwnZone/Executables/KnownMalware$ radare2 RansomwareSample.exe
[0x0040166a]> ?
Usage: [ .][times][cmd][~grep][a[@iter]addr!size][|>pipe] ; ...
Append '?' to any char command to get detailed help
Prefix with number to repeat command N times (f.ex: 3x)
?var=value      available command alias for "env" command
*[?]
off[=0x]value   pointer read/write data/values (see ?pv, ?wx, ?wv)
(macro arg0 arg1) in Once
.[-](m)f!lsh|cmd Define macro or load r2, cparsse or rlang file
[?]
[?]
[cmd] 2.Using Terminal send/listen for remote commands (tcp://, raps://, udp://, http://, <fd>)
<...>           radare2 push escaped string into the RCons.readchar buffer
[?]
You will have a search for bytes, regexps, patterns, ... will enter radare2
[?]
[cmd] commands. run given command as in system(3)
#[] !lang [...] Hashbang to run an rlang script
a[] analysis commands
b[] KnownMalware/RansomwareSample.exe
c[] [arg] compare block with given data
C[] code metadata (comments, format, hints, ...)
d[] debugger commands
e[] [a[=b]] list/get/set config evaluable vars
f[] [name][sz][at] add flag at current address
g[] [arg] generate shellcodes with r_egg
i[] [file] get info about opened file from r_bin
k[] [sdb-query] run sdb-query, see k? for help, 'k *', 'k **' ...
l[] [filepattern] list files and directories
L[] [-] [plugin] list, unload load r2 plugins
m[] mountpoints commands
o[] [file] ([offset]) open file at optional address
p[] [len] print current block with format and length
P[] project management utilities
q[] [ret] quit program with a return value
```

Next run the radare2 command ‘aaa’ to analyse all and autoname functions.

```
[0x0040166a]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x0040166a]>
```

Run the radare2 command ‘ii’ to show imports. This will show us what ‘external’ Functions the malware specimen is using.

```
[0x0040166a]> ii
[Imports]
nth vaddr bind type lib name
1 0x0040d034 NONE FUNC KERNEL32.dll HeapSize
2 0x0040d038 NONE FUNC KERNEL32.dll GetProcessHeap
3 0x0040d03c NONE FUNC KERNEL32.dll LCMpStringW
4 0x0040d040 NONE FUNC KERNEL32.dll HeapReAlloc
5 0x0040d044 NONE FUNC KERNEL32.dll GetStringTypeW
6 0x0040d048 NONE FUNC KERNEL32.dll GetFileType
7 0x0040d04c NONE FUNC KERNEL32.dll SetStdHandle
8 0x0040d050 NONE FUNC KERNEL32.dll FlushfileBuffers
9 0x0040d054 NONE FUNC KERNEL32.dll GetConsoleCP
10 0x0040d058 NONE FUNC KERNEL32.dll GetConsoleMode
11 0x0040d05c NONE FUNC KERNEL32.dll SetFilePointerEx
12 0x0040d060 NONE FUNC KERNEL32.dll CreateFileW
13 0x0040d064 NONE FUNC KERNEL32.dll CloseHandle
14 0x0040d068 NONE FUNC KERNEL32.dll WriteConsoleW
15 0x0040d06c NONE FUNC KERNEL32.dll GetModuleHandleW
16 0x0040d070 NONE FUNC KERNEL32.dll LocalAlloc
17 0x0040d074 NONE FUNC KERNEL32.dll QueryPerformanceCounter
18 0x0040d078 NONE FUNC KERNEL32.dll GetCurrentProcessId
19 0x0040d07c NONE FUNC KERNEL32.dll GetCurrentThreadId
20 0x0040d080 NONE FUNC KERNEL32.dll GetSystemTimeAsFileTime
21 0x0040d084 NONE FUNC KERNEL32.dll InitializeSListHead
22 0x0040d088 NONE FUNC KERNEL32.dll IsDebuggerPresent
23 0x0040d08c NONE FUNC KERNEL32.dll UnhandledExceptionFilter
24 0x0040d090 NONE FUNC KERNEL32.dll SetUnhandledExceptionFilter
25 0x0040d094 NONE FUNC KERNEL32.dll GetStartupInfoW
26 0x0040d098 NONE FUNC KERNEL32.dll IsProcessorFeaturePresent
27 0x0040d09c NONE FUNC KERNEL32.dll GetCurrentProcess
28 0x0040d0a0 NONE FUNC KERNEL32.dll TerminateProcess
```

Run the radare2 ‘axt’ command to analyse x-references to the address of CryptAcquireContextA

```
[0x0040166a]> axt 0x0040d000
main 0x4010b5 [CALL] call dword [sym.imp.ADVAPI32.dll_CryptAcquireContextA]
[0x0040166a]>
```

The output shows that the function is being called from the main() function, at address 0x4010b5. Now we run “s” command to seek to the main() function of the program and we use “pdfs” command to print a function summary of the main function.

```
[0x0040166a]> s main
[0x00401000]> pdfs
-- section:.text:
-- eip:
0x00401000 [00] -r-x section size 49152 named .text
0x00401007 sym.imp.CRYPT32.dll_CryptStringToBinaryA
0x0040101d str.BEGIN_PUBLIC_KEY _____MIIBIjANBgkqhkiG9wBAQEEFAACQ8AMIIIBCgKCAQEAA0mJMtHJ48QoAlJe4_cMT0FhPOupjCHDnPXDC7P3UNShZByD
ikVc0geeRng0syQc10zxTySSa2TzY46x3WAUDhRc3702CJ_o8V3NVPf2uegh6QYXhaHveGJ9Zi0TowFJ_nnAuq7zrWAszz8wP700vXHgLvZ_xIrswS9HR2vcQ9RAU_
yc6vbUu4MAVL40mq3Vo_t9ZuBukOndPhsbkM_dQZ1N8Io0FUANO9Kzis28We0Cd_gsZbJzFTxdHzi57obj_rGuJX_NzCUZP8sUi0_YfLdqPvZsOB0TV3UrJNRcu_NE3
yqbgw9s_3D1FBs86tqzjig6PukMTO_aGwIDAQAB____END_PUBLIC_KEY_____
0x00401022 call esi
0x0040102a sym.imp.KERNEL32.dll_LocalAlloc
0x00401032 call edi
0x00401042 str.BEGIN_PUBLIC_KEY _____MIIBIjANBgkqhkiG9wBAQEEFAACQ8AMIIIBCgKCAQEAA0mJMtHJ48QoAlJe4_cMT0FhPOupjCHDnPXDC7P3UNShZByD
ikVc0geeRng0syQc10zxTySSa2TzY46x3WAUDhRc3702CJ_o8V3NVPf2uegh6QYXhaHveGJ9Zi0TowFJ_nnAuq7zrWAszz8wP700vXHgLvZ_xIrswS9HR2vcQ9RAU_
yc6vbUu4MAVL40mq3Vo_t9ZuBukOndPhsbkM_dQZ1N8Io0FUANO9Kzis28We0Cd_gsZbJzFTxdHzi57obj_rGuJX_NzCUZP8sUi0_YfLdqPvZsOB0TV3UrJNRcu_NE3
yqbgw9s_3D1FBs86tqzjig6PukMTO_aGwIDAQAB____END_PUBLIC_KEY_____
0x0040104c call esi
0x0040104e sym.imp.CRYPT32.dll_CryptDecodeObjectEx
0x00401072 call esi
0x0040107c call edi
0x004010a0 call esi
0x004010a2 DWORD dwFlags
0x004010a7 DWORD dwProvType
0x004010a9 LPCSTR szProvider
0x004010a9 str.Microsoft_Enhanced_Cryptographic_Provider_v1.0
0x004010ae LPCSTR szContainer
0x004010b0 HCRYPTPROV *phProv
0x004010b5 call dword [sym.imp.ADVAPI32.dll_CryptAcquireContextA]
0x004010bb HCRYPTKEY *phKey
0x004010c0 DWORD dwFlags
0x004010c2 HCRYPTKEY hPubKey
[0x0040166a]>
```

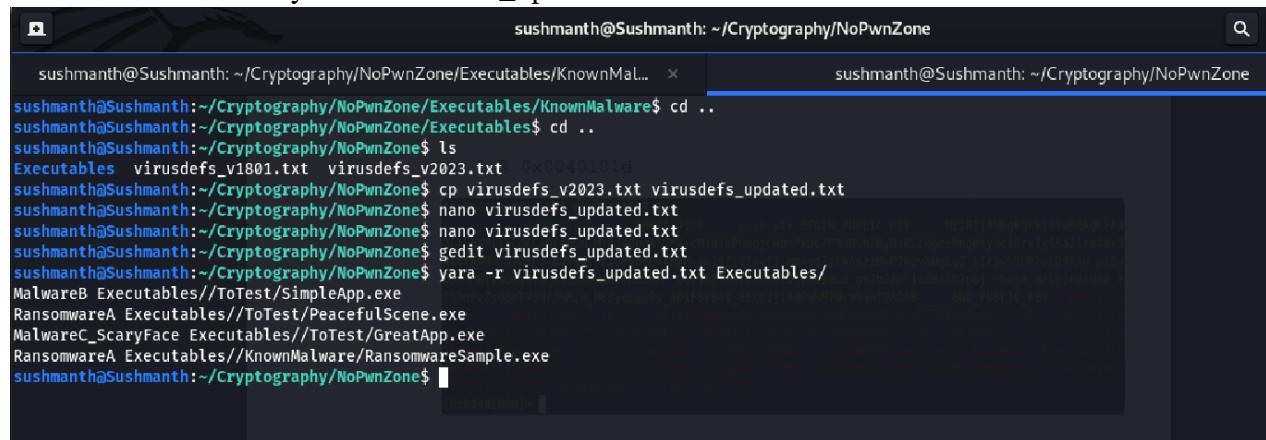
The public key is loaded by the “CryptStringToBinaryA” function. For seeing actual string, we run “pd 1 @ 0x00401042 1” command, radare2 disassembling the instruction shows us the actual text of the public key.



```
[0x00401000]> pd 1 @0x00401042 1
| 0x00401042  68601c4100 push str.BEGIN_PUBLIC_KEY_____MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBCgKCAQEAOmJNthJ48QoAlJe4_cMT0fhPOupjCHDnPXDC7P3UNShZBikVc0geeRng0syQc10zxTySSa2IYa46x3WAULDWHR370ZCJ_08V3NVPf2uegh6QYXhaHveGJ9Zi0TowfJ_mnAuq7zrWA5zz8wP700vXHgLvZ_xIrsW59HR2vcQ9RAU_yc6vbUu4MAVl40mq3Vo_t9ZuBuk0WdPhsbkM_dQZ1N8IoOfUAN0Kz1s28We0Cd_gsZbJzFTixdHzi57obj_rGuJX_NzCUZP8sUi0_YflDqPvzsOB0tV3UrJNRcu_NE3yqbgw9s_3D1FBsB6I_86tqzji6gPukMTO_aGawIDAQAB_____END_PUBLIC_KEY ; 0x411c60 ; ----- BEGIN PUBLIC KEY-----MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBCgKCAQEAOmJNthJ48QoAlJe4+cMT0fhPOupjCHDnPXDC7P3UNShZBikVc0geeRng0syQc10zxTySSa2IYa46x3WAULDWHR370ZCJ+08V3NVPf2uegh6QYXhaHveGJ9Zi0TowfJ+mnAuq7zrWA5zz8wP700vXHgLvZ+xIrsW59HR2vcQ9RAU/yc6vbUu4MAVl40mq3Vo+t9ZuBuk0WdPhsbkM+/dQZ1N8IoOfUAN0Kz1s28We0Cd+gsZbJzFTixdHzi57obj/rGuJX+NzCUZP8sUi0/YflDqPvzsOB0tV3UrJNRcu/NE3yqbgw9s/3D1FBsB6I+86tqzji6gPukMTO+aGawIDAQAB-----END PUBLIC KEY-----
```

The highlighted data is the public key. Now we copy this string and add it to virusdefs\_updated.txt file. Now run the yara command given below:

```
yara -r virusdefs_updated.txt Executables/
```



```
sushmanth@sushmanth: ~/Cryptography/NoPwnZone
sushmanth@Sushmanth:~/Cryptography/NoPwnZone/Executables/KnownMal... x
sushmanth@Sushmanth:~/Cryptography/NoPwnZone/Executables/KnownMalware$ cd ..
sushmanth@Sushmanth:~/Cryptography/NoPwnZone/Executables$ cd ..
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ ls
Executables virusdefsv1801.txt virusdefsv2023.txt 0x0040101d
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ cp virusdefsv2023.txt virusdefsv_updated.txt
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ nano virusdefsv_updated.txt
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ nano virusdefsv_updated.txt
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ gedit virusdefsv_updated.txt
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$ yara -r virusdefsv_updated.txt Executables/
MalwareB Executables//ToTest/SimpleApp.exe
RansomwareA Executables//ToTest/PeacefulScene.exe
MalwareC_ScaryFace Executables//ToTest/GreatApp.exe
RansomwareA Executables//KnownMalware/RansomwareSample.exe
sushmanth@Sushmanth:~/Cryptography/NoPwnZone$
```

The result shows the original RansomwareSample.exe and another file “PeacefulScene.exe” is added to ToTest folder.

Question: Is matching on a public key a reasonable way to detect ransomware?

Answer: yes, it is a reasonable way.

Question: If the cyber criminals discovered that their ransomware was being detected by anti-Virus software, how could they create a new variant of RansomwareA to evade our new Detection rule?

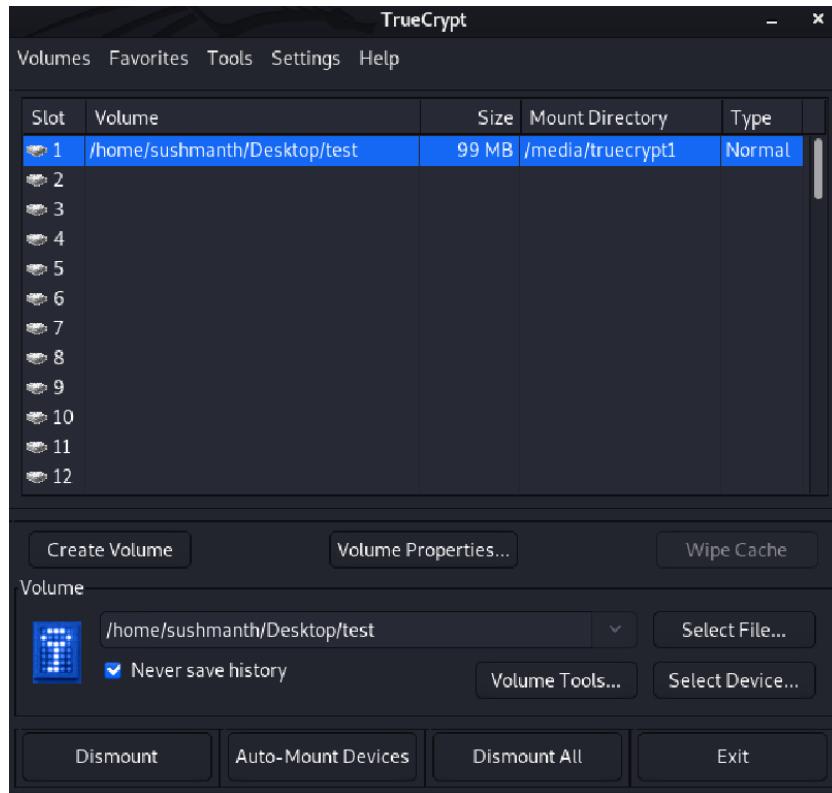
Answer: They create a new public key.

Question: Question: If you did not match on the whole public key, but merely matched on the string “BEGIN PUBLIC KEY”, would that cause problems?

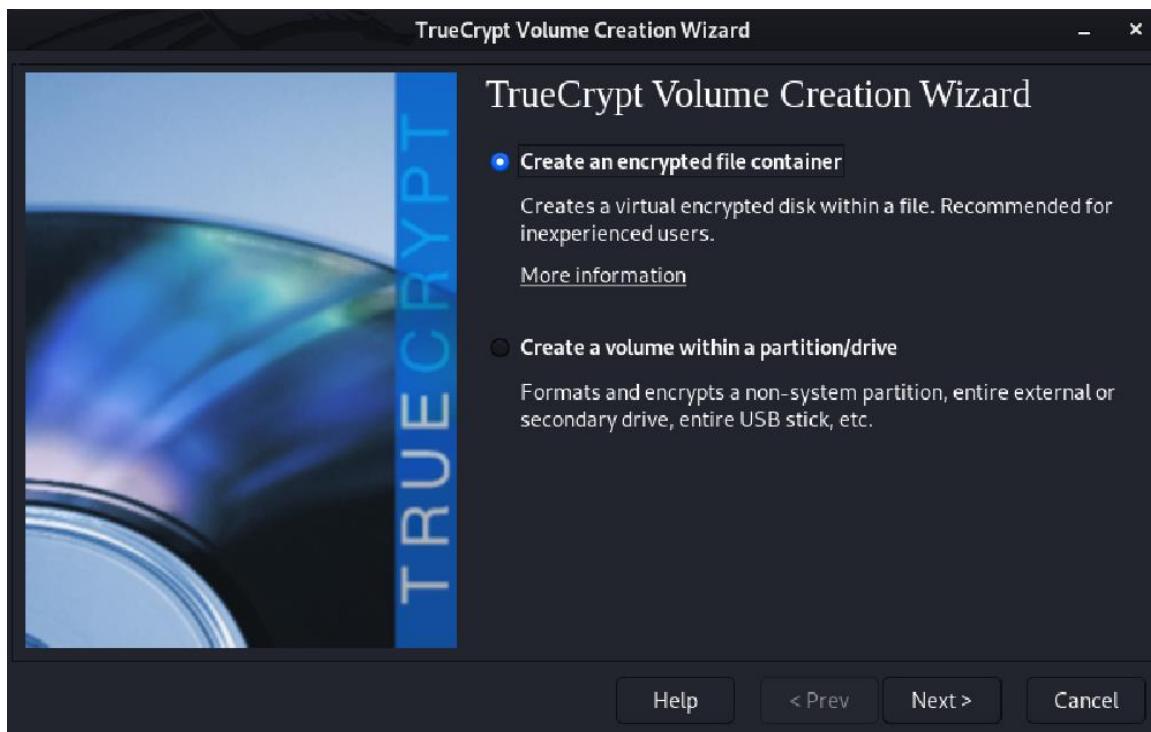
Answer: yes, this will cause problems because if the whole key did not match, we will have a lot results which are not malicious so the real malicious file may get ignored.

## PART – 2

Running the TrueCrypt software, which should result in image presented below.



To create a new volume, Select the 'Volumes' menu button in the top left-hand corner followed by 'Create New Volume'. The following page given below is opened.



As per the image above, you will notice that TrueCrypt provides three means by which you can encrypt a specific section of the hard disk. Using the documentation provided with TrueCrypt, you should familiarise yourself with the different forms of encryption that can be utilised. This will significantly benefit you in this unit. The option that we will be selecting as a part of this activity is 'Create an encrypted file container'.

**Question:** What is an encrypted file container?

**Answer:** This is a file which you can store other files inside. It is encrypted. You can only get the files inside with the right software, and the right password. When it is closed you can copy the container, rename it, delete it, or even attach it to an email message. An encrypted container is a virtual volume stored in an encrypted file. To gain access to the encrypted data (i.e. mount) you need to enter the correct password.

**Question:** How does the encrypted file container function?

**Answer:** Use them for protecting information that only you can access, or for sharing files securely. An encrypted container is much like any other file:

- you can copy it, rename it, move it to a different folder
- you can delete it by moving to the Recycle Bin or "Trash"
- you can copy the encrypted container onto a USB stick or portable storage device
- you could place it in shared file space, and allow other people access to it
- you could even attach it to an email message

When you open it, it decrypts and becomes a place just like a disk or a folder, where you can store or use files. Only people with the right software and password can open it.

**Question:** How does the encrypted file container differ from encrypting an entire system partition?

**Answer:** The container is a file, that can reside anywhere. You can copy or move it and the data inside remains secure. Encrypting the whole partition encrypts all content in that partition. To securely move something from there, you would need move it to another encrypted partition or container.

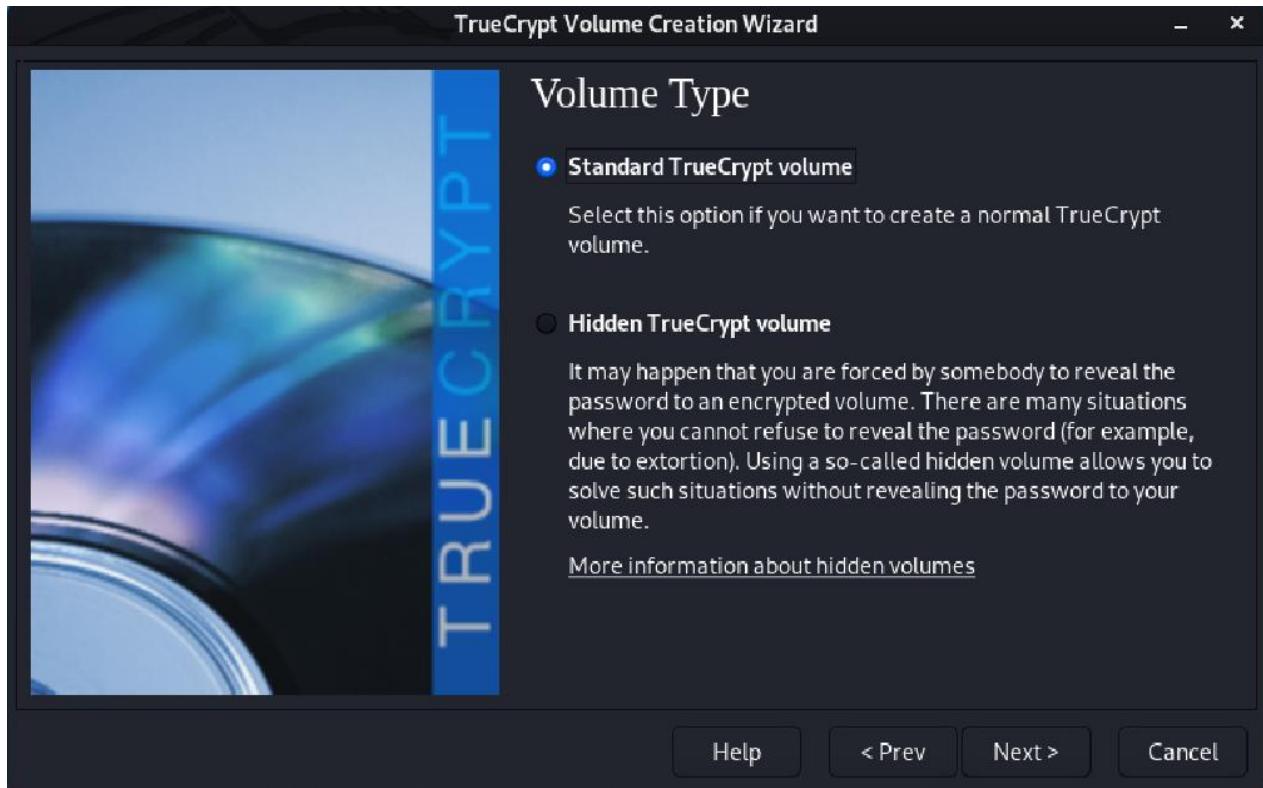
Performance is pretty much similar in the case of reads/ writes.

In the case of data loss, things get more complicated. Results really depend on the type of damage.

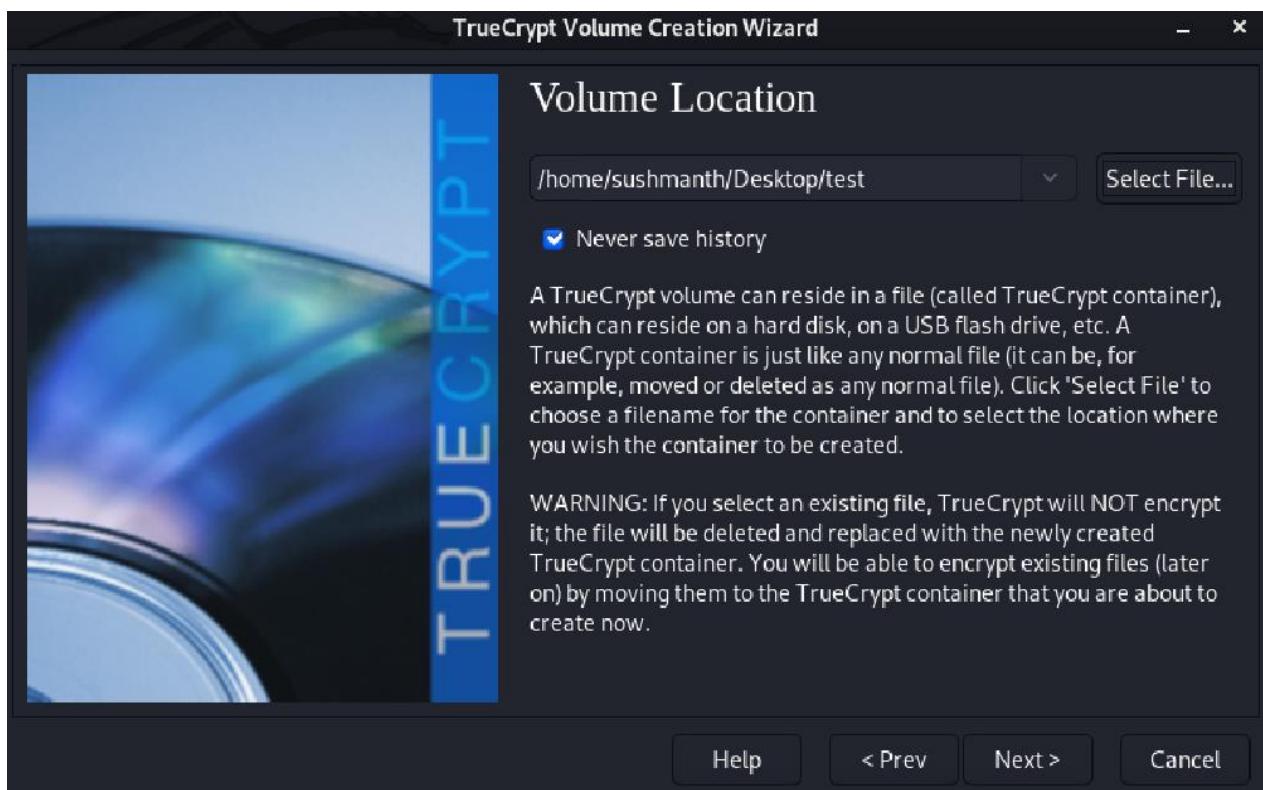
The container file is definitely easier to backup and restore, but has the disadvantage that if you have no backup system whatsoever and you damage the container file, there could be massive damage inside it or it even become unusable.

In the case of full encrypted partition, damage like bad clusters has less impact than it has on a file container, but it presents the disadvantage of total possible failure in case of various critical partition areas damage. To choose between the two you should consider what data you store there, how large is it, how much of it updates/changes and how do you intent to back it up.

Now, the steps involved in creating a standard TrueCrypt volume for this activity. Select Standard TrueCrypt volume.



Choose the path to be saved in



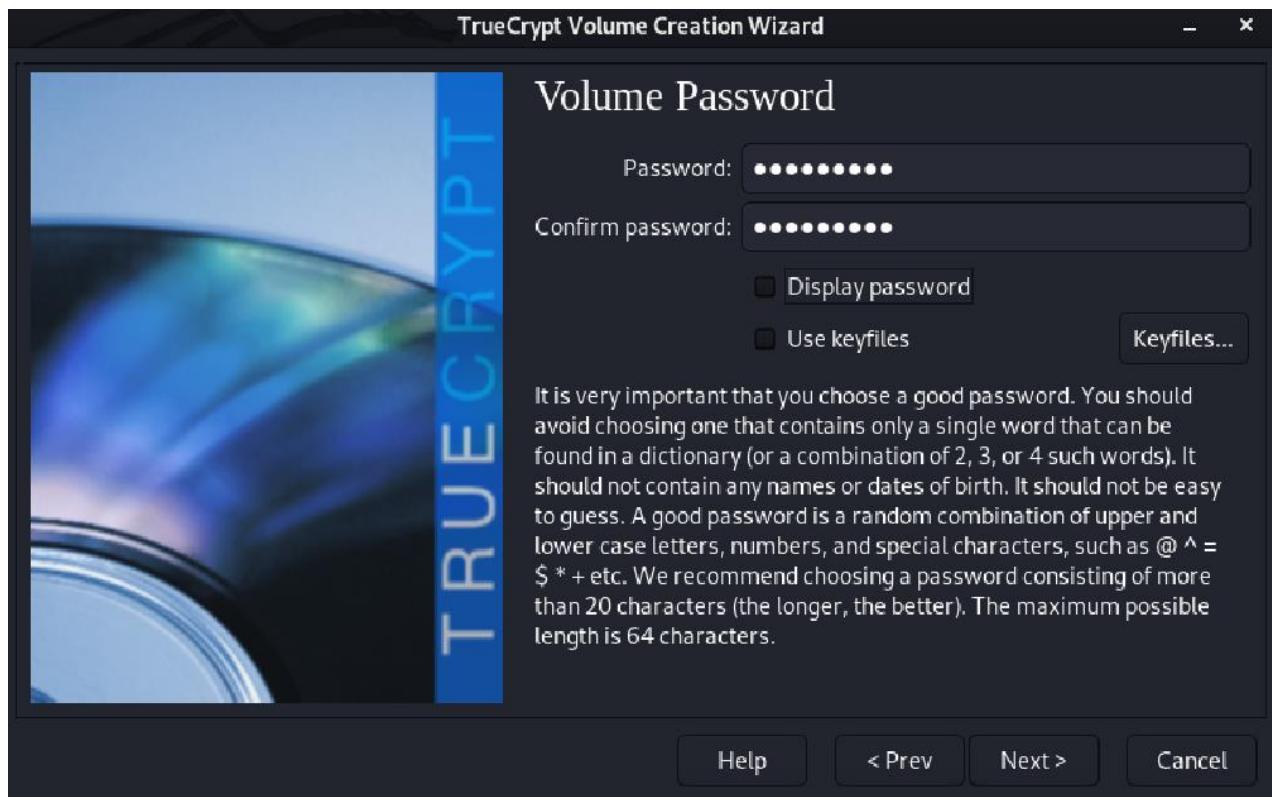
Select AES Encryption algorithm and click next.



Give the volume size. We can take it 100Mb.

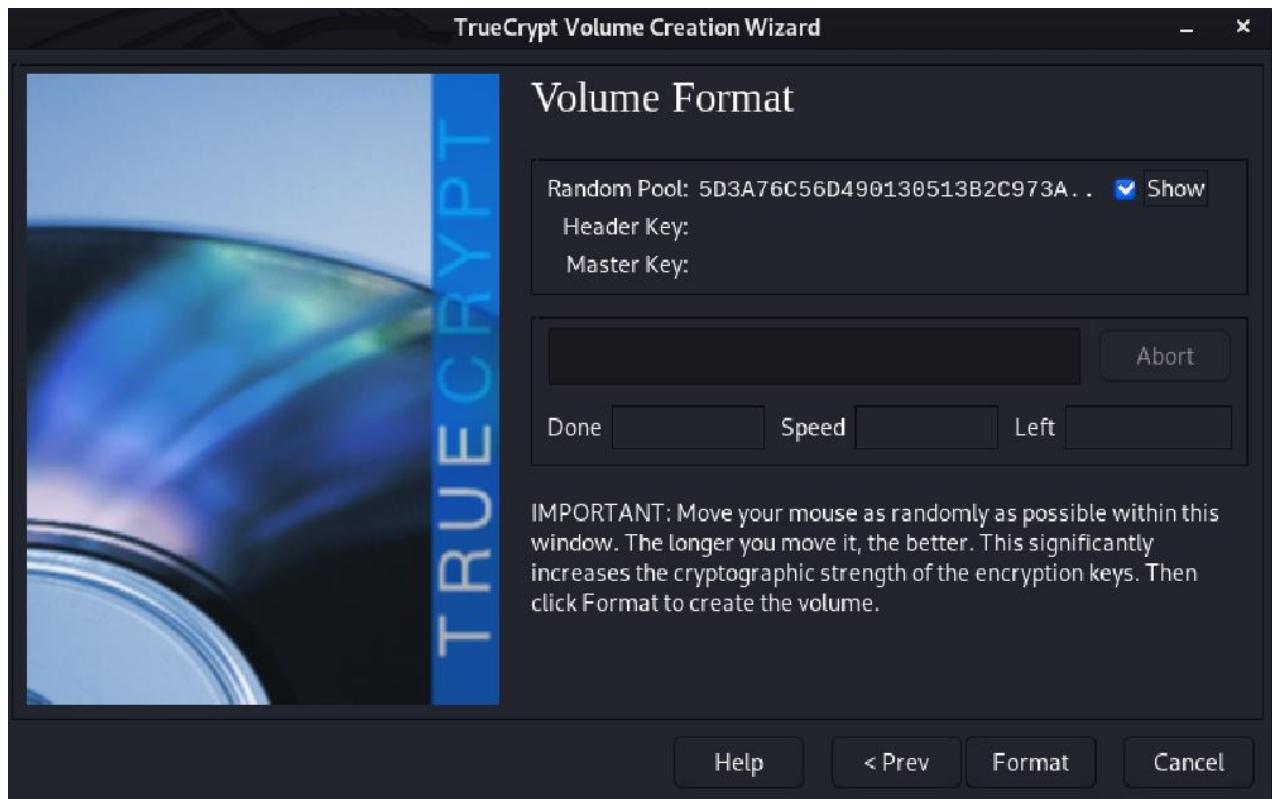


Give the Volume password in next page.

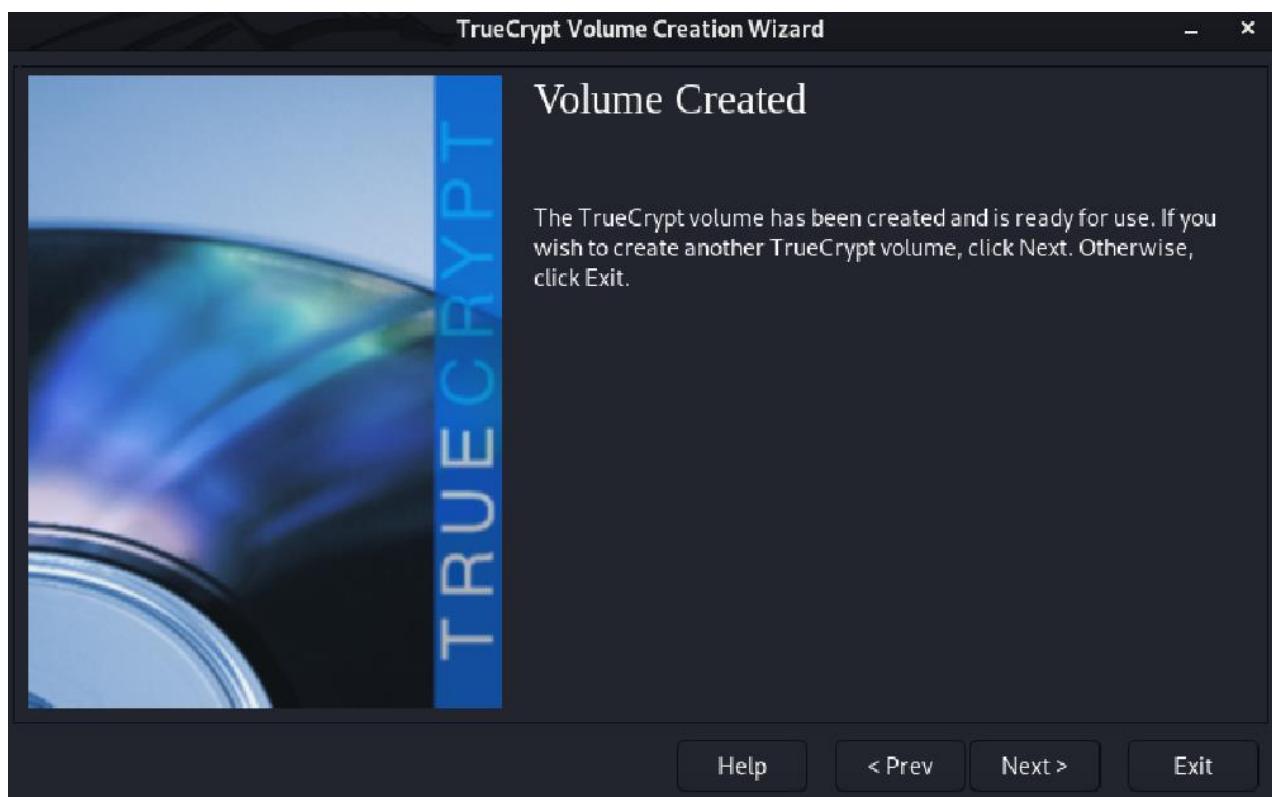


Choose the Format option in next page.





Then finally, Volume has created successfully.



Question: What happens to the drive (in My Computer, Computer or Windows Explorer) when it is dismounted?

Answer: It means that you are removing the storage area that was created on your physical hard drive from the operating system temporarily.

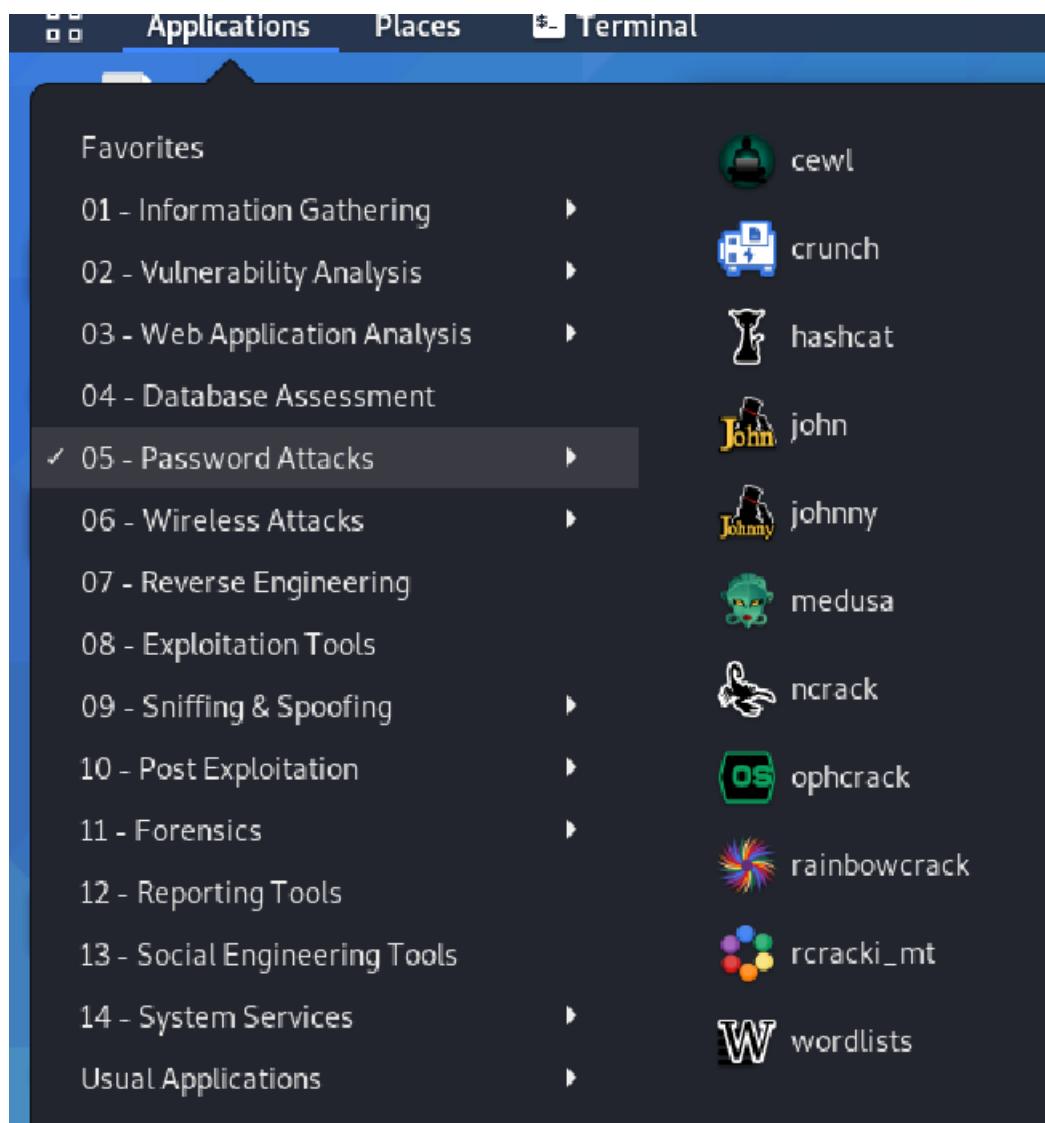
Question: Did the size of the TrueCrypt container file increase, decrease or remain the same after files were copied into it? What caused the container to react in this manner?

Answer: Initially the container size was 100 MB later after adding the files it has increased to 105MB. The size of the TrueCrypt container file increase after files were copied into it.

Question: If you delete the encrypted file container what will happen to the files inside?

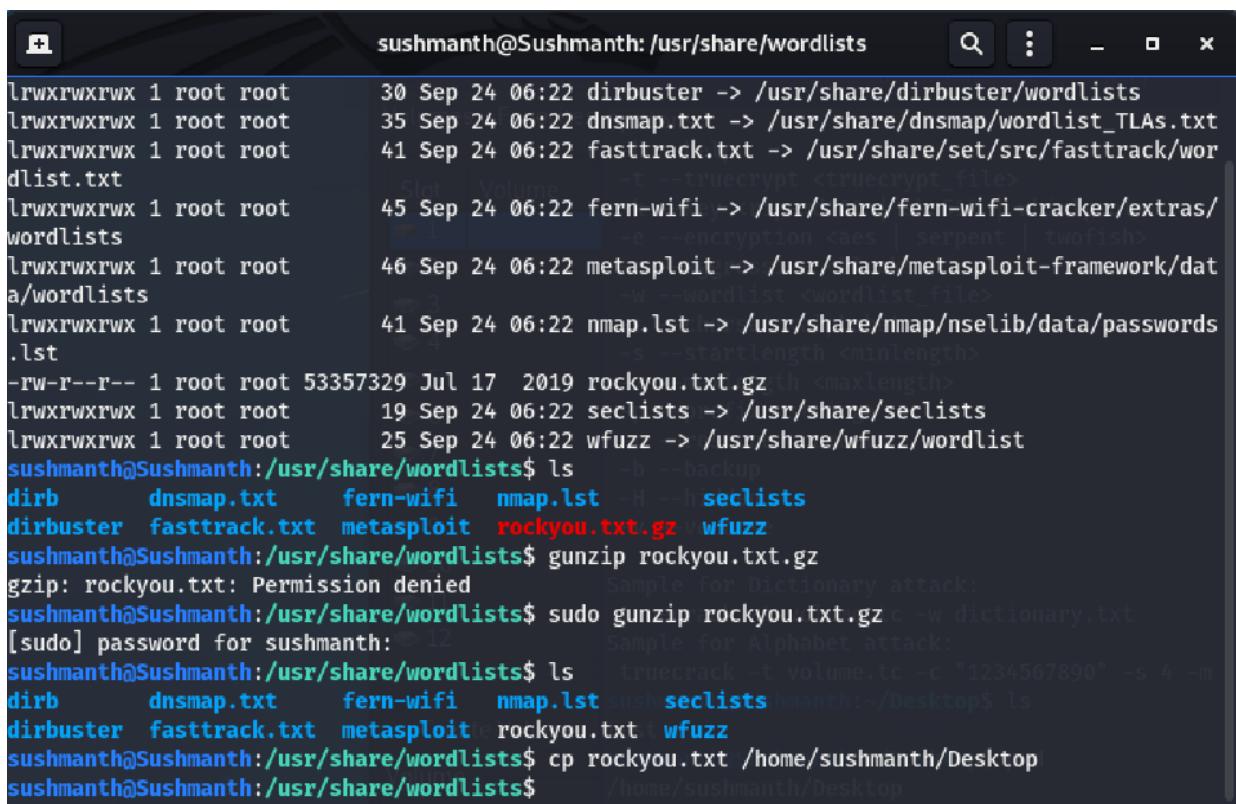
Answer: If you delete the encrypted file container then files inside the container will never be opened.

Copy the TrueCrypt container file from your host workstation onto the desktop within the Kali Linux virtual machine and open a Terminal window. (The container can be copied from the host desktop to Kali Linux desktop using drag and drop because we can set bidirectional option).



The approach that will be utilised to crack the TrueCrypt container will be through a dictionary attack approach. Fortunately, Kali Linux comes with a large wordlist that can be adopted for the purposes of carrying out the attack. Using the menu function click Applications > **05 – Password Attacks > wordlists**, as per the screenshot below. (Please note that wordlists can also be found under Applications > **05 – Password Attacks > Password Profiling & Wordlists**).

By following the above step a terminal window should pop up which automatically places you within the **/usr/share/wordlists** directory as shown in the graphical image below. The command ls -al will list the directory/folder contents. Depending on your previous usage of your Kali Linux virtual machine, the wordlist file may already be decompressed and stored as a rockyou.txt file. In contrast, if the rockyou file in the directory is being displayed with a .gz extension then it needs to be decompressed. Executing the command sudo gunzip **rockyou.txt.gz** will decompress the file (enter the password kali when requested).



A terminal window titled "sushmanth@Sushmanth:/usr/share/wordlists". The window shows a list of files in the directory:

```
sushmanth@Sushmanth: /usr/share/wordlists
ls
lrwxrwxrwx 1 root root    30 Sep 24 06:22 dirbuster -> /usr/share/dirbuster/wordlists
lrwxrwxrwx 1 root root    35 Sep 24 06:22 dnsmap.txt -> /usr/share/dnsmap/wordlist_TLAs.txt
lrwxrwxrwx 1 root root    41 Sep 24 06:22 fasttrack.txt -> /usr/share/set/src/fasttrack/wor
dlist.txt
lrwxrwxrwx 1 root root    45 Sep 24 06:22 fern-wifi -> /usr/share/fern-wifi-cracker/extras/
wordlists
lrwxrwxrwx 1 root root    46 Sep 24 06:22 metasploit -> /usr/share/metasploit-framework/dat
a/wordlists
lrwxrwxrwx 1 root root    41 Sep 24 06:22 nmap.lst -> /usr/share/nmap/nselib/data/passwords
.lst
-rw-r--r-- 1 root root 53357329 Jul 17 2019 rockyou.txt.gz
lrwxrwxrwx 1 root root    19 Sep 24 06:22 seclists -> /usr/share/seclists
lrwxrwxrwx 1 root root    25 Sep 24 06:22 wfuzz -> /usr/share/wfuzz/wordlist
sushmanth@Sushmanth: /usr/share/wordlists$ ls
dirb      dnsmap.txt   fern-wifi   nmap.lst   -H --h seclists
dirbuster  fasttrack.txt metasploit  rockyou.txt.gz  wfuzz
sushmanth@Sushmanth: /usr/share/wordlists$ gunzip rockyou.txt.gz
gzip: rockyou.txt: Permission denied
sushmanth@Sushmanth: /usr/share/wordlists$ sudo gunzip rockyou.txt.gz c -w dictionary.txt
[sudo] password for sushmanth: 
sushmanth@Sushmanth: /usr/share/wordlists$ ls
dirb      dnsmap.txt   fern-wifi   nmap.lst   -H --h seclists
dirbuster  fasttrack.txt metasploit  rockyou.txt  wfuzz
sushmanth@Sushmanth: /usr/share/wordlists$ cp rockyou.txt /home/sushmanth/Desktop
sushmanth@Sushmanth: /home/sushmanth/Desktop$
```

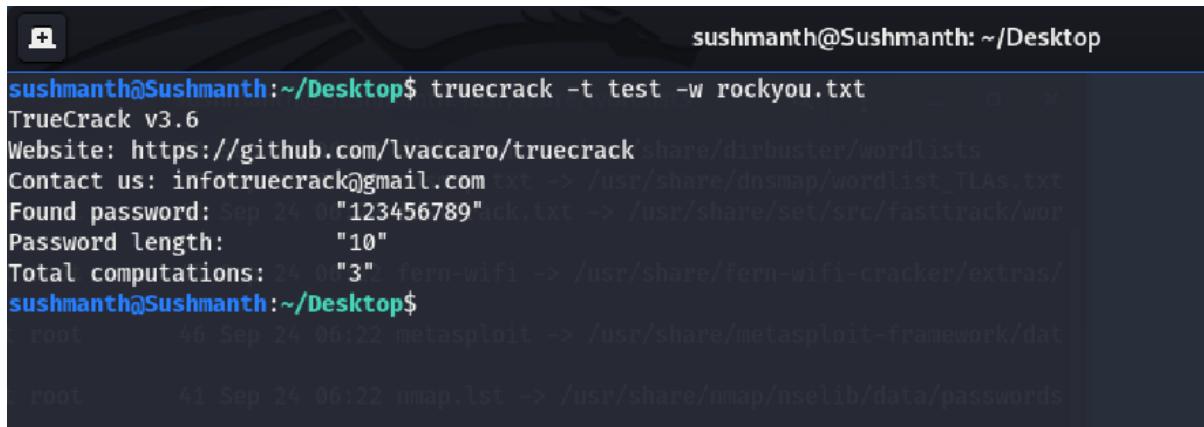
To simplify the cracking process, everything is going to be undertaken on the Desktop i.e. **/home/sushmanth/Desktop**. Thus the rockyou.txt file will need to be copied to the desktop using the following command **cp rockyou.txt.gz /home/root/Desktop/**.

Before Truecrack can be installed use the following commands to update Kali Linux

- **sudo apt-get update && sudo apt-get upgrade** (Enter kali as password)
- Please note this may take approximately 30-40 minutes. Depends upon internet speed

The Truecrack command can be directly executed within the terminal window as per the screenshot above which will display a list of usage functions and options that will enable the password cracking process. A very simple approach will be used to crack the password in this test case by executing the following command

Here After checking the Flags from the man page by using \$truecrack then will get to know the which flags are used.



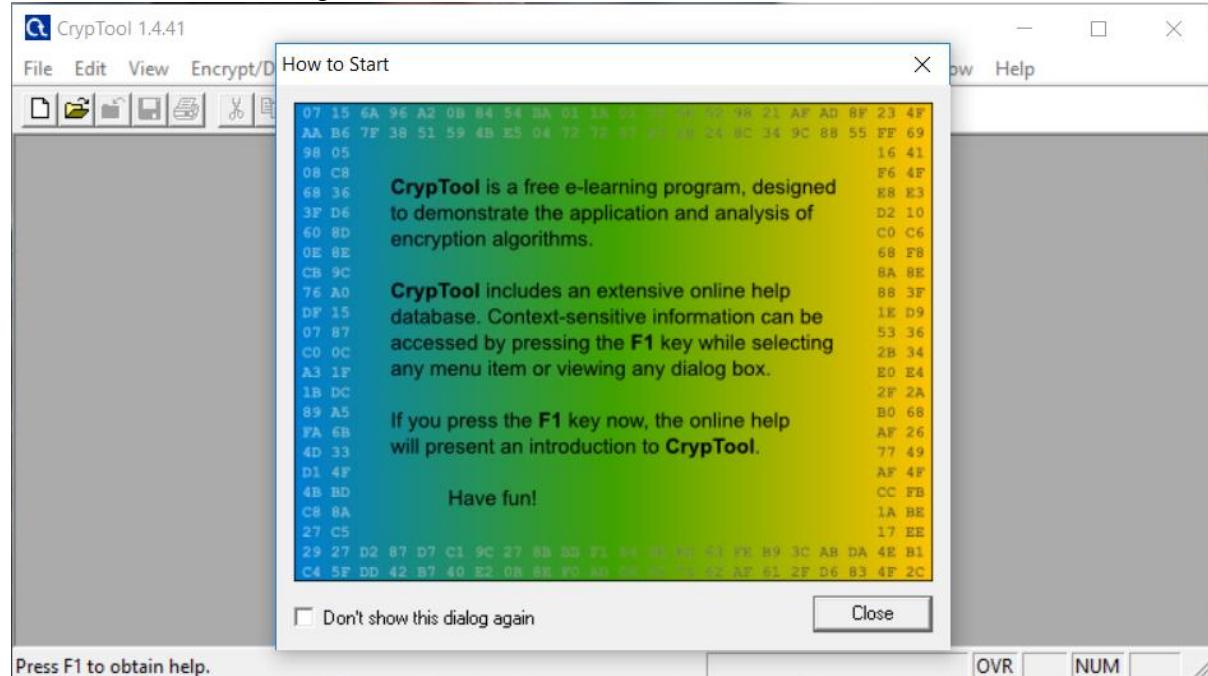
```
sushmanth@Sushmanth: ~/Desktop
sushmanth@Sushmanth:~/Desktop$ truecrack -t test -w rockyou.txt
TrueCrack v3.6
Website: https://github.com/lvaccaro/truecrack/share/dirbuster/wordlists
Contact us: infottruecrack@gmail.com
Found password: Sep 24 00:123456789"ack.txt -> /usr/share/dnsmap/wordlist_TLAs.txt
Password length: "10"
Total computations: 24 0 "3" fern-wifi -> /usr/share/fern-wifi-cracker/extras/
sushmanth@Sushmanth:~/Desktop$
root      46 Sep 24 06:22 metasploit -> /usr/share/metasploit-framework/dat
root      41 Sep 24 06:22 nmap.1st -> /usr/share/nmap/nselib/data/passwords
```

As per the preceding screenshot truecrack managed to successfully crack the password for the container file using the supplied wordlist. It is recommended that you repeat the process in this activity by creating alternate TrueCrypt container files using passwords of different strengths and by toggling the varying options within truecrack assessing the different results.

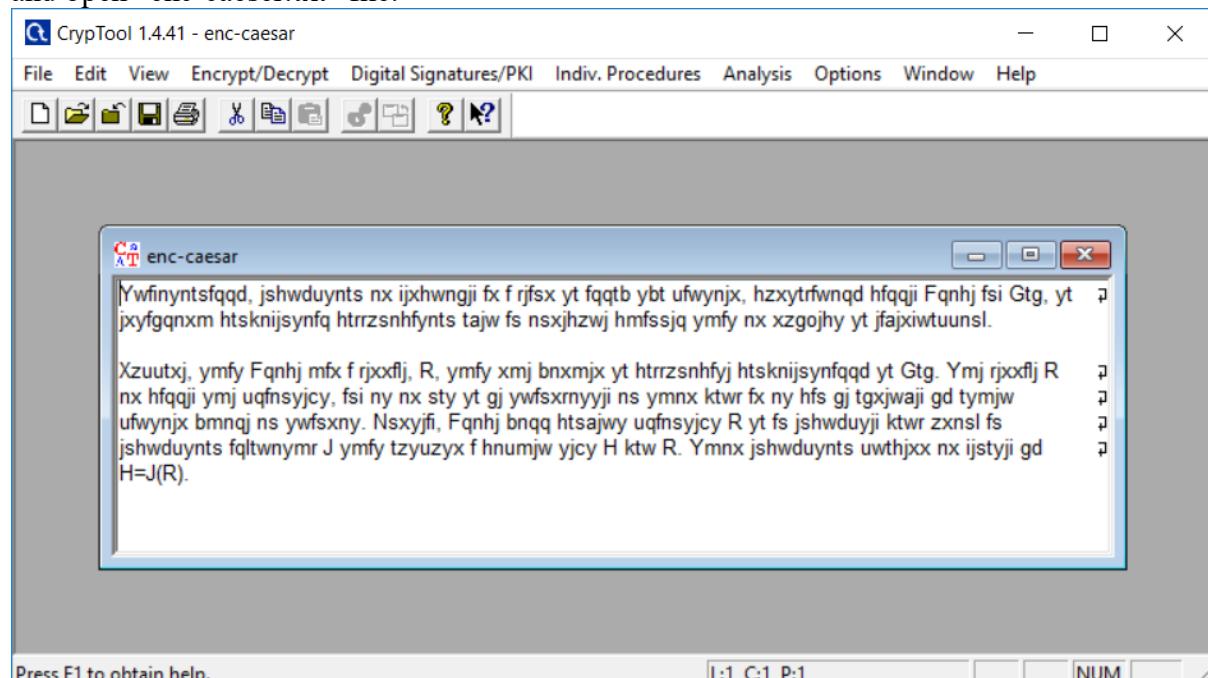
## PART – 3

In this part, we are going to use a cryptographic tool known as CrypTool. It is a tool for applying and analysing cryptographic algorithms.

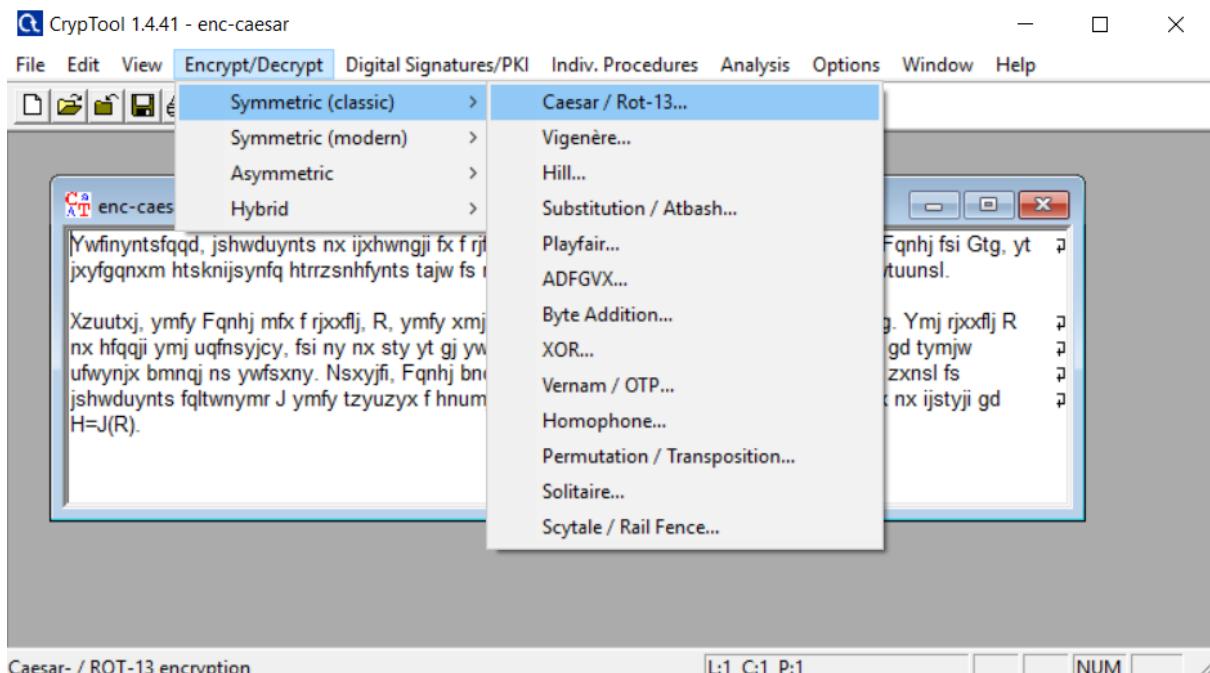
- 1) After installing it, open that tool. It shows a message that explain how to start CrypTool as shown in the below image.



- 2) Initially the start-up page displays a text file named as 'startingexample-en.txt'. Close that and open "enc-caesar.txt" file.



After opening, we observe that the content in that file is unreadable. Because the file is encrypted. To decrypt the message using Caesar cipher, we should choose Caesar/ROT-13 as shown in below image.



Question: How is the ciphertext-only attack being carried out?

Answer: In this method, the attacker has access to a set of ciphertexts. He does not have access to corresponding plaintext. COA is said to be successful when the corresponding plaintext can be determined from a given set of ciphertext.

Question: What is the advantage of using such an attack?

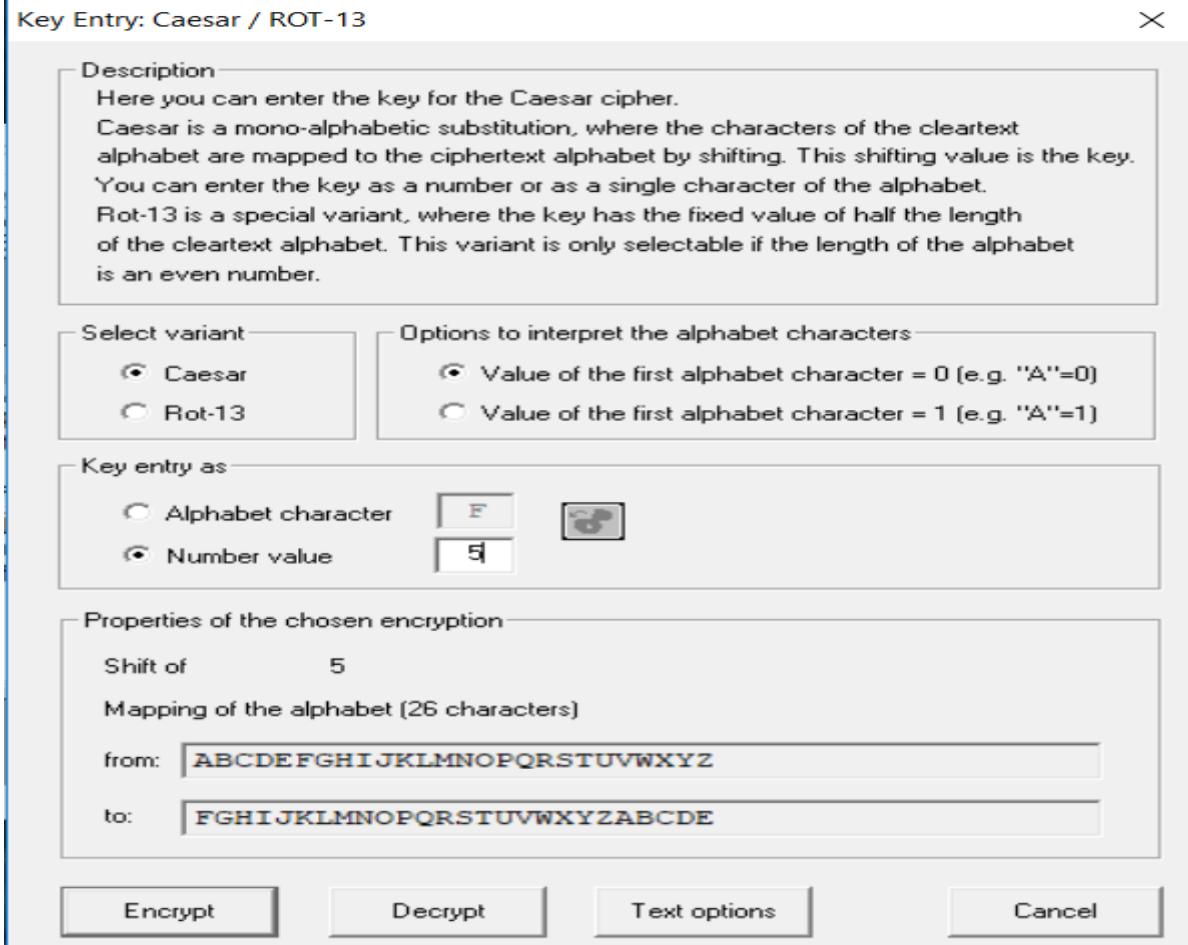
Answer: The patterns and context of the message is used to derive the contents of the message. So, less data is required to determine the plain text.

Question: What is the disadvantage of using such an attack?

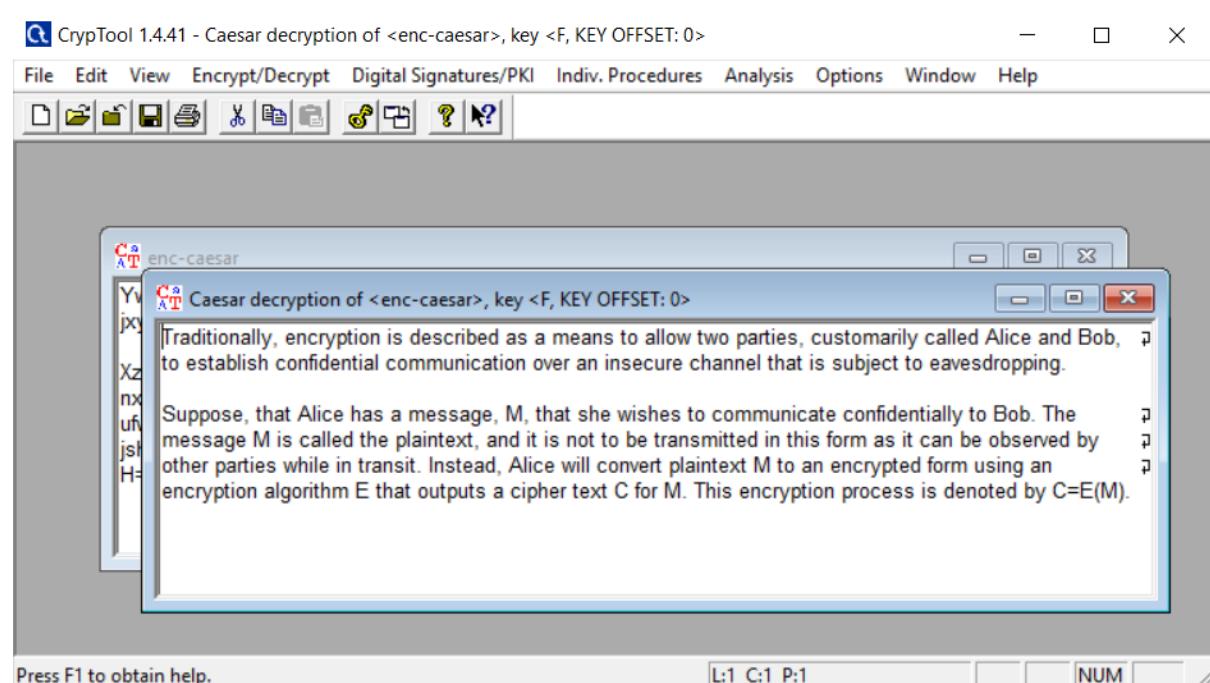
Answer: It is difficult to implement.

3) Now, a window will open after selecting Caesar/ROT-13. We should choose the options as per our requirements in the window.

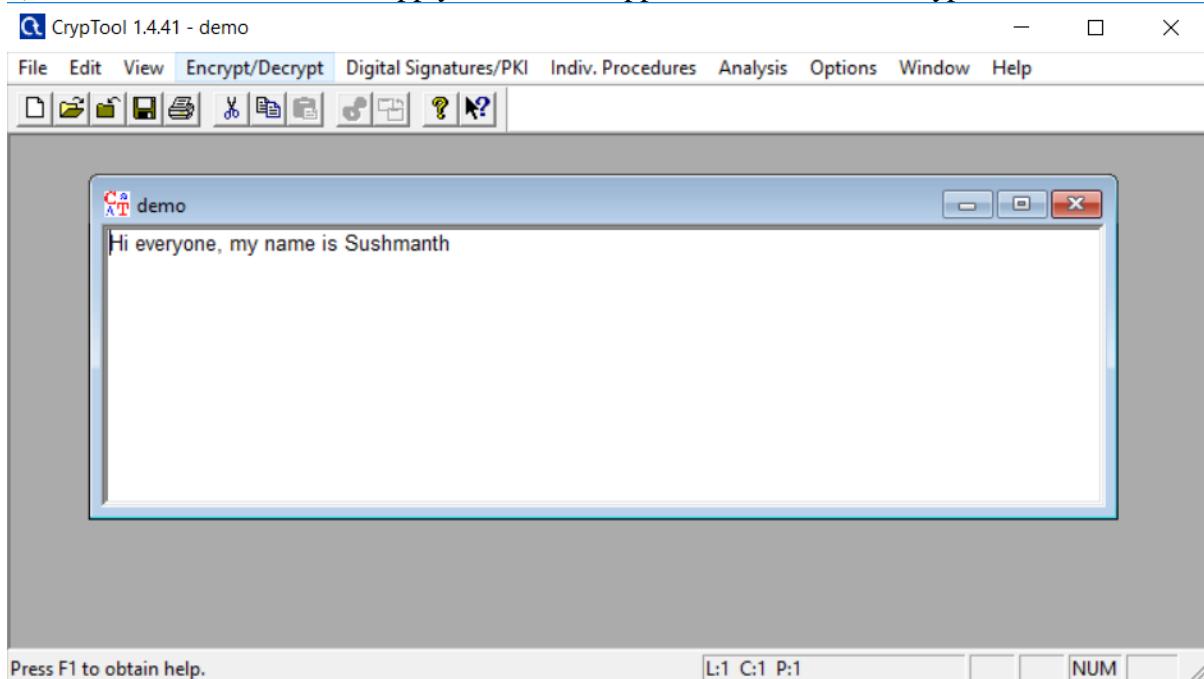
In this case, the index of the array of alphabets starts from 0. Here numerical value is known as key. After experimenting with some values finally I kept 5. So, the shift value of alphabets will be 5. Then we should click on decrypt.



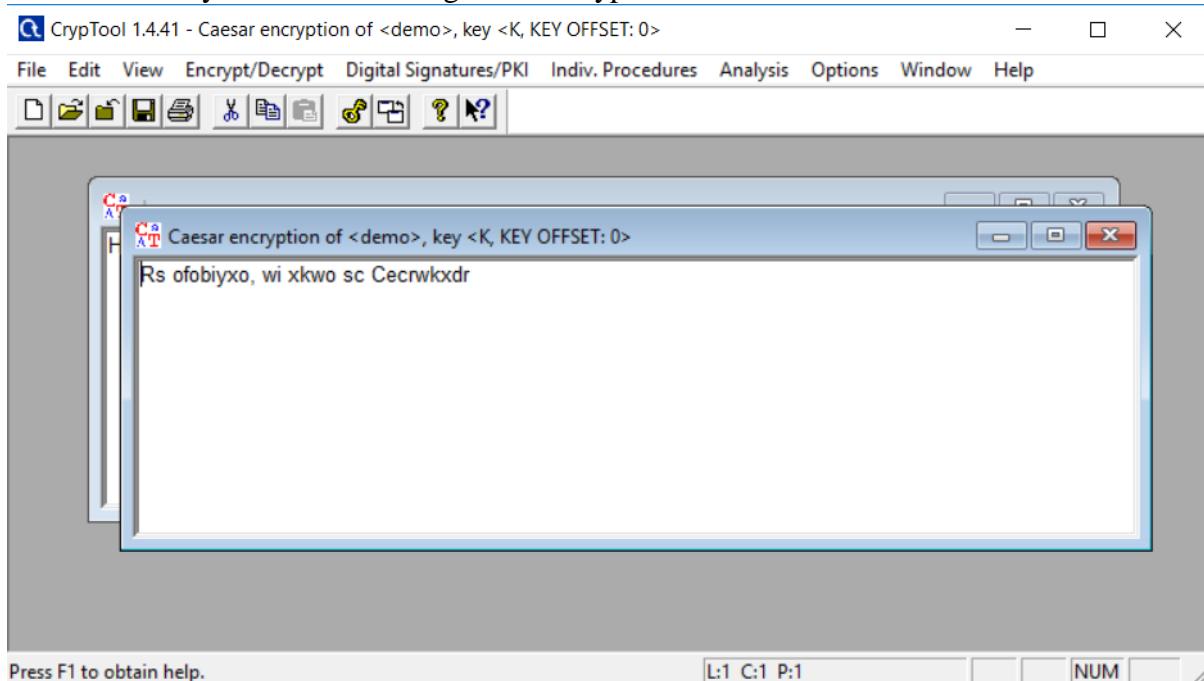
- 4) After clicking the Decrypt button with key 5, the text in “enc-caesar.txt” will get decrypted. Then we will get plain text that is readable.



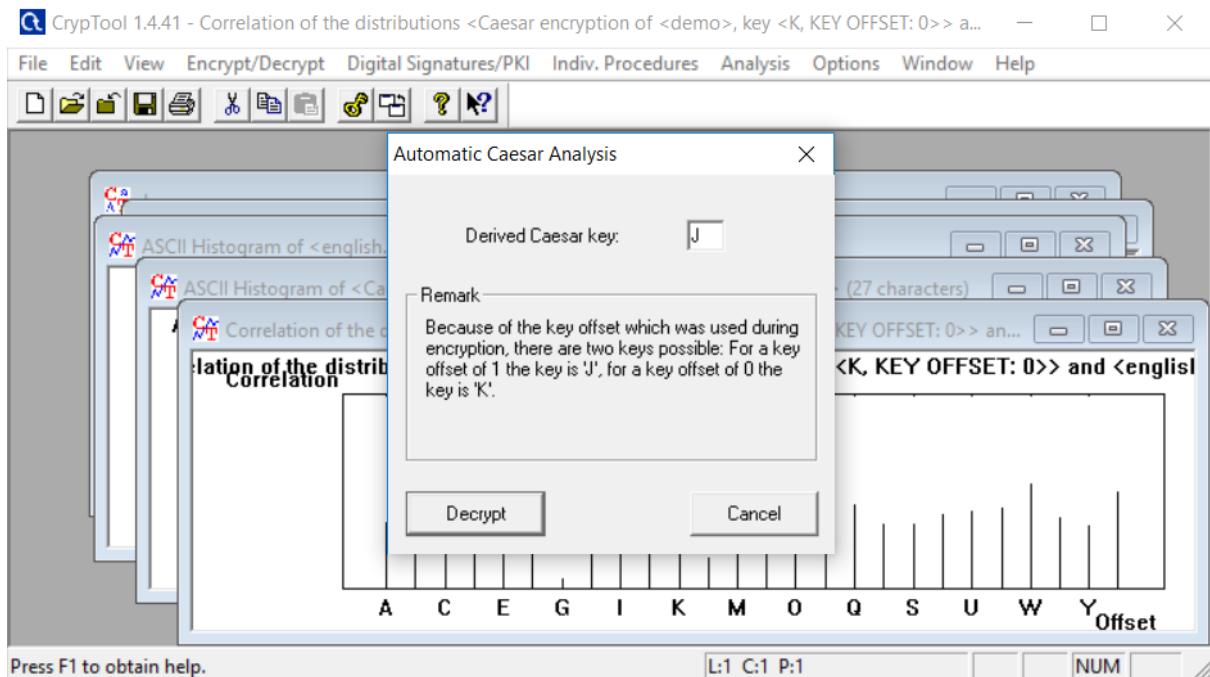
5) Next, I took a small text to apply brute-force approach. Hence the encrypted text is obtained.



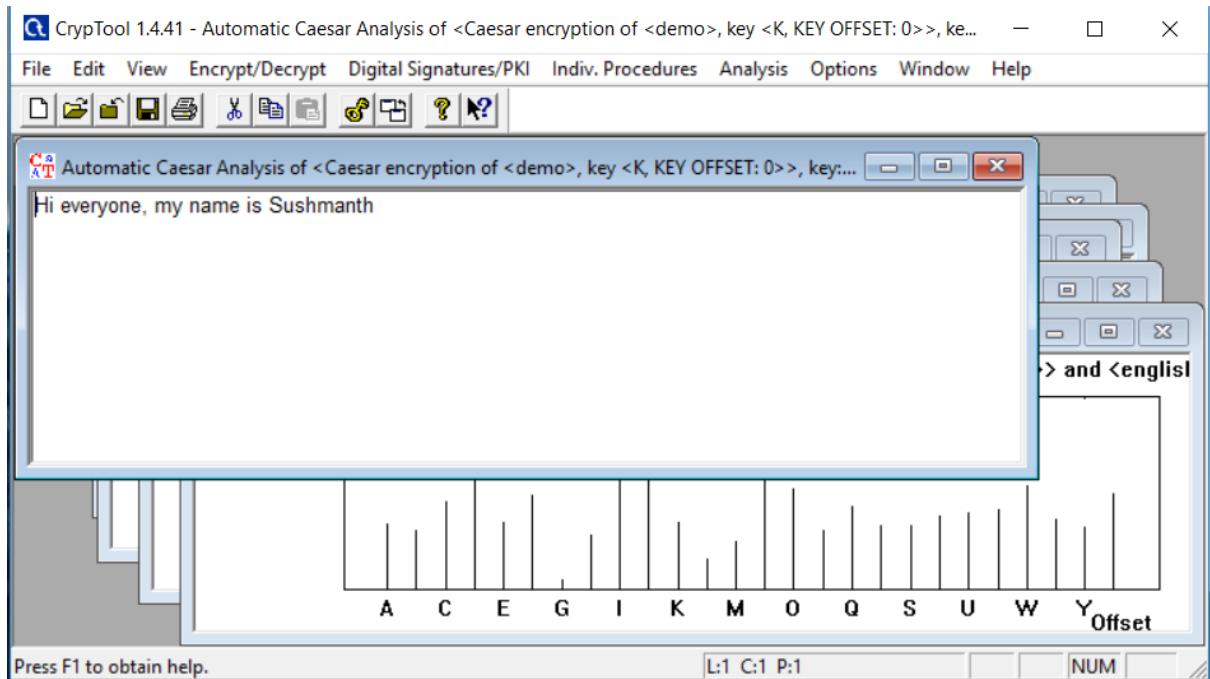
I have taken key value 10. Then I got the encrypted text.



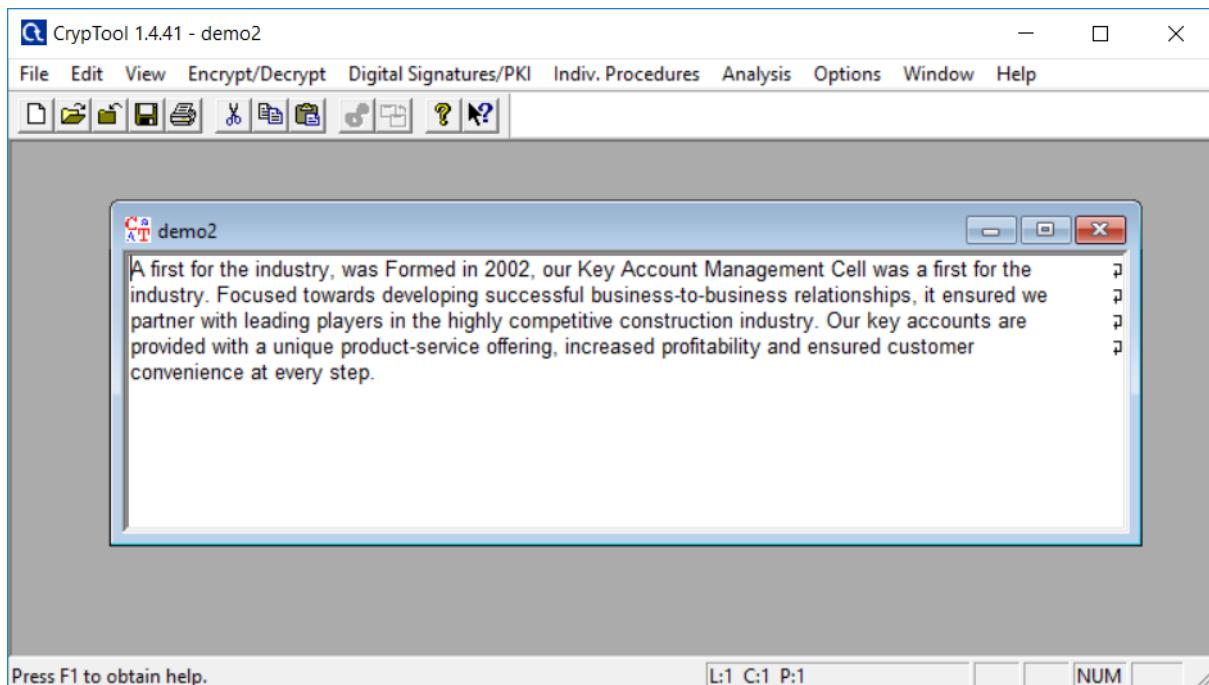
6) Here we are applying brute-force attack. Here the key is taken as 'J'. Hence after clicking on decrypt the entire cipher text gets decrypted.



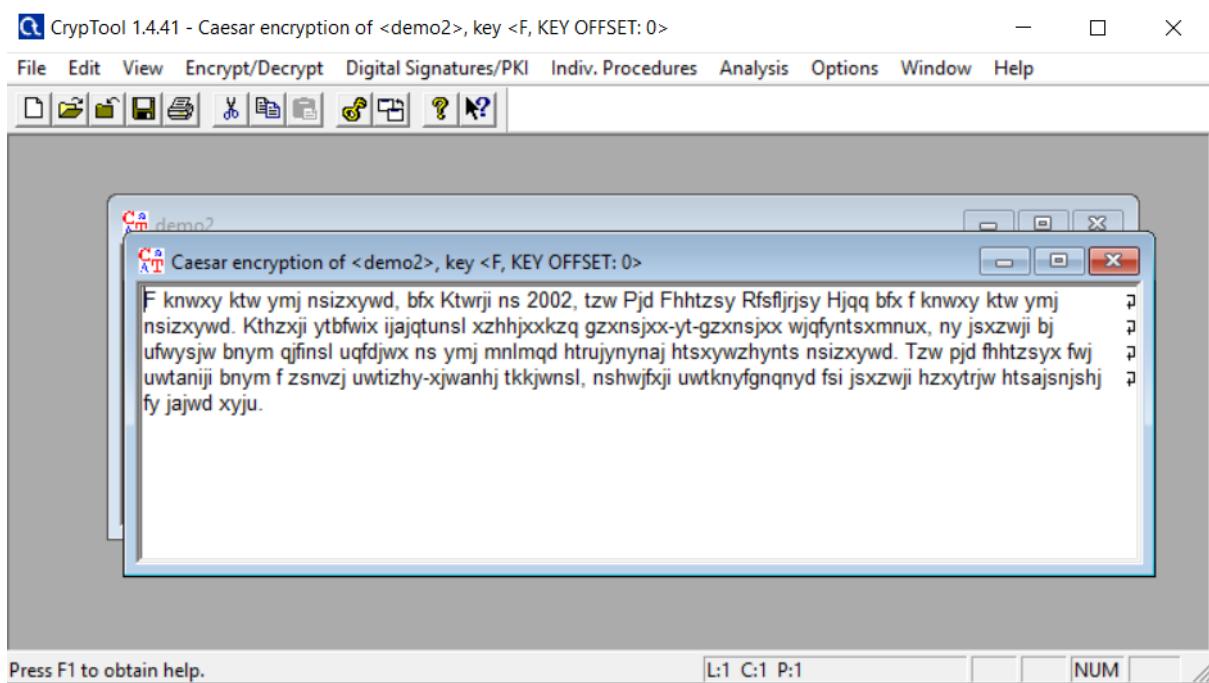
7) The decrypted text is shown below. The plain text will obtain.



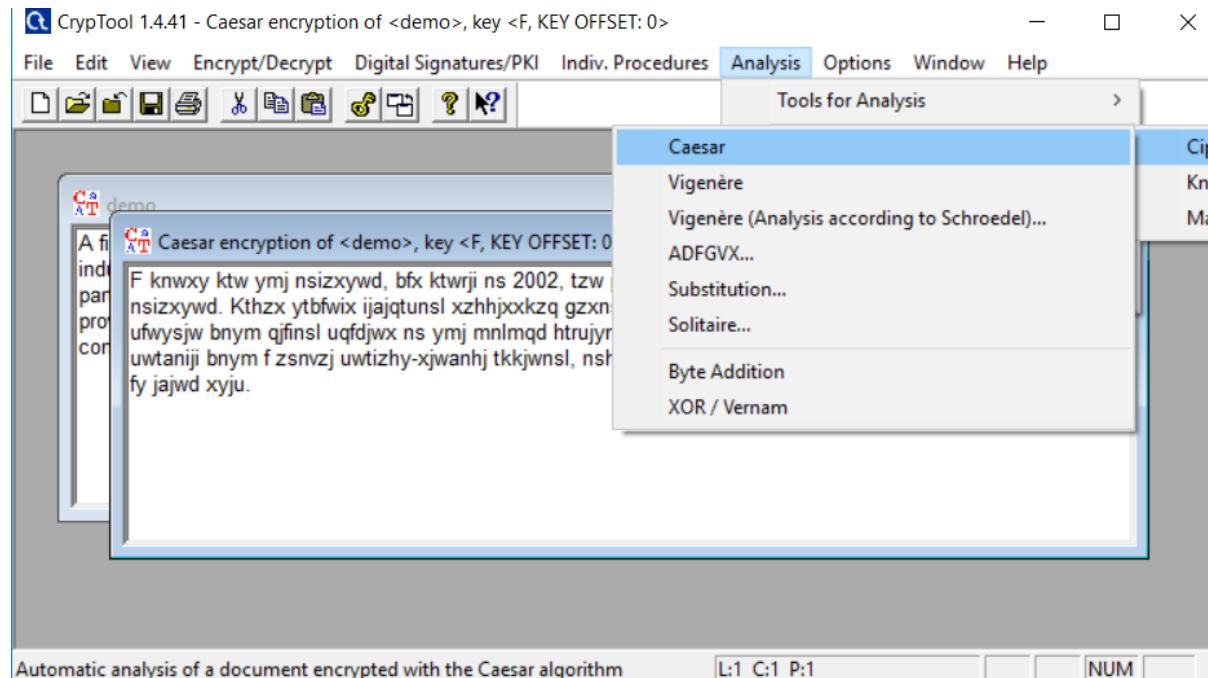
8) Now instead of taking a single line, we are taking a paragraph as a plain text.



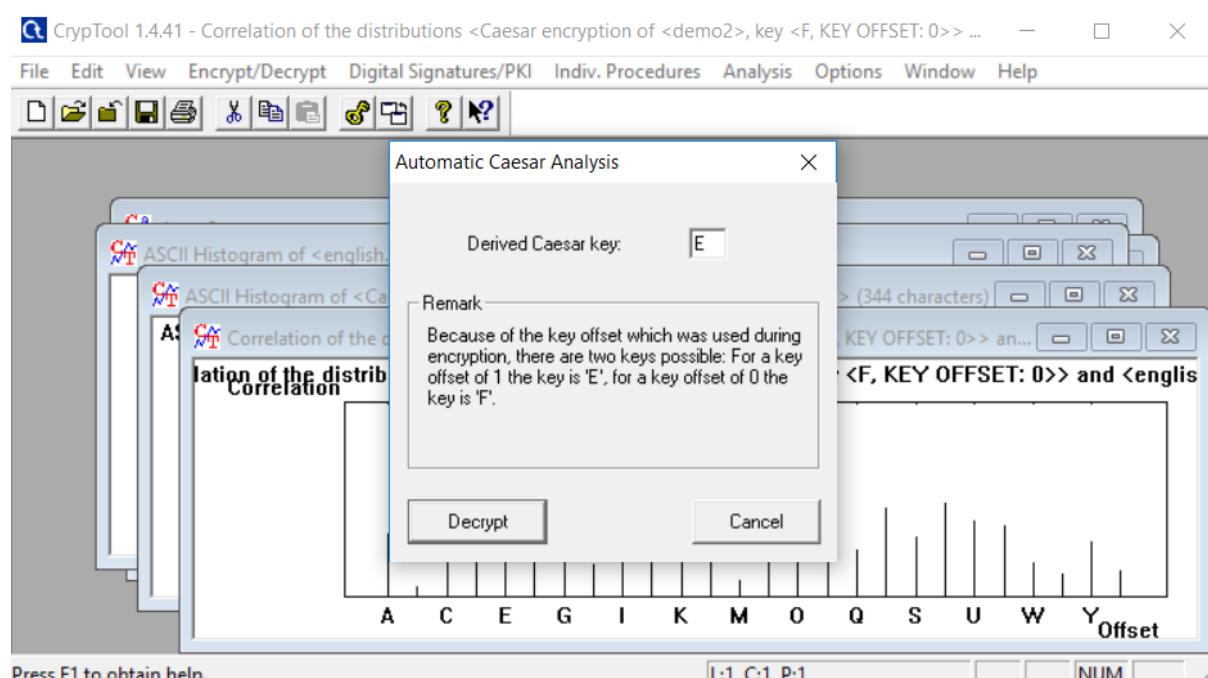
Using Caesar cipher with key value 5, the text got encrypted as shown below.



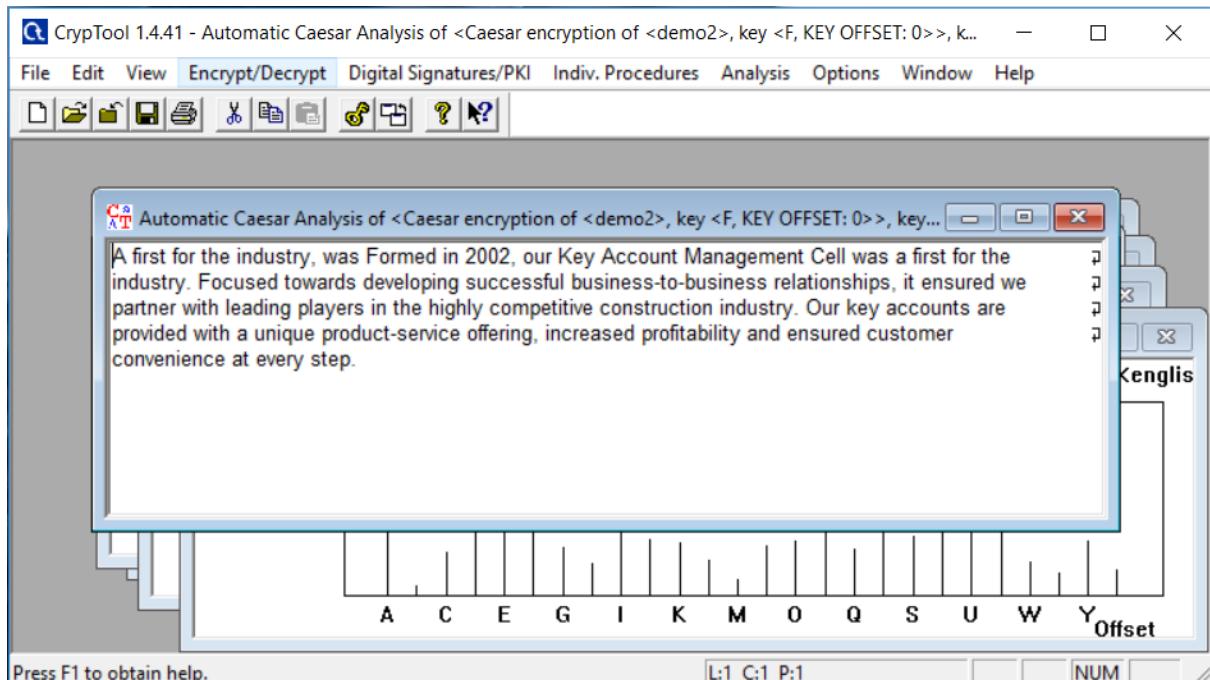
9) After obtaining cipher text, now using brute-force approach the cipher text is going to be decrypted. The brute-force attack is applied in the same way as mentioned in the above image.



10) Here the key is taken as 'E'. Hence after clicking on decrypt the entire cipher text gets decrypted.

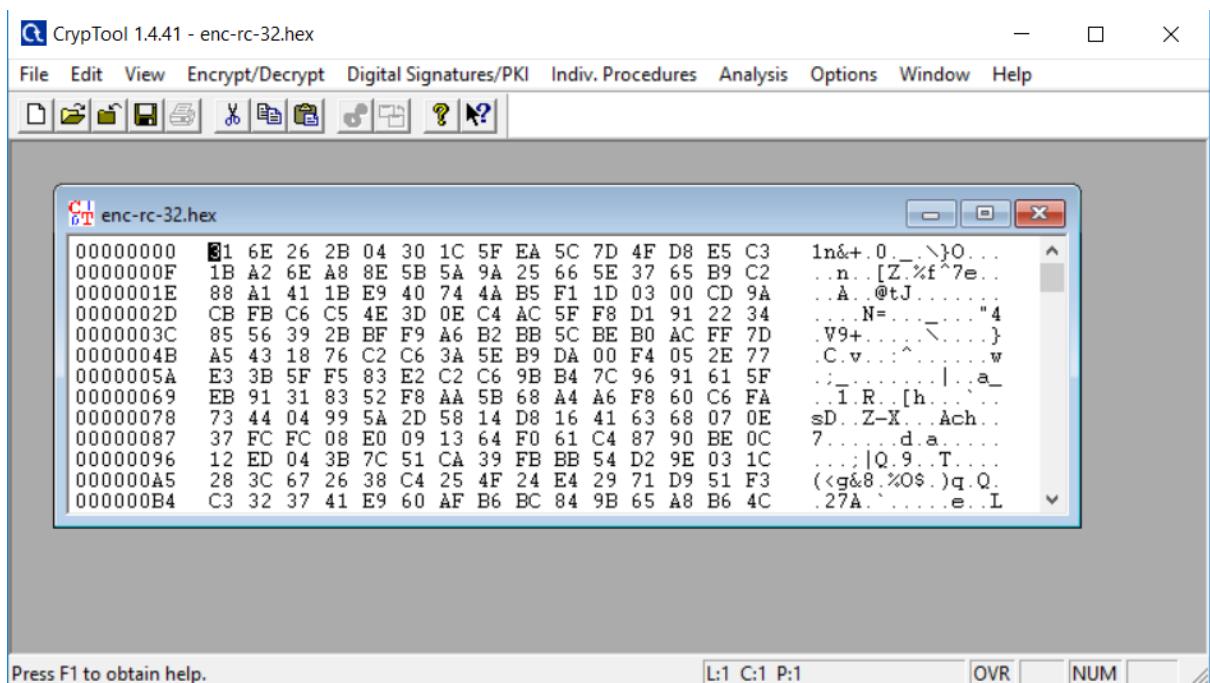


The plain text is obtained and shown below.

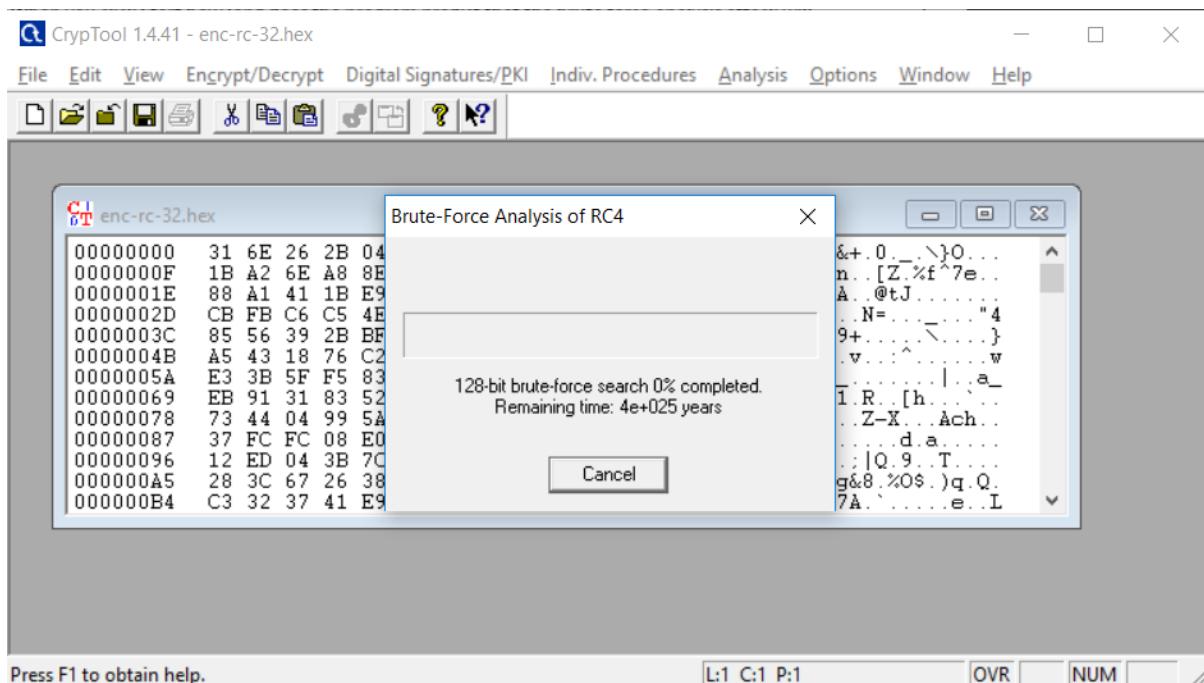


11) Now open the encrypted file “enc-rc-32.hex”. As it is in hexadecimal format, it is unreadable.

First, attempt to use a brute force attack on this encrypted file. As per the file name we know we are using an Rivest Cipher and fortunately the educational tool CrypTool supports both RC2 and RC4. Using the menu options access Analysis - Symmetric Encryption (modern) - RC2.



**Question:** When you click Start how long does the program predict that the brute force analysis attack will take?



Now to decrypt it we are going to apply the brute-force attack. Here we are using rivest cipher. As here the RC4 is chosen and key length is given as 128 bits. After clicking on start. The process starts. But it takes  $4e+025$  years to complete the process.

**Question:** Why is such a brute force attack simply not feasible?

**Answer:** It is not feasible because the attacker should try every possible key one by one for decryption. If the key size is large, then there will be a greater number of possible keys. The time to complete the attack would be very high if the key is long.

**Question:** What could you have done to diminish the amount of time required to perform the attack?

**Answer:** If we have a list of some of the popular leaked passwords by trying different passwords, that can help us to find the passwords in a faster way.

Since there is not enough time or power to brute-force a 128-bit key, click cancel. Obviously, the file still needs to be decrypted. Try using the RC4 option. So, using the instruction from point 16, select RC4 and select a 128-bit key length.

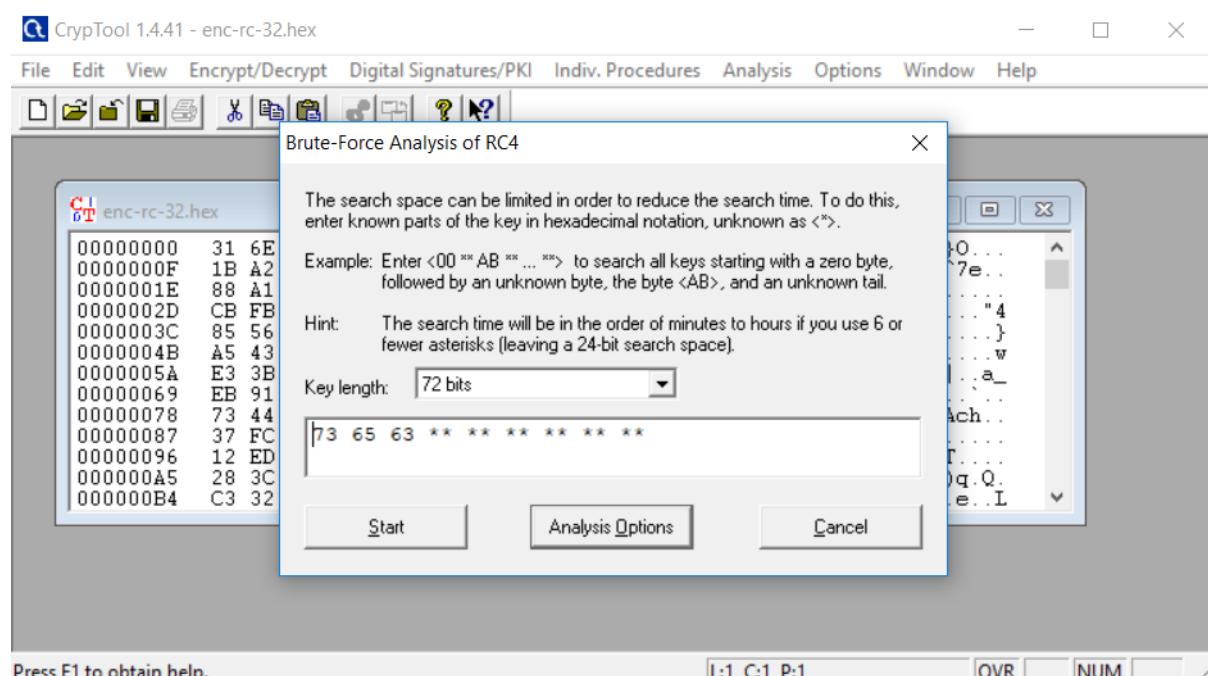
**Question:** Has the amount of time required increased or decreased?

**Answer:** Increased

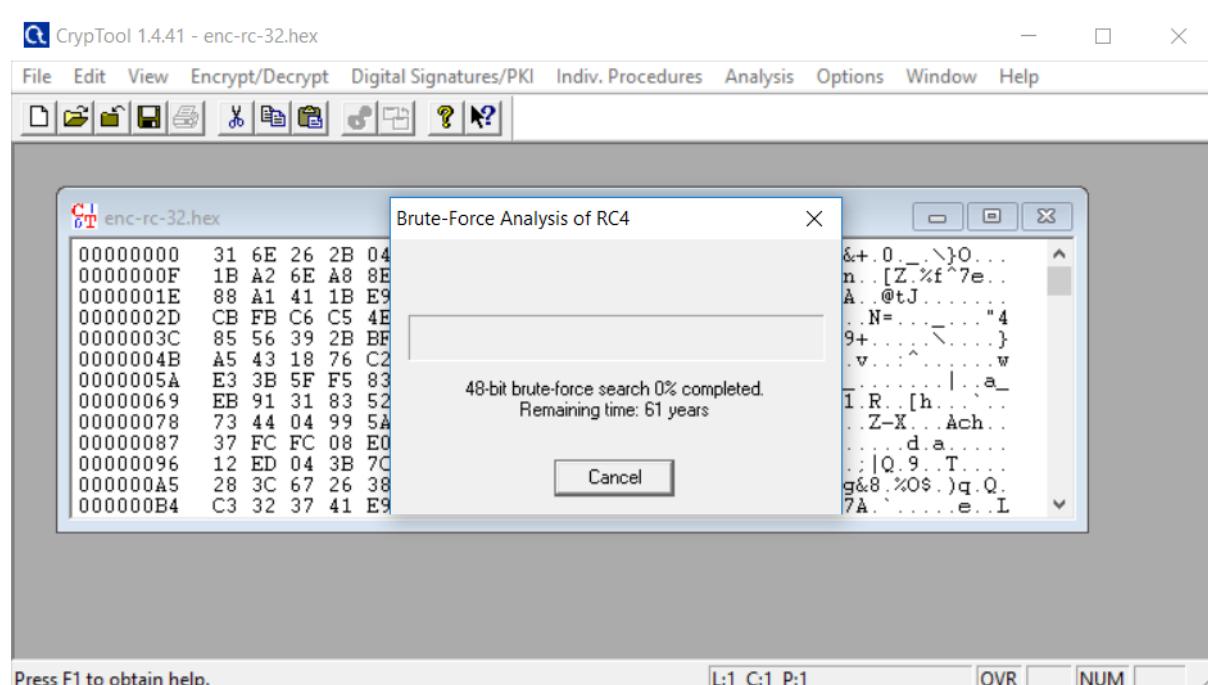
So, the file still has not been decrypted. The file is definitely encrypted using RC4 and the key length is only 72-bits.

As per the image below, the program permits a user to include specific hexadecimal values to reduce the time of the key length analysis. Hence, the first 24-bits (hex) are respectively 73 65 63.

13) Now, I have given the key length as 72 bits. Then I have given some hexadecimal values to reduce the time. Then after clicking on ‘start’ the process starts.



When we click the start button, it shows the time to complete the process.



14) Here as in the above image we can see the time taking for decryption is very long so now the process is cancelled.

Question: How much time is required to brute force attack the cipher text this time?

Answer: 61 Years

Question: Undertake research to identify the similarities and differences between the RC4 and RC2 algorithm.

Answer: The difference between RC2 and RC4 algorithms are the total rounds in RC2 are 16 and in RC4 are 256 rounds. The key length in RC2 is 8 to 1024 bits and in RC4 is 40 to 2048 bits. RC2 is a block cipher and RC4 is a stream cipher.

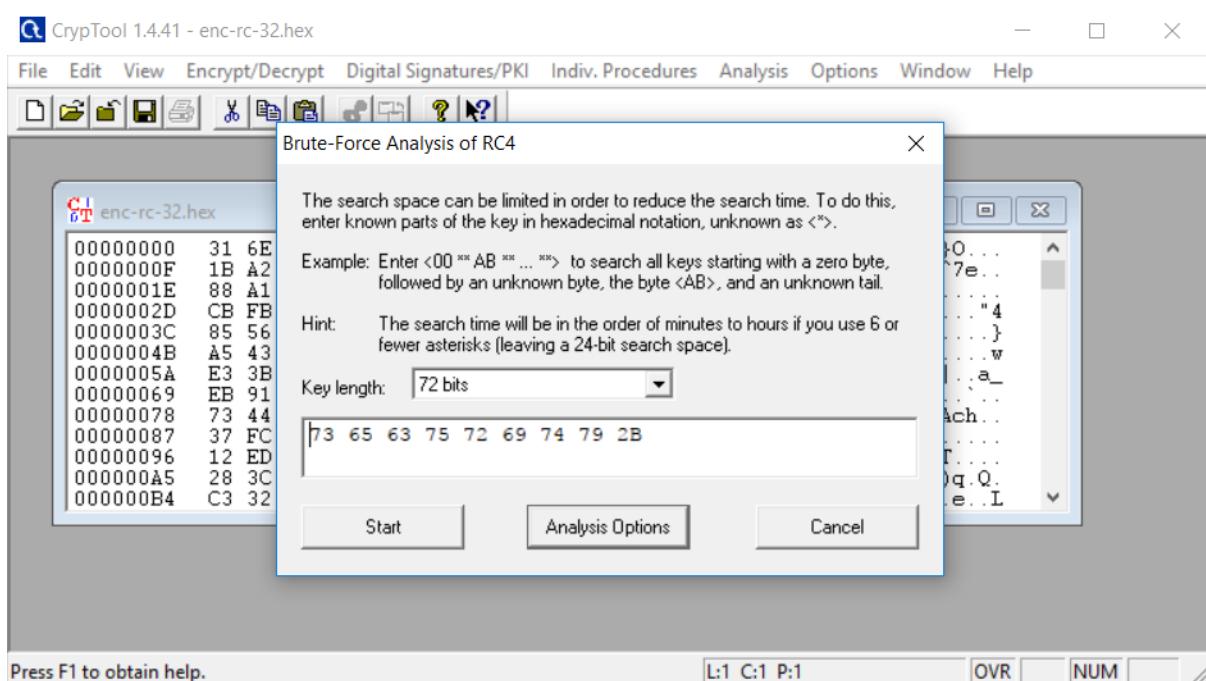
The similarities between RC2 and RC4 algorithms are: In case of security RC2 and RC4 both are vulnerable.

15) Now the hexadecimal value of ‘security+’ (ASCII key) is found using an online tool.

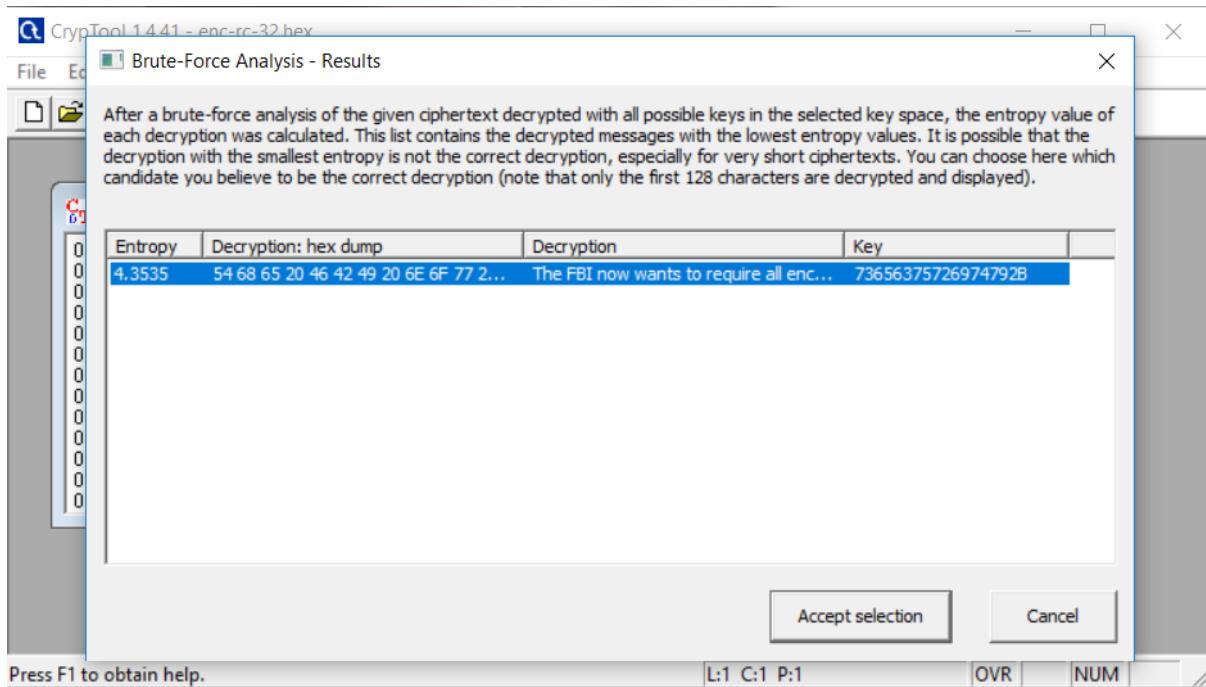
The screenshot shows a web-based ASCII-to-hex converter. At the top, there is a text input field labeled "Paste text or drop text file" containing the text "security+". Below this, under "Character encoding", a dropdown menu is set to "ASCII". Under "Output delimiter string (optional)", a dropdown menu is set to "Space". At the bottom left, there are three buttons: "Convert" (with a circular arrow icon), "Reset" (with a cross icon), and "Swap" (with a swap icon). To the right of these buttons is a text area showing the converted hex values: "73 65 63 75 72 69 74 79 2b". In the top right corner of this text area is a small green circular icon with a white letter "G".

The hexadecimal value of “security+” is:

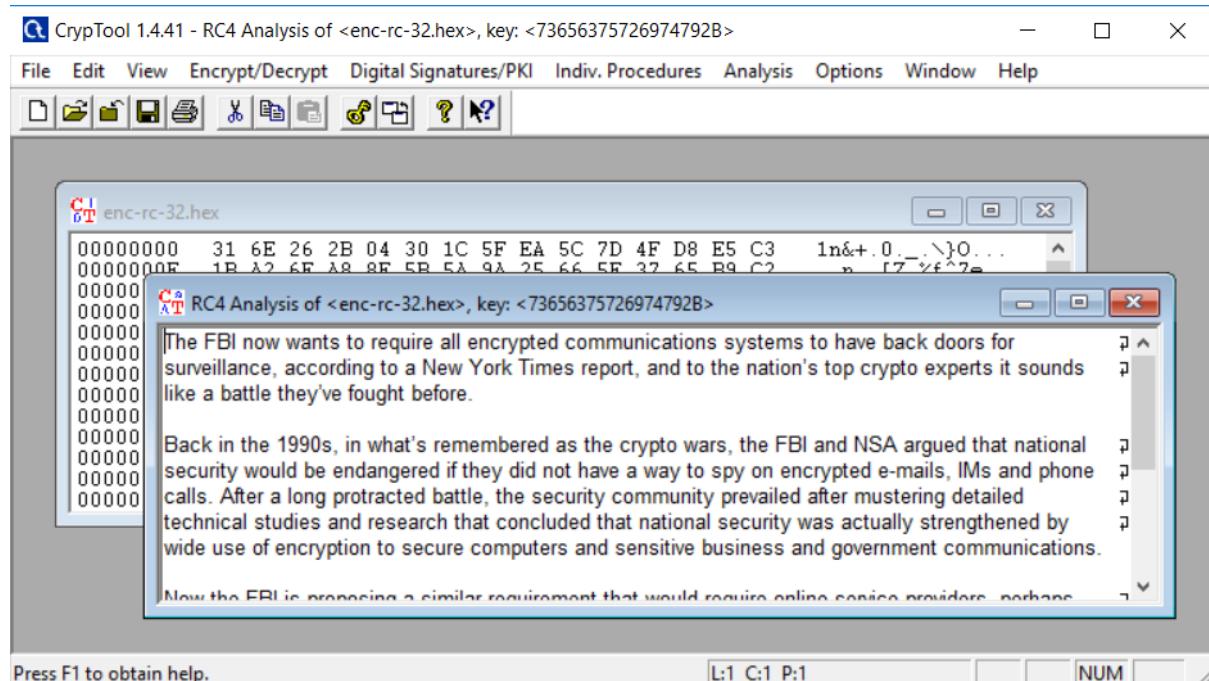
73 65 63 75 72 69 74 79 2B



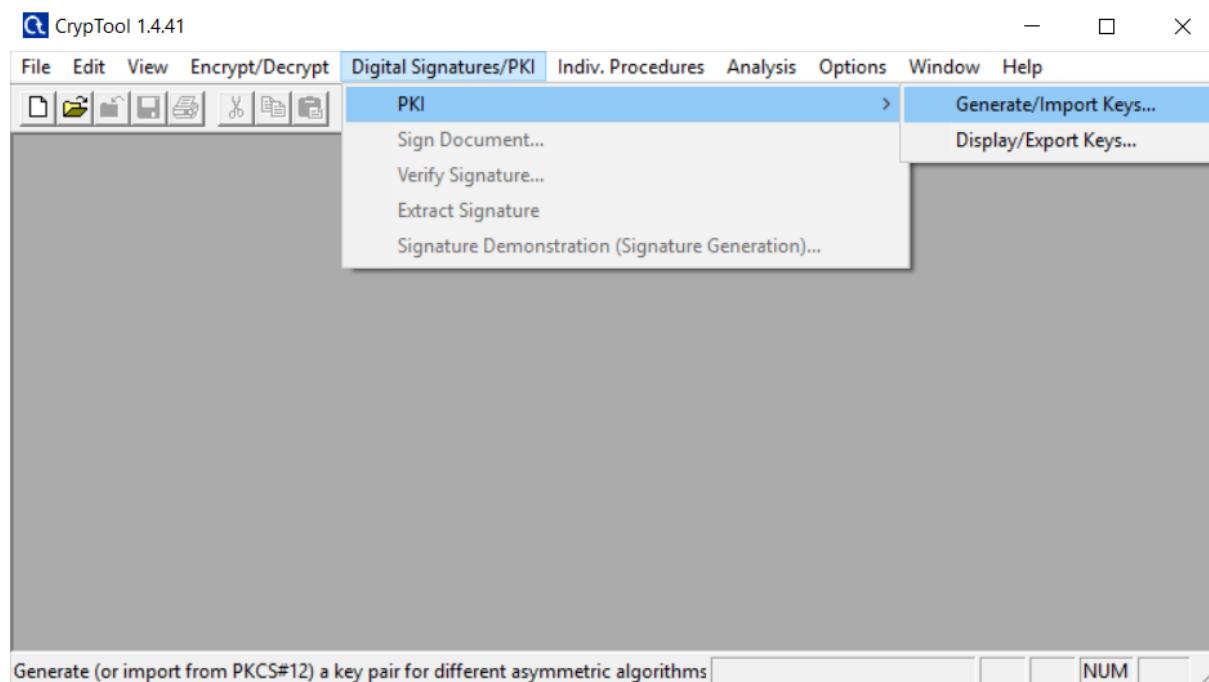
16) After getting the key in hexadecimal now we have to enter that data and click on accept selection.



17) Now the plaintext of the ‘enc-rc-32.hex’ file is obtained after the decryption.



18) So, now for asymmetric encryption a set of public and private keys need to be generated. So, we need to go to digital signatures/PKI and choose Generate/Import Keys.



Question: Why is this user data required when utilising asymmetric encryption?

Answer: The user data that has been given can only access the keys that are generated.

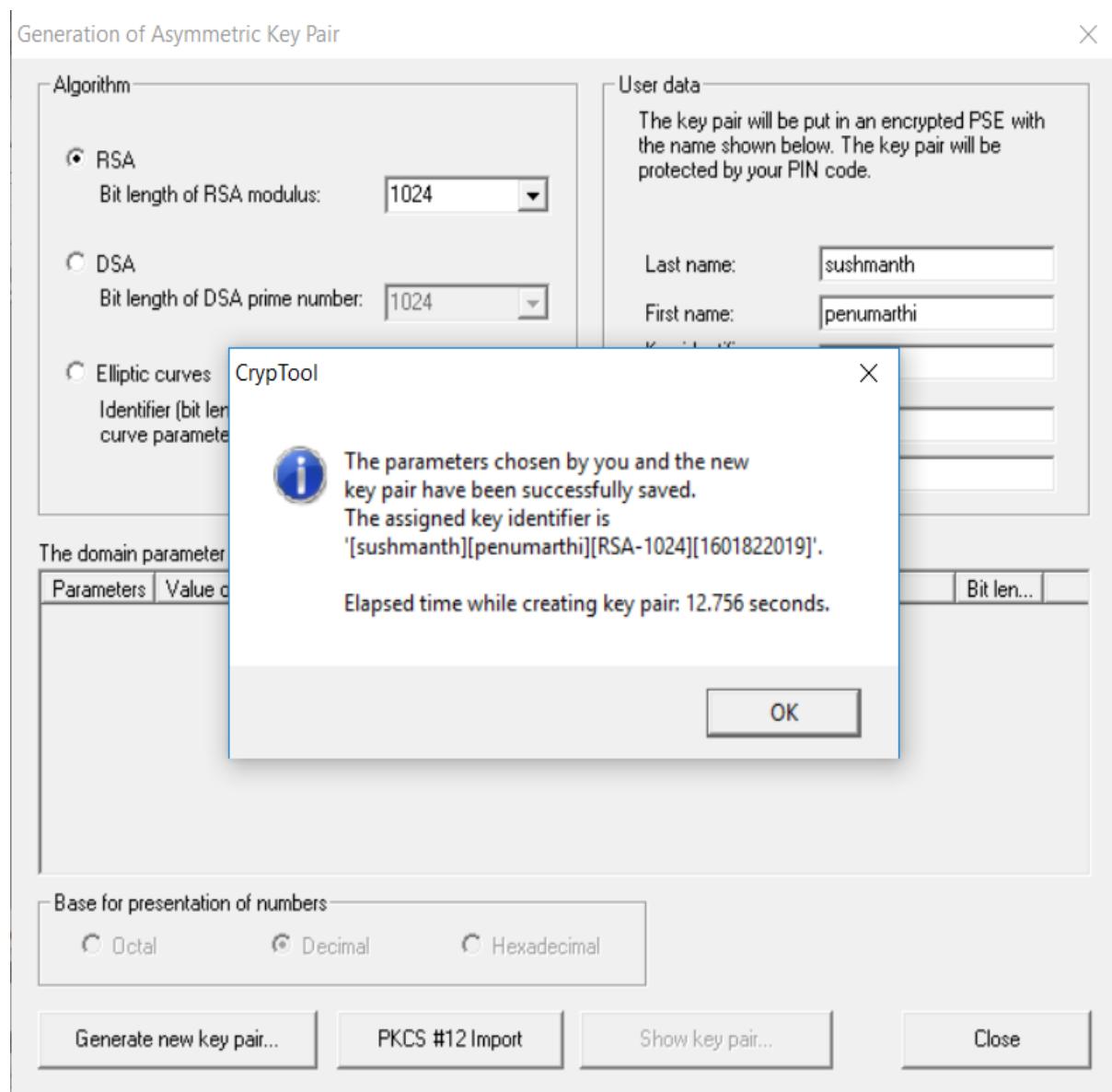
Question: Why does the program require that you move your mouse around when generating your keys?

Answer: If we move the mouse the system generates some random data.

Question: How does this process differ from using symmetric encryption?

Answer: Symmetric encryption uses one key for both encryption and decryption, and the asymmetric encryption uses a public key for encryption and a private key for decryption.

19) Here, we have to fill some of the required fields like last name, first name and pin. I have chosen the RSA algorithm and gave the user data as mentioned in the above image.



Question: Why does the encryption component ask us to choose our recipient (think back to symmetric versus asymmetric encryption)

Answer: The file that has been sent can be decrypted by only the selected recipient.

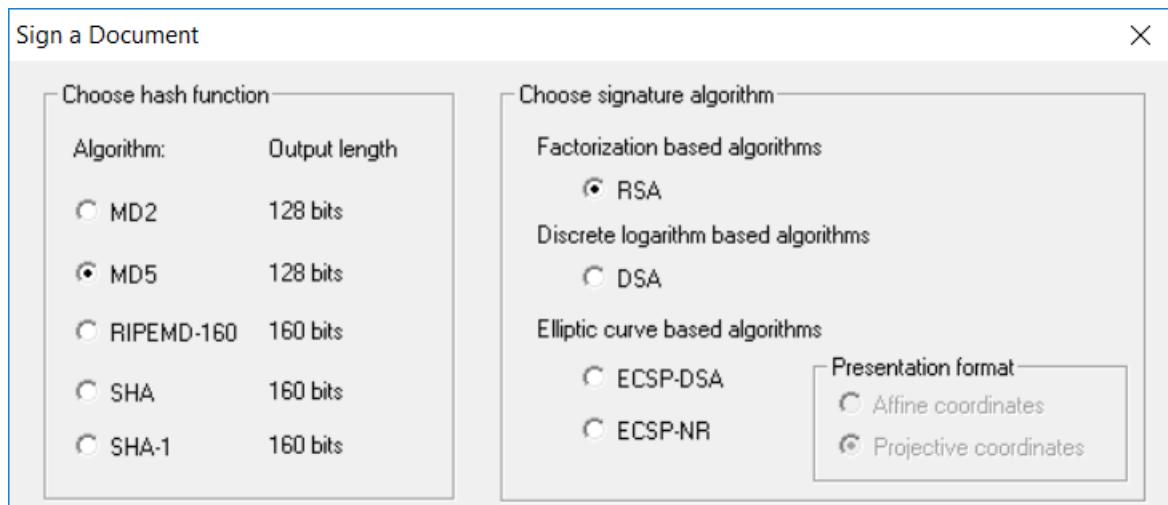
Question: Which key (private or public) is being used to encrypt the document and why?

Answer: In asymmetric encryption Public key is used because it is infeasible to determine the decryption key

Question: Imagine you accidentally forgot the pin or have deleted your key pairs. As a result, you decide to generate a new private and public key pair. If you use the same user data as you did the first time, will the cipher-text be the same or different (you may like to try and experiment with this)?

Answer: I think the ciphertext will be different.

20) Using the default startingexample-en.txt file as our base text, you will now attempt to encrypt the digital document. Click **Encrypt/Decrypt - Asymmetric - RSA Encryption**.



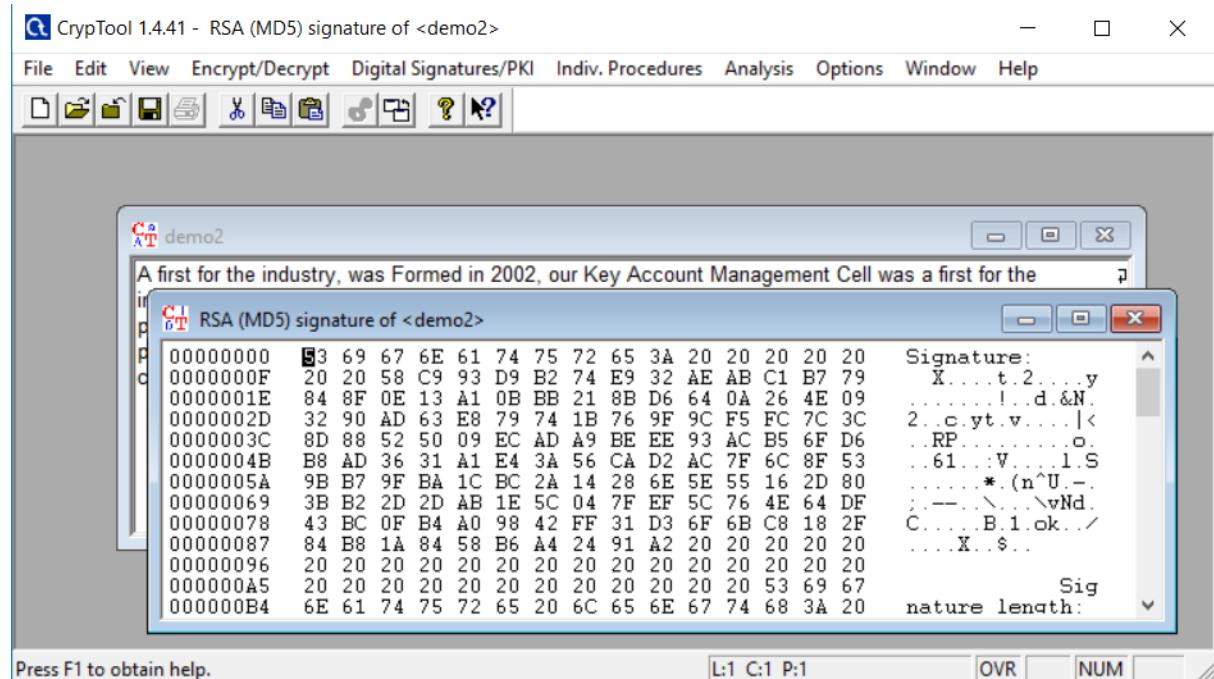
Here, to sign the document initially I have chosen the MD5 hash function.

Question: Why aim of security does signing a document ensure?

Answer: Digital signatures ensure that the documents have not changed without authorization.

Last name	First name	Key type	Key identifier	Created	Internal ID no.
HybridEncrypti...	Bob	EC-prime239v1	PIN=1234	09.05.2007 14:51:14	1178702474
SideChannelAt...	Bob	RSA-512	PIN=1234	06.07.2006 15:21:34	1152179494
sushmanth	penumarthi	RSA-1024		04.10.2020 20:03:39	1601822019

Once you have encrypted the digital document successfully, you should notice a new file has been created with lots of hexadecimal outputs.



Question: What type of information is now contained in this document?

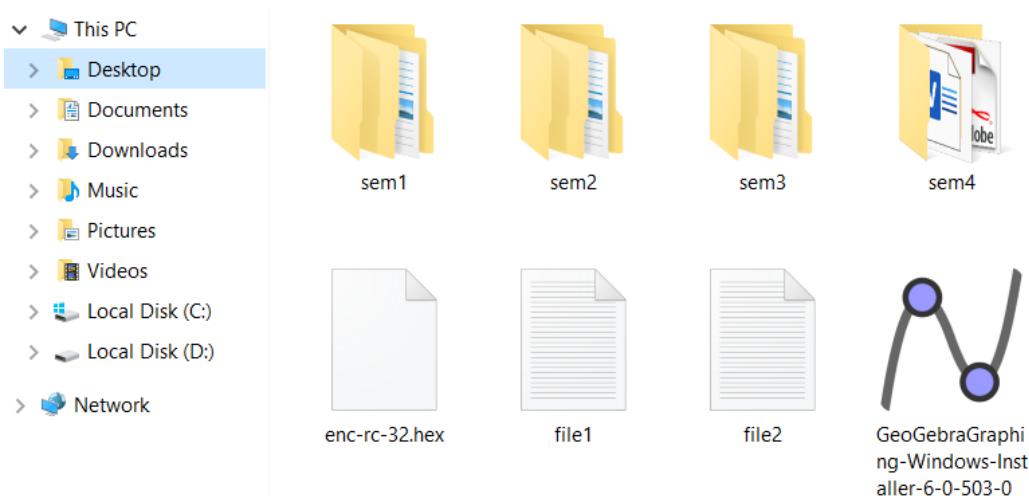
Answer: Signature, hash function, Signature Length, plain text.

Question: How is digitally signing a document beneficial?

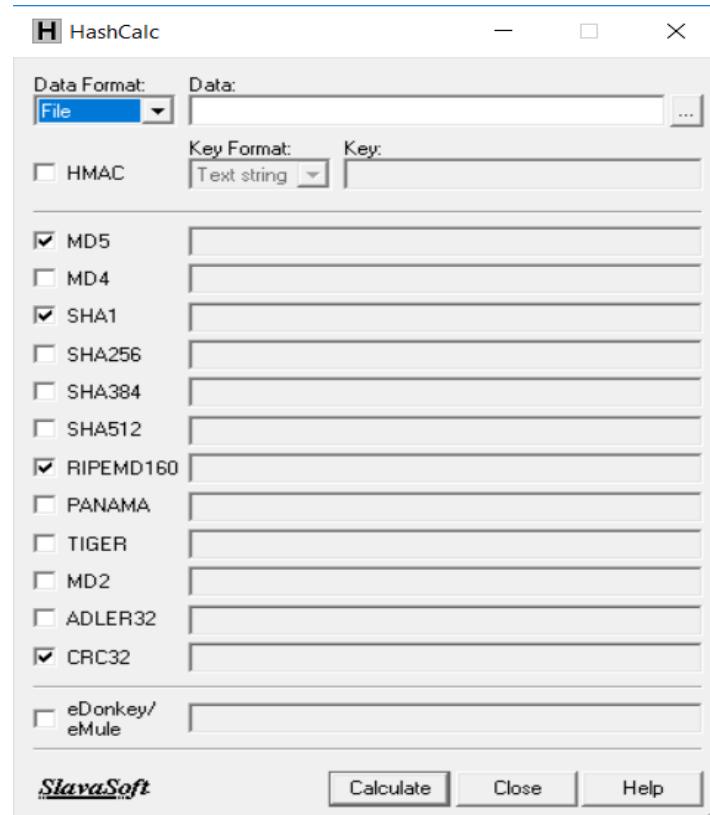
Answer: The benefits of digital signatures have more offices and companies choosing them to make their workplace more efficient and secure. It reduces the risk of alteration of documents. The document cannot be altered without authorization.

## Hash Calc

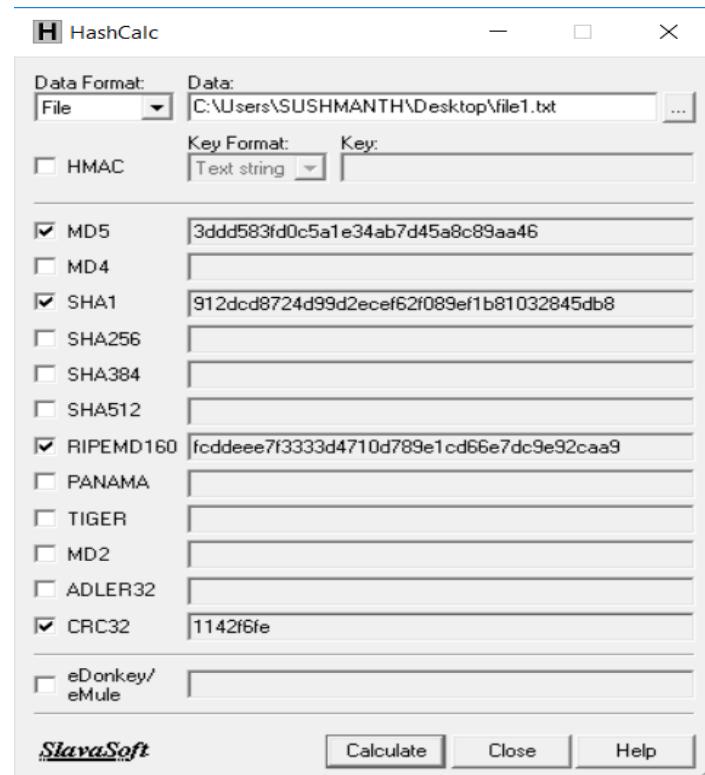
24) Initially, I have created two test files “file1.txt” and “file2” on my desktop.



25) Now I have opened HashCalc tool and chosen data format as ‘File’ and many hash functions in which I consider MD5 hash function.



26) Now calculate and note down the MD5 value for file1.txt.



Next calculate and note down the MD5 value for file2.txt



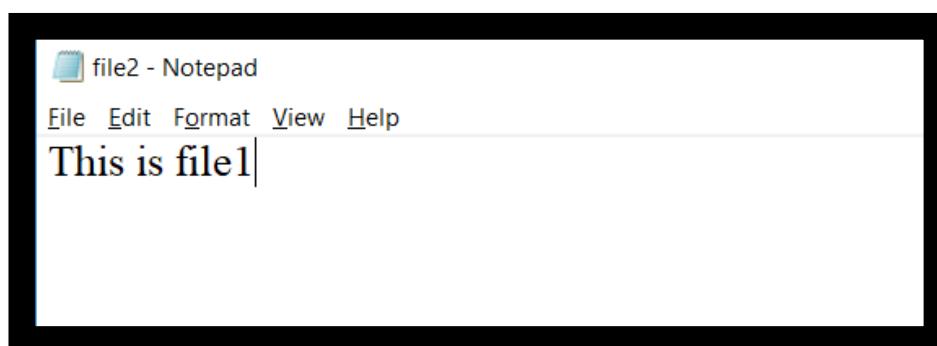
Question: Has the MD5 value changed?

Answer: The MD5 value for file-1 and file-2 are different as the content inside each of the files is different.

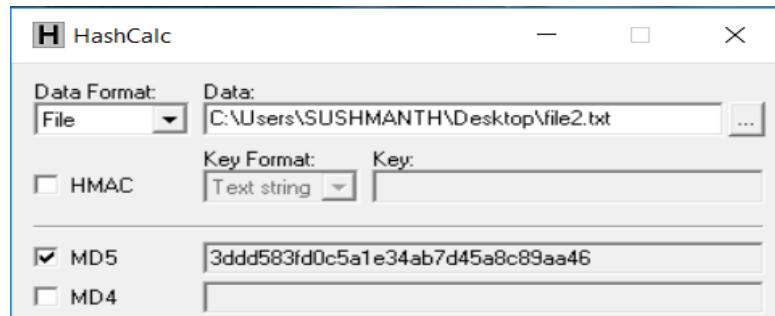
Question: Why did this particular process occur?

Answer: The content inside each of the files is different. So, the hash values are also different.

Now I have deleted the file2.txt and renamed file1 as file2.txt.

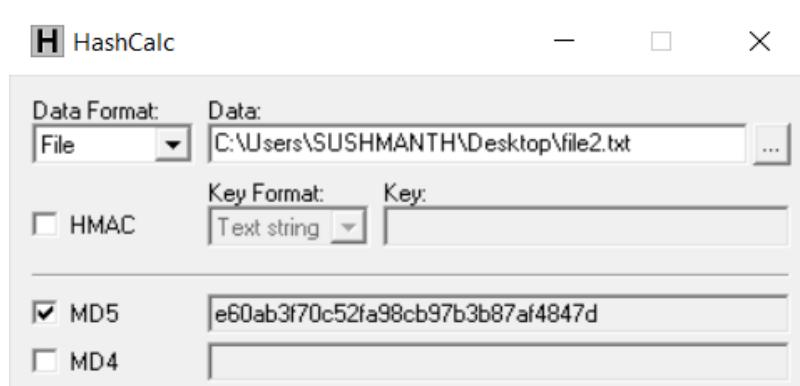


Now calculate the hash value for that file.



It has got same as the file1.txt

Now change the content in file2.txt to "This is file2" and calculate the hash value.



It has got same as the file2.txt (before rename) which was deleted.

Question: Has the MD5 value changed?

Answer: Yes, it is changed because the content inside the file is changed.

Question: Does it correspond to a particular MD5 value obtained previously?

Answer: Yes, it is the hash value of original file-2 which was deleted.

Question: What changes need to be made to the file for it to alter the resultant MD5 value?

Answer: We need to change the content inside the file in order to change the MD5 hash value of a file.