

**Secure Communications Module**

**Non-Coding Labs Based Report**

**James Finglas – B00094138**

Department of Informatics

School of Informatics and Engineering

Institute of Technology Blanchardstown

Dublin 15.

Word count: 7924

**BN311 Bachelor of Science in Computing**

**Digital Forensics & Cyber Security**

**29/3/2019**

Contents

[1.0 Introduction 3](#_Toc4763844)

[2.0 Decoding Classic Ciphers & Extra Decoding challenge 3](#_Toc4763845)

[2.1 Decoding Classic Ciphers 3](#_Toc4763846)

[2.1.1 Exercise 1 3](#_Toc4763847)

[2.1.2 Exercise 2 8](#_Toc4763848)

[2.1.3 Exercise 3 11](#_Toc4763849)

[2.1.4 Exercise 4 14](#_Toc4763850)

[2.1.5 Extra Decoding challenge 15](#_Toc4763851)

[2.2: Reflecting on Decoding Classic Ciphers & Extra Decoding challenge 15](#_Toc4763852)

[3: Numbers Station 16](#_Toc4763853)

[4.0 Salted Hashes 18](#_Toc4763854)

[4.1 Cracking the Hashes 19](#_Toc4763855)

[Reflecting on Salted Hashes: 23](#_Toc4763856)

[5.0 Digital Certificates & Signatures 24](#_Toc4763857)

[5.1 Viewing Certs on Your PC 24](#_Toc4763858)

[5.2 Exchanging Signed Documents – JAR Tool File 28](#_Toc4763859)

[6.0 Wireshark 33](#_Toc4763860)

[6.1 Setting Up FTP 33](#_Toc4763861)

[6.2 Pcap Challenges 36](#_Toc4763862)

[6.2.1 Pcap Challenge 1 36](#_Toc4763863)

[6.2.2 Pcap Challenge 2 36](#_Toc4763864)

[7.0 Encrypted Wireless Capture 38](#_Toc4763865)

[8.0 Securing Internet Traffic 41](#_Toc4763866)

[9.0 TOR & the Dark Web 46](#_Toc4763867)

[10.0 Tor Website 53](#_Toc4763868)

[10.1 Configuring Tor 54](#_Toc4763869)

[Conclusions 57](#_Toc4763870)

[Glossary 57](#_Toc4763871)

[References: 58](#_Toc4763872)

# 1.0 Introduction

Herein is documented the answers to the questions or challenges posed by the labs covered in this (2018-19) terms Secure Communications Module. We are task with keeping a small lab journal documenting our methods, results and understanding of the tasks being carried out.

# 2.0 Decoding Classic Ciphers & Extra Decoding challenge

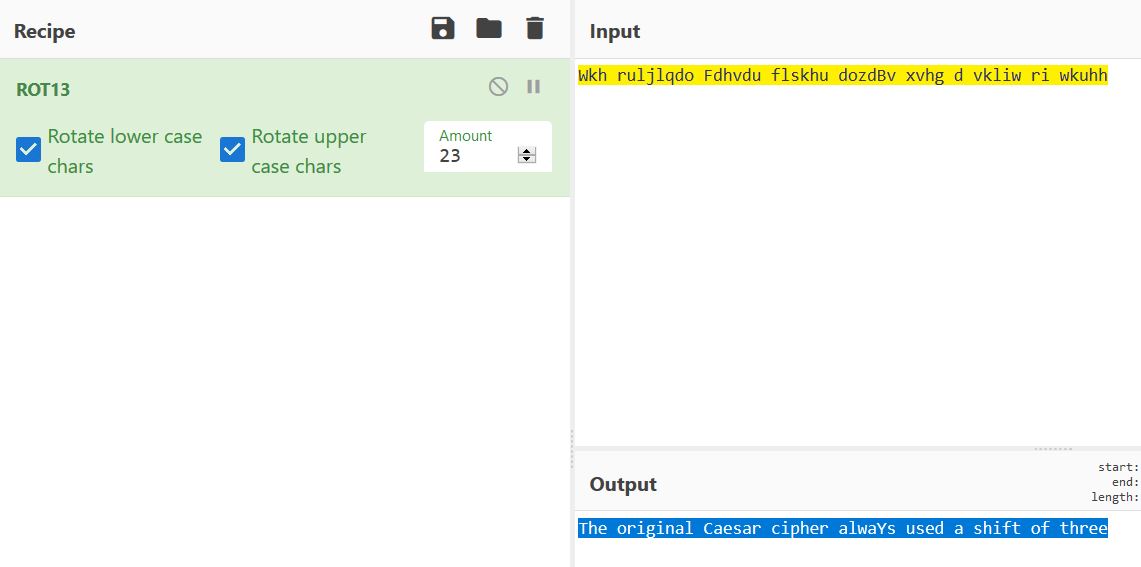
## 2.1 Decoding Classic Ciphers

This first lab was a very simple lab designed to make us examine and decode some of the most basic common and oldest ciphers known. For the most part this was a simple case of using online decoders and documenting the process as follows.

### 2.1.1 Exercise 1

#### A:

First up we are tasked with decoding the following sequence: Wkh ruljlqdo Fdhvdu flskhu dozdBv xvhg d vkliw ri wkuhh. We are told that this is a classic Caesar Cipher.

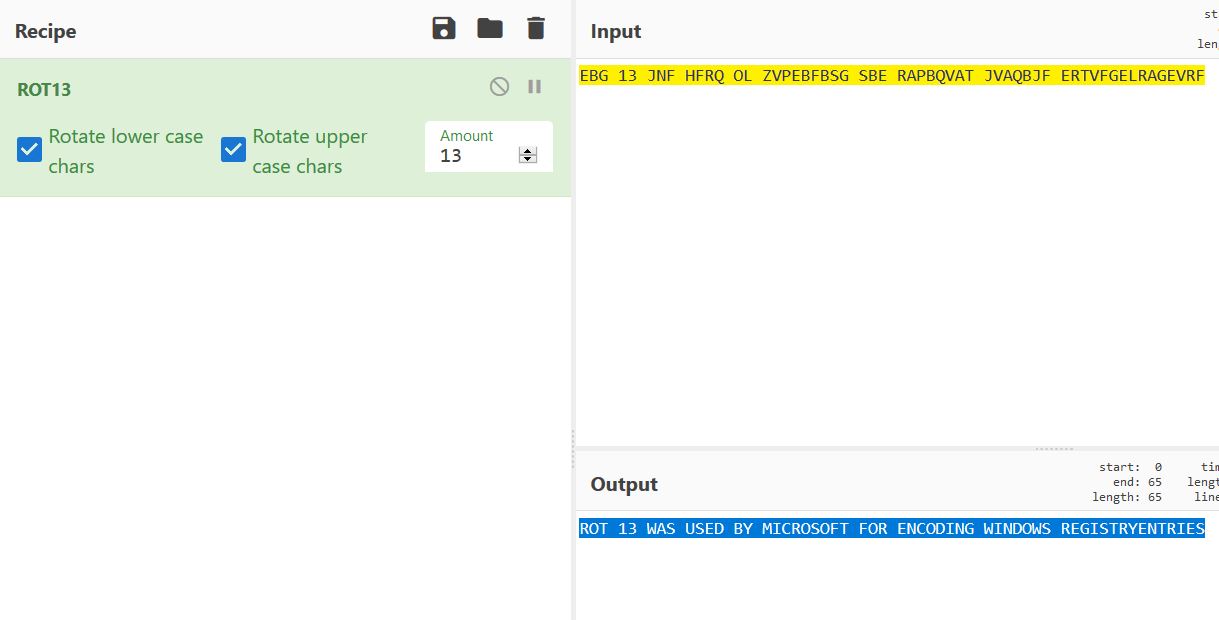


As we can see above this is simply a matter of entering the text into CyberChef and selecting ROT13 and adjusting the rotation until we get the desired output.

Decoded text: the original Caesar Cipher always used a shift of three.

#### B:

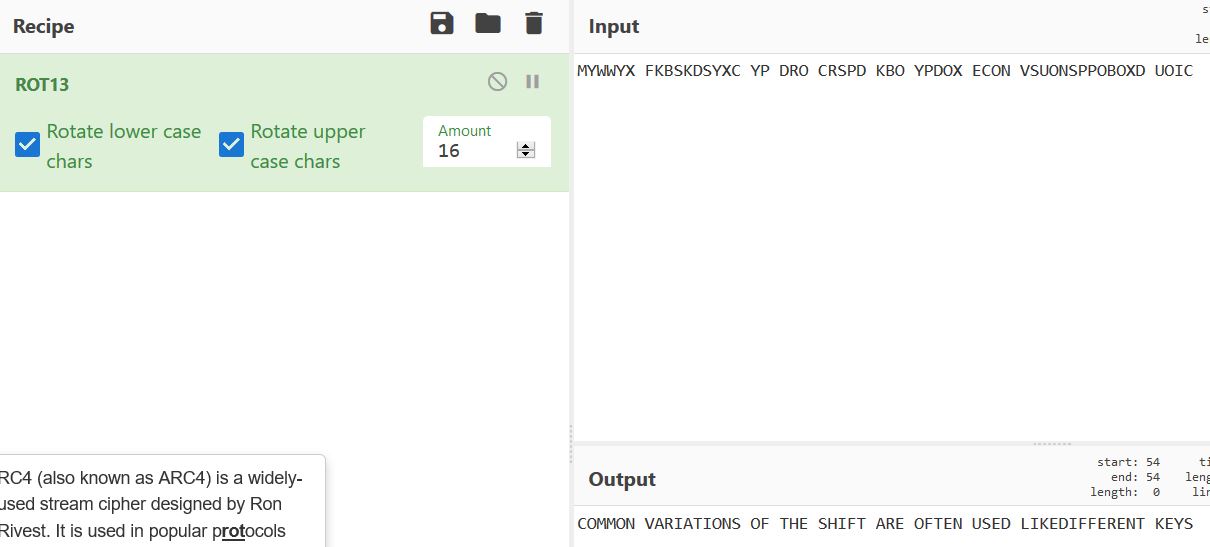
The next challenge is to decode the following string of ROT13 encoded data: EBG 13 JNF HFRQ OL ZVPEBFBSG SBE RAPBQVAT JVAQBJF ERTVFGELRAGEVRF. We did this by applying the same method as the previous challenge.



Decoded text: ROT 13 WAS USED BY MICROSOFT FOR ENCODING WINDOWS REGISTRY ENTRIES

#### C:

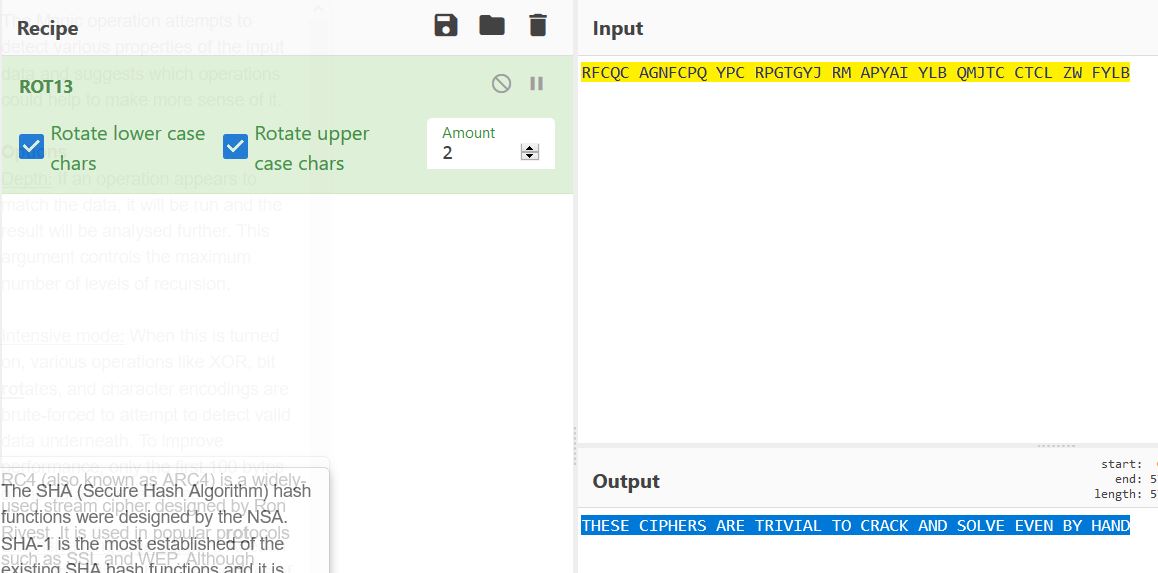
Next, we are tasked with cracking 3 shift ciphertexts strings, the first being: MYWWYX FKBSKDSYXC YP DRO CRSPD KBO YPDOX ECON VSUONSPPOBOXD UOIC.



Once again this is discovered to be ROT13. Decoded text: COMMON VARIATIONS OF THE SHIFT ARE OFTEN USED LIKEDIFFERENT KEYS.

#### D:

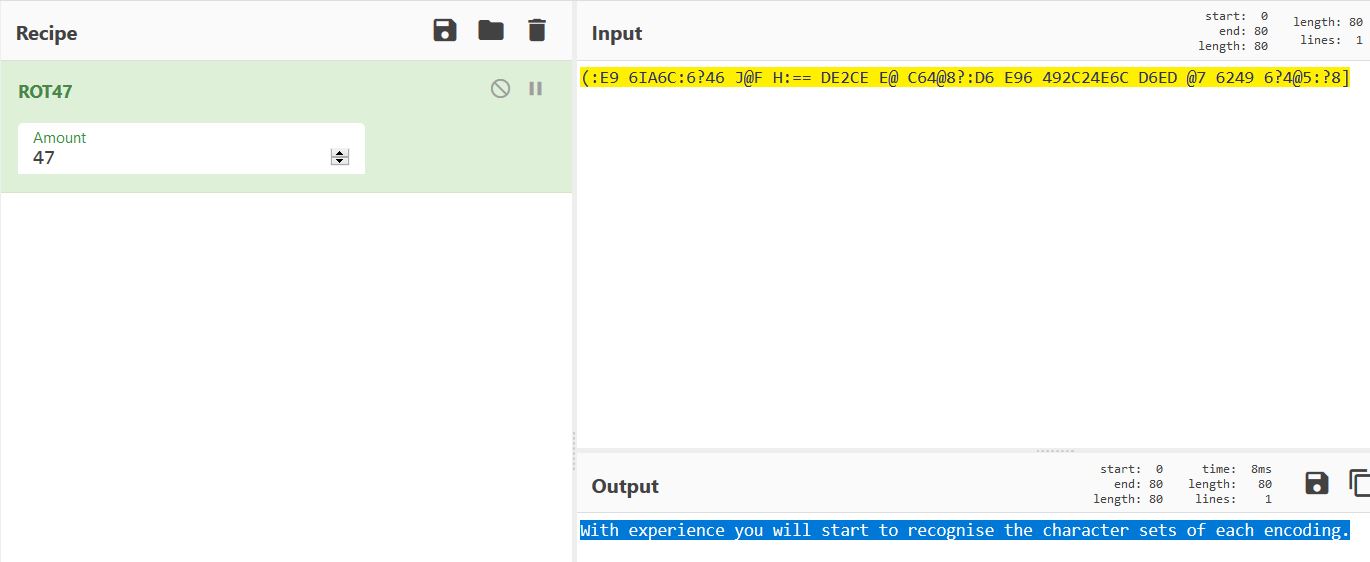
Continuing with shift ciphers the next string of cipher text provide is: RFCQC AGNFCPQ YPC RPGTGYJ RM APYAI YLB QMJTC CTCL ZW FYLB.



Yet another rot 13 cipher with the decoded text being: THESE CIPHERS ARE TRIVIAL TO CRACK AND SOLVE EVEN BY HAND.

#### E:

And finally for shift ciphers the last string of cipher text is: (:E9 6IA6C:6?46 J@F H:== DE2CE E@ C64@8?:D6 E96 492C24E6C D6ED @7 6249 6?4@5:?8]. Now, this cipher immediately appears different. AS we can see it contains numbers and special characters.  
  
Rot13 however is not an alpha numeric cipher and it can only contain letters. Rot5, another f the most common old ciphers contain only numbers, therefore the most likely cipher we have covered that is alpha numeric is Rot47.



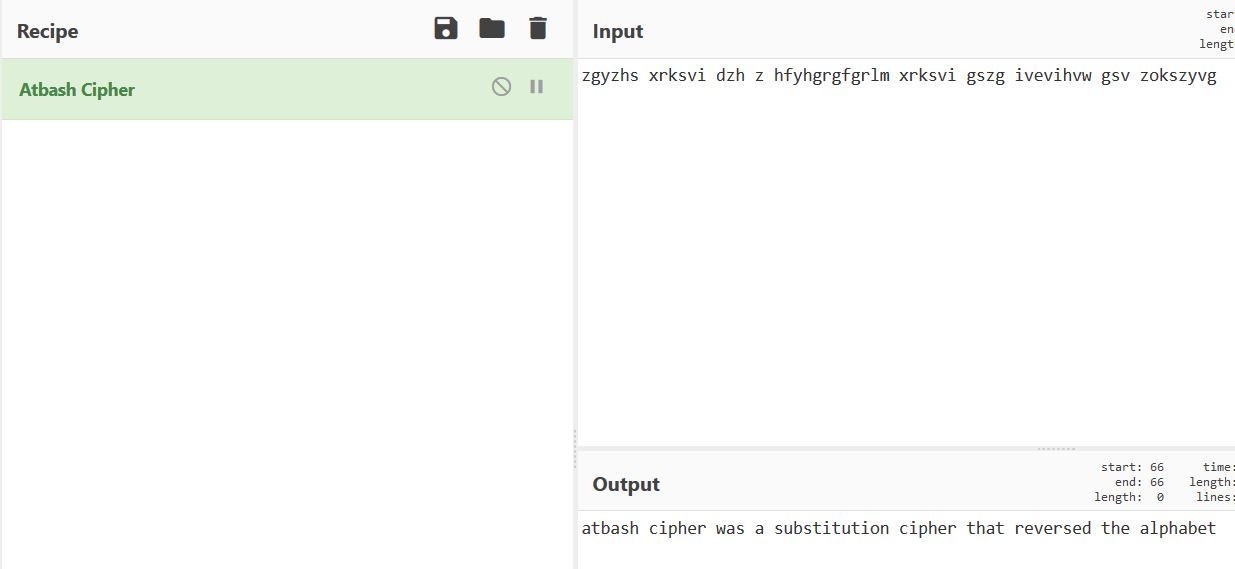
Sure enough, it is indeed Rot47 with the decided text being: With experience you will start to recognise the character sets of each encoding.

### 2.1.2 Exercise 2

Now we move on from basic shift ciphers into basic substitutions ciphers.

#### A:

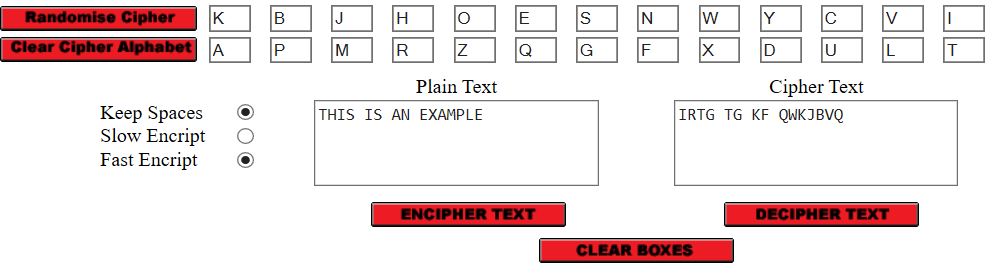
First up we are provided with some Atbash cipher text and asked to decode this: zgyzhs xrksvi dzh z hfyhgrgfgrlm xrksvi gszg ivevihvw gsv zokszyvg.



Decoded text: atbash cipher was a substitution cipher that reversed the alphabet

#### B:

Things get a little more difficult now, as we move into the Karma Sutra cipher. We are provided with this cipher text: Irtg tg kf qwkjbvq. As well as this Upper half: KBJHOESNWYCVI and this Lower half: APMRZQGFXDULT translation table. Karma sutra works by substituting any letters found in the upper half with the corresponding letter found in the lower half. Both halves are 13 digits characters long making up the 26 characters of the alphabet and cannot be repeated. For example, as we can see, the first letter in our provided upper half is ‘K’ and the first provided in the lower half is ‘A’. So, all K’s become A’s.



Decoded text: THIS IS AN EXAMPLE

#### C:

This time we are asked to try to break a Cipher, but we are given no useful hints. The only way to break this cipher therefore, is by frequency analysis. Frequency analysis is where one attempts to anaylize how often a letter is used and the likely hood that any given letter can be matched up to it. Two factors reat reduce the difficulty of this. Firstly, the longer the entire string is, and secondly the more often a letter or character is repeated.  
  
While this can be a lengthy and often time-consuming process to do manually, luckily great online tools exist to speed this process up. I chose ’Quipquip’ which is an online website capable to attempting brute fore frequency analysis attacks on encrypt text.



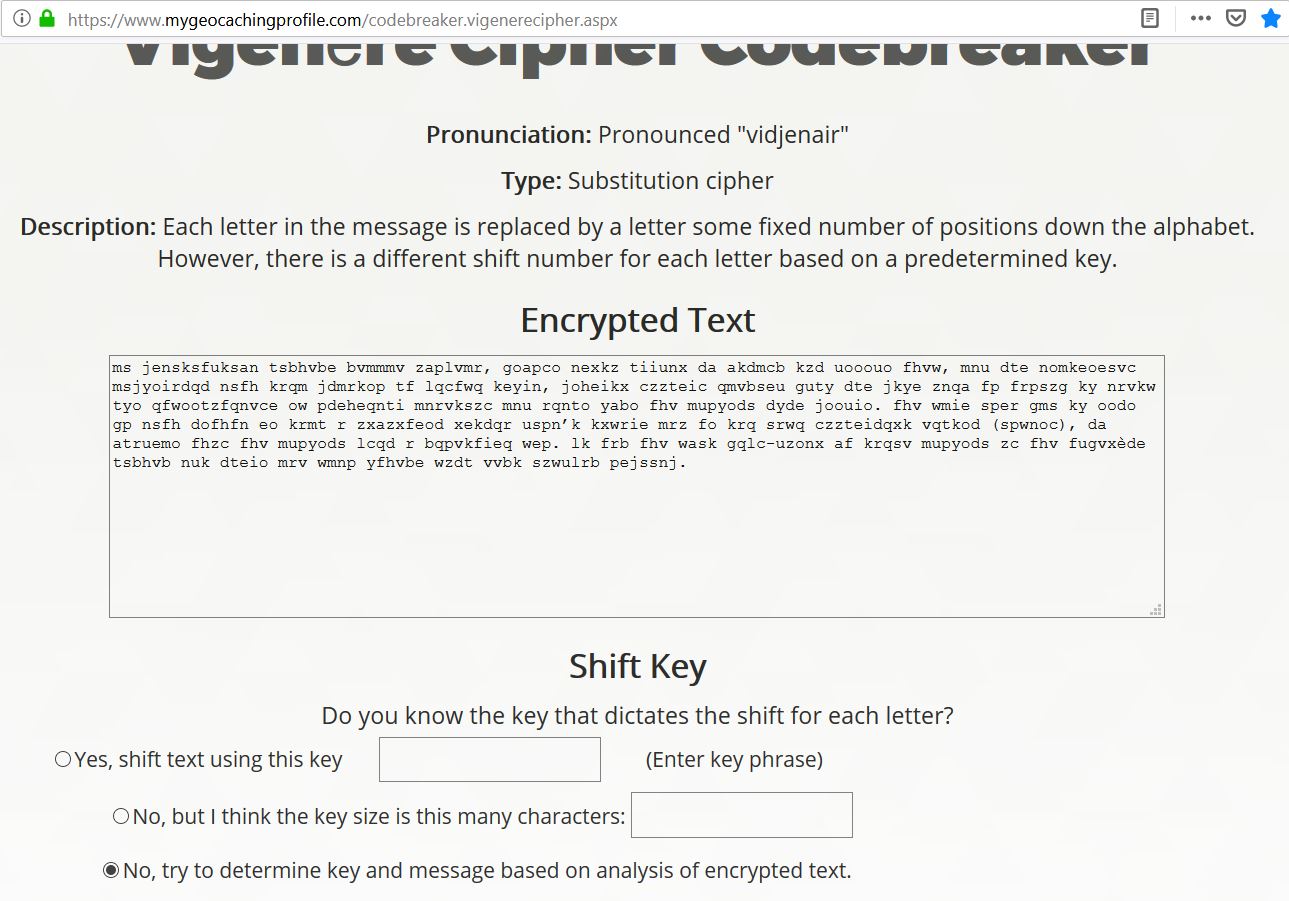
Decode text: frequency analysis is the key to breaking substitution ciphers the longer the message the easier it becomes.

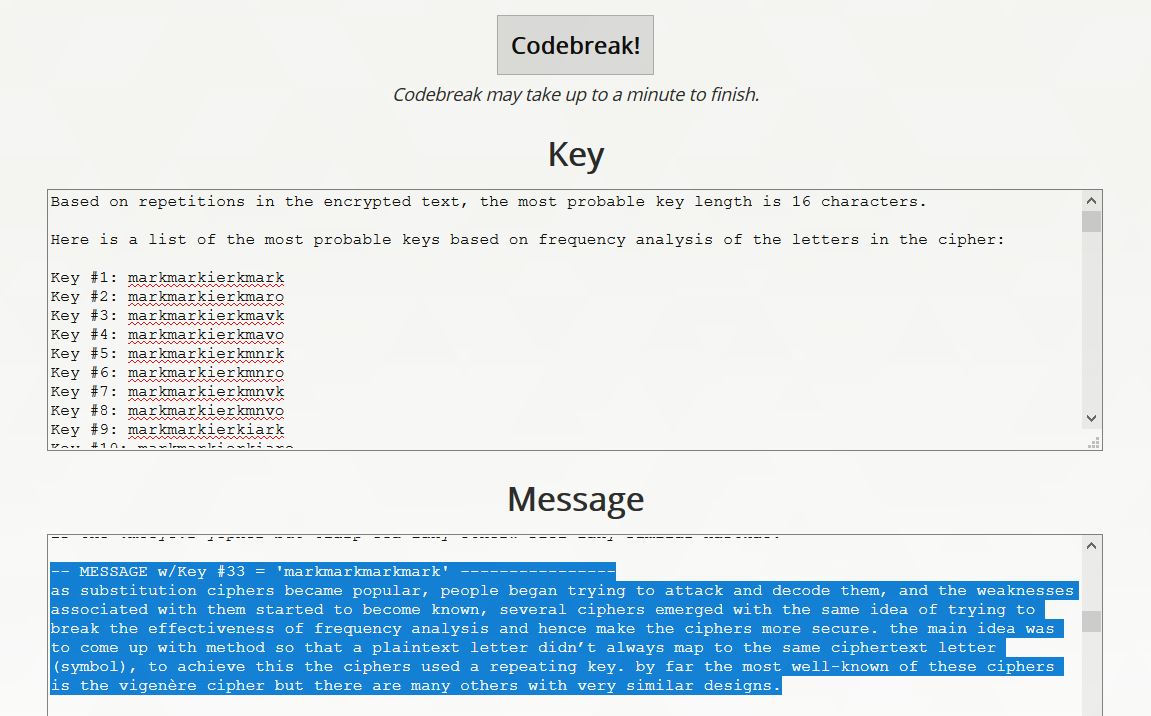
### 2.1.3 Exercise 3

This time around we are asked to break a Vigenère Cipher. This is also done by frequency analysis. However, I found my tools a bit lacking here so I began to search for more powerful brute force tool. And I found one which is better for statistical analysis.

So, we are tasked with deciphering Vigenère cipher: mS jeNSkSfUkSaN TSbHVbe BVMMMV ZaPlVMR, gOaPcO NEXKZ TiiUNX da AkdMCb KZD UOOOUO fHVW, MNU dTE nOMKeOeSVc MSjYOIRdQD nSfH kRQM jdMRkOP Tf LQCfWQ KeYiN, jOhEiKX CZZTEic QMVbSEU gUTY dTE jKYE ZNQA fP fRpSZG kY NRVKW TYO QFWOOTZfQNVce OW PdEheQNTi MNRVkSZc MNU RQNTO YAbO fHV MUPYOdS dYdE jOOUiO. FHV WMIe SPER gMS kY OOdO gP nSfH dOfHfN eO kRMT R ZXAZXfEod XEkdQR USPN’k KXWRie MRZ fO kRQ SRWQ CZZTEidQXk VQTkOd (SpWNOc), da ATRUEmO fHZc fHV MUPYOdS lcQD R bQPVKfIeQ WEp. lk FRb fHV WaSk gQLc-UZOnX aF kRQSV MUPYOdS Zc fHV FUGVXèdE TSbHVb NUk dTEiO MRV WMNp YfHVbe WZdT VVbk SZWULRb PEjSSNj.

I elected to use this site: <https://www.mygeocachingprofile.com/codebreaker.vigenerecipher.aspx> , which I found to be an extremely useful site for statistical analysis based code breaking.





Decoded text: as substitution ciphers became popular, people began trying to attack and decode them, and the weaknesses associated with them started to become known, several ciphers emerged with the same idea of trying to break the effectiveness of frequency analysis and hence make the ciphers more secure. the main idea was to come up with method so that a plaintext letter didn’t always map to the same ciphertext letter (symbol), to achieve this the ciphers used a repeating key. by far the most well-known of these ciphers is the Vigenère cipher but there are many others with very similar designs.

### 2.1.4 Exercise 4

This time around we are asked to solve a transposition cipher. The cipher text provided was: NRAATIOSSPCOTNIRHIEPCSAOLDLAELMLCUORPNESTTMAUSNIIOHEACTEQNUIA CTHOHENTGEREDOHFRTOTEETLNSEIRTEAXTAPBLYINCGIGNTAIDRI. For this attack I elected to use dcode.fr. While this site is often not quite as successful as others, it does appear to be more effective where transposition attacks are concerned.



As we can see above, the attack has been successful, however we still need to extract and break up the decode output into readable text. So, we take the following: ATRANSPOSITIONCIPHERALSOCALLEDCOLUMNSPERMUTATIONISATECHNIQUETOCHANGETHEORDEROFTHELETTERSINATEXTBYPLACINGITINAGIRD and we scan through it, breaking it up into readable text and the decoded text we retrieve is: A TRANSPOSITION CIPHER ALSO CALLED COLUMNS PERMUTATION IS A TECHNIQUE TO CHANGE THE ORDER OF THE LETTERS IN A TEXT BY PLACING IT IN A GIRD.

## 2.1.5 Extra Decoding challenge

For this bonus Lab we simply had to decode some basic hashes. For this I used online tools such as rapid tables and CyberChef.

1: V2VsbCBkb25lIHlvdSBzb2x2ZWQgcGFydCAx - (base64) = Well done you solved part 1

2: d8578edf8458ce06fbc5bb76a58c5ca4 - (md5) = qwerty

3: Pna lbh oryvrir Zvpebfbsg hfrq guvf gb rapbqr vgf ertvfgel inyhrf! - (rot13) = Can you believe Microsoft used this to encode its registry values!

4: (6== 5@?6[ E9:D :D 2 >F49 =6DD @7E6? FD65 6?4@5:?8 D496>6 - (rot47) = Well done, this is a much less often used encoding scheme

5: q293VFOcoKOlMKAmnKMyVUEbnKZtq2SmVUMypaxtqUWcL2g5YPO3MJkfVTEiozHtn

J4tp29fqzyh MlO0nTymYPOiozk5VT9hMFOgo3WyVUEiVTqiYPOao29xVTk1L2fAPt== - (base 64 – N-ZA-Mn-za-m0-9+/=) = wow! impressive this was very tricky, well done in solving this, only one more to go, good luck

6: EVIB DVOO WLMV LM HLOERMT GSV URMZO KILYOVN, SLKVUFOOB RG

WRWM'G GZPV BLF GLL OLMT ZMW BLF VMQLBVW RG. - (anagram solved with quip quip) = VERY WELL DONE ON SOLVING THE FINAL PROBLEM, HOPEFULLY IT DIDN'T TAKE YOU TOO LONG AND YOU ENJOYED IT

### 2.2: Reflecting on Decoding Classic Ciphers & Extra Decoding challenge

While I cannot profess to fully and completely understand each form of encryption and decryption we covered, I have gained a considerable overview of the concept of encryption and decryption. Rot5, 13 and 47 particularly will and indeed have proved very useful to me in my personal endeavors in capture the flag tournaments and challenges with my Ethical Hacking Society.   
  
Furthermore, I feel I gained valuable insight into the weaknesses of many forms of encryption. And most importantly I have learned that it is foolish to rely on encryption alone as a means of securing data or access to data. It is powerful only when used as a part of a whole security package.

# 3: Numbers Station

The numbers station was a fascinating lab. A look back at encryption that helped to shape the world in which we live today. Although it was difficult at first to grasp the concept of how the encryption method works, it is actually not massively complicated compared to modern standards though it is when viewed in the context in which it was used, meaning world war 2, now possible to look back and envision the men and women of both armies and resistance groups employing this encryption method to save lives.

The method works first by listening to a recorded and repeating signal broadcast from a numbers radio station. This repeating signal broadcasts the operators/stations callsign number. This is for identification purposes and not used in encoding or decoding.  
  
Next, the operator repeats a string of numbers. This is the one-time pad. Next, we have the cipher text. This is the secret message. We take the numbers contained in the one time pad, and we add them to the numbers of the cipher text sequentially, and we modulus this by 10. The resulting output is our cipher text.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | A | T |  | O | N | E |  | S | I | R |
| 2 | B | C | D | F | G | H | J | K | L | M |
| 6 | P | Q | U | V | W | X | Y | Z | # | / |

Here we have our decryption straddling table. This table gives us the ability to ultimately decrypt our decryption text. We start with the 26 letters of the alphabet and we extract a string of text made up a maximum of 10 characters, including 2 spaces. As we can see above we have used AT ONE IR, as they did in world war 2 and 2 spaces. The locations at which the spaces fall, denote that these numbers in our decryption text indicate a pair, which is lead by the number at this location. As we can see, our spaces fall at 2 and 6. This indicates that every leading 2 and 6 indicates that it, and the following number are a pair.

The final two lines are populated as follows. Our first space lands at 2, therefore our first line will be the 2 line, and the second space lands at 6 so the second line will be the 6 line. These lines indicated that any leading 2 will be read into the first line, and its pair number indicates at what point the decryption text should be applied. And any leading 6 group is applied to the 6 line which the second number of the group being decrypted according to its location on this line. So, for example, taking the grouped numbers 22 and 66, and applying these to the lines, we see that 2 indicated in the first pair that it should be applied to the 2 line, and the second two gives us a D. and the same logic applied to a ^^ pair, applied to the 6 line, at location 6 gives us a J.

Finally, in order to correctly populated this first and second line, we apply the letters of the alphabet, omitting any letters used in the original decode text string ‘AT ONE SIR’, never reusing any letters. The final Two can vary from table to table and are usually a hash and or a slash. These act as essentially wildcards. However, they are not needed in our decryption.

Once our table is fully populated, decryption is simple. We take our text and arrange it so that we have group up our pairs, in this case denoted by any leading 2, or 6.

We therefore apply our number to our table one by one. If a number is not part of a leading group denoted by a 2 or 6, it is applied directly to the first line containing “AT ONE SIR” and the location on which falls on this line gives us its corresponding letter. IF the number is part of a group lead by a 2 or a 6, it is applied to its corresponding 2, or 6 line, and its location on that line gives us its decryption character.  
  
Having broken down the inner workings of the numbers station encryption and decryption process we can begin to attempt to decrypt our message using the data provided by our number station and our lecturer.

Callsign: 39715

One Time Pad Key:

66153 77185 10800 54937 48159 83271 12892 07132 34987 53954 23074

Cipher Text:

66475 19274 92028 78494 24146 68542 17507 39398 32348 59378 70636  
  
One Time Pad Key + Cipher Text modulus 10 =

22528 86359 02828 22321 62295 41713 29299 36420 66225 02222 93600

Decryption text:

Every leading 2 & 6 = groups of 2 represented by []

[22]5[28] 8[63]59 0[28][28] [22]3[21] [62][29]5 41713 [29][29]9 3[64][20] [66][22]5 0[22][22] 93[60]0

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | A | T |  | O | N | E |  | S | I | R |
| 2 | B | C | D | F | G | H | J | K | L | M |
| 6 | P | Q | U | V | W | X | Y | Z | # | / |

Plaintext:

DELIVER ALL DOCUMENTS TOMMROW BY DEAD DROP A

# 4.0 Salted Hashes

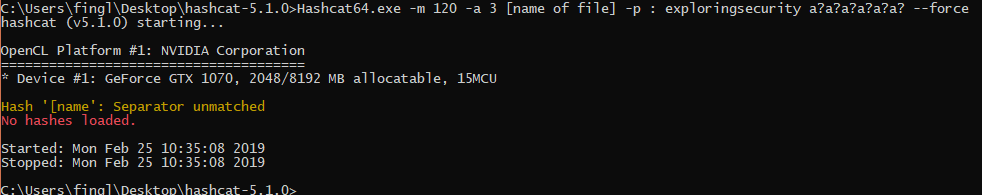
For this Lab we are tasked with assisted a police investigation. We are tasked with cracking 6 salted hashes to aid the police. To facilitate this, based off the polices initial investigation and attempts to crack the passwords we are provided with some simple information and a table as seen below.

* The password policy file was changed in May 2010, passwords created after this date are alphanumeric 5-7 characters in length. Passwords created before these dates are believed to have consisted of only digits and 5-7 characters in length.
* It’s believed that all passwords have used the same salt, and that the value is somewhere in our data.
* The database dump was from MySQL database  The site’s domain name is [www.exploringsecurity.com](http://www.exploringsecurity.com)
* Some of the captured JavaScript code from the site, reveals the salt format as CommonHash($salt,$pass)

  
  
From this information I was able to discern some facts and make some educated guesses to begin my investigation. With some passwords being only numeric it is reasonable to presume that these should be crack-able with a brute force attack. The passwords that are Alpha numeric may require a wordlist, however we will first begin with a brute force attack.  
  
All passwords contain the same salt, so if we examine the information provide we must search for a string that could be used repeatedly. The only piece of information that fits this requirement is the sites domain name. Therefore, we will use this as it is presented as the salt.  
  
The captured java code reveals the salt type. There are two types of encryption that match this type of salt/hash type. MD5 and sha1. MD5 is not very secure and reasonably easy to crack. So, I will assume it is sha1 to start off. If necessary I will return and try MD5.

## 4.1 Cracking the Hashes

I began my attempts to crack the first hash as I have done before, entering the salt directly in the command and only having the hash in the has file. In the past this proved successful. Where ‘John the ripper has always be able to discern the hash and salt for itself, Hashcat has always required my intervention to discern this in the past. The syntax I used was:

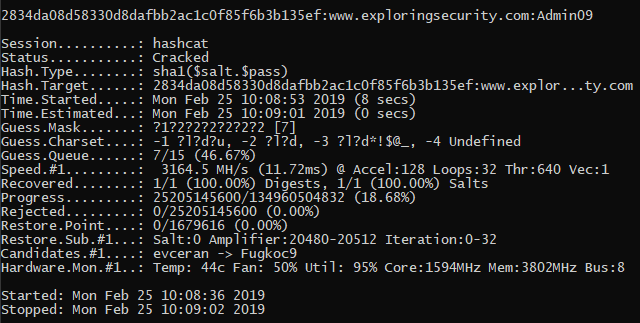


*Here we see the separator error*

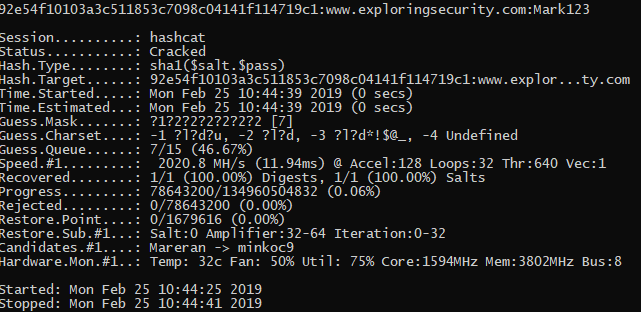
This syntax returned an error regarding a separator mismatch. I could not work out this error. However, during my investigation, I came across forum post where people suggested adding the salt in the file. With no other options I tried this. The format of the file to be cracked was now as follows: 2834da08d58330d8dafbb2ac1c0f85f6b3b135ef:www.exploringsecurity.com.

The syntax also needed to be adjusted. The new syntax was: hashcat64.exe -m 120 -a 3 hash1.txt.

This proved successful and I cracked the first hash in under 60 seconds having passed through 6 guess ques and getting a hit on the 7th as seen below with the password being: Admin09.

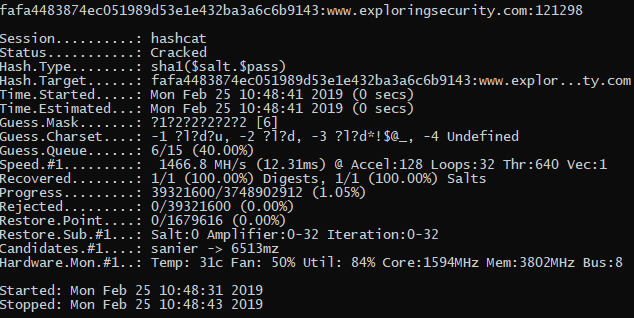


We then proceed to hash number 2 and applied the same method, which again proved successful. The password this time was: Mark123.

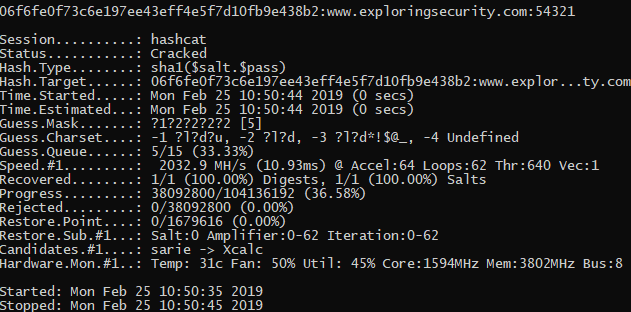
  
  
Next up, hash number 3. And we have another success. The password this time was: qwerty.

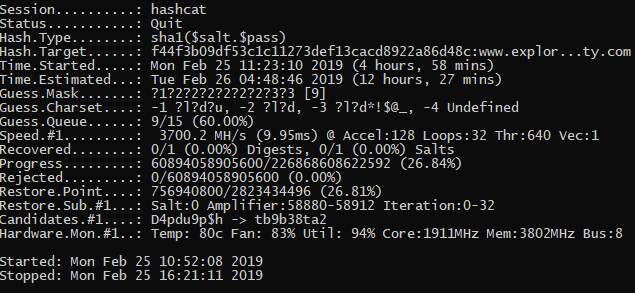


I now moved on to has number 4. This again proved successful. This time on the 6th guess mask with the password being: 121298

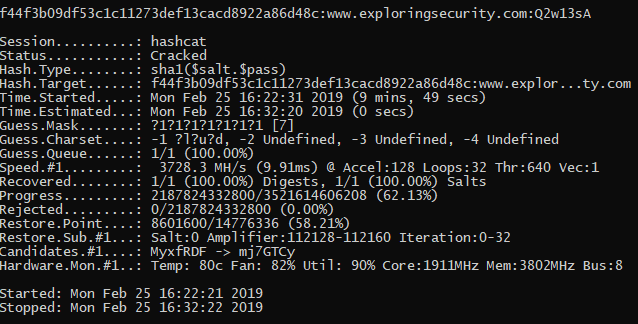


And on to hash number 5. Another success, this time on the 5th guess mask, with the password being: 54321.



Finally, we move on to the last hash. This has proved more difficult. Passing through 7 guess masks to the 8th. This concerned me as I felt like I was getting into areas where passwords could take days to crack. However, I left it running for a few hours to see if anything came of it.  
  


Above we can see that the attack has been running for almost 5 hours. I was reasonably sure that something was wrong at this point. So I cancelled the attack and decide to manually force it to re run through the 7th guess mask again before re-evaluating my thought process. This proved to be the correct approach. I cannot say why, but the automatic attack was failing to fully attack the password in the 7th guess mark range. A manual attack with the syntax: C:\Users\fingl\Desktop\hashcat-5.1.0>hashcat64.exe -m 120 -a 3 -1 ?l?u?d hash6.txt ?1?1?1?1?1?1?1, adding a character set and defining the character set as specifically 14 characters long, placing it in the 7th guess mark range forced Hashcat to properly attack the password in this range and sure enough this cracked the password, although it too 10 mins rather than 1 this time. With the password being: Q2w13a.



## Reflecting on Salted Hashes:

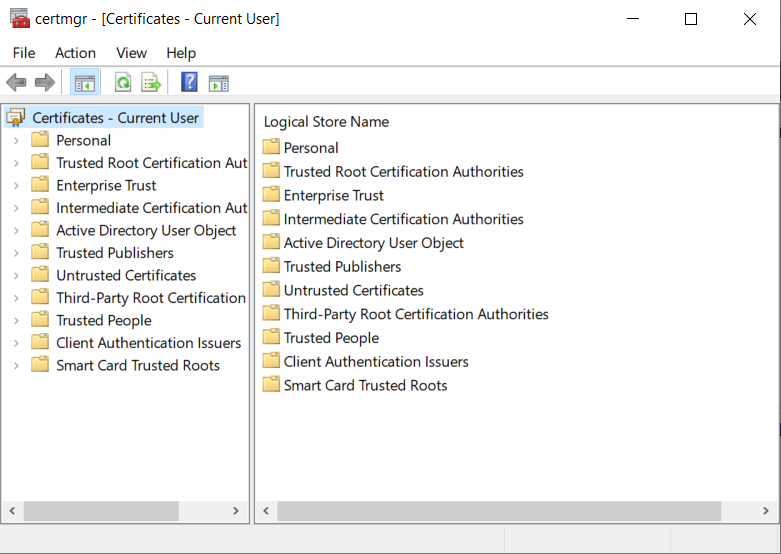
This was an enjoyable lab. I learned a few useful things about Hashcat. Where John the Ripper is a useful and powerful tool particularly with its ability to automatically examine formats, Hashcat is more complex. Once the user has correctly determined the format and entered the right information is extremely powerful and more diverse in its ability to attack passwords.  
  
Having made the incorrection assumption based off my previous use of Hashcat that it cannot crack a hash file that contains the salt, I have learned that I must be more careful in the syntax that I use and the way I enter the salt. I must be very careful when examining the format of the password and salt combination.  
  
I am as yet unsure why the given format is $pass.$salt and yet the salt is actually entered after the password in the hash file, however in this case it is not important for me to understand why, only that it is entered in that order.

# 5.0 Digital Certificates & Signatures

Next, we shall investigate digital certificates and signatures beginning very simply with investigating those found on our local machine before generating some of our own locally.

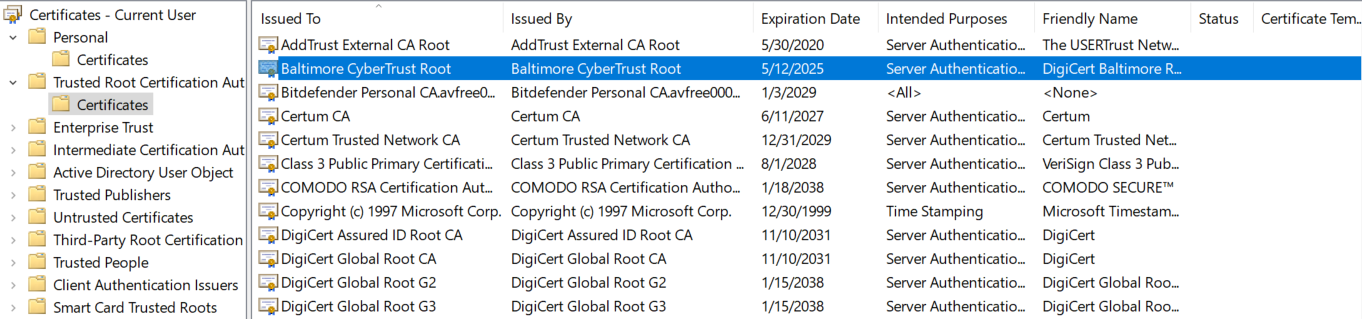
### 5.1 Viewing Certs on Your PC

For this lab we simply have to find and view the certs stored on our local machine, both in windows and in our preferred browser. We begin, by opening the certificate management tool by opening our search function in windows and navigating to certmgr.msc.



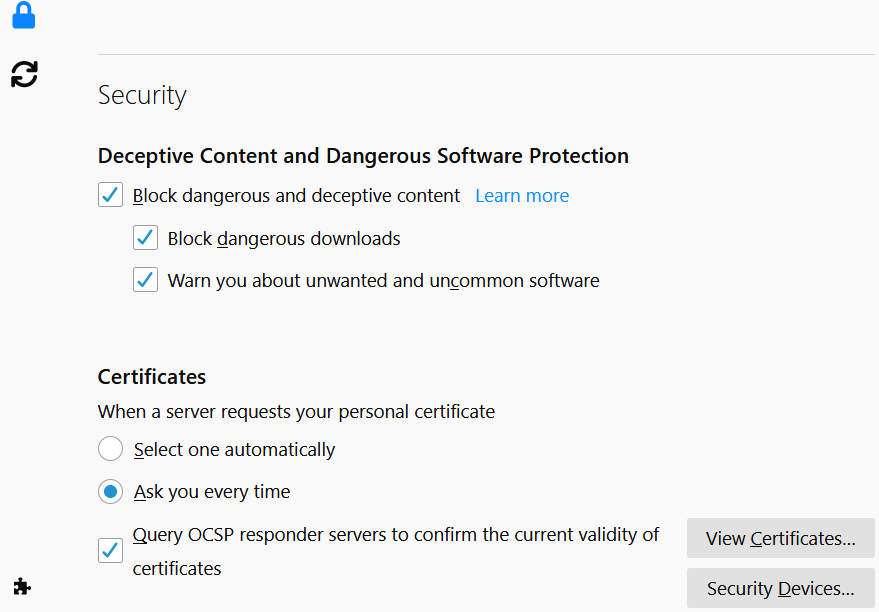
*Windows certificate management tool.*

From here, we may explore and view the certificates stored locally on our device. There are hundreds of certificates for us to explore, a sample of which can be seen below.



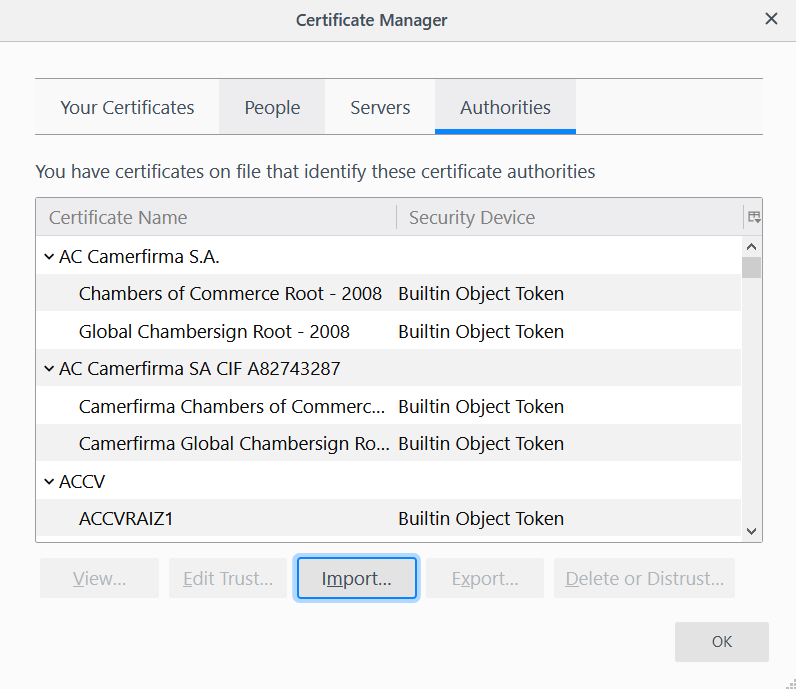
*A sample of certificates stored on my local machine.*

Now, I examined the certificates stored in my browser. I did this by entering the options menu in Mozilla Firefox, selecting the security lock icon and scrolling down to security, before finally selecting ‘view certificates’.



*How to enter FireFox’s certificate browsing menu.*

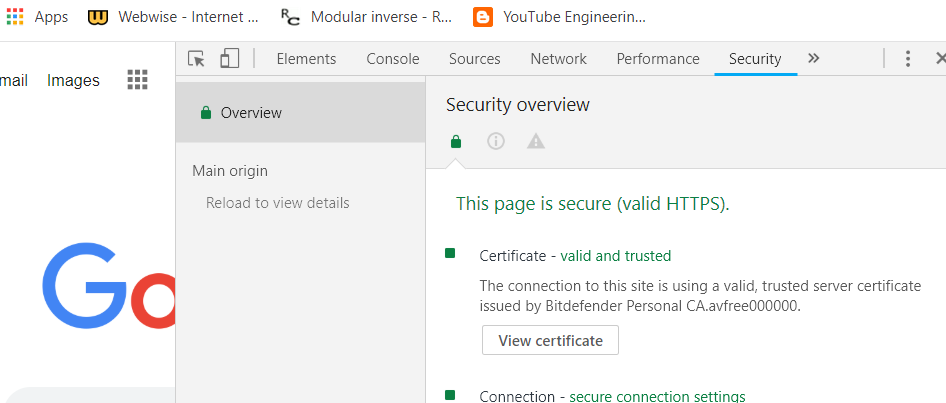
Within the Mozilla Firefox certificate manager, we can again view the many certificate stored locally within our config files. We can also view in depth information, edit our trust settings, import or export individually certificates as well as deleting or distrusting certificates which may have expired or been revoked by the issuing authority.



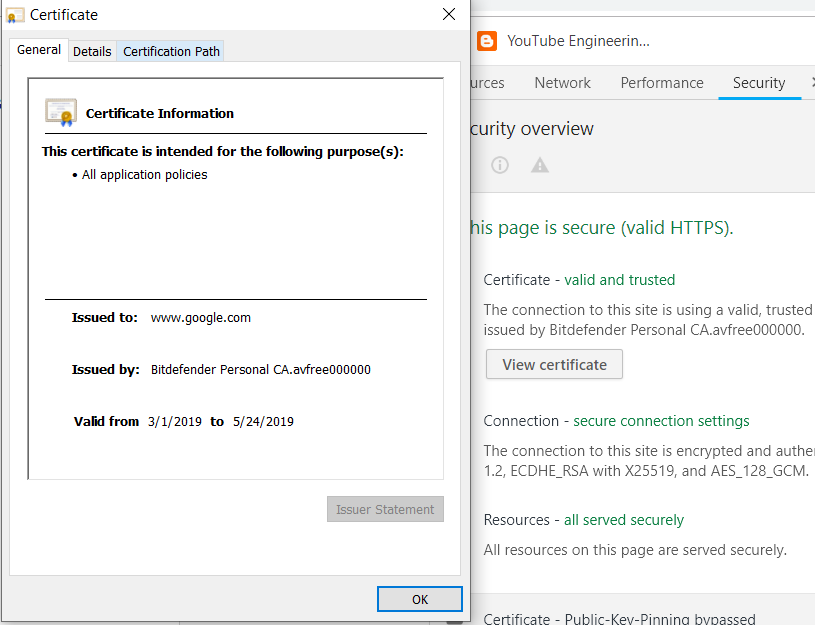
*Mozilla Firefox’s internal certificate management tool.*

Finally, we viewed a certificate in Google Chrome which takes a different approach. Rather then a management tool, Google Chrome allows for inspecting the certifate directly from the page. This is done by opening the developer tools as follows:

* Simply press F12. Or…
* Select the three dots on the upper right-hand side of the screen, the right of the URL bar.
* Next, select more tools. A pop-up menu will appear.
* From the pop-up menu select developer tools.
* Once the developer tools screen is open you will see developer tools overlaying the page whch you are viewing.
* Select the security tab from within the developer tools options, and you will be presented with a button to view the sites certificate, as seen below.



*How to select the options to view a certificate from within the developer mode options in Google Chrome.*

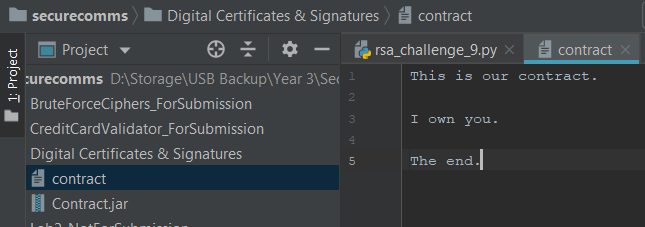


*Here we see a certificate as viewed through Google Chrome.*

### 5.2 Exchanging Signed Documents – JAR Tool File

For this lab we are tasked with creating a contract document, which we will then place inside a java jar file before generating signature keys which we will then apply to our contract jar.

The first step is to create the contract.txt document which I did via Pycharm. Next, I used the command: jar cvf Contract.jar contract. This created a new .jar file called Contract, which contains our original contract.txt file.



*Here we see our original contract.txt file and our new Contract.jar fle.*

Then we generated signature keys for our file. To do this we must create a key pair. And to do that we must create a keystore. We shall call out keystore markstore3 as per the lab instructions. We create our keystore with the command: keytool -genkey -alias signLegal -keystore markstore1.

Next, we must enter the following details to complete our keystore:

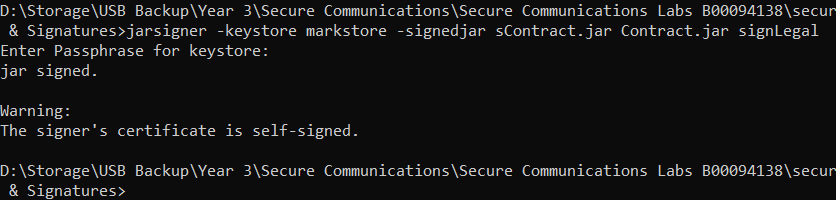
* Enter keystore password: balloon53
* What is your first and last name? [Unknown]: Mark Cummins
* What is the name of your organizational unit? [Unknown]: Legal
* What is the name of your organization? [Unknown]: XYZ
* What is the name of your City or Locality? [Unknown]: Blanchardstown
* What is the name of your State or Province? [Unknown]: Dublin
* What is the two-letter country code for this unit? [Unknown]: IE

We Should then be asked to verify if the information entered is correct. Before finally, being asked to enter our keystore password: cat876, however, I was not prompted to enter a password at this point.



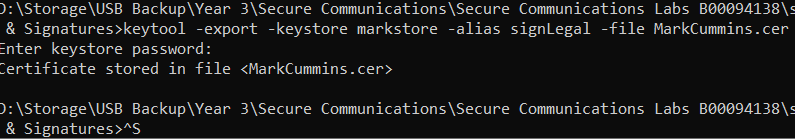
*Creating our keystore.*

Now we must sign our jar, by applying the private key of our key pair to the Contract.jar file. This is done by entering the command: jarsigner -keystore markstore3 -signedjar sContract.jar Contract.jar signLegal.  
  
Having done this I was issued a warning which can be seen below, informing me that the certificate is self-signed.



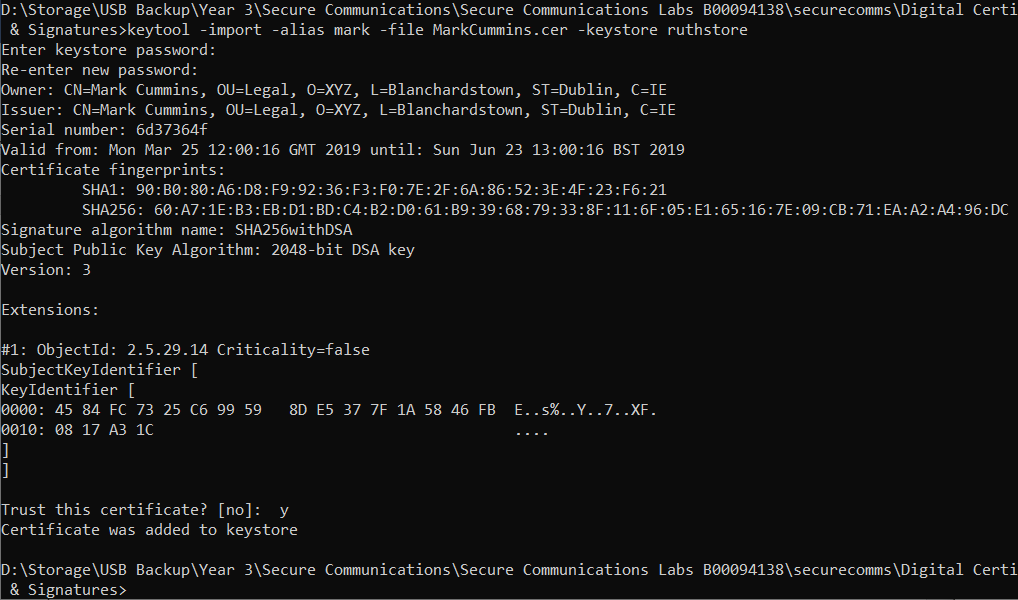
*Here we can see the warning I received upon applying the signature to my Contract.jar file.*

Next, we will create a second keystore to simulate receiving the file on another computer. Within this keystore we will attempt to import the public key from the marstore3 keystore, to access the signed Contract.jar file. But first of course we must export the public key so that we can offer it up to other users to decrypt anything encrypted with our private key. This we will do by exporting our certificate to a new file with the following command: keytool -export -keystore markstore3 -alias SignLegal -file MarkCummins.cer.



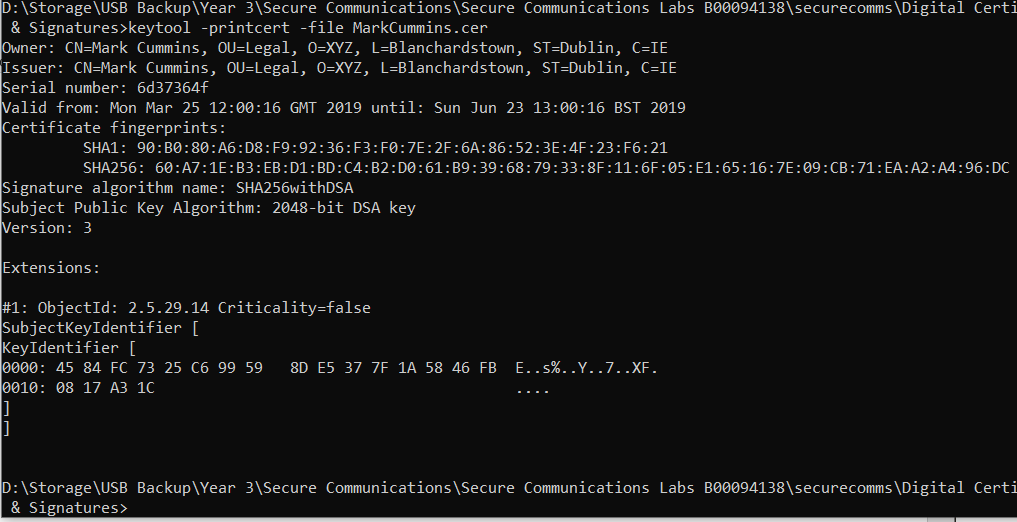
*Here, we have exported our certificate so that our public key can be extracted by other users.*

Rather than using the previous command we will attempt to import and create the new keystore called ruthstore at the same time using the command: keytool – import -alias mark -file MarckCummins.cer -keystore ruthstore.



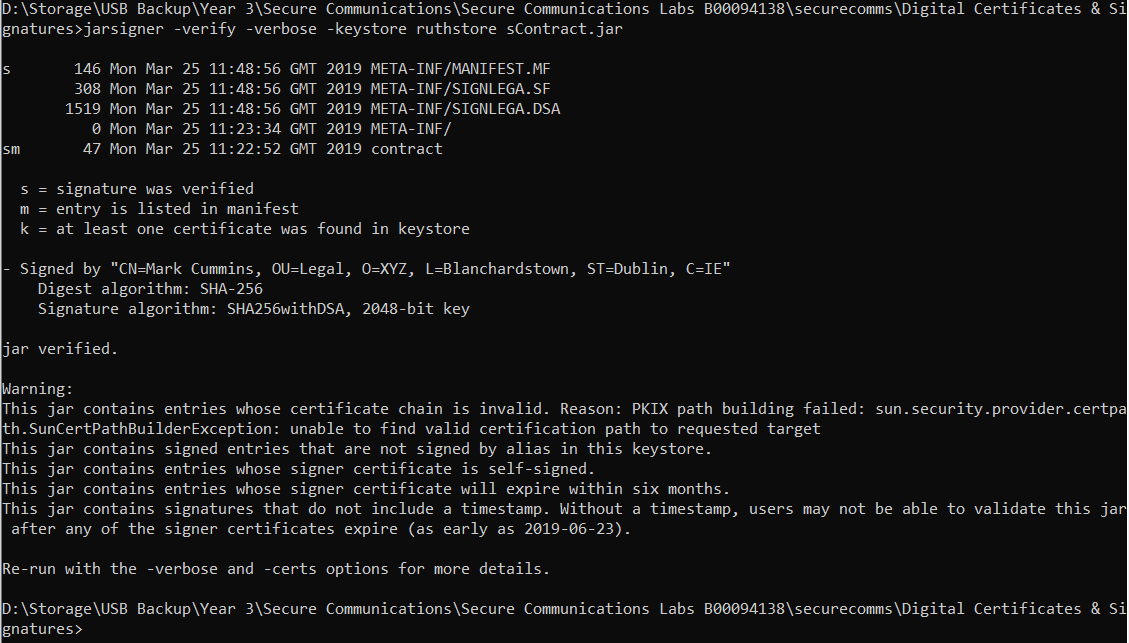
*Here, we have created the ruthstore keystore and imported the public key from the MarCummins.cer file, trusting the certificate in the process.*

Next, we shall test the ability to print the details of the certificate via the command: keytool =printcert -file MarckCummins.cer.



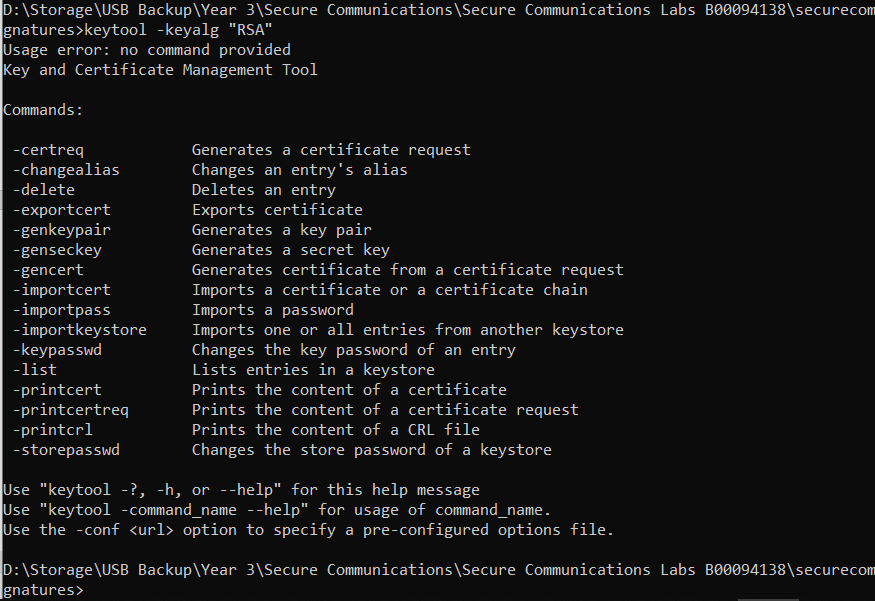
*Here, we see the details of the certificate printed to the screen.*

Now we can attempt to verify a signed jar file as Ruth using the command: jarsigner -verify -verbose -keystore ruthstore sContract.jar. This was successful but once again I received an unexpected warning due to self-signed certificates.



*Here we see the jar verified, but with an unexpected error due to self-signed certificates.*

The final part of this lab suggested that we must try to substitute our file scontract.jar with a fake and attempt to see if we can create an MD5 collisions using the command: -keyalg “rsa”. This command allows us to replace 0hte standard encryption algorithm with MD5. I could not fully understand what I was being asked to do nor could not get the command to work, as seen below.



*Here, I attempted to alter the encryption algorithm as requested with no success as the command is not recognized.*

# 6.0 Wireshark

## 6.1 Setting Up FTP

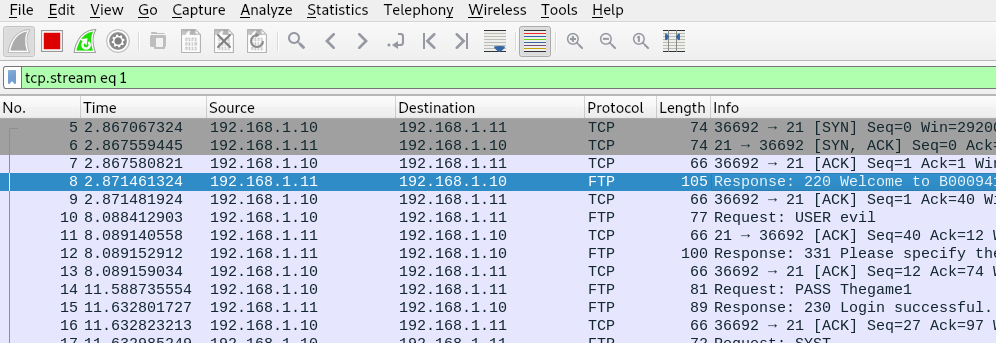
I elected to use VFSTPD server on a ubuntu VM as my FTP SERVER. I installed the program with the command: sudo apt-get install VSFTPD. I then navigated to the /ect/vsftpd.conf file and edited the settings, to either enable or add the following config settings:

* local\_enable=YES
* write\_enable=YES
* #anonymous\_enable=NO
* #anon\_upload\_enable=YES
* ascii\_upload\_enable=YES
* ascii\_download\_enable=YES
* user\_sub\_token=$USER
* ftp\_banner=Welcome to b00094138 FTP saervice
* chroot\_local\_user=YES
* chroot\_list\_enable=YES
* chroot\_list\_file=/etc/vsftpd.chroot\_list
* local\_root=/home/$USER/Public\_html
* allow\_writeable\_chroot=YES
* ls\_recurse\_enable=YES

Next, I created a file in the /etc folder called vsftpd.chroot\_list. I created a user called ‘evil’ with the password of ‘Thegame1’. I added this user to the vsftpd.chroot\_list so that this user, upon entering the correct password would have ful access as permitted by the vsftpd.config file.

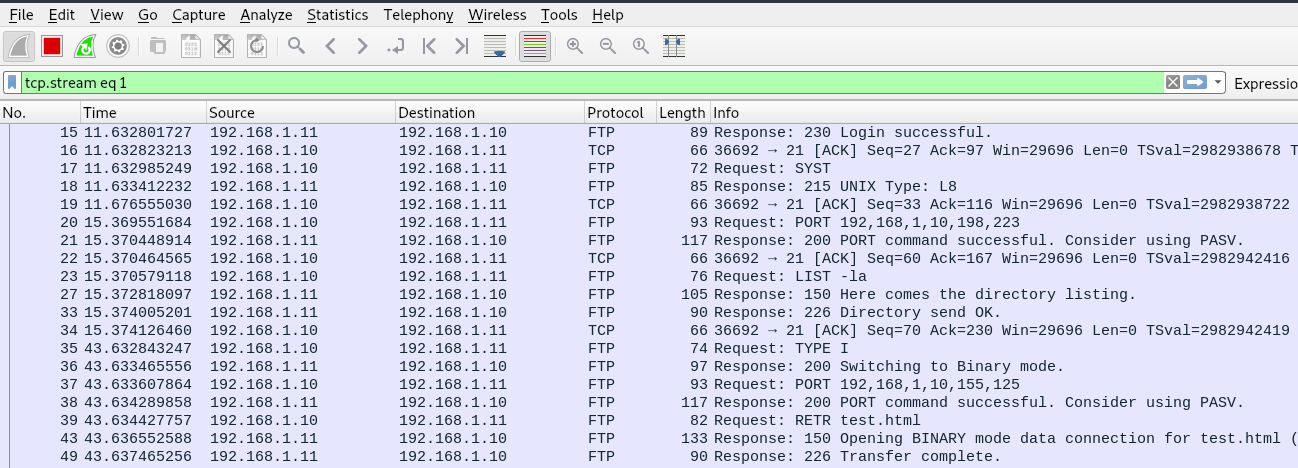
Our assigned task is to the connect and download the test.html file contained on the default log in folder location. While doing this we must capture traffic and analyse the data. We must try to extract the user name and password which is sent via plaintext. We should then download the html file and attempt to view the contents of the file again in plaintext which will allow us to rebuild the html file.

Having set up the FTP server on our ubuntu client, we logged in via our Parrot OS client with the command FTP 192.168.1.11. We are prompted to input our user name and password.



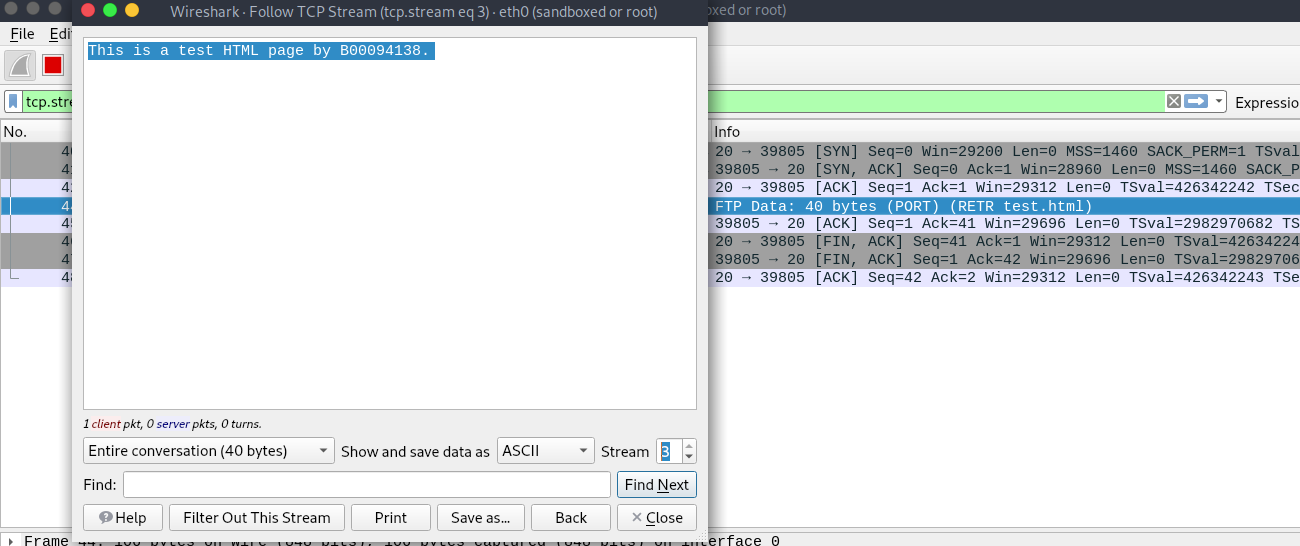
*Here, we can see our login is successful, and our username and password is captured in plaintext.*

Having captured our username and password, we next download the file from the FTP folder location.



*Here we see our file transfer of test.html has been successful.*

Now we will attempt to find and follow the stream, to identify the contents of the file in plaintext.



*Here, we see the contents of the file in plaintext.*

Having identified the stream where our file was transferred, we follow the stream and step thu=rough the stream packet by packet until we identify the plaintext contents of the file. This allows us to reconstruct the file by simply creating a new HTML file and inputting the contents of the captured HTML file.

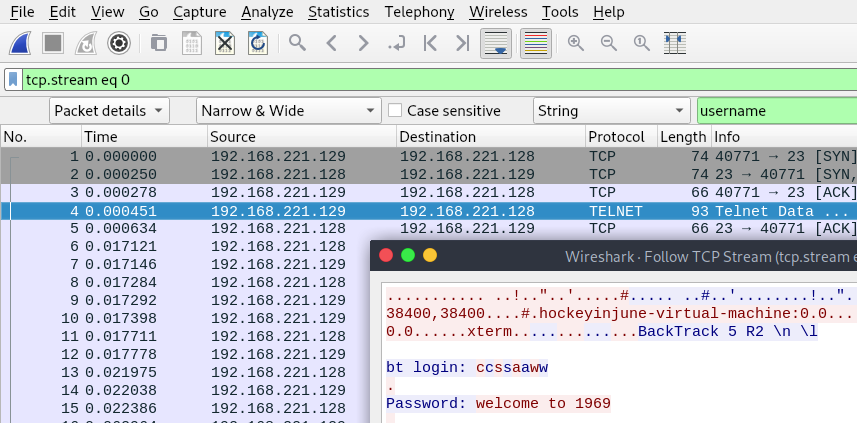
## 6.2 Pcap Challenges

### 6.2.1 Pcap Challenge 1

For Pcap challenge 1, we are tasked at analysing a Pcap Wireshark packet, and attempting to extract the username and password from the data. This was a very simple task. Upon opening the Pcap file, telnet traffic was immediately identified, and telnet being a very old program using very old protocols is known to send traffic unencrypted like standard FTP. So the telnet data stream was followed and the details where extracted easily in plain text.

Login: ccssaaww

Password: welcome to 1969.



*Here, we can see the plaintext username and password extracted from pcap file 1.*

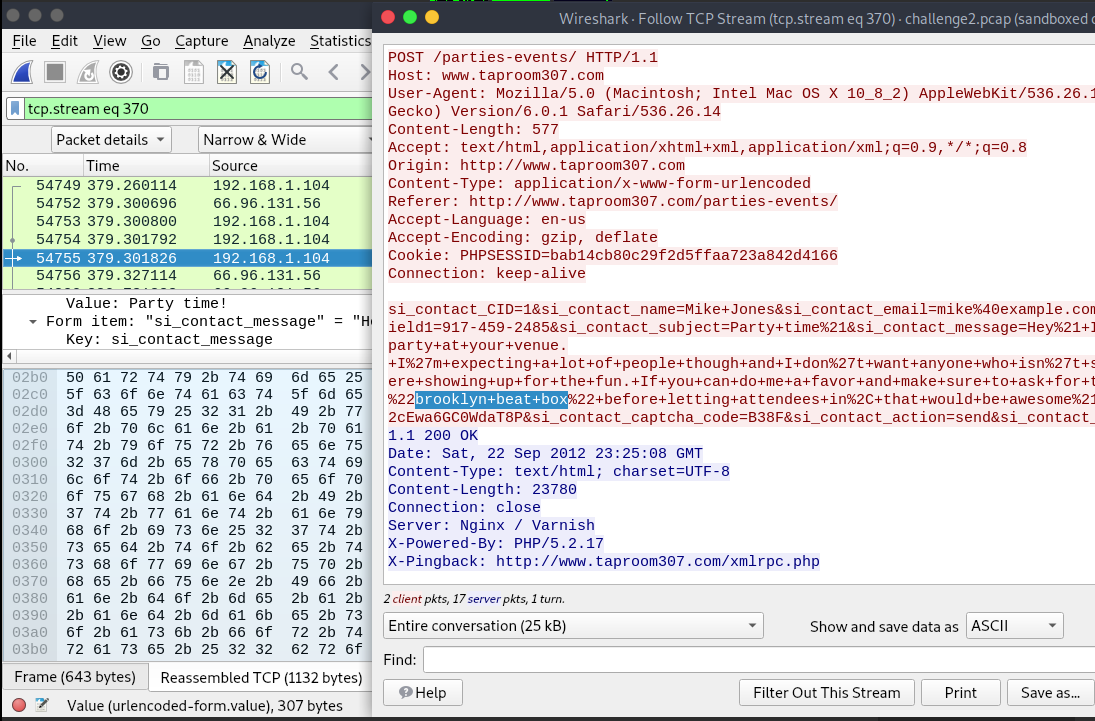
### 6.2.2 Pcap Challenge 2

For the second Pcap Challenge we are tasked with finding the secret password challenge phrase to be used to access the private party being arranged through the communication captured on the Pcap file.

This was simply a matter of going through the Pcap file, stream by stream searching for Strings which may be relevant using keyword phrases inspired by this clue given in the lab notes:

“Some dude I know is planning a party at some bar in New York! I really want to go but he's really strict about who gets let in to the party. I managed to find this packet capture of when the dude registered the party but I don't know what else to do. Do you think there's any way you can find out the secret password to get into the party for me? By the way, my favorite hockey player ever is mario lemieux.”.

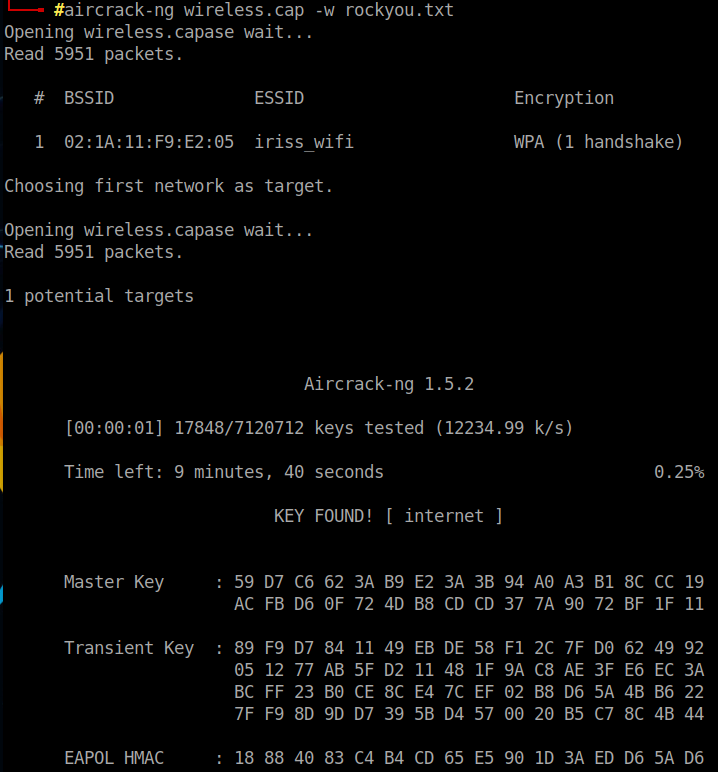
While this was a simple lab, I did struggle a little to understand the application of filters in Wireshark. I spent some time struggling with an unknown issue that preventing me from properly searching using keyword string searches. Also, I was searching the wrong type of data, searching the packet list. After about 20 mins of struggle I restarted Wireshark and altered my filters to search first byte data the packet details. I search strings such as Mario and secret and New York and party, and finally hit pay dirt with party. Even then there was a lot of data to sift through and I could not find a more efficient way to search through the streams. But I did finally retrieve the secret phrase to be used to gain access to the party which was: Brooklyn beat box.



*Here, we see the secret phrase extracted in plain text using a keyword search for the string party in the packet details.*

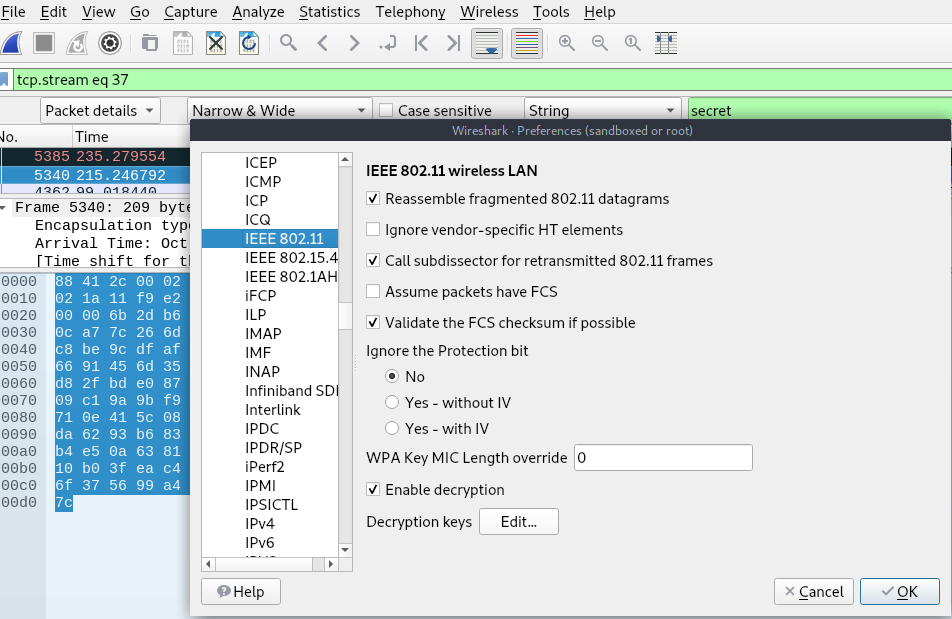
# 7.0 Encrypted Wireless Capture

For this challenge we are tasked with decrypting a hidden message within an encryption capture file from a wireless network. The first step of this is second nature at this point which simply involves using Aircrack to attack the WPA encrypted file (wireless.cap) with a wordlist (rockyou.txt). I broke the encryption in 1 sec meaning the password was undoubtedly weak. The command used was: aircrack-ng wireless.cap -w rockyou.txt. As can be seen below, the password was ‘internet.



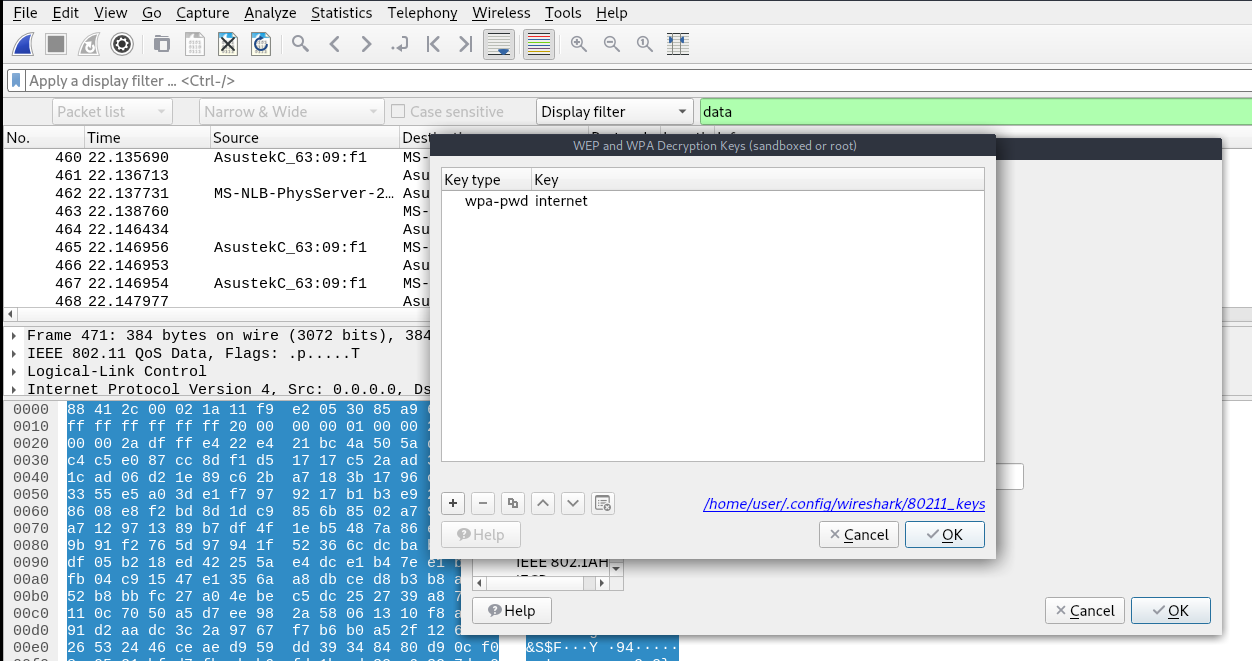
*The wireless.cap file cracked with the password: internet, revealed.*

Next, things got slightly more difficult as we ventured into unfamiliar territory. We had to figure out how to use the encryption key within Wireshark, to apply the key to the file and decrypt the captured traffic. This took a bit of guess work as some of the information I found online proved a little out of date. But finally, I discovered it was a matter of going into the edit tab and selecting preferences. From here we selecting the protocols drop down menu and scroll down to IEEE 802.11 and select it. Once selected we chose the edit decryption keys option on the right as seen below.

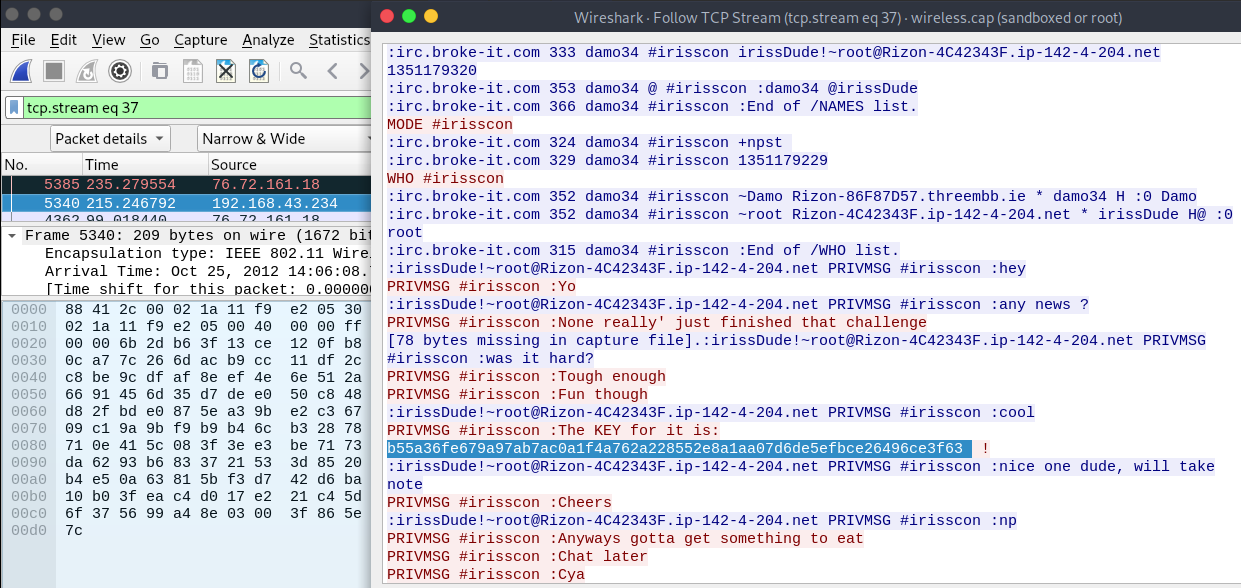


*Select Edit key to enter our decryption key.*

The last step in decrypting the cap file is to apply the encryption key. This is done by selecting the small plus, which allows us to enter a key type, which in this case is wpa-pwd and the key which as previously shown is internet. Once we hit ok, the file should be decrypted, and in our case, it is.



Here, we see our key entered to decrypt our .cap file.



Having explored the file, we find this IRC conversation where two users are discussing the key to a CTF challenge. The secret message is the key to the CTF challenge: b55a36fe679a97ab7ac0a1f4a762a228552e8a1aa07d6de5efbce26496ce3f63.

# 8.0 Securing Internet Traffic

This lab tasks us with investigating SSL and TLS security by examining their handshake protocols by using some tools including Wireshark and some built in tools at www.ssllabs.com.

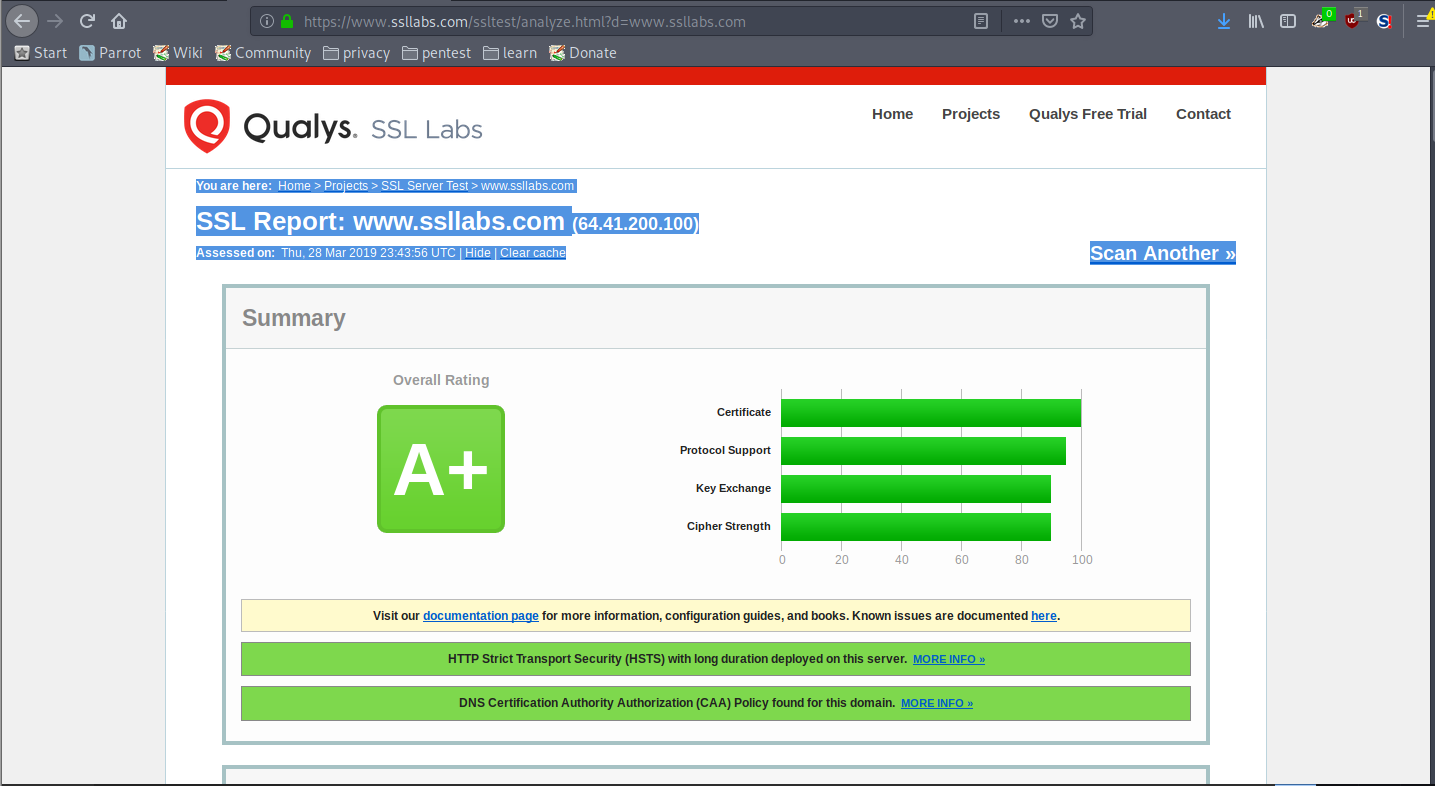
I began this lab by starting up Wireshark on my Parrot VM and connecting to https://moodle.itb.ie. This allowed me to view the TLS handshake live. Interestingly, as was pointed out in the lecture just today we see that despite it not being the latest version of TLS, even our colleges Moodle page is still using version 1.2 TLS. We can also see that the client initiates the handshake process with its client hello message. This is followed by the servers Server Hello message. This facilitates the exchange of protocols, and certificates which will govern how the client and server will talk and ensure authentication in the communication between the two parties.

Next, the key exchange takes place ensuring that each end of the communication has the opposing ends public key in order t be able to decrypt all traffic. Finally, the handshake is ended, and a new session ID is started, and the server terminates the handshake., and encrypted traffic begins to flow.



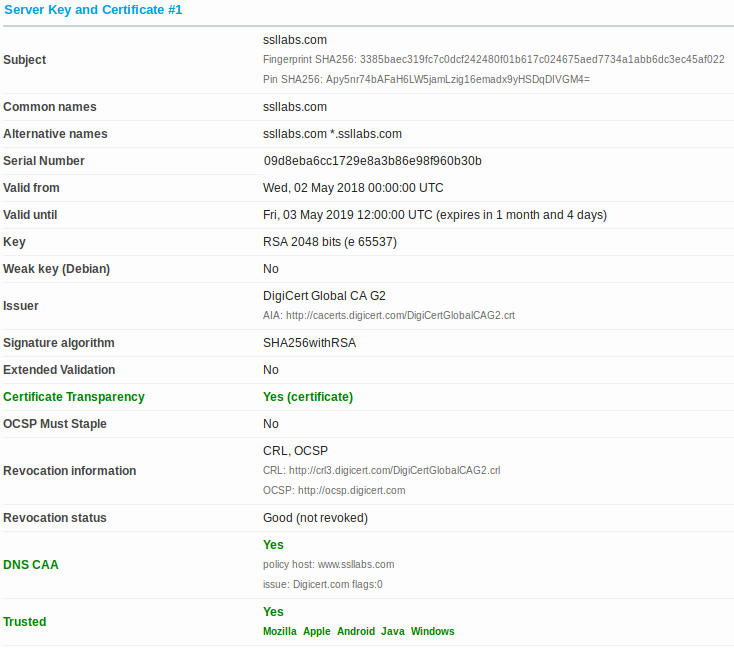
*Here, we can see the TLS handshake in its entirety.*

The second part of this lab had us investigate some of the tools and resources available at Qualys SSL LABS. I elected to run the SSL test on the services own website and was not surprised to see it reived it an A+ rating, as one would expect.



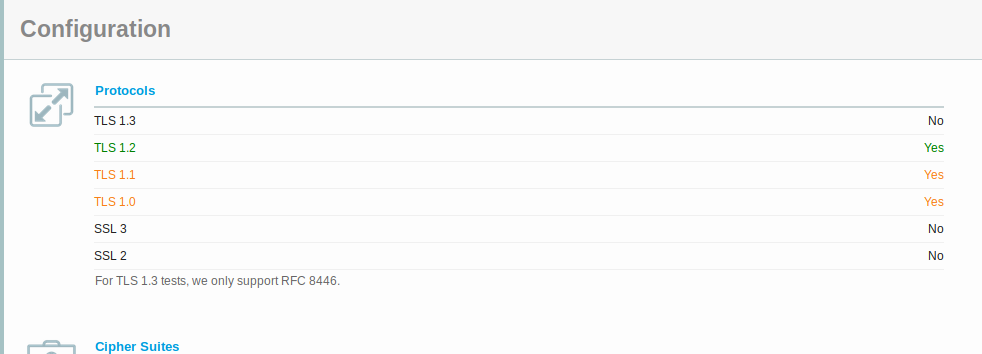
*Qualys SSL LABs own website receives an A+ rating from its own SSL test.*

Below, we see some interesting information pertaining to the site’s security certificate, including the key algorithm type used as well as the expiry date.



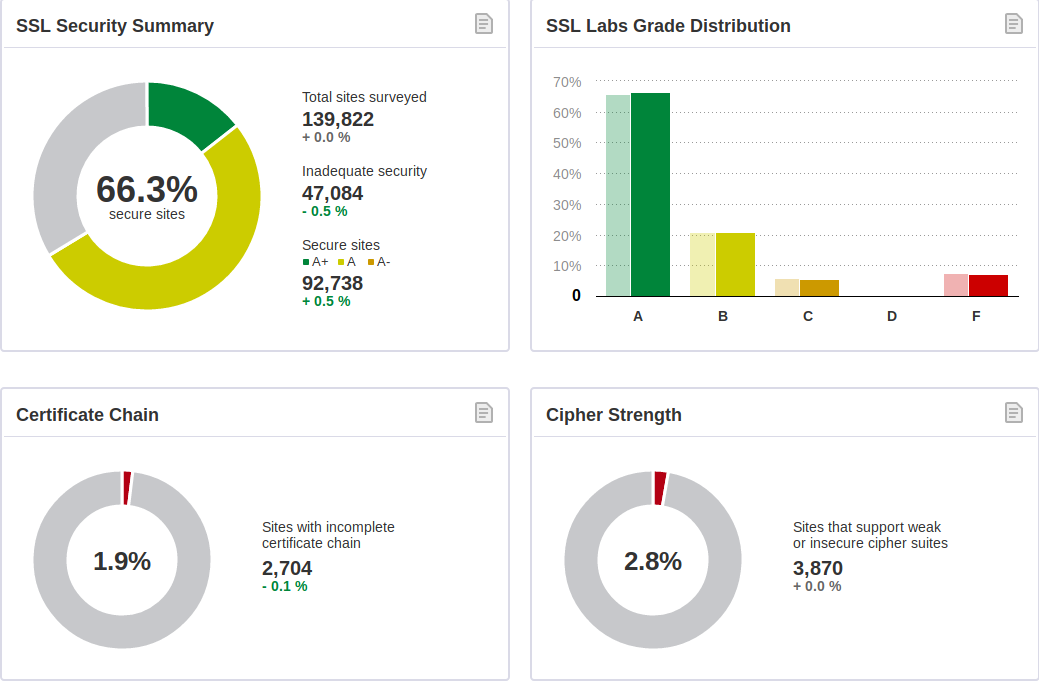
*Here we see the websites security certificate information.*

Another useful piece of information can be seen below. We can see a compatibility chart for TLS handshake protocols. Once again, we see that even this site is lacking compatibility with TLS version 1.3 which is suffering from a slow roll out, much in the same way IP version 6 is.



*TLS handshake protocol compatibility chart for https://www.ssllabs.com.*

The websites Pulse report reveals interesting date sets relating to security levels of sites scanned by the service. We can see statistics on sites with incomplete certificates and sites that support weak or insecure encryption methods such as SSL for example. The chart supports the supposition that TLS 1.3 is indeed slow to be implemented among most sites scanned.



Qualys Pulse report, March 02, 2019.

The SSL rating guide provides a thorough point by point breakdown of the key factors of encryption and the roles they play such as protocol support, cipher strength, key exchange certificate inspection and the change log for SSL/TLS covering some of the most important changes in the various iterations of SSL/TLS.

# 9.0 TOR & the Dark Web

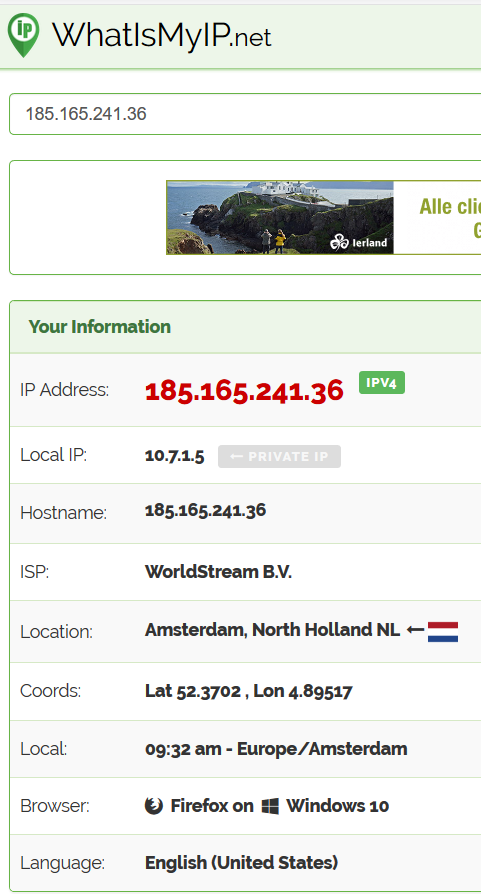
For this lab we are tasked with installing and using the Tor browser to test its ability to hide our IP address. Tor is web browser design by the US Navy to hide the geo-location and IP address of user’s to help hide a ships location. It does this by adding a layered encapsulation header to a packet. Each packet contains a series of layered encapsulations, each one contained the destination of the next hop along with the hops public key. The next hop can only decrypt the previous hops encapsulated layer and the only data contained inside this layer is the next hop. In visual terms it works as follows.

(((A-B)a) - ((B-C)b) – ((C-D)c)))

A sends the packet to B. B uses A’s public key to decrypt the data, which tells it to forward all data to

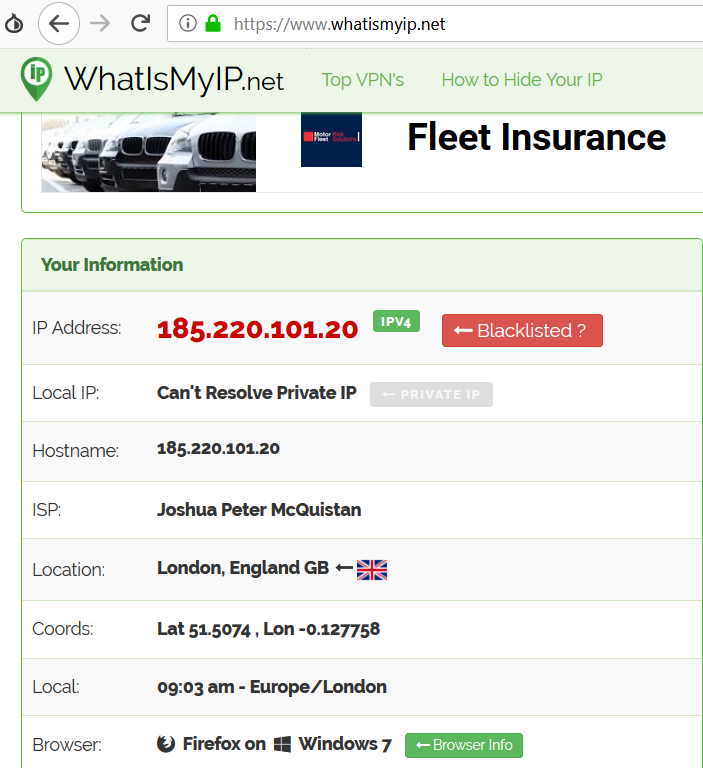
C. C then use’s B’s public key to decrypt the next layer. This tells C to send the packet to the final destination, D. When the packet arrives, it strips away the final layer revealing the inner IP packet ins standard encapsulated form. The guarantees the senders location remains unknown. Tor does not however encrypt any information other than the IP address of the sender to ensure anonymity.

Next, it was time to test a normal browser versus Tor. I was also using my VPN too. Therefore, I expect my location to show up as the Netherlands via a normal browser. It does, as seen below. I first tested to see if my IP showed up via a normal browser, which is did.



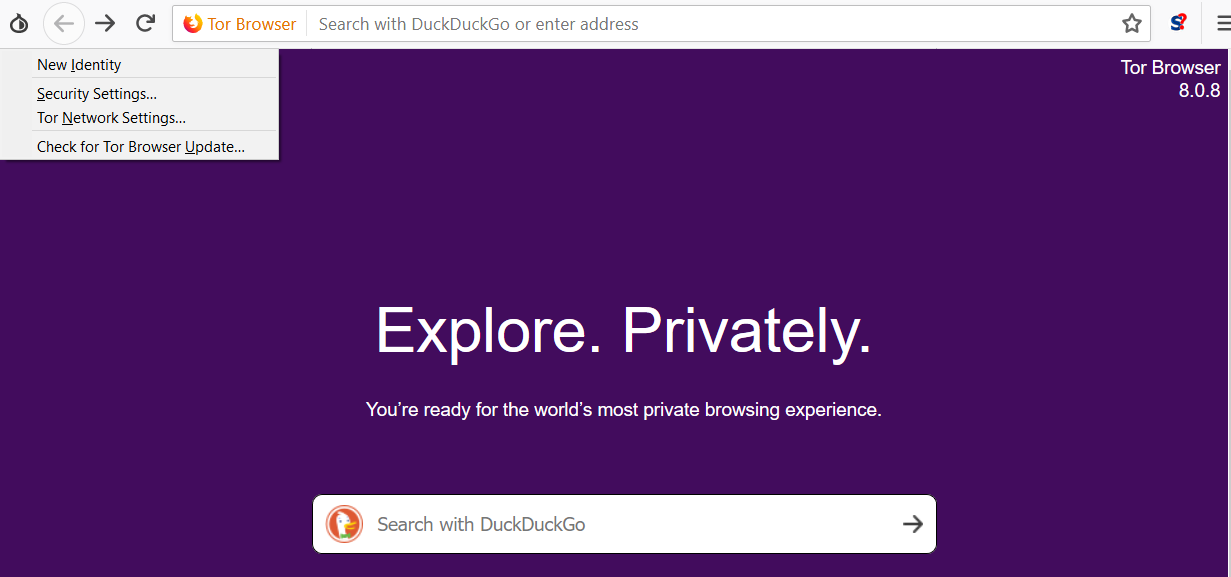
*My public IP as displayed via Firefox web browser.*

Next, I tested to see if my IP would be displayed while visiting the same site through the Tor browser. What I found was that my IP was listed but incorrect, and my location was listed as London, which it is not.

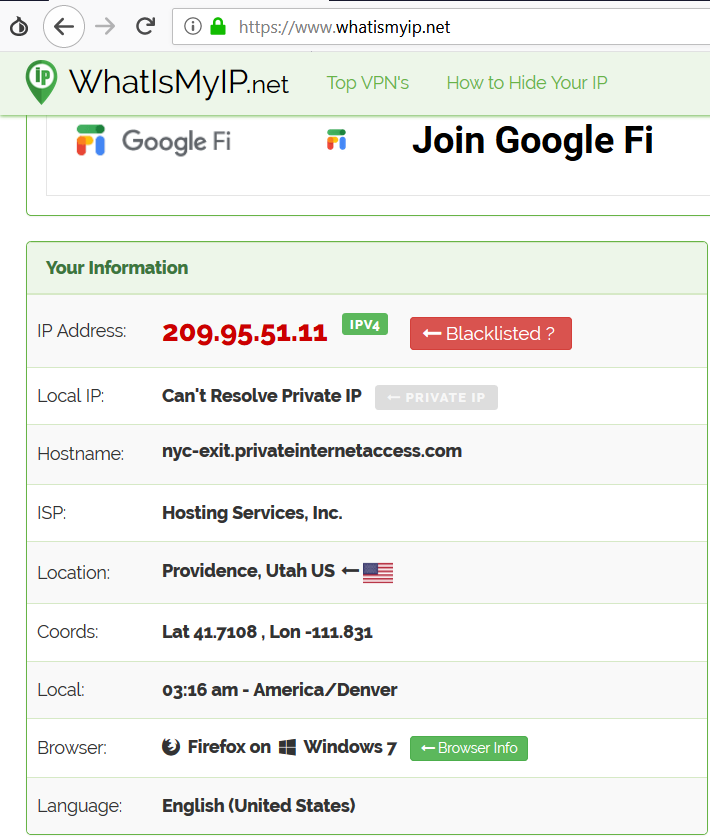


*My IP and location as displayed through the Tor browser, which are both incorrect as one would expect while visiting this site on the Tor browser.*

Next, I altered my address, to test if the new identity function correctly adjusts to once again hide my location via a different node. This is done access the small onion icon at the top left of the Tor browser window and select the ‘New Identity’ option.



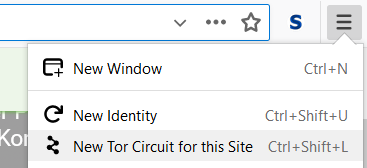
*Here, I select a ‘New Identity’ to test the new identity function to see if they are changing.*



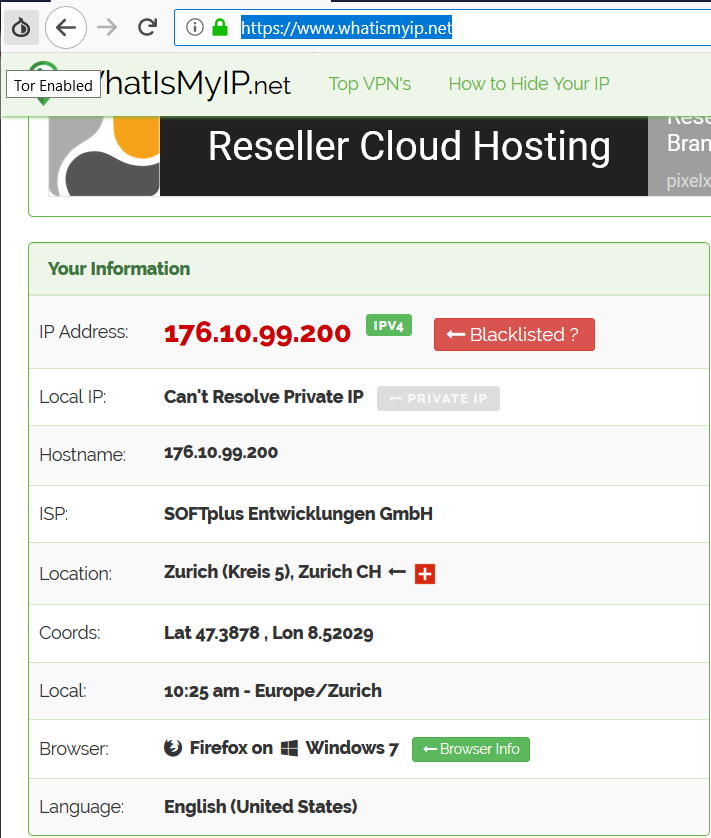
*Here we see that my IP is once again showing a different and incorrect IP and location.*

As we can see above, the circuit through the nodes is changing and assigning as new anonymous identity.

Next up, I tested the ‘new circuit function’. This is accessed via the three lines which allows one to access the options.



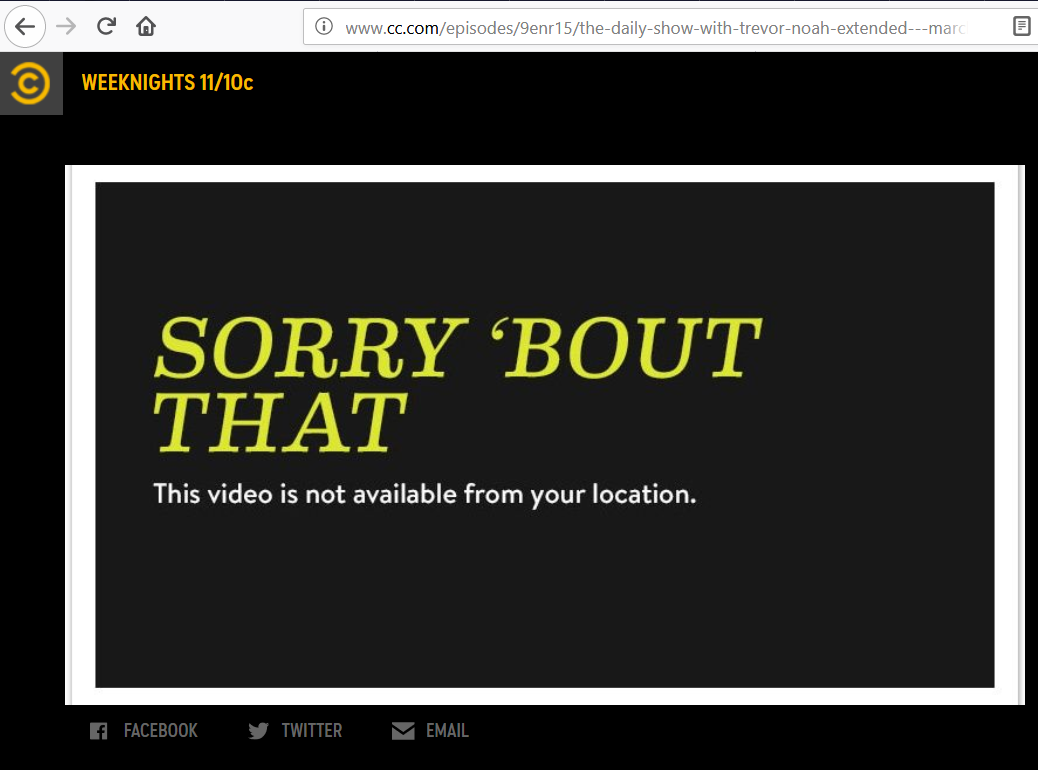
*Here I access the new circuit function.*



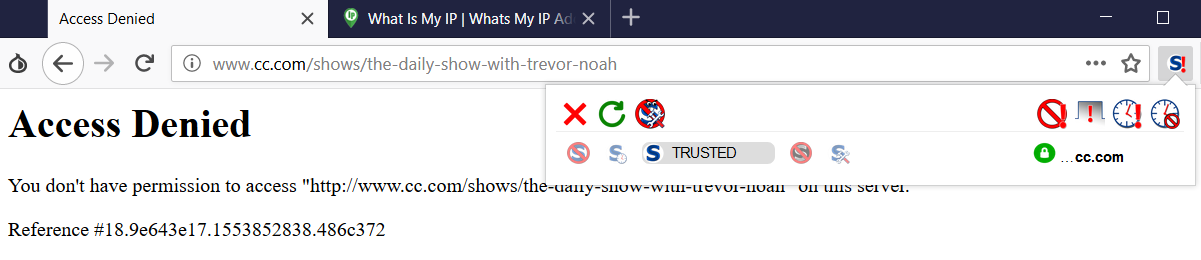
*Here we see that I have rerouted my connection.*

Above, we can see that I har rerouted my connection for this site. This has not provided a new identity, but instead has simply reconnected me to the site via a different set of internal Tor nodes.

Next, I will test the geo location functions of Tor by attempting to access a website that is restricts access by geo location.



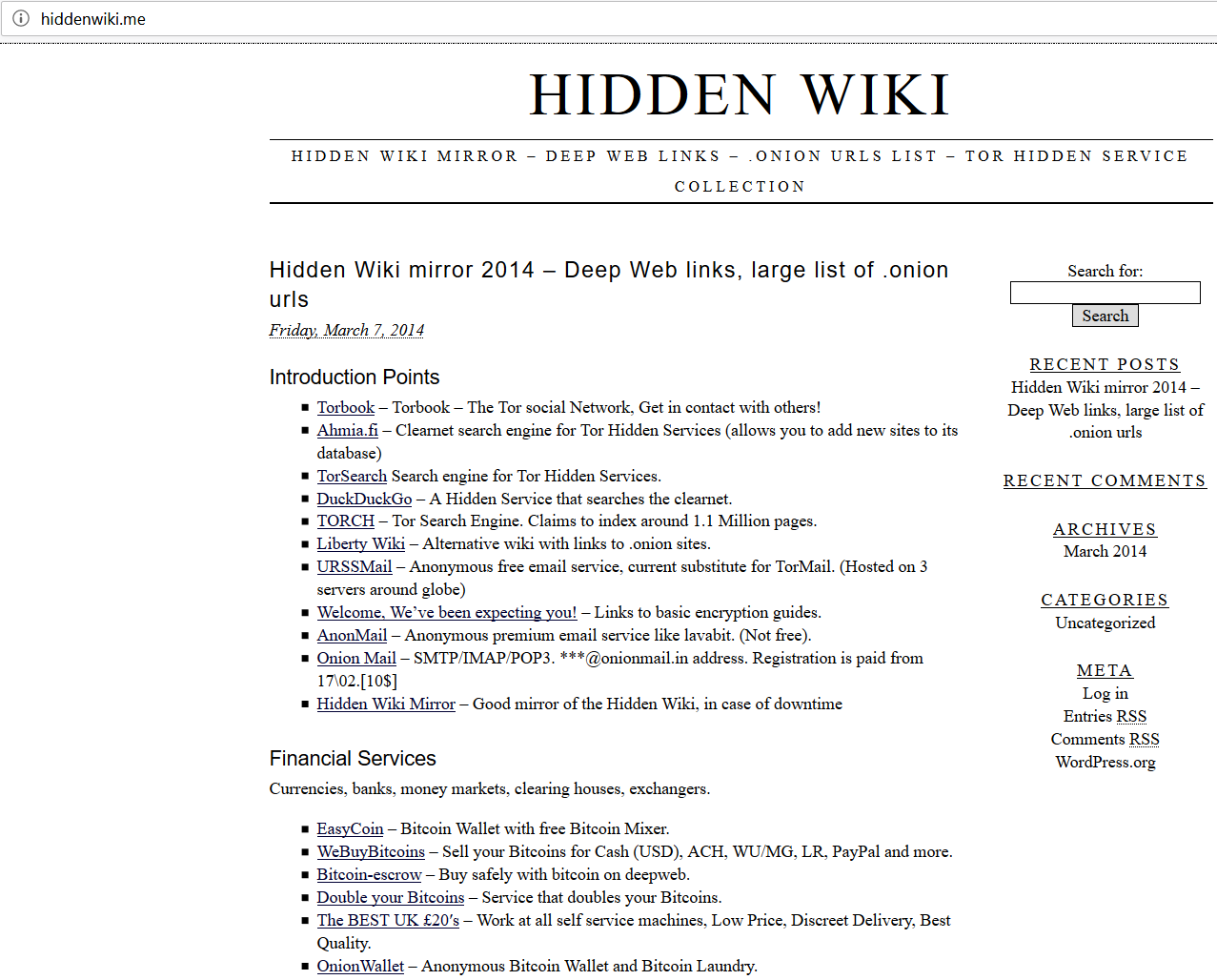
*Here, we can see that the Comedy Central website has prevented me from accessing it services as I am outside the US region.*



*My attempt to access the website via the Tor browser failed.*

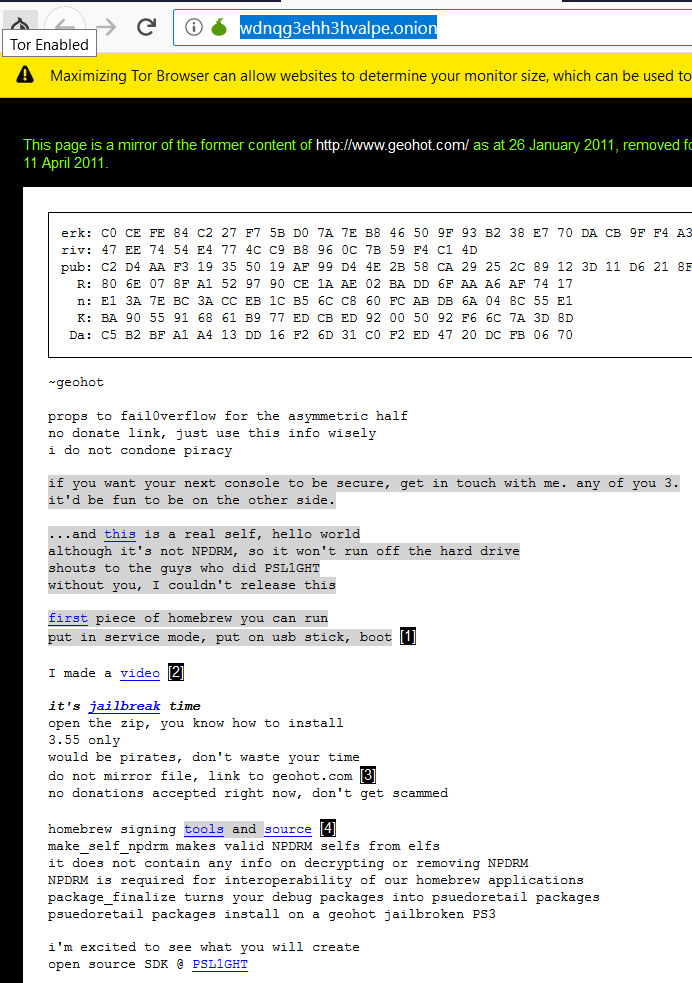
I was unable to access the website, as it seems the Tor network is not permitted to prevent exactly what I am doing. However, the geo location services are working as my IP was at this time set to Texas. I confirmed that was the case by connection to the service via a standard browser with my IP set to the US region via a VPN and it was successful. So, it seems that some site are preventing the Tor network from functioning even when JavaScript is allowed.

Finally, for this lab we take a quick peek into the Dark Web. I first search for a hidden wiki. These sites are known to document sites that make up the dark web. They come and go quite frequently.



*Here we see an example of a hidden wiki page.*

Next, we take a quick peek at an example of a custom dark web web-page. I will not explore too many of these as they can be inherently risky. But here is an example of a site that perhaps should not exist. It is a page dedicated to Jail-breaking the Sony PlayStation 3, which is clearly a violation of the terms of service relating to this product ( my visiting of this page should not be seen as an endorsement of the site or activities there in, which I neither support nor condone). I have purposefully cut out some fo the pages content so as not to accidentally provide any information that would actually facilitate jail-breaking a device.



*Here is an example of a Dark-Web website. This one is dedicated to Jail-breaking The PlayStation 3, which is a violation of the terms of service*

# 10.0 Tor Website

For this task we are challenged with the creation of our own Tor based onion website. I used the guide located here**(1)**. To achieve this, I elected to create the site on my Parrot OS VM. I must first ensure I am running the latest version of nginx and Tor and ensure that the tor service is stopped with the following commands:

* apt-get install nginx (this was up to date)
* apt-get install Tor (this was up to date)
* service tor stop

Next, we created a hosting area for our onion website. To begin I created a directory to hold my website. The following command was used: mkdir -p /usr/share/nginx/onions/myonion. This will create a default onion-based website within the myonion folder. I then created and populated a page, index.html, with the following command: echo “This is the test onion page for B00094138” > /usr/share/nginx/onions/myonion/index.html. I must then take ownership of this folder with the following command: chown -R debian-tor:debian-tor /usr/share/nginx/onions/myonion.

A port must be assigned for nginx to listen on. This is done with the follow commands:

* nano /etc/nginx/sites-available/default
* #Remove the existing config and replace with  
  server{

listen 127.0.0.1:9070;

root /usr/share/nginx/onions/myonion;

index index.html index.html;

server\_name foo.bar; #This will be amended later

location / {

autoindex on;

}

}

#Save and exit

Nano /etc/nginx/nginx.conf

#Locate the http block and add

Server\_tokens off;

#Save and exit

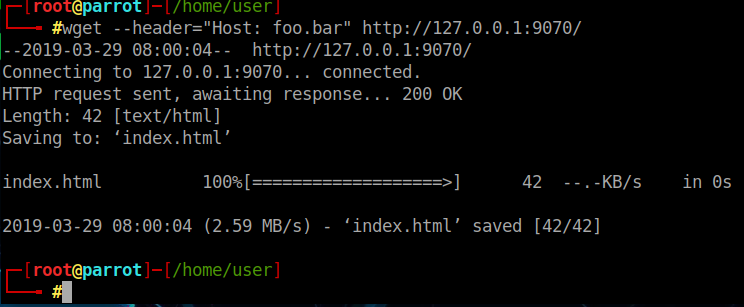
The nginx conf file must also be altered with the following code:

* Nano /etc/nginx/nginx.conf
* Server\_tokens off;

Now we shall test our site by starting the nginx service and attempting to host with the commands:

* Service nginx start
* wget –header=”host: foo.bar” http://127.0.0.1:9070/

Below we can see the confirmation that our site exists. But we have not yet completed out setup.



*Here we see the confirmation that page has successfully been created.*

## 10.1 Configuring Tor

Configuring the tor client allows us to hide our service. This is done with the following commnds:

* cd /etc/tor
* mv torrc torrc-default
* nano torrc
* #Insert the following lines

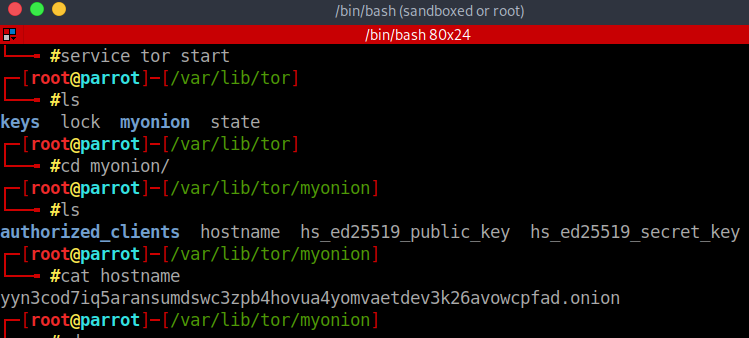
SocksPort 0 #what port to open for local application connections

SocksListenAddress 127.0.0.1 #accept connections only from localhost

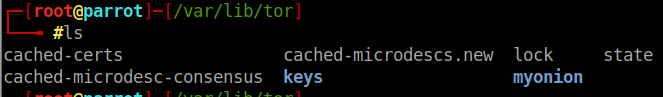
* RunAsDaemon 1
* DataDirectory /var/lib/tor
* HiddenServiceDir /var/lib/tor/myonion/
* HiddenServicePort 80 127.0.0.1:9070

Next, we start Tor with the command: service tor start, and this should create two new files containing our hostname and our private key, which we will view with the follow commands:

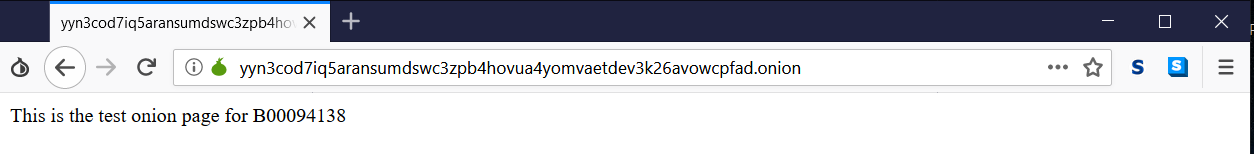
* cat /var/lib/tor/myonion/hostname
* cat /var/lib/tor/myonion/private\_key



*Here we can see our public and private keys as well as our onion hostname have been generated.*



*Here, we can see the storage location of any certs and keys exchange via our .onion.*



*Here we can see my .onion web page being accessed from my host Windows machine via the tor browser. But I will add a few extra layers of security.*

Finally, we alter our service to initiate at boot with the following commands:

* update-rc.d nginx defaults
* update-rc.d tor defaults

As an OP-SEC precaution we will also set up fail2ban to restrict access to our sites bash history, with the following commands:

* apt-get install fail2ban
* cat << EOM > /etc/fail2ban/jail.local

#make the following alterations

[ssh]

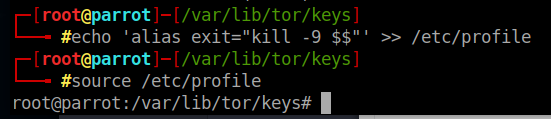
* enabled = true
* maxretry =3
* banaction = iptables-allports

[ssh-ddos]

* enabled = true
* maxretry = 3
* EOM
* service fail2ban restart
* update-rc.d fail2ban defaults

And now we disable access to our bash history with the commands:

* echo ‘alias exit=”kill -9 $$”’ >> /etc/profile
* source /etc/profile



*Here we can ses that I have removed access to the bash history for this profile which is serving up the /onion web server.*

# Conclusions

This year in cryptography was very worthwhile. At times difficult yet rewarding. The classic decoding and ciphers will no doubt prove useful going forward in CTF tournaments, but more importantly it paved th way for understand AES, DES and RSA basic encryption methods by giving an understand of how ciphers work.  
  
The numbers station was one of my favourite assignments thus far, being a world war 2history buff, I found this exciting and most enlightening. While I do not imagine that the methods used will be used very much going forward in practical use, it will once again pay off in CTFS and the ability to understand substitution ciphers using boards or graphs will be useful as this is still a method used in some encryption methods.  
  
Salted hashes did not teach us very much new, although it did give us some hands on experience with hash cracking and how long it may take in a real world scenario.  
  
Digital signatures are a new fascinating area for us. I did not envision that these could have such an impact on security in the real world. Nor did I anticipate that cryptography a world built in security could have such a trust-based element based within it resulting in what could potentially be a fatal flaw.  
  
Learning to apply cracked passwords to a wireless capture file to decrypt a file answered a question I had been asking myself for a long time. This is found very rewarding and I suspect this skill will pay huge dividends when I move on to studying SIEMs, attack vectors and honey pots. This is a skill I will take forward with me for years to come and hope to build up in terms of my ability to drill down into packets. I certainly feel that I must work harder to increase my speed and efficiency in terms of my ability to search through packets.  
  
The Tor Labs initially didn’t hold much interest for me. While I appreciate the value of the Tor network and the anonymity it offers, but I found the process of creating my onion quite fun. This is another skill that will prove quite useful for securing services for companies who may wish to avoid the public internet sector in an effort to greater secure their services but removing them from the indexed zone of the internet.

Secure programming, much as Penetration Testing did last term, proved to be a thoroughly rewarding and enjoyable module and I would like to extend my thanks to my Lecturer Mark Cummins for his excellent tutelage this term. Once again, I have added a significant array of tools to my skillset thanks to his dedication.

# Glossary

SSl: Secure Socket Layer, the most common widely implemented transport layer security protocol. Replaced by TLS.

TLS: Transport Layer Security, successor of SSL.

SIEMs: Security Information and Event Management System.

# References:

1. Tasker B., Hostinf TOR hidden services (.onions), [Online]. Available at: https://www.bentasker.co.uk/documentation/security/290-hosting-tor-hidden-ervices-onions. [Accessed March 2019].