Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

LAB PLAN

Total number of experiments 10

Total number of turns required 10

Number of turns required for

Experiment Number	Turns	Scheduled Day
Experiment 1	1	Day 1
Experiment 2	1	Day 2
Experiment 3	1	Day 3
Experiment 4	1	Day 4
Experiment 5	1	Day 5
Experiment 6	1	Day 6
Experiment 7	1	Day 7
Experiment 8	1	Day 8
Experiment 9	1	Day 9
Experiment 10	1	Day 10
Experiment 11	1	Day 11
Experiment 12	1	Day 12
Experiment 13	1	Day 13
Experiment 14	1	Day 14
Experiment 15	1	Day 15
Experiment 16	1	Day 16

Distribution of Lab Hours:

Attendance	05 minutes
Explanation of features of language	15 minutes
Explanation of experiment	15 minutes
Performance of experiment	70 minutes
Viva / Quiz / Queries	15 minutes
Total	**120 Minutes (2 Hrs.)**

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Lab Objective and Outcome

Objective

The objective of this course is to impart necessary and practical knowledge of components of cyber security lab and develop skills required to build real-life based projects.

- Understand and implement the cipher techniques.
- Demonstrate the knowledge of cyber security and implement different cyber security algorithms.
- Understand the different types of attacks on the systems.
- Analyze different packet sniffing tools and identify TCP/UDP datagram.
- Installation and understanding of rootkits and snot tools and their various options.
- Demonstrate how to secure the data and create digital signature.

Course Outcomes

After completion of this course, students will be able to –

7CS4-22.1	**Apply** the cryptographic algorithms for data communication.
7CS4-22.2	**Understand** implementation of various security tools and algorithms.
7CS4-22.3	Apply the Digital signature for secure data transmission and secure data storage.
7CS4-22.4	**Utilize** the different open-source tools for network security and analysis.
7CS4-22.5	**Demonstrate** intrusion detection system using network security tool.

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Experiments

Experiment 1: Implementation of Caesar Cipher

The Caesar Cipher, named after Julius Caesar of Ancient Rome, is a type of substitution cipher where each letter of the original (plaintext) message is substituted with another letter.

Encrypting: But how do we decide what letter is replaced by what? That's where the **key** comes into play. Found in almost every encryption algorithm, the key determines **how the data is encrypted**.

In the Caesar cipher, the key is a number from 0 to 25, because there are 26 letters in the alphabet. This means that for any given message, there are 26 different ways we can encrypt the message.

For each letter, the key determines **which letter is replacing the current letter, by counting down the alphabet**. In the following example, let's say we wanted to encrypt the letter B with a key of 3, we would find the 3rd letter that appears after B - which is C, D, then finally E.

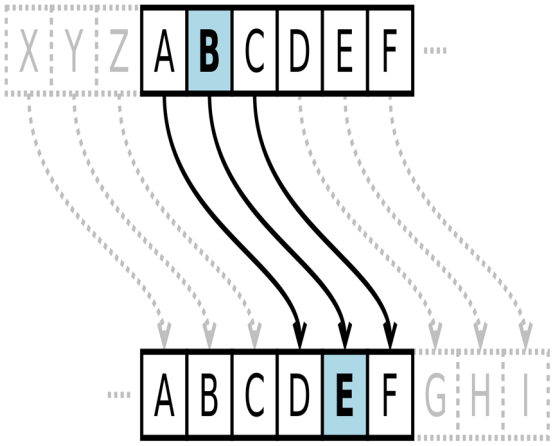


Fig. Encryption in Caesar Cipher Algorithm

Decrypting: Decrypting works in a very similar way - except this time, instead of counting "down" the alphabet, you count "up"! As an example, let's try to decrypt ELOOB using a key of 3 - because we know the result should be our original plaintext, BILLY. Let's start by aligning our alphabets:

Caesar Cipher algorithm Implementation in python:

```
class CaesarCipher():
    def __init__(self,key,message):
        self.key = key
        self.message = message
        self.cipher = ""
        self.PlainText = ""
```

```
    def encrypt(self):
        alpha ="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        beta = "abcdefghijklmnopqrstuvwxyz"

        for char in self.message:
            if char in alpha:
                charindex = (alpha.index(char)+ self.key) % len(alpha)
                self.cipher = self.cipher + alpha[charindex]
            elif char in beta:
                charindex = (beta.index(char)+ self.key)% len(beta)
                self.cipher = self.cipher + beta[charindex]
            else:
                self.cipher = self.cipher + char
        return self.cipher

    def decrypt(self):
        alpha ="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        beta = "abcdefghijklmnopqrstuvwxyz"
        for char in self.cipher:
            if char in alpha:
                charindex = (alpha.index(char) - self.key)% len(alpha)
                self.PlainText = self.PlainText + alpha[charindex]
            elif char in beta:
                charindex = (beta.index(char) - self.key)% len(beta)
                self.PlainText = self.PlainText + beta[charindex]
            else:
                self.PlainText = self.PlainText + char
        return self.PlainText

def main():
    message = input("Enter the Plain Text: ")
    key = int(input("Enter the key: "))
    caesar = CaesarCipher(key, message)
    CipherText = caesar.encrypt()
    print(" Cipher text of given message: ", CipherText)
    PlainText = caesar.decrypt()
    print(" Origonal message: ", PlainText)

if __name__ == "__main__":
    main()
```

Swami Keshvanand Institute of Technology, Management &Gramothan,
Ramnagaria, Jagatpura, Jaipur-302017, INDIA
Approved by AICTE, Ministry of HRD, Government of India
Recognized by UGC under Section 2(f) of the UGC Act, 1956
Tel. : +91-0141- 5160400Fax: +91-0141-2759555
E-mail: info@skit.ac.in Web: www.skit.ac.in

Experiment 2: Implementation of Rail Fence Cipher

The Rail Fence cipher works by writing your message on alternate lines across the page, and then reading off each line in turn. For example, the plaintext "defend the east wall" is written as shown below, with all spaces removed.

D		F		N		T		E		A		T		A		L
	E		E		D		H		E		S		W		L	

The ciphertext is then read off by writing the top row first, followed by the bottom row, to get "DFNTEATALEEDHESWL".

Encryption

To encrypt a message using the Rail Fence Cipher, you must write your message in zigzag lines across the page, and then read off each row. Firstly, you need to have a key, which for this cipher is the number of rows you are going to have. You then start writing the letters of the plaintext diagonally down to the right until you reach the number of rows specified by the key. You then bounce back up diagonally until you hit the first row again. This continues until the end of the plaintext. For the plaintext we used above, "defend the east wall", with a key of 3, we get the encryption process shown below.

D				N				E				T				L		
	E		E		D		H		E		S		W		L		X	
		F				T				A				A				X

Note that at the end of the message we have inserted two "X"s. These are called nulls, and act as placeholders. We do this to make the message fit neatly into the grid (so that there are the same number of letters on the top row, as on the bottom row. Although not necessary, it makes the decryption process a lot easier if the message has this layout. The ciphertext is read off row by row to get "DNETLEEDHESWLXFTAAX".

Decryption

The decryption process for the Rail Fence Cipher involves reconstructing the diagonal grid used to encrypt the message. We start writing the message but leaving a dash in place of the spaces yet to be occupied. Gradually, you can replace all the dashes with the corresponding letters and read off the plaintext from the table.

We start by making a grid with as many rows as the key is, and as many columns as the length of the ciphertext. We then place the first letter in the top left square, and dashes diagonally downwards where the letters will be. When we get back to the top row, we place the next letter in the ciphertext. Continue like this across the row and start the next row when you reach the end.

For example, if you receive the ciphertext "TEKOOHRACIRMNREATANFTETYTGHH", encrypted with a key of 4, you start by placing the "T" in the first square. You then dash the diagonal down spaces until you get back to the top row and place the "E" here. Continuing to fill the top row you get the pattern below. The first row of the decryption process for the Rail Fence Cipher. We have a table with 4 rows because the key is 4, and 28 columns as the ciphertext has length 28.

T						E						K						O						O			
	-				-		-				-		-				-		-				-		-			
		-		-				-		-				-		-				-		-				-	
			-						-						-						-						-

The second stage in the decryption process.

T						E						K						O						O			
	H				R		A				C		I				R		M				N		R			
		-		-				-		-				-		-				-		-				-	
			-						-						-						-						-

The third stage in the decryption process.

T						E						K						O						O			
	H				R		A				C		I				R		M				N		R			
		E		A				T		A				N		F				T		E				T	
			-						-						-						-						-

The fourth and final stage in the decryption process.

From this we can now read the plaintext off following the diagonals to get "they are attacking from the north".

Rail Fence cipher Implementation in python:

```
class RailFenceCipher():
    def __init__(self, message, key):
        self.message = message
        self.key = key
        self.CipherText = ""
        self.PlainText = ""
        self.Matrix = []

    def getmatrix(self):
        self.Matrix =[[0 for col in range(len(self.message))] for row in range(self.key)]
        return self.Matrix

    def encrypt(self):
        RailMatrix = self.getmatrix()
        # Putting the message in RailMatrix diagonally
        row = 0
        dir_down = 0
        for col in range(len(self.message)):
            RailMatrix[row][col] = self.message[col]
            if row == 0 or (row < self.key-1 and dir_down == 0):
                dir_down = 0
                row += 1
            else:
                row -= 1
                dir_down = 1

    # Reading the row to get cipher Text
        for sublist in RailMatrix:
            for value in sublist:
                if value == 0:
                    self.CipherText = self.CipherText
                else:
                    self.CipherText = self.CipherText + value
```

```
        return self.CipherText
# Decryption using Rail Fence Cipher
    def decrypt(self):
        CipherMatrix = self.getmatrix()
        #mark the diagonal of ciphermatrix
        row = 0
        dir_down = 0
        for col in range(len(self.message)):
            CipherMatrix[row][col] = 1
            if row == 0 or (row < self.key-1 and dir_down == 0):
                dir_down = 0
                row += 1
            else:
                row -= 1
                dir_down = 1
     #putting the cipher text into rows of cipher Matrix
        col = 0
        for sublist in CipherMatrix:
            i = 0
            for value in sublist:
                if value == 1:
                    sublist[i] = self.CipherText[col]
                    col += 1
                    i += 1
                else:
                    i += 1
    #printing the Origonal message after decryption
        result = []
        row = 0
        for col in range(len(self.message)):
            result.append(CipherMatrix[row][col])
            if row == 0 or (row < self.key-1 and dir_down == 0):
                dir_down = 0
                row += 1
            else:
                row -= 1
                dir_down = 1
        self.PlainText ="".join(result)
        return self.PlainText

    def __str__(self):
        return "Input Message: {} \nInput Key: {} \nCipher Text: {} \nPlain Text:
{}".format(self.message,self.key,self.CipherText, self.PlainText)
```

```
def main():
    message = input("Enter the Plain Text: ")
    key = int(input("Enter the key: "))
    RailFence = RailFenceCipher(message, key)
    RailFence.encrypt()
    RailFence.decrypt()
   print("****************************************")
    print(RailFence)
    print("****************************************")
if __name__ == "__main__":
    main()
```

Experiment 3: Columnar Transposition Cipher

Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns. In its simplest form, it is the Route Cipher where the route is to read down each column in order. For example, the plaintext "a simple transposition" with 5 columns looks like the grid below

A	S	I	M	P
L	E	T	R	A
N	S	P	O	S
I	T	I	O	N

Plaintext written across 5 columns. If we now read down each column, we get the ciphertext "ALNISESTITPIMROOPASN".

Encryption

We first pick a keyword for our encryption. We write the plaintext out in a grid where the number of columns is the number of letters in the keyword. We then title each column with the respective letter from the keyword. We take the letters in the keyword in alphabetical order and read down the columns in this order. If a letter is repeated, we do the one that appears first, then the next and so on. As an example, let's encrypt the message "The tomato is a plant in the nightshade family" using the keyword tomato. We get the grid given below.

T	O	M	A	T	O
5	**3**	**2**	**1**	**6**	**4**
T	H	E	T	O	M
A	T	O	I	S	A
P	L	A	N	T	I
N	T	H	E	N	I
G	H	T	S	H	A
D	E	F	A	M	I
L	Y	X	X	X	X

The plaintext is written in a grid beneath the keyword. The numbers represent the alphabetical order of the keyword, and so the order in which the columns will be read. We have written the keyword above the grid of the plaintext, and the numbers telling us which order to read the columns in. Notice that the first "O" is 3 and the second "O" is 4, and the same thing for the two "T"s.

Decryption

The decryption process is significantly easier if nulls have been used to pad out the message in the encryption process. Below we shall talk about how to go about decrypting a message in both scenarios.

Firstly, if nulls have been used, then you start by writing out the keyword and the alphabetical order of the letters of the keyword. You must then divide the length of the ciphertext by the length of the keyword. The answer to this is the number of rows you need to add to the grid. You then write the ciphertext down the first column until you reach the last row. The next letter becomes the first letter in the second column (by the alphabetical order of the keyword), and so on.

As an example, we shall decrypt the ciphertext "ARESA SXOST HEYLO IIAIE XPENG DLLTA HTFAX TENHM WX" given the keyword potato. We start by writing out the keyword and the order of the letters. There are 42 letters in the ciphertext, and the keyword has six letters, so we need 42 ÷ 6 = 7 rows.

P	O	T	A	T	O
4	2	5	1	6	3

We have the keyword and the order of the letters in the keyword. We also know there are 7 rows.

Now we start by filling in the columns in the order given by the alphabetical order of the keyword, starting with the column headed by "A". After the first column is entered, we have the grid shown to the right.

P	O	T	A	T	O
4	2	5	1	6	3
			A		
			R		
			E		
			S		
			A		
			S		
			X		

We continue to add columns in the order specified by the keyword.

P	O	T	A	T	O
4	2	5	1	6	3
	O		A		
	S		R		
	T		E		
	H		S		
	E		A		
	Y		S		
	L		X		

After inserting the second column.

P	O	T	A	T	O
4	2	5	1	6	3
	O		A		O
	S		R		I
	T		E		I
	H		S		A
	E		A		I
	Y		S		E
	L		X		X

After inserting the third column.

P	O	T	A	T	O
4	2	5	1	6	3
P	O	T	A	T	O
E	S	A	R	E	I
N	T	H	E	N	I
G	H	T	S	H	A
D	E	F	A	M	I
L	Y	A	S	W	E
L	L	X	X	X	X

The completely reconstructed grid.

Now we read off the plaintext row at a time to get "potatoes are in the nightshade family as well".

Column Transposition cipher Implementation in python:

```
import math
class columnTransposition():
    def __init__(self, message, key):
        self.message = message
        self.key = key
        self.CipherText = ""
        self.PlainText = ""

    def __str__(self):
        return "Input Message: {} \nInput Key: {} \nCipher Text: {} \nPlain Text:
{}".format(self.message,self.key,self.CipherText, self.PlainText)

    def encrypt(self):
        col = len(self.key)
        # fill the row of matrix using message
        row = math.ceil(len(self.message)/len(self.key))
        PlainMatrix = [[" " for col in range(len(self.key))] for i in range(row)]
        message_index = 0
        for sublist in PlainMatrix:
            for value in range(len(self.key)):
                sublist[value] = self.message[message_index]
                if message_index < len(self.message)-1:
                    message_index += 1
                else:
                    break
        # Reading the columns to generate cipher text
        sorted_keys = sorted(self.key) # to aarange in order
        for i in range(col):
            col_read_index = self.key.index(sorted_keys[i])
            for row in PlainMatrix:
                self.CipherText = self.CipherText + row[col_read_index]
```

```
        return self.CipherText


    def decrypt(self):
        # Create the cipher text matrix and putting the cipher text
        row = math.ceil(len(self.CipherText)/len(self.key))
        CipherMatrix = [[" " for col in range(len(self.key))] for col in range(row)]
        cipher_index = 0
        sorted_keys = sorted(self.key)
        for i in range(len(self.key)):
            for row in CipherMatrix:
                col_select = self.key.index(sorted_keys[i])
                row[col_select] = self.CipherText[cipher_index]
                cipher_index += 1
        # Reading the cipher Matrix to generate origonal message
        for row in CipherMatrix:
            for char in row:
                self.PlainText = self.PlainText + char
        return self.PlainText


def main():
    message = input("Enter the Plain Text: ")
    key = input("Enter the key: ")
    ColTransposition = columnTransposition(message, key)
    ColTransposition.encrypt()
        #ColTransposition.decrypt()
    ColTransposition.decrypt()
        print( ColTransposition)
  if __name__ == "__main__":
    main()
```

Experiment 4: Password Cracking Techniques- Brute force & Dictionary attack

A **password**, also known as a PIN, passcode, or secret code, in its simplest form, it is just secret word or phrase used for authentication, to determine identity of a person. Nowadays when you hear the word password, you automatically assume they are talking about a website or something related to computers and other electronic devices, but computers haven't always been around, passwords have.

Passwords are a big part of our daily life. We have passwords to protect our email, voicemail, phones, ATM cards, lockers, online banking, wireless networks,encrypted data etc.

As you can see, we need to keep them secure because if they fall into the wronghands bad things will happen. Let's look at some possible outcomes that could happen if such a thing happened.

- Your bank account could be emptied to fund a large purchase
- Your sensitive email could be read by someone.
- You could be visited by the police station after your wireless network was used to illegal activity.

It is extremely important to know how to secure and create strong passwords.

Password cracking is the act of recovering passwords through unconventional and usually unethical methods from data that has been stored or sent through a computer system.

Password cracking is a very popular computer attack because once a high-level user password is cracked, you've got the power! There's no longer a need to search for vulnerabilities and all that other mumbo jump needed to take over asystem that we won't be discussing in this section.

Also, everyone is susceptible to a password cracking attack. Unless you live in a remote, technology absent area, you have a password for something, and there'susually something to gain from obtaining your password.

Password cracking can be used for both good and evil. If I forgot my password fora certain system

or program, I might try cracking it before I completely give up on it. Now if it's for any other reason, then it probably has an evil basis and is most likely illegal as well.

To show you how real and popular this form of attack is today, here are a fewrecent happenings.

- Password cracking was used to take over a few high-profile twitter accounts.
- Wal-Mart was a victim of a security breach where sensitive information was taken. Password cracking was one of the many methods used to gain entry.
- 10,000 cracked Hotmail passwords were publicly posted, and every day crackers continue to post new lists on forums all over the internet.
- phpBB.com was hacked and their 200,000+ username/password database was dumped and made publicly available to anyone willing to download it. Of those passwords, over 80,000 were reported to have had been cracked.

There are many different types of password cracking methods

1. **Brute Force Attacks:**

A brute force attack uses trial-and-error to guess login info, encryption keys, or find a hidden web page. Hackers work through all possible combinations hoping to guess correctly.

These attacks are done by 'brute force' meaning they use excessive forceful attempts to try and 'force' their way into your private account(s).

This is an old attack method, but it's still effective and popular with hackers. Because depending on the length and complexity of the password, cracking it can take anywhere from a few seconds to many years.

Here's how hackers benefit from brute force attacks:

- Profiting from ads or collecting activity data
- Stealing personal data and valuables
- Spreading malware to cause disruptions
- Hijacking your system for malicious activity
- Ruining a website's reputation

Tools Aid Brute Force Attempts

Guessing a password for a particular user or site can take a long time, so hackers have developed tools to do the job faster.

Automated tools help with brute force attacks. These use rapid-fire guessing that is built to create every possible password and attempt to use them. Brute force hacking software can find a single dictionary word password within one second.

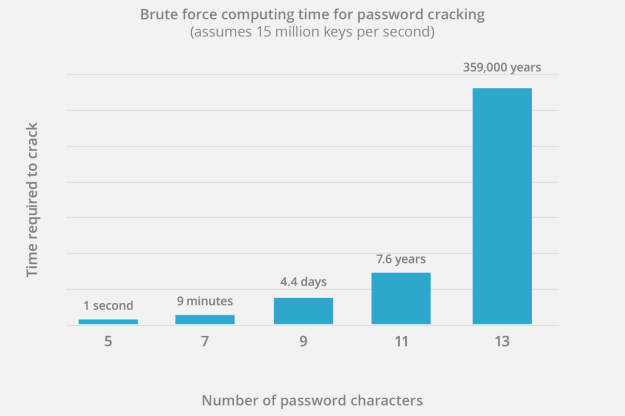
Tools like these have workarounds programmed in them to:

- Work against many computer protocols (like FTP, MySQL, SMPT, and Telnet)
- Allow hackers to crack wireless modems.
- Identify weak passwords
- Decrypt passwords in encrypted storage.
- Translate words into leetspeak — "don'thackme" becomes "d0n7H4cKm3," for example.
- Run all possible combinations of characters.
- Operate dictionary attacks.

How to Defend Against Brute Force Attacks

Brute force attacks need time to run. Some attacks can take weeks or even months to provide anything usable. Most of the defenses against brute force attacks involve increasing the time required for success beyond what is technically possible, but that is not the only defense.

- **Increase password length**: More characters equal more time to brute force crack



- **Increase password complexity**: More options for each character also increase the time to brute force crack
- **Limit login attempts**: Brute force attacks increment a counter of failed login attempts on most directory services – a good defense against brute force attacks is to lock out users after a few failed attempts, thus nullifying a brute force attack in progress

- **Implement Captcha**: Captcha is a common system to verify a human is a human on websites and can stop brute force attacks in progress
- **Use multi-factor authentication**: Multi-factor authentication adds a second layer of security to each login attempt that requires human intervention which can stop a brute force attack from success

Introduction to the 10 Most Popular Password Cracking Tools

1. **Brutus:**

   Brutus is one of the most popular remote online password cracking tools. It claims to be the fastest and most flexible password cracking tool. This tool is free and is only available for Windows systems. It was released back in October 2000. It supports HTTP (Basic Authentication), HTTP (HTML Form/CGI), POP3, FTP, SMB, Telnet and other types such as IMAP, NNTP, NetBus, etc. You can also create your own authentication types. This tool also supports multi-stage authentication engines and is able to connect 60 simultaneous targets. It also has resume and load options. So, you can pause the attack process any time and then resume whenever you want to resume. This tool has not been updated for many years. Still, it can be useful for you.

2. **RainbowCrack:** RainbowCrack is a hash cracker tool that uses a large-scale time-memory trade off process for faster password cracking than traditional brute force tools. Time-memory trade off is a computational process in which all plain text and hash pairs are calculated by using a selected hash algorithm. After computation, results are stored in the rainbow table. This process is very time consuming. But, once the table is ready, it can crack a password must faster than brute force tools. You also do not need to generate rainbow tablets by yourselves. Developers of RainbowCrack have also generated LM rainbow tables, NTLM rainbow tables, MD5 rainbow tables and Sha1 rainbow tables. Like RainbowCrack, these tables are also available for free. You can download these tables and use for your password cracking processes.

   Download Rainbow tables here: http://project-rainbowcrack.com/table.htm

   A few paid rainbow tables are also available, which you can buy from here: http://project-

rainbowcrack.com/buy.php

This tool is available for both Windows and Linux systems. Download Rainbow crack here: http://project-rainbowcrack.com/

3. **Wfuzz:** Wfuzz is another web application password cracking tool that tries to crack passwords with brute forcing. It can also be used to find hidden resources like directories, servlets and scripts. This tool can also identify different kind of injections including SQL Injection, XSS Injection, LDAP Injection, etc in Web applications.

   Key features of Wfuzz password cracking tool:

   - Capability of injection via multiple points with multiple dictionary
   - Output in colored HTML
   - Post, headers, and authentication data brute forcing
   - Proxy and SOCK Support, Multiple Proxy Support
   - Multi-Threading
   - Brute force HTTP Password
   - POST and GET Brute forcing
   - Time delay between requests
   - Cookies fuzzing

4. **Cain and Abel**: Cain and Abel is a well-known password cracking tool that is capable of handling a variety of tasks. The most notable thing is that the tool is only available for Windows platforms. It can work as sniffer in the network, cracking encrypted passwords using the dictionary attack, recording VoIP conversations, brute force attacks, cryptanalysis attacks, revealing password boxes, uncovering cached passwords, decoding scrambled passwords, and analyzing routing protocols.

   Cain and Abel does not exploit any vulnerability or bugs. It only covers security weakness of protocols to grab the password. This tool was developed for network administrators, security professionals, forensics staff, and penetration testers. Download here: http://www.oxid.it/ca_um/

5. **John the Ripper:** John the Ripper is another well-known free open source password cracking tool for Linux, Unix and Mac OS X. A Windows version is also available. This tool can detect weak passwords. A pro version of the tool is also available, which offers better features and

native packages for target operating systems. You can also download Openwall GNU/*/Linux that comes with John the Ripper. Download John the Ripper here: http://www.openwall.com/john/

6. **THC Hydra:** THC Hydra is a fast network logon password cracking tool. When it is compared with other similar tools, it shows why it is faster. New modules are easy to install in the tool. You can easily add modules and enhance the features. It is available for Windows, Linux, Free BSD, Solaris and OS X. This tool supports various network protocols. Currently it supports Asterisk, AFP, Cisco AAA, Cisco auth, Cisco enable, CVS, Firebird, FTP, HTTP-FORM-GET, HTTP-FORM-POST, HTTP-GET, HTTP-HEAD, HTTP-PROXY, HTTPS-FORM-GET, HTTPS-FORM-POST, HTTPS-GET, HTTPS-HEAD, HTTP-Proxy, ICQ, IMAP, IRC, LDAP, MS-SQL, MYSQL, NCP, NNTP, Oracle Listener, Oracle SID, Oracle, PC-Anywhere, PCNFS, POP3, POSTGRES, RDP, Rexec, Rlogin, Rsh, SAP/R3, SIP, SMB, SMTP, SMTP Enum, SNMP, SOCKS5, SSH (v1 and v2), Subversion, Teamspeak (TS2), Telnet, VMware-Auth, VNC and XMPP. Download THC Hydra here: https://www.thc.org/thc-hydra/ If you are a developer, you can also contribute to the tool's development.

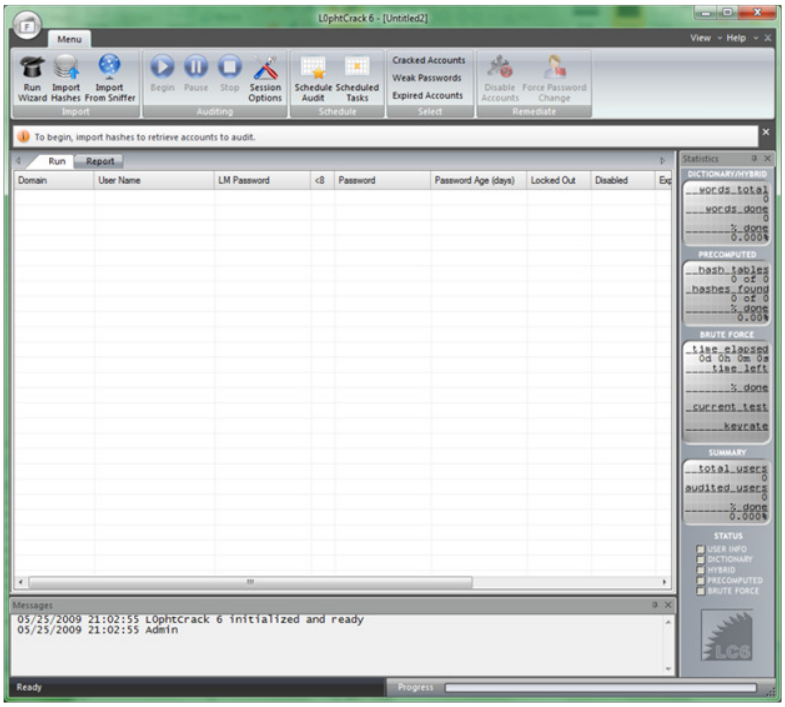
7. **Medusa:** Medusa is also a password cracking tool similar to THC Hydra. It claims to be a speedy parallel, modular and login brute forcing tool. It supports HTTP, FTP, CVS, AFP, IMAP, MS SQL, MYSQL, NCP, NNTP, POP3, PostgreSQL, pcAnywhere, rlogin, SMB, rsh, SMTP, SNMP, SSH, SVN, VNC, VmAuthd and Telnet. While cracking the password, host, username and password can be flexible input while performing the attack.

   Medusa is a command line tool, so you need to learn commands before using the tool. Efficiency of the tool depends on network connectivity. On a local system, it can test 2000 passwords per minute. With this tool, you can also perform a parallel attack. Suppose you want to crack passwords of a few email accounts simultaneously. You can specify the username list along with the password list. Read more about this here: http://foofus.net/goons/jmk/medusa/medusa.html Download Medusa here: http://www.foofus.net/jmk/tools/medusa-2.1.1.tar.gz

8. **OphCrack:** OphCrack is a free rainbow-table based password cracking tool for Windows. It is

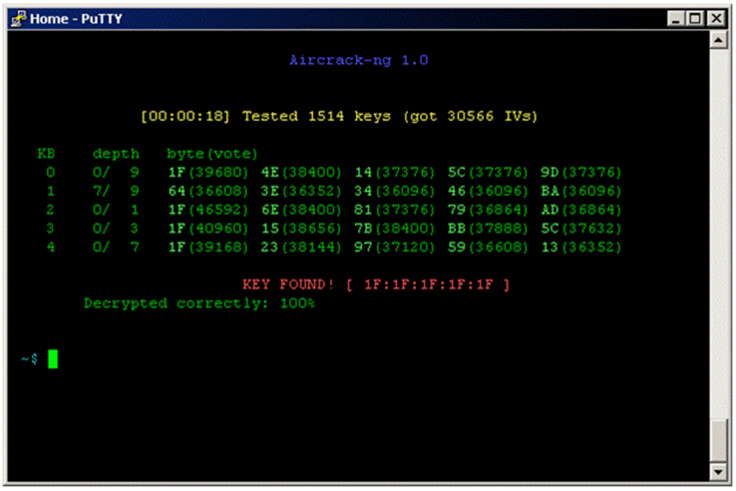
the most popular Windows password cracking tool, but can also be used on Linux and Mac systems. It cracks LM and NTLM hashes. For cracking Windows XP, Vista and Windows 7, free rainbow-tables are also available. A live CD of OphCrack is also available to simplify the cracking. One can use the Live CD of OphCrack to crack Windows-based passwords. This tool is available for free. Download OphCrack here: http://ophcrack.sourceforge.net/ Download free and premium rainbow tables for OphCrack here: http://ophcrack.sourceforge.net/tables.php

9. **L0phtCrack:** L0phtCrack is an alternative to OphCrack. It attempts to crack Windows password from hashes. For cracking passwords, it uses Windows workstations, network servers, primary domain controllers, and Active Directory. It also uses dictionary and brute force attacking for generating and guessing passwords. It was acquired by Symantec and discontinued in 2006. Later L0pht developers again re-acquired it and launched L0phtCrack in 2009. It also comes with a schedule routine audit feature. One can set daily, weekly or monthly audits, and it will start scanning on the scheduled time. L0phtCrack: http://www.l0phtcrack.com/



10 **Aircrack-NG:** Aircrack-NG is a WiFi password cracking tool that can crack WEP or WPA passwords. It analyzes wireless encrypted packets and then tries to crack passwords via its

cracking algorithm. It uses the FMS attack along with other useful attack techniques for cracking password. It is available for Linux and Windows systems. A live CD of Aircrack is also available.



If you want to use AirCrack NG for password cracking, read tutorials here: http://www.aircrack-ng.org/doku.php?id=getting_started

Download AirCrack-NG here: http://www.aircrack-ng.org/

2. Dictionary Attack

A Dictionary Attack as an attack vector used by the attacker to break in a system, which is password protected, by putting technically every word in a dictionary as a form of password for that system. This attack vector is a form of Brute Force Attack.

The dictionary can contain words from an English dictionary and also some leaked list of commonly used passwords and when combined with common character replacing with numbers, can sometimes be very effective and fast.

- **Difference between Brute Force and Dictionary Attack:** The difference with brute force attack is that, in brute force, many possible key permutations are checked whereas, in the dictionary attack, only the words with most possibilities of success are checked and are less time consuming than brute force.
- **What to avoid while selecting your password:** There are a few things which were very common a few years back and still exist. Most of the password cracking tools start from there. Passwords that fall into this category are most easy to crack. These are the few password

mistakes which you should avoid:

- Never use a dictionary word
- Avoid using your pet's name, parent name, your phone number, driver's license number or anything which is easy to guess.
- Avoid using passwords with sequence or repeated characters: For Ex: 1111111, 12345678 or qwerty, asdfgh.

- **The longer the password, the harder it is to crack**: Password length is the most important factor. If you select a small password, password cracking tools can easily crack it by using few words combinations. A longer password will take a longer time in guessing. You're your password at least 8 characters long.
- **Always use a combination of characters, numbers, and special characters**: This is another thing which makes passwords hard to crack. Password cracking tools try the combination of one by one. Have a combination of small characters, capital letters, and special characters. Suppose if you have only numbers in your password. Password cracking tools only need to guess numbers from 0-9. Here only length matters. But having a password combination of a-z, A-Z, 0-9 and other special characters with a good length will make it harder to crack. This kind of password sometimes takes weeks to crack.
- **Variety in passwords:** One important thing you must always take care. Never use same password everywhere. Cyber criminals can steal passwords from one website and then try it on other websites too.

- **The top 15 passwords on the 2017 list.**
  - 123456, Password, 12345678, qwerty, 12345, 123456789, letmein, 1234567, Football, iloveyou, admin, welcome, monkey, login, abc123
- **The top 11 worst passwords of 2012:**
  - Password, 123456, 12345678, abc123, qwerty, monkey, letmein, dragon, 111111, baseball, iloveyou

Experiment 5: Diffie Hellman key exchange algorithm

The question of key exchange was one of the first problems addressed by a cryptographic protocol. This was prior to the invention of public key cryptography. The Diffie-Hellman key agreement protocol (1976) was the first practical method for establishing a shared secret over an unsecured communication channel.

In Public key encryption schemes are secure only if authenticity of the public key is assured. Diffie-Hellman key exchange is a simple public key algorithm. The protocol enables 2 users to establish a secret key using a public key scheme based on **discrete algorithms**. The protocol is secure only if the authenticity of the 2 participants can be established. or this scheme, there are 2 publicly known numbers:

- A prime number q
- An integer *a* that is a primitive root of q.

(Note: Primitive root of a prime number P is one, whose powers module P generate all the images from 1 to P-1)

Algorithm steps:

1. The first step is for Alice and Bob to agree on a large prime p and a nonzero integer g modulo p. Alice and Bob make the values of p and g public knowledge.
2. The next step is for Alice to pick a secret integer a that she does not reveal to anyone, while at the same time Bob picks an integer b that he keeps secret.
3. Alice uses their secret integers to compute: $A \equiv g^a \pmod p$
4. Bob uses their secret integers to compute: $B \equiv g^b \pmod p$
5. They next exchange these computed values; Alice sends A to Bob and Bob sends B to Alice.
6. Finally, Bob and Alice again use their secret integers to compute.
    a. Alice uses their secret integers to compute: $A' \equiv B^a \pmod p$
    b. Bob uses their secret integers to compute: $B' \equiv A^b \pmod p$
7. The values that they compute are same i.e $A' = B'$.

Implementation of Diffie-Hellman algorithm

```
import math

# Shared Variables (public key) Used by Alice and Bob

sharedPrime = int(input("Enter the prime number (p): "))    # p

sharedBase = int(input("Enter the primitive root of prime number p (g): "))      # g


# Begin

print( "Publicly Shared Variables:")

print( "    Publicly Shared Prime: " , sharedPrime )

print( "    Publicly Shared Base:  " , sharedBase )


# Alice Sends Bob A = g^a mod p

aliceSecret = int(input("Alice's choose self private key (a): "))     # a

A = int(math.pow(sharedBase,aliceSecret) % sharedPrime)

print( "\n  Alice Sends Over Public Chanel: " , A )


# Bob Sends Alice B = g^b mod p

bobSecret = int(input("Bob's choose self private key (b): "))      # b

B = int(math.pow(sharedBase,bobSecret) % sharedPrime)

print( "Bob Sends Over Public Chanel: ", B )


print( "************************************************" )

print( "Privately Calculated Shared Secret:" )
```

```
# Alice Computes Shared Secret: s = B^a mod p

aliceSharedSecret = int(math.pow(B,aliceSecret) % sharedPrime)

print( "    Alice Shared Secret: ", aliceSharedSecret )


# Bob Computes Shared Secret: s = A^b mod p

bobSharedSecret = int(math.pow(A,bobSecret) % sharedPrime)

print( "    Bob Shared Secret: ", bobSharedSecret )


if aliceSharedSecret is bobSharedSecret:

   print("Alice and Bob having same key:")

else:

   print("Generated keys are different....Please check your code again:")
```

Experiment 6: Implementation of Playfair cipher

The Playfair cipher starts with creating a key table. The key table is a 5×5 grid of letters that will act as the key for encrypting your plaintext. Each of the 25 letters must be unique and one letter of the alphabet is omitted from the table (as there are 25 spots and 26 letters in the alphabet).

To encrypt a message, one would break the message into diagrams (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD" and map them out on the key table. The two letters of the diagram are considered as the opposite corners of a rectangle in the key table. Note the relative position of the corners of this rectangle. Then apply the following 4 rules, in order, to each pair of letters in the plaintext:

1. If both letters are the same (or only one letter is left), add an "X" after the first letter
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively
3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively
4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair.

D. Playfair Cipher

Example1: Plaintext: CRYPTO IS TOO EASY **Key =** INFOSEC **Ciphertext:** ??

Grouped text: CR YP TO IS TO XO EA SY

Ciphertext: AQ TV YB NI YB YF CB OZ

I / J	N	F	O	S
E	C	A	B	D
G	H	K	L	M
P	Q	R	T	U
V	W	X	Y	Z

ALGORITHM:

STEP-1: Read the plain text from the user.

STEP-2: Read the keyword from the user.

STEP-3: Arrange the keyword without duplicates in a 5*5 matrix in the row order and fill the remaining cells with missed out letters in alphabetical order. Note that 'i' and 'j' takes the same cell.

STEP-4: Group the plain text in pairs and match the corresponding corner letters by forming a rectangular grid.

STEP-5: Display the obtained cipher text.

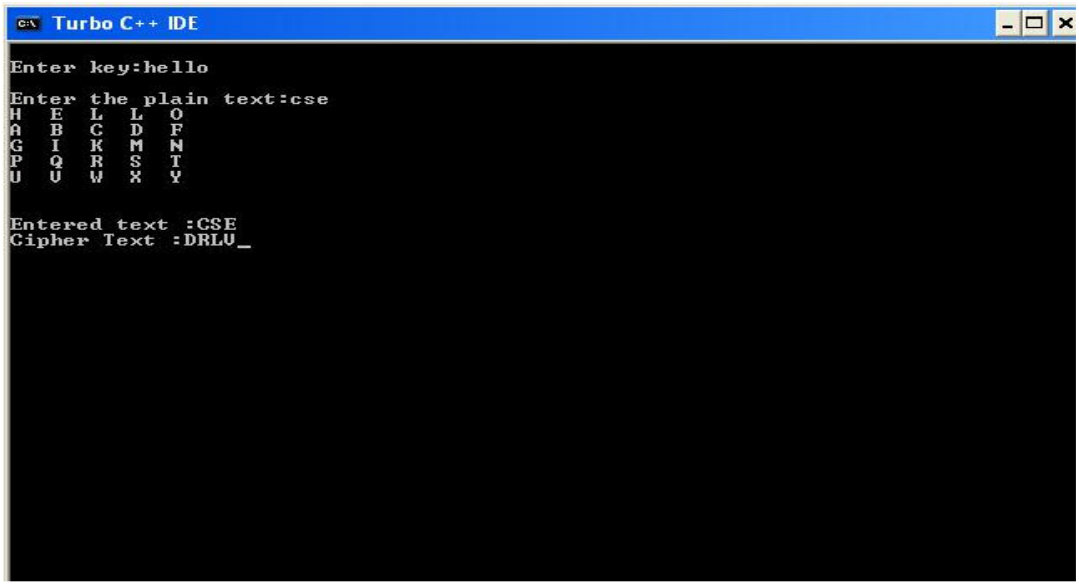
PROGRAM: (Playfair Cipher)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
#define MX 5
void playfair(char ch1,char ch2, char key[MX][MX])
{
        int i,j,w,x,y,z;
        FILE *out;
if((out=fopen("cipher.txt","a+"))==NULL)
{
        printf("File Currupted.");
}
for(i=0;i<MX;i++)
{
        for(j=0;j<MX;j++)
        {
        if(ch1==key[i][j])
        {
        w=i;
        x=j;
        }
        else if(ch2==key[i][j])
        {
        y=i;
        z=j;
}}}
//printf("%d%d %d%d",w,x,y,z);
if(w==y)
{
        x=(x+1)%5;z=(z+1)%5;
        printf("%c%c",key[w][x],key[y][z]);
        fprintf(out, "%c%c",key[w][x],key[y][z]);
```

```
}
else if(x==z)
{
        w=(w+1)%5;y=(y+1)%5;
        printf("%c%c",key[w][x],key[y][z]);
        fprintf(out, "%c%c",key[w][x],key[y][z]);
}
else
{
        printf("%c%c",key[w][z],key[y][x]); fprintf(out,
        "%c%c",key[w][z],key[y][x]);
}
fclose(out);
}
void main()
{
        int i,j,k=0,l,m=0,n;
        char key[MX][MX],keyminus[25],keystr[10],str[25]={0};
        char
        alpa[26]={'A','B','C','D','E','F','G','H','I','J','K','L'
        ,'M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'}
        ;
        clrscr();
        printf("\nEnter key:");
        gets(keystr);
        printf("\nEnter the plain text:");
        gets(str);
        n=strlen(keystr);
        //convert the characters to uppertext for (i=0; i<n; i++) {

                if(keystr[i]=='j')keystr[i]='i';
                else if(keystr[i]=='J')keystr[i]='I'; keystr[i] =
                toupper(keystr[i]);
        }
        //convert all the characters of plaintext to uppertext for (i=0; i<strlen(str); i++) {

                if(str[i]=='j')str[i]='i';
                else if(str[i]=='J')str[i]='I';
                str[i] = toupper(str[i]);
        }
        j=0;
        for(i=0;i<26;i++)
        {
                for(k=0;k<n;k++)
                {
                if(keystr[k]==alpa[i])
                break;
```

```
                else if(alpa[i]=='J')
                break;
                }
                if(k==n)
                {
                keyminus[j]=alpa[i];j++;
                }
        }


        //construct key keymatrix
        k=0;
        for(i=0;i<MX;i++)
        {
                for(j=0;j<MX;j++)
                {
                if(k<n)
                {
                key[i][j]=keystr[k];
                k++;}
                else
                {
                key[i][j]=keyminus[m];m++;
                }
                printf("%c       ",key[i][j]);
                }
                printf("\n");
        }
        printf("\n\nEntered text :%s\nCipher Text :",str); for(i=0;i<strlen(str);i++)
        {

        if(str[i]=='J')str[i]='I';
        if(str[i+1]=='\0')
        playfair(str[i],'X',key);
        else
        {
                if(str[i+1]=='J')str[i+1]='I';
                if(str[i]==str[i+1])
                playfair(str[i],'X',key);
                else
                {
                playfair(str[i],str[i+1],key);i++;
                }}
        }
getch();
}
```

OUTPUT:


Turbo C++ IDE
Enter key:hello
Enter the plain text:cse
H E L L O
A B C D F
G I K M N
P Q R S T
U V W X Y
Entered text :CSE
Cipher Text :DRLU_

Experiment 7: Implementation of hill cipher

Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B=1... Z = 25, is used, but this i s not an essential feature of the cipher. To encrypt a message, each block of n letters is multiplied by an invertible $n \times n$ matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible $n \times n$ matrices (modulo 26).

$$\begin{bmatrix} 2 & 4 & 5 \\ 9 & 2 & 1 \\ 3 & 17 & 7 \end{bmatrix} \begin{bmatrix} 0 \\ 19 \\ 19 \end{bmatrix} = \begin{bmatrix} 171 \\ 57 \\ 456 \end{bmatrix} \pmod{26} = \begin{bmatrix} 15 \\ 5 \\ 14 \end{bmatrix} = \text{'PFO'}$$

ALGORITHM:

STEP-1: Read the plain text and key from the user.

STEP-2: Split the plain text into groups of length three.

STEP-3: Arrange the keyword in a 3*3 matrix.

STEP-4: Multiply the t wo matrices to obtain the cipher text of length three.

STEP-5: Combine all these groups to get the complete cipher text.

PROGRAM: (Hill Cipher)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main(){
    unsigned int a[3][3]={{6,24,1},{13,16,10},{20,17,15}}; unsigned int
    b[3][3]={{8,5,10},{21,8,21},{21,12,8}}; int i,j, t=0;
    unsigned int c[20],d[20];
    char msg[20];
    clrscr();
    printf("Enter plain text\n ");
    scanf("%s",msg);
    for(i=0;i<strlen(msg);i++)
    {   c[i]=msg[i]-65;
            printf("%d ",c[i]);
    }
```

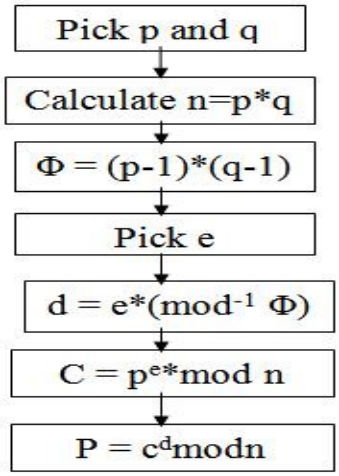
```
        for(i=0;i<3;i++)
        { t=0; for(j=0;j<3;j++)
        {
                t=t+(a[i][j]*c[j]);
        }
                d[i]=t%26;
        }
printf("\nEncrypted Cipher Text :");
for(i=0;i<3;i++)
printf(" %c",d[i]+65);
for(i=0;i<3;i++)
{
        t=0;
        for(j=0;j<3;j++)
        {
                t=t+(b[i][j]*d[j]);
        }
        c[i]=t%26;
}
printf("\nDecrypted Cipher Text :");
for(i=0;i<3;i++)
printf(" %c",c[i]+65);
getch();
return 0;
}
```

Experiment 8: Implementation of RSA

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given to everyone. A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integer m:

$$(m^e)^d = m \pmod n$$

The public key is represented by the integers n and e; and, the private key, by the integer d. m represents the message. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key.



ALGORITHM:

STEP-1: Select two co-prime numbers as p and q.

STEP-2: Compute n as the product of p and q.

STEP-3: Compute (p-1)*(q-1) and store it in z.

STEP-4: Select a random prime number e that is less than that of z.

STEP-5: Compute the private key, d as e * mod^{-1}(z).

STEP-6: The cipher text is computed as $message^e$ * mod n.

STEP-7: Decryption is done as $cipher^d$mod n.

PROGRAM: (RSA)

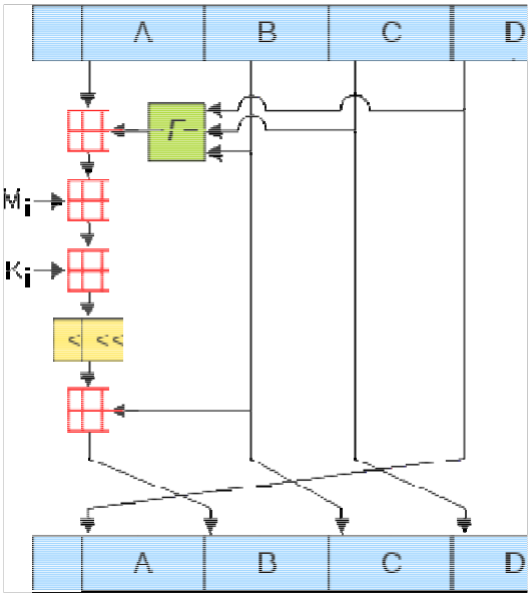
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
        long int
        p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
        char msg[100];
        int prime(long int);
        void ce();
        long int cd(long int);
        void encrypt();
        void decrypt();
void main()
{
        clrscr();
        printf("\nENTER FIRST PRIME NUMBER\n");
        scanf("%d",&p);
        flag=prime(p);
        if(flag==0)
          {
                printf("\nWRONG INPUT\n");
                getch();
          }
          printf("\nENTER ANOTHER PRIME NUMBER\n");
          scanf("%d",&q);
          flag=prime(q);
          if(flag==0||p==q)
          {
            printf("\nWRONG INPUT\n");
            getch();
          }
          printf("\nENTER MESSAGE\n");
          fflush(stdin);
          scanf("%s",msg);
          for(i=0;msg[i]!=NULL;i++)
          m[i]=msg[i];
          n=p*q;
          t=(p-1)*(q-1);
          ce();
          printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
          for(i=0;i<j-1;i++)
          printf("\n%ld\t%ld",e[i],d[i]);
```

```
            encrypt();
            decrypt();
            getch();
}
int prime(long int pr)
{
int i;
j=sqrt(pr);
for(i=2;i<=j;i++)
{
if(pr%i==0)
return 0;
}
return 1;
}
void ce()
{
int k;
k=0;
for(i=2;i<t;i++)
{
  if(t%i==0)
  continue;
  flag=prime(i);
  if(flag==1&&i!=p&&i!=q)
  {
  e[k]=i;
  flag=cd(e[k]);
  if(flag>0)
  {
  d[k]=flag;
  k++;
  }
  if(k==99)
  break;
  } } }
  long int cd(long int x)
  {
  long int k=1;
  while(1)
  {
  k=k+t;
  if(k%x==0)
  return(k/x);
  } }
  void encrypt() {
  long int pt,ct,key=e[0],k,len;
```

```
i=0;
len=strlen(msg);
while(i!=len) {
pt=m[i];
pt=pt-96;
k=1;
for(j=0;j<key;j++)
{ k=k*pt; k=k%n;
}
temp[i]=k;
ct=k+96;
en[i]=ct;
i++;
}
en[i]=-1;
printf("\nTHE ENCRYPTED MESSAGE IS\n");
for(i=0;en[i]!=-1;i++) printf("%c",en[i]);
}
void decrypt()
{
long int pt,ct,key=d[0],k; i=0;
while(en[i]!=-1)
{
ct=temp[i];
k=1;
for(j=0;j<key;j++)
{
k=k*ct;
k=k%n;
}
pt=k+96;
m[i]=pt;
i++;
}
m[i]=-1;
printf("\nTHE DECRYPTED MESSAGE IS\n");
for(i=0;m[i]!=-1;i++) printf("%c",m[i]);
}
```

Experiment 9: IMPLEMENTATION OF MD5

MD5 processes a variable-length message into a fixed-length output of 128 bits. The input message is broken up int o chunks of 512-bit blocks. The message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to b ring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with 64 bits representing the length of the original message, modulo 2^{64}.The main M D5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C, and D. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state



ALGORITHM:

STEP-1: Read the 128-bi t plain text.

STEP-2: Divide into four blocks of 32-bits named as A, B, C and D.

STEP-3: Compute the functions f, g, h and i with operations such as, rotations, permutations, etc,.

STEP-4: The output of these functions are combined together as F and performed circular shifting and then given to key round.

STEP-5: Finally, right shift of 's' times are performed and the results are combined together to produce the final output.

PROGRAM:(MD5)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include<conio.h>
typedef union uwb
{
        unsigned w;
        unsigned char b[4];
} MD5union;
         typedef unsigned DigestArray[4];
         unsigned func0( unsigned abcd[] ){
         return ( abcd[1] & abcd[2]) | (~abcd[1] & abcd[3]);} unsigned func1( unsigned
         abcd[] ){
         return ( abcd[3] & abcd[1]) | (~abcd[3] & abcd[2]);} unsigned func2( unsigned
         abcd[] ){
         return abcd[1] ^ abcd[2] ^ abcd[3];} unsigned func3( unsigned
         abcd[] ){ return abcd[2] ^ (abcd[1] |~ abcd[3]);} typedef unsigned
         (*DgstFctn)(unsigned a[]);
unsigned *calctable( unsigned *k)
{
         double s, pwr;
         int i;
         pwr = pow( 2, 32);
for (i=0; i<64; i++)
{
         s = fabs(sin(1+i));
         k[i] = (unsigned)( s * pwr );
}
         return k;
}
unsigned rol( unsigned r, short N )
{
        unsigned       mask1 = (1<<N) -1;
        return ((r>>(32-N)) & mask1) | ((r<<N) & ~mask1);
```

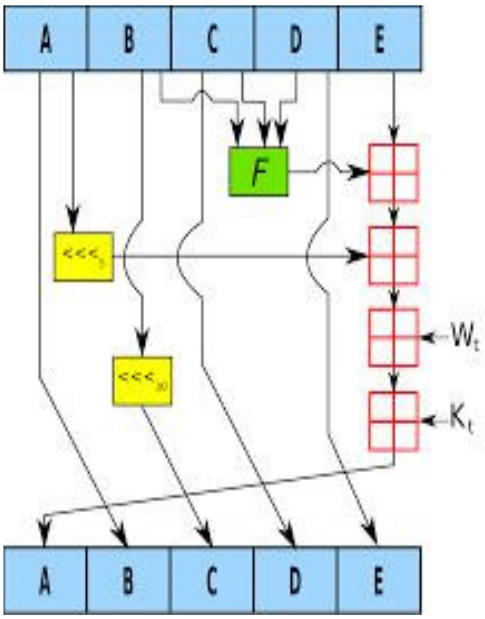
```
}
unsigned *md5( const char *msg, int mlen)
{
        static DigestArray h0 =         { 0x67452301, 0xEFCDAB89,
        0x98BADCFE, 0x10325476 };
        static DgstFctn ff[]            = { &func0, &func1, &func2, &func3};
        static short M[] = {            1,   5,3,7};
        static short O[] = {            0,   1,5,0};
        static short rot0[] = {             7,12,17,22};
        static short rot1[] = {             5, 9,14,20};
        static short rot2[] = {             4,11,16,23};
        static short rot3[] = {             6,10,15,21};
        static short *rots[]            = {rot0, rot1, rot2, rot3 };
        static unsigned kspace[64];
        static unsigned *k;
        static DigestArray h;
        DigestArray abcd;
        DgstFctn fctn;
        short m, o, g;
        unsigned f;
        short *rotn;
        union
        {
                unsigned w[16];
                char            b[64];
        }mm;
                int os = 0;
                int grp, grps, q, p;
                unsigned char *msg2;
                if (k==NULL) k= calctable(kspace);
        for (q=0; q<4; q++) h[q] = h0[q];                       // initialize
        {
                  grps      = 1 + (mlen+8)/64;
                  msg2 = malloc( 64*grps);
                  memcpy( msg2, msg, mlen);
                  msg2[mlen] = (unsigned char)0x80;
                  q = mlen + 1;
          while (q < 64*grps){ msg2[q] = 0; q++ ; } {
                  MD5union u;
                  u.w = 8*mlen;
                  q -= 8;
```

```
            memcpy(msg2+q, &u.w, 4 );
        }
        }
        for (grp=0; grp<grps; grp++)
        {
            memcpy( mm.b, msg2+os, 64);
            for(q=0;q<4;q++) abcd[q] = h[q];
            for (p = 0; p<4; p++)
            {
                fctn = ff[p];
                rotn = rots[p];
                m = M[p]; o= O[p];
            for (q=0; q<16; q++)
            {
                g = (m*q + o) % 16;
                f = abcd[1] + rol( abcd[0]+ fctn(abcd)+k[q+16*p]
                +  mm.w[g], rotn[q%4]); abcd[0] =
                abcd[3]; abcd[3] = abcd[2]; abcd[2] =
                abcd[1];
                abcd[1] = f;
            }}
        for (p=0; p<4; p++)
        h[p] += abcd[p];
        os += 64;
        }
    return h;}
    void main()
    {
        int j,k;
        const char *msg = "The quick brown fox jumps over the lazy dog";
        unsigned *d = md5(msg, strlen(msg));
        MD5union u;
        clrscr();
        printf("\t MD5 ENCRYPTION ALGORITHM IN C \n\n");
        printf("Input String to be Encrypted using MD5 : \n\t%s",msg);
        printf("\n\nThe MD5 code for input string is: \n");
        printf("\t= 0x");
        for (j=0;j<4; j++){
        u.w = d[j];
        for (k=0;k<4;k++) printf("%02x",u.b[k]);
    }
```

```
printf("\n");
printf("\n\t MD5 Encyption Successfully
Completed!!!\n\n");
getch();
system("pause");
getch();}
```

Experiment 10: IMPLEMENTATION OF SHA-I

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function. SHA-1 produces a 160-bit hash value known as a message digest. The way this algorithm works is that for a message of size < 264 bits it computes a 160-bit condensed output called a message digest. The SHA-1 algorithm is designed so that it is practically infeasible to find two input messages that hash to the same output message. A hash function such as SHA-1 is used to calculate an alphanumeric string that serves as the cryptographic representation of a file or a piece of data. This is called a digest and can serve as a digital signature. It is supposed to be unique and non-reversible.



ALGORITHM:

STEP-1: Read the 256-bit key values.

STEP-2: Divide into five equal-sized blocks named A, B, C, D and E.

STEP-3: The blocks B, C and D are passed to the function F.

STEP-4: The resultant value is permuted with block E.

STEP-5: The block A is shifted right by 's' times and permuted with the result of step-4

STEP-6: Then it is permuted with a weight value and then with some other key pair and taken as the first block.

STEP-7: Block A is taken as the second block and the block B is shifted by 's' times and taken as the third block.

STEP-8: The blocks C and D are taken as the block D and E for the final output.

PROGRAM: (Secure Hash Algorithm)

```
import java.security.*;

public class SHA1 {

    public static void main(String[] a) { try {

    MessageDigest md = MessageDigest.getInstance("SHA1");
    System.out.println("Message digest object info: "); System.out.println("
    Algorithm = " +md.getAlgorithm()); System.out.println(" Provider = "
    +md.getProvider()); System.out.println(" ToString = " +md.toString()); String
    input = "";

    md.update(input.getBytes()); byte[] output = md.digest();
    System.out.println();
    System.out.println("SHA1(\""+input+"\") =
    +bytesToHex(output)); input = "abc";


    md.update(input.getBytes());

    output = md.digest();

    System.out.println();

    System.out.println("SHA1(\""+input+"\") = "

    +bytesToHex(output));
```

```
        input = "abcdefghijklmnopqrstuvwxyz";
        md.update(input.getBytes()); output = md.digest();

        System.out.println();

        System.out.println("SHA1(\"" +input+"\") = "

        +bytesToHex(output));

        System.out.println("""); } catch (Exception e) {
        System.out.println("Exception: " +e);
}

}

public static String bytesToHex(byte[] b)

{
        char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E',
        'F'}; StringBuffer buf = new StringBuffer(); for (int j=0; j<b.length; j++) {

        buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
        buf.append(hexDigit[b[j] & 0x0f]); } return buf.toString(); }}
```

Experiment 11: Implementation of digital signature standard

ALGORITHM:

STEP-1: Alice and Bob are investigating a forgery case of x and y.

STEP-2: X had document signed by him, but he says he did not sign that document digitally.

STEP-3: Alice reads the two prime numbers p and a.

STEP-4: He chooses a random co-primes alpha and beta and the x's original signature x.

STEP-5: With these values, he applies it to the elliptic curve cryptographic equation to obtain y.

STEP-6: Comparing this 'y' with actual y's document, Alice concludes that y is a forgery.

PROGRAM: (Digital Signature Standard)

```
import java.util.*;
import java.math.BigInteger;
class dsaAlg {
        final static BigInteger one = new BigInteger("1"); final static BigInteger zero =
        new BigInteger("0");
public static BigInteger getNextPrime(String ans)
{
        BigInteger test = new BigInteger(ans); while
(!test.isProbablePrime(99)) e:
{test = test.add(one);
}
        return test;
}
public static BigInteger findQ(BigInteger n)
{BigInteger start = new BigInteger("2"); while (!n.isProbablePrime(99)) {
        while (!((n.mod(start)).equals(zero)))
        {start = start.add(one);
        }
        n = n.divide(start);
}
        return n;
}
public static BigInteger getGen(BigInteger p, BigInteger q,
Random r)
{
        BigInteger h = new BigInteger(p.bitLength(), r); h = h.mod(p);
        return h.modPow((p.subtract(one)).divide(q), p);
}
```

```
public static void main (String[] args) throws
java.lang.Exception
{
        Random randObj = new Random();
        BigInteger p = getNextPrime("10600"); /* approximate prime */
        BigInteger q = findQ(p.subtract(one)); BigInteger g = getGen(p,q,randObj);
        System.out.println(" \n simulation of Digital Signature Algorithm \n");

        System.out.println(" \n global public key components are:\n");
        System.out.println("\np is: " + p);
        System.out.println("\nq is: " + q);
        System.out.println("\ng is: " + g);
        BigInteger x = new BigInteger(q.bitLength(), randObj); x = x.mod(q);
        BigInteger y = g.modPow(x,p);
        BigInteger k = new BigInteger(q.bitLength(), randObj); k = k.mod(q);
        BigInteger r = (g.modPow(k,p)).mod(q);
        BigInteger hashVal = new BigInteger(p.bitLength(), randObj);
        BigInteger kInv = k.modInverse(q);
        BigInteger s = kInv.multiply(hashVal.add(x.multiply(r))); s = s.mod(q);
        System.out.println("\nsecret information are:\n");
        System.out.println("x (private) is:" + x);
        System.out.println("k (secret)              is: " + k);
        System.out.println("y (public)              is: " + y);
        System.out.println("h (rndhash) is: " + hashVal);
        System.out.println("\n generating digital signature:\n");
        System.out.println("r is : " + r);
        System.out.println("s is : " + s);
        BigInteger w = s.modInverse(q);
        BigInteger u1 = (hashVal.multiply(w)).mod(q); BigInteger u2 =
        (r.multiply(w)).mod(q);
        BigInteger v = (g.modPow(u1,p)).multiply(y.modPow(u2,p));
        v = (v.mod(p)).mod(q);
        System.out.println("\nverifying digital signature
        (checkpoints)\n:");
        System.out.println("w          is : " + w);
        System.out.println("u1 is : " + u1);
        System.out.println("u2 is : " + u2);
        System.out.println("v          is : " + v);
if (v.equals(r))
{System.out.println("\nsuccess: digital signature is verified!\n " + r);
}
else
{
        System.out.println("\n error: incorrect digital signature\n ");
}}}
```

Experiment 12: Wireshark Tutorial

The purpose of this document is to introduce the packet sniffer Wireshark. Wireshark would be used for the lab experiments. This document introduces the basic operation of a packet sniffer, installation, and a test run of Wireshark.

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color-coding, and other features that let you dig deep into network traffic and inspect individual packets.

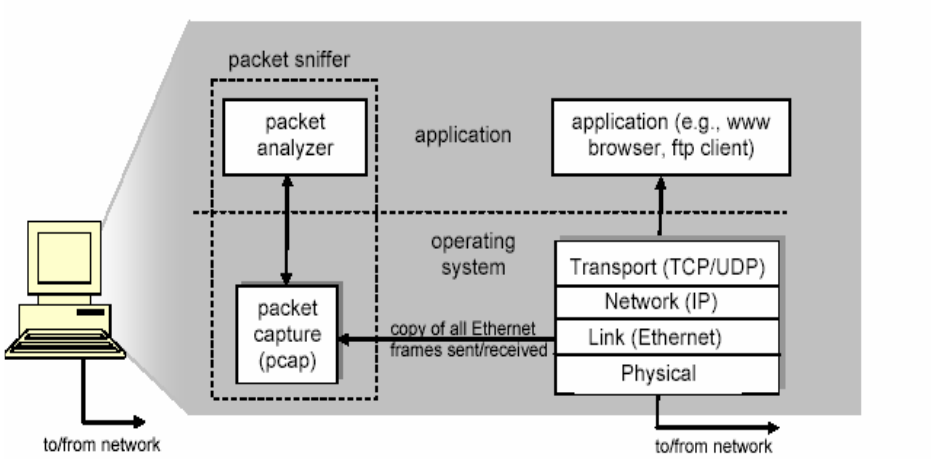


Figure 1: Packet sniffer structure

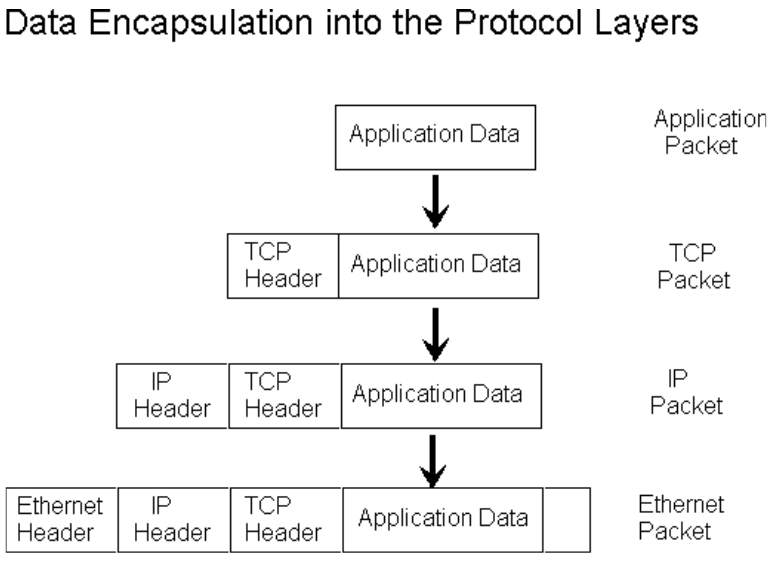


Figure 1b: Protocol Data Encapsulation

PACKET SNIFFER

The basic tool for observing the messages exchanged as packets on the network using a variety of protocols is called a **packet sniffer**. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a *copy* of packets that are sent / received from/by application and protocols executing on your machine.

Figure 1 shows the structure of a packet sniffer. At the right of Figure 1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The overall packet structure. The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer and consists of two parts. The **packet capture library** receives a copy of every link-layer frame that is sent from or received by your computer. Messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. **In Figure 1,** the assumed physical media is an Ethernet, and so all upper-layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you access to all messages sent/received from/by all protocols and applications executing in your computer. The second component of a packet sniffer is the **packet analyzer**, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must "understand" the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 1.

The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet fram. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP

protocol and so, for example, knows that the first bytes of an HTTP message will contain the string "GET," "POST," or "HEAD".

We will be using the Wireshark packet sniffer [http://www.wireshark.org/] for these labs, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. It's an ideal packet analyzer for our labs – it is stable, has a large user base and well-documented support that includes a user-guide: http://www.wireshark.org/docs/wsug_html_chunked/), a set of well-crafted man pages at (http://www.wireshark.org/docs/man-pages/), and a list of very detailed FAQ (http://www.wireshark.org/faq.html), rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, Token-Ring, FDDI, serial (PPP and SLIP), 802.11 wireless LANs, and ATM connections (if the OS on which it's running allows Wireshark to do so).

Getting Wireshark

Wireshark can be installed in any operating system, just go to
https://www.wireshark.org/download.html

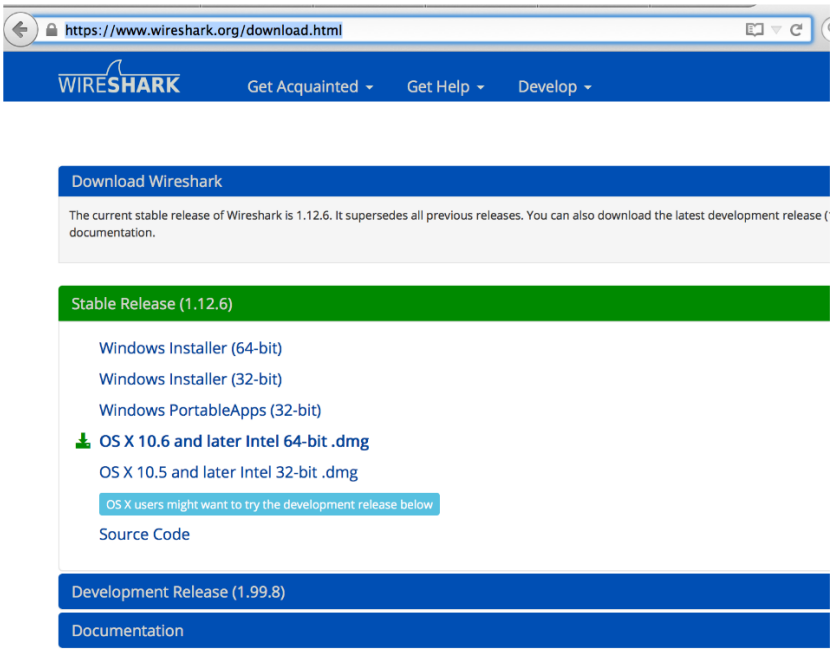


Figure 2: Wireshark Download Page.

You can download the version of wireshark that matches your operating system.

Running Wireshark

When you run the Wireshark program, the Wireshark graphical user interface shown in Figure 2 will de displayed. Initially, no data will be displayed in the various windows.

For Mac OSX users, you need to have XQuartz or X11 installed for Wireshark to work!!! Also, the first time you open Wireshark, it will take several seconds to start so be patient.

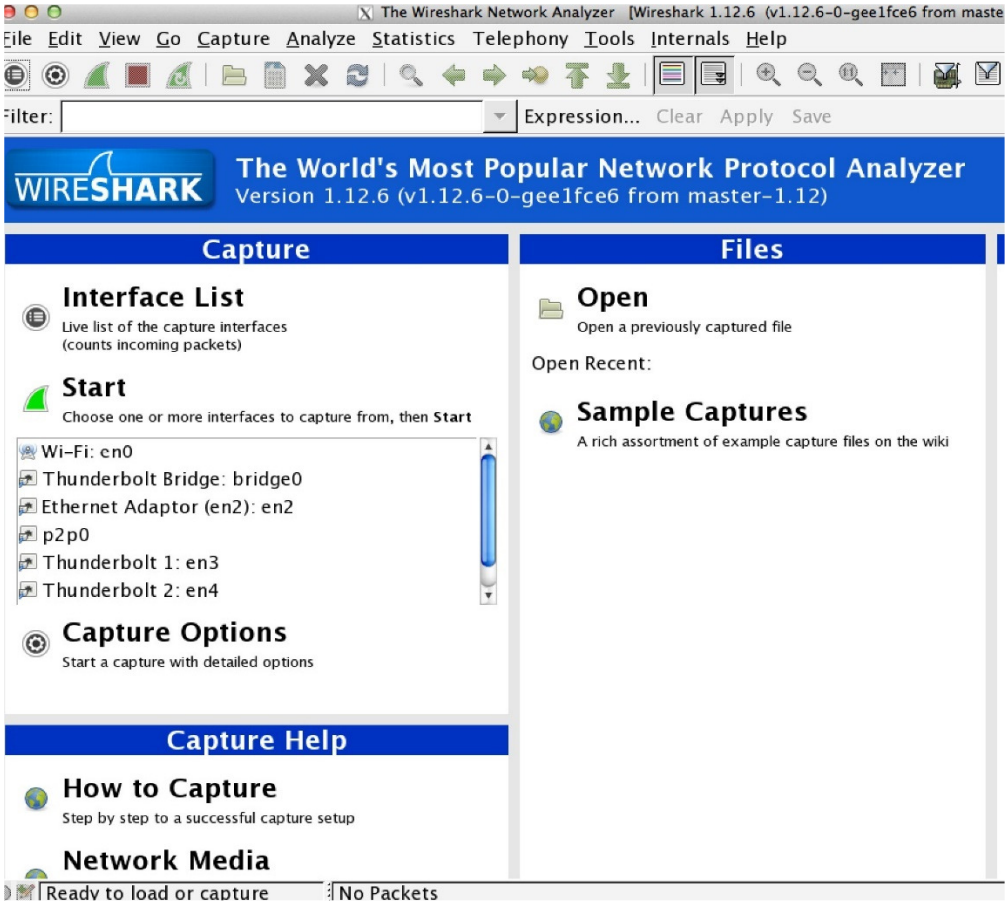


Figure 3: Initial screen for Wireshark.

You will need to select one of the Wireshark interfaces, if you are using your laptop connected over the WiFi, then you need to select the WiFi interface. If you are at a server, you need to select the Ethernet interface being used. In general, you can select any interface but that does not mean that traffic will flow through that interface. The network interfaces (i.e., the physical connections) that

your computer has to the network are shown. The attached snapshot was taken from my computer. You may not see the exact same entries when you perform a capture in the 237 Lab. You will notice that eth0 and eth1 will be displayed. Click "Start" for interface eth0. Packet capture will now begin - all packets being sent / received from/by your computer are now being captured by Wireshark!

After you select the interface, you should cick on "START". If everything goes well, you should get a view similar to Figure 2.

WIRESHARK USER INTERFACE

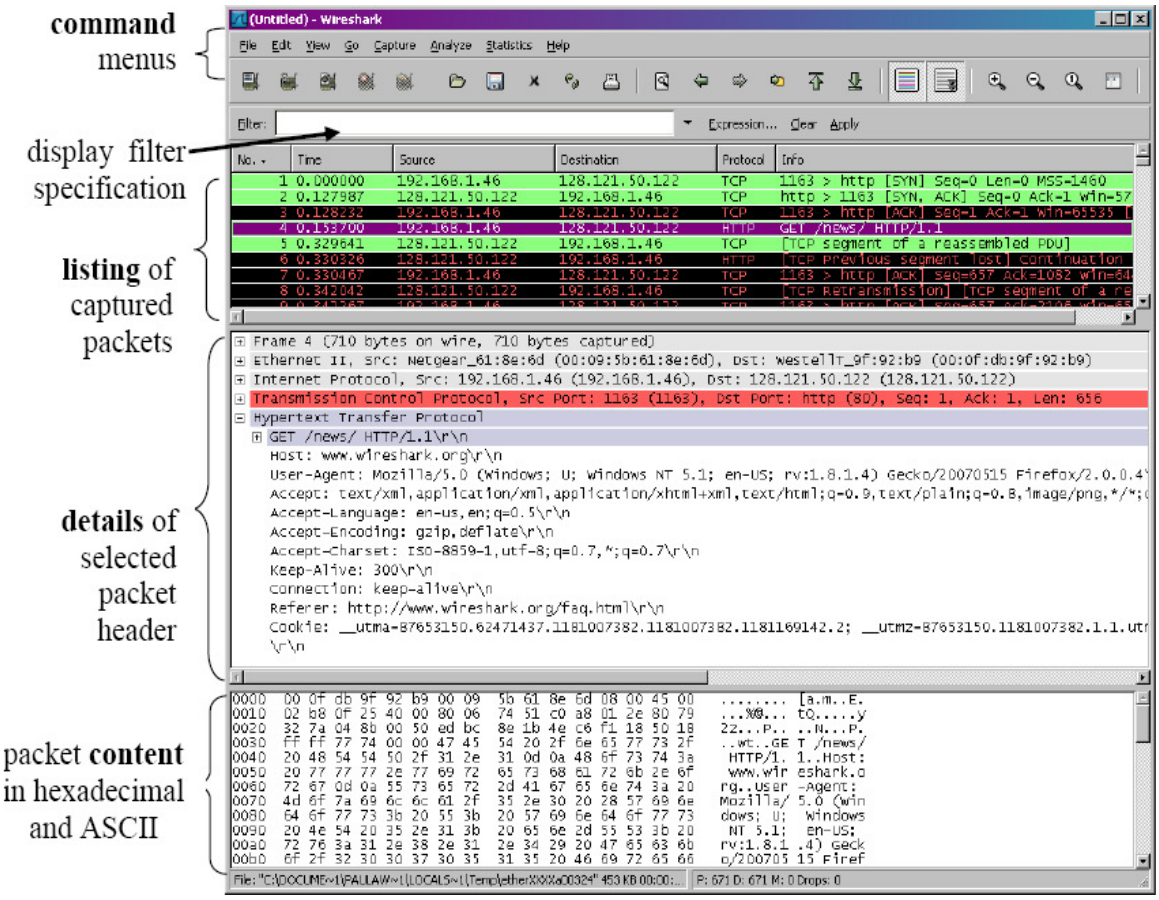


Figure 4a: Wireshark Graphical User Interface for Windows OS.

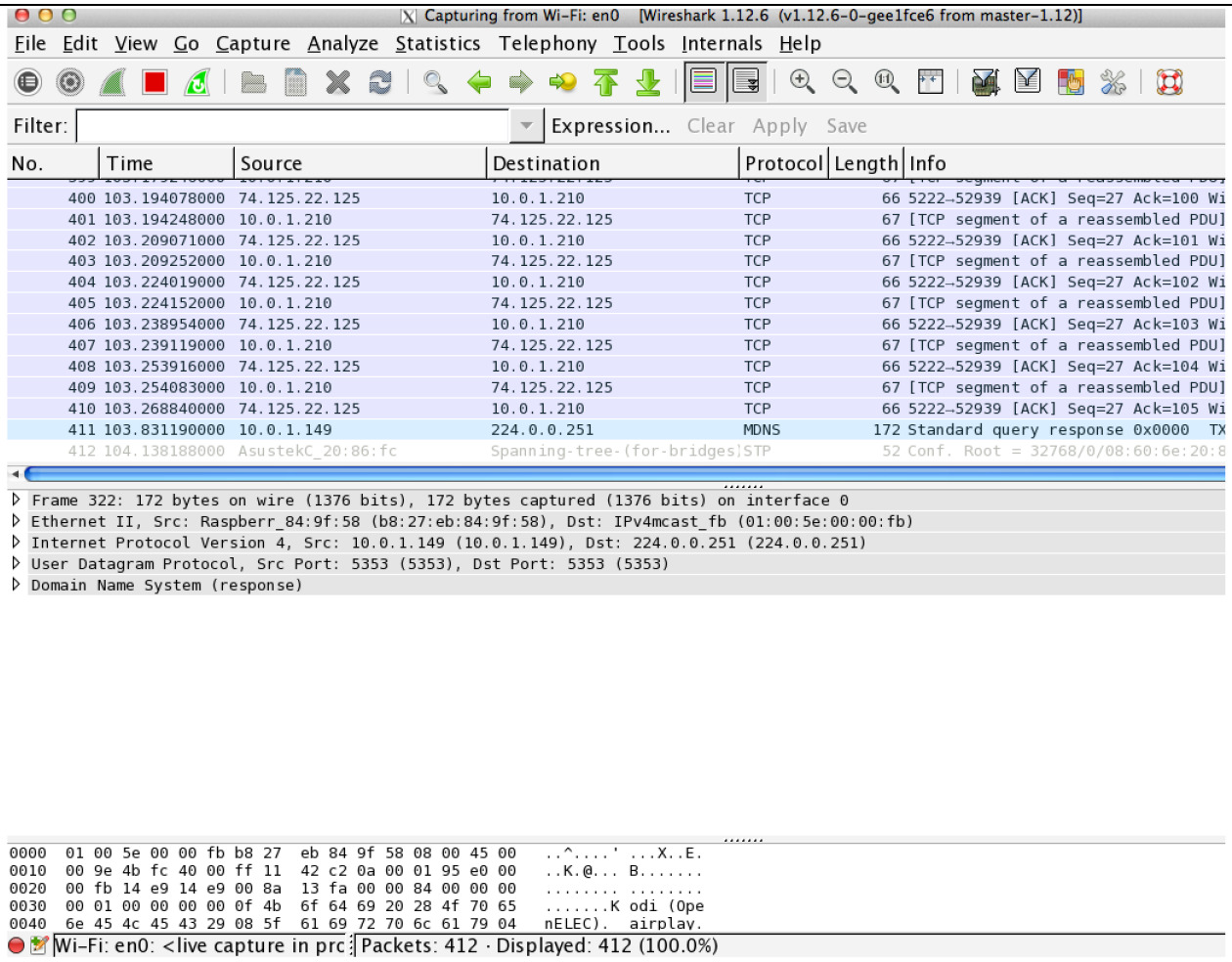


Figure 4b: Wireshark Graphical User Interface for Mac OSX.

The Wireshark interface has five major components:

- The command **menus are** standard pulldown menus located at the top window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data and exit the Wireshark application. The Capture menu allows you to begin packet capture.

- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is *not* a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.

- The packet-**header details window** provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packet-listing window, place the cursor over the packet's one-line summary in the packet-listing window and click with the left mouse button.). These details include information about the Ethernet frame and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the right-pointing or down-pointing arrowhead to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.

- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.

- Towards the top of the Wireshark graphical user interface, is the **packet-display filter field,** into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

Capturing Packets

After downloading and installing Wireshark, you can launch it and click the name of an interface under Interface List to start capturing packets on that interface. For example, if you want to capture traffic on the wireless network, click your wireless interface. You can configure advanced features by clicking Capture Options, but this isn't necessary for now.

Test Run

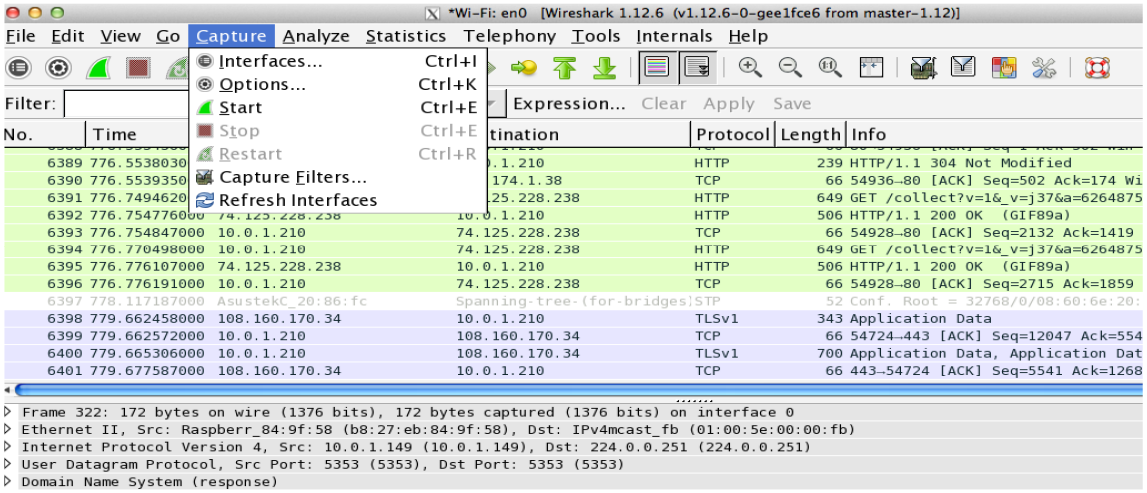
The best way to learn about any new piece of software is to try it out! Do the following

1. Start up your favorite web browser.

2. Start up the Wireshark software. You will initially see a window similar to that shown in **Figure 3. You need to select an interface and press Start.**

2. After your browser has displayed the http://www.gmu.edu page, stop Wireshark packet

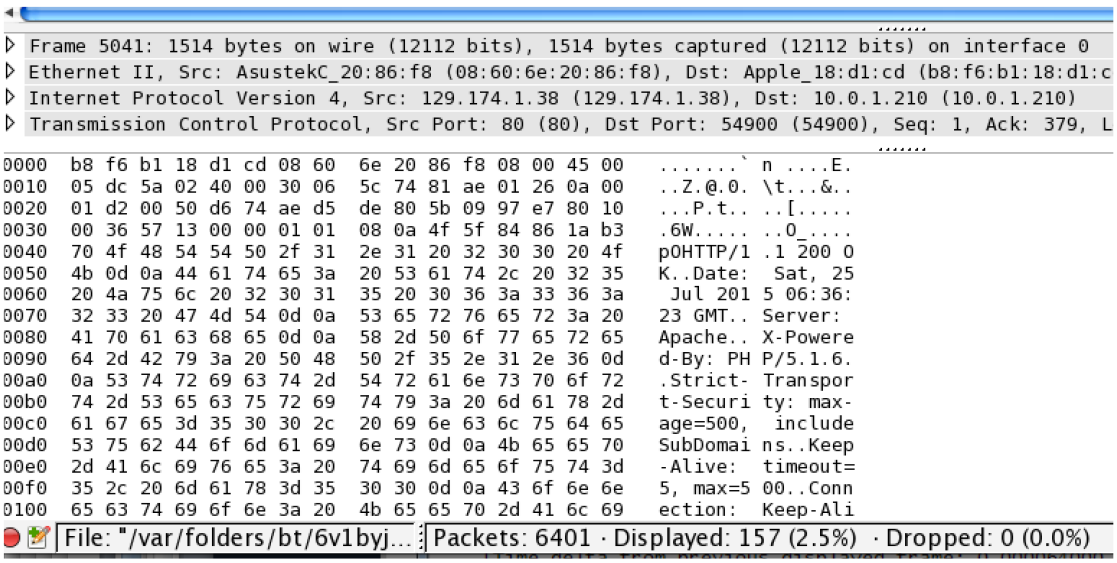
capture by selecting stop in the Wireshark capture window. This will cause the Wireshark capture window to disappear and the main Wireshark window to display all packets captured since you began packet capture see image below:



4. Color Coding: You'll probably see packets highlighted in green, blue, and black. Wireshark uses colors to help you identify the types of traffic at a glance. By default, green is TCP traffic, dark blue is DNS traffic, light blue is UDP traffic, and black identifies TCP packets with problems — for example, they could have been delivered out-of-order.

Inspecting Packets

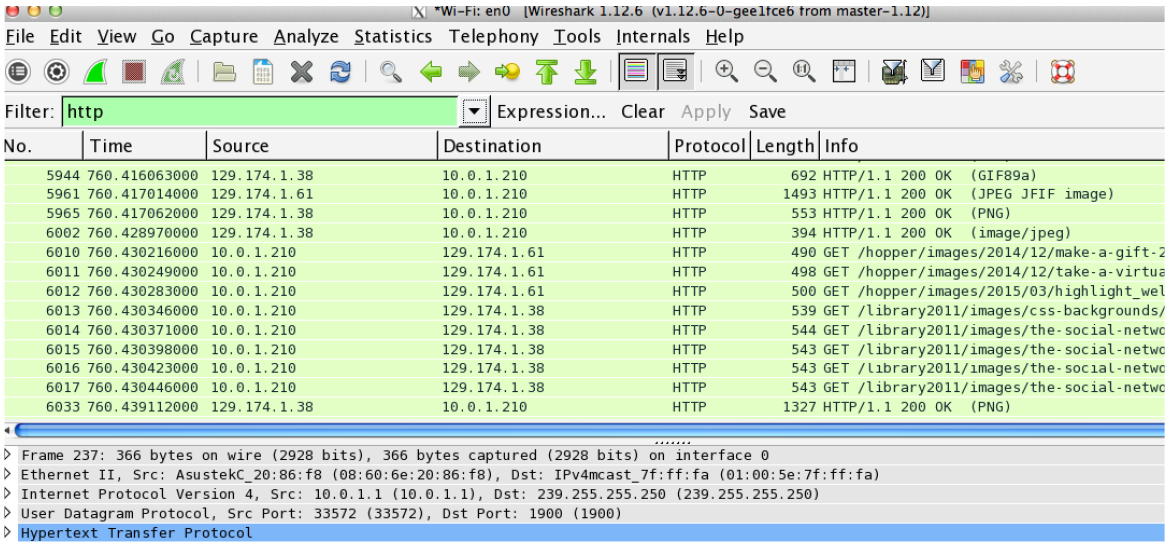
To inspect packets, click on one of the packets and go to the bottom pane:



Notice that although there is a lot of interesting information, the view of the packet is not very easy to read.

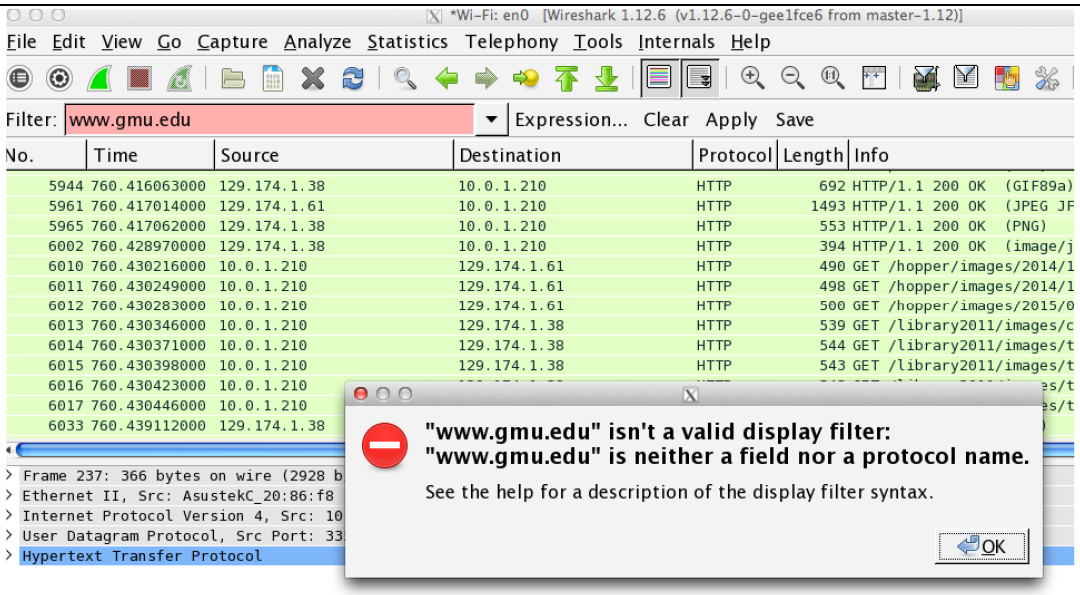
Inspecting Packet Flows (Network Connections)

5. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! However, as you will notice the HTTP messages are not clearly shown because there are many other packets included in the packet capture. Even though the only action you took was to open your browser, there are many other programs in your computer that communicate via the network in the background. To filter the connections to the ones we want to focus on, we have to use the filtering functionality of Wireshark by typing "http" in the filtering field as shown below:

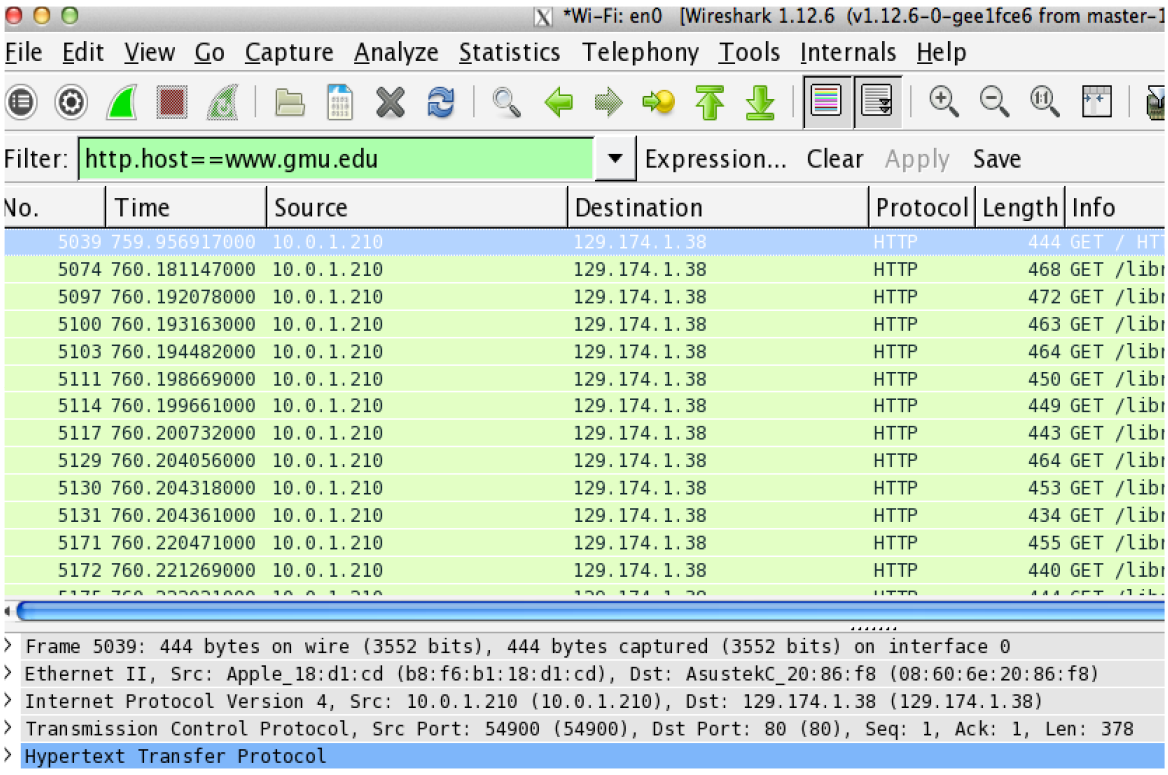


Notice that we now view only the packets that are of protocol HTTP. However, we also still do not have the exact communication we want to focus on because using HTTP as a filter is not descriptive enough to allow us to find our connection to http://www.gmu.edu. We need to be more precise if we want to capture the correct set of packets.

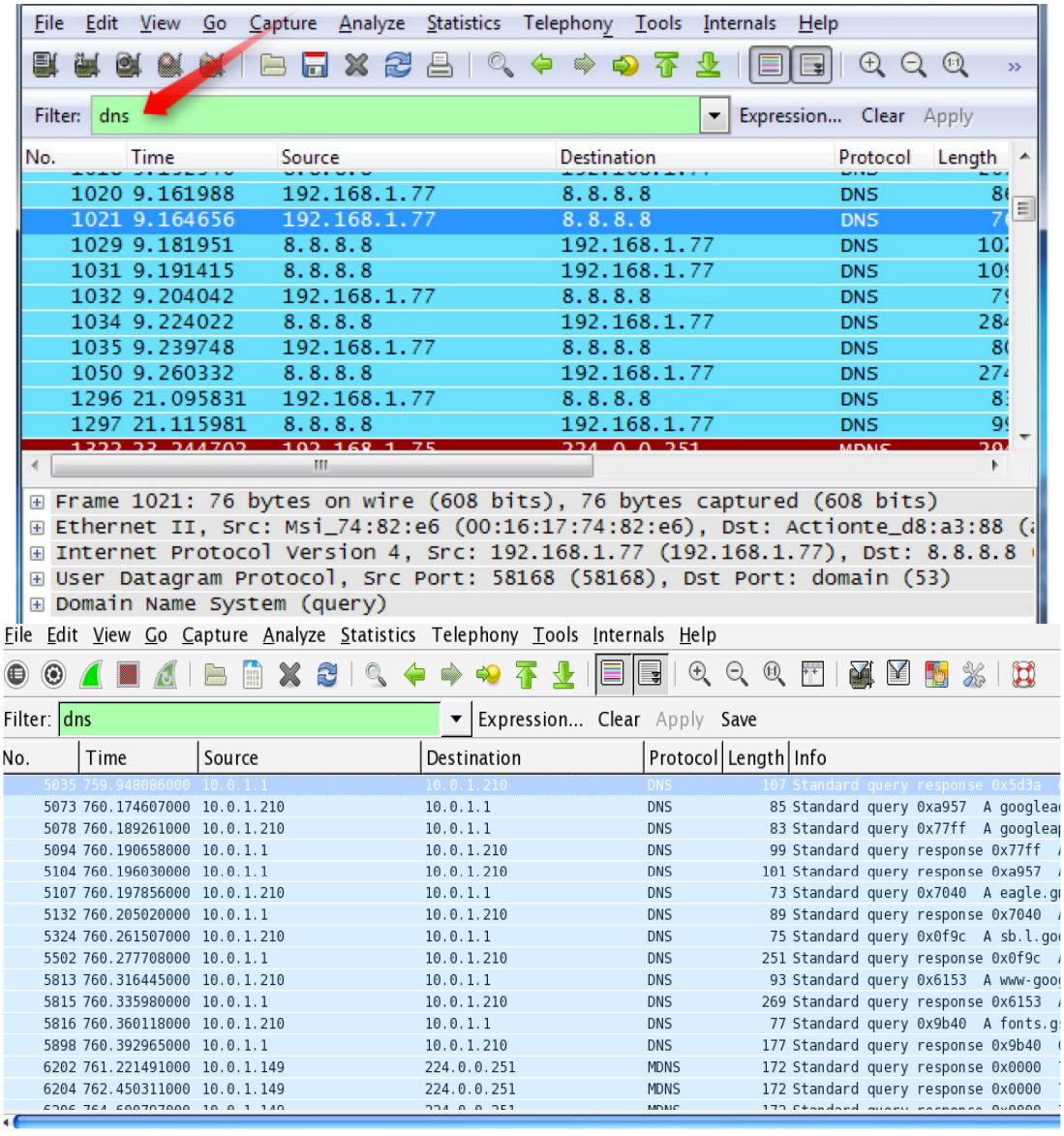
6. One potential filter that comes to mind is to type the destination host (www.gmu.edu) directly in the filter area. Unfortunately, this will end up bringing up an error (see screenshot below) and will not work because Wireshark does not have the ability to discern which protocol fields you need to match.



7. To further filter packets in Wireshark, we need to use a more precise filter. By setting the http.host==www.gmu.edu, we are restricting the view to packets that have as an http host the www.gmu.edu website. Notice that we need two equal signs to perform the match "==" not just one!

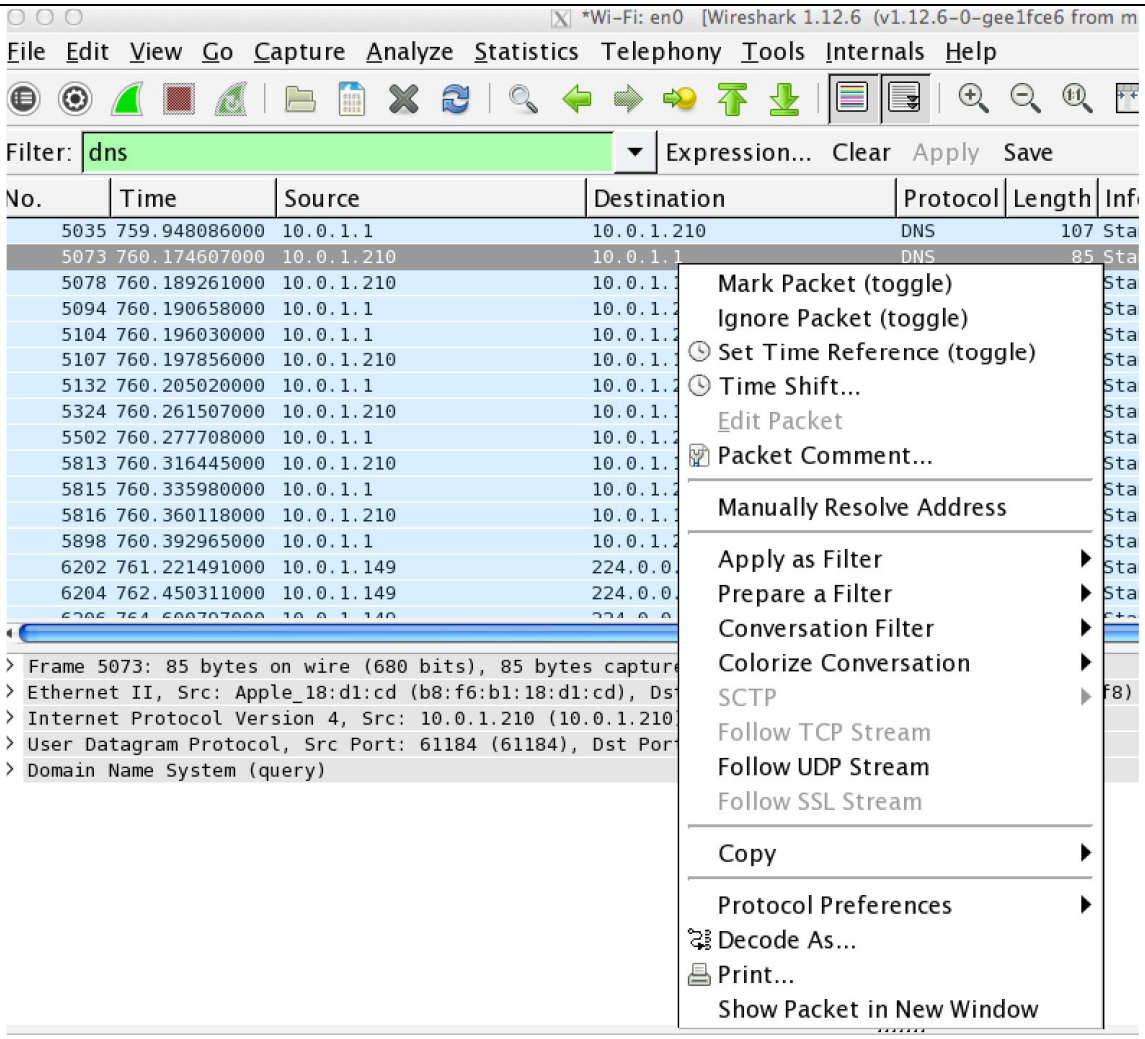


8. Let's try another protocol, for instance DNS, type DNS to the filter area:



Do you notice that the packets are colored differently?

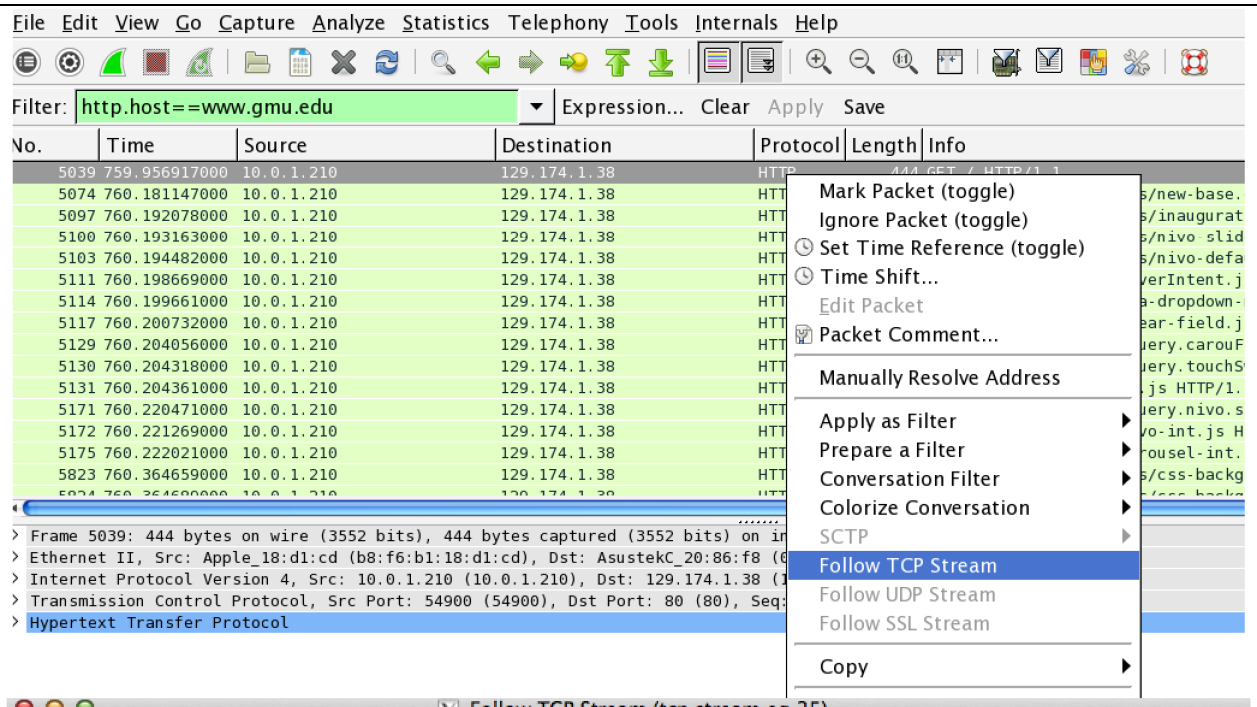
Let's try now to find out what are those packets contain by following one of the conversations (also called network flows), select one of the packets and press the right mouse button (if you are on a Mac use the command button and click), you should see something similar to the screen below:

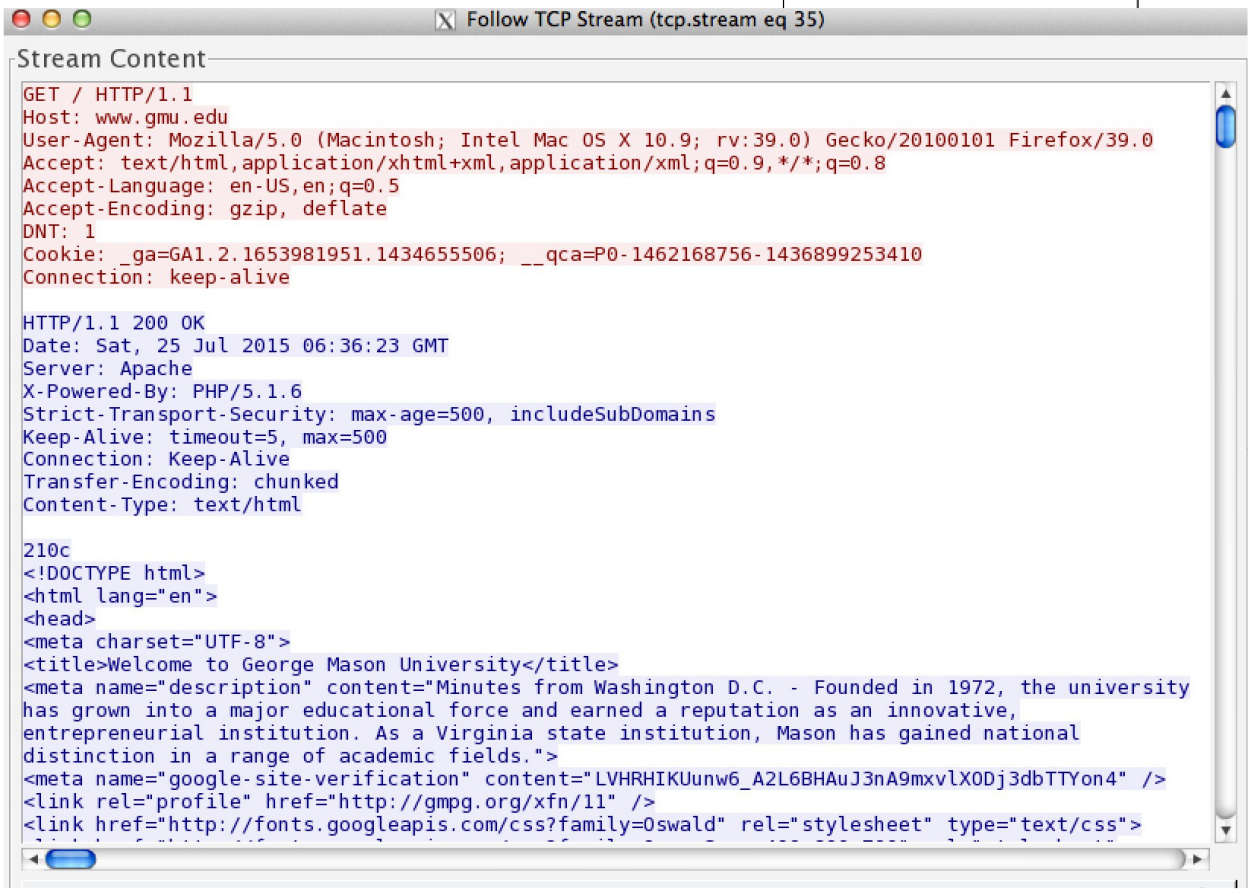


From the drop-down list select the **"Follow UDP Stream"** selection.

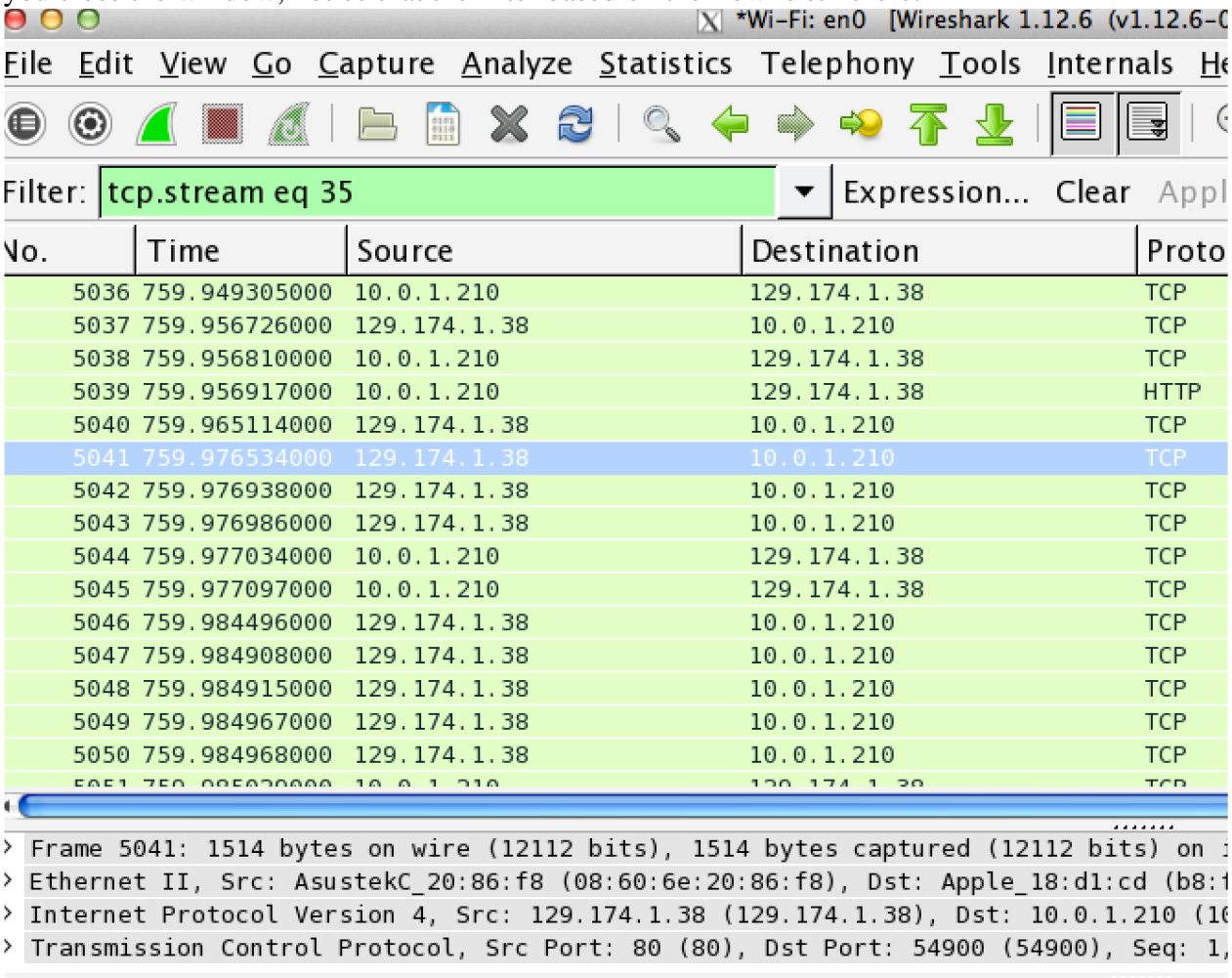
You should be able to get something similar to the screen below:

If we close this window and change the filter back to "http.host==www.gmu.edu" and then follow a packet from the list of packets that match that filter, we should get the something similar to the screens in the next page:

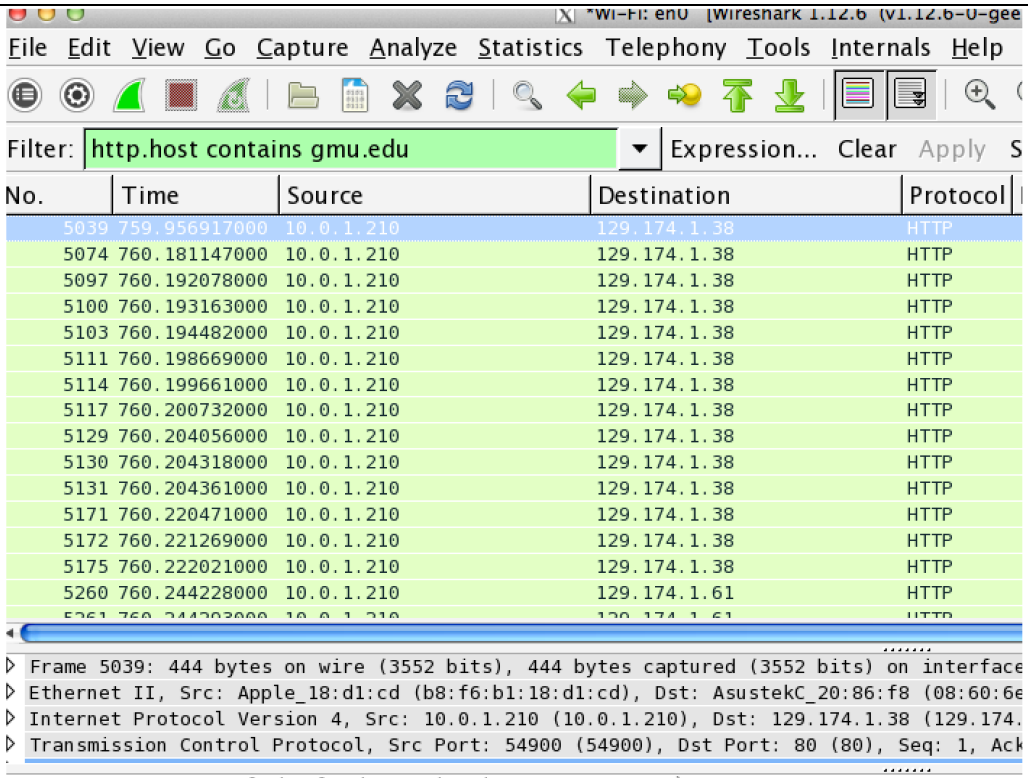

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: http.host==www.gmu.edu
Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
5039 759.956917000 10.0.1.210 129.174.1.38
5074 760.181147000 10.0.1.210 129.174.1.38
5097 760.192078000 10.0.1.210 129.174.1.38
5100 760.193163000 10.0.1.210 129.174.1.38
5103 760.194482000 10.0.1.210 129.174.1.38
5111 760.198669000 10.0.1.210 129.174.1.38
5114 760.199661000 10.0.1.210 129.174.1.38
5117 760.200732000 10.0.1.210 129.174.1.38
5129 760.204056000 10.0.1.210 129.174.1.38
5130 760.204318000 10.0.1.210 129.174.1.38
5131 760.204361000 10.0.1.210 129.174.1.38
5171 760.220471000 10.0.1.210 129.174.1.38
5172 760.221269000 10.0.1.210 129.174.1.38
5175 760.222021000 10.0.1.210 129.174.1.38
5823 760.364659000 10.0.1.210 129.174.1.38
Mark Packet (toggle)
Ignore Packet (toggle)
Set Time Reference (toggle)
Time Shift...
Edit Packet
Packet Comment...
Manually Resolve Address
Apply as Filter
Prepare a Filter
Conversation Filter
Colorize Conversation
SCTP
Follow TCP Stream
Follow UDP Stream
Follow SSL Stream
Copy
Frame 5039: 444 bytes on wire (3552 bits), 444 bytes captured (3552 bits) on in
Ethernet II, Src: Apple_18:d1:cd (b8:f6:b1:18:d1:cd), Dst: AsustekC_20:86:f8 (0
Internet Protocol Version 4, Src: 10.0.1.210 (10.0.1.210), Dst: 129.174.1.38 (1
Transmission Control Protocol, Src Port: 54900 (54900), Dst Port: 80 (80), Seq:
Hypertext Transfer Protocol


Follow TCP Stream (tcp.stream eq 35)
Stream Content
GET / HTTP/1.1
Host: www.gmu.edu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:39.0) Gecko/20100101 Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: _ga=GA1.2.1653981951.1434655506; __qca=P0-1462168756-1436899253410
Connection: keep-alive
HTTP/1.1 200 OK
Date: Sat, 25 Jul 2015 06:36:23 GMT
Server: Apache
X-Powered-By: PHP/5.1.6
Strict-Transport-Security: max-age=500, includeSubDomains
Keep-Alive: timeout=5, max=500
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
210c
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Welcome to George Mason University</title>
<meta name="description" content="Minutes from Washington D.C. - Founded in 1972, the university has grown into a major educational force and earned a reputation as an innovative, entrepreneurial institution. As a Virginia state institution, Mason has gained national distinction in a range of academic fields.">
<meta name="google-site-verification" content="LVHRHIKUunw6_A2L6BHAuJ3nA9mxvlXODj3dbTTYon4" />
<link rel="profile" href="http://gmpg.org/xfn/11" />
<link href="http://fonts.googleapis.com/css?family=Oswald" rel="stylesheet" type="text/css">
<link

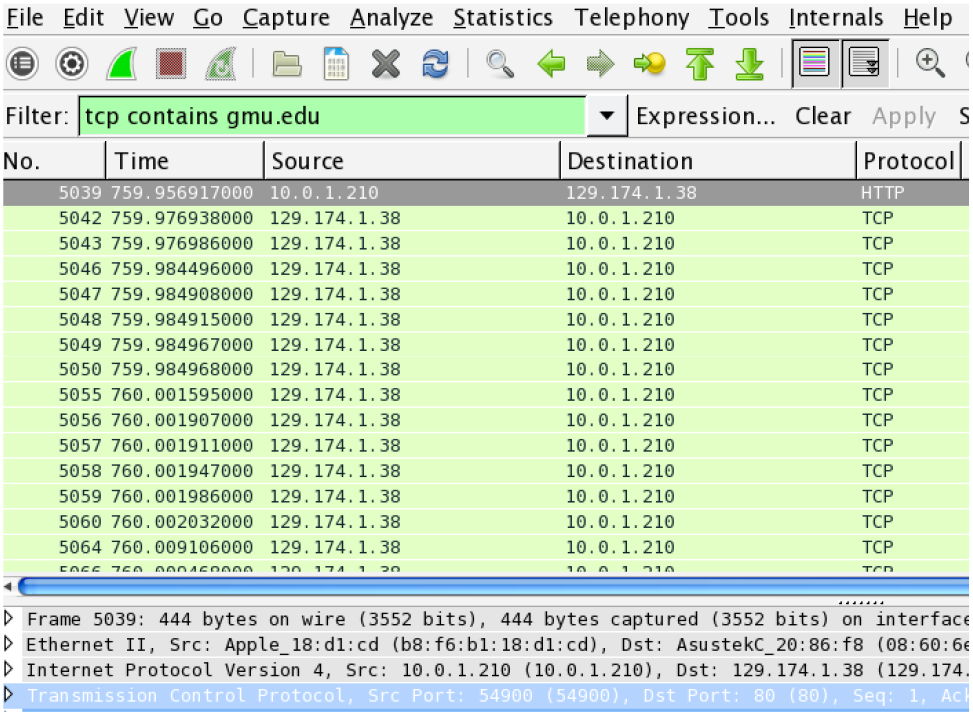
If you close the window, notice that the filter based on the flow is stil there:



Besides streams we can also use the keyword “contains” to allow some freedom for searching. Let’s see how:



Also, we can use **"contains" with other protocols!**



Experiment 13: Installation of rootkits and study about the variety of options.

Introduction:

- Rootkit is a stealth type of malicious software designed to hide the existence of certain process from normal methods of detection and enables continued privileged access to a computer.
- Breaking the term rootkit into the two component words, root and kit, is a useful way to define it. Root is a UNIX/Linux term that's the equivalent of Administrator in Windows. The word kit denotes programs that allow someone to obtain root/admin-level access to the computer by executing the programs in the kit
  - all of which is done without end-user consent or knowledge.
- A rootkit is a type of malicious software that is activated each time your system boots up.
- Rootkits are difficult to detect because they are activated before your system's OS has completely booted up.
- A rootkit often allows the installation of hidden files, processes, hidden user accounts etc. in the systems OS.
- Rootkits can intercept data from terminals, network connections, and the keyboard.
- Rootkits have two primary functions: remote command/control (back door) and software eavesdropping.
- Rootkits allow someone executing files, accessing logs, monitoring user activity, and even changing the computer's configuration.
- The presence of a rootkit on a network was first documented in the early 1990s. At that time, Sun and Linux operating systems were the primary targets for a hacker looking to install a rootkit.
- Today, rootkits are available for several operating systems, including Windows, and are ever more difficult to detect on any network.

Procedure to install Rootkit Tool

STEP-1: Download Rootkit Tool from GMER website www.gmer.net.

STEP-2: This displays the Processes, Modules, Services, Files, Registry, RootKit / Malwares, Autostart, CMD of local host.

STEP-3: Select Processes menu and kill any unwanted process if any.

STEP-4: Modules menu displays the various system files like .sys, .dll

STEP-5: Services menu displays the complete services running with Autostart, Enable, Disable, System, Boot.

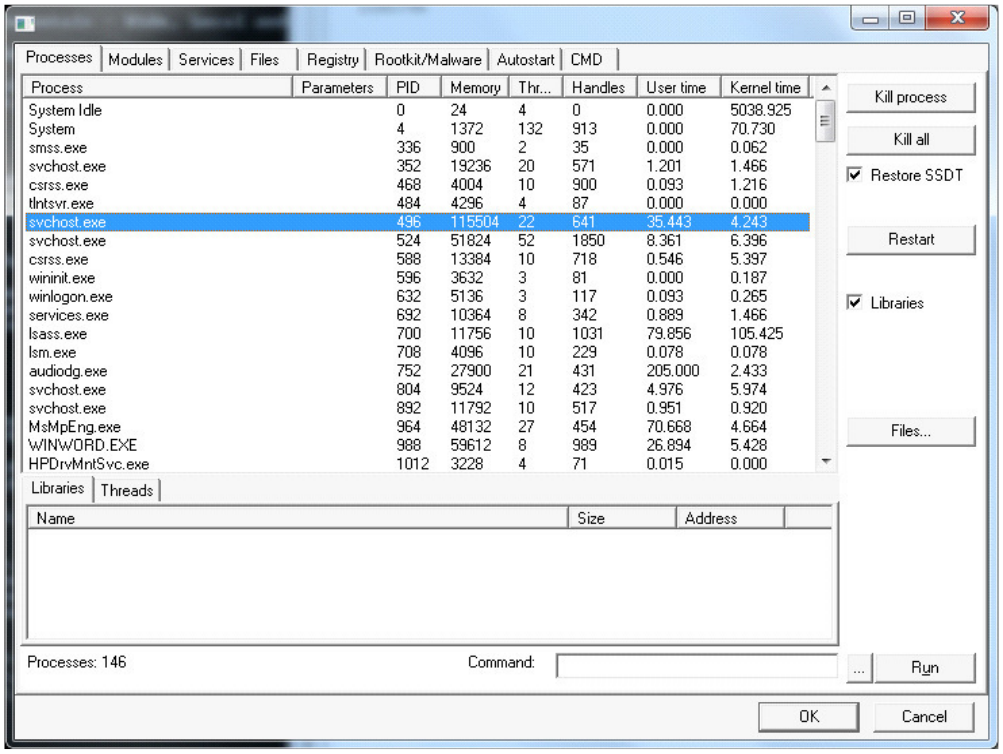
STEP-6: Files menu displays full files on Hard-Disk volumes.

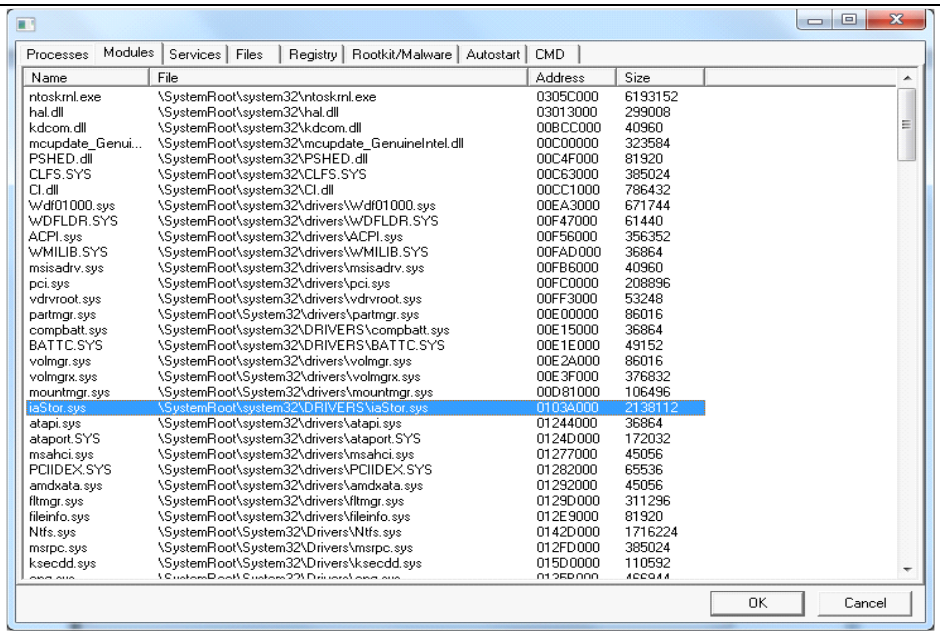
STEP-7: Registry displays Hkey_Current_user and Hkey_Local_Machine.

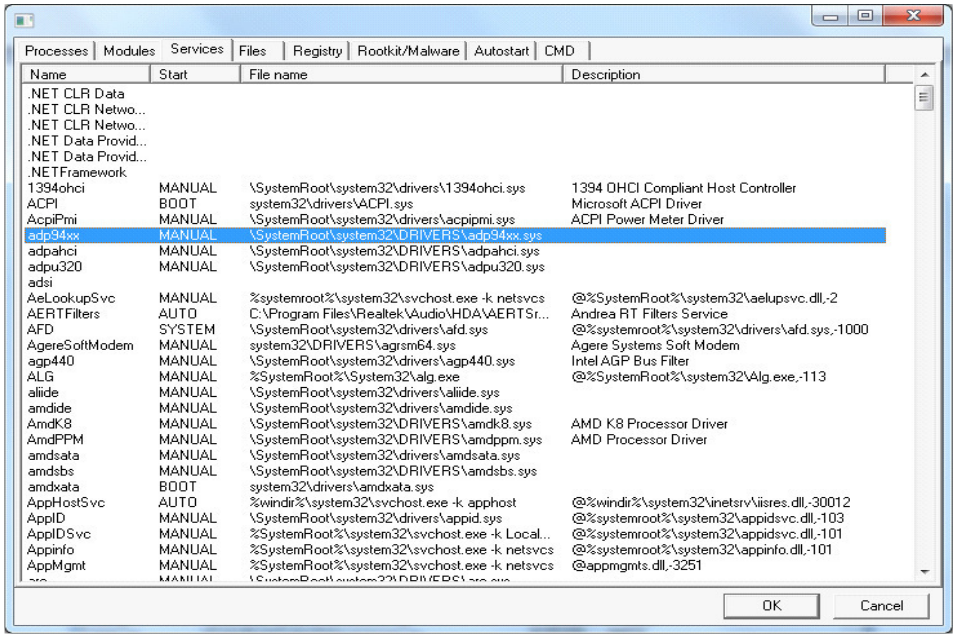
STEP-8: Rootkits / Malwares scans the local drives selected.

STEP-9: Autostart displays the registry base Autostart applications.

STEP-10:CMD allows the user to interact with command line utilities or Registry




Processes
Modules
Services
Files
Registry
Rootkit/Malware
Autostart
CMD
Name
File
Address
Size
ntoskrnl.exe \SystemRoot\system32\ntoskrnl.exe 0305C000 6193152
hal.dll \SystemRoot\system32\hal.dll 03013000 299008
kdcom.dll \SystemRoot\system32\kdcom.dll 00BCC000 40960
mcupdate_Genui... \SystemRoot\system32\mcupdate_GenuineIntel.dll 00C00000 323584
PSHED.dll \SystemRoot\system32\PSHED.dll 00C4F000 81920
CLFS.SYS \SystemRoot\system32\CLFS.SYS 00C63000 385024
CI.dll \SystemRoot\system32\CI.dll 00CC1000 786432
Wdf01000.sys \SystemRoot\system32\drivers\Wdf01000.sys 00EA3000 671744
WDFLDR.SYS \SystemRoot\system32\drivers\WDFLDR.SYS 00F47000 61440
ACPI.sys \SystemRoot\system32\drivers\ACPI.sys 00F56000 356352
WMILIB.SYS \SystemRoot\system32\drivers\WMILIB.SYS 00FAD000 36864
msisadrv.sys \SystemRoot\system32\drivers\msisadrv.sys 00FB6000 40960
pci.sys \SystemRoot\system32\drivers\pci.sys 00FC0000 208896
vdrvroot.sys \SystemRoot\system32\drivers\vdrvroot.sys 00FF3000 53248
partmgr.sys \SystemRoot\System32\drivers\partmgr.sys 00E00000 86016
compbatt.sys \SystemRoot\system32\DRIVERS\compbatt.sys 00E15000 36864
BATTC.SYS \SystemRoot\system32\DRIVERS\BATTC.SYS 00E1E000 49152
volmgr.sys \SystemRoot\system32\drivers\volmgr.sys 00E2A000 86016
volmgrx.sys \SystemRoot\System32\drivers\volmgrx.sys 00E3F000 376832
mountmgr.sys \SystemRoot\System32\drivers\mountmgr.sys 00D81000 106496
iaStor.sys \SystemRoot\system32\DRIVERS\iaStor.sys 0103A000 2138112
atapi.sys \SystemRoot\system32\drivers\atapi.sys 01244000 36864
ataport.SYS \SystemRoot\system32\drivers\ataport.SYS 0124D000 172032
msahci.sys \SystemRoot\system32\drivers\msahci.sys 01277000 45056
PCIIDEX.SYS \SystemRoot\system32\drivers\PCIIDEX.SYS 01282000 65536
amdxata.sys \SystemRoot\system32\drivers\amdxata.sys 01292000 45056
fltmgr.sys \SystemRoot\system32\drivers\fltmgr.sys 0129D000 311296
fileinfo.sys \SystemRoot\system32\drivers\fileinfo.sys 012E9000 81920
Ntfs.sys \SystemRoot\System32\Drivers\Ntfs.sys 0142D000 1716224
msrpc.sys \SystemRoot\System32\Drivers\msrpc.sys 012FD000 385024
ksecdd.sys \SystemRoot\System32\Drivers\ksecdd.sys 015D0000 110592
OK
Cancel


Processes
Modules
Services
Files
Registry
Rootkit/Malware
Autostart
CMD
Name
Start
File name
Description
.NET CLR Data
.NET CLR Netwo...
.NET CLR Netwo...
.NET Data Provid...
.NET Data Provid...
.NETFramework
1394ohci MANUAL \SystemRoot\system32\drivers\1394ohci.sys 1394 OHCI Compliant Host Controller
ACPI BOOT system32\drivers\ACPI.sys Microsoft ACPI Driver
AcpiPmi MANUAL \SystemRoot\system32\drivers\acpipmi.sys ACPI Power Meter Driver
adp94xx MANUAL \SystemRoot\system32\DRIVERS\adp94xx.sys
adpahci MANUAL \SystemRoot\system32\DRIVERS\adpahci.sys
adpu320 MANUAL \SystemRoot\system32\DRIVERS\adpu320.sys
adsi
AeLookupSvc MANUAL %systemroot%\system32\svchost.exe -k netsvcs @%SystemRoot%\system32\aelupsvc.dll,-2
AERTFilters AUTO C:\Program Files\Realtek\Audio\HDA\AERTSr... Andrea RT Filters Service
AFD SYSTEM \SystemRoot\system32\drivers\afd.sys @%systemroot%\system32\drivers\afd.sys,-1000
AgereSoftModem MANUAL system32\DRIVERS\agrsm64.sys Agere Systems Soft Modem
agp440 MANUAL \SystemRoot\system32\drivers\agp440.sys Intel AGP Bus Filter
ALG MANUAL %SystemRoot%\System32\alg.exe @%SystemRoot%\system32\Alg.exe,-113
aliide MANUAL \SystemRoot\system32\drivers\aliide.sys
amdide MANUAL \SystemRoot\system32\drivers\amdide.sys
AmdK8 MANUAL \SystemRoot\system32\DRIVERS\amdk8.sys AMD K8 Processor Driver
AmdPPM MANUAL \SystemRoot\system32\DRIVERS\amdppm.sys AMD Processor Driver
amdsata MANUAL \SystemRoot\system32\drivers\amdsata.sys
amdsbs MANUAL \SystemRoot\system32\DRIVERS\amdsbs.sys
amdxata BOOT system32\drivers\amdxata.sys
AppHostSvc AUTO %windir%\system32\svchost.exe -k apphost @%windir%\system32\inetsrv\iisres.dll,-30012
AppID MANUAL \SystemRoot\system32\drivers\appid.sys @%systemroot%\system32\appidsvc.dll,-103
AppIDSvc MANUAL %SystemRoot%\system32\svchost.exe -k Local... @%systemroot%\system32\appidsvc.dll,-101
Appinfo MANUAL %SystemRoot%\system32\svchost.exe -k netsvcs @%systemroot%\system32\appinfo.dll,-101
AppMgmt MANUAL %SystemRoot%\system32\svchost.exe -k netsvcs @appmgmts.dll,-3251
OK
Cancel

Experiment 14: An Intrusion detection system (SNORT)

An intrusion detection system is a device, or software application that monitors a network or systems for malicious activity or policy violations. SNORT is a widely IDS/IPS. Snort runs in 3 different modes: IDS mode, Logging mode and Sniffer mode

Snort Installation: visit the website www.snort .org. Follow the below steps:

1. Sudo apt install snort
2. Snort -V
3. we need to configure our HOME_NET value.
4. ifconfig
5. sudo gedit /etc/snort/snort.conf
6. Still snort have no rules
7. Verify by using the following commands
8. sudo snort -T -i wifi0 -c /etc/snort/snort.conf
9. snort -T -i wifi0 -c C:\Snort\etc\snort.conf
10. Here we are telling Snort to test (**-T**) the configuration file (**-c** points to its location) on the **wifi0** interface (enter your interface value if it's different).

```
4150 Snort rules read
    3476 detection rules
    0 decoder rules
    0 preprocessor rules
3476 Option Chains linked into 290 Chain Headers
0 Dynamic rules
+++++++++++++++++++++++++++++++++++++++++++++++++++
```

Rule Header

- alert – Rule action. Snort will generate an alert when the set condition is met.
- any – Source IP. Snort will look at all sources.
- any – Source port. Snort will look at all ports.
- -> – Direction. From source to destination.
- $HOME_NET – Destination IP. We are using the HOME_NET value from the snort.conf file.
- any – Destination port. Snort will look at all ports on the protected network.

Rule Options

- msg:"ICMP test" – Snort will include this message with the alert.
- sid:1000001 – Snort rule ID. Remember all numbers < 1,000,000 are reserved, this is why we are starting with 1000001
- rev:1 – Revision number. This option allows for easier rule maintenance.
- classtype:icmp-event – Categorizes the rule as an "icmp-event", one of the predefined Snort categories. This option helps with rule organization.
- **Again execute the test following command:**

```
++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
451 Snort rules read
    7 detection rules
    153 decoder rules
    291 preprocessor rules
451 Option Chains linked into 5 Chain Headers
++++++++++++++++++++++++++++++++++++++++++++++++++
```

IDS Mode

- Display alerts on the console.
- sudo snort -A console -q -c /etc/snort/snort.conf -i eht0
- **snort -i 3 -c C:\Snort\etc\snort.conf -A console**
- -c to point the configuration file.
- Specifying the interface (-i).
- The -A console option prints alerts to standard output.
- -q for the quit
- Ping using the other command prompt.
- You should see alerts generated for every ICMP Echo request and Echo reply message, with the message text we specified in the **msg** option

Experiment 15: ARP Poisoning

Address Resolution Protocol (ARP) is used to convert IP address to physical address. The host sends an ARP broadcast on the network, and the recipient computer responds with its MAC address. The resolved IP/MAC address is then used to communicate. ARP poisoning is sending fake MAC addresses to the switch so that it can associate the fake MAC addresses with the IP address of a computer on a network and hijack the traffic.

ARP Poisoning:

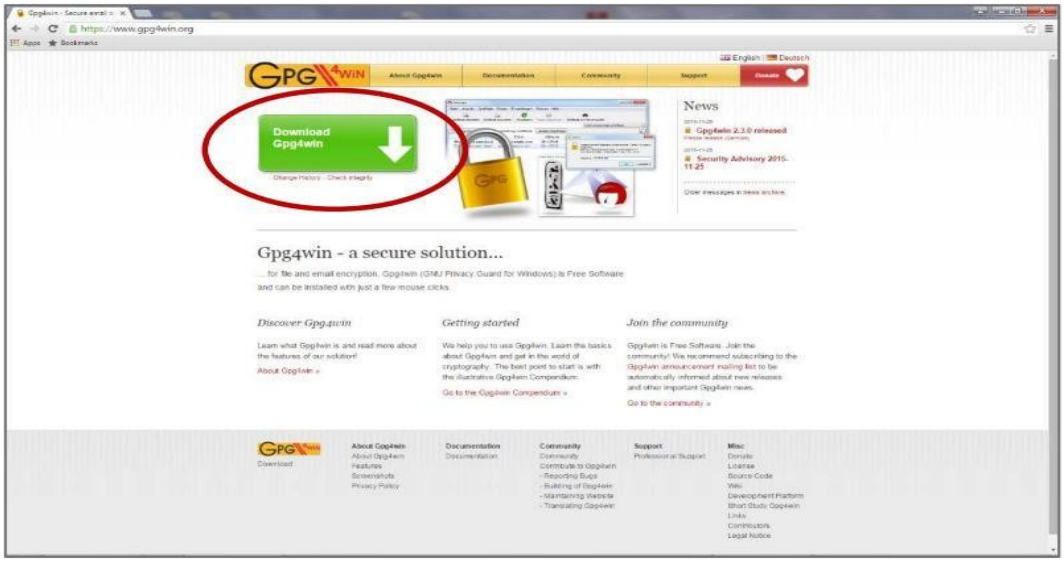
- ARP is used to convert IP address to physical address.
- The host sends an ARP broadcast on the network, and the recipient computer responds with its MAC address.
- The resolved IP/MAC address is then used to communicate.
- ARP poisoning is sending fake MAC addresses to the switch so that it can associate the fake MAC addresses with the IP address of a computer on a network and hijack the traffic.
- Open the command prompt and enter the following command
- Arp –a
- **-a** is the parameter to display to contents of the ARP cache.
- dynamic entries are added and deleted automatically when using TCP/IP sessions with remote computers.
- Static entries are added manually and are deleted when the computer is restarted, and the network interface card restarted or other activities that affect it.
- ipconfig /all command to get the IP and MAC address
- arp –s 192.168.0.56 74-86-----
- arp –a
- arp –d 192.168.0.56 // To delete the entry
- Download ettercap for windows

Experiment 16: secure data storage, secure data transmission and for creating digital signatures (GNUPG)

INSTALLING THE SOFTW ARE:

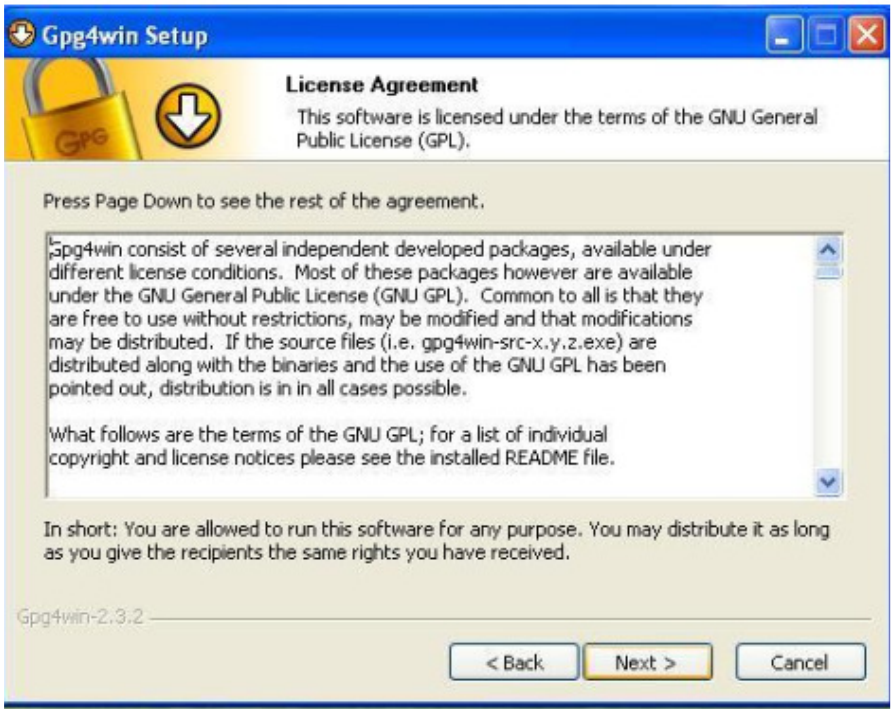
1. Visit www.gpg4win.org. Click on the "Gpg4win 2.3.0" button

2. On the following screen, click the "Download Gpg4win" button.



3. When the "Welcome" screen is displayed, click the "Next" button

4. When the "License Agreement" page is displayed, click the "Next" button



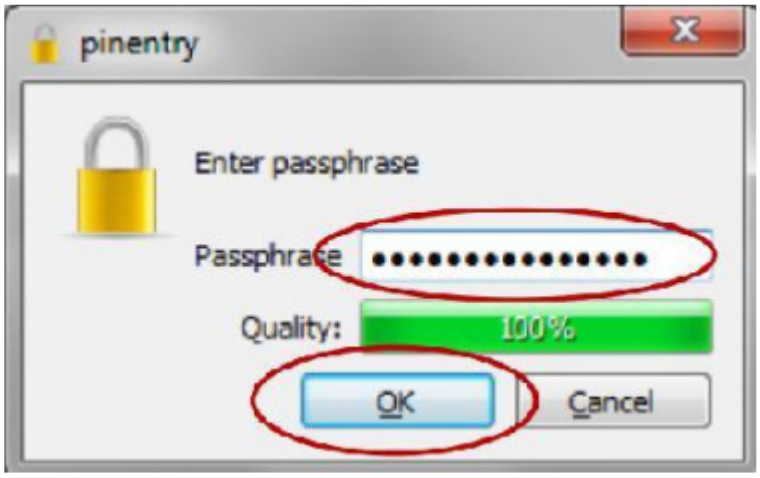
5. Set the check box values as specified below, then click the "Next" button

6. Set the location where you want the software to be installed. The default location is fine. Then, click the "Next" button.

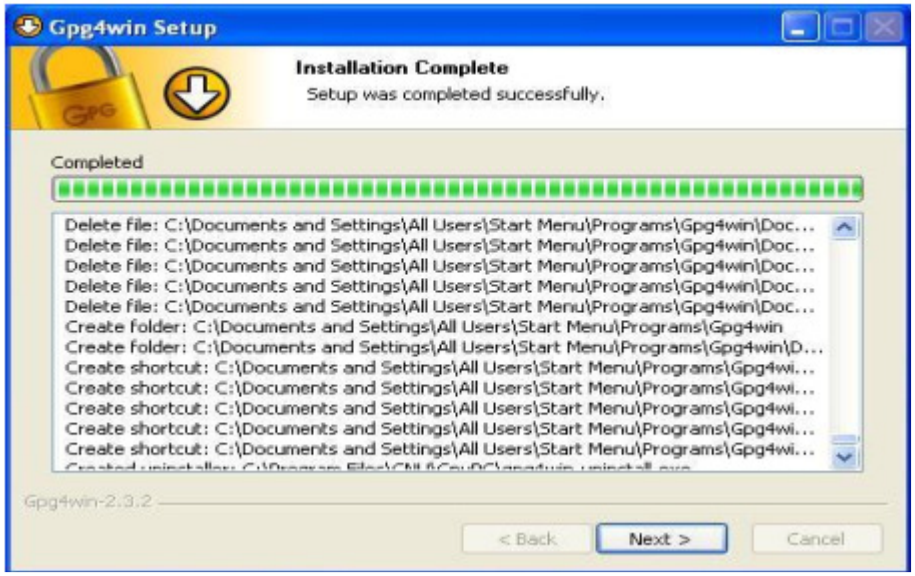
7. Specify where you want shortcuts to the software placed, then click the "Next" button

8. If you selected to have a GPG shortcut in your Start Menu, specify the folder in which it will be placed. The default "Gpg4win" is OK. Click the "Install" button to continue

9. A warning will be displayed if you have Outlook or Explorer opened. If this occurs, click the "OK" button.



10. The installation process will tell you when it is complete. Click the "Next" button



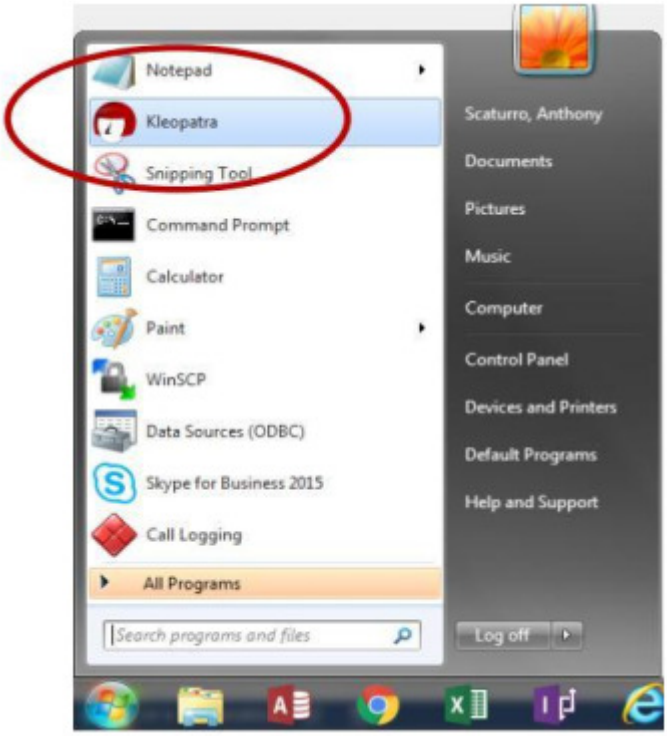
11. Once the Gpg4win setup wizard is complete, the following screen will be displayed. Click the "Finish" button

12. If you do not uncheck the "Show the README file" check box, the README file will be displayed. The window can be closed after you've reviewed it

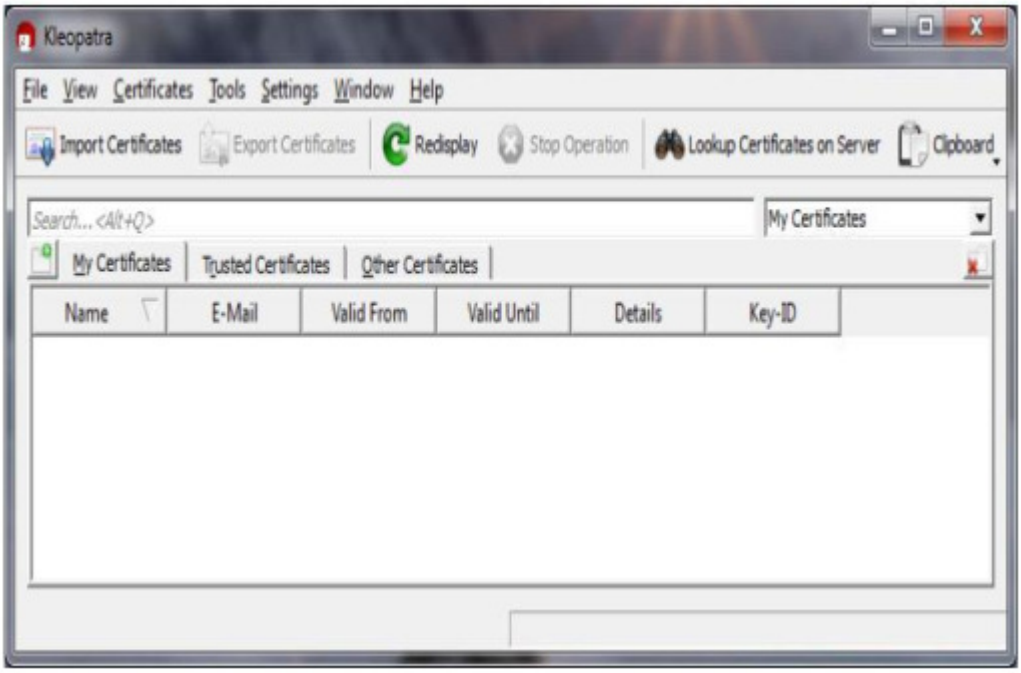
CREATING YOUR PUBLIC AND PRIVATE KEYS

GPG encryption and decryption is based upon the keys of the person who will be receiving the encrypted file or message. Any individual who wants to send the person an encrypted file or message must possess the recipient's public key certificate to encrypt the message. The recipient must have the associated private key, which is different than the public key, to be able to decrypt the file. The public and private key pair for an individual is usually generated by the individual on his or her computer using the installed GPG program, called "Kleopatra" and the following procedure:

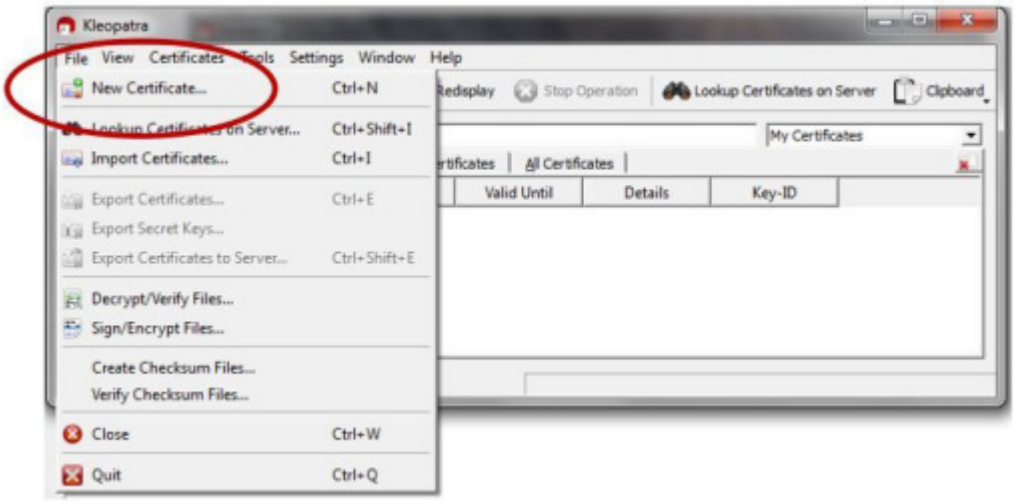
1. From your start bar, select the "Kleopatra" icon to start the Kleopatra certificate management software



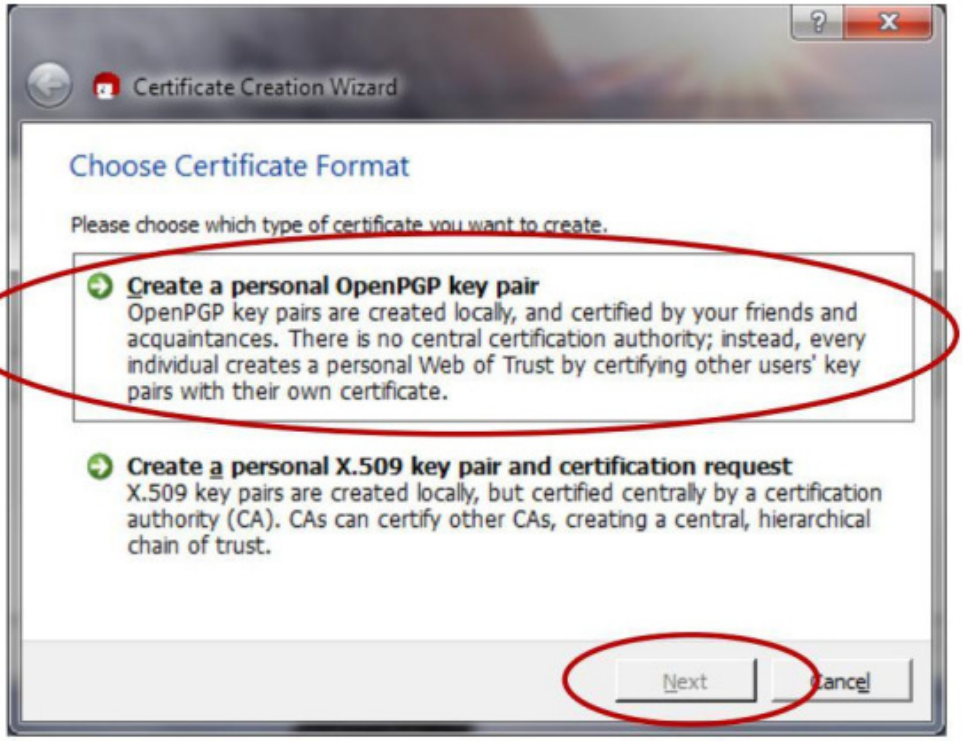
2. The following screen will be displayed



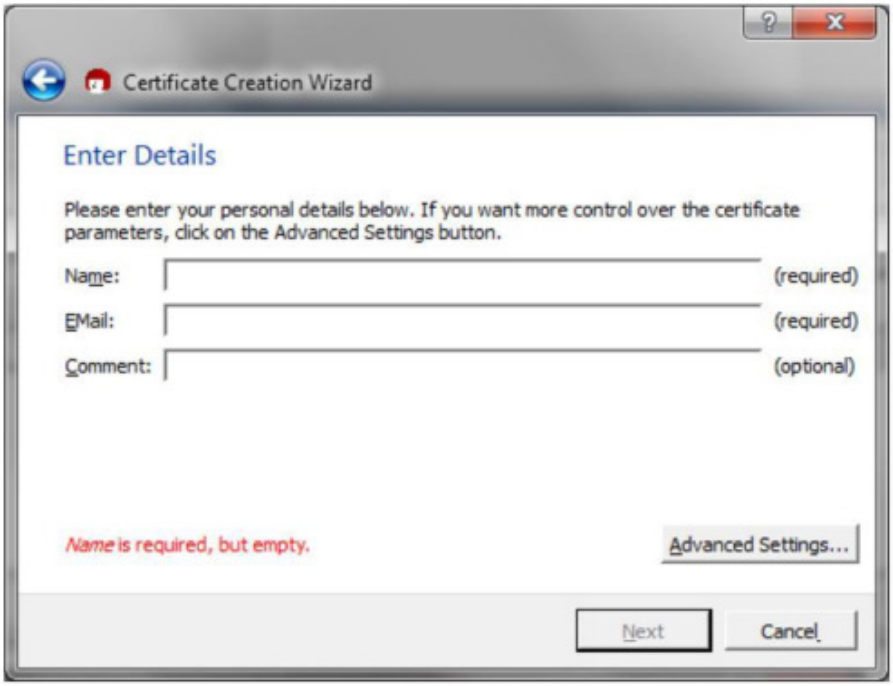
3. From the "File" dropdown, click on the "New Certificate" option



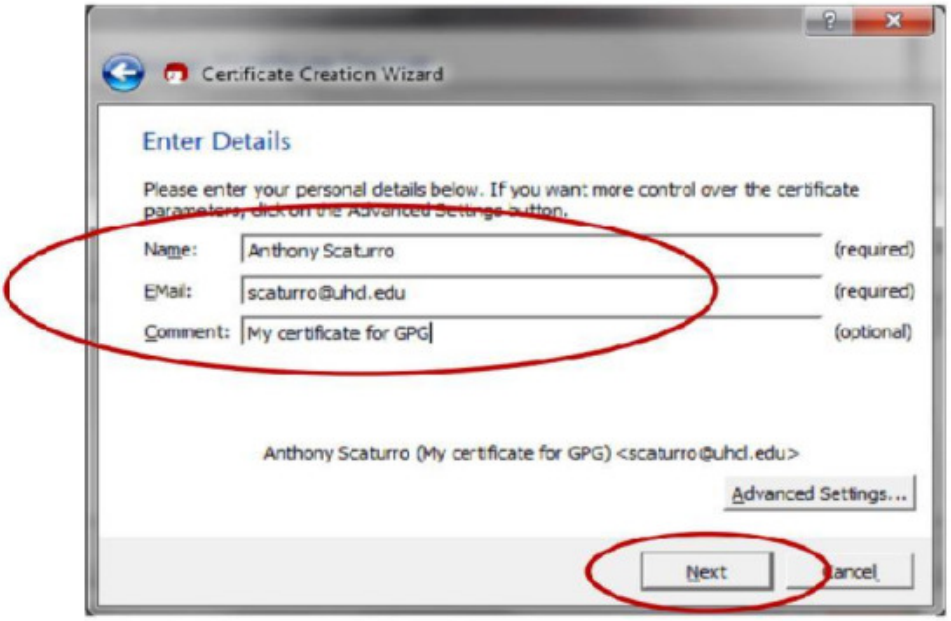
4. The following screen will be displayed. Click on "Create a personal OpenGPG key pair" and the "Next" button



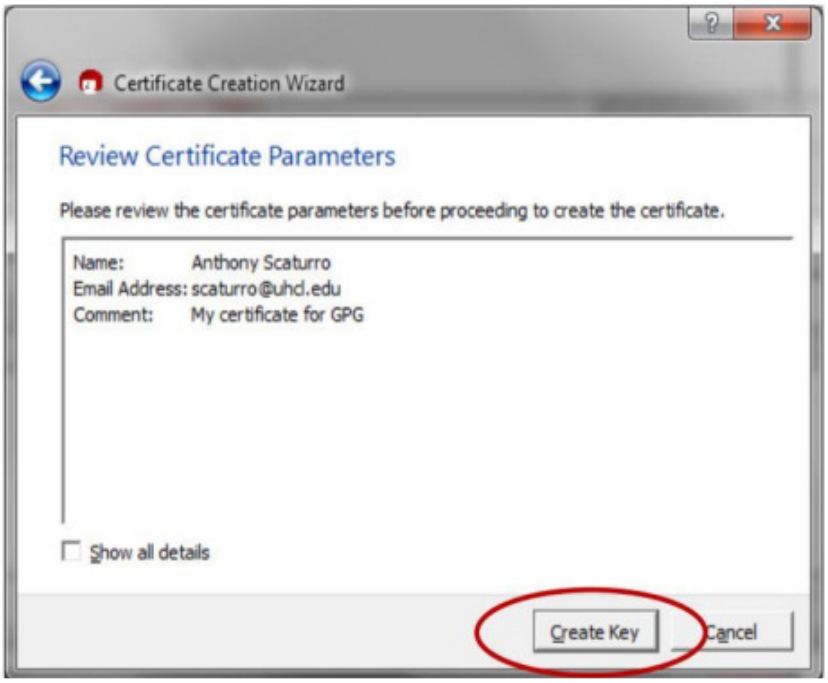
5. The Certificate Creation Wizard will start and display the following:



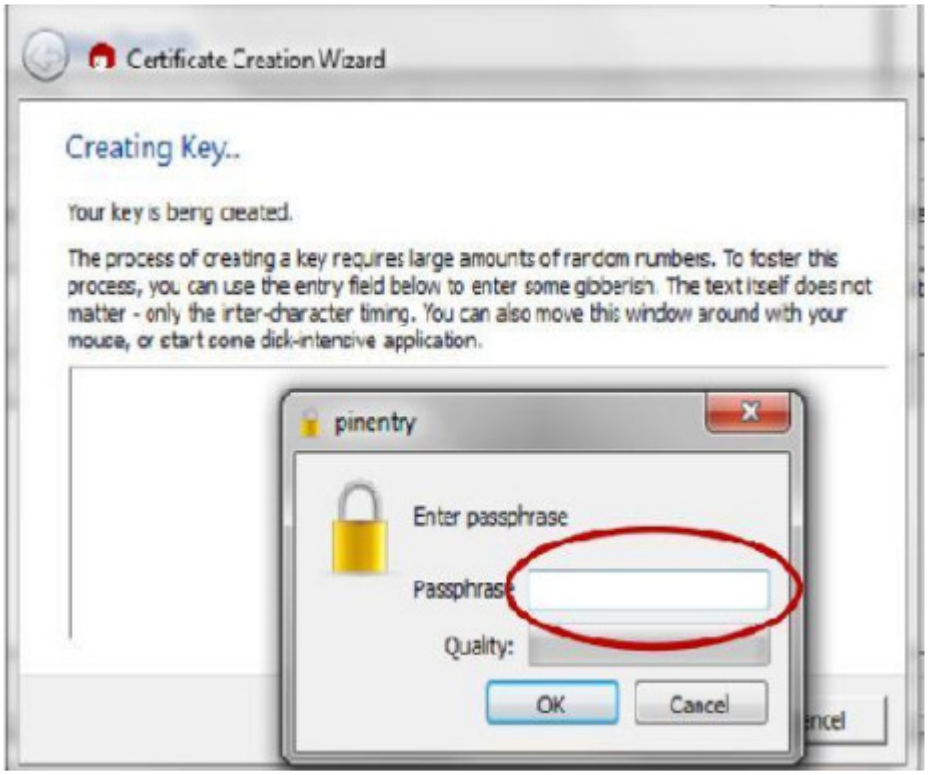
6. Enter your name and e-mail address. You may also enter an optional comment. Then, click the "Next" button



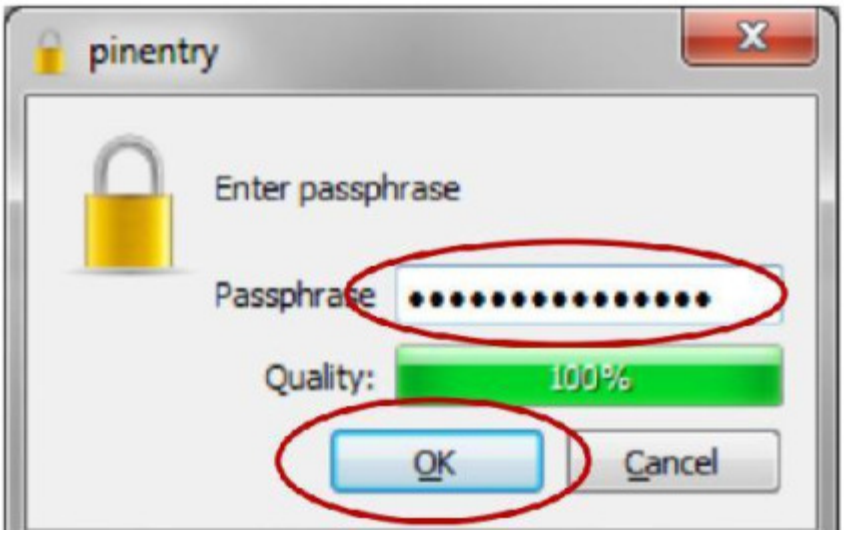
7. Review your entered values. If OK, click the "Create Key" button



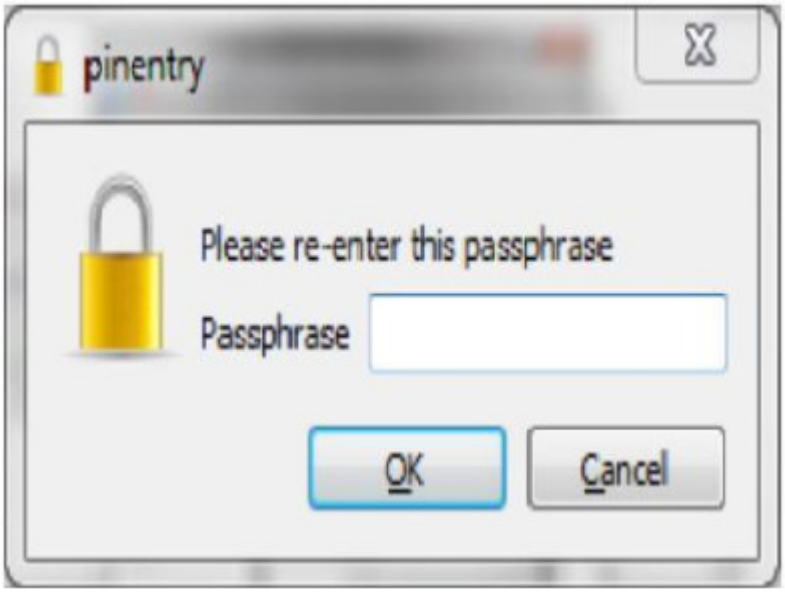
8. You will be asked to enter a passphrase



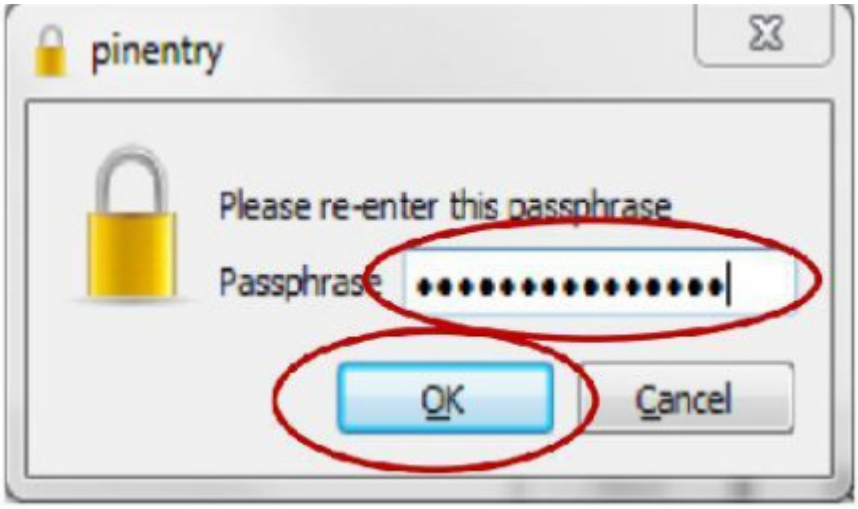
9. The passphrase should follow strong password standards. After you've entered your passphrase, click the "OK" button.



10. You will be asked to re-enter the passphrase

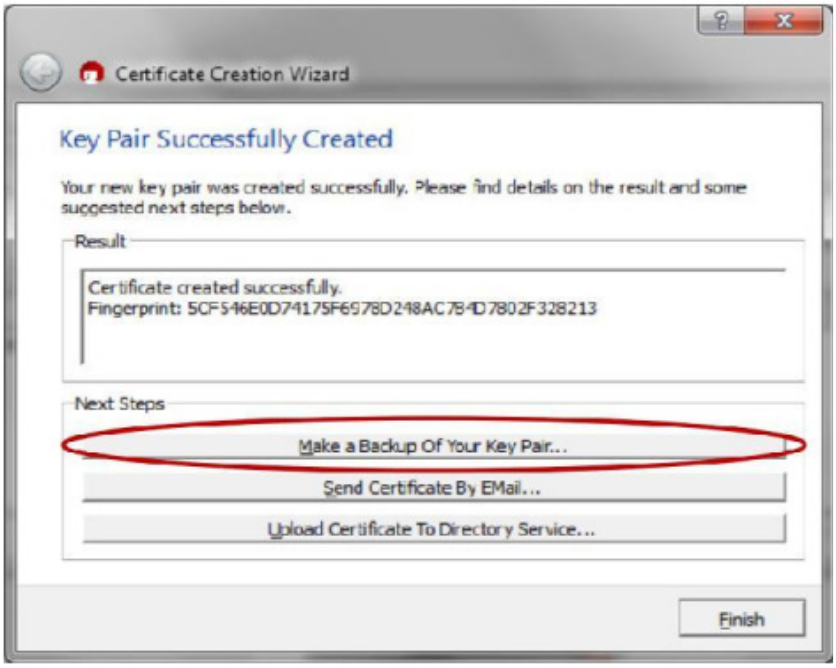


11. Re-enter the passphrase value. Then click the "OK" button. If the passphrases match, the certificate will be created

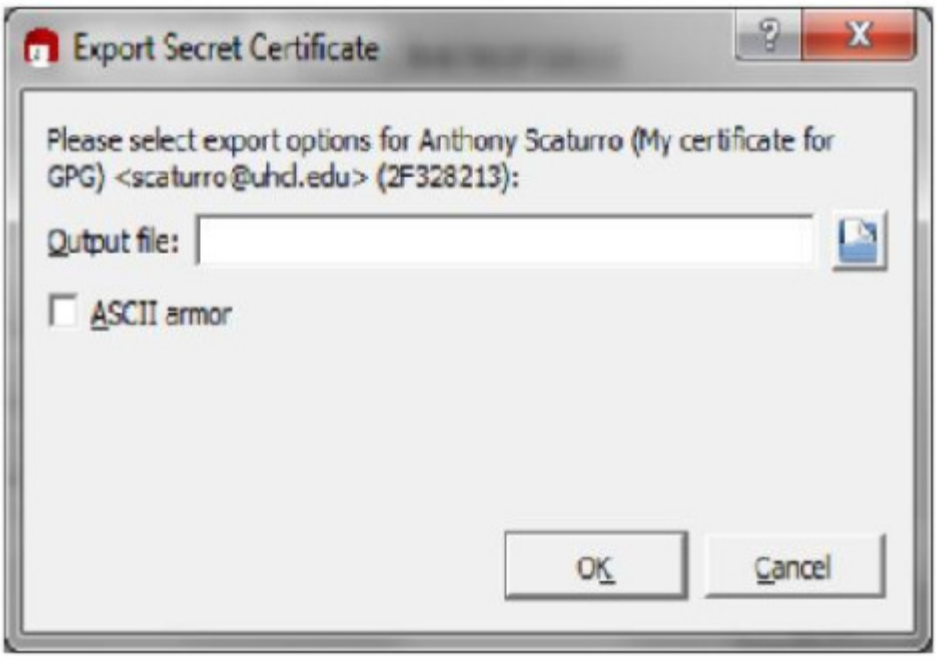


12. Once the certificate is created, the following screen will be displayed. You can save a backup of your public and private keys by clicking the "Make a backup Of Your Key Pair" button. This backup can be used to copy certificates onto other authorized

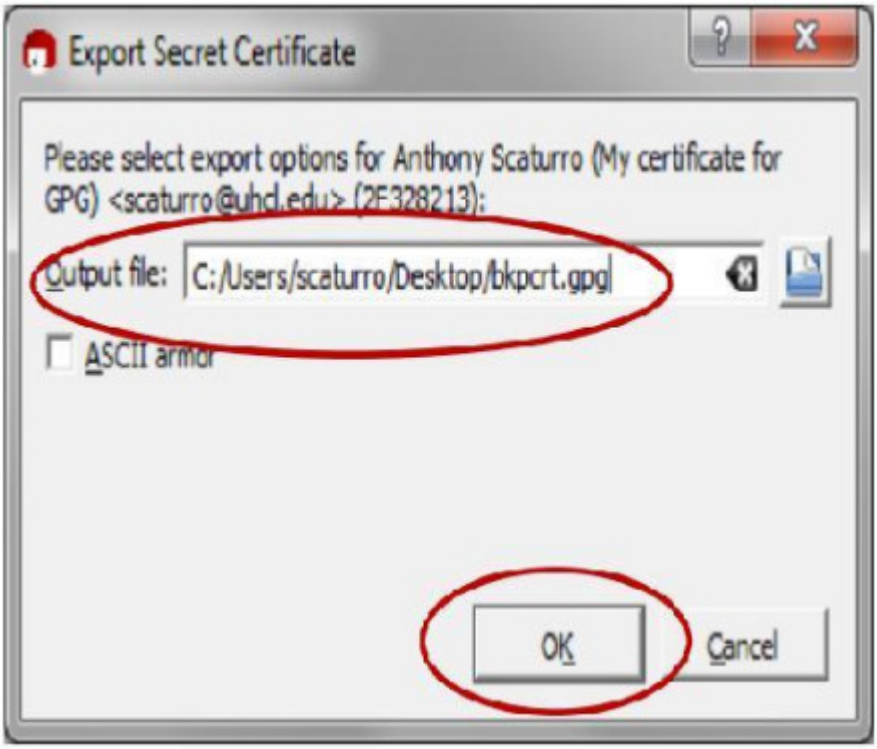
computers.



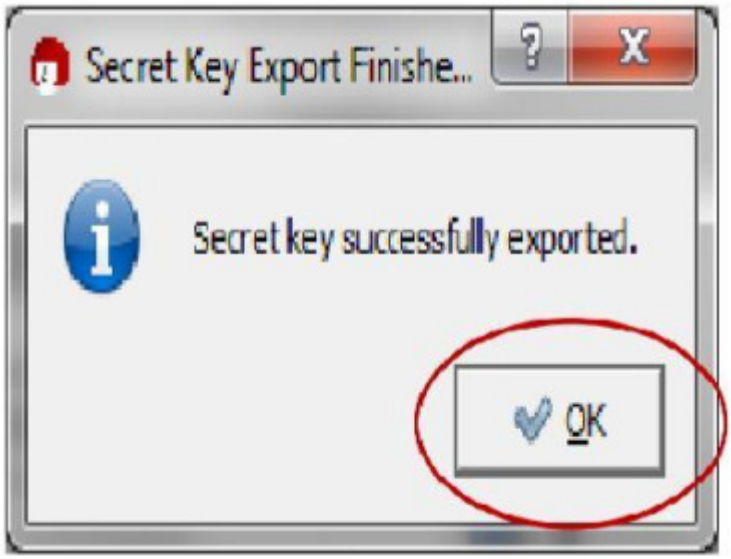
13. If you choose to backup your key pair, you will be presented with the following screen:



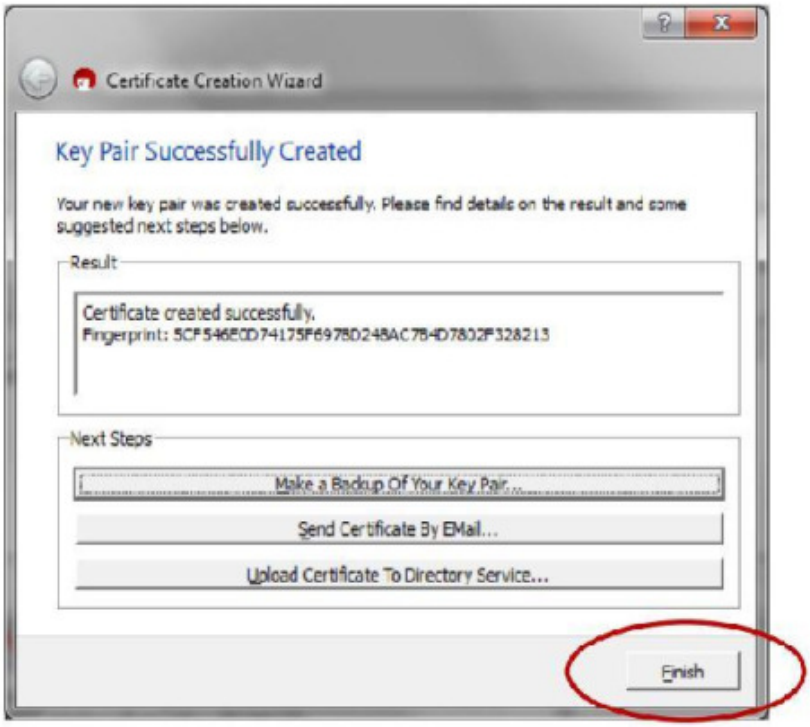
14. Specify the folder and name the file. Then click the "OK" button.



15. After the key is exported, the following will be displayed. Click the "OK" button



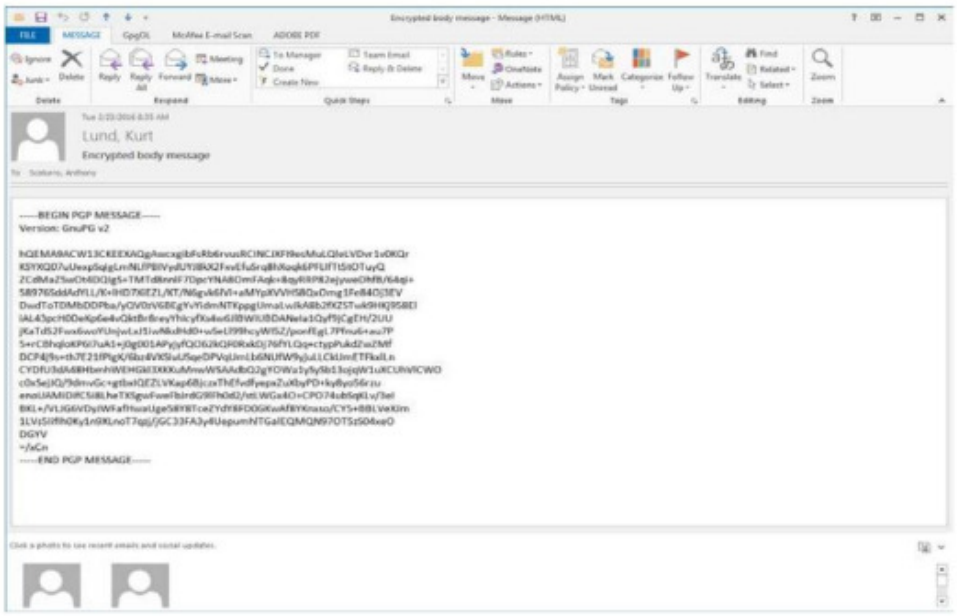
16. You will be returned to the "Key Pair Successfully Created" screen. Click the "Finish" button.



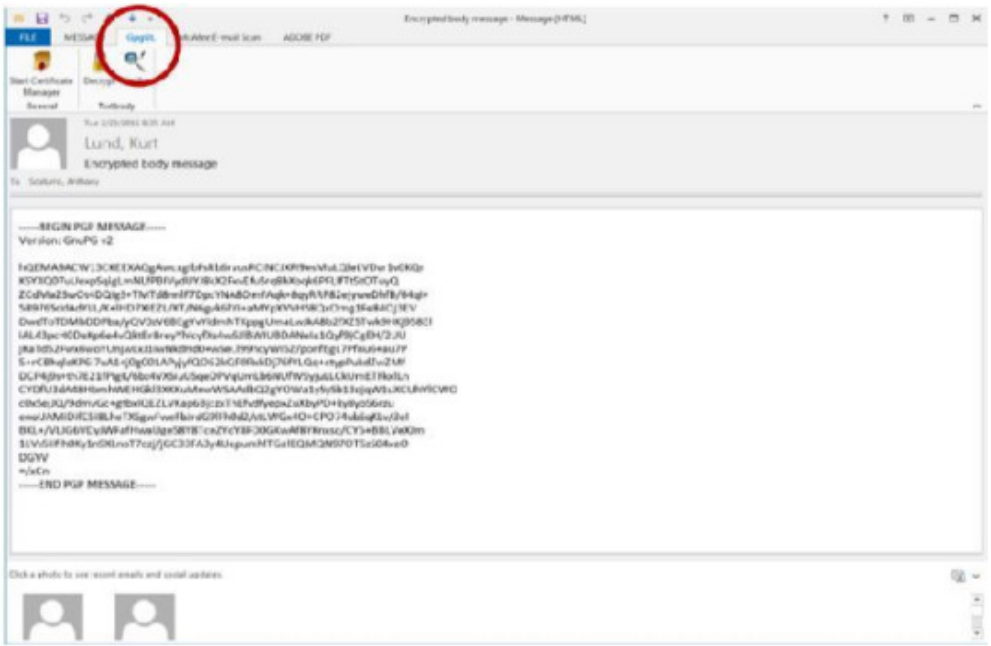
17. Before the program closes, you will need to confirm that you want to close the program by clicking on the "Quit Kleopatra" button

DECRYPTING AN ENCRYPTED E-MAIL THAT HAS BEEN SENT TO YOU:

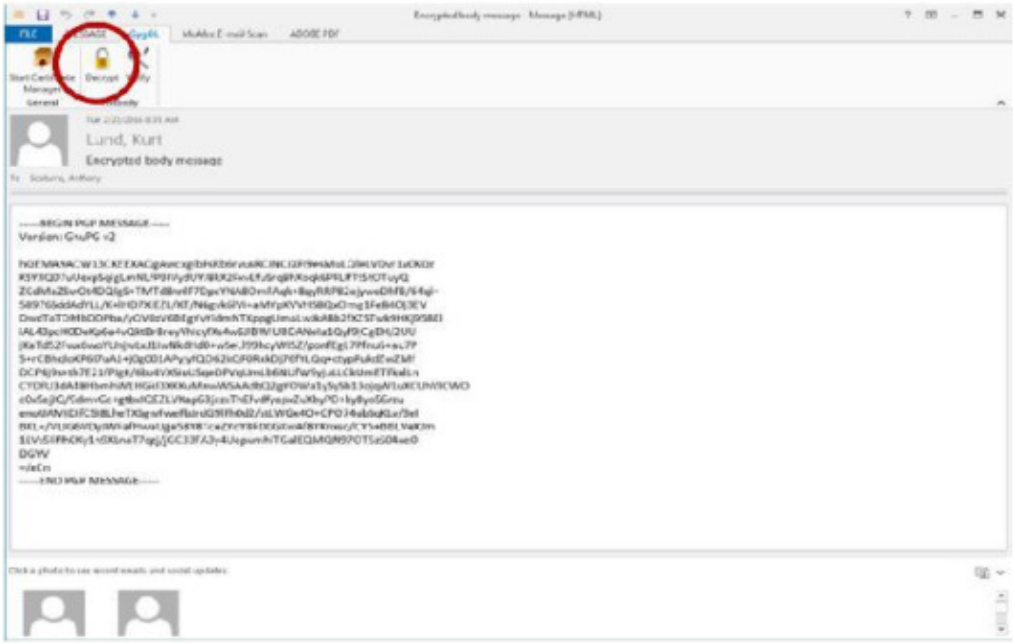
1. Open the e-mail message



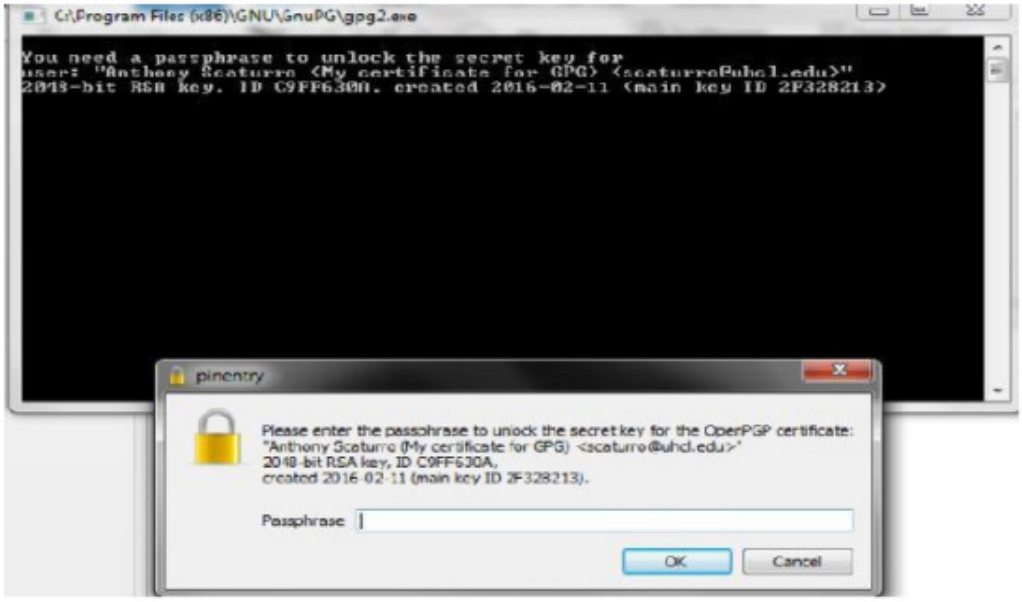
2. Select the GpgOL tab



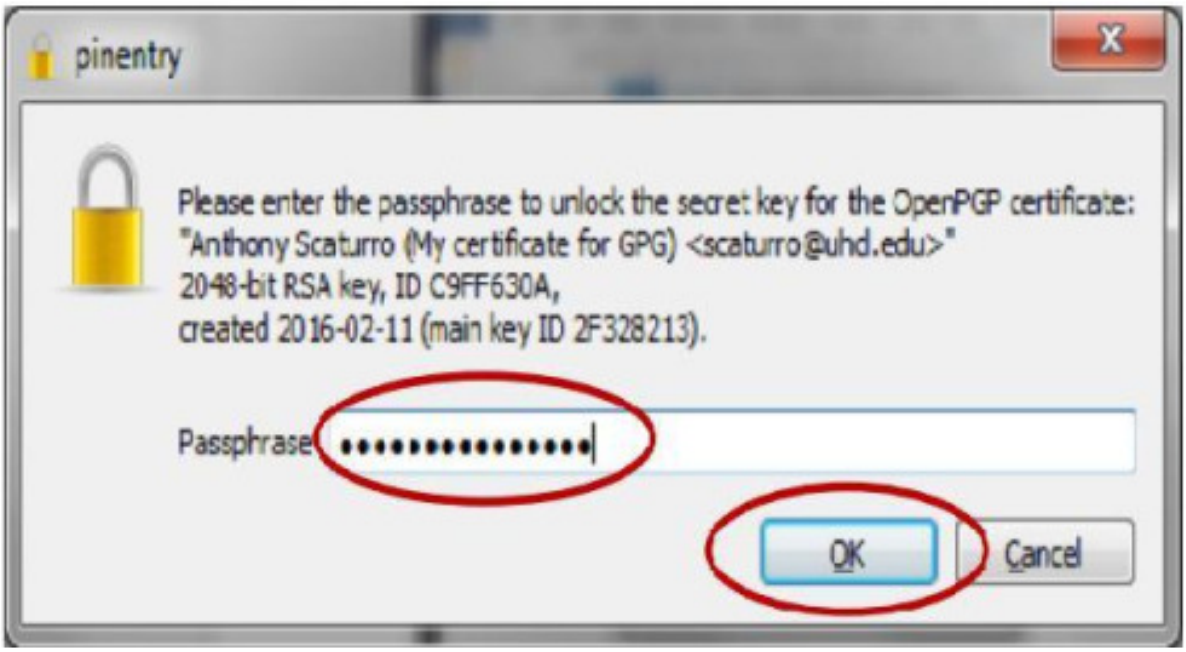
3. Click the "Decrypt" button



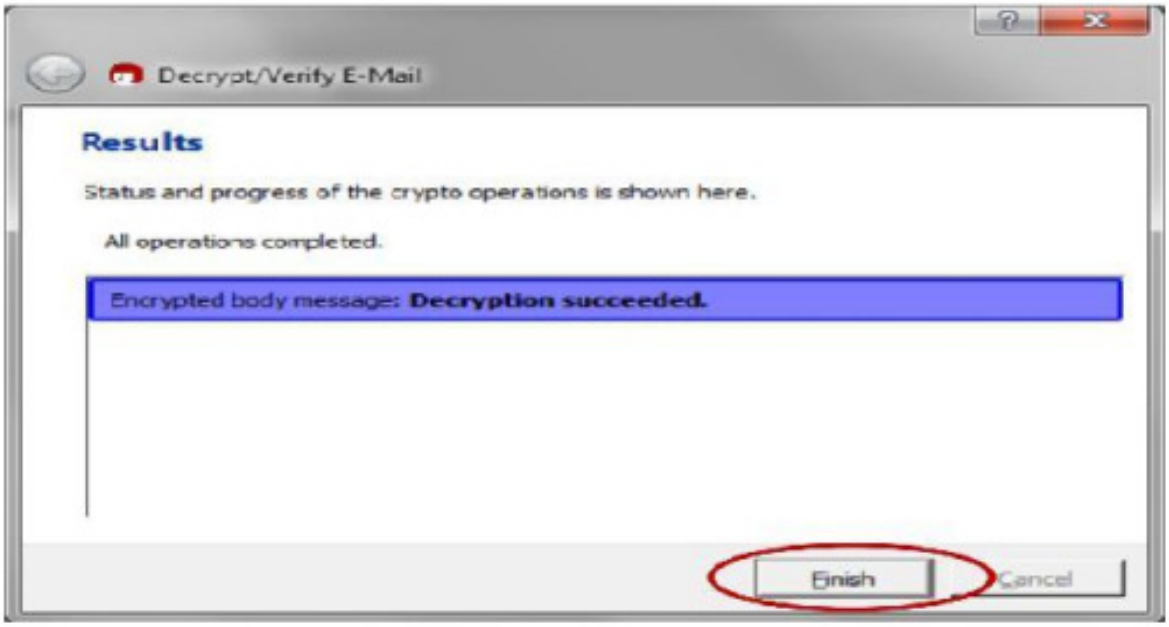
4. A command window will open along with a window that asks for the Passphrase to your private key that will be used to decrypt the incoming message



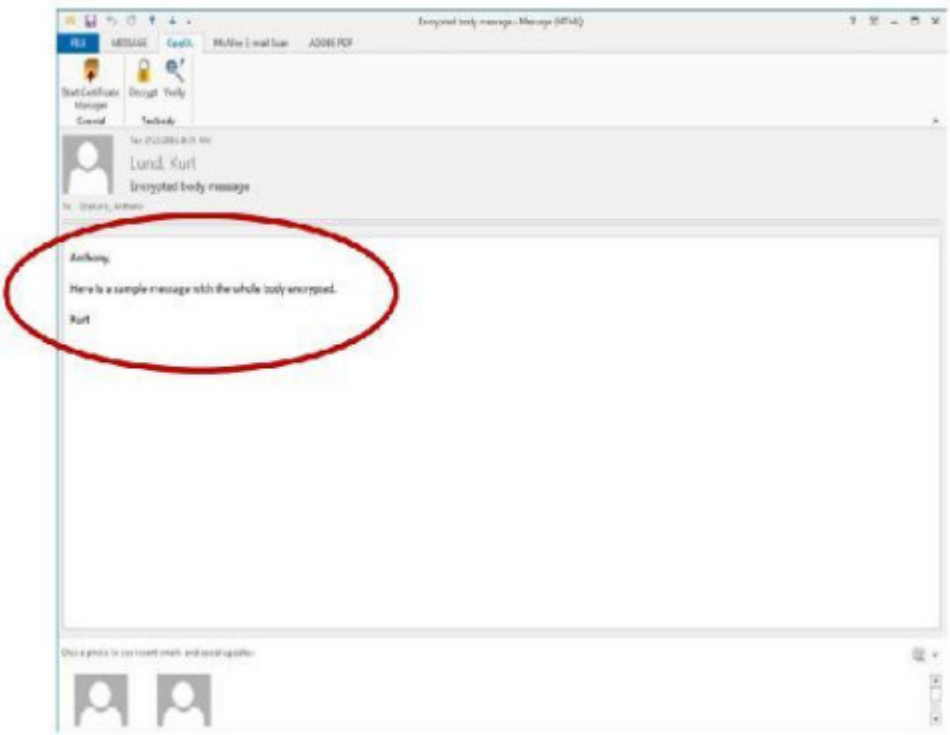
5. Enter your passphrase and click the “OK” butto



6. The results window will tell you if the decryption succeeded. Click the “Finish” button top close the window



7. Your unencrypted e-mail message body will be displayed



8. When you close the e-mail you will be asked if you want to save the e-mail message in its unencrypted form. For maximum security, click the "No" button. This will keep the message encrypted within the e-mail system and will require you to enter your passphrase each time you reopen the e-mail message

Viva- Questions

1. **What is public-key cryptography?**

   It is also known as asymmetric cryptography. In this type of cryptography, a key pair consists of public key and private key is used to securely exchange the message between two parties. One is used to encrypt the message and other key is used to decrypt the message. The advantage of public-key cryptography is that you only need to keep your private key secure and distribute the public key.

2. **What is block cipher?**

   Block cipher is an algorithm that converts a block of plaintext into ciphertext. It takes two inputs, n bits of fixed length of blocks and a secret key and output are n bits of ciphertext.

   3. What is stream cipher? Name a most widely used stream cipher.

   Stream cipher is symmetric encryption algorithm with takes one bit or one byte as input and encrypted with a secret key called keystream. The keystream generator function produces keys K1, K2, K3 … which are XORed with plaintext P1, P2, P3 … to produce cipher text C1, C2, C3

3. **What are the differences among encoding, encryption, and hashing?**

   Encoding: Basically, encoding is used to protect the integrity of data as it crosses through communication network to keep its original message upon arriving. It is primarily an insecure function because it is easily reversible. Encryption: Encryption is basically designed for confidentiality and data integrity and reversible only if you have the appropriate key.
   Hashing: With hashing the operation is one-way i.e., non-reversible. It takes an input (or message) and returns a fixed-size string, which is called the hash value.

4. **What are Brute Force Attacks?**

   Brute forcing is a mechanism which is used by an attacker to break the encryption of data by applying a set of various keys. Cryptanalyst has a set of number of keys and apply them one by one to the encryption algorithm until he gets the right key.

5. **What is the difference between DSA and RSA?**

   DSA means Digital Signature Algorithm. The basic difference between DAS and RSA is RSA can encrypt and sign, however DSA is only used for digital signature.

6. **What are Digital Signatures?**

Digital signature is an attachment to an electronic message used for security purpose. It is used to verify the authenticity of the sender.

7. **What is message authentication?**

   When the verifier validates the digital signature using public key of a sender, he is assured that signature has been created only by sender who possess the corresponding secret private key and no one else.

8. **What are the services provided by digital certificates?**

   Digital certificate provide authentication, nonrepudiation, and integrity services.

9. **What are Digital certificates?**

   Digital certificates are digital documents attesting to the binding of a public key to an individual or other entity. They allow verification of the claim that a given public key does in fact belong to a given individual. Certificates help prevent someone from using a phony key to impersonate someone else.

10. **What is Secure Sockets Layer (SSL)?**

    The Secure Sockets Layer (SSL) is a computer networking protocol that manages server authentication, client authentication and encrypted communication between servers and clients.

11. **What is the technology available to ensure data privacy and integrity during transmission?**

    1. Controlling Access within the Network
    2. Encrypting Data for Network Transmission
    3. Secure Sockets Layer (SSL) Protocol
    4. Firewall

12. **What are the services provided by digital certificates?**

    Digital certificate provide authentication, nonrepudiation, and integrity services

13. **What is Rootkit?**

    The term Rootkit originally referred to a collection of tools used to gain administrative access on UNIX operating systems. The collection of tools often included well-known system monitoring tools that were modified to hide the actions of an unauthorized user. An unauthorized user would replace the existing tools on the system with the modified versions preventing authorized users from discovering the security breach. Rootkit - “A tool used to protect backdoors and other tools

from detection by administrators"

14. What are called Rootkits in Windows?

They refers to programs that use system hooking or modification to hide files, processes, registry keys, and other objects to hide programs and behaviors. Windows rootkits do not necessarily include any functionality to gain administrative privileges. In fact, many Windows rootkits require administrative privileges to even function.

15. What are the basic classes of Rootkits?

Two basic classes of Windows rootkits: kernel mode rootkits & user mode rootkits.

16. Why Rootkits are used?

Rootkits are used by criminals for a variety of purposes, usually to turn a computer into part of a botnet, which can then, in turn, go on to infect other computers or send spam email messages. The rootkit owner can install key loggers to capture user-entered passwords for online banking and similar activities or steal the user's personal details to use for identity fraud.

17. How Rootkits stay undetected?

Many rootkits infect the boot sectors of the computer's hard disk, allowing them to load before the computers operating system. The rootkit then patches the operating system and changes common functions to hide its existence. For example, the root kit could intercept calls for a list of files in a directory, removing its own file names before showing the results to the user, so it would appear as if the directory is clean.

18. **What are the Rootkit capabilities?**

Current Rootkit Capabilities: Rootkits Hide processes, Hide files, Hide registry entries, Hide services, Completely bypass personal firewalls, Undetectable by antivirus, Remotely undetectable, Covert channels - undetectable on the network, Defeat cryptographic hash checking, Install silently, All capabilities ever used by viruses or worms.

19. **What is Intrusion Detection System (IDS)?**

With the development of network technologies and applications, network attacks are greatly increasing both in number and severity. As a key technique in network security domain, Intrusion Detection System (IDS) plays vital role of detecting various kinds of attacks and secures the networks. Main purpose of IDS is to find out intrusions among normal audit data

and this can be considered as classification problem.

20. **What are the activities of IDS?**

Intrusion detection systems (IDS) are an effective security technology, which can detect, prevent and possibly react to the attack. It performs monitoring of target sources of activities, such as audit and network traffic data in computer or network systems, requiring security measures, and employs various techniques for providing security services.

21. **What is intrusion or intruder?**

Intrusion: Attempting to break into or misuse your system. Intruders may be from outside the network or legitimate users of the network. Intrusion can be a physical, system or remote intrusion. Intrusion Detection Systems look for attack signatures, which are specific patterns that usually indicate malicious or suspicious intent.

22. **What is Snort?**

Snort is an open-source network intrusion prevention system, capable of performing real time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more.

23. **What are the uses of Snort?**

Snort has three primary uses: It can be used as a straight packet sniffer like tcpdump, a packet logger (useful for network traffic debugging, etc.), or as a full-blown network intrusion prevention system. The privacy of the Snort community is very important to source fire.

24. **What are the three modes of Snort?**

Snort can be configured to run in three modes: 1. Sniffer mode 2. Packet Logger mode 3. Network Intrusion Detection System Mode Sniffer mode: snort –v Print out the TCP/IP packets header on the screen. Packet Logger mode: snort –dev –l c:\log [create this directory in the C drive] and snort will automatically know to go into packet logger mode, it collects every packet it sees and places it in log directory. Network Intrusion Detection System mode: snort –d c:\log –h ipaddress/24 –c nort conf.