

Practical Workbook

Network & Information Security

(CT-460)



Name

Year

Batch

Roll No

Department

Department of Computer Science & Information Technology
NED University of Engineering & Technology

Practical Workbook

Network & Information Security

(CT-460)



Prepared By

Prof. Dr. Shariq Mahmood Khan
Professor, CS&IT

Approved By

Prof. Dr. Muhammad Mubashir Khan
Chairman, CS&IT

Department of Computer Science & Information Technology
NED University of Engineering & Technology

CONTENTS

Lab Session No.	Object	Signature
1	Implementation of Traditional Ciphers and the Attacks on Them	
2	Implementation of Polygraphic Substitution Ciphers (Hill Cipher) and the Attack on It	
3	Implementation of Polygraphic Substitution Ciphers (Playfair Cipher) and the Attack on It	
4	Analyze and find hidden information in Text or Data Files	
5	Hash Functions, Demonstration of MD5 and MD5 Hash Collision	
6	Folder Security Implementation Through BATCH Programming	
7	Data Encryption Standard (DES)	
8	Diffie Helman Key Exchange Method	
9	Network Traffic Processing and Analysis in promiscuous mode	
10	ARP Poisoning to conduct Man In The Middle (MITM) Attack	
11	NMAP - A Stealth Port Scanner	
12	RSA (Rivest–Shamir–Adleman)	
13	Attack on RSA encryption with short RSA modulus	
14	Introduction to Firewall & Intrusion Detection System	

Lab Session 01

Object: Implementation of Traditional Ciphers and the Attacks on Them

Theory:

Monoalphabetic Ciphers

A monoalphabetic cipher is one in which the plain text letters are mapped to cipher text characters using a single alphabetic key. The Caesar-shift cipher, in which each letter is shifted depending on a numeric key, and the atbash cipher, in which each letter is mapped to the letter symmetric to it around the center of the alphabet, are two examples of monoalphabetic ciphers.

In this lab, we will study the following monoalphabetic ciphers

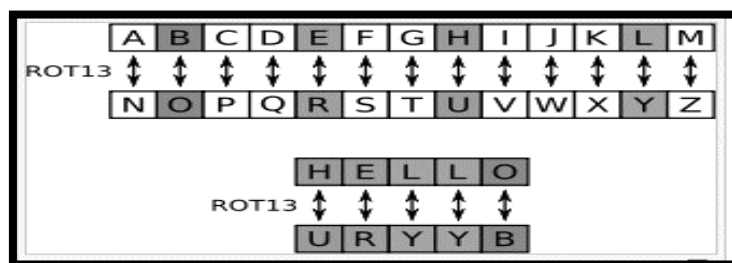
- Caesar Cipher
- Rot – 13
- Atbash Cipher

Caesar Cipher

A Caesar cipher, named after Julius Caesar, also known as Caesar's cipher, the shift cipher, Caesar's code, or Caesar shift, is one of the simplest and most extensively used encryption algorithms in cryptography. It is a substitution cipher in which each letter in the plaintext is replaced with a letter located a specified number of places down the alphabet. With a left shift of 3, for example, D becomes A, E becomes B, and so on.

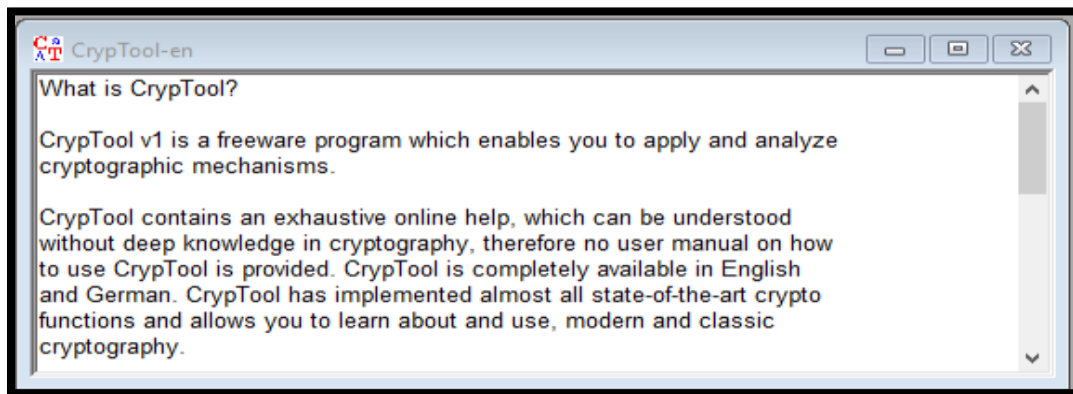
Rot -13

ROT13 ("rotate by 13 places," frequently hyphenated ROT-13) is a basic Caesar cipher used to obfuscate text by substituting each letter with the letter thirteen letters down the alphabet.



This lab provides an example illustrating the use of the Caesar encryption algorithm. To make it easier to follow the steps that need to be performed with CrypTool, the example is illustrated with a number of screenshots.

First of all, to acquaint ourselves with the Caesar encryption algorithm we will open a document, encrypt it and then decrypt it again. We will then try to get the computer to work out the key with which a plaintext is encrypted. Therefore, we open the file CrypTool-en.txt out of the directory CrypTool\examples via the menu File \ Open.




We will now encrypt this document using the Caesar encryption algorithm. To do this, we select the menu Encrypt/Decrypt \ Symmetric (classic) \ Caesar/Rot-13. Then the following which this dialog box appears. The key we enter is the letter F. Additionally, we change the options how to interpret the alphabet characters: The first alphabet character is set to 0.

Key Entry: Caesar / ROT-13

Description
 Here you can enter the key for the Caesar cipher.
 Caesar is a mono-alphabetic substitution, where the characters of the cleartext alphabet are mapped to the ciphertext alphabet by shifting. This shifting value is the key. You can enter the key as a number or as a single character of the alphabet.
 Rot-13 is a special variant, where the key has the fixed value of half the length of the cleartext alphabet. This variant is only selectable if the length of the alphabet is an even number.

Select variant
☒ Caesar
☐ Rot-13

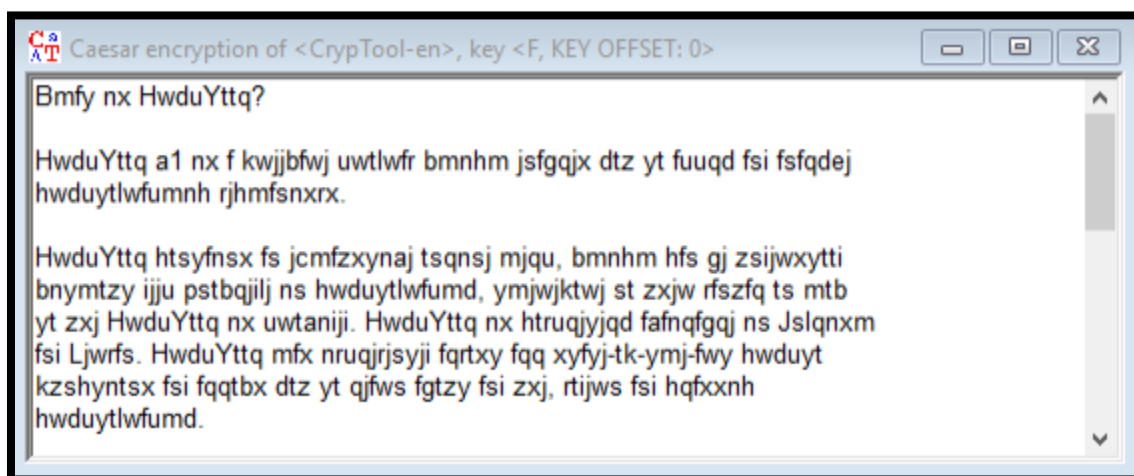
Options to interpret the alphabet characters
☒ Value of the first alphabet character = 0 (e.g. "A"=0)
☐ Value of the first alphabet character = 1 (e.g. "A"=1)

Key entry as:
☐ Alphabet character 
☒ Number value

Properties of the chosen encryption
 Shift of 5
 Mapping of the alphabet (26 characters)
 from: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 to: FGHIJKLMNOPQRSTUVWXYZABCDE

Encrypt Decrypt Text options Cancel

Clicking on the Encrypt button opens a new window that contains the encrypted text. On closer examination of the text it becomes apparent that the letters have been shifted by 5 position, so that the initial word, CrypTool, is now HwduYttq.



The plaintext version of this encrypted document can now be obtained by selecting Encrypt/Decrypt \ Classical \ Caesar again. In the dialog box which now appears we enter the key with which the document was encrypted (F). This time we do not want the text to be encrypted, but instead to be decrypted. Therefore, the Decrypt button must be selected.

Clicking on the Decrypt button tells CrypTool to go ahead and decrypt the text. The plaintext appears immediately. We have seen how a text is encrypted using the Caesar encryption algorithm and then decrypted again.

Under the classical Caesar encryption algorithm only the letters are encrypted. During encryption, the lower case letters were converted to upper case letters and encrypted, but all the other characters such as punctuation characters and formatting characters (blank characters and line breaks) were omitted.

In the classical encryption algorithms CrypTool retains the formatting by default. This can be disabled via the menu option Options \ Text Options. In the following dialog box the option Keep characters not present in the alphabet unchanged and the option Distinguish between uppercase and lowercase must be disabled.

We now return to the window containing the plaintext (by clicking on it with the mouse) and encrypt this document once again with the Caesar encryption algorithm, using the key F. From the results one can see immediately that the encrypted text has the same content as the previous version but this time it contains only upper case letters and the formatting has been removed. For example, the first characters in the encrypted text are once again HWDUYTTQ (the ciphertext of CrypTool).

We would like to save this document under a different name. Select File \ Save As to access the dialog box used for saving a document under a new name. As file name we enter CrypTool.Caesar.txt.

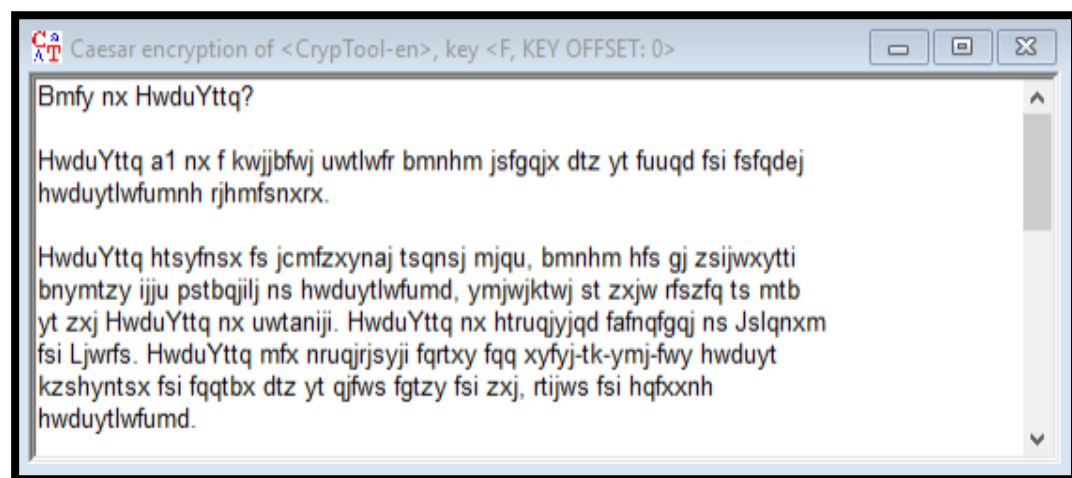
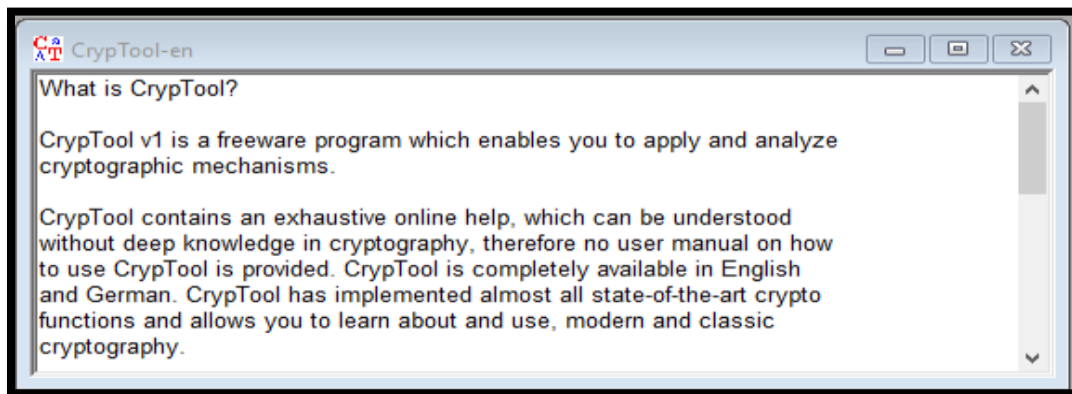
The decryption operation works in exactly the same way as described above. We decrypt this document once again and the plaintext version is restored. This of course now consists only of a sequence of upper case letters. Because no distinction is made any longer between upper and lower case letters, and especially because of the absence of formatting, this text is more difficult to read. However, if we compare the document briefly with the original text, we can see that it is the same text. The original text begins with the words, "CrypTool is a program which will enable you ..." These words appear once again in the first line of the decrypted text.

Security of Caesar Cipher

Frequency Analysis:

One of the known ciphertext attacks is frequency analysis. It is based on the frequency of letters or groups of letters in a ciphertext. Different letters are employed with varying frequency in all languages.

Having now learned the steps involved in encrypting and decrypting a document, we would like to take a look at the security of the Caesar encryption algorithm. To do this, all the windows should be closed apart from the window containing the plaintext and the version of the encrypted text in which the formatting is retained. The only two windows now open should be the following:



First, make the window containing the plaintext the active window again by clicking on it with the mouse. We now have the frequency distribution of the letters (Analysis \ General \ Histogram) calculated. We repeat the same procedure after making the other window active.

When we examine the histogram of the encrypted document, we can see that the letter frequencies have merely been shifted by five position. That means that this old encryption algorithm is not secure. Moreover, there are only 26 possible keys. The Caesar encryption algorithm can be broken easily by a ciphertext-only attack. To perform such an attack, restore the window containing the encrypted text to the active window and select Analysis \ Ciphertextonly \ Caesar. The text will automatically be analyzed.

By analyzing the superposition, it is possible to discover the key which was used to encrypt this document. In this case it was the letter F. When you click on the Decrypt button in the message window, the plaintext appears, i.e. the text that has been decrypted with the key F that was discovered. So CrypTool managed successfully to find the key with which the document had been encrypted. The only information it needed to do this was the encryption algorithm.

Atbash Cipher

Atbash cipher (also known as mirror cipher, reversed alphabet, or reverse alphabet) is a monoalphabetical substitution cipher that takes its name and origins from the Hebrew alphabet.

Atbash encryption employs a replacement alphabet and its reciprocal, which is a combination of the normal and reverse alphabets (mirrored).

Normal	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Reverse	ZYXWVUTSRQPONMLKJIHGFEDCBA

Because there is just one method to do this, the Atbash cipher provides no communication security because it lacks any kind of key. If many collating orders are available, the one used in encryption can be used as a key, but this does not provide considerably greater security because just a few letters reveal which one was chosen.

Polyalphabetic Ciphers

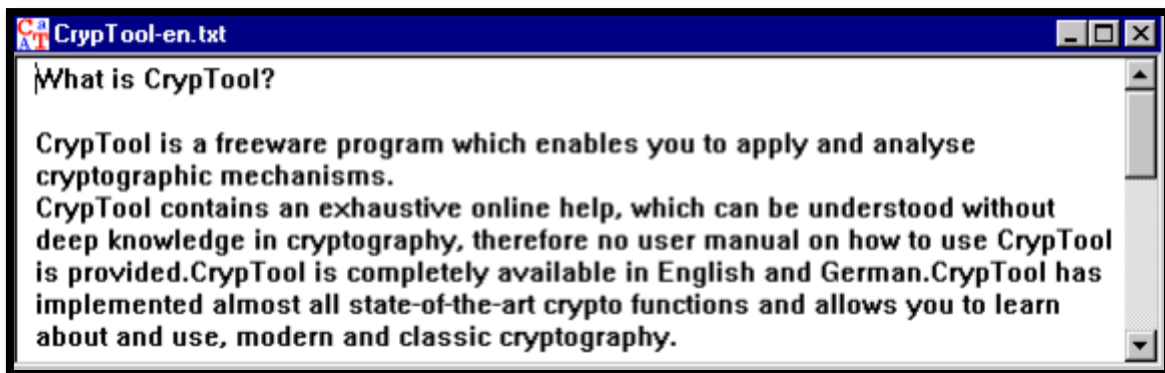
A polyalphabetic cipher is any substitution cypher that employs numerous substitution alphabets. Though it is a simplified particular case, the Vigenère cipher is arguably the most well-known example of a polyalphabetic encryption.

Vigenère Cipher

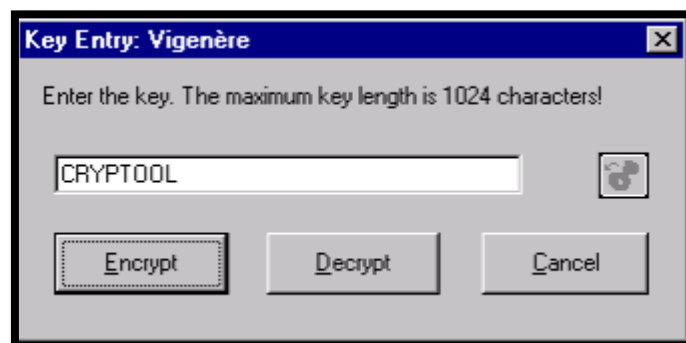
The Vigenère cipher encrypts alphabetic text using a sequence of interwoven Caesar ciphers depending on the letters of a keyword. It makes use of polyalphabetic substitution.

First of all, to acquaint ourselves with the Vigenère encryption algorithm we will open a document, encrypt it and then decrypt it again. We will then try to get the computer to work out the key with which a plaintext is encrypted. To this end we open a document which contains a part of the CrypTool Help text

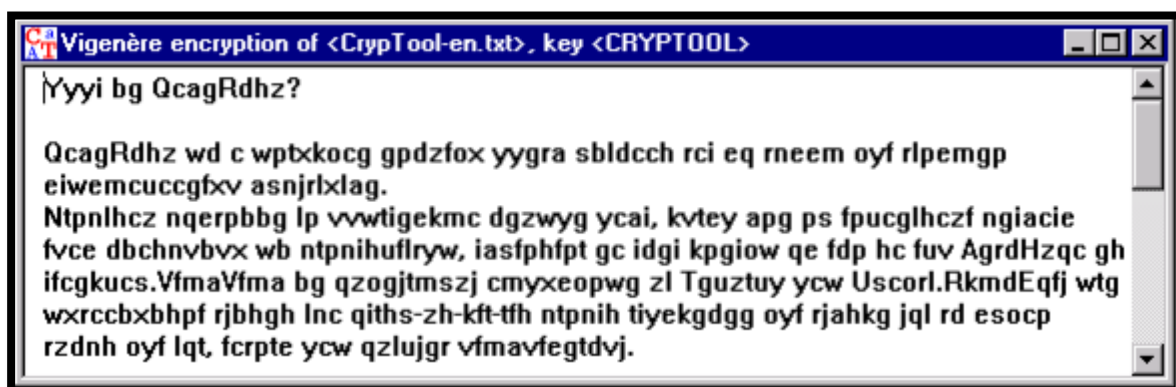
"Introduction to CrypTool". The file name of the document to be opened is CrypTool-en.txt. This file is opened via the menu selection File \ Open.



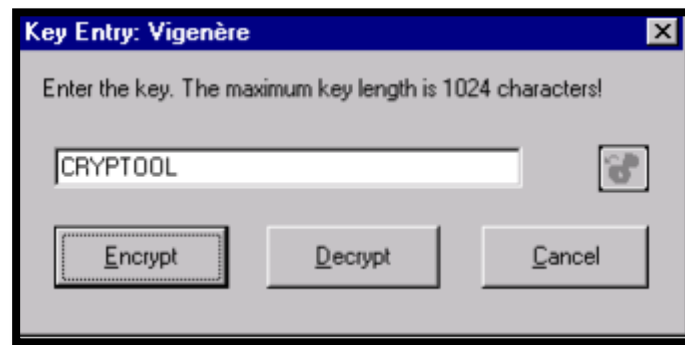
We now want to encrypt this document using the Vigenère encryption algorithm. To do this, we select Encrypt/Decrypt \ Classical \ Vigenère, following which this dialog box appears. As key, we enter CRYPTOOL.



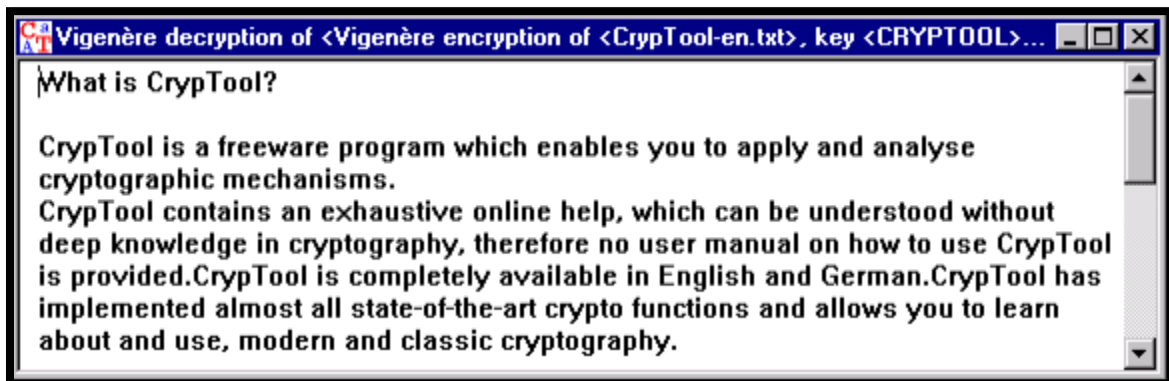
Clicking on the Encrypt button opens a new window that contains the encrypted text.



The plaintext version of this encrypted document can now be obtained simply by selecting Encrypt/Decrypt \ Classical \ Vigenère. In the dialog box which then appears we enter the key with which the document was encrypted (CRYPTOOL). This time we do not want the text to be encrypted, but instead to be decrypted. Therefore the Decrypt field must be selected. This is done by clicking on the field itself or on the radial button to the left of the field.

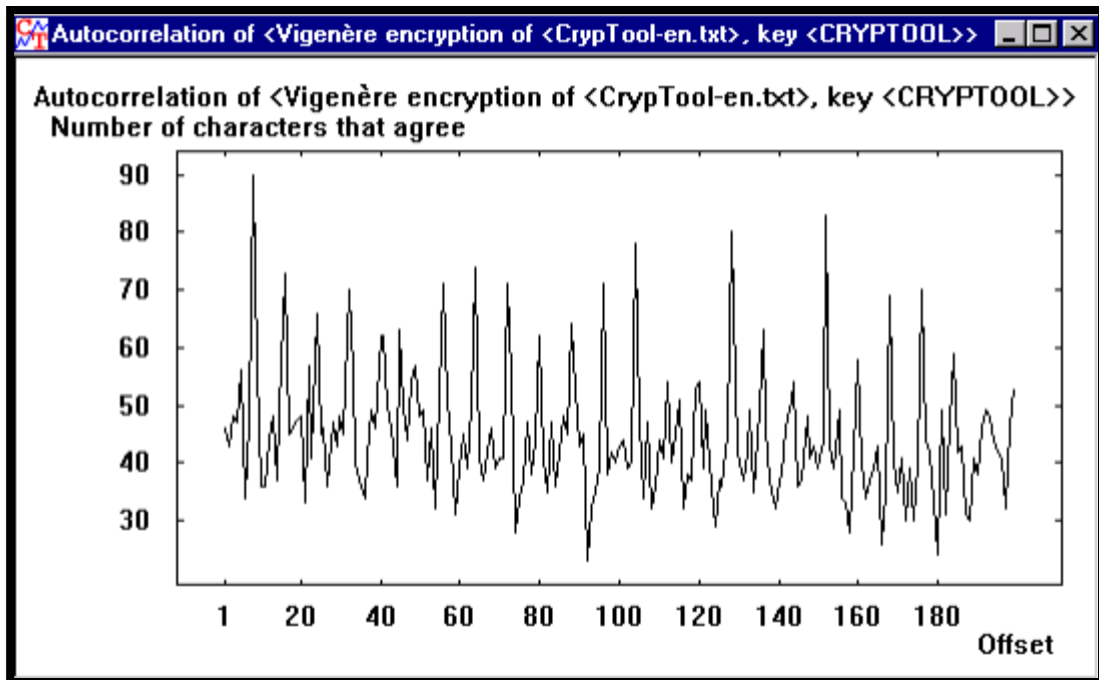


Clicking on the Decrypt button instructs CrypTool to decrypt the text. The plaintext appears immediately.

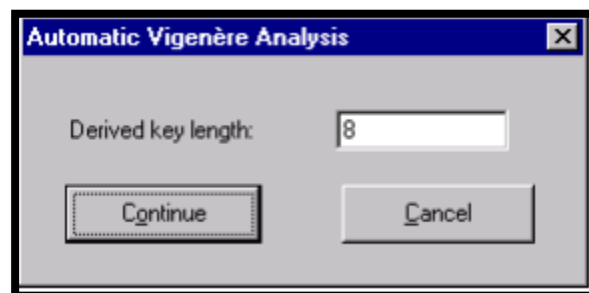


Security of Vigenère Cipher

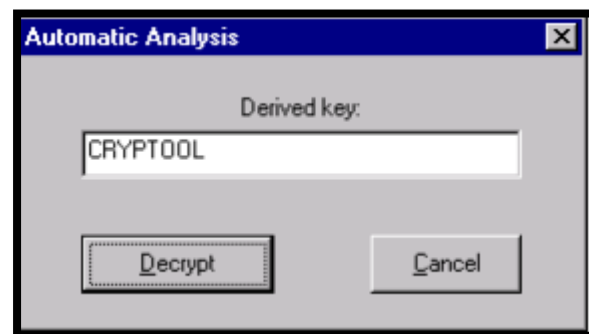
We now wish to find the key with which the document was encrypted. Restore the window containing the encrypted document (via a mouse click) and select Analysis \ ciphertext only \ Vigenère, following which the text will automatically be analyzed. First of all a new, autocorrelation window opens.



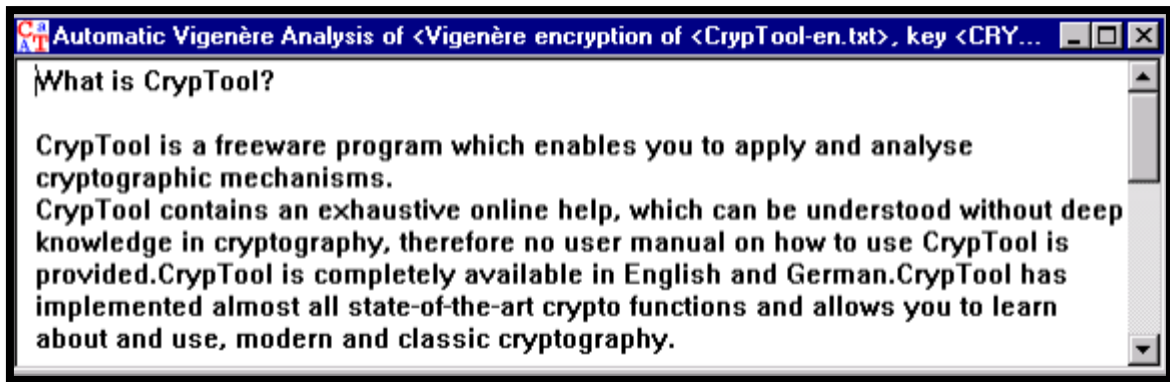
It is possible to work out the key length from the regular peaks in the autocorrelation. It is correctly calculated at 8 characters.



When the Continue button is clicked, another dialog box opens to display the key that has been computed.



Clicking on Decrypt closes this dialog box and opens another window which contains the unencrypted text.



As you can see, the distinction between upper and lower case and the formatting (blank characters, line breaks etc.) have been retained. Special characters such as umlauts, hyphens and slashes were kept on in the encrypted document. Normally in a classical encryption algorithm all formatting is removed and only the letters are encrypted, but to make the text easier for you to read and compare, CrypTool retains the formatting in the encrypted text (and hence also in the decrypted text). By disabling the formatting options accessed via Options \ Text Options, it is possible to disable retention of formatting and, by checking the relevant boxes again, to reinstate formatting.

Exercise:

1. Write the description of the project you are working on and encrypt it using the first letter of your name using caesar cipher.

2. Why rot-13 is considered an inverse of itself? Encrypt the same text twice using the same key and see the result?

3. What does frequency analysis tell us about any encryption algorithm knowing that some letters are more common in the English language than others? How can this be used in cryptanalysis? (Break the encrypted text "pgf wpxgtukva", what is the key? In CrypTool, run analysis->tools->histogram)

4. How would you define the vigenère cipher with respect to caesar cipher where the key length is 1. Using brute force decrypt the text "ZL ANZR VF XUNA NAQ V NZ ABG N GREEBEVFG" using vigenère cipher where the key is a single letter.

5. What is atbash substitution, how does it work? Is the atbash a weak or strong cipher and why? If we go in to the atbash menu and select key entry: remaining characters are filed in ascending order, what is the effect of choosing a key 'LEMON' in one instance and in another instance the key 'ZXC'. Write two lines about the CSIT department of NED University and determine which key is stronger?

6. Use Cryptool to analyze the cryptogram “Glh dylpo eesza jrk nxztv bzhe xkr pdmc gbk” obtained via Vigenère cipher.

Lab Session 02

Object: Implementation of Polygraphic Substitution Ciphers (Hill Cipher) and the Attacks on It

Theory:

Polygraphic Substitution Ciphers

Polygraphic substitution is a cypher in which blocks of letters are uniformly substituted. When the length of the block is known, more precise names are used: for example, a bigraphic cypher is one in which pairs of letters are replaced.

Simple substitution cyphers were improved upon, and polygraphic substitution cyphers were created. They were extremely popular during the Renaissance and were used throughout Europe for centuries.

Polygraphic substitution cyphers function by breaking the plaintext into many sections and substituting each with a word, a single letter or integer, or anything else. To decode ciphertext letters, reverse the substitution and reverse the phrases.

In this lab we are going to study two polygraphic ciphers and the attacks on them. These polygraphic ciphers are as follows:

- Hill Cipher
- Playfair Cipher

Hill Cipher

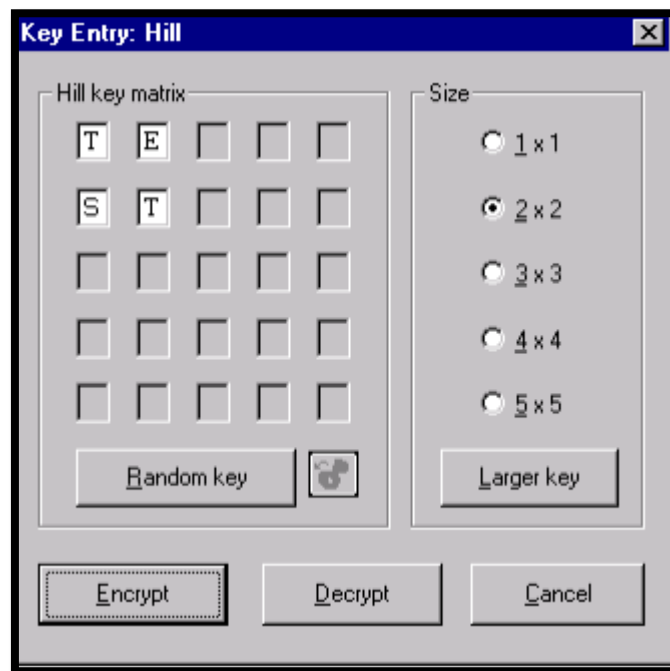
The Hill cypher uses linear algebra to create a polygraphic substitution cypher. A modulo 26 integer is assigned to each letter. The basic scheme A = 0, B = 1,..., Z = 25 is frequently employed, although it is not a requirement for the encryption to work. To encrypt a message, an invertible $n \times n$ matrix is multiplied by each block of n letters (considered an n -component vector) against modulus 26. Each block is multiplied by the inverse of the encryption matrix in order to decode the message.

The cypher key is an invertible $n \times n$ matrix that is used for encryption. It should be chosen at random from a collection of invertible $n \times n$ matrices (modulo 26).

The text HILLCIPHER is to be encrypted with the key T E S T. To do this, a new document is created (via File \ New) and the text HILLCIPHER is entered. Under the classical encryption algorithms lower case letters are always replaced by the corresponding upper-case letters. However, CrypTool keeps upper- and lower-case letters distinct in order to make the text easier for you to read. These two aspects can be disabled independently of each other via the options Keep characters not present in the alphabet unchanged and Keep uppercase / lowercase (if possible) in the Text Options dialog box. Therefore, it does not matter whether the text contains lower case letters. To demonstrate this, we will use Hillcipher instead of HILLCIPHER:



We now select Encrypt/Decrypt \ Classical \ Hill, following which this dialog box appears:

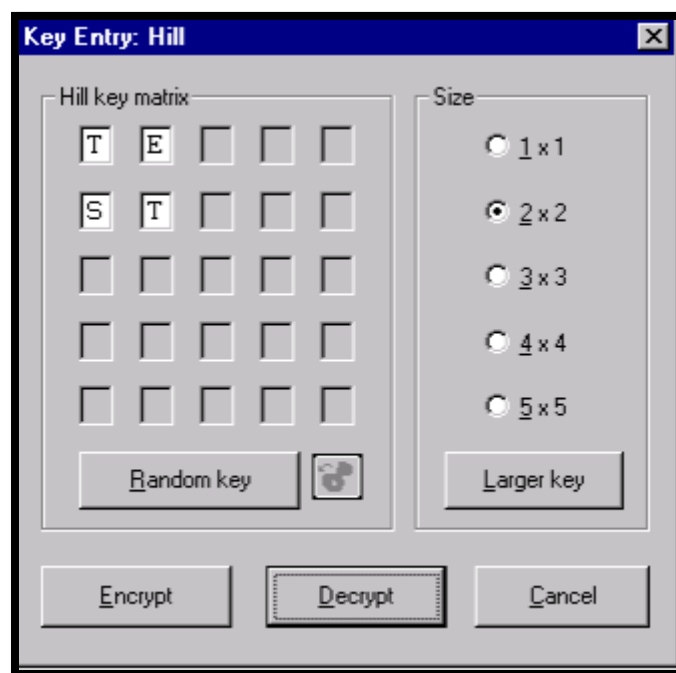


After entering the key T E S T it can be saved in the clipboard (in order to avoid having to enter it again later on at the decryption stage) by clicking on the Copy button. Information on the clipboard will be found

in the Help facility on the toolbar. Now click on the Encrypt button, and a new window that contains the encrypted text opens.



The encrypted text is now decrypted for checking purposes. The decryption procedure is identical to encryption, except that this time the Decrypt option must be selected, i.e., the radial button next to that option must be selected. This is done by clicking on the field with the mouse. The key can now either be typed in manually as before or else it can be inserted from the clipboard using the Paste button if it had previously been copied.



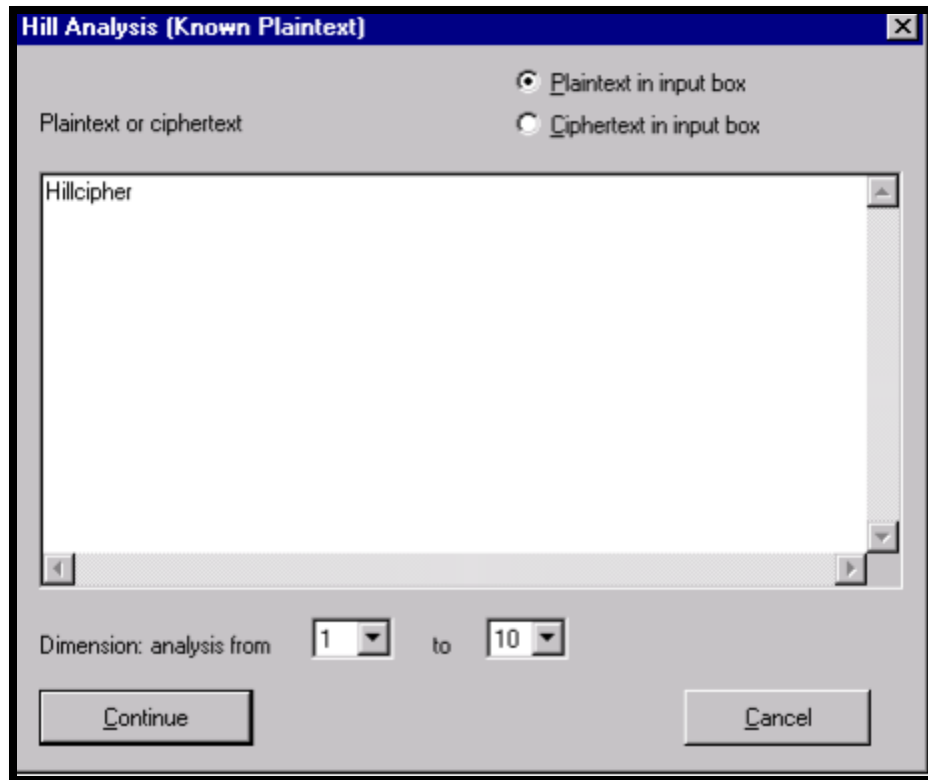
Now click on Decrypt, and the plaintext reappears.



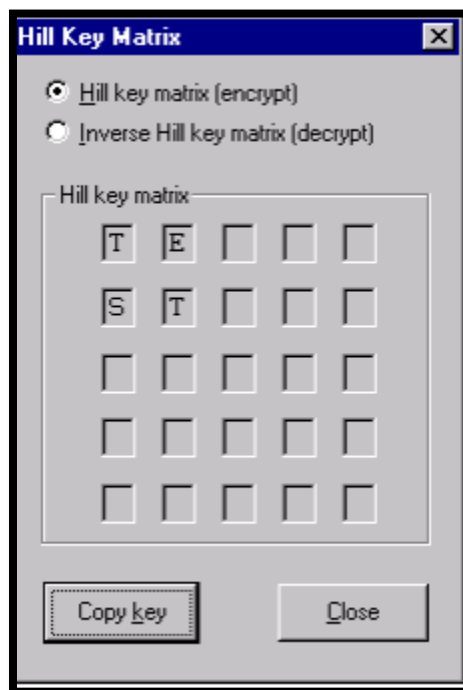
Known Plaintext Attack:

The known-plaintext attack (KPA) is a cryptanalysis attack paradigm in which the attacker has access to both the plaintext (also known as a crib) and the encrypted version of the plaintext (ciphertext). These can be utilized to uncover other secrets, such as hidden keys and code books.

CrypTool will now work out the key using a Known Plaintext attack. Such an attack requires that both the plaintext and the encrypted text are available. The easiest approach is to copy the plaintext (mark with the mouse pointer and then press the key combination Ctrl+C) from one of the two windows which contain Hillcipher to the clipboard. Now make the window containing the encrypted text the active window again (by clicking on it with the mouse) and select the menu option Analysis \ Known Plaintext \ Hill to view the Analysis Hill cipher window. The plaintext is then pasted from clipboard into this window (using the key combination Ctrl+V).



The analysis starts when the Continue button is clicked and the key T E S T appears in the following window.



In this way the key was worked out simply from knowing the plaintext and the corresponding encrypted text.

Exercise

1. Explain the working of the Hill cipher. Encrypt the text "Hello World" with a key $K = \begin{bmatrix} C & B \\ D & E \end{bmatrix}$.

What is the result?

2. Using analysis in Cryptool, apply the known plain text attack and find out the key of Hill Cipher.
Known Plain Text "friday" and corresponding cipher text is "PQCFKU"

Lab Session 03

Object: Implementation of Polygraphic Substitution Ciphers (Playfair Cipher) and the Attacks on It

Theory:

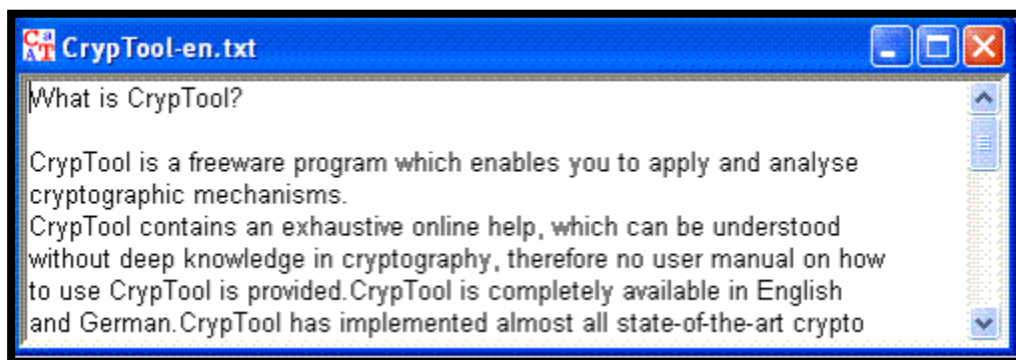
Playfair Cipher

The Playfair cypher was the first digraph substitution cypher to be used in practice. The cypher was devised in 1854 by Charles Wheatstone, but it was given the name Playfair after Lord Playfair, who encouraged its usage. Unlike standard cyphers, playfair cypher encrypts a pair of alphabets (digraphs) rather than a single alphabet.

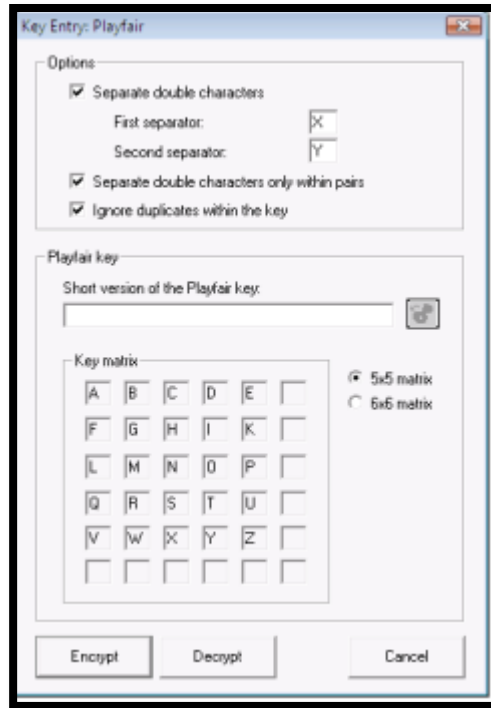
We start with investigating the Playfair encryption algorithm by opening a document, encrypting it and decrypting it again. After that we will try to find the key that was used to encrypt the plaintext by using the manual Playfair analysis step by step.

1. Playfair Encryption

Open the document to be encrypted: e.g., CrypTool-en.txt, which is part of the online help section. You can open that document via the menu File \ Open.



Now we encrypt this document using the Playfair encryption algorithm. From the menu, choose Encrypt/Decrypt \ Symmetric (classic) \ Playfair. The following dialog box is displayed:



Using the standard 5x5 key matrix, the Playfair alphabet consists of the normal capital letters A to Z, where J is taken out.

Please use PEGASOS as the key. From this key the following Playfair key matrix will be calculated:

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

When you push the button Encrypt the Playfair key entry dialog will be closed and two windows are opened: one with the pre-processed plaintext and another with the encrypted text:



- The pre-processing step removes all characters from the cleartext that are not part of the Playfair alphabet and inserts the character X between two equal characters in a row. E.g. the cleartext CrypTool ist (German for CrypTool is ...) will be pre-processed to CRYPTOXOLIST:
- Based on the Playfair key matrix above, the pre-processed text can now be encrypted. The basic idea is that two consecutive characters (a di-gram) determine the two opposite edges of a rectangle in the key matrix. We consider one dimensional rectangle for the special case, when both characters are in the same row or column of the key matrix

To proceed with the next step (decryption) you can avoid entering the key again by recovering it from the ciphertext window. To do this

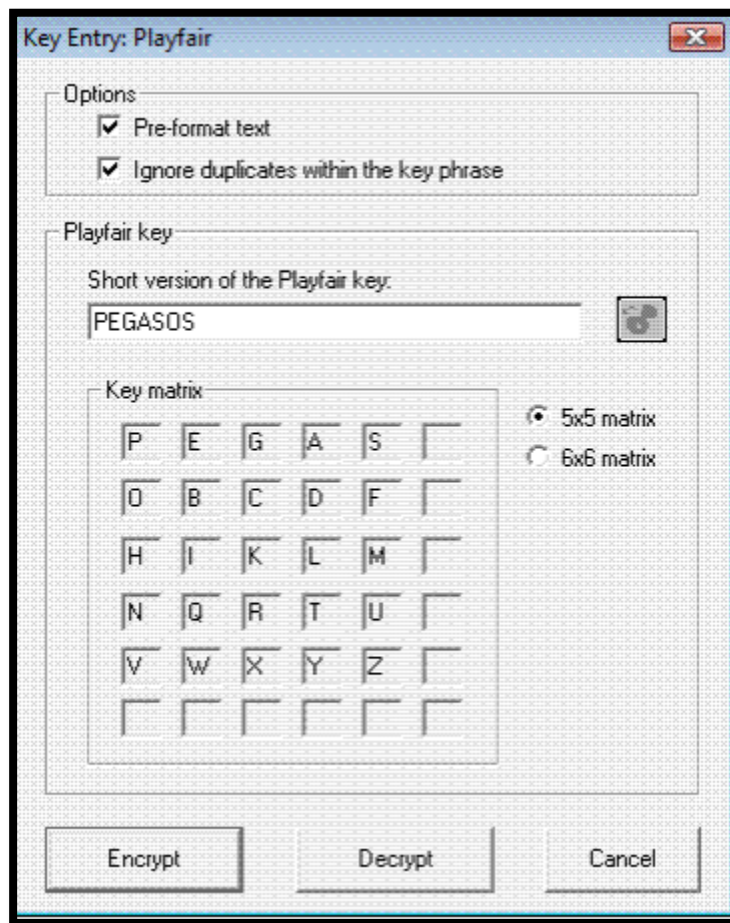
- activate the window with the ciphertext
- click on the key icon or select menu Edit \ Show Key
- press Copy key

Now the key matrix is stored in CrypTool's key store.

Playfair Decryption

We now can recover the plaintext of the document encrypted in the previous step. Select the menu Encrypt/Decrypt \ Symmetric (classic) \ Playfair.

Enter the encryption key (PEGASOS). If you have saved the key in the internal key storage, then you can recover it by clicking on the key icon.



Press the Decrypt button to recover the plaintext.

Manual Analysis of the Playfair Cipher

Let us now try to decipher a text encrypted with the Playfair cipher where we don't know the key (ciphertext-only analysis).

To do this we load the text to be decrypted from the file Playfair-enc-de.txt (this file can be found in the examples directory distributed with CrypTool):



This text was created with the Playfair encryption algorithm without pre-processing: This option effectively leaves characters not contained in the Playfair alphabet unencrypted (e.g. blanks, punctuation marks, line feeds), so that the plaintext word and sentence boundaries are obvious.

Additionally, the insertion of the separating X between two equal, consecutive characters is switched off so double characters are left unencrypted.

CrypTool does only support a semi-manual analysis of the Playfair cipher. Therefore, we need some clue about the document to be analyzed. Let us assume that the text is a German letter, which often start with the same words.

Let us now open the Playfair analysis dialog window by selecting the menu Analysis \ Symmetric Encryption (classic) \ Manual Analysis \ Playfair.

Here you see the text of the opened ciphertext file in the first row of the field "Analysis result".

Playfair Analysis

☐ Text was pre-formatted
☒ Duplicates are ignored
☒ 5x5 Matrix ☐ 6x6 Matrix

☒ Update expected plaintext using the key matrix

	LRQU	Nb?	Horizontal	Vertical	row or col?
A	XXXX	/			
B	XXXX	/			
C	XXXX	/			
D	XXXX	/			
E	XXXX	/			
F	XXXX	/			
G	XXXX	/			
H	XXXX	/			
I	XXXX	/			
K	XXXX	/			
L	XXXX	/			
M	XXXX	/			
N	XXXX	/			
O	XXXX	/			
P	XXXX	/			
Q	XXXX	/			
R	XXXX	/			
S	XXXX	/			
T	XXXX	/			
U	XXXX	/			
V	XXXX	/			
W	XXXX	/			
X	XXXX	/			

Analysis result: The following rows contains ciphertext, input or result of the frequency analysis, found plaintext
 TGIG.HRRIEFR.XVNIX.AXL.RRGGRX...IBO.ARGXNHK.HRIRGNRG.UTPU.KNRETF.OBH

Expected plaintext:

We assume: The letter probably begins with Sehr geehrte Damen und Herren (German for Dear Sirs).

Enter this text in the field labeled Expected plaintext. After each key stroke the Letter information table is updated.

We now already have a couple of di-grams fixed:

SE > TG AM > VN

HR > IG EN > IX

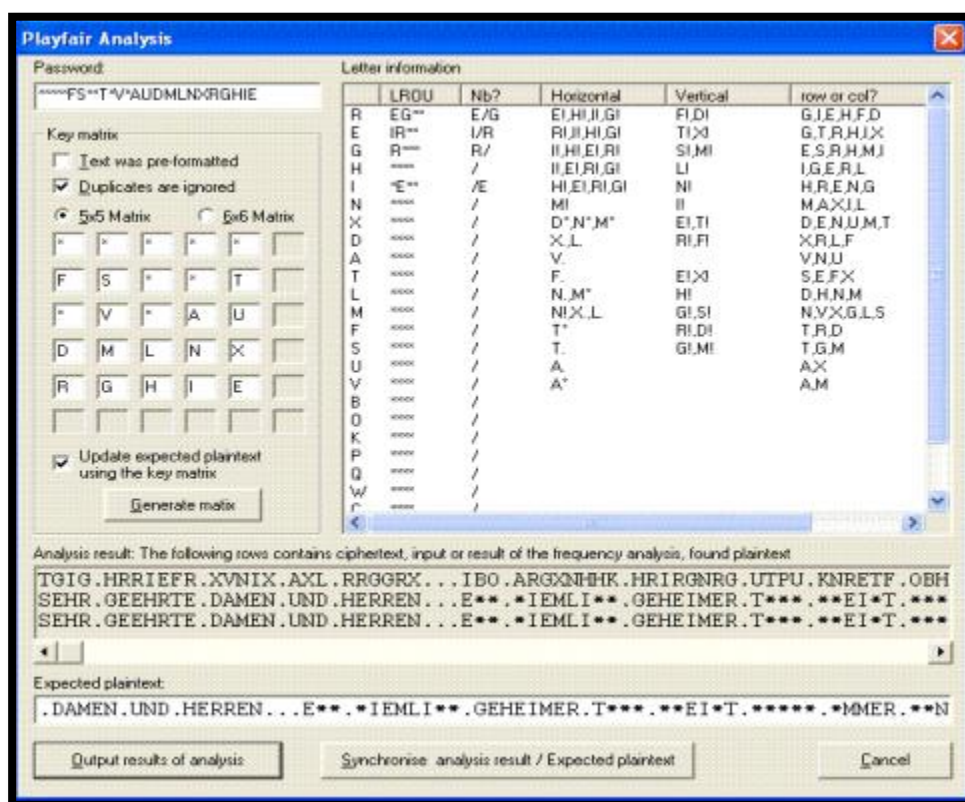
GE > HR UN > AX

EH > RI DH > LR

RT > EF ER > RG

ED > RX RE > GR

By pressing the Generate matrix button, those parts of the Playfair key matrix are calculated, that can be determined based on the given information.



Press the button Synchronise analysis result \ Expected plaintext. The selected checkbox-option Update expected plaintext causes that the result of the decryption of the ciphertext based on the partial matrix is copied into the field "Expected plaintext". Characters that cannot (yet) be decrypted are marked with the wildcard character '*' and can now be replaced by guessing.

Now the character sequence E**.*IEMLI** catches our eye (at least the eye of people knowing some German). If we complete this with the field "Expected plaintext" to read EIN.ZIEMLICH two more di-gram mappings result: IN > BO and CH > HK. Based on the previous matrix and the first di-gram mapping we conclude, that the characters I,N,B,O have to be in the same column, i.e. B must be directly below I and O directly below N. Clicking on Generate matrix swaps row two and four, then row two and three etc. resulting in the following key matrix:

*	*	*	K	B
U	Z	*	*	A
X	D	L	M	N
T	F	S	C	O
E	R	G	H	I

Unfortunately, decrypting based on this matrix results in words that do not exist in the German language:

SEHR.GEEHRTE.DAMEN.UND.HERREN...EIN.ZIEMLICH.

GEHEIMER.*X**BLEIOT.NICH*.IMMER.**NGE.GEHEIM..

MI*.*DXUNDLICHEN.GRUX**EN

However, we can now guess the cleartext with high probability, except for the word *X**, which we correct to read **** (this is to avoid conflicts). We now correct the remaining words to read:

SEHR.GEEHRTE.DAMEN.UND.HERREN...EIN.ZIEMLICH.

GEHEIMER.****BLEIBT.NICHT.IMMER.LANGE.GEHEIM..

MIT.FREUNDLICHEN.GRUESSEN

T	F	S	K	B
U	Z	V	W	A
X	D	L	M	N
*	*	*	C	O
E	R	G	H	I

In a final step we correct TE** to read TEXT giving the di-gram mapping XT > PU. The Playfair analysis now gives us the key matrix

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

The cleartext originally had been encrypted with the password HIERGIBTESWASZUENTDECKEN (see advice regarding the key matrix), and the options Preformat text and Ignore duplicates within the key phrase were deactivated. We finally get the complete plaintext:

SEHR.GEEHRTE.DAMEN.UND.HERREN...EIN.ZIEMLICH.

GEHEIMER.TEXT.BLEIBT.NICHT.IMMER.LANGE.GEHEIM..

MIT.FREUNDLICHEN.GRUESSEN

After clicking the button Output results of analysis, the Playfair analysis dialog window is closed and the cleartext is displayed.

Remark:

Advice regarding the key matrix:

A cyclicly shifted key matrix leads to the same result. Thus, the originally chosen password cannot be concluded from the calculated matrix.

Exercise:

1. Encrypt the text "SH IS HE RX RY LO VE SH EA TH LE DG ER "with a key K= SHERRY what is the result?

2. Encrypt and Decrypt your name as a text and NED as a key using Playfair cipher and show the steps involved? Verify the answer using Cryptool.

Lab Session 04

Object: Analyze and find hidden information in Text or Data Files

Theory:

Steganography

Steganography is the art of concealing a secret message within (or even on top of) a non-secret object. That something can be anything you want it to be. Many forms of steganography nowadays include hiding a secret piece of text within a photograph. Alternatively, you may hide a secret message or script inside a Word or Excel document.

Steganography's goal is to hide and deceive. It is a type of covert communication in which communications are hidden using any media. It isn't cryptography because it doesn't encrypt data or require the usage of a key. Instead, it's a type of data concealment that may be done in a variety of ways. Steganography is a technique that permits secrecy – and deception – in the same way that cryptography is a science that mostly provides privacy.

A text file containing secret information hidden inside an apparently harmless image file is one example of this. The receiver would receive a typical image file if this image file was delivered as an email attachment. You may, however, retrieve the hidden file using the correct tools. You may also include secret messages in the metadata of a file as text strings.

Steganography and Attacks Delivery

Today, attackers automate assaults using PowerShell and BASH scripts. Pen testers are the same way. Actual scripts have been embedded in macro-enabled Excel and Word documents, for example. When a victim opens the Excel or Word document, the hidden script is activated.

The attacker does not need to utilize deception to persuade the victim to use Steghide. The hacker – or pen tester – is "living off the land" in this situation. To take use of standard Windows apps and features like Excel and PowerShell, the attacker is employing a steganographic application. All the victim has to do is read the document, and a chain of bad events begins.

How Steganography is Used

Open your command prompt, change directory where Steghide is extracted and type Steghide to view the help content of the application.

```
Command Prompt
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\ForensicUser>cd Downloads
C:\Users\ForensicUser\Downloads>cd steghide
C:\Users\ForensicUser\Downloads\steghide>steghide
steghide version 0.5.1



the first argument must be one of the following:
embed, --embed          embed data
extract, --extract      extract data
info, --info            display information about a cover- or stego-file
info <filename>         display information about <filename>
encinfo, --encinfo      display a list of supported encryption algorithms
version, --version       display version information
license, --license       display steghide's license
help, --help            display this usage information

embedding options:
-ef, --embedfile        select file to be embedded
-ef <filename>          embed the file <filename>
-cf, --coverfile        select cover-file
-cf <filename>          embed into the file <filename>
-p, --passphrase        specify passphrase
-p <passphrase>         use <passphrase> to embed data
-sf, --stegofile        select stego file
-sf <filename>          write result to <filename> instead of cover-file
-e, --encryption        select encryption parameters
```

We will always have a cover document and a hidden document while using Steghide. The file formats that we can use are listed below (notice that owing to their file structure, Cover Documents can only be pictures or audio files).

Cover Document	Secret Document
JPG	No Format Restriction
BMP	
WAV	
AU	

We're going to be using our cover document '**laptop.jpg**' and our secret document '**secret.txt**'

Name	Date modified	Type	Size
 laptop.JPG	25/04/2022 16:19	JPG File	18 KB
 secret.txt	25/04/2022 16:27	Text Document	0 KB

Now use Steghide to conceal secret.txt and embed it inside laptop.jpg. The command we're using, steghide embed -cf laptop.jpg -ef secret.txt, is explained here.

- steghide – Chooses the tool we'll employ.

- embed – This option chooses the mode we wish to employ (embedding files)
- -cf laptop.jpg – Choosing a cover image (the file we want to hide data inside)
- -ef secret.txt – This option selects the file to embed (the file we want to hide)



When prompted for a password, you may use whatever you choose. This will be used to extract the data afterwards in order to safeguard its secrecy, which means that only the intended recipient will be able to see it.

```
C:\Users\ForensicUser\Downloads\steghide>steghide embed -cf laptop.JPG -ef secret.txt
Enter passphrase:
Re-Enter passphrase:
embedding "secret.txt" in "laptop.JPG"... done
```

Laptop.jpg has been rewritten to become laptop.jpg containing an embedded file. We may use the -sf flag to output it to a new file if we wish to generate a new 'stego' file. We used -sf laptopsecret.jpg to output to a new file in the screenshot below.

```
C:\Users\ForensicUser\Downloads\steghide>steghide embed -cf laptop.JPG -ef secret.txt -sf laptopsecret.jpg
Enter passphrase:
Re-Enter passphrase:
embedding "secret.txt" in "laptop.JPG"... done
writing stego file "laptopsecret.jpg"... done
```

A new jpeg image file is created in the folder named as laptopsecret.jpg

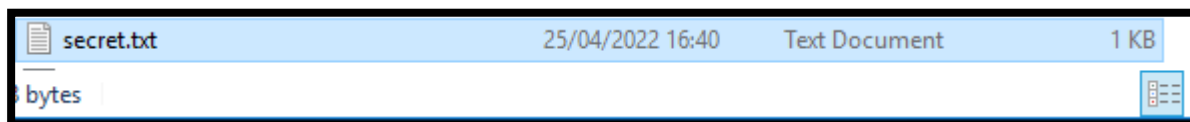
 laptop.JPG	25/04/2022 16:34	JPG File	11 KB
 laptopsecret.jpg	25/04/2022 16:35	JPG File	11 KB

We know laptopsecret.jpg is a stego file with an embedded file, and we also know the password is 'password.' 1 To recover the hidden file, use the following command: laptopsecret.jpg steghide extract -sf

- extract – Selects the mode we want to use
- steghide – Selects the tool we want to use (extracting files)
- -sf laptopsecret.jpg – This option selects the steganography file to be extracted (combination of cover file and hidden data)

```
C:\Users\ForensicUser\Downloads\steghide>steghide extract -sf laptopsecret.jpg
Enter passphrase:
wrote extracted data to "secret.txt".
```

When prompted for the passphrase, we enter it, and the tool will extract secret.txt and place it in the current directory.



Exercise

1. Create a secret message containing your name and roll number along with “The World is Real. Be good to yourself.” Embed the file ‘secretmessage.txt’ inside the cover file ‘coverfile.jpg’ and name the output stegofile ‘hiddenmessage.jpg’. What is the full command you would use to do this?

2. Extract the secret message send by your fellow student. What is the message and how to extract it?

Lab Session 05

Object: Hash Functions, Demonstration of MD5 and MD5 Hash Collision

Theory:

Hash Functions

A hash function is a (mathematical) function which receives an input of arbitrary length and returns a function value (hash value) of a fixed length (usually 128 or 160 bits). The hash functions which are used in cryptography should be called one-way hash functions.

The hash function itself must be public, i.e., everyone should be able to (efficiently) calculate the hash value for a given input. Conversely, however, it must be (computationally) infeasible to find an input for a given value which possesses exactly the predetermined value as hash value (this is referred to as a one-way characteristic).

In cryptographic practice, generally hash functions must satisfy a second, more stringent requirement: it must be impossible to find two values which are mapped by the hash function onto the same hash value (collisions), since otherwise it would be possible to replace an already signed message after the event. Hash functions are especially used in connection with digital signatures.

Here, for reasons of efficiency, the hash value of a message is signed instead of the message itself. The hash value $H = H(M)$ of an original message M , is used to check, if the received message M' has been changed.

Example: Some earlier computer magazines added row check sums to the printouts of source code examples. Using these check sums, it was possible to recover fast typing errors of the printed source code, because only the row check numbers of the source code have to be checked to discover an error within a row (CDs and downloads didn't exist in these times).

For these checksum methods it is easy to find two different row messages $R1$, $R2$ with the same row check sum - meaning $H(R1) = H(R2)$ but $R1 \neq R2$: So it is possible to compute a malicious modified source code, without the possibility to detect this modification only via checking the row check sums. We call two different rows with equal check sum a collision. In the notion of hash values we call this a collision if

$M \neq M'$ and $H(M) = H(M')$

For cryptographic purposes we must consider the requirement that for the hashing method H and any message M it must be practically impossible to compute a collision. This means to compute a fake message $M' \neq M$, with the characteristic $H(M') = H(M)$.

In the example with the printed source code in earlier computer magazines simple checksum methods have been satisfying, because messages (lines) were not faked by will, but only accidentally mistyped.

CrypTool provides cryptographically strong hash methods via the **Indiv. Procedures \ Hash** menu. These hash methods compute for any document a hash value of constant length, independent whether if the input length is 1 byte or 1 Mbyte. The length of the output (hash value) is always between 16 bytes and 64 bytes (between 128 bit and 512 bit) long - depending on the hash method.

Message Digest MD5

The MD5 algorithm produces a hash value of length 128 bits (= 16 bytes).

Step 1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

Step 2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^{64} , then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.) At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let $M[0 \dots N-1]$ denote the words of the resulting message, where N is a multiple of 16.

Step 3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Step 4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

$$F(X,Y,Z) = XY \vee \text{not}(X) Z$$

$$G(X,Y,Z) = XZ \vee Y \text{not}(Z)$$

$$H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z$$

$$I(X,Y,Z) = Y \text{ xor } (X \vee \text{not}(Z))$$

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of \vee since XY and $\text{not}(X)Z$ will never have 1,s in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, then each bit of F(X,Y,Z) will be independent and unbiased. The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T[1... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times $\text{abs}(\sin(i))$, where I is in radians.

Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D. This completes the description of MD5.

MD5 Hash Collision

The lab uses a tool called “Fast MD5 Collision Generation”, which was written by Marc Stevens. The name of the binary is called `md5collgen` in SEED VM, and it is installed inside the `/usr/bin` folder.

In this task, we will generate two different files with the same MD5 hash values. The beginning parts of these two files need to be the same, i.e., they share the same prefix. We can achieve this using the **md5collgen** program, which allows us to provide a prefix file with any arbitrary content. The way how the program works is illustrated in Figure 1. The following command generates two output files, `out1.bin` and `out2.bin`, for a given a prefix file `prefix.txt`:

```
$ md5collgen -p prefix.txt -o out1.bin out2.bin
```

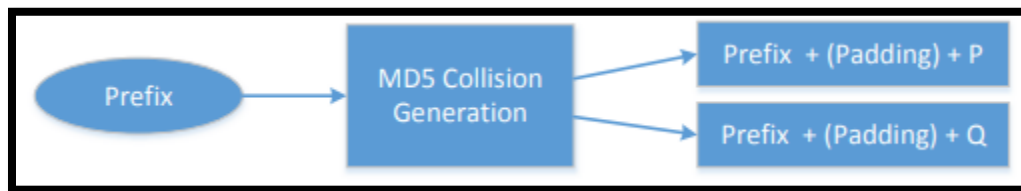


Figure 1 MD5 collision generation from a prefix

We can check whether the output files are distinct or not using the `diff` command. We can also use the **md5sum** command to check the MD5 hash of each output file. See the following commands:

```
$ diff out1.bin out2.bin
$ md5sum out1.bin
$ md5sum out2.bin
```

Since `out1.bin` and `out2.bin` are binary, we cannot view them using a text-viewer program, such as `cat` or more; we need to use a binary editor to view (and edit) them. A hex editor software called **bles** is already installed in SEED VM. Please use such an editor to view these two output files, and describe your observations.

Exercise

1. Are the data (128 bytes) generated by md5collgen completely different for the two output files?
Please identify all the bytes that are different.

2. If the length of your prefix file is not multiple of 64, what is going to happen?

3. Calculate Hash or fingerprint of your name using MD5, SHA-1 and SHA-256 algorithms and explain the result.

4. The hash compute for any document the output is a constant length, independent whether if the input length is 1 byte or 1 Mbyte. Explain how the hash algorithms achieve this fixed length output.

Lab Session 06

Object: Folder Security Implementation Through Batch Processing

Theory:

In DOS, OS/2, and Microsoft Windows, a batch file is a text file containing a series of commands intended to be executed by the command interpreter. When a batch file is run, the shell program (usually COMMAND.COM or cmd.exe) reads the file and executes its commands, normally line-by-line. Batch files are useful for running a sequence of executables automatically and are often used by system administrators to automate tedious processes.

DOS batch files have the filename extension .bat (or .BAT because file names are case insensitive in DOS, Windows and OS/2). There are a number of subtle differences between the .bat and .cmd extensions which affect the error levels returned, among other things; it is a common myth that .bat are run by COMMAND.COM while .cmd are run by CMD.EXE - this is not true and is not the reason for the differences. Also, the Windows 9x family only recognizes the .bat extension.

In this lab session we are going to implement folder security through Batch Programming

PROCEDURE

1. Create a New Folder and named it "test".
2. Move your files that you want to secure in to that folder.
3. Write the following script in a notepad to secure your folder and save it with the name "LOCK.bat"

```
@echo off

echo *****

echo *** Securing ..... Folder ***

echo *****

set /p folder= Which Folder ?

ren %folder% %folder%.{20D04FE0-3AEA-1069-A2D8-08002B30309D}

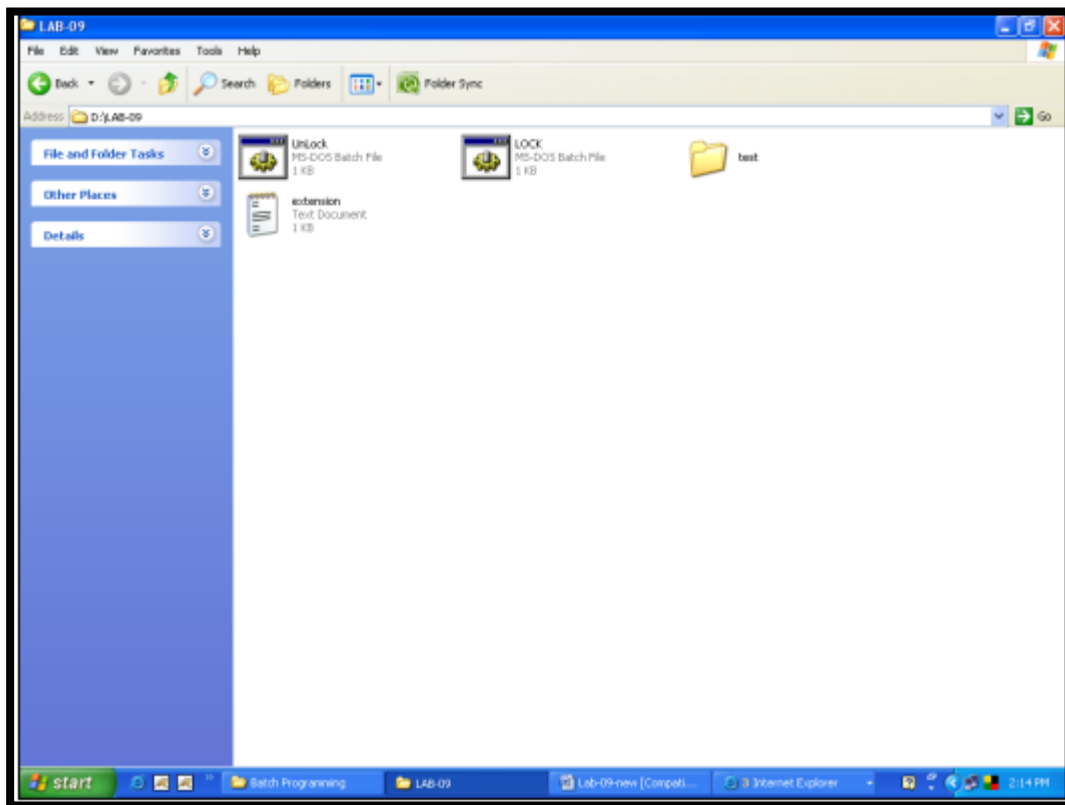
echo Done
```

4. Write the following script in a notepad to UN-Lock your folder and save it with the name "UNLOCK.bat"

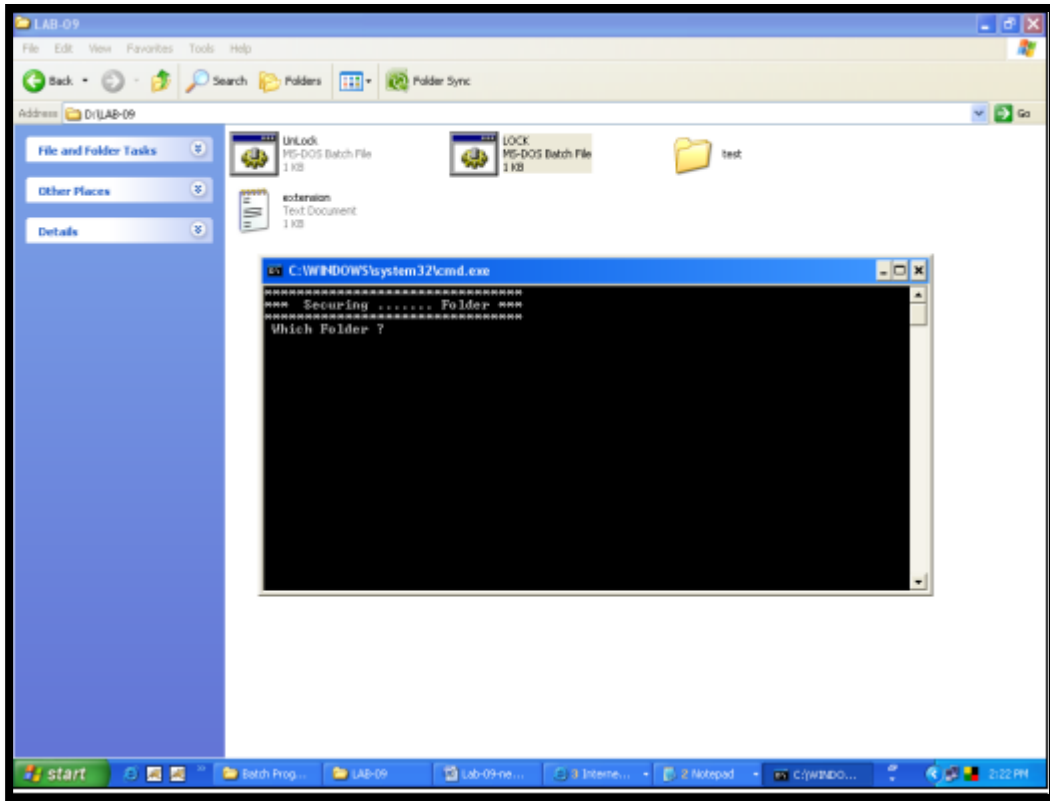
@echo off

set /p folder= Which Folder To be UnLock ?

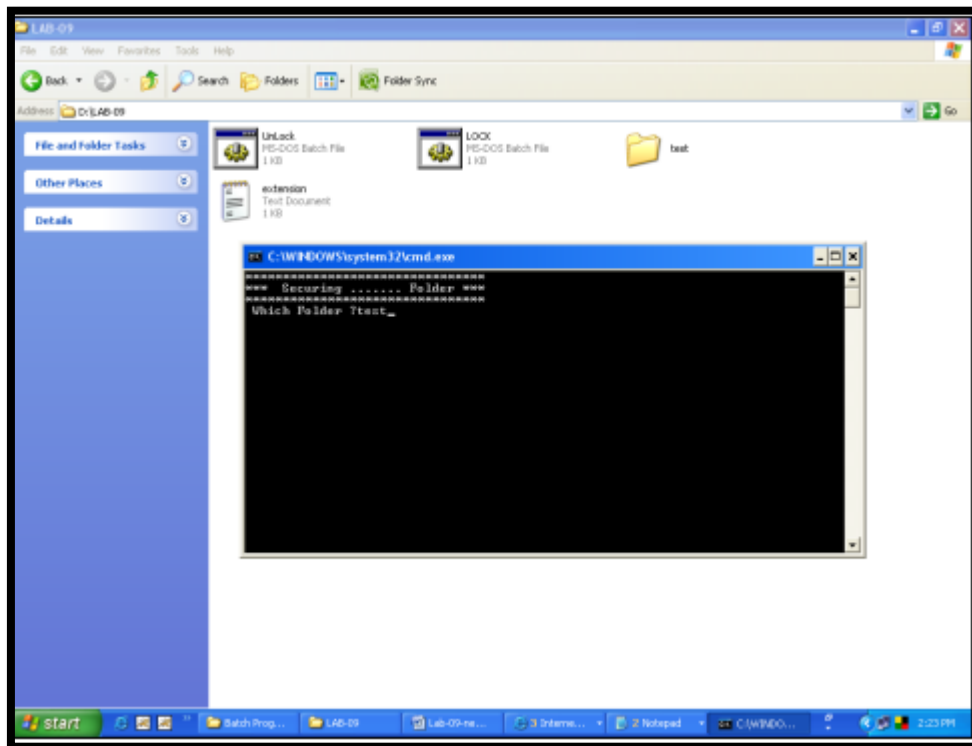
ren %folder%.{20D04FE0-3AEA-1069-A2D8-08002B30309D} tes



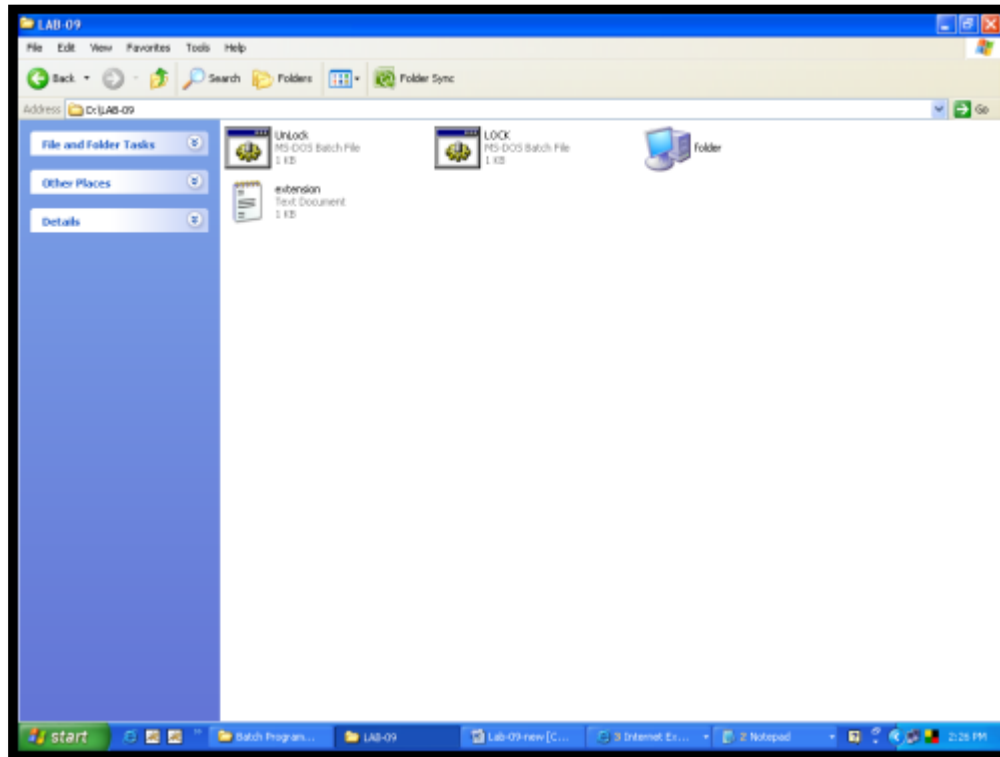
5. Now double click on "LOCK.bat"



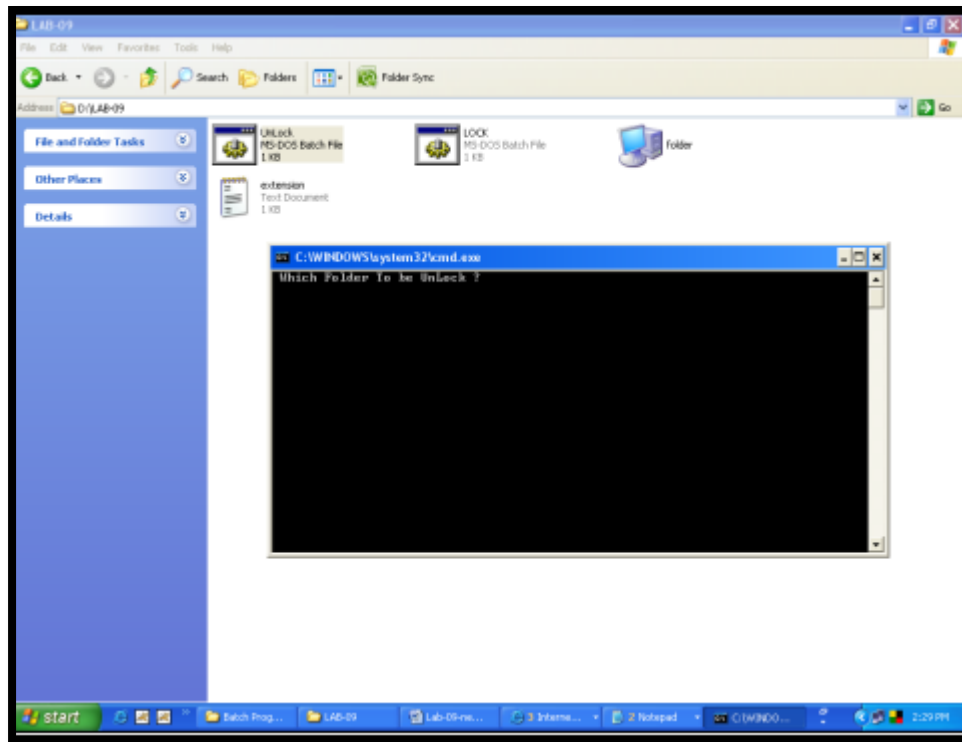
6. Write the Name of folder that you want to secure. In our scenario it is “test” and then press Enter



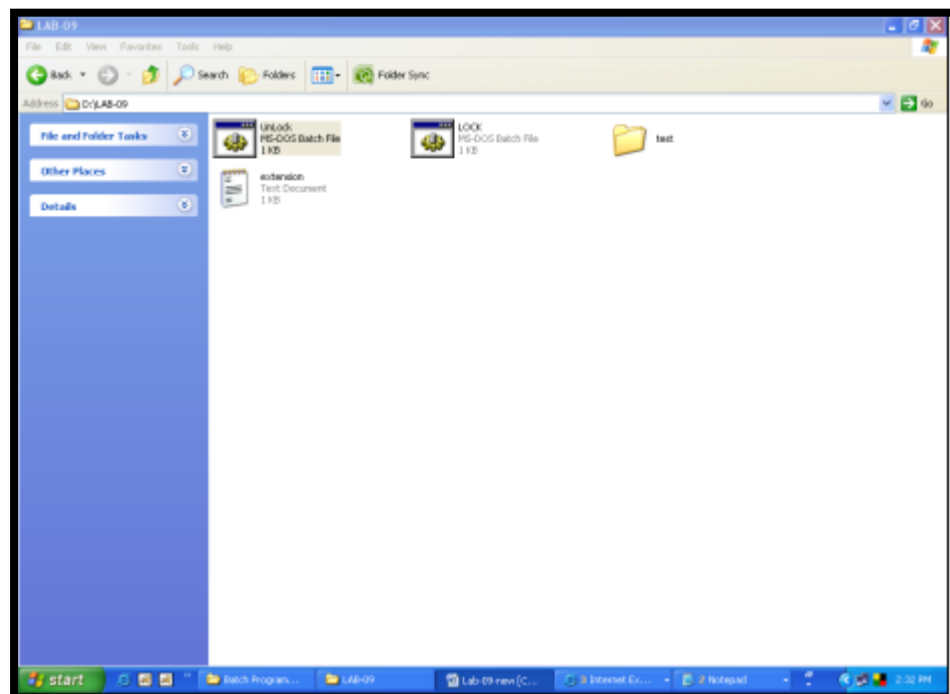
7. After pressing enter your test folder is renamed as folder and representation is converted in to MY Computer.



8. If you double click on “Folder” it directly open My Computer means the files that you put in test folder are not accessible and are being secure.
9. To retrieve your test folder double click on Unlock.bat file



10. Write the name of folder like in our scenario is “Folder” and then press Enter.



Exercise

1. In the DOS Batch Programming what does @ECHO OFF means.

2. What Command you use to take input in Batch Programming.

3. Write the command to rename NED.bat file to BCIT.bat file.

Lab Session 07

Object: Data Encryption Standard (DES)

Theory:

CrypTool offers two variations of the DES cipher, electronic codebook (ECB) and cipher-block chaining (CBC), these are both known as block cipher modes of operation. DES breaks down a plaintext message into 64 bit sized blocks; these are then converted to 64 bit blocks of ciphertext. Conventionally these ciphertext blocks have no relation to one another's encoding; this is the former of the two methods, ECB. There is however a weakness with this method, the DES encodes two identical plaintext blocks as two identical ciphertext blocks, in a message where there could be large repetition of a 64 bit block, say an image, this could reveal patterns in the ciphertext message and ultimately lead to it being deciphered. CBC is a method that overcomes this by combining each block with its preceding block with an exclusive-OR logic gate. A factor that can make this cipher at times impractical is that the whole message must be transmitted error free in order for it to be comprehensibly deciphered at the receiver side. With the ECB method an error in one block will only effect the deciphering of that one block and not the entire message.

Using DES with CrypTool

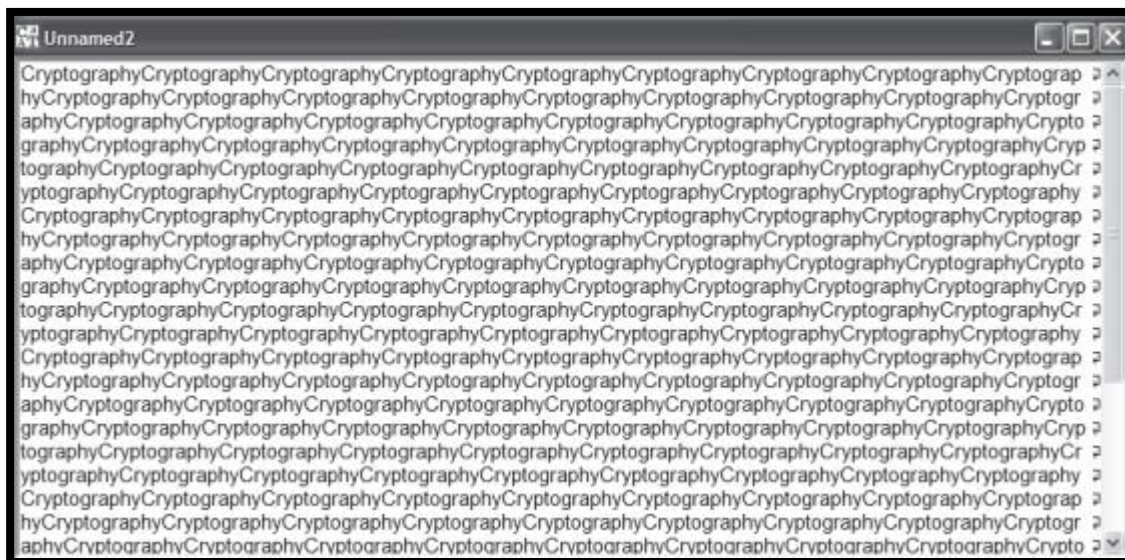
Open a new file and type a plaintext message or alternatively open the file examples/Startingexample-en.txt. Next click from the menu Encrypt/Decrypt > Symmetric (modern) > DES (ECB)... This presents a key selection window; this key must be 64 bits long, which equates to 16 hexadecimal figures. For simplicity use the default key of: 00 00 00 00 00 00 00 00

Select Encrypt and there should be presented a window showing the data encrypted in hexadecimal form and its corresponding ASCII representation. To decrypt the message again select Encrypt/Decrypt > Symmetric (modern) > DES (ECB)... Use the same key and select Decrypt, and the original message will be displayed in hexadecimal representation. Selecting View > Show as text displays it in ASCII; you may also notice some of the formatting is lost in the process or some padding is added.

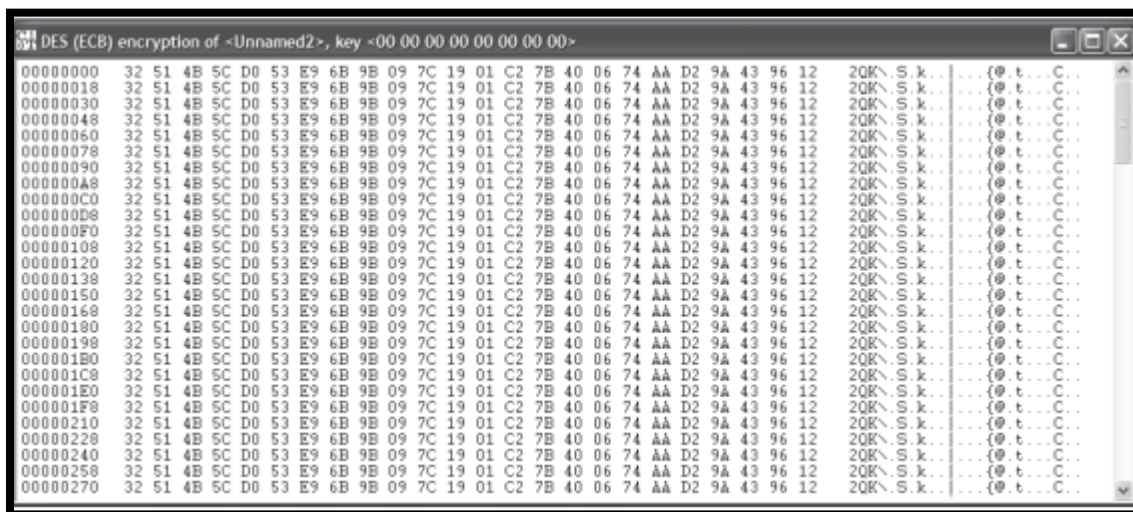
Encrypt the same message using the same process as above only selecting Encrypt/Decrypt > Symmetric (modern) > DES (CBC)... instead. Comparing the two encrypted messages, you should notice they start with the same 8 bytes, but then they are not the same but both are illegible.

Comparison of ECB and CBC

Now create a new plaintext file, choose a word and type it in, now copy and paste this word until there are about 100 repetitions. For the below example the word “Cryptography” was used:



Encrypt this first with the ECB variant of the DES cipher, there should be a notable pattern present (the word is 12 bytes long, 8 bytes form a block, all 3 blocks you see the same pattern):



A pattern like this could potentially help a cryptanalyst decipher the data. Next encrypt the same message using the CBC method.

```
DES (CBC) encryption of <Unnamed2>, key <00 00 00 00 00 00 00 00>
00000000 32 51 4B 5C D0 53 E9 6B E8 7D 74 2C 44 7D 9F 2B A9 5F BC 03 F1 20K\S.k.)t.D}+.
00000015 5F D2 76 00 C7 29 8A 2E 3F B7 E6 8B 9C 5B 73 7C 75 1F 5B B1 2B -.v.).?...[s]u[.+
0000002A 6A D6 56 18 35 E6 82 58 BC 51 90 15 AE C4 A7 4F BF 1A B6 6A C7 j.V.5..X.Q....O...j.
0000003F 98 02 95 66 68 F8 8F 1B 77 8B 37 D7 1B 76 06 A8 25 9A A0 FC B6 ...fh...v.7..v..%...
00000054 92 FC AC B7 F8 F7 0C 98 2E FD 1E 81 9F 97 06 3B 0C 4E 43 81 EB ...NC.
00000069 26 6E 6B 51 45 5A 38 F7 FE F0 88 91 13 2F 73 91 86 4D 76 A0 F3 &nkQEz8...../s..Mv.
0000007E 21 14 A1 68 EE 70 F4 A5 D7 42 65 AE 1C 23 9F 4B B8 57 6C D2 37 !..h.p...Be...#K.Wl.7
00000093 CA A3 3F 45 AE A8 02 78 28 4C 97 68 A3 1E 0A 0C BA DB CD B5 9F ...?E...x(L.h.....
000000A8 80 4E 71 8C 95 18 2D 61 6F 1E 41 0F 1A 28 02 15 37 F9 08 DF 1F ..Nq...-ao.A..(..7...
000000BD 63 EA 47 05 6A 2D 01 BA 66 69 85 5F D0 A0 77 A1 FD BA 3C DF DA c.G.j-...fi...w...<...
000000D2 AB D0 5E DB DD 1B 9B 51 E2 7B 8B 4C 10 95 57 4D 54 97 41 5C 46 ...^...Q.{...L..WHT A\F
000000E7 9F 15 75 E6 95 BB 93 A4 97 47 88 E8 80 7C 40 3A 92 7D B7 70 04 ..u.....G...[0:..).p
000000FC 50 99 B5 FB 1A 2B 05 9A 81 C6 5C 82 82 FB EA 21 1B 48 A9 1C D8 P...+.....\...[...H...
00000111 09 73 D7 50 7C A7 9F AD 66 09 DB 2D 9E 5B F9 FA A4 F0 BF 1E 3B ..s.P|...f...-...[...:..
00000126 97 DF D1 E7 5C D4 09 00 8D 40 F4 0B 3B 8D 10 B2 C7 20 4B DF EA ...\. ...@... ..K...
0000013B AE 64 67 23 88 CC 4A 02 1C 4C 87 2C 27 FE 55 2A B7 21 12 DC 7E ..dg#..J..L...U*..!..~
00000150 81 15 EE CE 6C 90 C7 97 EB 1E 79 06 06 C8 EF DF 54 49 45 E1 0B ...l...y... ..TIE..
00000165 66 CD 6E AD 88 AD DA 57 40 8D 5F E6 52 3A ED 91 72 19 E8 E5 8A ..f.n...U@...R...r...
0000017A BB CE 5A 43 97 EA 12 1D D8 0E D6 94 68 9A 38 32 06 99 FF FB B3 ..ZC... ..h.82...
0000018F A7 59 2D 98 0F F1 DB BD 79 D1 CD AE BB 91 1F 57 8A C4 69 D9 BC ..Y... ..y... ..U...i...
000001A4 85 A4 BA 2A 0D 69 1F F5 AB 49 BE 69 AB 10 02 E0 66 7F F1 89 D4 ...*..i...I..i...f...
000001B9 C6 CC C1 E9 2D A6 5B D5 F2 41 2B 88 F7 AE 7D 25 FD CA 7F 9F 75 ...-...[...A+...}%...u
000001CE B7 97 15 11 67 98 F5 01 50 17 A9 5F 52 43 61 20 E4 F4 34 59 43 ...g...P...RCa...4YC
000001E3 37 44 5D C2 CF 51 AF B3 48 D9 B0 C6 EF CC 3F 7A C4 22 F5 EE 68 7D].Q..H... ..?z...h
000001F8 6F D4 7C 9A 91 98 B9 05 4B DC C9 21 6F 97 F8 1E FC AF BB 4A 90 ..o.|...K...lo...J...
0000020D 50 D9 53 D8 16 51 4A 5C FF B1 66 F2 84 0F 50 D8 45 67 1B 13 01 P.S..QJ\..f...P.Eg...
00000222 EA 85 99 82 AC 6A 2E 09 30 6A 34 DA E6 A4 EF F8 A8 76 37 5A 15 ...j...0j4... ..v7Z..
```

As shown with the above example there should be no obvious pattern present making this a much stronger form of DES encryption.

Exercise

1. What are ECB and CBC and their purpose? How do they differ?

2. Why are the following keys considered to be weak keys of DES. Think about applying these keys to cryptool preferably trying to encrypt text with these keys twice.

K1= 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

K2= F E F E F E F E F E F E F E F E

K3= 1 F 1 F 1 F 1 F 0 E 0 E 0 E 0 E

K4= E 0 E 0 E 0 E 0 F 1 F 1 F 1 F 1

3. Why are the following keys considered to be weak keys of DES. Think about applying these keys to cryptool preferably trying to encrypt text with these keys twice.

- K1= 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

- K2= F E F E F E F E F E F E F E F E

- K3= 1 F 1 F 1 F 1 F 0 E 0 E 0 E 0 E

- K4= E 0 E 0 E 0 E 0 F 1 F 1 F 1 F 1

Lab Session 08

Object: Diffie Helman Key Exchange Method

Theory:

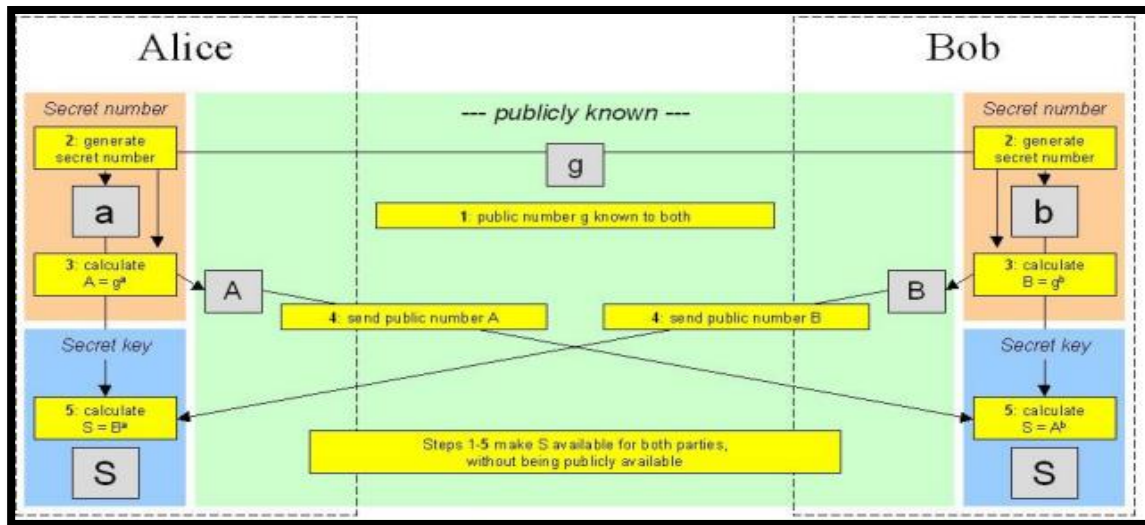
The Diffie-Hellman Key Exchange is a means for two parties to jointly establish a shared secret over an unsecure channel, without having any prior knowledge of each other.

The symmetric encryption methods require that the keys are securely transported between sender and receiver. Each pair of sender and receiver shares a common secret key. For technical reasons and to evade the huge efforts for mass direct key exchange of the different keys, other methods have been invented, which depend on a public key agreement. The session key is generated interactively and unknown by others for each session.

Besides the asymmetric encryption methods multiple methods for key exchange exist, which make use of public and private keys too. The most famous and even today implemented method was developed by Diffie and Hellman.

Diffie-Hellman Key Exchange Protocol

The Diffie-Hellman key exchange protocol was developed by Whitfield Diffie, Martin E. Hellman and Ralph Merkle in 1976. CrypTool contains a visualization of the Diffie-Hellman key exchange protocol. You can access it via the menu item Individual Procedures / Diffie-Hellman Demonstration. Despite communicating via a non-secure line, the protocol enables two or more parties to negotiate on a secret key which is solely known to the participating parties. For better comprehension the two participating parties will be described as Alice and Bob. The following drawing is to give a quick overview of the Diffie-Hellman key exchange protocol:



The secret and common key, often described as session key, depends on the following parameters:

- Public parameters (p and g)
- Secret of Alice
- Secret of Bob
- Shared key of Alice
- Shared key of Bob

The protocol is based on the discrete logarithm problem, that is making use of a so-called one way-function property: Whereas the computation of a mathematical one-way-function is of low complexity, the inverse operation is extremely hard to accomplish.

The creation of the session key at the end of the protocol is based on such a one-way-function:

Even if a potential attacker has knowledge of the shared keys of both Alice and Bob, it is hardly possible for him to compute the session key; depending on the key length, it can easily turn out as impossible for the attacker to calculate the session key.

Increasing the prime module p directly affects the complexity to compute the session key. Anyhow, the additional expenses to compute the exponential function are bearable for Alice and Bob.

Contrary to the situation of Alice and Bob, the attacker needs to compute the discrete logarithm, which is the inversion of the exponential function. This effort increases much more than the effort necessary for the two communication partners.

For this reason, the safety of the Diffie-Hellman key exchange protocol depends essentially on the size of the prime module p . In practical prime module numbers are considered safe if their bit length has at least 1024 bit, which complies with a 300-digit decimal number. Besides the prime module p , the secret numbers of Alice and Bob need to be of a certain length in order to prevent the attacker from correctly guessing the session key by chance. Thus, in practice secrets should be at least 80 bit long (25-digit decimal number).

Remark 1: The key exchange protocol according to Diffie-Hellman is NOT an encryption method; it is merely used to agree on a secret and common key.

Remark 2: The Diffie-Hellman protocol is vulnerable against "Man-In-The-Middle" attacks:

The attacker attempts to personate as Alice for Bob and at the same time to personate as Bob against Alice (the attacker stands in the middle of the communication between Alice and Bob). If he is successful then he obtains via the Diffie-Hellman protocol a common secret key with Alice and a second common secret key with Bob.

Exercise

1. Calculate the Symmetric Key using Diffie-Hellman Key Exchange method while considering the following values $g=4$, $p=13$ and $a=10$, $b=8$. Verify the Cryptool generated key with the manual calculations.

2. How Man in the Middle Attach is possible in Diffie-Hellman Key Exchange method.

Lab Session 09

Object: Network Traffic Processing and Analysis in promiscuous mode

Theory:

Computers and network devices communicate with each other by sending or receiving information over tiny bundles of electronic signals called packets. This is how a packet is represented while on wire. In memory, a packet is represented by 1s and 0s. Flow of different packets to and from different computers over the network is said to constitute network traffic. Sniffers are tools which are used to capture and process network traffic.

- Wireshark
- Ipgrab
- Ettercap
- Dsniff
- Tcpdump

All are free and open source

PROCEDURE

Apply the following commands in wireshark and observe the result.

- **\$man tshark**

You must be root to run tshark and most other sniffers

- **\$tshark -v**

Prints version and other information

Can be used without root

- **\$tshark -h**

Prints help summary

Can be used without root

- **#tshark -D**

Lists all the interfaces available on the host

- **#tshark -i eth0**

Capture on interface eth0

- **#tshark -V**

Verbose output

- **#tshark -c 100**

Stop after capturing 100 packets

Default is infinite

Use ctrl-c to terminate

- **#tshark -r outputfile.pcap**

Read packets from the outputfile.pcap

root not needed

Useful for experimentation

- **#tshark -r outputfile.pcap > textfile.txt**

You can create a text file of the output

- **#tshark -x**

Also output in hex and ASCII

- **#tshark -n**

By default names are resolved

It generates noise on the network

-n disables name resolution

Avoids detection from anti sniffers

- **#tshark -p**

Don't put the interface in promiscuous mode

Promiscuous mode

A number of packets arrive at an NIC, By default, the NIC chooses only those packets whose destination MAC address matches its own i.e., it captures only packets which are meant for the NIC Foreign packets are discarded.

When in promiscuous mode, the NIC captures all the packets regardless of the destination MAC address in the packet. This way it also captures which are meant for other NICs i.e., other computers. In promiscuous mode, the NIC selects and captures all the packets. None are discarded.

A sniffer running in promiscuous mode in a non-switched LAN will capture ALL the traffic of the network including those of other computers.

In a LAN using a switched hub, a packet is sent only to the NIC for which it is meant. A packet is not broadcast to all the host as in case of a non-switched hub. A sniffer running even in promiscuous mode will not be able to capture traffic of other computers unless traffic is redirected in some way.

Filter Expressions

By default a sniffer captures all the packets arriving on the interface. To capture only selective packets we use filter expressions with a sniffer. Filter expressions are characteristics are sniffers which are built using libpcap. Most of the sniffers are built using libpcap.

- **# tshark host 172.16.30.1**

This expression will captures only packets which are meant for host 72.16.30.1

- **# tshark src host 172.16.30.1**

This expression will capture packets which originates from host 172.16.30.1

- **# tshark tcp**

This will only capture packets with TCP header in them. Rest of the packets are not decoded. Similarly, you can use „udp“, „ip“, „arp“, „icmp“, etc. instead of tcp.

- **#tshark src host 172.16.30.1 and dst port 80**

This captures all outgoing HTTP traffic from host 172.16.30.1. An HTTP server uses port 80 to listen to HTTP traffic. A browser may use any ephemereal port above 1024 to connect to HTTP server.

- **#tshark ether src host 00:0c:f1:dc:6a:a7**

This captures packets from the host whose MAC address is as given.

- **#tshark net 172.16**

This will capture all the traffic meant for network beginneing with 172.16

You can also use „src net“ and „dst net“.

- **#tshark less 500**

This will capture all the packets whose length is less than or equal to 500 bytes.

Similarly you can use „greater 500“.

- **Filter expressions can be combined using and, or, not operators.**
- **#tshark “host 172.16.30.1 and (host 172.16.30.5 or host 172.16.30.10)”**

This will capture any communication of 172.16.30.1 with either 172.16.30.5 or 172.16.30.10. Rest of the traffic will be discarded .Enclose complex filter expressions in “ ”.

Exercise

1. What is Promiscuous Mode?

2. Write the expression that will captures only packets which are meant for host 72.16.30.1.

3. Describe this filter “tshark src host 172.16.30.1 and dst port 80” in your own words.

Lab Session 10

Object: ARP Poisoning to conduct Man In The Middle (MITM) Attack

Theory:

An IP address is used to find the link layer address, such as the MAC address, using the Address Resolution Protocol (ARP), a communication protocol. The ARP protocol is a fairly straightforward protocol that lacks any security measures. One frequent assault on the ARP protocol is the poisoning of the cache. Attackers can trick the victim into adopting counterfeit IP-to-MAC mappings by employing such a tactic. As a result, packets from the victim may be sent to the machine with the falsified MAC address, potentially opening the door for man-in-the-middle attacks.

Network Setup

You need two machines in order to complete this lab. The first computer is used for the assault, while the second serves as the victim client. To configure as a client and an attacker computer, use a virtual machine running Ubuntu.

- Install Ettercap on 'Ubuntu': Ettercap is another powerful tool often used for man-in-the-middle attacks.
- \$sudo apt - get install ettercap – graphical

ARP Poisoning Attack

An essential component of the ARP protocol is the ARP cache. The ARP protocol's resolution of a mapping between a MAC address and an IP address results in caching of the result. As a result, if the mapping is already stored in the cache, the ARP protocol does not need to be repeated. However, because the ARP protocol is stateless, fraudulently generated ARP packets can readily poison the cache. Such an attack is known as The ARP cache poisoning attack.

ARP fundamentally lacks an internal mechanism for authentication, making responses vulnerable to forging. It is simple to divert traffic from a victim to oneself by sending bogus ARP responses. You can now launch a number of assaults. You could stop the traffic, which would be equivalent to a denial-of-service attack. You may sniff all of the victim's communication by listening to it and forwarding it. The traffic might potentially be altered before being sent.

You will utilise ettercap to carry out an ARP poisoning in this assignment. You'll operate ettercap using its GUI interface. You'll need two computers for this operation; one will serve as the victim client and the other as the attacker.

Step 1:

- On the client machine, ping the host machine and the attacker.
- Use “arp -n” command to check the current ARP table and confirm the MAC addresses.

Step 2:

- On the attacker machine, you need to first enable IP forwarding so that the attacker machine will act as the intermediary between the victim and the intended destination. Login as the root and execute

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

- Then, you need to modify the ettercap configuration file. Open /etc/ettercap/etter.conf, find the “Linux” section, and uncomment the 4 lines there. Now run “sudo ettercap -G” to start ettercap program.
 1. Select “Sniff → Unified sniffing”, and set the network interface (usually eth0 or eth1)
 2. Select “Hosts → Scan for hosts”, then “Hosts → Host List”. Click on the victim client’s IP address and click “Add to Target 1”.
 3. Select “Mitm → Arp poisoning”, then check “Sniff remote connections”.
 4. Select “Start → Start sniffing”

Step 3

- Check the ARP table on the client machine and describe your observation.
- Install wireshark and Start wireshark on the client machine (You can also use tcpdump). Set filters to display only ARP packets. In the display filter, type the following:
ip.src==spoofedipaddress&& tcp.port==80
- Stop wireshark on the client side and start another wireshark monitoring on the attacker machine.
- Open the web browser on the client machine and visit <http://www.villanova.edu>. Type in some keywords in the search field and press “Go”.

Step 4

- Go back to the attacker machine and check the wireshark window to see if the keywords are captured.

Step 5

- Select “Mitm → Stop mitm attack”, and “Start → Stop sniffing”. Then quit ettercap program.

Exercise

1. Examine the client machine's ARP table and explain what you observed.

2. How many responses did the client receive for a certain ARP request?

3. Are the responses being spoofed? Are there valid replies? How can you identify them?

4. According to the trace, describe how this attack works.

Lab Session 11

Object: NMAP - A Stealth Port Scanner

Theory:

Nmap is a free, open-source port scanner available for both UNIX and Windows. It has an optional graphical front-end, ZENmap, and supports a wide variety of scan types, each one with different benefits and drawbacks.

The two basic scan types used most in Nmap are

1. Basic Scan Types [-sT, -sS]
2. TCP connect() scanning [-sT].

SYN scanning (also known as half-open, or stealth scanning) [-sS].

TCP connect() Scan [-sT]

These scans are so called because UNIX sockets programming uses a system call named connect() to begin a TCP connection to a remote site. If texttt{connect()} succeeds, a connection was made. If it fails, the connection could not be made (the remote system is offline, the port is closed, or some other error occurred along the way). This allows a basic type of port scan, which attempts to connect to every port in turn, and notes whether or not the connection succeeded. Once the scan is completed, ports to which a connection could be established are listed as textit{open}, the rest are said to be closed.

This method of scanning is very effective and provides a clear picture of the ports you can and cannot access. If a connect() scan lists a port as open, you can definitely connect to it - that is what the scanning computer just did! There is, however, a major drawback to this kind of scan; it is very easy to detect on the system being scanned. If a firewall or intrusion detection system is running on the victim, attempts to connect() to every port on the system will almost always trigger a warning. Indeed, with modern firewalls, an attempt to connect to a single port which has been blocked or has not been specifically "opened" will usually result in the connection attempt being logged. Additionally, most servers will log connections and their source IP, so it would be easy to detect the source of a TCP connect() scan.

For this reason, the TCP Stealth Scan was developed.

- **SYN Stealth Scan [-sS]**

TCP Connection -- Three way handshake

- **Flag**

SYN (Synchronize).

ACK (Acknowledge).

FIN (Finished) and

RST (Reset).

SYN or Stealth scanning makes use of this procedure by sending a SYN packet and looking at the response. If SYN/ACK is sent back, the port is open and the remote end is trying to open a TCP connection. The scanner then sends an RST to tear down the connection before it can be established fully; often preventing the connection attempt appearing in application logs. If the port is closed, an RST will be sent. If it is filtered, the SYN packet will have been dropped and no response will be sent. In this way, Nmap can detect three port states - open, closed and filtered. Filtered ports may require further probing since they could be subject to firewall rules which render them open to some IPs or conditions, and closed to others.

PROCEDURE

Modern firewalls and Intrusion Detection Systems can detect SYN scans, but in combination with other features of Nmap, it is possible to create a virtually undetectable SYN scan by altering timing and other options. Apply the following commands on NMAP and observe the results.

- **Ping Scan [-sP]**

This scan type lists the hosts within the specified range that responded to a ping. It allows you to detect which computers are online, rather than which ports are open.

UDP Scan [-sU]

Scanning for open UDP ports is done with the -sU option. With this scan type, Nmap sends 0-byte UDP packets to each target port on the victim. Receipt of an ICMP Port Unreachable message signifies the port is closed, otherwise it is assumed open.

- **IP Protocol Scans [-sO]**

The IP Protocol Scans attempt to determine the IP protocols supported on a target. Nmap sends a raw IP packet without any additional protocol header (see a good TCP/IP book for information about IP packets), to each protocol on the target machine. Receipt of an ICMP Protocol Unreachable message tells us the protocol is not in use, otherwise it is assumed open.

- **Version Detection [-sV]**

Version Detection collects information about the specific service running on an open port, including the product name and version number. This information can be critical in determining an entry point for an attacker.

- **OS Fingerprinting**

The -O option turns on Nmap's OS fingerprinting system. Used alongside the -v verbosity options, you can gain information about the remote operating system and about its TCP Sequence Number generation

- **IPv6**

The -6 option enables IPv6 in Nmap (provided your OS has IPv6 support). Currently only TCP connect, and TCP connect ping scan are supported.

- **Verbose Mode**

Highly recommended, -v

Use -v twice for more verbosity. The option -d can also be used (once or twice) to generate more verbose output.

Exercise

1. What is the difference between a TCP-connect scan and a SYN scan?

2. What is the purpose of the `-sP` command line switch? Show practical demonstration.

3. What is the purpose of the `-sS` command line switch? Show practical demonstration.

4. What command would you issue to scan for computers running web servers?

Lab Session 12

Object: RSA (Rivest–Shamir–Adleman)

Theory:

The most well-known asymmetric cryptosystem is the RSA algorithm, named after its authors, Ron Rivest, Adi Shamir and Leonard Adleman. A special characteristic of the RSA cryptosystem is that extensive calculations are necessary to generate the RSA key before RSA encryption or decryption can take place. First of all the RSA parameters p , q , N and the Euler number $\phi(N)$ are calculated:

Choose two different prime numbers p and q at random and calculate the so-called RSA modulus $N = pq$. The Euler number $\phi(N) = (p-1)(q-1)$ is calculated from the prime factors p and q .

In a second step, the public RSA exponent e is determined and from this together with $\phi(N)$ the secret RSA exponent d is calculated:

Choose the number e : $1 < e < \phi(N)$, with the property that e is relatively prime to $\phi(N)$. An especially popular value for e is $2^{16}+1 = 65537$, as in most cases this is co-prime to $\phi(N)$ and is especially well-suited for square and multiply exponentiation resulting in a very fast public key operation.

The secret exponent $d = e^{-1} \bmod \phi(N)$ is calculated from the public exponent e as the multiplicative inverse modulo $\phi(N)$.

The RSA Cryptosystem

RSA using the private and public key -- or using only the public key

☒ Choose two prime numbers p and q . The number $N = pq$ is the public RSA modulus and $\phi(N) = (p-1)(q-1)$ is the Euler number. Public key e is coprime to $\phi(N)$. The private key $d = e^{-1} \bmod \phi(N)$ is calculated from this.

☐ For the purpose of data encryption or certificate checking it is sufficient to enter the public RSA parameters: the RSA modulus N and the public key e .

Prime number entry

Prime number p : 22171

Prime number q : 57347

Generate prime numbers...

RSA parameters:

RSA modulus N : 1271440337 (public)

$\phi(N) = (p-1)(q-1)$: 1271360820 (secret)

Public key e : 65537

Private key d : 502185293

Update parameters

In the dialog "The RSA Cryptosystem" you can enter the prime numbers p and q directly. If you do not enter any prime numbers for p or q , an error message will be displayed. By clicking on the Generate prime numbers button, you can search for random prime numbers from within specified number intervals. The

Update parameters button is used to calculate the secret exponent d from e and $\phi(N)$ as described above. If e is not co-prime to $\phi(N)$, an error message will be displayed, asking you to choose a new number for the public exponent e .

After successful generation of the RSA key, the asymmetric RSA key pair is displayed:

(N,e) is the public key and (N,d) is the secret key. An RSA key bit length is calculated from the number of bits of the binary representation of the RSA modulus N .

After the key has been generated, anyone can encrypt a message with the RSA algorithm using the public key (N,e) , but only the owner of the secret key (N,d) can decrypt the message again with the RSA algorithm.

RSA encryption of messages

After the RSA key generation, the modulus N and the RSA key e can be published. Anyone could encrypt a message to the owner of the secret RSA key or check a digital signature.

If you generate the RSA key by your own in the RSA demo dialog the public RSA parameter appears already in the dialog. In the other case you enter the public RSA parameter. The message can now be encrypted.

To this end, enter the "plaintext" message M that is to be encrypted. For example:

"Verkaufen Sie am 14.07.00 meine Aktienpakete, und kaufen Sie mir dafür eine Südsee Insel" [Sell my shares on 14 July 2000 and use the proceeds to buy me a South Sea Island].

When you click on Encrypt, first of all the message is converted to numbers. For this purpose, message M is split up into blocks of k characters ($M = M[1] \# M[2] \# \dots \# M[j]$), whereby $k \cdot 8$ is always smaller than the bit length of RSA modulus N . The message blocks ($M[i]$, $i = 1, 2, \dots, j$) are converted for encryption into numbers ($m[i]$, for $i = 1, 2, \dots, j$) and encrypted with the public RSA key (N,e) according to the following formula:

$$c[i] = m[i]^e \pmod{N} \text{ for } i = 1, 2, \dots, j.$$

Because the public key (N,e) is available to everyone and the secret key (N,d) remains confidential, we can establish that

Anyone can encrypt a message with the public key. But only the owner of the secret key can decrypt the message again

The screenshot shows a dialog box titled "RSA encryption using e / decryption using d". It has two radio buttons for "Input as": "text" (selected) and "numbers". There is a button labeled "Options for alphabet and number system...". Below the input type, there is a text field for "Input text" containing the German sentence "Verkaufen Sie am 14.07.00 meine Aktienpakete, und kaufen Sie mir dafür eine Südsee Insel". Below this, a line of text states: "The Input text will be separated into segments of Size 3 (the symbol '#' is used as separator)." followed by the segmented text: "Ver # kau # fen # Si # e a # m 1 # 4.0 # 7.0 # 0 m # ein # e A # kti # enp # ake # te, # un # d k # auf # en". Below that, a line says "Numbers input in base 10 format." followed by a long string of numbers separated by '#'. Below this, a line says "Encryption into ciphertext $c[i] = m[i]^e \pmod{N}$ " followed by another long string of numbers separated by '#'. At the bottom, there are three buttons: "Encrypt", "Decrypt", and "Close".

The conversion of the plaintext M to numbers $m = m[1] \# m[2] \# \dots \# m[j]$ and the encrypted version $c = c[1] \# c[2] \# \dots \# c[j]$ are displayed in the two lines below the data input field.

The character # does not have any significance here: it is simply a visual separator between different number blocks to be encrypted or decrypted.

You can specify the block length k by clicking on Options for the alphabet and the number system. In the dialog accessed by this option you can also determine the basis for representation of numbers in RSA encryption and you can experiment with the RSA variant from the story Dialogue of the Sisters.

Instead of entering text you can also specify in the options that you wish to directly encrypt a sequence of plaintext numbers $m = m[1] \# m[2] \# \dots \# m[j]$:

The screenshot shows the same dialog box as before, but with the "numbers" radio button selected. The "Input text" field is empty. Below the input type, a line states: "Plaintext coded in numbers of base 10." followed by a long string of numbers separated by '#'. Below that, a line says "Encryption into ciphertext $c[i] = m[i]^e \pmod{N}$ " followed by another long string of numbers separated by '#'. The "Encrypt", "Decrypt", and "Close" buttons are still at the bottom.

RSA decryption of a message

Only if you are the owner of the secret RSA key d you are able to decrypt messages which are encrypted with the corresponding public key. If you just have access to the public RSA parameter N and e you can try to break the RSA key (compute the private key, works only for short keys) via the factorization attack.

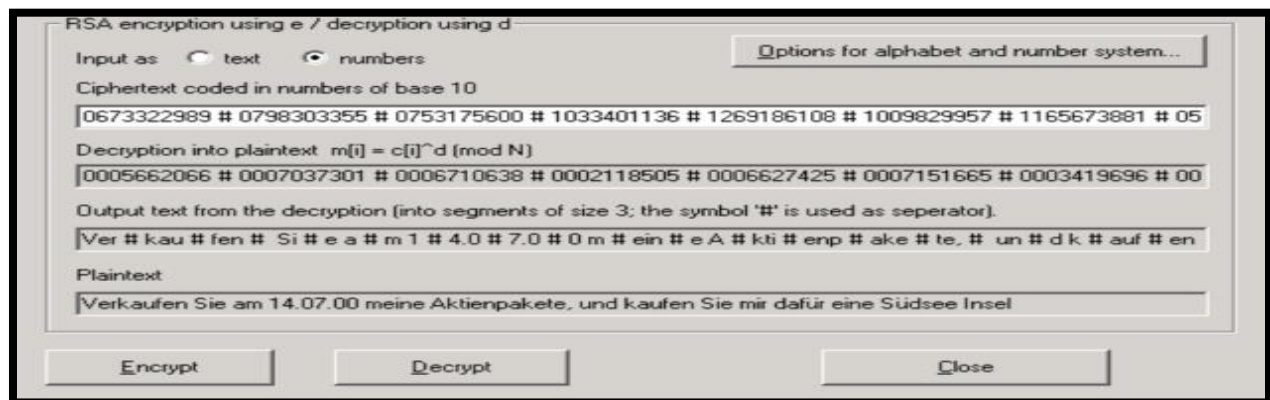
To decrypt RSA-encrypted messages, enter the numeric code of the encrypted message. When you click on **Decrypt**, the entered ciphertext

$$c = c[1] \# c[2] \# \dots \# c[j]$$

is decrypted using the secret key (N, d) according to the following formula:

$$m[i] = c[i]^d \bmod N \text{ for } i = 1, \dots, j.$$

The numbers $m = m[1] \# m[2] \# \dots \# m[j]$ are then transformed to the original text message M undoing the block splitting:



Exercise

1. Perform RSA encryption and decryption. The parameters used here are small. Verify your results with cryptool?
 - Choose two distinct prime numbers, such as $P=61$ $q=53$
 - Compute $n = pq$ giving $n=?$
 - Compute the totient of the product as $\phi(n) = (p - 1)(q - 1)$
 - Choose any number $1 < e < 3120$ that is co-prime to 3120. Choosing a prime number for e leaves us only to check that e is not a divisor of 3120.
 - Compute d , the modular multiplicative inverse of $e \pmod{\phi(n)}$ yielding $d=?$

Lab Session 13

Object: Attack on RSA encryption with short RSA modulus

Theory:

The analysis is performed in two stages: first of all the prime factorization of the RSA modulus is calculated using factorization, and then in the second stage the secret key for encryption of the message is determined. After this, the cipher text can be decrypted with the cracked secret key.

We will figure out plaintext given

RSA modulus $n = 63978486879527143858831415041$

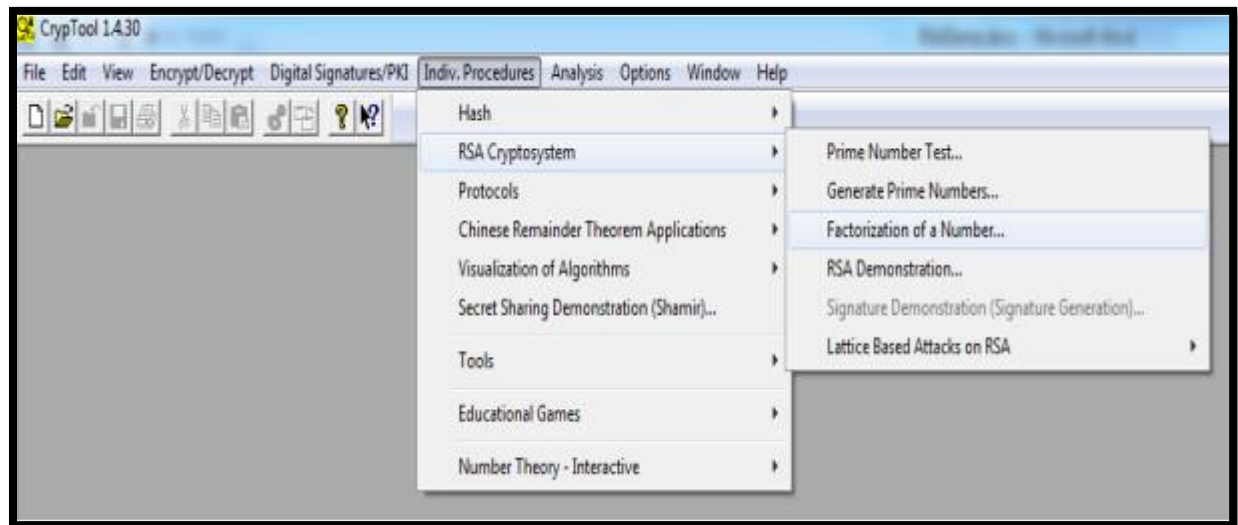
Public exponent $e = 17579$

Cipher text = 45411667895024938209259253423, 16597091621432020076311552201,

46468979279750354732637631044, 32870167545903741339819671379

1. Factorization of the RSA modulus with the aid of prime factorization.

To break down the natural number, select menu **sequence Indiv. Procedure/RSA Cryptosystem / Factorization of a Number**.



2. The two components of the public key is

RSA modulus $n = 63978486879527143858831415041$

Public exponent $e = 17579$

Enter $n=63978486879527143858831415041$ as input and click **Continue**.

The screenshot shows a window titled "Factorization of a Number". It has three main sections. The top section, "Algorithms for factorization", contains a list of algorithms with checkboxes: Brute-force, Brent, Pollard, Williams, Lenstra, and Quadratic sieve, all of which are checked. The middle section, "Input", contains a text box with the number "63978486879527143858831415041" and a label "Enter the number to be factorized:". The bottom section, "Factorization (stepwise)", contains a "Continue" button. Below this, the "Factorization" section shows the result: "The factorization is represented in the format <z1^a1 * z2^a2 * ... * zn^an>. Composite numbers are highlighted in red." It also shows "Last factorization through: Pollard" and "Found 2 factors in 0.261 seconds." The "Factorization result:" is displayed in a text box as "145295143558111 * 440334654777631". There are "Details" and "Close" buttons at the bottom of the window.

It is interesting to see which procedure broke down the RSA modulus the fastest.

3. Calculate the secret key d from the prime factorization of n and the public key e :

With the knowledge of the prime factors $p = 145295143558111$ and $q = 440334654777631$ and the public key $e = 17579$, we are in a position to decrypt the ciphertext.

4. Open the next dialog box via menu selection **Indiv. Procedure/RSA Cryptosystem/RSA Demonstration:.**
5. Enter $p = 145295143558111$ and $q = 440334654777631$ and the public key $e = 17579$.
6. Click on Alphabet and number system options and make the following settings:

Alphabet options: **Specify alphabet**

RSA variant: **Normal**

Method for coding a block into number: **Number system**

Block length: **14**

Number system: **Decimal**

The screenshot shows a dialog box titled "RSA Demonstration Options" with a close button (X) in the top right corner. The dialog is organized into several sections with expandable/collapsible headers:

- Alphabet options:** Contains two radio buttons: "All 256 ASCII characters" and "Specify alphabet:". The "Specify alphabet:" option is selected. To the right of these buttons is a label "Number of characters:" followed by the value "27". Below the radio buttons is a text input field containing the string "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
- RSA variant:** Contains two radio buttons: "Normal" and "Dialogue of the Sisters". The "Normal" option is selected.
- Method for coding a block into numbers:** Contains two radio buttons: "b-adjc" and "Number system". The "Number system" option is selected.
- Block length:** Contains a text area with the following text: "The number of characters that are encrypted with each RSA operation. The maximum size of a block is limited by the bit length of the modulus N, the number of characters in the alphabet, and the encoding method." Below this text is a label "Block length in characters:" followed by a text input field containing the value "14" and a note "(Maximum block length 14 characters)".
- Number system:** Contains a text area with the following text: "The numbers for encryption and decryption will be represented in the following radix:". Below this text are four radio buttons: "Decimal", "Binary", "Octal", and "Hexadecimal". The "Decimal" option is selected.

At the bottom of the dialog are two buttons: "OK" and "Cancel".

7. Enter the following cipher text in the input text field. And click Decrypt button.

45411667895024938209259253423,

16597091621432020076311552201,

46468979279750354732637631044,

32870167545903741339819671379

The screenshot shows the 'RSA Demonstration' window. It has a title bar with a close button. The main area contains several sections:

- Instructions:** Two radio buttons. The first is selected: 'Choose two prime numbers p and q. The composite number $N = pq$ is the public RSA modulus, and $\phi(N) = (p-1)(q-1)$ is the Euler totient. The public key e is freely chosen but must be coprime to the totient. The private key d is then calculated such that $d = e^{-1} \pmod{\phi(N)}$ '. The second option is 'For data encryption or certificate verification, you will only need the public RSA parameters: the modulus N and the public key e.'
- Prime number entry:** Two text boxes for 'Prime number p' (145295143558111) and 'Prime number q' (440334654777631). A 'Generate prime numbers...' button is to the right.
- RSA parameters:** Four text boxes: 'RSA modulus N' (63978486879527143858831415041) labeled '(public)', ' $\phi(N) = (p-1)(q-1)$ ' (63978486879526558229033079300) labeled '(secret)', 'Public key e' (17579), and 'Private key d' (10663687727232084624328285019). An 'Update parameters' button is to the right.
- Encryption/Decryption options:** 'Input as' with radio buttons for 'text' and 'numbers' (selected). A button 'Alphabet and number system options...' is to the right.
- Ciphertext:** A text box containing '7091621432020076311552201 # 46468979279750354732637631044 # 32870167545903741339819671379'.
- Decryption:** A text box showing the result of the decryption: '00000000000001401202118011200 # 00000000000001421130205181900 # 000000000000011805001301'.
- Output text:** A text box showing 'NATURAL # NUMBERS # ARE MADE # BY GOD'.
- Plaintext:** A text box showing 'NATURAL NUMBERS ARE MADE BY GOD'.
- Buttons:** 'Encrypt', 'Decrypt' (highlighted with a dashed border), and 'Close'.

Check your results: **“NATURAL NUMBERS ARE MADE BY GOD”**

Exercise

In RSA, practical difficulty of factoring the product of two large prime numbers is known as the factoring problem. This is what RSA is based on. The prime factors must be kept secret. If the public key is large enough, only someone with knowledge of the prime factors can feasibly decode the message.

If we know $N = 63978486879527143858831415041$ (95 bit, 29 decimal digits) and then try this number $N = 351573870816322547022741576341143304183$ (129 bit, 39 digit). Find the factors using cryptool by going in to Indiv. Procedures -> RSA Cryptosystem -> Factorization of a number. Then enter the number and find the factors. What does this tell you about the difficulty level of finding the factors in both cases? What are the factors in both cases? What algorithm was used last to factorize in both cases? Show practical demonstration.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Lab Session 14

Object: Introduction to Firewall & Intrusion Detection System

Theory:

Firewall:

A firewall is simply the barrier that stands between a private internal network and the open Internet at its most basic level. The basic function of a firewall is to let safe traffic through while blocking harmful traffic.

A firewall is a network security tool that keeps track of and filters incoming and outgoing network traffic in accordance with previously specified security rules for an organization.

Intrusion Detection System

A monitoring system called an intrusion detection system (IDS) monitor for abnormal activity and sends out alarms when it does. A security operations center (SOC) analyst or incident responder can analyze the problem and take the necessary steps to eliminate the danger based on these notifications.

The primary distinction between an IDS and a firewall is that whereas a firewall blocks and filters traffic, an IPS/IDS detects an attack and notifies the system administrator or stops it depending on settings. Based on a set of specified rules, a firewall permits traffic.

The Snort Intrusion Detection Systems will be studied by the students in this lab. The Snort IDS, a signature-based intrusion detection system used to find network threats, will be studied by the students. Snort may also function as a straightforward packet logger. For the purposes of this lab, the students will create their own IDS rules and use snort as a packet sniffer.

Install snort as the operating system in the Ubuntu virtual machine image. You may use the installation instructions on the snort website to install the most recent version of the program. Keep in mind that each OS has different installation guidelines. The steps listed below explain how to install Snort on Linux using the program's source code.


```
Source Fedora Centos FreeBSD Windows
wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
wget https://www.snort.org/downloads/snort/snort-2.9.8.2.tar.gz

tar xvfz daq-2.0.6.tar.gz
cd daq-2.0.6
./configure && make && sudo make install

tar xvfz snort-2.9.8.2.tar.gz
cd snort-2.9.8.2
./configure --enable-sourcefire && make && sudo make install
```

Your system can lack some libraries while you install Snort. The necessary libraries must also be installed.

Configuring and Starting the Snort IDS

We must configure Snort once it has been installed. Snort's configuration data are kept in the `/etc/snort/snort.conf` file. To access the snort configurations file, you must switch to the root account.

The Snort must be started after configuration. To start the service, just run the following command.

```
/etc/init.d/snort start
```

Or

```
$ service snort start
```

Snort Rules

An IDS that uses signatures called Snort creates rules to find intrusions. The `/etc/snort/rules` directory houses all of Snort's rules. The files that contain Snort rules are seen in the screenshot below.

```
attack-responses.rules      community-web-dos.rules     policy.rules
backdoor.rules             community-web-iis.rules     pop2.rules
bad-traffic.rules          community-web-misc.rules    pop3.rules
chat.rules                 community-web-php.rules     porn.rules
community-bot.rules        ddos.rules                 rpc.rules
community-deleted.rules    deleted.rules              rservices.rules
community-dos.rules        dns.rules                  scan.rules
community-exploit.rules    dos.rules                  shellcode.rules
community-ftp.rules        experimental.rules         smtp.rules
community-game.rules       exploit.rules              snmp.rules
community-icmp.rules       finger.rules               sql.rules
community-imap.rules       ftp.rules                  telnet.rules
community-inappropriate.rules icmp-info.rules            tftp.rules
community-mail-client.rules icmp.rules                 virus.rules
community-misc.rules       imap.rules                 web-attacks.rules
community-nntp.rules       info.rules                 web-cgi.rules
community-oracle.rules     local.rules                web-client.rules
community-policy.rules     misc.rules                 web-coldfusion.rules
community-sip.rules        multimedia.rules           web-frontpage.rules
community-smtp.rules       mysql.rules                web-iis.rules
community-sql-injection.rules netbios.rules              web-misc.rules
community-virus.rules      nntp.rules                 web-php.rules
community-web-attacks.rules oracle.rules                x11.rules
community-web-cgi.rules    other-ids.rules
community-web-client.rules p2p.rules
```

A true rule from the /etc/snort/rules/web-misc.rules is seen in the screenshot below.

```
# NOTES: this signature looks for someone accessing the file "active.log" via
# a web server.  By allowing anyone on the internet to view the web access
# logs, attackers can gain information about your customers that probably
# should not be made public.
#
# This logfile is made available from the WebActive webserver.  This webserver
# is no longer maintained and should be replaced with an actively maintained
# webserver.  If converting to another webserver is not possible, remove read
# access to this file.
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-MISC active.log access"; flow:to_s
erver,established; uricontent:"/active.log"; nocase; reference:bugtraq,1497; reference:cve,2000-0642;
reference:nessus,10470; classtype:web-application-activity; sid:1851; rev:6;)
291,2 67%
```

Writing and Adding a Snort Rule

The basic snort rule will be added next. At /etc/snort/rules/local.rules, you should add your own rules. The local.rules file should now contain the line alert icmp any any -> any any. ("ICMP Packet detected," "sid:1000001," "rev. 1")

Simply put, this rule states that if an ICMP packet is discovered, an alarm will be reported. Any IP address might be the source of the ICMP packet, and the rule ID is 1000001. Choose a SID for your own rules that is bigger than 1000000. The local rules file's contents once the rule has been added are seen in the screenshot down below.

```
root@ubuntu: /home/csc5991-student
$Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
#
# LOCAL RULES
#
# This file intentionally does not come with signatures.  Put your local
# additions here.
# Added by Fengwei Zhang, you need to add more rules here.....
alert icmp any any -> any any (msg:"ICMP Packet found - Fengwei"; sid:1000001; rev:1;)
~
~
~
~
"/etc/snort/rules/local.rules" 9L, 358C 1,1 All
```

You must type the following command to restart the snort service for the rule to take effect.

```
$ service snort restart
```

or

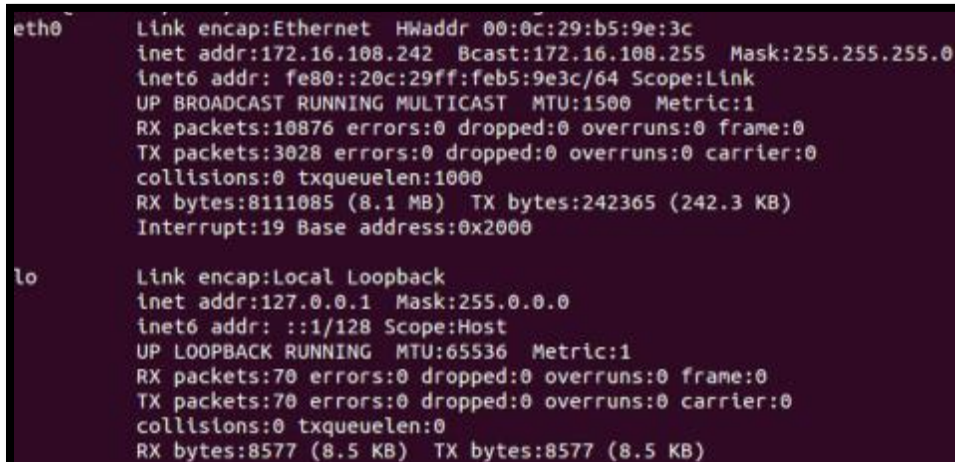
```
$ /etc/init.d/snort restart
```

Triggering an Alert for the New Rule

You simply need to send an ICMP message to the VM image where Snort is running to start an alert for the new rule. By using the following command, you must first determine the VM's IP address.

```
$ ifconfig
```

For instance, the IP address in the snapshot is 172.16.108.242, and it displays the execution result on my VM image.



```
eth0      Link encap:Ethernet  HWaddr 00:0c:29:b5:9e:3c
          inet addr:172.16.108.242  Bcast:172.16.108.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feb5:9e3c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10876 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3028 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8111085 (8.1 MB)  TX bytes:242365 (242.3 KB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:70 errors:0 dropped:0 overruns:0 frame:0
          TX packets:70 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8577 (8.5 KB)  TX bytes:8577 (8.5 KB)
```

The next step is to launch a terminal on your host. You can open a terminal in one of the two ways listed below if your host runs Windows.

1. To launch a command Prompt session using only your keyboard, click "Win-R," type "cmd," and hit "Enter."
2. Use your mouse to click "Start | Program Files | Accessories | Command Prompt" to launch a Command Prompt session.

To send ping messages to the VM once you have a terminal, simply execute the following command.

```
$ ping 172.16.108.242
```

After you send the ping messages, the alerts should be triggered, and you can find the log messages in `/var/log/snort/snort.log`. However, the `snort.log` file will be binary format. You need to use a tool, called `u2spewfoo`, to read it. The screenshot below shows the result of reading the snort alerts.

```

(Event)
  sensor id: 0      event id: 1      event second: 1460488935      event microsecond: 860268
  sig id: 1000001  gen id: 1      revision: 1      classification: 0
  priority: 0      ip source: 172.16.108.1 ip destination: 172.16.108.242
  src port: 8      dest port: 0      protocol: 1      impact_flag: 0 blocked: 0
  mpls label: 0    vland id: 0      policy id: 0

Packet
  sensor id: 0      event id: 1      event second: 1460488935
  packet second: 1460488935      packet microsecond: 860268
  linktype: 1      packet_length: 98
[ 0] 00 0C 29 B5 9E 3C 00 50 56 C0 00 08 08 00 45 00 ..)...<.PV.....E.
[ 16] 00 54 1F B4 00 00 40 01 29 E1 AC 10 6C 01 AC 10 .T....@.)...l...
[ 32] 6C F2 08 00 3C F6 0F 50 00 00 57 0D 4A E7 00 0D l...<..P..W.J...
[ 48] 1E B5 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 .....
[ 64] 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 ..... !"#$$%
[ 80] 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 &'()*+,-./012345
[ 96] 36 37 67

(Event)
  sensor id: 0      event id: 2      event second: 1460488935      event microsecond: 860297
  sig id: 1000001  gen id: 1      revision: 1      classification: 0
  priority: 0      ip source: 172.16.108.242 ip destination: 172.16.108.1
  src port: 0      dest port: 0      protocol: 1      impact_flag: 0 blocked: 0
  mpls label: 0    vland id: 0      policy id: 0

Packet
  sensor id: 0      event id: 2      event second: 1460488935
  packet second: 1460488935      packet microsecond: 860297
  linktype: 1      packet_length: 98
[ 0] 00 50 56 C0 00 08 00 0C 29 B5 9E 3C 08 00 45 00 .PV.....)...<..E.
[ 16] 00 54 FC 22 00 00 40 01 4D 72 AC 10 6C F2 AC 10 .T..."@.Mr..l...
[ 32] 6C 01 00 00 44 F6 0F 50 00 00 57 0D 4A E7 00 0D l...D..P..W.J...
[ 48] 1E B5 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 .....
[ 64] 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 ..... !"#$$%
[ 80] 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 &'()*+,-./012345

```

You can see that the SID is 1000001, and the alerts are generated by the ICMP messages.

Exercise

1. Given a network that has 1 million connections daily where 0.1% (not 10%) are attacks. If the IDS has a true positive rate of 95% what false alarm rate do I need to achieve to ensure the probability of an attack, given an alarm is 95%?

2. What is a zero-day attack?

3. Write and add another snort rule and show me you trigger it.

- a. The rule you added (from the rules file)
- b. A description of how you triggered the alert
- c. The alert itself from the log file (after converting it to readable text)
