GHARDA FOUNDATION'S
GHARDA INSTITUTE OF TECHNOLOGY
A/P:-LAVEL, TALUKA: KHED, DIST. RATNAGIRI, STATE: MAHARASTRA

LABORATORY MANUAL

SUBJECT: - SYSTEM SECURITY LAB

Class: - T.E. (COMP)

Semester: - VI

Prepared by
Prof. V.M.Nair
Asst. Professor
Computer Engineering Dept

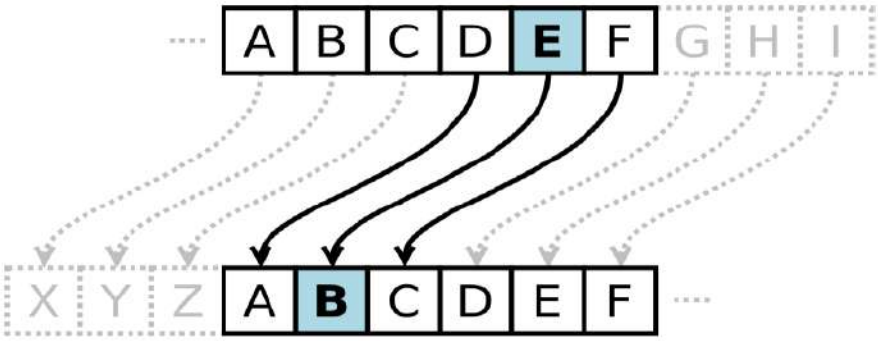
List of Experiments	
Sr.No	**Title**
1	Design and Implementation of Caesar Cipher.
2	Design and Implementation of Play fair Cipher.
3	Design and Implementation of Vigenere Cipher.
4	Design and Implementation of a product cipher using Substitution and Transposition.
5	Design and Implementation of a Data Encryption Standard (DES).
6	Study of Study the use of network reconnaissance tools like WHOIS, dig, traceroute, nslookup to gather information about networks and domain registrars.
7	Implementation of Diffie Hellman key exchange algorithm.
8	**a)** Implementation and analysis of RSA cryptosystem and **b)** Digital signature scheme using RSA/El Gamal.
9	For varying message sizes, test integrity of message using MD-5, SHA-1, and analyse the performance of the two protocols. Use crypt APIs.
10	Study of packet sniffer tools: wireshark: 1. Download and install wireshark and capture icmp, tcp, and http packets in promiscuous mode. 2. Explore how the packets can be traced based on different filters.
11	Download and install nmap. Use it with different options to scan open ports, perform OS fingerprinting, do a ping scan, tcp port scan, udp port scan, xmas scan etc.
12	Detect ARP spoofing using nmap and/or open source tool ARPWATCH and wireshark. Use arping tool to generate gratuitous arps and monitor using wireshark.
13	Simulate buffer overflow attack using Cppcheck .Splint etc.,
14	Explore the GPG tool of Linux to implement email security.

Experiment No. 1

Aim: - Design and Implementation of Caesar Cipher.

Theory :-

The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is named after Julius Caesar, who apparently used it to communicate with his generals.



Example

To pass an encrypted message from one person to another, it is first necessary that both parties have the 'key' for the cipher, so that the sender may encrypt it and the receiver may decrypt it. For the caesar cipher, the key is the number of characters to shift the cipher alphabet.

Here is a quick example of the encryption and decryption steps involved with the caesar cipher. The text we will encrypt is 'defend the east wall of the castle', with a shift (key) of 1.

```
plaintext:  defend the east wall of the castle
ciphertext: efgfoe uif fbtu xbmm pg uif dbtumf
```

Mathematical Description

First we translate all of our characters to numbers, 'a'=0, 'b'=1, 'c'=2, ... , 'z'=25. We can now represent the caesar cipher encryption function, e(x), where x is the character we are encrypting, as:

$$e(x) = (x + k) \pmod{26}$$

Where k is the key (the shift) applied to each letter. After applying this function the result is a number which must then be translated back into a letter. The decryption function is :

$$e(x) = (x - k) \pmod{26}$$

Conclusion:-

Thus we have illustrated the implementation of Caesar Cipher.

Experiment No. 2

Aim: - Design and Implementation of Play fair Cipher.

Theory:-

The Playfair Cipher was first described by Charles Wheatstone in 1854, and it was the first example of a Digraph Substitution Cipher. It is named after Lord Playfair, who heavily promoted the use of the cipher to the military.

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

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

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

EXAMPLE:

D. Playfair Cipher

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

Grouped text: CR YP TO IS TO XO EA SY

Ciphertext: AQ TV YB NI YB YF CB OZ

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

ALGORITHM:

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

STEP-2: Read the keyword from the user.

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

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

STEP-5: Display the obtained cipher text.

Conclusion:-

Thus we have illustrated the implementation of Play fair Cipher.

Experiment No. 3

Aim: - Design and Implementation of Vigenere Cipher.

Theory: -

To encrypt, a table of alphabets can be used, termed a tabula recta, Vigenère square, or Vigenère table. It consists of the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows. The alphabet used at each point depends on a repeating keyword.

Each row starts with a key letter. The remainder of the row holds the letters A to Z. Although there are 26 key rows shown, you will only use as many keys as there are unique letters in the key string, here just 5 keys, {L, E, M, O, N}. For successive letters of the message, we are going to take successive letters of the key string, and encipher each message letter using its corresponding key row. Choose the next letter of the key, go along that row to find the column heading that matches the message character; the letter at the intersection of [key-row, msg-col] is the enciphered letter.

EXAMPLE:

ALGORITHM:

STEP-1: Arrange the alphabets in row and column of a 26*26 matrix.
STEP-2: Circulate the alphabets in each row to position left such that the first letter is attached to last.
STEP-3: Repeat this process for all 26 rows and construct the final key matrix.
STEP-4: The keyword and the plain text is read from the user.
STEP-5: The characters in the keyword are repeated sequentially so as to match with that of the plain text.
STEP-6: Pick the first letter of the plain text and that of the keyword as the row indices and column indices respectively.
STEP-7: The junction character where these two meet forms the cipher character.
STEP-8: Repeat the above steps to generate the entire cipher text.

Conclusion:-

Thus we have illustrated the implementation of Vigenere Cipher.

Experiment No. 4

Aim: - Design and Implementation of a product cipher using Substitution and Transposition.

Theory:-

Substitution cipher is a method of encryption by which units of plaintext are replaced with cipher text according to a regular system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.

Transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the cipher text constitutes a permutation of the plaintext. That is, the order of the units is changed.

Substitution ciphers can be compared with Transposition ciphers. In a transposition cipher, the units of the plaintext are rearranged in a different and usually quite complex order, but the units themselves are left unchanged. By contrast, in a substitution cipher, the units of the plaintext are retained in the same sequence in the cipher text, but the units themselves are altered.

1. **Caesar Cipher:** In cryptography, a Caesar cipher, also known as a Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a shift of 3, A would be replaced by D, B would become E, and so on. The method is named after Julius Caesar, who used it to communicate with his generals.

   Example:
   The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. For instance, here is a Caesar cipher using a left rotation of three places (the shift parameter, here 3, is used as the key):
   Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Cipher: DEFGHIJKLMNOPQRSTUVWXYZABC

2. **Columnar Transposition:** In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword.

   In a regular columnar transposition cipher, any spare spaces are filled with nulls; in an irregular columnar transposition cipher, the spaces are left blank. Finally, the message is read off in columns, in the order specified by the keyword.

Algorithm/Procedure:

- **Substitution**

1. Display menu of operation - e for encryption and d for decryption.
2. Accept choice from user
3. If choice is encryption.
   a) Accept plaintext from user
   b) Accept key from user.
   c) Take k = 0.
   d) Extract kth character from string.
   e) Add key to it and get new value.
   f) If new value > 26
      New value = New value % 26.
   g) Add as kth character of ciphertext.
   h) Increment k.
   i) If(k < length(plaintext)) goto step 'd'.
   j) Display plaintext and ciphertext(output).
4. If choice is decryption
   k) Accept cipher text from user
   l) Accept key from user.
   m) Take k = 0.
   n) Extract k^{th} character from string.

o) Subtract key from it and get new value.

p) If new value > 26

New value = New value % 26.

q) Add as k^{th} character of plaintext.

r) Increment k.

s) If(k < length(cipher text)) goto step 'd'.

t) Display cipher text and plaintext(output).

5. Ask user want to continue or not
6. If yes, go to step 2;else stop.

- **Transposition**

1. Count how many letters are in your ciphertext (for example, 75) and factor that number (75=5*5*3).
2. Create all of the possible matrices to fit this ciphertext (in our case, 3x25, 5x15, 15x5, 25x3).
3. Write the ciphertext into these matrices down the columns.
4. For each of your matrices, consider all of the possible permutations of the columns (for n columns, there are n! possible rearrangements). In our case, we hope that the message was enciphered using one of the last two matrices (the 15x5 and the 25x3), since in those cases, we have only 6 and 120 possibilities to check (3! = 6, 5! = 120, 15! ~ 1.31x10^12, 25! ~ 1.55 x 10^25).
5. Rearrange each matrix to see if you get anything intelligible. Read the message off row-by row. Note that this is much more easily done by a computer than by hand, but it is doable (for small matrices).

Conclusion:

A product cipher is a composite of two or more elementary ciphers with the goal of producing a cipher which is more secure that any of the individual components. In product cipher substitution and transposition are applied to create confusion and diffusion in the text message.

Experiment No. 5

Aim: - Design and Implementation of a Data Encryption Standard (DES).

Theory:-

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted k1 to k16. Given that "only" 56 bits are actually used for encrypting, there can be 256 different keys.

The main parts of the algorithm are as follows:

_ Fractioning of the text into 64-bit blocks

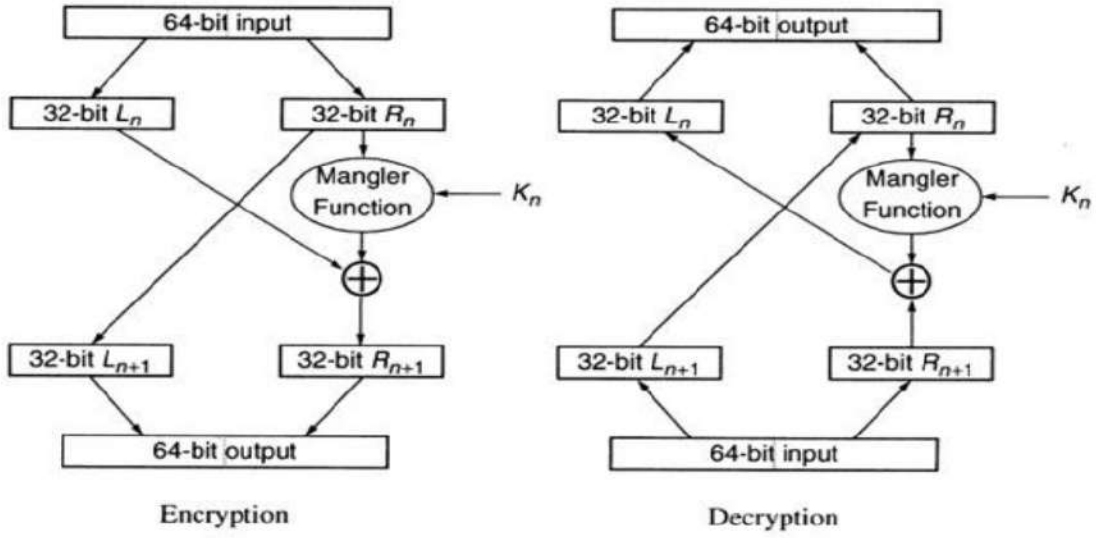
_ Initial permutation of blocks

_ Breakdown of the blocks into two parts: left and right, named L and R

_ Permutation and substitution steps repeated 16 times

_ Re-joining of the left and right parts then inverse initial permutation

EXAMPLE:



ALGORITHM:

STEP-1: Read the 64-bit plain text.

STEP-2: Split it into two 32-bit blocks and store it in two different arrays.

STEP-3: Perform XOR operation between these two arrays.

STEP-4: The output obtained is stored as the second 32-bit sequence and the original second 32-bit sequence forms the first part.

STEP-5: Thus the encrypted 64-bit cipher text is obtained in this way. Repeat the same process for the remaining plain text characters.

Conclusion:

Thus the data encryption standard algorithm had been implemented successfully.

Experiment No. 6

Aim:- Study of Study the use of network reconnaissance tools like WHOIS, dig, traceroute, nslookup to gather information about networks and domain registrars.

Theory:-

1. WHOIS : WHOIS is the Linux utility for searching an object in a WHOIS database. The WHOIS database of a domain is the publicly displayed information about a domains ownership, billing, technical, administrative, and nameserver information. Running a WHOIS on your domain will look the domain up at the registrar for the domain information. All domains have WHOIS information. WHOIS database can be queried to obtain the following information via WHOIS:

- Administrative contact details, including names, email addresses, and telephone numbers
- Mailing addresses for office locations relating to the target organization
- Details of authoritative name servers for each given domain

Example: Querying Facebook.com (in Figure 6.1)

2. Dig - Dig is a networking tool that can query DNS servers for information. It can be very helpful for diagnosing problems with domain pointing and is a good way to verify that your configuration is working. (in Figure 6.2)

3. Traceroute - traceroute prints the route that packets take to a network host. Traceroute utility uses the TTL field in the IP header to achieve its operation. For users who are new to TTL field, this field describes how much hops a particular packet will take while traveling on network. So, this effectively outlines the lifetime of the packet on network. This field is usually set to 32 or 64. Each time the packet is held on an intermediate router, it decreases the TTL value by 1. When a router finds the TTL value of 1 in a received packet then that packet is not forwarded but instead discarded. After discarding the packet, router sends an ICMP error message of —Time exceeded back to the source from where packet generated. The ICMP packet that is sent back contains the IP address of the router. So now it can be easily understood that traceroute operates by sending packets with TTL value starting from

1 and then incrementing by one each time. Each time a router receives the packet, it checks the TTL field, if TTL field is 1 then it discards the packet and sends the ICMP error packet containing its IP address and this is what traceroute requires. So traceroute incrementally fetches the IP of all the routers between the source and the destination. (in Figure 6.3)

4. Nslookup - The nslookup command is used to query internet name servers interactively for information. nslookup, which stands for "name server lookup", is a useful tool for finding out information about a named domain. By default, nslookup will translate a domain name to an IP address (or vice versa). For instance, to find out what the IP address of microsoft.com is, you could run the command: (in Figure 6.4)

Conclusion:

Various reconnaissance tools are studies and used to gather primary network information.

Experiment No. 7

Aim: - Implementation of Diffie Hellman key exchange algorithm.

Theory :-

Diffie Hellman key exchange is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

The Diffie–Hellman key exchange algorithm solves the following dilemma. Alice and Bob want to share a secret key for use in a symmetric cipher, but their only means of communication is insecure. Every piece of information that they exchange is observed by their adversary Eve. How is it possible for Alice and Bob to share a key without making it available to Eve? At first glance it appears that Alice and Bob face an impossible task. It was a brilliant insight of Diffie and Hellman that the difficulty of the discrete logarithm problem for F^*_p provides a possible solution.

The simplest, and original, implementation of the protocol uses the Multiplicative group of integers modulo p, where p is prime and g is primitive root mod p. Here is an example of the protocol:

1. Alice and Bob agree to use a prime number p=23 and base g=5.
2. Alice chooses a secret integer Xa=6, then sends Bob (g^XA) mod p.
   56 mod 23 = 8.
3. Bob chooses a secret integer Xb=15, then sends Alice (g^XB) mod p.
   515 mod 3 = 19.
4. Alice computes YA = (g^XA) mod p.
   196 mod 23 = 2.
5. Bob computes YB = (g^XB) mod p.
   815 mod 23 = 2.

In the original description, the Diffie Hellman exchange by itself does not provide authentication of the communicating parties and is thus vulnerable to a man-in-the-middle attack. A person in the middle may establish two distinct Diffie Hellman key exchanges, one

with Alice and the other with Bob, effectively masquerading as Alice to Bob, and vice versa, allowing the attacker to decrypt (and read or store) then re-encrypt the messages passed between them. A method to authenticate the communicating parties to each other is generally needed to prevent this type of attack.

Algorithm:

Alice and Bob, two users who wish to establish secure communications. We can assume that Alice and Bob know nothing about each other but are in contact.

STEP-1: Both Alice and Bob shares the same public keys g and p.

STEP-2: Alice selects a random public key a.

STEP-3: Alice computes his secret key A as g^a mod p.

STEP-4: Then Alice sends A to Bob.

STEP-5: Similarly Bob also selects a public key b and computes his secret key as B and sends the same back to Alice.

STEP-6: Now both of them compute their common secret key as the other one's secret key power of a mod p.

Conclusion:-

Thus the Diffie-Hellman key exchange algorithm had been successfully implemented.

Experiment No. 8

Aim: - a) Implementation and analysis of RSA cryptosystem and **b)** Digital signature scheme using RSA/El Gamal.

Theory:-

In cryptography, **RSA** (which stands for Rivest, Shamir and Adleman who first publicly described it) is an algorithm for public-key cryptography. It is the first algorithm known to be suitable for signing as well as encryption, and was one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys and the use of up-to-date implementations.

The RSA algorithm involves three steps: key generation, encryption and decryption.

Key generation

RSA involves a **public key** and a **private key.** The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.
   For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a primality test.
2. Compute $n = pq$.
   n is used as the modulus for both the public and private keys
3. Compute $\varphi(n) = (p - 1)(q - 1)$, where φ is Euler's totient function.
4. Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e,\varphi(n)) = 1$, i.e. e and $\varphi(n)$ are coprime.
   e is released as the public key exponent.
   e having a short bit-length and small Hamming weight results in more efficient encryption - most commonly 0x10001 = 65537. However, small values of e (such as 3) have been shown to be less secure in some settings.[4]
5. Determine $d = e^{-1} \bmod \varphi(n)$; i.e. d is the multiplicative inverse of e mod $\varphi(n)$.
   This is often computed using the extended Euclidean algorithm.

d is kept as the private key exponent.

The **public key** consists of the modulus n and the public (or encryption) exponent e. The **private key** consists of the private (or decryption) exponent d which must be kept secret.

Notes:

- An alternative, used by PKCS#1, is to choose d matching $de \equiv 1 \bmod \lambda$ with $\lambda = \text{lcm}(p-1, q-1)$, where lcm is the least common multiple. Using λ instead of $\varphi(n)$ allows more choices for d. λ can also be defined using the Carmichael function, $\lambda(n)$.
- The ANSI X9.31 standard prescribes, IEEE 1363 describes, and PKCS#1 allows, that p and q match additional requirements: be strong primes, and be different enough that Fermat factorization fails.

Encryption

Alice transmits her public key (n,e) to Bob and keeps the private key secret. Bob then wishes to send message **M** to Alice.

He first turns **M** into an integer $0 < m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c corresponding to

$$c = m^e \pmod{n}.$$

This can be done quickly using the method of exponentiation by squaring. Bob then transmits c to Alice.

Decryption

Alice can recover m from c by using her private key exponent d via computing

$$m = c^d \pmod{n}.$$

Given m, she can recover the original message **M** by reversing the padding scheme.

(In practice, there are more efficient methods of calculating c^d using the pre computed values below.)

Security:

The security of the RSA cryptosystem is based on two mathematical problems: the problem of factoring large numbers and the RSA problem. Full decryption of an RSA ciphertext is thought to be infeasible on the assumption that both of these problems are hard, i.e., no efficient algorithm exists for solving them. Providing security against *partial* decryption may require the addition of a secure padding scheme.

b) RSA Digital signature scheme

A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, that the sender cannot deny having sent the message (authentication and non-repudiation), and that the message was not altered in transit.

To sign: use a private signing algorithm

To verify: use a public verification algorithm

Alice wants to sign message m. She computes the signature of m (let's call it y) and sends the signed message (m,y) to Bob. Bob gets (m,y), runs the verification algorithm on it. The algorithm returns "true" iff y is Alice's signature of m.

The basic protocol:

1. Alice encrypts the document with her private key.
2. Alice sends the signed document to Bob.
3. Bob decrypts the document with Alice's public key.

RSA Signature Scheme

1. Alice chooses secret odd primes p,q and computes n=pq.
2. Alice chooses e_A with $\gcd(e_A,\Phi(n))=1$.
3. Alice computes $d_A = e_A^{-1} \bmod \Phi(n)$.
4. Alice's signature is $y = m^{d}_{A} \bmod n$.
5. The signed message is (m,y).
6. Bob can verify the signature by calculating $z = y^{e}_{a} \bmod n$. (The signature is valid iff m=z).

Conclusion:-

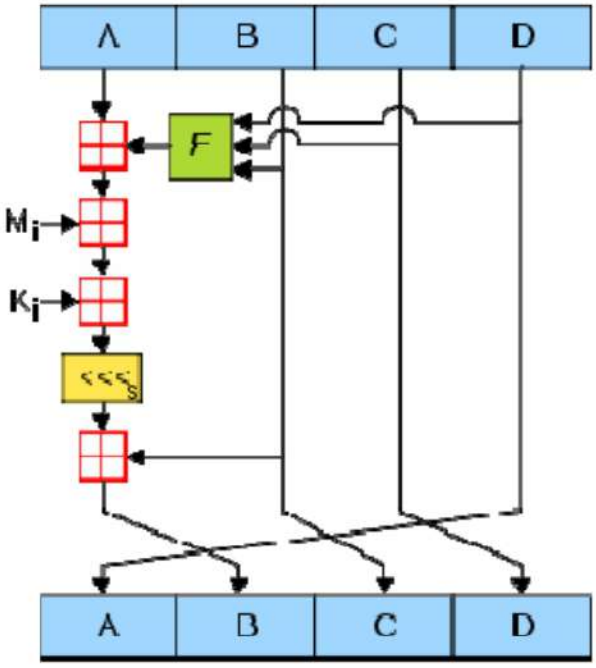
RSA is a strong encryption algorithm. RSA implements a public-key cryptosystem that allows secure communications and digital signatures, and its security rests in part on the difficulty of factoring large numbers.

Experiment No. 9

Aim: - For varying message sizes, test integrity of message using MD-5, SHA-1, and analyse the performance of the two protocols. Use crypt APIs.

Theory: -

MD5 (Message Digest algorithm 5) is a widely used cryptographic hash function with a 128 bit hash value. An MD5 hash is typically expressed as a 32 digit hexadecimal number. MD5 processes a variable length message into a fixed length output of 128 bits. The input message is broken up into chunks of 512 bit blocks (sixteen 32bit little endian integers) ; The message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64bit integer representing the length of the original message, in bits.



From above figure: One MD5 operation. MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. *F* is a nonlinear function; one function is used in each round.

M_i denotes a 32bit block of the message input, and K_i denotes a 32bit constant, different for each operation.

The main MD5 algorithm operates on a 128bit state, divided into four 32bit words, denoted *A*, *B*, *C* and *D*. These are initialized to certain fixed constants. The main algorithm then operates on each 512bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed *rounds*; each round is composed of 16 similar operations based on a nonlinear function *F*, modular addition, and left rotation.

ALGORITHM:

STEP-1: Read the 128-bit plain text.

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

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

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

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

Conclusion:

The main aim of message digest algorithm is to ensure integrity of message. The strength of MD5 and SHA 1 algorithm lies in the chaining function, because of which integrity of message cannot be compromised. Thus the implementation of MD5 hashing algorithm had been implemented successfully.

Experiment No. 10

Aim: - Study of packet sniffer tools: wireshark: 1. Download and install wireshark and capture icmp, tcp, and http packets in promiscuous mode. 2. Explore how the packets can be traced based on different filters.

Theory:-

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

Features of Wireshark:

- Available for UNIX and Windows.
- Capture live packet data from a network interface.
- Open files containing packet data captured with tcpdump/WinDump, Wireshark, and a
- number of other packet capture programs.
- Import packets from text files containing hex dumps of packet data.
- Display packets with very detailed protocol information.
- Export some or all packets in a number of capture file formats.
- Filter packets on many criteria.
- Search for packets on many criteria.
- Colorize packet display based on filters.

Capturing Packets

After downloading and installing wireshark, you can launch it and click the name of an interface under Interface List to start capturing packets on that interface. For example, if you want to capture traffic on the wireless network, click your wireless interface. You can configure advanced features by clicking Capture Options.

Installation of Wireshark:

sudo apt-get install wireshark (In Figure 10.1)

Running Wireshark

When you run the Wireshark program, the Wireshark graphical user interface shown in Figure 10.2 will de displayed.

The Wireshark interface has five major components: (In Figure 10.3)

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data.
- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is *not* a packet number contained in any protocol's header), the time at which the packet was captured.
- The **packet-header details window** provides details about the packet selected (highlighted) in the packet listing window. (To select a packet in the packet listing window, place the cursor over the packet's one-line summary in the packet listing window and click with the left mouse button.).
- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the Wireshark graphical user interface, is the **packet display filter field,** into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows)

After downloading and installing wireshark, you can launch it and click the name of an interface under Interface List to start capturing packets on that interface. For example, if you want to capture traffic on the wireless network, click your wireless interface. You can configure advanced features by clicking Capture Options.

As soon as you click the interface's name, you'll see the packets start to appear in real time. Wireshark captures each packet sent to or from your system. If you're capturing on a wireless interface and have promiscuous mode enabled in your capture options, you'll also see other the other packets on the network. (In Figure 10.4)

Click the stop capture button near the top left corner of the window when you want to stop capturing traffic.

Sample Exercises:

1. Capture the total number of HTTP GET requests. (In Figure 10.5)
2. Capture HTTP response (In Figure 10.6)
3. Capture DNS packets (In Figure 10.7)
4. Capture TCP Port 80 and UDP Port 80 (In Figure 10.8)

Conclusion:

Wireshark installation and network traffic analysis using packet sniffing is done. Detailed information about packets are explored by applying filters.

Experiment No. 11

Aim: - Download and install nmap. Use it with different options to scan open ports, perform OS fingerprinting, do a ping scan, tcp port scan, udp port scan, xmas scan etc.

Theory:-

Nmap (Network Mapper) is a security scanner originally written by Gordon Lyon (also known by his pseudonym Fyodor Vaskovich) used to discover hosts and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host and then analyzes the responses. Unlike many simple port scanners that just send packets at some predefined constant rate, Nmap accounts for the network conditions (latency fluctuations, network congestion, the target interference with the scan) during the run. Also, owing to the large and active user community providing feedback and contributing to its features, Nmap has been able to extend its discovery capabilities beyond simply figuring out whether a host is up or down and which ports are open and closed; it can determine the operating system of the target, names and versions of the listening services, estimated uptime, type of device, and presence of a firewall.

Nmap features include:

- Host Discovery – Identifying hosts on a network. For example, listing the hosts which respond to pings or have a particular port open.
- Port Scanning – Enumerating the open ports on one or more target hosts.
- Version Detection – Interrogating listening network services listening on remote devices to determine the application name and version number.
- OS Detection – Remotely determining the operating system and some hardware characteristics of network devices.

Basic commands working in Nmap:

- For target specifications: nmap <target's URL or IP with spaces between them>
- For OS detection: nmap -O <target-host's URL or IP>
- For version detection: nmap -sV <target-host's URL or IP>

SYN scan is the default and most popular scan option for good reasons. It can be performed quickly, scanning thousands of ports per second on a fast network not hampered by restrictive firewalls. It is also relatively unobtrusive and stealthy since it never completes TCP connections.

Installation of Nmap:

$ sudo apt-get install nmap (In Figure 11.1)

Sample Exercises:

- nmap -sP 192.168.12.93/22
  Ping scans the network, listing machines that respond to ping. (In Figure 11.2)
- FIN scan (-sF)
  Sets just the TCP FIN bit. (In Figure 11.3)
- **Scan using TCP SYN scan (default)**
  This command determines whether the port is listening. Using this command is a technique called half-open scanning. It is called half-open scanning because you don't establish a full TCP connection. Instead, you only send a SYN packet and wait for the response. If you receive a SYN/ACK response that means the port is listening: (In Figure 11.4)
- **Detect OS and services**
  This is the command to scan and search for the OS (and the OS version) on a host. This command will provide valuable information for the enumeration phase of your network security assessment (if you only want to detect the operating system, type nmap -O 192.168.0.9) (In Figure 11.5)
- **Standard service detection**
  This is the command to scan for running service. Nmap contains a database of about 2,200 well-known services and associated ports. Examples of these services are HTTP (port 80), SMTP (port 25), DNS (port 53), and SSH (port 22): (In Figure 11.6)
- -sO (IP protocol scan)

IP protocol scan allows you to determine which IP protocols (TCP, ICMP, IGMP, etc.) are supported by target machines. This isn't technically a port scan, since it cycles through IP protocol numbers rather than TCP or UDP port numbers. (In Figure 11.7)

- **Perform a thorough scan on a system**

  You can reveal all the information about a host system using the -A flag as shown below. This will reveal all the information pertaining to the host system such as the underlying OS, open ports, services running and their versions, etc.

  From the output, you can see that the command performs os and service detection, giving you detailed information such as the type of service and its version, and the port it is running on. The command usually takes a while to run but it is thorough and gives you all you need about the particular host system. (In Figure 11.8)

- **Scanning a particular port**

  To scan a specific port and check if it is open use the -p flag in the syntax below: (In Figure 11.9)

  ```
  $ nmap  -p port_number IP-address
  ```

  For example, to scan port 80 on a host system run:

  ```
  $ nmap  -p 80  192.168.43.103
  ```

  To scan a range of ports, for example between 80-433 use the syntax:

  ```
  $ nmap  -p 25-443  192.168.43.13 or $ nmap -p 80,443 192.168.43.13
  ```

- **Scan the most popular ports**

  Using "–top-ports" parameter along with a specific number lets you scan the top X most common ports for that host, as we can see: (In Figure 11.10)

  ```
  nmap --top-ports 10 192.168.12.93
  ```

- **Display host interfaces and routes**

  To display interfaces and routes on a particular host use the --iflist flag. The "–iflist" command will produce a list of the relevant interfaces and routes as shown. (In Figure 11.11)

Conclusion :

Nmap is studied and different types of nmap scans are used to gather host and network related information.

Experiment No. 12

Aim:- Detect ARP spoofing using nmap and/or open source tool ARPWATCH and wireshark. Use arping tool to generate gratuitous arps and monitor using wireshark.

Theory:-

Arpwatch is an open source computer software program that helps you to monitor Ethernet traffic activity (like Changing IP and MAC Addresses) on your network and maintains a database of ethernet/ip address pairings. It produces a log of noticed pairing of IP and MAC addresses information along with a timestamps, so you can carefully watch when the pairing activity appeared on the network. It also has the option to send reports via email to an network administrator when a pairing added or changed.

This tool is especially useful for Network administrators to keep a watch on ARP activity to detect ARP spoofing or unexpected IP/MAC addresses modifications.

Installation of ARPwatch:

$ sudo apt-get install arpwatch (In Figure 12.1)

To watch a specific interface, type the following command with -i and device name. (In Figure 12.2)

```
$ arpwatch -i eth0
```

So, whenever a new MAC is plugged or a particular IP is changing corresponding to MAC address on the network, you will notice syslog entries at **/var/log/syslog** or **/var/log/message** file. (In Figure 12.2)

```
$ tail -f /var/log/syslog
```

We can also check current **ARP** table, by using following command. (In Figure 12.3)

```
$ arp -a
```

Conclusion:

ARP spoofing is common attack launched in the network. It can be detected using ARPWATCH and some other network monitoring tools.

Experiment No. 13

Aim: - Simulate buffer overflow attack using Cppcheck .Splint etc.,

Theory:-

1. Cppcheck : Cppcheck is a tool for static C/C++ code analysis (CLI). Cppcheck is a commandline tool that tries to detect bugs that your C/C++ compiler doesn't see. It is versatile, and can check non-standard code including various compiler extensions, inline assembly code, etc. Its internal preprocessor can handle includes, macros, and several pre-processor commands. While Cppcheck is highly configurable, you can start using it just by giving it a path to the source code.

It includes checks for:

- pointers to out-of-scope auto variables;
- assignment of auto variables to an effective parameter of a function;
- out-of-bounds errors in arrays and STL;
- missing class constructors;
- variables not initialized by a constructor;
- use of memset, memcpy, etcetera on a class;
- non-virtual destructors for base classes;
- operator= not returning a constant reference to itself;
- use of deprecated functions (mktemp, gets, scanf);
- exceptions thrown in destructors;
- memory leaks in class or function variables;
- C-style pointer cast in C++ code;
- redundant if;
- misuse of the strtol or sprintf functions;
- unsigned division or division by zero;
- unused functions and struct members;
- passing parameters by value;
- misuse of signed char variables;
- unusual pointer arithmetic (such as "abc" + 'd');
- dereferenced null pointers;

- ❖ incomplete statements;
- ❖ misuse of iterators when iterating through a container;
- ❖ dereferencing of erased iterators;
- ❖ * use of invalidated vector iterators/pointers;

Step 1: Installation of cppcheck:

$ sudo apt-get install cppcheck (In Figure 13.1)

Step 2: Checking Vulnerability

sample2.c

```
char firstChar1 (/*@null@*/ char *s)
{
      return *s;
}
char firstChar2 (/*@null@*/ char *s)
{
      if (s == NULL) return '\0';
      return *s;
}
```

$ cppcheck sample2.c (In Figure 13.2)

2. Splint : Splint is a tool for statically checking C programs for security vulnerabilities and programming mistakes. Splint does many of the traditional lint checks including unused declarations, type inconsistencies, use before definition, unreachable code, ignored return values, execution paths with no return, likely infinite loops, and fall through cases. More powerful checks are made possible by additional information given in source code annotations. Annotations are stylized comments that document assumptions about functions, variables, parameters and types. In addition to the checks specifically enabled by annotations, many of the traditional lint checks are improved by exploiting this additional information.

Splint is designed to be flexible and allow programmers to select appropriate points on the effort benefit curve for particular projects. As different checks are turned on and more information is given in code annotations the number of bugs that can be detected increases dramatically. (In Figure 13.3)

Problems detected by Splint include:

- Dereferencing a possibly null pointer

- Using possibly undefined storage or returning storage that is not properly defined
- Type mismatches, with greater precision and flexibility than provided by C compilers
- Violations of information hiding
- Memory management errors including uses of dangling references and memory leaks
- Dangerous aliasing
- Modifications and global variable uses that are inconsistent with specified interfaces
- Problematic control flow such as likely infinite loops, fall through cases or incomplete switches and suspicious statements
- Buffer overflow vulnerabilities
- Dangerous macro implementations or invocations
- Violations of customized naming conventions

Example 1:

$ splint sample2.c (In Figure 13.4)

Example 2:

Sample3.c

```
#include <stdio.h>
#include <string.h>
int main(void)
{
char buff[15];
int pass = 0;
printf("\n Enter the password : \n");
gets(buff);
if(strcmp(buff, "thegeekstuff"))
{
printf ("\n Wrong Password \n");
}
else
{
printf ("\n Correct Password \n");
pass = 1;
}
if(pass)
{
/* Now Give root or admin rights to user*/
printf ("\n Root privileges given to the user
\n");
}
return 0;
}
```

$ splint sample3.c (In Figure 13.5)

Example 3:

Sample4.c

```
#include <stdio.h>
#include <string.h>
char password[] = "password";
int get_password() {
int auth_ok = 0;
char buff[16];
printf("Enter password: ");
scanf("%s", buff);
if(strncmp(buff, password, sizeof(password)) == 0)
auth_ok = 1;
return auth_ok;
}
void success() {
printf("Success!\n");
}
int main(int argc, char** argv) {
int res = get_password();
if (res == 0) {
printf("Failure\n");
return 0;
}
success();
return 0;
}
```

$ splint sample4.c (In Figure 13.6)

Conclusion:

Software vulnerabilities causing buffer overflow are studied and detected using Splint and cppcheck.

Experiment No. 14

Aim: - Explore the GPG tool of Linux to implement email security.

Theory:-

Pretty Good Privacy (PGP) is a data encryption and decryption computer program that provides cryptographic privacy and authentication for data communication. PGP is often used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications.

PGP encryption uses a serial combination of hashing, data compression, symmetric-key cryptography, and finally public-key cryptography; each step uses one of several supported algorithms. Each public key is bound to a user name and/or an e-mail address. The first version of this system was generally known as a web of trust to contrast with the X.509 system, which uses a hierarchical approach based on certificate authority and which was added to PGP implementations later. Current versions of PGP encryption include both options through an automated key management server.

GNU Privacy Guard (GnuPG or GPG) is a free software replacement for Symantec's PGP cryptographic software suite. GnuPG is a hybrid-encryption software program because it uses a combination of conventional symmetric-key cryptography for speed, and public-key cryptography for ease of secure key exchange, typically by using the recipient's public key to encrypt a session key which is only used once. This mode of operation is part of the OpenPGP standard and has been part of PGP from its first version.

Procedure:

Step1: Installation of gpg

$ sudo apt-get install gnupg (In Figure 14.1)

Step 2: Generation of Key

$ gpg --gen-key (In Figure 14.2.1 & 14.2.2)

Step 3: Listing Keys

$ gpg --list-keys (In Figure 14.3)

(Followed, Figure 14.4.1, 14.4.2, Figure 14.5.1 & 14.5.2, Figure 14.6 & Figure 14.7)

Signing process:

(In Figure 14.8, 14.9)

Encryption and Decryption

(In Figure 14.10)

Encryption

(In Figure 14.11, 14.12, 14.13)

Decryption:

(In Figure 14.14.1, 14.14.2, 14.15 & 14.16)

Conclusion:

GPG is used for authentication and privacy to messages over the internet. GPG was originated to address the security concerns of plain e-mail or text messages. GnuPG is used to demonstrate usage of GPG.