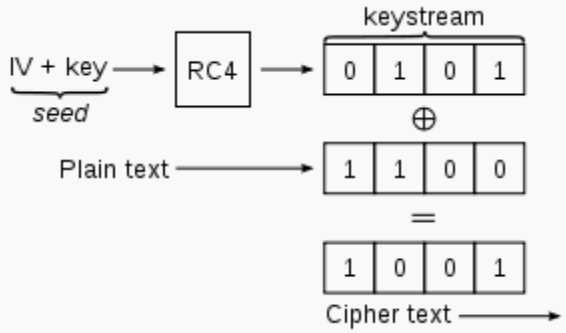
EXPERIMENT 1

Aim: Write a program to make study of different security protocols like WEP/ WPA2 PSK, 802.1x EAP security protocol. And Implement RC4 Algorithm.

Theory:

Wired Equivalent Privacy (**WEP**) is a security algorithm for IEEE 802.11 wireless networks. Introduced as part of the original 802.11 standard ratified in 1997, its intention was to provide data confidentiality comparable to that of a traditional wired network. WEP, recognizable by the key of 10 or 26 hexadecimal digits, was at one time widely in use and was often the first security choice presented to users by router configuration tools.



Basic WEP encryption: RC4 keystream XORed with plaintext

A 64-bit WEP key is usually entered as a string of 10 hexadecimal (base 16) characters (0–9 and A–F). Each character represents 4 bits, 10 digits of 4 bits each gives 40 bits; adding the 24-bit IV produces the complete 64-bit WEP key (4 bits × 10 + 24 bits IV = 64 bits of WEP key). Most devices also allow the user to enter the key as 5 ASCII characters (0–9, a–z, A–Z), each of which is turned into 8 bits using the character's byte value in ASCII (8 bits × 5 + 24 bits IV = 64 bits of WEP key); however, this restricts each byte to be a printable ASCII character, which is only a small fraction of possible byte values, greatly reducing the space of possible keys.

A 128-bit WEP key is usually entered as a string of 26 hexadecimal characters. 26 digits of 4 bits each gives 104 bits; adding the 24-bit IV produces the complete 128-bit WEP key (4 bits × 26 + 24 bits IV = 128 bits of WEP key). Most devices also allow the user to enter it as 13 ASCII characters (8 bits × 13 + 24 bits IV = 128 bits of WEP key).

A 152-bit and 256-bit WEP systems are available from some vendors. As with the other WEP variants, 24 bits of that is for the IV, leaving 128 or 232 bits for actual protection. These 128 or 232 bits are typically entered as 32 or 58 hexadecimal characters (4 bits × 32 + 24 bits IV = 152 bits of WEP key, 4 bits × 58 + 24 bits IV = 256 bits of WEP key). Most devices also allow the user

to enter it as 16 or 29 ASCII characters (8 bits × 16 + 24 bits IV = 152 bits of WEP key, 8 bits × 29 + 24 bits IV = 256 bits of WEP key).

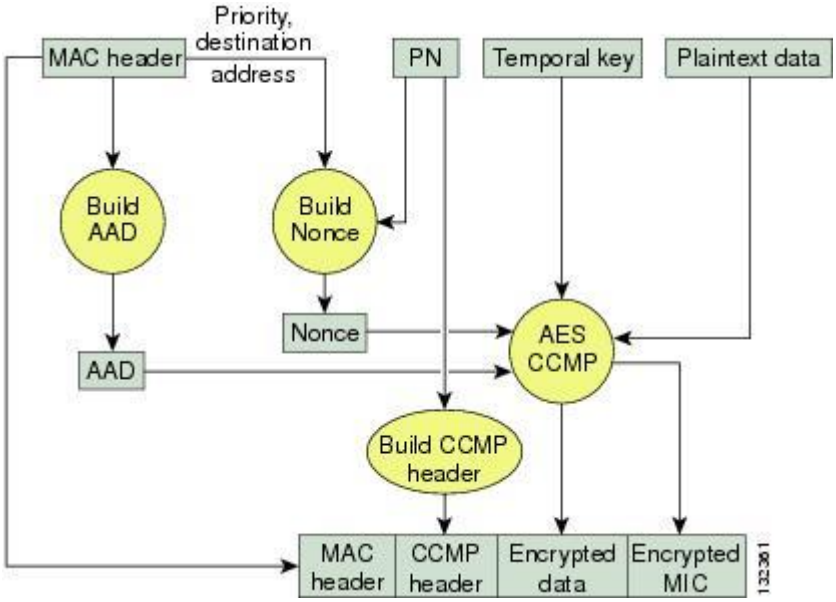
WPA2

WPA2 is a security scheme that specifies two main aspects of your wireless security:

- Authentication: Your choice of PSK ("Personal") or 802.1X ("Enterprise").
- Encryption: Always AES-CCMP.

If you're using WPA2 security on your network, you have two authentication choices: You either have to use a single password for the whole network that everyone knows (this is called a Pre-Shared Key or PSK), or you use 802.1X to force each user to use his own unique login credentials (e.g. username and password).

Regardless of which authentication type you've set up your network to use, WPA2 always uses a scheme called AES-CCMP to encrypt your data over the air for the sake of confidentiality, and to thwart various other kinds of attacks.



WPA is an 802.11i-based security solution from the Wi-Fi Alliance that addresses the vulnerabilities of WEP. WPA uses Temporal Key Integrity Protocol (TKIP) for encryption and dynamic encryption key generation by using either a pre-shared key, or RADIUS/802.1x-based authentication. The mechanisms introduced into WPA were designed to address the weakness of the WEP solution without requiring hardware upgrades. WPA2 is the next generation of Wi-Fi security and is also based on the 802.11i standard. It is the approved Wi-Fi Alliance interoperable implementation of the ratified IEEE 802.11i standard. WPA 2 offers two classes of certification: Enterprise and Personal. Enterprise requires support for RADIUS/802.1x-based authentication and pre-shared key (Personal) requires only a common key shared by the client and the AP.

The key exchange will be done by using PMK (Pairwise Master Key) and EAP-TLS

Key Management for Link Layer Security Key Management for Link Layer Security

If you're using WPA2 security on your network, you have two authentication choices: You either have to use a single password for the whole network that everyone knows (this is called a Pre-Shared Key or PSK), or you use 802.1X to force each user to use his own unique login credentials (e.g. username and password).

Regardless of which authentication type you've set up your network to use, WPA2 always uses a scheme called AES-CCMP to encrypt your data over the air for the sake of confidentiality, and to thwart various other kinds of attacks.

802.1x EAP-

802.1X is "EAP over LANs" or EAPoL. EAP stands for "Extensible Authentication Protocol", which means its kind of a plug-in scheme for various authentication methods.

If your wireless router's user interface has "802.1X" on a list of *encryption* types, then it probably means "802.1X with dynamic WEP", which is an old scheme where 802.1X is used for authentication, and per-user per-session WEP keys are dynamically generated as part of the authentication process, and thus WEP is ultimately the encryption method used.

RC4 Algorithm-

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define N 256   // 2^8

void swap(unsigned char *a, unsigned char *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int KSA(char *key, unsigned char *S) {

    int len = strlen(key);
    int j = 0;
```

```
    for(int i = 0; i < N; i++)
        S[i] = i;

    for(int i = 0; i < N; i++) {
        j = (j + S[i] + key[i % len]) % N;

        swap(&S[i], &S[j]);
    }

    return 0;
}

int PRGA(unsigned char *S, char *plaintext, unsigned char *ciphertext) {

    int i = 0;
    int j = 0;

    for(size_t n = 0, len = strlen(plaintext); n < len; n++) {
        i = (i + 1) % N;
        j = (j + S[i]) % N;

        swap(&S[i], &S[j]);
        int rnd = S[(S[i] + S[j]) % N];

        ciphertext[n] = rnd ^ plaintext[n];

    }

    return 0;
}

int RC4(char *key, char *plaintext, unsigned char *ciphertext) {

    unsigned char S[N];
    KSA(key, S);

    PRGA(S, plaintext, ciphertext);

    return 0;
}

int main(int argc, char *argv[]) {
```

```
    if(argc < 3) {
        printf("Usage: %s <key> <plaintext>", argv[0]);
        return -1;
    }

    unsigned char *ciphertext = malloc(sizeof(int) * strlen(argv[2]));

    RC4(argv[1], argv[2], ciphertext);
    printf("Encrypted: ");

    for(size_t i = 0, len = strlen(argv[2]); i < len; i++)
        printf("%02hhX", ciphertext[i]);

    return 0;
}
```

Output:

```
C:\Users\karan\Desktop>gcc rc4.c -o rc4

C:\Users\karan\Desktop>.\rc4 G "Darren Shan"
Encrypted: 1515552503200F0937EAA7
```

13/09/2020 **Mohd. Aftab Alam**
02013302717

EXPERIMENT 2

Aim: Write a Program to implement Advanced Encryption Standard (AES).

Theory:

AES is based on a design principle known as a substitution-permutation network, combination of both substitution and permutation, and is fast in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification *per se* is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4 × 4 column-major order matrix of bytes, termed the *state*, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field.

For instance, if there are 16 bytes, , these bytes are represented as this matrix:

The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of cycles of repetition are as follows:

- 10 cycles of repetition for 128-bit keys.
- 12 cycles of repetition for 192-bit keys.
- 14 cycles of repetition for 256-bit keys.

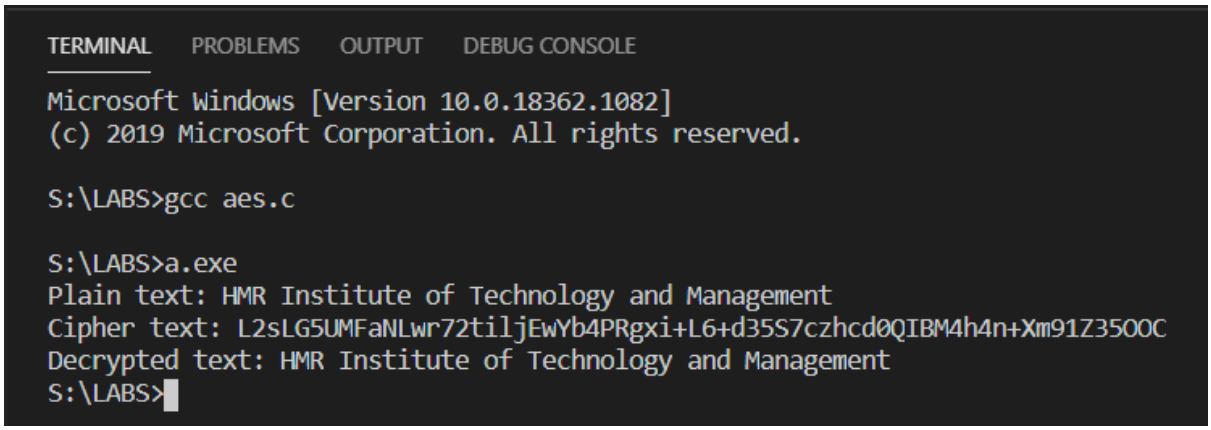
Each round consists of several processing steps, each containing four similar but different stages, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mcrypt.h>
#include <math.h>
#include <stdint.h>

int encrypt(void* buffer, int buffer_len, char* IV, char* key,intkey_len ) {
```

```
  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
if( buffer_len % blocksize != 0 ){return 1;}

mcrypt_generic_init(td, key, key_len, IV);
mcrypt_generic(td, buffer, buffer_len);
mcrypt_generic_deinit (td);
mcrypt_module_close(td);

  return 0;
}

int decrypt(void* buffer, int buffer_len, char* IV, char* key, int key_len ) {
  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
if( buffer_len % blocksize != 0 ){return 1;}

mcrypt_generic_init(td, key, key_len, IV);
mdecrypt_generic(td, buffer, buffer_len);
mcrypt_generic_deinit (td);
mcrypt_module_close(td);

  return 0;
}

void display(char* ciphertext, int len) {
  int v;
for(v=0; v<len; v++) {
printf("%d ", ciphertext[v]);
  }
printf("\n");
}

int main()
{
   MCRYPT td, td2;
char * plaintext = "HMR Institute of Technology and Management";
   char* IV = "AAAAA";
   char* key = "0123abc456def789";
   int keysize = 16;
   char* buffer;
   int buffer_len = 16;

   buffer =  calloc(1, buffer_len);
strncpy(buffer, plaintext, buffer_len);
printf("==C==\n");
printf("Plain: %s\n", plaintext);
encrypt(buffer, buffer_len, IV, key, keysize);
printf("Cipher: "); display(buffer, buffer_len);
decrypt(buffer, buffer_len, IV, key, keysize);
printf("Decrypt: %s\n", buffer);

   return 0;
}
```

Output:


TERMINAL
PROBLEMS
OUTPUT
DEBUG CONSOLE
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.
S:\LABS>gcc aes.c
S:\LABS>a.exe
Plain text: HMR Institute of Technology and Management
Cipher text: L2sLG5UMFaNLwr72tiljEwYb4PRgxi+L6+d35S7czhcd0QIBM4h4n+Xm91Z35OOC
Decrypted text: HMR Institute of Technology and Management
S:\LABS>

13/09/2020 **Mohd. Aftab Alam**
02013302717

EXPERIMENT 3

Aim: Write a program for Caesar Cipher algorithm to encrypt and decrypt the data.

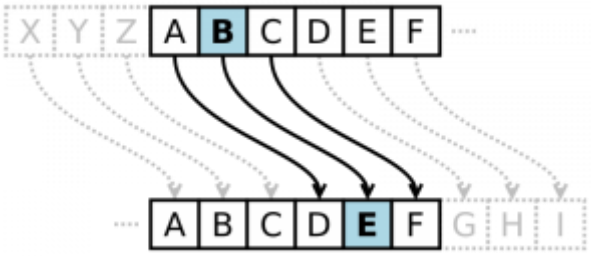
Theory:

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.

The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as-

$$E_n(x) = (x+n) \bmod 26$$
Encryption phase with shift n

$$D_n(x) = (x-n) \bmod 26$$
Decryption phase with shift n



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

When encrypting, a person looks up each letter of the message in the "plain" line and writes down the corresponding letter in the "cipher" line. Deciphering is done in reverse.

Ciphertext: WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
Plaintext: the quick brown fox jumps over the lazy dog

13/09/2020

Mohd. Aftab Alam
02013302717

Encryption:

```
#include<stdio.h>
int main()
{
char message[100], ch;
int i, key;
   printf("Enter a message to encrypt: ");
   gets(message);
   printf("Enter key: ");
   scanf("%d", &key);
   for(i = 0; message[i] != '\0'; ++i){
      ch = message[i];
      if(ch>= 'a' &&ch<= 'z'){
         ch = ch + key;
         if(ch> 'z'){
            ch = ch - 'z' + 'a' - 1;}
         message[i] = ch;}
      else if(ch>= 'A' &&ch<= 'Z'){
         ch = ch + key;
         if(ch> 'Z'){
            ch = ch - 'Z' + 'A' - 1;
         }
         message[i] = ch;
      }
   }
   printf("Encrypted message: %s", message);
   return 0;
}
```

Output:

```
C:\Users\karan\Desktop>gcc caesar.c -o caesar

C:\Users\karan\Desktop>.\caesar
Enter a message to encrypt: Darren
Enter key: 7
Encrypted message: Khyylu
```

Decryption:

```
#include<stdio.h>
int main()
{
   char message[100],ch;
   int i,key;
   printf("Enter a message to decrypt: ");
   gets(message);
   printf("Enter key: ");
   scanf("%d",&key);
   for(i=0;message[i]!='\0';++i){
      ch=message[i];
      if(ch>='a'&&ch<='z'){
         ch=ch-key;
         if(ch<'a'){
            ch=ch+'z'-'a'+1;
         }
         message[i]=ch;
      }
      else if(ch>='A'&&ch<='Z'){
         ch=ch-key;
         if(ch<'A'){
            ch=ch+'Z'-'A'+1;
         }
message[i]=ch;
      }
   }
   printf("Decrypted message: %s",message);
   return 0;
}
```

Output:

```
C:\Users\karan\Desktop>gcc decryptcaesar.c -o decryptcaesar

C:\Users\karan\Desktop>.\decryptcaesar
Enter a message to decrypt: Khyylu
Enter key: 7
Decrypted message: Darren
```