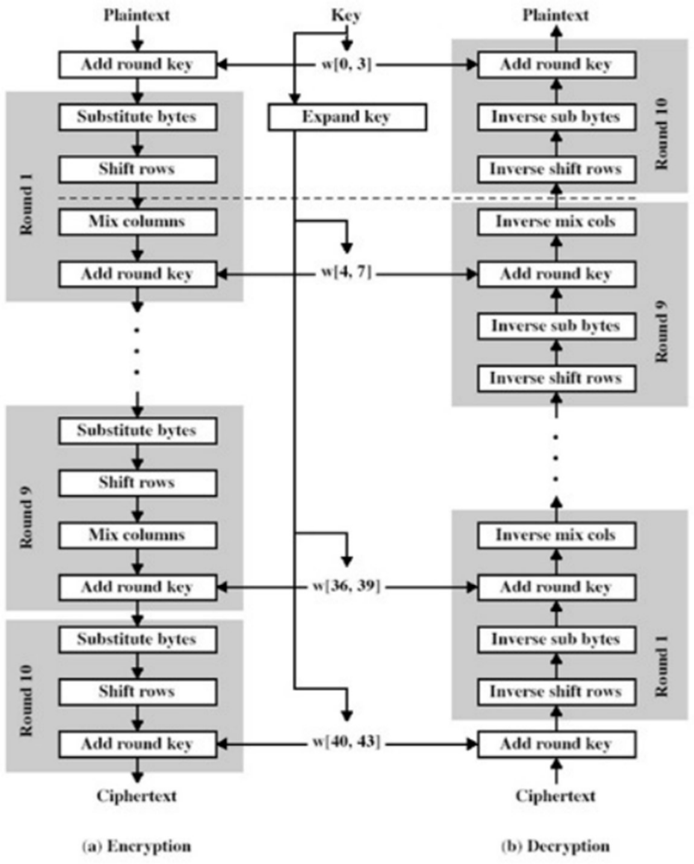
Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 7

Title: Implementation of AES Algorithm.

Theory:
Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.



Points to remember

- AES is a block cipher.
- The key size can be 128/192/256 bits.
- Encrypts data in blocks of 128 bits each.

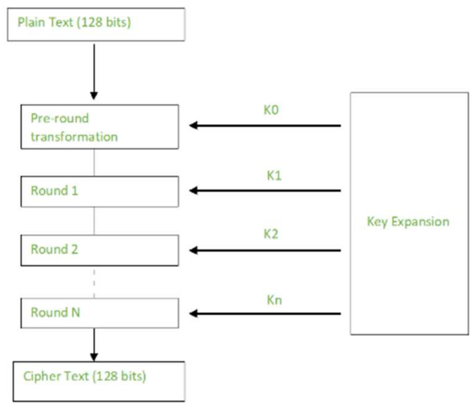
That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text as output. AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

Working of the cipher :
AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits (or 16 bytes) of the input data at a time.

The number of rounds depends on the key length as follows :

- 128 bit key – 10 rounds
- 192 bit key – 12 rounds
- 256 bit key – 14 rounds

Creation of Round keys :
A Key Schedule algorithm is used to calculate all the round keys from the key. So the initial key is used to create many different round keys which will be used in the corresponding round of the encryption.



Encryption :
AES considers each block as a 16 byte (4 byte x 4 byte = 128) grid in a column major arrangement.

```
[ b0 | b4 | b8 | b12 |
| b1 | b5 | b9 | b13 |
| b2 | b6 | b10| b14 |
| b3 | b7 | b11| b15 ]
```

Each round comprises of 4 steps :

- SubBytes
- ShiftRows
- MixColumns
- Add Round Key

The last round doesn't have the MixColumns round.

The SubBytes does the substitution and ShiftRows and MixColumns performs the permutation in the algorithm.

SubBytes :
This step implements the substitution.

SubBytes :
This step implements the substitution.

In this step each byte is substituted by another byte. Its performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte. The result of this step is a 16 byte (4 x 4) matrix like before.

The next two steps implement the permutation.

ShiftRows :
This step is just as it sounds. Each row is shifted a particular number of times.

- The first row is not shifted
- The second row is shifted once to the left.
- The third row is shifted twice to the left.
- The fourth row is shifted thrice to the left.

(A left circular shift is performed.)

```
[ b0  | b1  | b2  | b3  ]         [ b0  | b1  | b2  | b3  ]
| b4  | b5  | b6  | b7  |    ->   | b5  | b6  | b7  | b4  |
| b8  | b9  | b10 | b11 |         | b10 | b11 | b8  | b9  |
[ b12 | b13 | b14 | b15 ]         [ b15 | b12 | b13 | b14 ]
```

MixColumns :
This step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

This step is skipped in the last round.

```
[ c0 ]        [ 2  3  1  1 ]  [ b0 ]
| c1 |  =     | 1  2  3  1 |     | b1 |
| c2 |      | 1  1  2  3 |     | b2 |
[ c3 ]      [ 3  1  1  2 ]     [ b3 ]
```

Add Round Keys :
Now the resultant output of the previous stage is XOR-ed with the corresponding round key. Here, the 16 bytes is not considered as a grid but just as 128 bits of data.

After all these rounds 128 bits of encrypted data is given back as output. This process is repeated until all the data to be encrypted undergoes this process.

Decryption :
The stages in the rounds can be easily undone as these stages have an opposite to it which when performed reverts the changes.Each 128 blocks goes through the 10,12 or 14 rounds depending on the key size.

The stages of each round in decryption is as follows :

- Add round key
- Inverse MixColumns
- ShiftRows
- Inverse SubByte

The decryption process is the encryption process done in reverse so i will explain the steps with notable differences.

Inverse MixColumns :
This step is similar to the MixColumns step in encryption, but differs in the matrix used to carry out the operation.

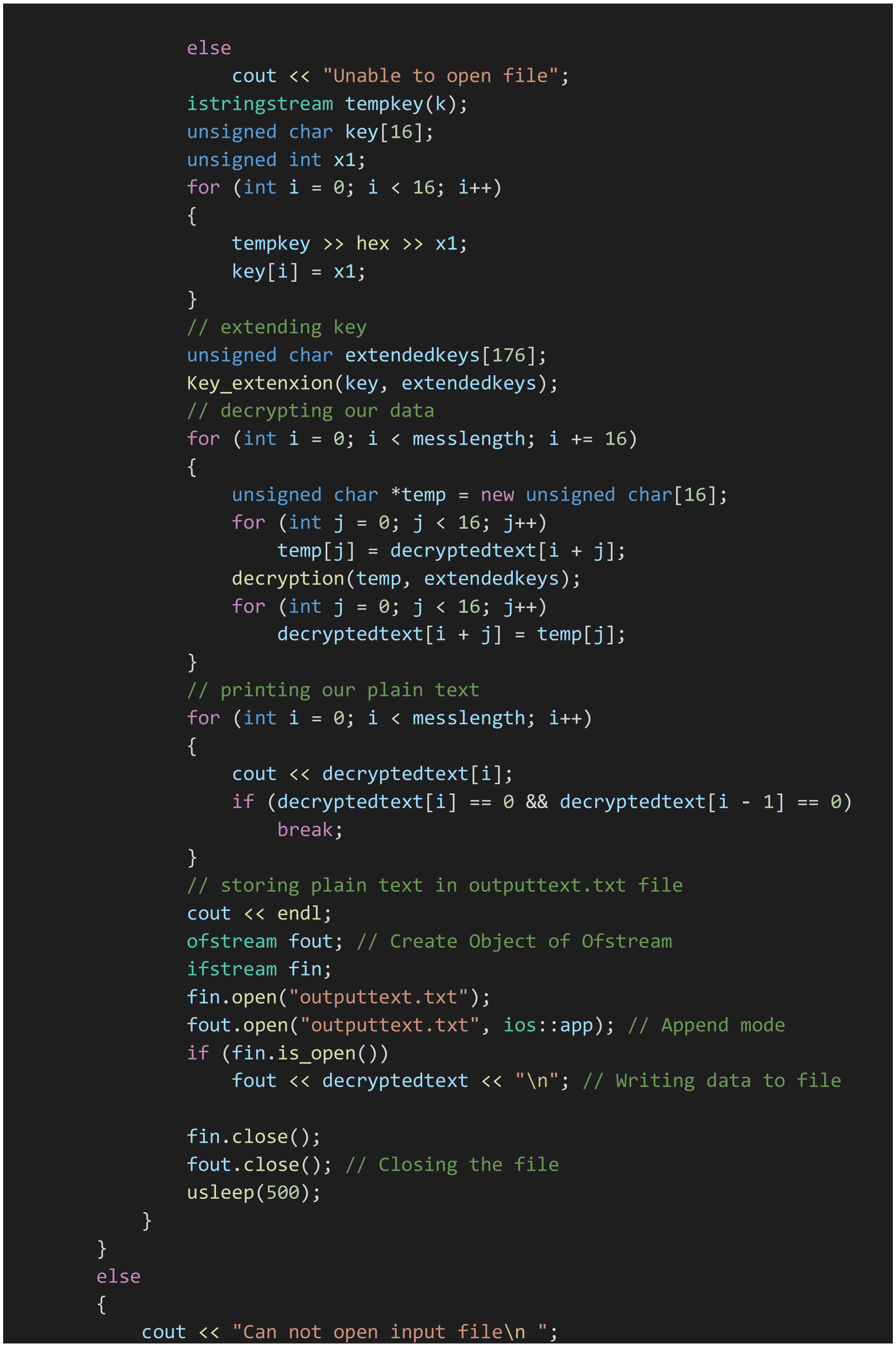
```
[ b0 ]        [ 14  11  13  9  ]  [ c0 ]
| b1 |  =     | 9   14  11  13 |     | c1 |
| b2 |      | 13  9   14  11 |     | c2 |
[ b3 ]        [ 11  13  9   14 ]     [ c3 ]
```

Inverse SubBytes :
Inverse S-box is used as a lookup table and using which the bytes are substituted during decryption.

Code Snapshots:

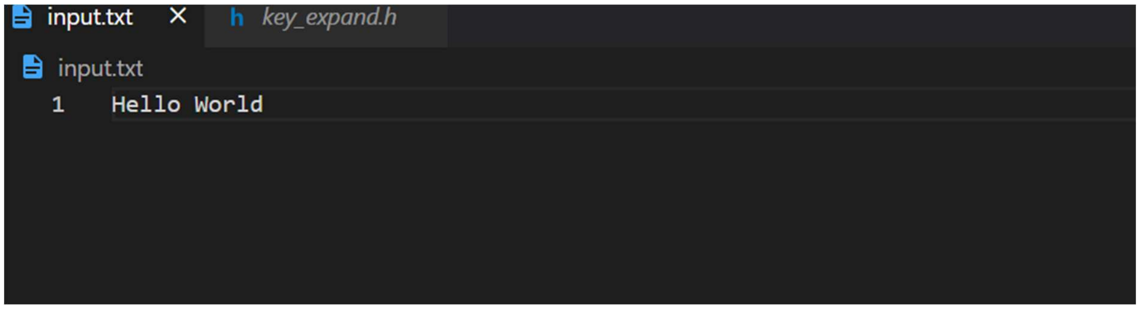
```
#include <iostream>
#include <fstream>
#include <cstring>
#include <sstream>
#include "key_expand.h"
#include "encoding.h"
#include "decoding.h"
#include <typeinfo>
#include <unistd.h>
using namespace std;
int main()
{
    // we will read from file input.txt
    int extendedlength = 0;
    int choice;
    string myText;
label:
    cout << "Welcome to 128 bits AES encryption" << endl;
    cout << endl;
    cout << "Enter you choice " << endl;
    cout << "1- Encoding" << endl;
    cout << "2- Decoding" << endl;
    cin >> choice;

    switch (choice)
    {
    case 1:
    {
        // encryption of text data
        ifstream File;
        string filepath = "encryption.aes";
        // clearing encryption.aes before editing
        File.open(filepath.c_str(), std::ifstream::out |
std::ifstream::trunc);
        if (!File.is_open() || File.fail())
        {
            File.close();
            printf("\nError : failed to erase file content !");
        }
        File.close();
        // reading plain text from input.txt
        fstream newfile;
        newfile.open("input.txt", ios::in); // open a file to perform read
operation using file object
        if (newfile.is_open())
        { // checking whether the file is open
            cout << "Reading plain text from input.txt .........\n";
```

```cpp
            usleep(1000);
            string tp;
            cout << "Reading KEY from key.txt ......\n";
            usleep(1000);
            cout << "Now encrypting ....\n";
            usleep(1000);
            cout << "writing encrypted data in encryption.aes ..\n";
            usleep(1000);
            cout << endl;
            while (getline(newfile, tp))
            {
                // read data from file object and put it into string.
                int messlength = tp.length();
                int extendedlength;
                if ((messlength % 16) != 0)
                {
                    extendedlength = messlength + (16 - (messlength % 16));
                }
                else
                {
                    extendedlength = messlength;
                }
                unsigned char *encryptedtext = new unsigned
char[extendedlength];
                for (int i = 0; i < extendedlength; i++)
                {
                    if (i < messlength)
                        encryptedtext[i] = tp[i];
                    else
                        encryptedtext[i] = 0;
                }
                // getting key from key.txt
                string k;
                ifstream infile;
                infile.open("key.txt");
                if (infile.is_open())
                {
                    getline(infile, k); // The first line of file should be
the key

                    infile.close();
                }

                else
                    cout << "Unable to open file";

                istringstream tempkey(k);
                unsigned char key[16];
                unsigned int x;
                for (int i = 0; i < 16; i++)
                {
                    tempkey >> hex >> x;
                    key[i] = x;
                }
                // extending key
                unsigned char extendedkeys[176];
                Key_extenxion(key, extendedkeys);

                // encrypting our plain text
                for (int i = 0; i < extendedlength; i += 16)
                {
                    unsigned char *temp = new unsigned char[16];
                    for (int j = 0; j < 16; j++)
                    {
                        temp[j] = encryptedtext[i + j];
                    }
                    encryption(temp, extendedkeys);
                    for (int j = 0; j < 16; j++)
                    {
                        encryptedtext[i + j] = temp[j];
                    }
                }
                // storing our encrypted data in encryption.aes
                ofstream fout; // Create Object of Ofstream
                ifstream fin;
                fin.open("encryption.aes");
                fout.open("encryption.aes", ios::app); // Append mode
                if (fin.is_open())
                    fout << encryptedtext << "\n"; // Writing data to file
                fin.close();
                fout.close();
            }
            cout << "128-bit AES encryption is done sucessfully\n";
            cout << "Data has been appended to file encryption.aes";
            newfile.close(); // close the file object.
        }
        break;
    }

    case 2:
    {
        cout << "Reading encrypted data from encryption.txt .........\n";
        usleep(1000);
        string tp;
        cout << "Reading KEY from key.txt ......\n";
        usleep(1000);
        cout << "Now Decrypting ....\n";
        usleep(1000);
        cout << "writing decrypted data in outputtext.txt ..\n";
        usleep(1000);
        cout << endl;
        cout << "Following is our decrypted text:- \n";
        // clearing outputtext file
        ifstream File;
        string filepath = "outputtext.txt";
        File.open(filepath.c_str(), std::ifstream::out |
std::ifstream::trunc);
        if (!File.is_open() || File.fail())
        {
            File.close();
            printf("\nError : failed to erase file content !");
        }
        File.close();

        ifstream MyReadFile;
        MyReadFile.open("encryption.aes", ios::in | ios::binary);
        if (MyReadFile.is_open())
        {
            while (getline(MyReadFile, myText))
            {
                cout.flush();
                char *x;
                x = &myText[0];
                int messlength = strlen(x);
                char *msg = new char[myText.size() + 1];

                strcpy(msg, myText.c_str());

                int n = strlen((const char *)msg);
                unsigned char *decryptedtext = new unsigned char[n];
                // decrypting our encrypted data
                for (int i = 0; i < n; i++)
                {
                    decryptedtext[i] = (unsigned char)msg[i];
                }
                // reading key from key.txt file
                string k;
                ifstream infile;
                infile.open("key.txt");
                if (infile.is_open())
                {
                    getline(infile, k); // The first line of file should be
the key

                    infile.close();
                }
```

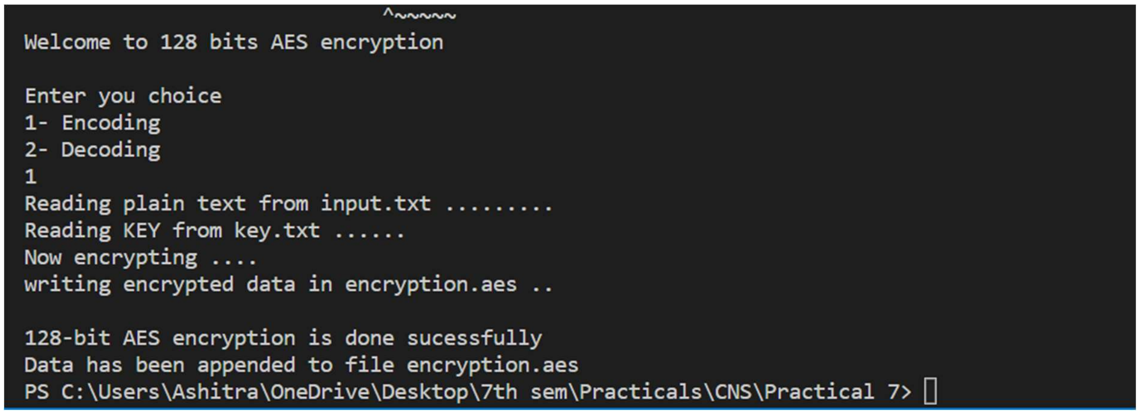


```
            else
                cout << "Unable to open file";
            istringstream tempkey(k);
            unsigned char key[16];
            unsigned int x1;
            for (int i = 0; i < 16; i++)
            {
                tempkey >> hex >> x1;
                key[i] = x1;
            }
            // extending key
            unsigned char extendedkeys[176];
            Key_extenxion(key, extendedkeys);
            // decrypting our data
            for (int i = 0; i < messlength; i += 16)
            {
                unsigned char *temp = new unsigned char[16];
                for (int j = 0; j < 16; j++)
                    temp[j] = decryptedtext[i + j];
                decryption(temp, extendedkeys);
                for (int j = 0; j < 16; j++)
                    decryptedtext[i + j] = temp[j];
            }
            // printing our plain text
            for (int i = 0; i < messlength; i++)
            {
                cout << decryptedtext[i];
                if (decryptedtext[i] == 0 && decryptedtext[i - 1] == 0)
                    break;
            }
            // storing plain text in outputtext.txt file
            cout << endl;
            ofstream fout; // Create Object of Ofstream
            ifstream fin;
            fin.open("outputtext.txt");
            fout.open("outputtext.txt", ios::app); // Append mode
            if (fin.is_open())
                fout << decryptedtext << "\n"; // Writing data to file

            fin.close();
            fout.close(); // Closing the file
            usleep(500);
        }
    }
    else
    {
        cout << "Can not open input file\n ";
        }
        cout << "\n Data has been appended to file outputtext.txt";
        MyReadFile.close();
        break;
    }
    }
}
```

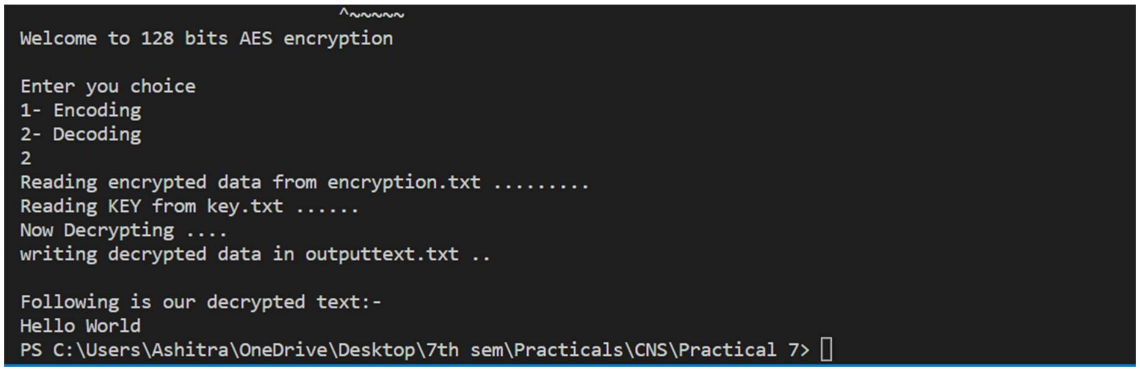
Output Snapshots:



```
Hello World
```



```
                                  ^~~~~~
Welcome to 128 bits AES encryption

Enter you choice
1- Encoding
2- Decoding
1
Reading plain text from input.txt .........
Reading KEY from key.txt ......
Now encrypting ....
writing encrypted data in encryption.aes ..

128-bit AES encryption is done sucessfully
Data has been appended to file encryption.aes
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Practical 7>
```



```
                                  ^~~~~~
Welcome to 128 bits AES encryption

Enter you choice
1- Encoding
2- Decoding
2
Reading encrypted data from encryption.txt .........
Reading KEY from key.txt ......
Now Decrypting ....
writing decrypted data in outputtext.txt ..

Following is our decrypted text:-
Hello World
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Practical 7>
```

Conclusion:

1. Advanced encryption standard (AES) algorithm is one of the efficient algorithm and it is widely supported and adopted on hardware and software.
2. This algorithm enables to deal with different key sizes such as 128, 192, and 256 bits with 128 bits block cipher.
3. AES has the ability to provide much more security compared to other algorithms like DES, 3DES etc.