Practical 1

➢ Implementation of SDES algorithm in C language.

Keygen.h

```c
// HEADER FILE FOR KEY GENERATION

#include <stdio.h>
#include <math.h>
#include <string.h>

void permuteKey(char key[], char permutedKey[])
{
    int i;
    int permutationTable[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

    for (i = 0; i < 10; i++)
    {
        permutedKey[i] = key[permutationTable[i] - 1];
    }
}

void generateLeftRightKey(char key[], char leftKey[], char rightKey[])
{
    int i;

    for (i = 0; i < 5; i++)
    {
        leftKey[i] = key[i];
    }

    for (i = 5; i < 10; i++)
    {
        rightKey[i - 5] = key[i];
    }
}

void leftShift(char key[], int shift)
{
    int i;
    char tempKey[5];

    strcpy(tempKey, key);

    for (i = 0; i < 5; i++)
    {
        key[i] = tempKey[(i + shift) % 5];
    }
}

void p8(char key[], char key1[])
{
    int i;
    int permutationTable[8] = {6, 3, 7, 4, 8, 5, 10, 9};
```

```c
    for (i = 0; i < 8; i++)
    {
        key1[i] = key[permutationTable[i] - 1];
    }

    key1[i] = '\0';
}

void generateKey1(char leftKey[], char rightKey[], char key1[])
{
    int i;
    char tempKey[10];

    leftShift(leftKey, 1);
    leftShift(rightKey, 1);

    for (i = 0; i < 5; i++)
    {
        tempKey[i] = leftKey[i];
    }

    for (i = 5; i < 10; i++)
    {
        tempKey[i] = rightKey[i - 5];
    }

    p8(tempKey, key1);
}

void generateKey2(char leftKey[], char rightKey[], char key2[])
{
    int i;
    char tempKey[10];

    leftShift(leftKey, 2);
    leftShift(rightKey, 2);

    for (i = 0; i < 5; i++)
    {
        tempKey[i] = leftKey[i];
    }

    for (i = 5; i < 10; i++)
    {
        tempKey[i] = rightKey[i - 5];
    }

    p8(tempKey, key2);
}

void generateKeys(char key[], char key1[], char key2[])
{
    char leftKey[5], rightKey[5], permutedKey[10];
```

```
    permuteKey(key, permutedKey);
    generateLeftRightKey(permutedKey, leftKey, rightKey);
    generateKey1(leftKey, rightKey, key1);
    generateKey2(leftKey, rightKey, key2);
}
```

Encrypt.h

```
// HEADER FILE FOR ENCRYPTION

#include <stdio.h>
#include <math.h>
#include <string.h>

void charToBinary(char input, char *output)
{
    int i;
    if (input == 0) {
        for (i = 0; i < 8; i++)
        {
            output[i] = '0';
        }
    }
    else {
        for (i = 8; i > 0; i--)
        {
            if (input % 2 == 0)
            {
                output[i - 1] = '0';
            }
            else
            {
                output[i - 1] = '1';
            }
            input = input / 2;
        }
    }
    output[8] = '\0';
}

void binaryToChar(char *input, char *output)
{
    int i;
    *output = 0;
    for (i = 0; i < 8; i++)
    {
        *output = *output + (input[i] - '0') * pow(2, 7 - i);
    }
}

void ip8(char *input, char *output)
{
    int i;
```

```
    int ip[] = {2, 6, 3, 1, 4, 8, 5, 7};
    for (i = 0; i < 8; i++)
    {
        output[i] = input[ip[i] - 1];
    }

    output[8] = '\0';
}

void devide(char *input, char *left, char *right)
{
    int i;
    for (i = 0; i < 4; i++)
    {
        left[i] = input[i];
        right[i] = input[i + 4];
    }

    left[4] = '\0';
    right[4] = '\0';
}

void ep(char *input, char *output)
{
    int i;
    int ep[] = {4, 1, 2, 3, 2, 3, 4, 1};
    for (i = 0; i < 8; i++)
    {
        output[i] = input[ep[i] - 1];
    }

    output[8] = '\0';
}

void xor8bit(char *input1, char *input2, char *output)
{
    int i;
    for (i = 0; i < 8; i++)
    {
        output[i] = (input1[i] - '0') ^ (input2[i] - '0') + '0';
    }

    output[8] = '\0';
}

void sBox(char *input, char *output)
{
    int s0[4][4] = {{1, 0, 3, 2},
                    {3, 2, 1, 0},
                    {0, 2, 1, 3},
                    {3, 1, 3, 2}};

    int s1[4][4] = {{0, 1, 2, 3},
                    {2, 0, 1, 3},
                    {3, 0, 1, 0},
```

```c
                        {2, 1, 0, 3}};

    int i = (input[0] - '0') * 2 + (input[3] - '0');
    int j = (input[1] - '0') * 2 + (input[2] - '0');
    int k = (input[4] - '0') * 2 + (input[7] - '0');
    int l = (input[5] - '0') * 2 + (input[6] - '0');

    output[0] = s0[i][j] / 2 + '0';
    output[1] = s0[i][j] % 2 + '0';
    output[2] = s1[k][l] / 2 + '0';
    output[3] = s1[k][l] % 2 + '0';

    output[4] = '\0';
}

void p4(char *input, char *output)
{
    int i;
    int p[] = {2, 4, 3, 1};
    for (i = 0; i < 4; i++)
    {
        output[i] = input[p[i] - 1];
    }

    output[4] = '\0';
}

void xor4bit(char *input1, char *input2, char *output)
{
    int i;
    for (i = 0; i < 4; i++)
    {
        output[i] = (input1[i] - '0') ^ (input2[i] - '0') + '0';
    }

    output[4] = '\0';
}

void combine(char *input1, char *input2, char *output)
{
    int i;
    for (i = 0; i < 4; i++)
    {
        output[i] = input1[i];
        output[i + 4] = input2[i];
    }

    output[8] = '\0';
}

void swap(char *input, char *output)
{
    int i;
    char temp;
```

```c
    for (i = 0; i < 4; i++)
    {
        temp = input[i];
        input[i] = output[i];
        output[i] = temp;
    }

    output[4] = '\0';
}

void ip8Inverse(char *input, char *output)
{
    int i;
    int ip[] = {4, 1, 3, 5, 7, 2, 8, 6};
    for (i = 0; i < 8; i++)
    {
        output[i] = input[ip[i] - 1];
    }

    output[8] = '\0';
}

void encryptForK(char *bin, char *key, char *output)
{
    // Step 2: IP8
    char ip8Output[9];
    ip8(bin, ip8Output);

    // Step 3: Devide 4n4
    char ip8Left[5], ip8Right[5];
    devide(ip8Output, ip8Left, ip8Right);

    // Step 4: EP
    char epOutput[9];
    ep(ip8Right, epOutput);

    // Step 5: XOR 8-bit
    char xorOutput[9];
    xor8bit(epOutput, key, xorOutput);

    // Step 6 and 7: S-Box
    char sBoxOutput[5];
    sBox(xorOutput, sBoxOutput);

    // Step 8: P4
    char p4Output[5];
    p4(sBoxOutput, p4Output);

    // Step 9:
    char xorOutput2[5];
    xor4bit(p4Output, ip8Right, xorOutput2);

    // Step 10: Combine S3
    char combineOutput[9];
```

```c
    combine(ip8Left, xorOutput2, combineOutput);

    // Step 11: Devide 4n4
    char combineLeft[5], combineRight[5];
    devide(combineOutput, combineLeft, combineRight);

    // Step 12: Swap
    swap(combineLeft, combineRight);

    // Step 13: Combine step 12 and generate output
    char combineOutput2[9];
    combine(combineLeft, combineRight, combineOutput2);

    // Copy the combined output to the final output
    strcpy(output, combineOutput2);
}

void encrypt(char c, char *k1, char *k2, char *output)
{
    // Step 1: Char to Binary
    char bin[9];
    charToBinary(c, bin);
    char output1[9], output2[9], output3[9];

    encryptForK(bin, k1, output1);
    encryptForK(output1, k2, output2);

    // Final inverse permutation
    ip8Inverse(output2, output3);

    output3[8] = '\0';

    // Copy the final output to the provided output buffer
    strcpy(output, output3);
}
```

SDES.c

```c
// SDES implementation in C
// REFERENCE: https://www.c-sharpcorner.com/article/s-des-or-simplified-data-encryption-standard/

#include <stdio.h>
#include <string.h>
#include "keygen.h"
#include "encrypt.h"

char key[10];
char key1[9];
char key2[9];

char inputString[100];
char output[9];
```

```c
char outputBits[900];


void main()
{
    int i;
    printf("Enter the 10 bit key: ");
    scanf("%s", &key);

    generateKeys(key, key1, key2);

    printf("Key 1: %s\n", key1);
    printf("Key 2: %s\n", key2);

    printf("Enter input string: ");
    scanf("%s", &inputString);

    int len = strlen(inputString);

    printf("Encrypted string: ");

    for (i = 0; i < len; i++)
    {
        encrypt(inputString[i], key1, key2, output);
        for (int j = 0; j < 8; j++)
        {
            outputBits[i * 8 + j] = output[j];
        }
    }

    printf("%s\n", outputBits);
}
```

OUTPUT

```
Enter the 10 bit key: 1010000010
Key 1: 10100100
Key 2: 01000011
Enter input string: Jatin
Encrypted string: 0101110001100100000100010000001001111101
```

Practical 2

Keygen.h

```c
// HEADER FILE FOR KEY GENERATION

#include <stdio.h>
#include <math.h>
#include <string.h>

void permuteKey(char key[], char permutedKey[])
{
    int i;
    int permutationTable[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

    for (i = 0; i < 10; i++)
    {
        permutedKey[i] = key[permutationTable[i] - 1];
    }
}

void generateLeftRightKey(char key[], char leftKey[], char rightKey[])
{
    int i;

    for (i = 0; i < 5; i++)
    {
        leftKey[i] = key[i];
    }

    for (i = 5; i < 10; i++)
    {
        rightKey[i - 5] = key[i];
    }
}

void leftShift(char key[], int shift)
{
    int i;
    char tempKey[5];

    strcpy(tempKey, key);

    for (i = 0; i < 5; i++)
    {
        key[i] = tempKey[(i + shift) % 5];
    }
}

void p8(char key[], char key1[])
{
    int i;
    int permutationTable[8] = {6, 3, 7, 4, 8, 5, 10, 9};
```

```c
    for (i = 0; i < 8; i++)
    {
        key1[i] = key[permutationTable[i] - 1];
    }

    key1[i] = '\0';
}

void generateKey1(char leftKey[], char rightKey[], char key1[])
{
    int i;
    char tempKey[10];

    leftShift(leftKey, 1);
    leftShift(rightKey, 1);

    for (i = 0; i < 5; i++)
    {
        tempKey[i] = leftKey[i];
    }

    for (i = 5; i < 10; i++)
    {
        tempKey[i] = rightKey[i - 5];
    }

    p8(tempKey, key1);
}

void generateKey2(char leftKey[], char rightKey[], char key2[])
{
    int i;
    char tempKey[10];

    leftShift(leftKey, 2);
    leftShift(rightKey, 2);

    for (i = 0; i < 5; i++)
    {
        tempKey[i] = leftKey[i];
    }

    for (i = 5; i < 10; i++)
    {
        tempKey[i] = rightKey[i - 5];
    }

    p8(tempKey, key2);
}

void generateKeys(char key[], char key1[], char key2[])
{
    char leftKey[5], rightKey[5], permutedKey[10];
```

```c
    permuteKey(key, permutedKey);
    generateLeftRightKey(permutedKey, leftKey, rightKey);
    generateKey1(leftKey, rightKey, key1);
    generateKey2(leftKey, rightKey, key2);
}
```

Encrypt.h

```c
// HEADER FILE FOR ENCRYPTION

#include <stdio.h>
#include <math.h>
#include <string.h>

void charToBinary(char input, char *output)
{
    int i;
    if (input == 0) {
        for (i = 0; i < 8; i++)
        {
            output[i] = '0';
        }
    }
    else {
        for (i = 8; i > 0; i--)
        {
            if (input % 2 == 0)
            {
                output[i - 1] = '0';
            }
            else
            {
                output[i - 1] = '1';
            }
            input = input / 2;
        }
    }
    output[8] = '\0';
}

void binaryToChar(char *input, char *output)
{
    int i;
    *output = 0;
    for (i = 0; i < 8; i++)
    {
        *output = *output + (input[i] - '0') * pow(2, 7 - i);
    }
}

void ip8(char *input, char *output)
{
    int i;
```

```c
    int ip[] = {2, 6, 3, 1, 4, 8, 5, 7};
    for (i = 0; i < 8; i++)
    {
        output[i] = input[ip[i] - 1];
    }

    output[8] = '\0';
}

void devide(char *input, char *left, char *right)
{
    int i;
    for (i = 0; i < 4; i++)
    {
        left[i] = input[i];
        right[i] = input[i + 4];
    }

    left[4] = '\0';
    right[4] = '\0';
}

void ep(char *input, char *output)
{
    int i;
    int ep[] = {4, 1, 2, 3, 2, 3, 4, 1};
    for (i = 0; i < 8; i++)
    {
        output[i] = input[ep[i] - 1];
    }

    output[8] = '\0';
}

void xor8bit(char *input1, char *input2, char *output)
{
    int i;
    for (i = 0; i < 8; i++)
    {
        output[i] = (input1[i] - '0') ^ (input2[i] - '0') + '0';
    }

    output[8] = '\0';
}

void sBox(char *input, char *output)
{
    int s0[4][4] = {{1, 0, 3, 2},
                    {3, 2, 1, 0},
                    {0, 2, 1, 3},
                    {3, 1, 3, 2}};

    int s1[4][4] = {{0, 1, 2, 3},
                    {2, 0, 1, 3},
                    {3, 0, 1, 0},
```

```c
                    {2, 1, 0, 3}};

    int i = (input[0] - '0') * 2 + (input[3] - '0');
    int j = (input[1] - '0') * 2 + (input[2] - '0');
    int k = (input[4] - '0') * 2 + (input[7] - '0');
    int l = (input[5] - '0') * 2 + (input[6] - '0');

    output[0] = s0[i][j] / 2 + '0';
    output[1] = s0[i][j] % 2 + '0';
    output[2] = s1[k][l] / 2 + '0';
    output[3] = s1[k][l] % 2 + '0';

    output[4] = '\0';
}

void p4(char *input, char *output)
{
    int i;
    int p[] = {2, 4, 3, 1};
    for (i = 0; i < 4; i++)
    {
        output[i] = input[p[i] - 1];
    }

    output[4] = '\0';
}

void xor4bit(char *input1, char *input2, char *output)
{
    int i;
    for (i = 0; i < 4; i++)
    {
        output[i] = (input1[i] - '0') ^ (input2[i] - '0') + '0';
    }

    output[4] = '\0';
}

void combine(char *input1, char *input2, char *output)
{
    int i;
    for (i = 0; i < 4; i++)
    {
        output[i] = input1[i];
        output[i + 4] = input2[i];
    }

    output[8] = '\0';
}

void swap(char *input, char *output)
{
    int i;
    char temp;
```

```c
    for (i = 0; i < 4; i++)
    {
        temp = input[i];
        input[i] = output[i];
        output[i] = temp;
    }

    output[4] = '\0';
}

void ip8Inverse(char *input, char *output)
{
    int i;
    int ip[] = {4, 1, 3, 5, 7, 2, 8, 6};
    for (i = 0; i < 8; i++)
    {
        output[i] = input[ip[i] - 1];
    }

    output[8] = '\0';
}

void encryptForK(char *bin, char *key, char *output)
{
    // Step 2: IP8
    char ip8Output[9];
    ip8(bin, ip8Output);

    // Step 3: Devide 4n4
    char ip8Left[5], ip8Right[5];
    devide(ip8Output, ip8Left, ip8Right);

    // Step 4: EP
    char epOutput[9];
    ep(ip8Right, epOutput);

    // Step 5: XOR 8-bit
    char xorOutput[9];
    xor8bit(epOutput, key, xorOutput);

    // Step 6 and 7: S-Box
    char sBoxOutput[5];
    sBox(xorOutput, sBoxOutput);

    // Step 8: P4
    char p4Output[5];
    p4(sBoxOutput, p4Output);

    // Step 9:
    char xorOutput2[5];
    xor4bit(p4Output, ip8Right, xorOutput2);

    // Step 10: Combine S3
    char combineOutput[9];
```

```
    combine(ip8Left, xorOutput2, combineOutput);

    // Step 11: Devide 4n4
    char combineLeft[5], combineRight[5];
    devide(combineOutput, combineLeft, combineRight);

    // Step 12: Swap
    swap(combineLeft, combineRight);

    // Step 13: Combine step 12 and generate output
    char combineOutput2[9];
    combine(combineLeft, combineRight, combineOutput2);

    // Copy the combined output to the final output
    strcpy(output, combineOutput2);
}

void encrypt(char c, char *k1, char *k2, char *output)
{
    // Step 1: Char to Binary
    char bin[9];
    charToBinary(c, bin);
    char output1[9], output2[9], output3[9];

    encryptForK(bin, k1, output1);
    encryptForK(output1, k2, output2);

    // Final inverse permutation
    ip8Inverse(output2, output3);

    output3[8] = '\0';

    // Copy the final output to the provided output buffer
    strcpy(output, output3);
}
```

BRUTEFORCE.c

```
// Implementation of brute force attack on the SDES algorithm

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "keygen.h"
#include "encrypt.h"

void numToKey(int num, char *key)
{
    // Initialize the key with all bits set to '0'
    for (int i = 0; i < 10; i++)
    {
        key[i] = '0';
    }
```

```c
    // Convert the number to binary and store it in the key
    for (int i = 9; i >= 0; i--)
    {
        key[i] = (num % 2) + '0';
        num = num / 2;
    }

    key[10] = '\0';
}

int main()
{
    FILE *filePointer;
    char inputStringFromFile[100];
    char outputBitsFromFile[900];
    char key[11], key1[9], key2[9];
    char output[8];
    char outputBits[900];

    int count = 1;

    filePointer = fopen("CTPT.txt", "r");

    if (filePointer == NULL)
    {
        printf("Error: Unable to open the file!\n");
        return 1; // Exit with error code
    }

    fscanf(filePointer, "%s", inputStringFromFile);
    fscanf(filePointer, "%s", outputBitsFromFile);

    printf("Input String from file: %s\n", inputStringFromFile);
    printf("Output Bits from file: %s\n", outputBitsFromFile);

    for (int i = 0; i < 1024; i++)
    {
        numToKey(i, key);

        generateKeys(key, key1, key2);

        int len = strlen(inputStringFromFile);

        for (int j = 0; j < len; j++)
        {
            encrypt(inputStringFromFile[j], key1, key2, output);
            for (int k = 0; k < 8; k++)
            {
                outputBits[j * 8 + k] = output[k];
            }
        }

        outputBits[len * 8] = '\0';
```

```c
        if (strcmp(outputBits, outputBitsFromFile) == 0)
        {
            printf("Key %d: %s\n",count++, key);
        }
    }

    fclose(filePointer); // Don't forget to close the file

    return 0;
}
```

CTPT.txt

SEM 6 > CRYPTO > BRUTEFORCE > CODE > CTPT.txt

```
j
01111100
```

OUTPUT

```
Input String from file: j
Output Bits from file: 01111100
Key 1: 0000001111
Key 2: 0001000111
Key 3: 0101010001
Key 4: 1000011011
Key 5: 1001010011
Key 6: 1010000010
Key 7: 1011001010
```

Practical 3

➢ Implementation of Extended Euclidean Algorithm

ExtendedEuclidean.c

```c
#include <stdio.h>

int t1 = 0, t2 = 1, q, r, t;

int extendedEuclidean(int a, int b)
{
    while (b != 0)
    {
        q = a / b;
        r = a % b;
        t = t1 - t2 * q;
        t1 = t2;
        t2 = t;
        a = b;
        b = r;
    }

    return a != 1 ? 0 : t1;
}

int main()
{

    int a, b, result;
    printf("Inverse of: ");
    scanf("%d", &a);
    printf("With modulo: ");
    scanf("%d", &b);

    result = extendedEuclidean(b, a);

    if (result == 0)
    {
        printf("Inverse does not exist\n");
        return 0;
    }

    else if (result < 0)
    {
        result += b;
    }

    printf("Inverse: %d\n", result);
    return 0;
}
```

OUTPUT

```
Inverse of: 15
With modulo: 26
Inverse: 7
```

Practical 4

➢ Implementation of RSA with all required algorithms

rsa.c

```c
#include <stdio.h>
#include <math.h>

int isPrime(int n)
{
    int i;
    for (i = 2; i < sqrt(n); i++)
    {
        if (n % i == 0)
        {
            return 0;
        }
    }
    return 1;
}

int gcd(int a, int b)
{
    if (b == 0)
    {
        return a;
    }

    return gcd(b, a % b);
}

int extendedEuclidean(int a, int b)
{
    int t1 = 0, t2 = 1, q, r, t;
    while (b != 0)
    {
        q = a / b;
        r = a % b;
        t = t1 - t2 * q;
        t1 = t2;
        t2 = t;
        a = b;
        b = r;
    }

    return a != 1 ? 0 : t1;
}

int fastExponentiation(int a, int b, int n)
{
    int bitCount = (int)ceil(log2(b)); // Cast log2 result to int

    int aBinary[bitCount];
    int modValues[bitCount];
```

```c
    int i;
    int result = 1; // Initialize result to 1

    for (i = bitCount - 1; i >= 0; i--)
    {
        aBinary[i] = b % 2;
        b = b / 2;
    }

    for (i = bitCount - 1; i >= 0; i--)
    {
        if (i == bitCount - 1) // Change i == 0 to i == bitCount - 1
        {
            modValues[i] = a % n;
        }
        else
        {
            modValues[i] = (modValues[i + 1] * modValues[i + 1]) % n; //
Change i - 1 to i + 1
        }
    }

    for (i = 0; i < bitCount; i++)
    {
        if (aBinary[i] == 1)
        {
            result = (result * modValues[i]) % n; // Add modulo operation
here
        }
    }
    return result;
}

int main()
{

    int p, q, e, d, n, phi;

    int m, c;

    printf("Enter the value of p: ");
    scanf("%d", &p);

    if (isPrime(p) == 0)
    {
        printf("p is not prime\n");
        return 0;
    }

    printf("Enter the value of q: ");
    scanf("%d", &q);

    if (isPrime(q) == 0)
    {
        printf("q is not prime\n");
```

```c
        return 0;
    }

    n = p * q;

    phi = (p - 1) * (q - 1);

    printf("Enter the value of e between 1 and %d: ", phi);
    scanf("%d", &e);

    if (e < 1 || e > phi)
    {
        printf("Invalid value of e\n");
        return 0;
    }

    if (gcd(e, phi) != 1)
    {
        printf("Invalid value of e\n");
        return 0;
    }

    d = extendedEuclidean(phi, e);

    if (d < 0)
    {
        d += phi;
    }

    printf("Public key: (%d, %d)\n", e, n);
    printf("Private key: (%d, %d)\n", d, n);

    printf("Enter the message: ");
    scanf("%d", &m);

    c = fastExponentiation(m, e, n);

    printf("Encrypted message: %d\n", c);

    m = fastExponentiation(c, d, n);

    printf("Decrypted message: %d\n", m);

    return 0;
}
```

OUTPUT

```
Enter the value of p: 53
Enter the value of q: 59
Enter the value of e between 1 and 3016: 3
Public key: (3, 3127)
Private key: (2011, 3127)
Enter the message: 10
Encrypted message: 1000
Decrypted message: 10
```

Practical 5

➢ Demonstration of CCA2 attack on RSA

cca2.c

```c
// CCA2 attak on RSA

#include <stdio.h>
#include <math.h>

int isPrime(int n)
{
    int i;
    for (i = 2; i < sqrt(n); i++)
    {
        if (n % i == 0)
        {
            return 0;
        }
    }
    return 1;
}

int gcd(int a, int b)
{
    if (b == 0)
    {
        return a;
    }

    return gcd(b, a % b);
}

int extendedEuclidean(int a, int b)
{
    int t1 = 0, t2 = 1, q, r, t;
    while (b != 0)
    {
        q = a / b;
        r = a % b;
        t = t1 - t2 * q;
        t1 = t2;
        t2 = t;
        a = b;
        b = r;
    }

    return a != 1 ? 0 : t1;
}

int fastExponentiation(int a, int b, int n)
{
    int bitCount = (int)ceil(log2(b)); // Cast log2 result to int

    int aBinary[bitCount];
```

```
    int modValues[bitCount];

    int i;
    int result = 1; // Initialize result to 1

    for (i = bitCount - 1; i >= 0; i--)
    {
        aBinary[i] = b % 2;
        b = b / 2;
    }

    for (i = bitCount - 1; i >= 0; i--)
    {
        if (i == bitCount - 1) // Change i == 0 to i == bitCount - 1
        {
            modValues[i] = a % n;
        }
        else
        {
            modValues[i] = (modValues[i + 1] * modValues[i + 1]) % n; //
Change i - 1 to i + 1
        }
    }

    for (i = 0; i < bitCount; i++)
    {
        if (aBinary[i] == 1)
        {
            result = (result * modValues[i]) % n; // Add modulo operation
here
        }
    }
    return result;
}

int main()
{

    int p, q, e, d, n, phi;

    int m, c;

    int m1, c1, decOfC1, mDec;

    printf("\n-----------------CHALLENGER WILL COMPUTE CIPHER TEXT-----------
-------\n\n");

    do
    {
        printf("Enter the value of p: ");
        scanf("%d", &p);
    } while (!isPrime(p));

    do
    {
```

```
        printf("Enter the value of q: ");
        scanf("%d", &q);
    } while (!isPrime(q));

    n = p * q;

    printf("\nValue of n: %d\n", n);

    phi = (p - 1) * (q - 1);

    printf("Value of phi: %d\n", phi);

    do
    {
        printf("\nEnter the value of e between 1 and %d: ", phi);
        scanf("%d", &e);
    } while (e < 1 || e > phi || gcd(e, phi) != 1);

    d = extendedEuclidean(phi, e);

    if (d < 0)
    {
        d += phi;
    }

    printf("Value of d: %d\n", d);

    printf("\nPublic key: (%d, %d)\n", e, n);
    printf("Private key: (%d, %d)\n", d, n);

    printf("\nEnter the message: ");
    scanf("%d", &m);

    c = fastExponentiation(m, e, n);

    printf("Encrypted message: %d\n", c);

    printf("\n\n-----------------CHALLENGER WILL GIVE %d TO ADVERSARY--------
---------\n\n", c);

    printf("\n-----------------ADVERSARY WILL CHOOSE M1-----------------
\n\n");

    printf("Enter the message: ");
    scanf("%d", &m1);
    printf("\n");

    printf("\n-----------------C1 = C*M1^e mod n-----------------\n\n");

    c1 = (c * fastExponentiation(m1, e, n)) % n;

    printf("Encrypted message: %d\n", c1);

    printf("\n\n-----------------ADVERSARY WILL QUERY DECRYPTION OF %d-------
----------\n\n", c1);

    decOfC1 = fastExponentiation(c1, d, n);
```

```
    printf("Decrypted message: %d\n", decOfC1);

    printf("\n\n-----------------ADVERSARY COMPUTES M1 INVERSE WITH RESPECT
TO N-----------------\n\n");

    int m1Inverse = extendedEuclidean(n, m1);

    if (m1Inverse < 0)
    {
        m1Inverse += n;
    }

    printf("M1 inverse: %d\n", m1Inverse);

    printf("\n\n-----------------ADVERSARY WILL COMPUTE M = C1*M1^-1 mod n---
--------------\n\n");

    mDec = (decOfC1 * m1Inverse) % n;

    printf("Decrypted message: %d\n\n", mDec);

    if (mDec == m)
    {
        printf("Adversary has successfully decrypted the message\n");
    }
    else
    {
        printf("Adversary has failed to decrypt the message\n");
    }

    return 0;
}
```

OUTPUT

```
-----------------CHALLENGER WILL COMPUTE CIPHER TEXT-----------------

Enter the value of p: 53
Enter the value of q: 59

Value of n: 3127
Value of phi: 3016

Enter the value of e between 1 and 3016: 3
Value of d: 2011

Public key: (3, 3127)
Private key: (2011, 3127)

Enter the message: 10
Encrypted message: 1000


-----------------CHALLENGER WILL GIVE 1000 TO ADVERSARY-----------------


-----------------ADVERSARY WILL CHOOSE M1-----------------

Enter the message: 61


-----------------C1 = C*M1^e mod n-----------------

Encrypted message: 1451


-----------------ADVERSARY WILL QUERY DECRYPTION OF 1451-----------------

Decrypted message: 610


-----------------ADVERSARY COMPUTES M1 INVERSE WITH RESPECT TO N-----------------

M1 inverse: 974


-----------------ADVERSARY WILL COMPUTE M = C1*M1^-1 mod n-----------------

Decrypted message: 10
```

Practical 6

➢ Implementation of Elgamal Algorithm for Encryption

elgamal.c

```c
#include <stdio.h>
#include <math.h>

int fastExponentiation(int a, int b, int n)
{
    int bitCount = (int)ceil(log2(b)); // Cast log2 result to int

    int aBinary[bitCount];
    int modValues[bitCount];

    int i;
    int result = 1; // Initialize result to 1

    for (i = bitCount - 1; i >= 0; i--)
    {
        aBinary[i] = b % 2;
        b = b / 2;
    }

    for (i = bitCount - 1; i >= 0; i--)
    {
        if (i == bitCount - 1) // Change i == 0 to i == bitCount - 1
        {
            modValues[i] = a % n;
        }
        else
        {
            modValues[i] = (modValues[i + 1] * modValues[i + 1]) % n; //
Change i - 1 to i + 1
        }
    }

    for (i = 0; i < bitCount; i++)
    {
        if (aBinary[i] == 1)
        {
            result = (result * modValues[i]) % n; // Add modulo operation
here
        }
    }
    return result;
}

void main()
{
    int p, g, x, y;
    int m, r, c1, c2, dec_m;

    printf("Enter the value of p: ");
    scanf("%d", &p);
```

```c
    printf("Enter the value of g: ");
    scanf("%d", &g);

    printf("\nEnter the private key x such that 1 < x < p-1: ");
    scanf("%d", &x);

    y = (int)pow(g, x) % p;

    printf("\nThe public key is: %d %d %d \n", p, g, y);
    printf("The private key is: %d %d %d \n", p, g, x);

    printf("\nEnter the message m: ");
    scanf("%d", &m);

    printf("\nEnter random number r such that 1 < r < p-1: ");
    scanf("%d", &r);

    c1 = fastExponentiation(g, r, p);
    c2 = fastExponentiation(y, r, p) * m % p;

    printf("\nThe cipher text is: %d %d \n", c1, c2);

    dec_m = c2 * fastExponentiation(c1, p - 1 - x, p) % p;

    printf("\nThe decrypted message is: %d \n", dec_m);
}
```

OUTPUT

```
Enter the value of p: 43
Enter the value of g: 3

Enter the private key x such that 1 < x < p-1: 7

The public key is: 43 3 37
The private key is: 43 3 7

Enter the message m: 14

Enter random number r such that 1 < r < p-1: 26

The cipher text is: 15 31

The decrypted message is: 14
```

Practical 7

➢ Implementation of Elgamal Algorithm for Digital Signature

Elgamal_digital_sign.c

```c
#include <stdio.h>
#include <math.h>

int fastExponentiation(int a, int b, int n)
{
    int result = 1;
    a = a % n;
    while (b > 0)
    {
        if (b % 2 == 1)
        {
            result = (result * a) % n;
        }
        b = b / 2;
        a = (a * a) % n;
    }
    return result;
}

int gcd(int a, int b)
{
    if (b == 0)
    {
        return a;
    }

    return gcd(b, a % b);
}

int hashFunction(int m, int p)
{
    return m * m % (p - 1);
}

int randomK(int p)
{
    for (int i = 2; i < p - 1; i++)
    {
        if (gcd(i, p - 1) == 1)
        {
            return i;
        }
    }
}

int inverse(int a, int b)
{
    for (int i = 1; i < b; i++)
    {
        if ((a * i) % b == 1)
```

```c
            {
                return i;
            }
        }
    }

    void main()
    {
        int p, g, x, y;
        int m, hash, k, s1, s2, kInv;
        int v1, v2;

        printf("-----------------SENDER SIDE-----------------\n\n");

        printf("Enter the value of p: ");
        scanf("%d", &p);

        printf("Enter the value of g: ");
        scanf("%d", &g);

        printf("\nEnter the private key x such that 1 < x < p-1: ");
        scanf("%d", &x);

        y = fastExponentiation(g, x, p);

        printf("\nThe public key is: %d %d %d \n", p, g, y);
        printf("The private key is: %d %d %d \n", p, g, x);

        printf("\nEnter the message m: ");
        scanf("%d", &m);

        // hash = hashFunction(m, p);
        hash = m;

        k = randomK(p);

        printf("The hash value is: %d\n", hash);
        printf("The random number k is: %d\n", k);

        kInv = inverse(k, p - 1);

        printf("The inverse of k is: %d\n", kInv);

        s1 = fastExponentiation(g, k, p);
        s2 = kInv * (hash - x * s1) % (p - 1);

        printf("The signature is: %d %d\n", s1, s2);

        printf("\n\n-----------------RECEIVER SIDE-----------------\n\n");

        v1 = fastExponentiation(g, hash, p);
        v2 = (fastExponentiation(y, s1, p) * fastExponentiation(s1, s2, p)) % p;

        printf("v1: %d\n", v1);
        printf("v2: %d\n", v2);
```

```c
    if (v1 == v2)
    {
        printf("The signature is verified\n");
    }
    else
    {
        printf("The signature is not verified\n");
    }
}
```

OUTPUT

```
-----------------SENDER SIDE-----------------

Enter the value of p: 19
Enter the value of g: 10

Enter the private key x such that 1 < x < p-1: 16

The public key is: 19 10 4
The private key is: 19 10 16

Enter the message m: 14
The hash value is: 16
The random number k is: 5
The inverse of k is: 11
The signature is: 3 -10


-----------------RECEIVER SIDE-----------------

v1: 7
v2: 4
The signature is not verified
```

Practical 8

➤ Implementation of your own designed pseudorandom number generation

random.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

unsigned long long int custom_rand(unsigned long long int seed)
{
    unsigned long long int multiplier = 6364136223846793005ULL;
    unsigned long long int increment = 1442695040888963407ULL;
    unsigned long long int modulus = 18446744073709551615ULL; // 2^64 - 1
(64-bit unsigned integer max value)

    // Calculate the next pseudo-random number using LCG algorithm
    seed = (multiplier * seed + increment) % modulus;

    return seed;
}

int main()
{
    unsigned long long int seed = 12345; // Initial seed value
    int i;

    int range_min = 0;
    int range_max = 100;

    // Use current time as seed for better randomness
    time_t t;
    time(&t);
    seed = t;

    // Generate and print 10 pseudo-random numbers within the specified range
    for (i = 0; i < 10; i++)
    {
        // Generate next pseudo-random number
        seed = custom_rand(seed);
        // map to the specified range
        printf("%d\n", range_min + (seed % (range_max - range_min + 1)));
    }

    return 0;
}
```

OUTPUT

```
45
99
24
22
98
38
3
33
71
65
```

Practical 9

➢ Implement Shamir secret sharing scheme

shamir.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

int random_number(long x, long a, long c, int m)
{
    // Calculate the next pseudo-random number using Linear Congruential
Generator (LCG) algorithm
    long result = (x * a + c) % m;
    int res = (int)result;
    return res;
}

int main()
{

    int S, n, k;
    printf("Enter Secret (S): ");
    scanf("%d", &S);
    printf("Enter Number Of Partitions(n): ");
    scanf("%d", &n);
    printf("Enter number of points(k): ");
    scanf("%d", &k);

    // Generate a random polynomial of degree (k-1) with coefficients from
LCG algorithm
    int polynomial[k];
    polynomial[0] = S;
    for (int i = 1; i < k; i++)
    {
        polynomial[i] = random_number(time(0), 17, time(0) + 3, 967);
    }

    // Generate random points on the polynomial
    int points[n][2];
    for (int i = 1; i < n + 1; i++)
    {
        points[i - 1][0] = i;
        int sum = 0;
        for (int j = 0; j < k; j++)
        {
            sum += pow(points[i - 1][0], j) * polynomial[j];
        }
        points[i - 1][1] = sum;
    }

    // Copy random points for Lagrange interpolation
    int random_points[k][2];
```

```
    for (int i = 0; i < k; i++)
    {
        random_points[i][0] = points[i][0];
        random_points[i][1] = points[i][1];
    }

    // Lagrange Interpolation (for finding S, the term without any
polynomial)
    double l[k];
    int found_Secret = 0;
    for (int i = 0; i < k; i++)
    {
        l[i] = 1.0;
        for (int j = 0; j < k; j++)
        {
            if (j != i)
            {
                // Calculate the Lagrange interpolation coefficients
                l[i] *= (double)-1 * (random_points[j][0]) /
(random_points[i][0] - random_points[j][0]);
            }
        }
        // Calculate the secret by summing up the Lagrange coefficients
multiplied by the y-values of the points
        found_Secret += (int)(l[i]) * random_points[i][1];
    }

    printf("\nSecret Found: %d\n", found_Secret);
    return 0;
}
```

OUTPUT

```
Enter Secret (S): 54683
Enter Number Of Partitions(n): 6
Enter number of points(k): 3

Secret Found: 54683
```

Practical 10

➢ Implementation of SHA 256 algorithm

Sha256.c

```c
#include <stdio.h>
#include <stdint.h>
#include <string.h>

// Rotate bits to the right
#define ROTR(x, n) (((x) >> (n)) | ((x) << (32 - (n))))

// SHA-256 Constants
const uint32_t K[64] = {
    0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
    0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
    0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
    0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
    0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
    0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
    0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
    0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
    0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2};

// SHA-256 initial hash values
const uint32_t initial_hash[8] = {
    0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a,
    0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19};

// SHA-256 Functions
#define CH(x, y, z) (((x) & (y)) ^ (~(x) & (z)))
#define MAJ(x, y, z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))
#define EP0(x) (ROTR(x, 2) ^ ROTR(x, 13) ^ ROTR(x, 22))
#define EP1(x) (ROTR(x, 6) ^ ROTR(x, 11) ^ ROTR(x, 25))
#define SIG0(x) (ROTR(x, 7) ^ ROTR(x, 18) ^ ((x) >> 3))
#define SIG1(x) (ROTR(x, 17) ^ ROTR(x, 19) ^ ((x) >> 10))

// SHA-256 Compression Function
void sha256_compress(uint32_t state[8], const uint8_t block[64])
{
    uint32_t W[64];
    uint32_t a, b, c, d, e, f, g, h;
    uint32_t T1, T2;

    // Prepare message schedule
    for (int t = 0; t < 16; ++t)
    {
```

```
        W[t] = (block[t * 4] << 24) | (block[t * 4 + 1] << 16) | (block[t * 4
+ 2] << 8) | block[t * 4 + 3];
    }
    for (int t = 16; t < 64; ++t)
    {
        W[t] = SIG1(W[t - 2]) + W[t - 7] + SIG0(W[t - 15]) + W[t - 16];
    }

    // Initialize working variables
    a = state[0];
    b = state[1];
    c = state[2];
    d = state[3];
    e = state[4];
    f = state[5];
    g = state[6];
    h = state[7];

    // Compression loop
    for (int t = 0; t < 64; ++t)
    {
        T1 = h + EP1(e) + CH(e, f, g) + K[t] + W[t];
        T2 = EP0(a) + MAJ(a, b, c);
        h = g;
        g = f;
        f = e;
        e = d + T1;
        d = c;
        c = b;
        b = a;
        a = T1 + T2;
    }

    // Update hash state
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
    state[4] += e;
    state[5] += f;
    state[6] += g;
    state[7] += h;
}

// SHA-256 hash function
void sha256(const uint8_t *message, size_t len, uint8_t hash[32])
{
    uint32_t state[8];
    uint32_t length_bits = len * 8;
    uint8_t block[64];
    size_t i;

    // Initialize hash state
    for (i = 0; i < 8; ++i)
    {
```

```c
            state[i] = initial_hash[i];
        }

        // Process message in 512-bit blocks
        while (len >= 64)
        {
            memcpy(block, message, 64);
            sha256_compress(state, block);
            message += 64;
            len -= 64;
        }

        // Append padding and length
        memset(block, 0, 64);
        memcpy(block, message, len);
        block[len] = 0x80;
        if (len < 56)
        {
            // There is enough room for padding and length in this block
            for (i = 0; i < 8; ++i)
            {
                block[56 + i] = (length_bits >> (56 - i * 8)) & 0xFF;
            }
            sha256_compress(state, block);
        }
        else
        {
            // We need an additional block to store padding and length
            sha256_compress(state, block);
            memset(block, 0, 64);
            for (i = 0; i < 8; ++i)
            {
                block[56 + i] = (length_bits >> (56 - i * 8)) & 0xFF;
            }
            sha256_compress(state, block);
        }

        // Write final hash
        for (i = 0; i < 8; ++i)
        {
            hash[i * 4] = (state[i] >> 24) & 0xFF;
            hash[i * 4 + 1] = (state[i] >> 16) & 0xFF;
            hash[i * 4 + 2] = (state[i] >> 8) & 0xFF;
            hash[i * 4 + 3] = state[i] & 0xFF;
        }
    }

    // Utility function to print hash in hexadecimal format
    void print_hash(const uint8_t hash[32])
    {
        for (int i = 0; i < 32; ++i)
        {
            printf("%02x", hash[i]);
        }
        printf("\n");
```

```c
}

int main()
{
    const char *message = "Hello, world!";
    uint8_t hash[32];
    sha256((uint8_t *)message, strlen(message), hash);
    printf("SHA-256 hash of '%s':\n", message);
    print_hash(hash);
    return 0;
}
```

OUTPUT

```
SHA-256 hash of 'Hello, world!':
e8662ae9c885cd93bc328d634ca36da611ac26f7632c9a939b665db581f5e674
```

Practical 11

➢ Implementation of PBC library and performing Bilinear pairing operation

Steps:

1. Install m4, flex and bison using below commands
   i. sudo apt-get install m4
   ii. sudo apt-get install flex
   iii. sudo apt-get install bison
2. Download gmp and install it using below commands
   i. configure
   ii. make
   iii. sudo make install
3. After installing gmp, go to https://crypto.stanford.edu/pbc/download.html and download pbc-0.5.14.tar.gz
4. Then go to pcb-0.5.14 folder and open the terminal in that folder
5. For performing bilinear pairing follow below commands
   i. configure
   ii. make
   iii. sudo make install
   iv. sudoldconfig
   v. pbc/pbc

6. Now for bilinear pairing go to location pcb-0.5.14/example and open terminal in that folder then write below commands
   i. gcc -o test bls.c -L. -lgmp -lpbc
   ii. ./test <~/Download/pbc-0.5.14/param/a.param

➤ CODE:

```c
#include <pbc/pbc.h>
#include <pbc/pbc_test.h>

int main(int argc, char **argv)
{
    pairing_t pairing;
    element_t g, h;
    element_t public_key, sig;
    element_t secret_key;
    element_t temp1, temp2;

    pbc_demo_pairing_init(pairing, argc, argv);

    element_init_G2(g, pairing);
    element_init_G2(public_key, pairing);
    element_init_G1(h, pairing);
    element_init_G1(sig, pairing);
    element_init_GT(temp1, pairing);
    element_init_GT(temp2, pairing);
    element_init_Zr(secret_key, pairing);

    printf("Short signature test\n");

    // generate system parameters
    element_random(g);
    element_printf("system parameter g = %B\n", g);

    // generate private key
    element_random(secret_key);
    element_printf("private key = %B\n", secret_key);

    // compute corresponding public key
    element_pow_zn(public_key, g, secret_key);
    element_printf("public key = %B\n", public_key);

    // generate element from a hash
    // for toy pairings, should check that pairing(g, h) != 1
```

```c
    element_from_hash(h, "hashofmessage", 13);
    element_printf("message hash = %B\n", h); // h^secret_key is the
signature // in real life: only output the first coordinate
    element_pow_zn(sig, h, secret_key);
    element_printf("signature = %B\n", sig);

    {
        int n = pairing_length_in_bytes_compressed_G1(pairing);
        // int n = element_length_in_bytes_compressed(sig);
        int i;
        unsigned char *data = pbc_malloc(n);
        element_to_bytes_compressed(data, sig);
        printf("compressed = ");
        for (i = 0; i < n; i++)
        {
            printf("%02X", data[i]);
        }
        printf("\n");
        element_from_bytes_compressed(sig, data);
        element_printf("decompressed = %B\n", sig);
        pbc_free(data);
    }

    // Verification part 1
    element_pairing(temp1, sig, g);
    element_printf("f(sig, g) = %B\n", temp1);

    // Verification part 2
    // should match above
    element_pairing(temp2, h, public_key);
    element_printf("f(message hash, public_key) = %B\n", temp2);

    if (!element_cmp(temp1, temp2))
    {
        // Signature is valid
    }
    {
        printf("signature verifies\n");
    }
    else
    {
        printf("*BUG* signature does not verify *BUG*\n");
    }

    {
        int n = pairing_length_in_bytes_x_only_G1(pairing);
        // int n = element_length_in_bytes_x_only(sig);
        int i;
        unsigned char *data = pbc_malloc(n);
        element_to_bytes_x_only(data, sig);
        printf("x-coord = ");
        for (i = 0; i < n; i++)
        {
            printf("%02X", data[i]);
```

```c
        }
        printf("\n");
        element_from_bytes_x_only(sig, data);
        element_printf("de-x-ed = %B\n", sig);
        element_pairing(temp1, sig, g);
        if (!element_cmp(temp1, temp2))
        {
            printf("signature verifies on first guess\n");
        }
        else
        {
            element_invert(temp1, temp1);
            if (!element_cmp(temp1, temp2))
            {
                printf("signature verifies on second guess\n");
            }
            else
            {
                printf("*BUG* signature does not verify *BUG*\n");
            }
        }

        pbc_free(data);
    }

    // a random signature shouldn't verify
    element_random(sig);
    element_pairing(temp1, sig, g);
    if (element_cmp(temp1, temp2))
    {
        printf("random signature doesn't verify\n");
    }
    else
    {
        printf("*BUG* random signature verifies *BUG*\n");
    }

    element_clear(sig);
    element_clear(public_key);
    element_clear(secret_key);
    element_clear(g);
    element_clear(h);
    element_clear(temp1);
    element_clear(temp2);
    pairing_clear(pairing);
    return 0;
}
```

Practical 12

Wireshark:

Wireshark functions as a network protocol analyzer, capturing packets from various network connections such as those between your computer and your home office or the internet. In the context of a typical Ethernet network, a packet refers to a discrete unit of data. Wireshark stands as the most commonly utilized packet sniffer globally. Like its counterparts, Wireshark performs three primary tasks:

1. Packet Capture: Wireshark actively listens to network connections in real-time, capturing entire streams of traffic which may encompass tens of thousands of packets simultaneously.
2. Filtering: Utilizing filters, Wireshark effectively sifts through the abundance of live data, enabling users to isolate specific information of interest.
3. Visualization: Wireshark not only permits users to delve into individual network packets but also facilitates the visualization of entire conversations and network streams, offering a comprehensive understanding of network activities.

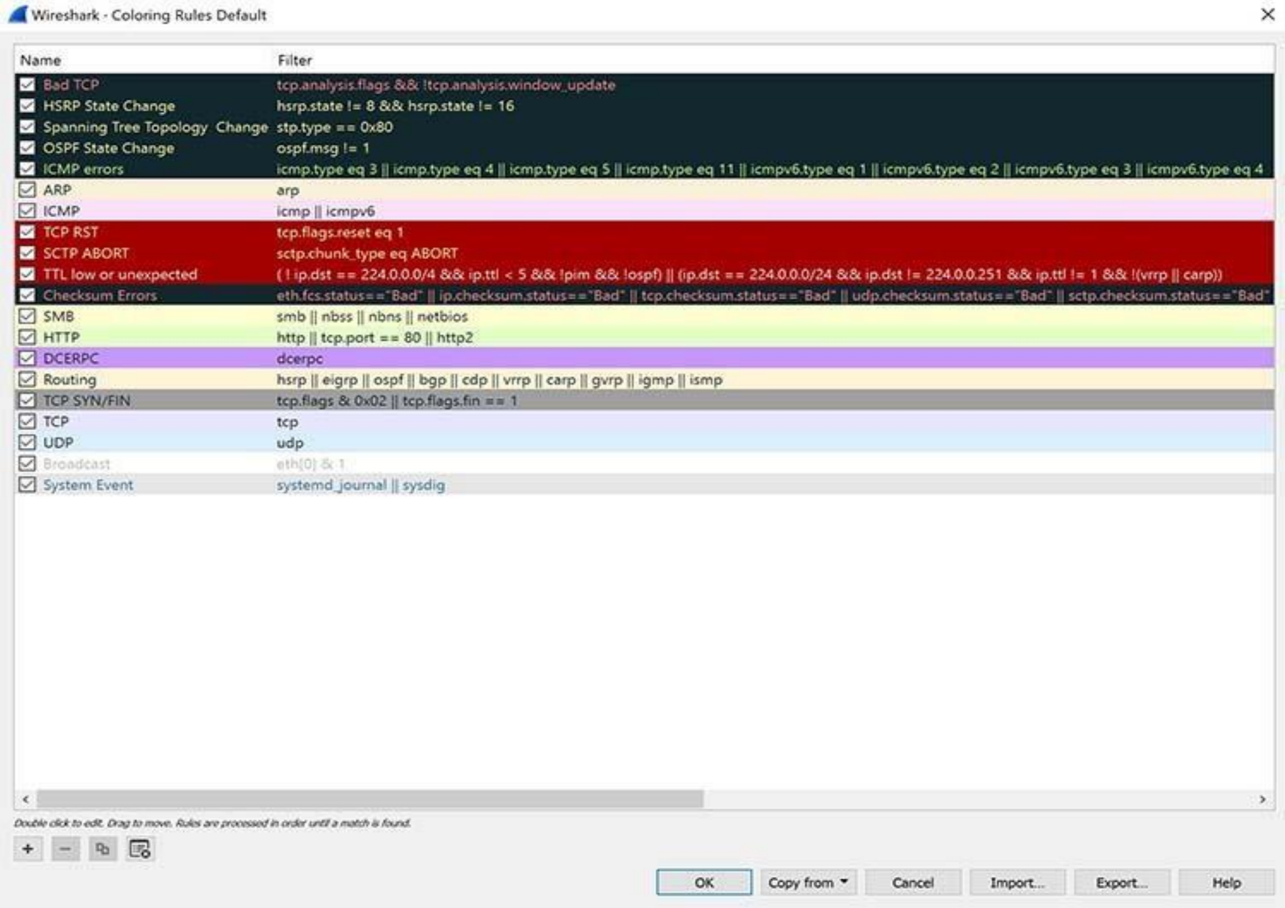
What are the Applications of Wireshark?

Wireshark serves various purposes, one of which is troubleshooting networks experiencing performance issues. Cybersecurity experts frequently employ Wireshark to track connections, inspect the contents of suspicious network transactions, and detect spikes in network traffic. It constitutes a significant component of any IT professional's arsenal – provided the professional possesses the requisite expertise to utilize it effectively.

Interpreting Color Codes in Wireshark:

Once packets are captured, understanding their significance becomes paramount. Wireshark simplifies this task by employing intuitive color coding to denote different packet types. The following table delineates the default colors assigned to major packet categories.

Color in Wireshark	Packet Type
Light purple	TCP
Light blue	UDP
Black	Packets with errors
Light green	HTTP traffic
Light yellow	Windows-specific traffic, including Server Message Blocks (SMB) and NetBIOS
Dark yellow	Routing
Dark gray	TCP SYN, FIN and ACK traffic



When to Utilize Wireshark?

Wireshark serves as a reliable tool embraced by a spectrum of entities including government agencies, educational institutions, corporations, small businesses, and nonprofits for diagnosing network issues. Moreover, Wireshark doubles as an educational resource.

Novices in the realm of information security can leverage Wireshark to grasp concepts such as network traffic analysis, the dynamics of communication in the presence of specific protocols, and the identification of anomalies during troubleshooting.

However, Wireshark has its limitations. Firstly, it proves ineffective for users lacking a fundamental understanding of network protocols. Regardless of its utility, no tool can adequately substitute for comprehensive knowledge. Consequently, proficient usage of Wireshark mandates a deep comprehension of network operations, encompassing elements like the three-way TCP handshake and various protocols such as TCP, UDP, DHCP, and ICMP.

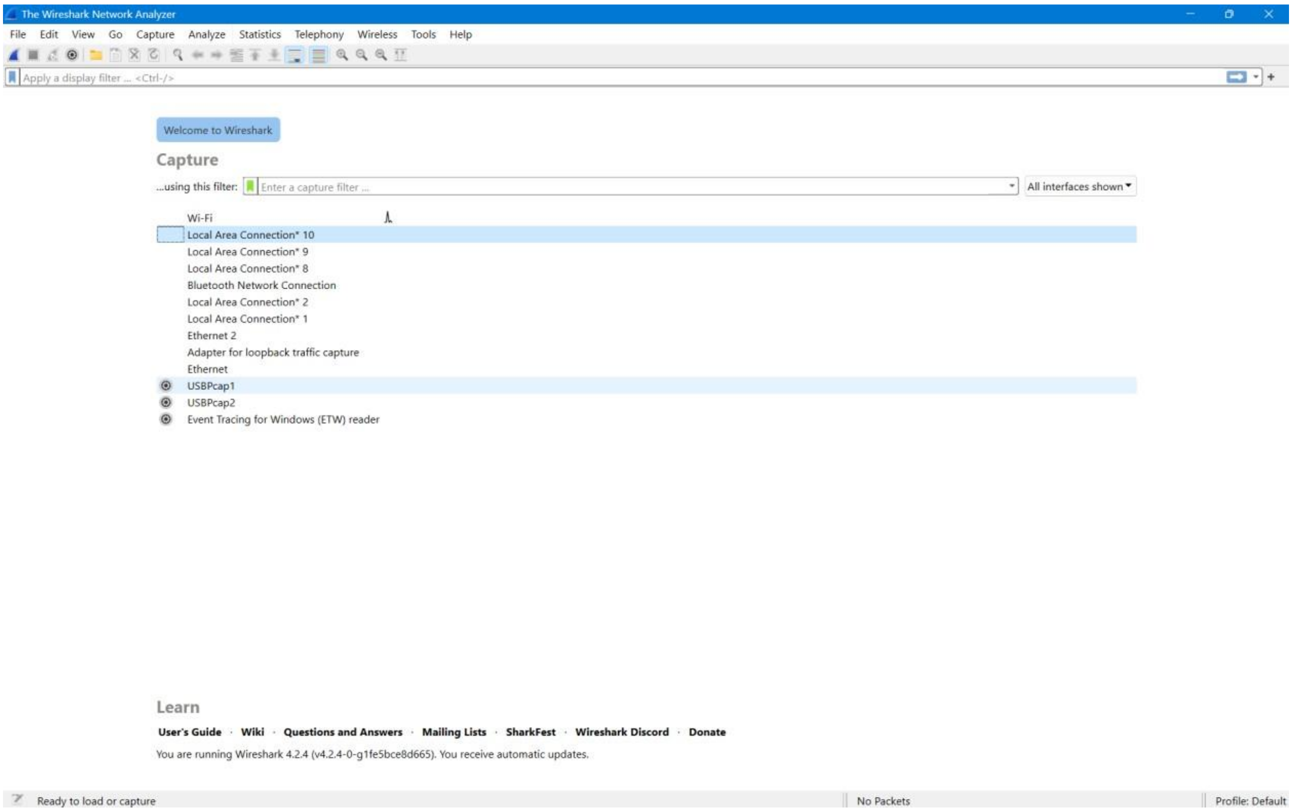
Secondly, under standard circumstances, Wireshark fails to capture traffic from all network systems. In modern network infrastructures employing switches, Wireshark, like other conventional packet-capturing tools, can only intercept traffic between the local computer and the remote system it engages with.

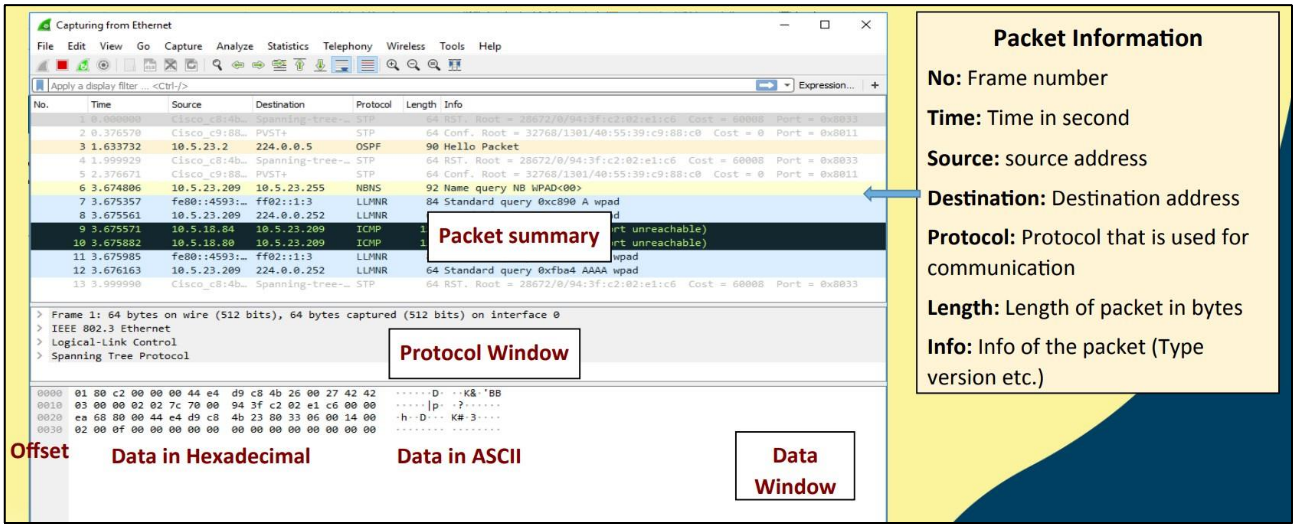
Thirdly, while Wireshark can detect malformed packets and implement color coding, it lacks actual alert functionalities, thus distinguishing itself from an intrusion detection system (IDS).

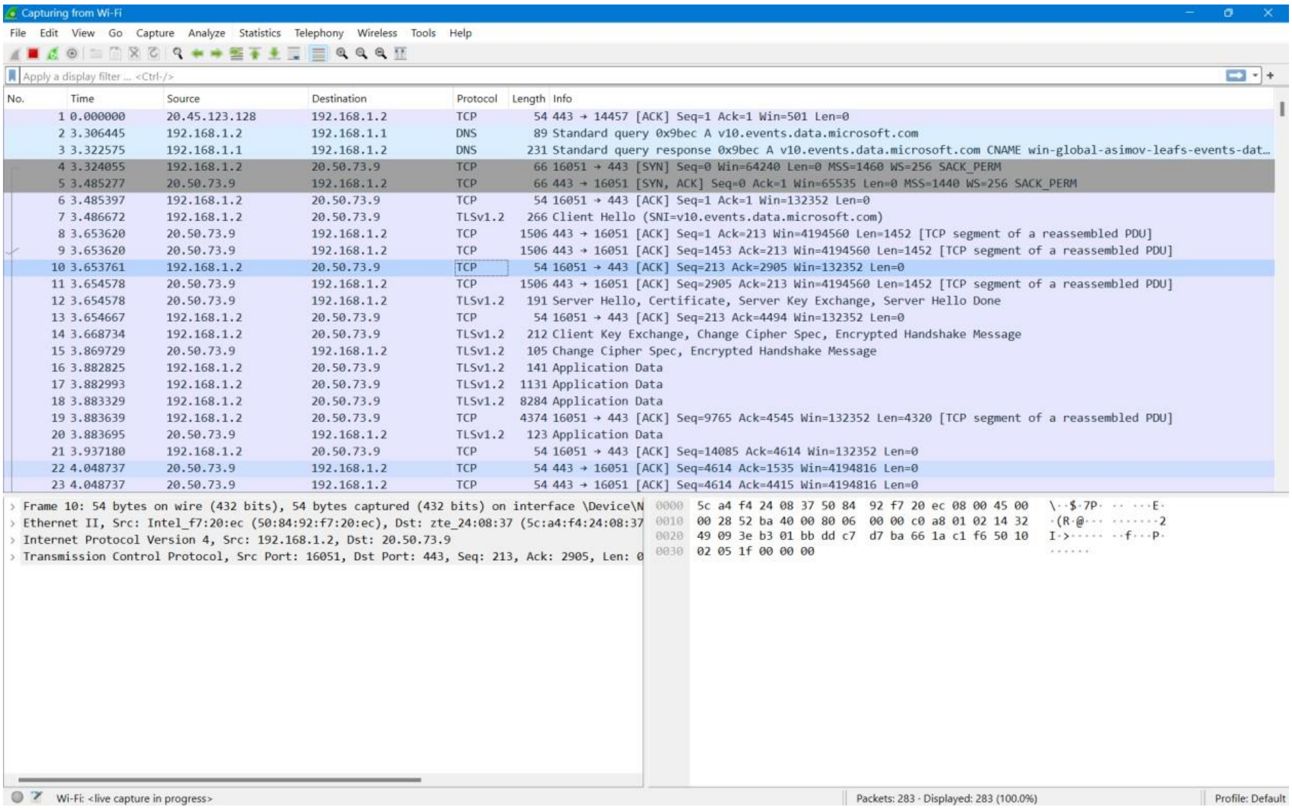
Fourthly, Wireshark proves ineffectual in decrypting encrypted traffic.

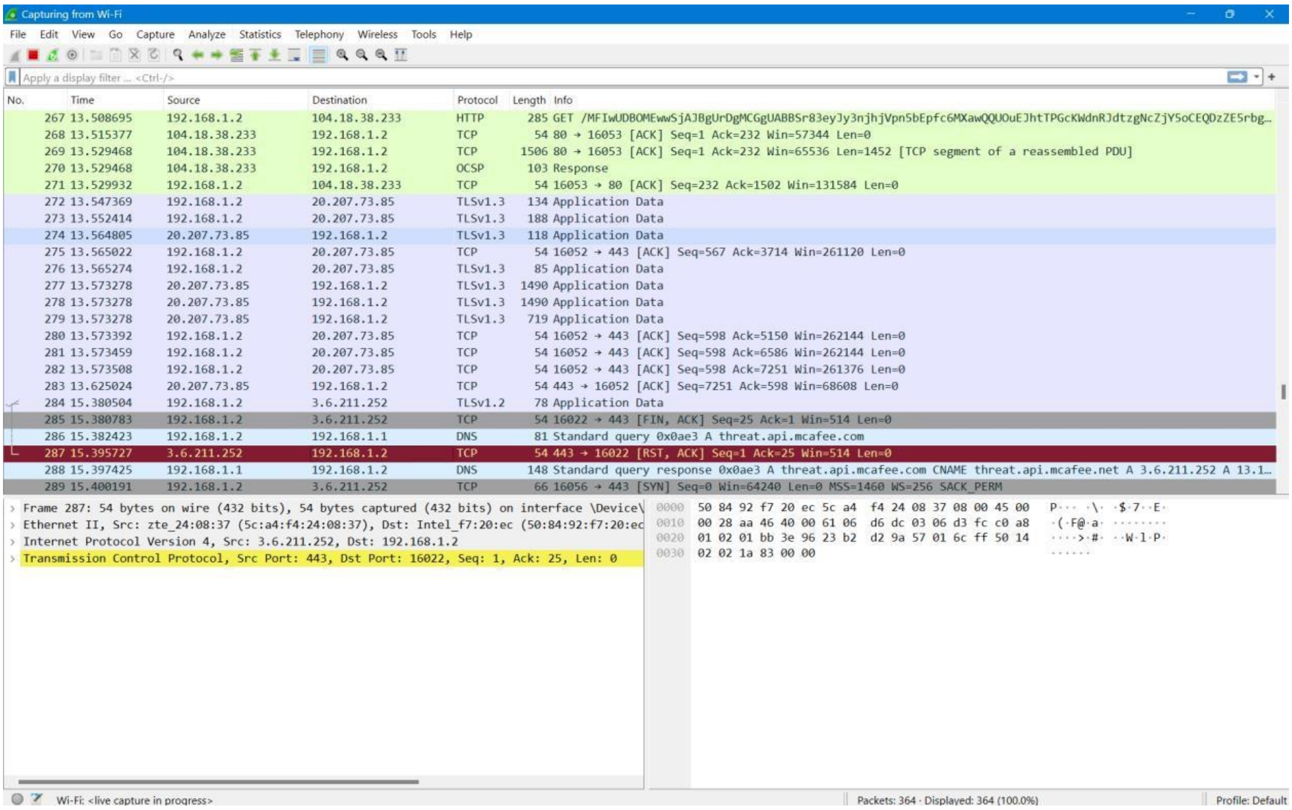
Lastly, Wireshark faces challenges in discerning the authenticity of IPv4 packets, particularly concerning the potential for spoofing. Differentiating between genuine and fabricated IP addresses necessitates a higher level of expertise from IT professionals and potentially supplementary software tools

EXAMPLE:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown
Wi-Fi
Local Area Connection* 10
Local Area Connection* 9
Local Area Connection* 8
Bluetooth Network Connection
Local Area Connection* 2
Local Area Connection* 1
Ethernet 2
Adapter for loopback traffic capture
Ethernet
USBPcap1
USBPcap2
Event Tracing for Windows (ETW) reader
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists · SharkFest · Wireshark Discord · Donate
You are running Wireshark 4.2.4 (v4.2.4-0-g1fe5bce8d665). You receive automatic updates.
Ready to load or capture
No Packets
Profile: Default


Capturing from Ethernet
Packet Information
No: Frame number
Time: Time in second
Source: source address
Destination: Destination address
Protocol: Protocol that is used for communication
Length: Length of packet in bytes
Info: Info of the packet (Type version etc.)
Packet summary
Protocol Window
Offset
Data in Hexadecimal
Data in ASCII
Data Window


Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 20.45.123.128 192.168.1.2 TCP 54 443 → 14457 [ACK] Seq=1 Ack=1 Win=501 Len=0
2 3.306445 192.168.1.2 192.168.1.1 DNS 89 Standard query 0x9bec A v10.events.data.microsoft.com
3 3.322575 192.168.1.1 192.168.1.2 DNS 231 Standard query response 0x9bec A v10.events.data.microsoft.com CNAME win-global-asimov-leafs-events-dat…
4 3.324055 192.168.1.2 20.50.73.9 TCP 66 16051 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5 3.485277 20.50.73.9 192.168.1.2 TCP 66 443 → 16051 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1440 WS=256 SACK_PERM
6 3.485397 192.168.1.2 20.50.73.9 TCP 54 16051 → 443 [ACK] Seq=1 Ack=1 Win=132352 Len=0
7 3.486672 192.168.1.2 20.50.73.9 TLSv1.2 266 Client Hello (SNI=v10.events.data.microsoft.com)
8 3.653620 20.50.73.9 192.168.1.2 TCP 1506 443 → 16051 [ACK] Seq=1 Ack=213 Win=4194560 Len=1452 [TCP segment of a reassembled PDU]
9 3.653620 20.50.73.9 192.168.1.2 TCP 1506 443 → 16051 [ACK] Seq=1453 Ack=213 Win=4194560 Len=1452 [TCP segment of a reassembled PDU]
10 3.653761 192.168.1.2 20.50.73.9 TCP 54 16051 → 443 [ACK] Seq=213 Ack=2905 Win=132352 Len=0
11 3.654578 20.50.73.9 192.168.1.2 TCP 1506 443 → 16051 [ACK] Seq=2905 Ack=213 Win=4194560 Len=1452 [TCP segment of a reassembled PDU]
12 3.654578 20.50.73.9 192.168.1.2 TLSv1.2 191 Server Hello, Certificate, Server Key Exchange, Server Hello Done
13 3.654667 192.168.1.2 20.50.73.9 TCP 54 16051 → 443 [ACK] Seq=213 Ack=4494 Win=132352 Len=0
14 3.668734 192.168.1.2 20.50.73.9 TLSv1.2 212 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
15 3.869729 20.50.73.9 192.168.1.2 TLSv1.2 105 Change Cipher Spec, Encrypted Handshake Message
16 3.882825 192.168.1.2 20.50.73.9 TLSv1.2 141 Application Data
17 3.882993 192.168.1.2 20.50.73.9 TLSv1.2 1131 Application Data
18 3.883329 192.168.1.2 20.50.73.9 TLSv1.2 8284 Application Data
19 3.883639 192.168.1.2 20.50.73.9 TCP 4374 16051 → 443 [ACK] Seq=9765 Ack=4545 Win=132352 Len=4320 [TCP segment of a reassembled PDU]
20 3.883695 20.50.73.9 192.168.1.2 TLSv1.2 123 Application Data
21 3.937180 192.168.1.2 20.50.73.9 TCP 54 16051 → 443 [ACK] Seq=14085 Ack=4614 Win=132352 Len=0
22 4.048737 20.50.73.9 192.168.1.2 TCP 54 443 → 16051 [ACK] Seq=4614 Ack=1535 Win=4194816 Len=0
23 4.048737 20.50.73.9 192.168.1.2 TCP 54 443 → 16051 [ACK] Seq=4614 Ack=4415 Win=4194816 Len=0
Frame 10: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\N
Ethernet II, Src: Intel_f7:20:ec (50:84:92:f7:20:ec), Dst: zte_24:08:37 (5c:a4:f4:24:08:37
Internet Protocol Version 4, Src: 192.168.1.2, Dst: 20.50.73.9
Transmission Control Protocol, Src Port: 16051, Dst Port: 443, Seq: 213, Ack: 2905, Len: 0
0000 5c a4 f4 24 08 37 50 84 92 f7 20 ec 08 00 45 00
0010 00 28 52 ba 40 00 80 06 00 00 c0 a8 01 02 14 32
0020 49 09 3e b3 01 bb dd c7 d7 ba 66 1a c1 f6 50 10
0030 02 05 1f 00 00 00
Wi-Fi: <live capture in progress>
Packets: 283 · Displayed: 283 (100.0%)
Profile: Default


Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
267 13.508695 192.168.1.2 104.18.38.233 HTTP 285 GET /MFIwUDBOMEwwSjAJBgUrDgMCGgUABBSr83eyJy3njhjVpn5bEpfc6MXawQQUOuEJhtTPGcKWdnRJdtzgNcZjY5oCEQDzZE5rbg…
268 13.515377 104.18.38.233 192.168.1.2 TCP 54 80 → 16053 [ACK] Seq=1 Ack=232 Win=57344 Len=0
269 13.529468 104.18.38.233 192.168.1.2 TCP 1506 80 → 16053 [ACK] Seq=1 Ack=232 Win=65536 Len=1452 [TCP segment of a reassembled PDU]
270 13.529468 104.18.38.233 192.168.1.2 OCSP 103 Response
271 13.529932 192.168.1.2 104.18.38.233 TCP 54 16053 → 80 [ACK] Seq=232 Ack=1502 Win=131584 Len=0
272 13.547369 192.168.1.2 20.207.73.85 TLSv1.3 134 Application Data
273 13.552414 192.168.1.2 20.207.73.85 TLSv1.3 188 Application Data
274 13.564805 20.207.73.85 192.168.1.2 TLSv1.3 118 Application Data
275 13.565022 192.168.1.2 20.207.73.85 TCP 54 16052 → 443 [ACK] Seq=567 Ack=3714 Win=261120 Len=0
276 13.565274 192.168.1.2 20.207.73.85 TLSv1.3 85 Application Data
277 13.573278 20.207.73.85 192.168.1.2 TLSv1.3 1490 Application Data
278 13.573278 20.207.73.85 192.168.1.2 TLSv1.3 1490 Application Data
279 13.573278 20.207.73.85 192.168.1.2 TLSv1.3 719 Application Data
280 13.573392 192.168.1.2 20.207.73.85 TCP 54 16052 → 443 [ACK] Seq=598 Ack=5150 Win=262144 Len=0
281 13.573459 192.168.1.2 20.207.73.85 TCP 54 16052 → 443 [ACK] Seq=598 Ack=6586 Win=262144 Len=0
282 13.573508 192.168.1.2 20.207.73.85 TCP 54 16052 → 443 [ACK] Seq=598 Ack=7251 Win=261376 Len=0
283 13.625024 20.207.73.85 192.168.1.2 TCP 54 443 → 16052 [ACK] Seq=7251 Ack=598 Win=68608 Len=0
284 15.380504 192.168.1.2 3.6.211.252 TLSv1.2 78 Application Data
285 15.380783 192.168.1.2 3.6.211.252 TCP 54 16022 → 443 [FIN, ACK] Seq=25 Ack=1 Win=514 Len=0
286 15.382423 192.168.1.2 192.168.1.1 DNS 81 Standard query 0x0ae3 A threat.api.mcafee.com
287 15.395727 3.6.211.252 192.168.1.2 TCP 54 443 → 16022 [RST, ACK] Seq=1 Ack=25 Win=514 Len=0
288 15.397425 192.168.1.1 192.168.1.2 DNS 148 Standard query response 0x0ae3 A threat.api.mcafee.com CNAME threat.api.mcafee.net A 3.6.211.252 A 13.1…
289 15.400191 192.168.1.2 3.6.211.252 TCP 66 16056 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
Frame 287: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\
Ethernet II, Src: zte_24:08:37 (5c:a4:f4:24:08:37), Dst: Intel_f7:20:ec (50:84:92:f7:20:ec
Internet Protocol Version 4, Src: 3.6.211.252, Dst: 192.168.1.2
Transmission Control Protocol, Src Port: 443, Dst Port: 16022, Seq: 1, Ack: 25, Len: 0
0000 50 84 92 f7 20 ec 5c a4 f4 24 08 37 08 00 45 00
0010 00 28 aa 46 40 00 61 06 d6 dc 03 06 d3 fc c0 a8
0020 01 02 01 bb 3e 96 23 b2 d2 9a 57 01 6c ff 50 14
0030 02 02 1a 83 00 00
Wi-Fi: <live capture in progress>
Packets: 364 · Displayed: 364 (100.0%)
Profile: Default