NITTE EDUCATION TRUST

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

An Autonomous Institution Approved by UGC/AICTE/Govt. of Karnataka
Accredited by NBA (Tier – I) and NAAC 'A+' Grade
Affiliated to Visveswaraya Technological University, Belagavi
Post Box No. 6429, Yelahanka, Bengaluru – 560 064, Karnataka, India

KNOWLEDGE • CHARACTER • UNITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Cryptography and Network Security Laboratory – 21CSL66

PART-A

1. Write a C program that contains a string (char pointer) with a value 'Hello world'. The program should AND, OR and XOR each character in this string with 127 and display the result.
2. Write a Java program to perform encryption and decryption using the following algorithms:
   a. Ceasar cipher b. Play fair cipher
3. Write a C program to implement the following:
   a. Vigenere Cipher using a Vigenere table.
   b. Rail fence Cipher using row and column transformation.
4. Write a C program to implement encryption and decryption using Hill Cipher method.
5. Write a C program to implement DES algorithm.
6. Write a Java program to implement AES algorithm.
7. Write a Java program to implement RSA algorithm logic.
8. Implement Diffie-Hellman key exchange mechanism using HTML and Java script.
9. Calculate the message digest of a text using the SHA-512 algorithm in Java.
10. Calculate the message digest of a text using the MDS algorithm using Java.

PART-B (7 hrs)

1. Design a mobile application for end-to-end encryption of short message service (SMS) using any symmetric/asymmetric cryptographic technique that can conceal message regarding student's results/notification on placements/Department's updates of Nitte Meenakshi Institute of Technology (NMIT) while on transit to another mobile device.

2. **The objectives to implement are as follows:**

Develop an android application for the NMIT that will ensure the encryption of every message transmitted within the network of the organization using any symmetric/asymmetric cryptographic technique. This application will provide security measures whenever information is transmitted from one mobile device to another as it is important to protect the information while it is on transit. The choice of Cryptographic technique depends on your project's needs. The below is the list of some cryptographic techniques popular for Android which can be used in your project for the implementation purpose.

- symmetric encryption
- asymmetric encryption
- hashing
- digital signature
- end-to-end encryption
- HMAC

Expected outputs:

Note: The project should include the key objectives. However, scope is not limited to mentioned objectives.

Program Statement 1: Write a C program that contains a string (char pointer) with a value 'Hello world'. The program should AND, OR and XOR each character in this string with 127 and display the result.

```
#include <stdio.h>
int main() {
   // Initialize the string
   char str[] = "Hello world";
   printf("Original string: %s\n\n", str);

   printf("Performing bitwise AND with 127:\n");
   for (int i = 0; str[i] != '\0'; i++) {
      str[i] = str[i] & 127;
      printf("%c", str[i]);
   }
   printf("\n\n");

   printf("Performing bitwise OR with 127:\n");
   for (int i = 0; str[i] != '\0'; i++) {
      str[i] = str[i] | 127;
      printf("%c", str[i]);
   }
   printf("\n\n");

   printf("Performing bitwise XOR with 127:\n");
   for (int i = 0; str[i] != '\0'; i++) {
      str[i] = str[i] ^ 127;
      printf("%c", str[i]);
   }
   printf("\n");
   return 0;
```

```
}
```

Execution Steps:

```
gedit program1.c
gcc program1.c
./a.out
```

Output:

```
Original String: Hello world
Performing Bitwise AND with 127:
Hello world
Performing Bitwise OR with 127:
------
Performing Bitwise XOR with 127:
------
```

Program Statement 2: Write a Java program to perform encryption and decryption using the following algorithms:
a. Ceasar cipher b. Playfair cipher

a) Ceaser Cipher

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

public class CeaserCipher {
   static Scanner sc = new Scanner(System.in);
   static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

   public static void main(String[] args) throws IOException {
      // TODO code application logic here
      System.out.print("Enter any String: ");
      String str = br.readLine();
```

```
    System.out.print("\nEnter the Key: ");
    int key = sc.nextInt();
    String encrypted = encrypt(str, key);
    System.out.println("\nEncrypted String is: " + encrypted);
    String decrypted = decrypt(encrypted, key);
    System.out.println("\nDecrypted String is: " + decrypted);
    System.out.println("\n");
}

public static String encrypt(String str, int key) {
    String encrypted = "";
    for (int i = 0; i < str.length(); i++) {
        int c = str.charAt(i);
        if (Character.isUpperCase(c)) {
            c = c + (key % 26);
            if (c > 'Z')
                c = c - 26;
        } else if (Character.isLowerCase(c)) {
            c = c + (key % 26);
            if (c > 'z')
                c = c - 26;
        }
        encrypted += (char) c;
    }
    return encrypted;
}

public static String decrypt(String str, int key) {
    String decrypted = "";
    for (int i = 0; i < str.length(); i++) {
```

```
            int c = str.charAt(i);
            if (Character.isUpperCase(c)) {
               c = c - (key % 26);
               if (c < 'A')
                  c = c + 26;
            } else if (Character.isLowerCase(c)) {
               c = c - (key % 26);
               if (c < 'a')
                  c = c + 26;
            }
            decrypted += (char) c;
         }
         return decrypted;
      }
}
```

Execution Steps:

gedit CaesarCipher.java
javac CaesarCipher.java
java CaesarCipher.java

Output:

Enter any String: Computer science

Enter the key: 3

Encrypted String is: frpsxwhu vflhqfh

Decrypted String is: computer science

b) Play fair Cipher

```
import java.util.Scanner;
```

```
public class PlayfairCipher {
    private char[][] matrix;
    private String key;
    private String alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

    public PlayfairCipher(String key) {
        this.key = key.toUpperCase().replaceAll("[^A-Z]", "");
        this.matrix = generateMatrix();
    }

    private char[][] generateMatrix() {
        char[][] matrix = new char[5][5];
        String keyStr = this.key + this.alphabet;
        int row = 0;
        int col = 0;

        for (int i = 0; i < keyStr.length(); i++) {
            char c = keyStr.charAt(i);
            if (col == 5) {
                col = 0;
                row++;
            }
            if (row == 5) break;
            if (!contains(matrix, c)) {
                matrix[row][col] = c;
                col++;
            }
        }
```

```
    return matrix;
}

private boolean contains(char[][] matrix, char c) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (matrix[i][j] == c) return true;
        }
    }
    return false;
}

public String encrypt(String plaintext) {
    plaintext = plaintext.toUpperCase().replaceAll("[^A-Z]", "");
    plaintext = plaintext.replaceAll("J", "I");
    StringBuilder ciphertext = new StringBuilder();

    for (int i = 0; i < plaintext.length(); i += 2) {
        char a = plaintext.charAt(i);
        char b = (i + 1 < plaintext.length()) ? plaintext.charAt(i + 1) : 'X';

        int[] posA = findPosition(a);
        int[] posB = findPosition(b);

        char encryptedA, encryptedB;
        if (posA[0] == posB[0]) {
            encryptedA = matrix[posA[0]][(posA[1] + 1) % 5];
            encryptedB = matrix[posB[0]][(posB[1] + 1) % 5];
        } else if (posA[1] == posB[1]) {
            encryptedA = matrix[(posA[0] + 1) % 5][posA[1]];
```

```
            encryptedB = matrix[(posB[0] + 1) % 5][posB[1]];
        } else {
            encryptedA = matrix[posA[0]][posB[1]];
            encryptedB = matrix[posB[0]][posA[1]];
        }

        ciphertext.append(encryptedA).append(encryptedB);
    }

    return ciphertext.toString();
}

public String decrypt(String ciphertext) {
    StringBuilder plaintext = new StringBuilder();

    for (int i = 0; i < ciphertext.length(); i += 2) {
        char a = ciphertext.charAt(i);
        char b = ciphertext.charAt(i + 1);

        int[] posA = findPosition(a);
        int[] posB = findPosition(b);

        char decryptedA, decryptedB;
        if (posA[0] == posB[0]) {
            decryptedA = matrix[posA[0]][(posA[1] + 4) % 5];
            decryptedB = matrix[posB[0]][(posB[1] + 4) % 5];
        } else if (posA[1] == posB[1]) {
            decryptedA = matrix[(posA[0] + 4) % 5][posA[1]];
            decryptedB = matrix[(posB[0] + 4) % 5][posB[1]];
        } else {
```

```
            decryptedA = matrix[posA[0]][posB[1]];
            decryptedB = matrix[posB[0]][posA[1]];
        }

        plaintext.append(decryptedA).append(decryptedB);
    }

    return plaintext.toString();
}

private int[] findPosition(char c) {
    int[] pos = new int[2];
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (matrix[i][j] == c) {
                pos[0] = i;
                pos[1] = j;
                return pos;
            }
        }
    }
    return pos;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the key:");
    String key = scanner.nextLine();
    System.out.println("Enter the plaintext:");
    String plaintext = scanner.nextLine();
```

```
        PlayfairCipher cipher = new PlayfairCipher(key);
        String ciphertext = cipher.encrypt(plaintext);
        System.out.println("Encrypted text: " + ciphertext);

        String decryptedText = cipher.decrypt(ciphertext);
        System.out.println("Decrypted text: " + decryptedText);

        scanner.close();
    }
}
```

Program 3: Write a C program to implement the following:

a. Vigenere Cipher using a Vigenere table.

b. Rail fence Cipher using row and column transformation.

a. Vigenere Cipher using a Vigenere table.

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

#define MAX_TEXT_LENGTH 1000
#define MAX_KEY_LENGTH 100

char vigenereTable[26][26];

void generateVigenereTable() {
    for (int i = 0; i < 26; i++) {
```

```
        for (int j = 0; j < 26; j++) {
            vigenereTable[i][j] = 'A' + (i + j) % 26;
        }
    }
}

void encryptVigenere(char *plaintext, char *key) {
    int plaintextLength = strlen(plaintext);
    int keyLength = strlen(key);
    int keyIndex = 0;

    for (int i = 0; i < plaintextLength; i++) {
        if (isalpha(plaintext[i])) {
            plaintext[i] = toupper(plaintext[i]);
            plaintext[i] = vigenereTable[key[keyIndex] - 'A'][plaintext[i] - 'A'];
            keyIndex = (keyIndex + 1) % keyLength;
        }
    }
}

void decryptVigenere(char *ciphertext, char *key) {
    int ciphertextLength = strlen(ciphertext);
    int keyLength = strlen(key);
    int keyIndex = 0;

    for (int i = 0; i < ciphertextLength; i++) {
        if (isalpha(ciphertext[i])) {
            ciphertext[i] = toupper(ciphertext[i]);
            for (int j = 0; j < 26; j++) {
                if (vigenereTable[key[keyIndex] - 'A'][j] == ciphertext[i]) {
```

```
                ciphertext[i] = 'A' + j;
                break;
            }
        }
        keyIndex = (keyIndex + 1) % keyLength;
    }
  }
}

int main() {
  generateVigenereTable();

  char plaintext[MAX_TEXT_LENGTH];
  char key[MAX_KEY_LENGTH];

  printf("Enter plaintext: ");
  fgets(plaintext, sizeof(plaintext), stdin);
  plaintext[strcspn(plaintext, "\n")] = '\0'; // Remove newline character

  printf("Enter key: ");
  fgets(key, sizeof(key), stdin);
  key[strcspn(key, "\n")] = '\0'; // Remove newline character

  encryptVigenere(plaintext, key);
  printf("Encrypted text: %s\n", plaintext);

  decryptVigenere(plaintext, key);
  printf("Decrypted text: %s\n", plaintext);

  return 0;
```

```
}
```

Output:

Enter plaintext:

Enter key:

Encrypted text:

Decrypted text:

b. Rail fence Cipher using row and column transformation.

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

#define MAX_TEXT_LENGTH 1000

void encryptRailFence(char *plaintext, int key) {
   int plaintextLength = strlen(plaintext);
   char rail[key][plaintextLength];

   for (int i = 0; i < key; i++) {
      for (int j = 0; j < plaintextLength; j++) {
         rail[i][j] = '\n';
      }
   }
```

```
    int row = 0;
    int direction = 0;

    for (int i = 0; i < plaintextLength; i++) {
        rail[row][i] = plaintext[i];
        if (row == 0 || row == key - 1) {
            direction = !direction;
        }
        row += direction ? 1 : -1;
    }

    printf("Encrypted text: ");
    for (int i = 0; i < key; i++) {
        for (int j = 0; j < plaintextLength; j++) {
            if (rail[i][j] != '\n') {
                printf("%c", rail[i][j]);
            }
        }
    }
    printf("\n");
}

void decryptRailFence(char *ciphertext, int key) {
    int ciphertextLength = strlen(ciphertext);
    char rail[key][ciphertextLength];

    for (int i = 0; i < key; i++) {
        for (int j = 0; j < ciphertextLength; j++) {
            rail[i][j] = '\n';
        }
```

```
}

int row = 0;
int direction = 0;

for (int i = 0; i < ciphertextLength; i++) {
   rail[row][i] = '*';
   if (row == 0 || row == key - 1) {
      direction = !direction;
   }
   row += direction ? 1 : -1;
}

int index = 0;
for (int i = 0; i < key; i++) {
   for (int j = 0; j < ciphertextLength; j++) {
      if (rail[i][j] == '*' && index < ciphertextLength) {
         rail[i][j] = ciphertext[index++];
      }
   }
}

printf("Decrypted text: ");
row = 0;
direction = 0;
for (int i = 0; i < ciphertextLength; i++) {
   printf("%c", rail[row][i]);
   if (row == 0 || row == key - 1) {
      direction = !direction;
   }
```

```
        row += direction ? 1 : -1;
    }
    printf("\n");
}

int main() {
    char plaintext[MAX_TEXT_LENGTH];
    int key;

    printf("Enter plaintext: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    plaintext[strcspn(plaintext, "\n")] = '\0'; // Remove newline character

    printf("Enter key: ");
    scanf("%d", &key);

    encryptRailFence(plaintext, key);
    decryptRailFence(plaintext, key);

    return 0;
}
```

Output:

Enter plaintext:

Enter key:

Encrypted text:

Decrypted text:

Program Statement 4: Write a C program to implement encryption and decryption using Hill Cipher method.

```
#include<stdio.h>
#include<string.h>
int main() {
   unsigned int a[3][3] = { { 6, 24, 1 }, { 13, 16, 10 }, { 20, 17, 15 } };
   unsigned int b[3][3] = { { 8, 5, 10 }, { 21, 8, 21 }, { 21, 12, 8 } };
   int i, j;
   unsigned int c[20], d[20];
   char msg[20];
   int determinant = 0, t = 0;
   ;
   printf("Enter plain text\n ");
   scanf("%s", msg);
   for (i = 0; i < 3; i++) {
      c[i] = msg[i] - 65;
      printf("%d ", c[i]);
   }
   for (i = 0; i < 3; i++) {
      t = 0;
      for (j = 0; j < 3; j++) {
         t = t + (a[i][j] * c[j]);
      }
      d[i] = t % 26;
   }
   printf("\nEncrypted Cipher Text :");
   for (i = 0; i < 3; i++)
      printf(" %c", d[i] + 65);
   for (i = 0; i < 3; i++) {
      t = 0;
      for (j = 0; j < 3; j++) {
         t = t + (b[i][j] * d[j]);
      }
```

```
    c[i] = t % 26;
  }
  printf("\nDecrypted Cipher Text :");
  for (i = 0; i < 3; i++)
    printf(" %c", c[i] + 65);
  return 0;
}
```

Output:

```
$ gcc HillCipher.c
$ ./a.out

Enter plain text: SAN
 18 0 13
Encrypted Cipher Text : R A J
Decrypted Cipher Text : S A N
```

Data Encryption Standard (DES)

The Data Encryption Standard (DES) is ***a symmetric-key block cipher*** published by the National Institute of Standards and Technology (NIST). It has a 56-bit key length which has played a significant role in data security**.** Data encryption standard (DES) has been found vulnerable to very powerful attacks therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of **64 bits** each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is **56 bits**.

The basic idea is shown below:

We have mentioned that DES uses a 56-bit key. Actually, The initial key consists of 64 bits. However, before the DES process even starts, every 8th bit of the key is discarded to produce a 56-bit key. That is bit positions 8, 16, 24, 32, 40, 48, 56, and 64 are discarded.

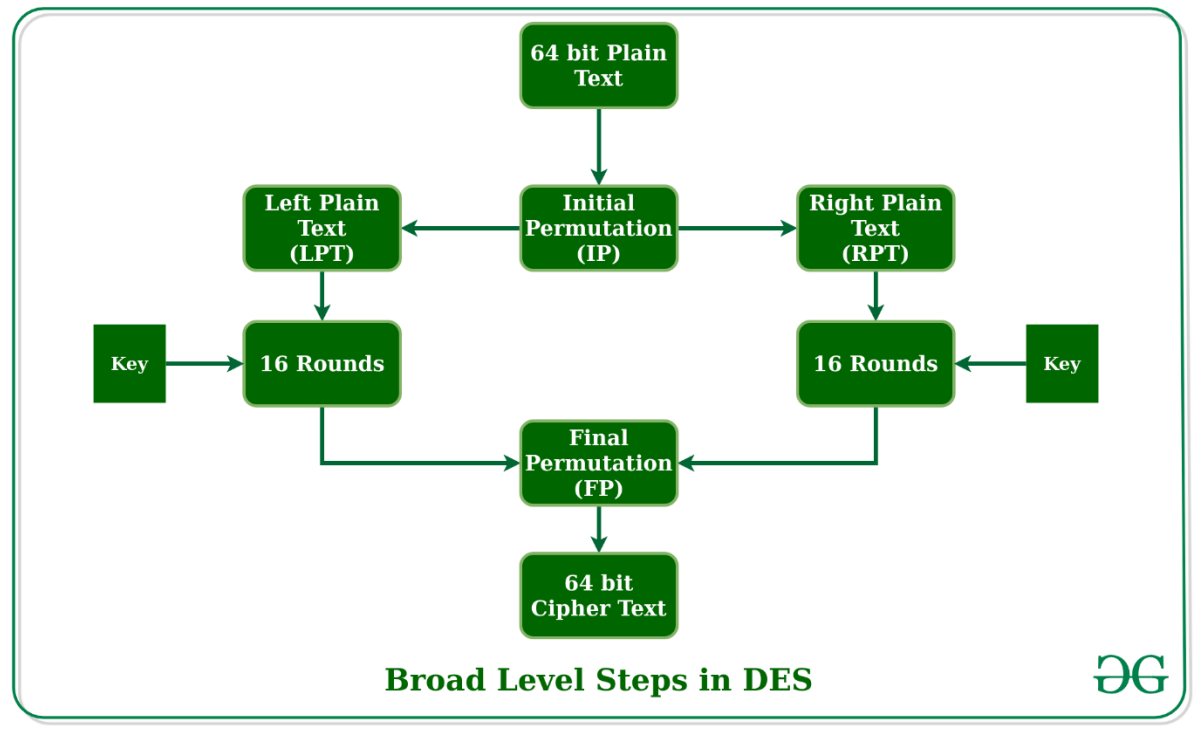
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Figure - discording of every 8th bit of original key

Thus, the discarding of every 8th bit of the key produces a **56-bit key** from the original **64-bit key**.

DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

- In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.
- The initial permutation is performed on plain text.
- Next, the initial permutation (IP) produces two halves of the permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT go through 16 rounds of the encryption process.
- In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
- The result of this process produces 64-bit ciphertext.



Initial Permutation (IP)

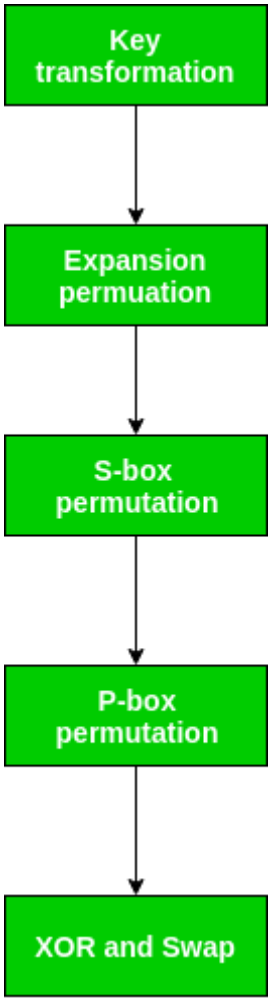
As we have noted, the initial permutation (IP) happens only once and it happens before the first round. It suggests how the transposition in IP should proceed, as shown in the figure. For example, it says that the IP replaces the first bit of the original plain text block with the 58th bit of the original plain text, the second bit with the 50th bit of the original plain text block, and so on.

This is nothing but jugglery of bit positions of the original plain text block. the same rule applies to all the other bit positions shown in the figure.

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	33	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Figure - Initial permutation table

As we have noted after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half-block consists of 32 bits, and each of the 16 rounds, in turn, consists of the broad-level steps outlined in the figure.



tep 1: Key transformation

We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

For example: if the round numbers 1, 2, 9, or 16 the shift is done by only one position for other rounds, the circular shift is done by two positions. The number of key bits shifted per round is shown in the figure.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure - number of key bits shifted per round

After an appropriate shift, 48 of the 56 bits are selected. From the 48 we might obtain 64 or 56 bits based on requirement which helps us to recognize that this model is very versatile and can

handle any range of requirements needed or provided. for selecting 48 of the 56 bits the table is shown in the figure given below. For instance, after the shift, bit number 14 moves to the first position, bit number 17 moves to the second position, and so on. If we observe the table , we will realize that it contains only 48-bit positions. Bit number 18 is discarded (we will not find it in the table), like 7 others, to reduce a 56-bit key to a 48-bit key. Since the key transformation process involves permutation as well as a selection of a 48-bit subset of the original 56-bit key it is called Compression Permutation.

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Figure - compression permutation

Because of this compression permutation technique, a different subset of key bits is used in each round. That makes DES not easy to crack.

Step 2: Expansion Permutation

Recall that after the initial permutation, we had two 32-bit plain text areas called Left Plain Text(LPT) and Right Plain Text(RPT). During the expansion permutation, the RPT is expanded from 32 bits to 48 bits. Bits are permuted as well hence called expansion permutation. This happens as the 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits. Then, each 4-bit block of the previous step is then expanded to a corresponding 6-bit block, i.e., per 4-bit block, 2 more bits are added.

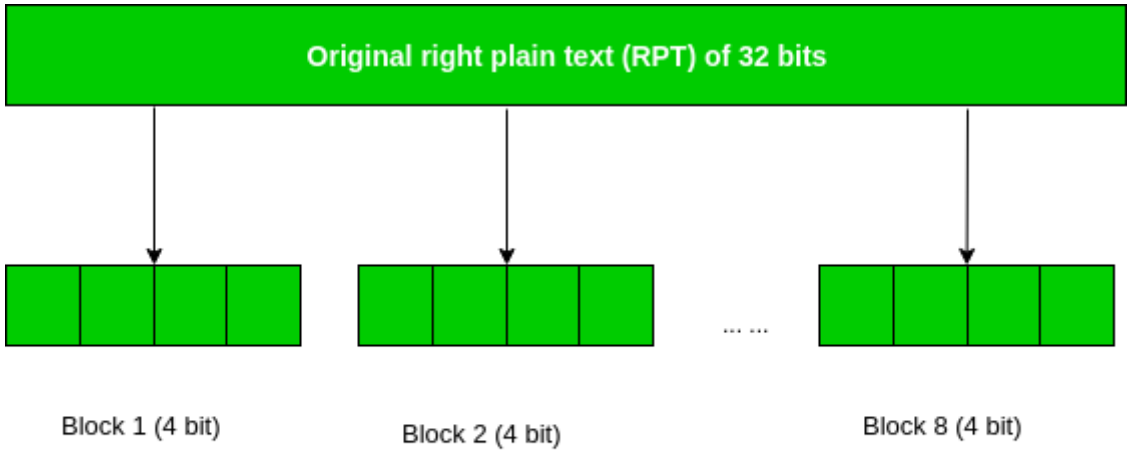


Figure - division of 32 bit RPT into 8 bit blocks

This process results in expansion as well as a permutation of the input bit while creating output. The key transformation process compresses the 56-bit key to 48 bits. Then the expansion permutation process expands the **32-bit RPT** to **48-bits**. Now the 48-bit key is XOR with 48-bit RPT and the resulting output is given to the next step, which is the **S-Box substitution**.

Program 5: Write a C program to implement DES algorithm.

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

int s_matrix0[4][4] = { {1,0,3,2},
                        {3,2,1,0},
                        {0,2,1,3},
                        {3,1,3,2} };

int s_matrix1[4][4] = { {0,1,2,3},
                        {2,0,1,3},
                        {3,0,1,0},
                        {2,1,0,3} };

int s0=0, s1=0;
int row=0, col=0;
int s0_binary[2], s1_binary[2];
int result[2];

int to_digit(int a, int b)
{
    int output;
    if(a==1 && b==1)
        output = 3;

    if(a==0 && b==1)
        output = 1;

    if(a==1 && b==0)
        output = 2;
```

```
    if(a==0 && b==0)
        output = 0;

    return output;
}

void to_binary(int num)
{
    int x;

    if(num == 3)
    {
        for(x=0; x<2; x++)
            result[x] = 1;
    }
    else if(num == 1)
    {
        result[0] = 0;
        result[1] = 1;
    }
    else if(num == 2)
    {
        result[0] = 1;
        result[1] = 0;
    }
    else if(num == 0)
    {
        for(x=0; x<2; x++)
            result[x] = 0;
    }
}

void main()
{
    int i,j,index;
```

```
    int k1[8], k2[8];
    int afterp10[11], ls1[10], ls2[10], afterip[8], afterep[8],
afterp4[4];
    int aftersboxesone[4], aftersboxestwo[4];
    int leftafterip[4], rightafterip[4];
    int leftafterep[4], rightafterep[4];
    int leftafterone[4], rightafterone[4];
    int afteripinverse[8];
    int afterone[8], aftertwo[8];

    int p10[10] = {3,5,2,7,4,10,1,9,8,6};
    int p8[8] = {6,3,7,4,8,5,10,9};

    int ip[8] = {2,6,3,1,4,8,5,7};
    int ep[8] = {4,1,2,3,2,3,4,1};
    int p4[4] = {2,4,3,1};
    int ipinverse[8] = {4,1,3,5,7,2,8,6};

    int key[11] = {1,0,1,0,0,0,0,0,1,0};
    int plain[8] = {0,1,1,0,1,1,0,1};

    printf("S-DES Key Generation and Encryption.\n");

    printf("\n----------KEY GENERATION----------\n");
    printf("\nEntered the 10-bit key is: ");
    for(i=0; i<10; i++)
        printf("%d ",key[i]);

    printf("\np10 permutation is defined as: ");
    for(i=0; i<=9; i++)
        printf("%d ",p10[i]);

    for(i=0; i<=9; i++)
    {
        index = p10[i];
        afterp10[i] = key[index - 1];
    }
```

```
afterp10[i] = '\0';

printf("\n\nAfter p10 = ");
for(i=0; i<=9; i++)
    printf("%d ",afterp10[i]);

for(i=0; i<5; i++)
{
    if(i == 4)
        ls1[i]=afterp10[0];
    else
        ls1[i]=afterp10[i+1];
}

for(i=5; i<10; i++)
{
    if(i == 9)
        ls1[i]=afterp10[5];
    else
        ls1[i]=afterp10[i+1];
}

printf("\n\nAfter LS-1 = ");
for(i=0; i<10; i++)
    printf("%d ", ls1[i]);

printf("\np8 permutation is defined as: ");
for(i=0;i<8;i++)
    printf("%d ",p8[i]);

index=0;
for(i=0; i<=9; i++)
{
    index = p8[i];
    k1[i] = ls1[index - 1];
}
```

```
printf("\n\n---->Key-1 is: ");
for(i=0;i<8;i++)
    printf("%d ",k1[i]);

for(i=0; i<3; i++)
{
    ls2[i]=ls1[i+2];
}
ls2[3]=ls1[0];
ls2[4]=ls1[1];

for(i=5; i<8; i++)
{
    ls2[i]=ls1[i+2];
}
ls2[8]=ls1[5];
ls2[9]=ls1[6];

printf("\n\nAfter LS-2 = ");
for(i=0; i<10; i++)
    printf("%d ", ls2[i]);

printf("\np8 permutation is defined as: ");
for(i=0;i<8;i++)
    printf("%d ",p8[i]);

index=0;
for(i=0; i<=9; i++)
{
    index = p8[i];
    k2[i] = ls2[index - 1];
}

printf("\n\n---->Key-2 is: ");
for(i=0;i<8;i++)
    printf("%d ",k2[i]);
```

```
printf("\n\n----------S-DES ENCRYPTION---------\n");
printf("\n---->Entered the 8-bit plaintext is: ");
for(i=0; i<8; i++)
    printf("%d ",plain[i]);


printf("\nInitial permutation is defined as: ");
for(i=0; i<8; i++)
    printf("%d ",ip[i]);

for(i=0; i<8; i++)
{
    index = ip[i];
    afterip[i] = plain[index - 1];
}
afterip[i] = '\0';

printf("\nAfter ip = ");
for(i=0; i<8; i++)
    printf("%d ", afterip[i]);

printf("\n\nExpand permutation is defined as: ");
for(i=0; i<8; i++)
    printf("%d ",ep[i]);

for(j=0; j<4; j++)
    leftafterip[j] = afterip[j];

for(j=0; j<4; j++)
    rightafterip[j] = afterip[j+4];

for(i=0; i<4; i++)
{
    index = ep[i];
    afterep[i] = rightafterip[index - 1];
}
```

```
for(i=4; i<8; i++)
{
    index = ep[i];
    afterep[i] = rightafterip[index - 1];
}
afterep[i] = '\0';

printf("\nAfter ep = ");
for(i=0; i<8; i++)
    printf("%d ", afterep[i]);

for(i=0; i<8; i++)
    k1[i] = k1[i] ^ afterep[i];

printf("\n\nAfter XOR operation with 1st Key= ");
for(i=0; i<8; i++)
    printf("%d ", k1[i]);

row = to_digit(k1[0],k1[3]);
col = to_digit(k1[1],k1[2]);
s0 = s_matrix0[row][col];

to_binary(s0);
for(j=0; j<2; j++)
    s0_binary[j] = result[j];

row = to_digit(k1[4],k1[7]);
col = to_digit(k1[5],k1[6]);

s1 = s_matrix1[row][col];

to_binary(s1);
for(j=0; j<2; j++)
    s1_binary[j] = result[j];

for(j=0; j<2; j++)
    aftersboxesone[j] = s0_binary[j];
```

```
    for(i=0,j=2; i<2,j<4; i++,j++)
        aftersboxesone[j] = s1_binary[i];

    printf("\n\nAfter first S-Boxes= ");
    for(i=0; i<4; i++)
        printf("%d ", aftersboxesone[i]);

    printf("\n\nP4 is defined as: ");
    for(i=0; i<4; i++)
        printf("%d ",p4[i]);

    for(i=0; i<4; i++)
    {
        index = p4[i];
        afterp4[i] = aftersboxesone[index - 1];
    }
    afterp4[i] = '\0';

    printf("\nAfter P4= ");
    for(i=0; i<4; i++)
        printf("%d ", afterp4[i]);

    for(i=0; i<4; i++)
        afterp4[i] = afterp4[i] ^ leftafterip[i];

    printf("\n\nAfter XOR operation with left nibble of after ip=
");
    for(i=0; i<4; i++)
        printf("%d ", afterp4[i]);

    for(i=0; i<4; i++)
        afterone[i] = rightafterip[i];

    for(i=0,j=4; i<4,j<8; i++,j++)
        afterone[j] = afterp4[i];
```

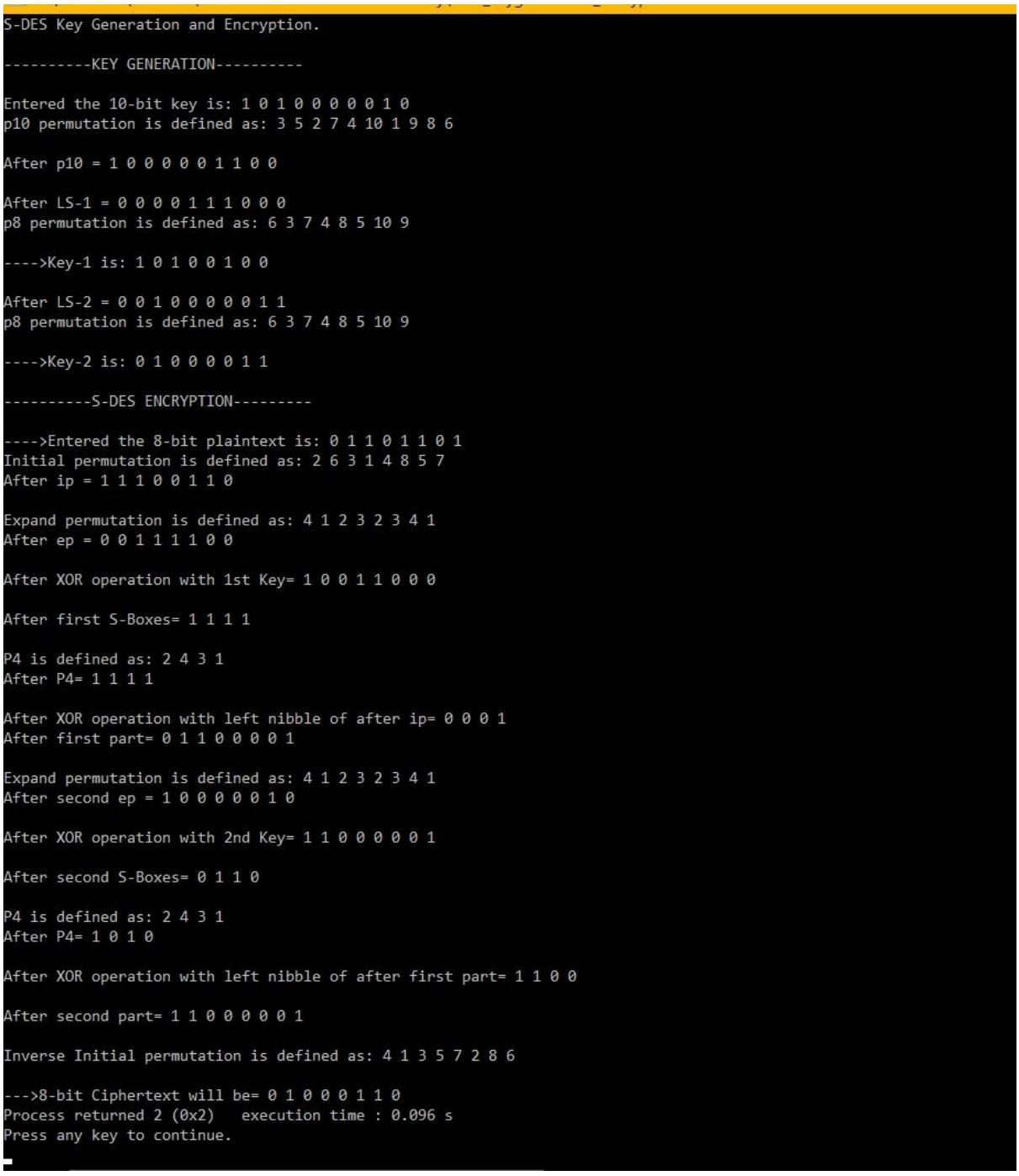
```
afterone[j] = '\0';

printf("\nAfter first part= ");
for(i=0; i<8; i++)
    printf("%d ", afterone[i]);

for(j=0; j<4; j++)
    leftafterone[j] = afterone[j];

for(j=0; j<4; j++)
    rightafterone[j] = afterone[j+4];

printf("\n\nExpand permutation is defined as: ");
for(i=0; i<8; i++)
    printf("%d ",ep[i]);

for(i=0; i<4; i++)
{
    index = ep[i];
    afterep[i] = rightafterone[index - 1];
}

for(i=4; i<8; i++)
{
    index = ep[i];
    afterep[i] = rightafterone[index - 1];
}
afterep[i] = '\0';

printf("\nAfter second ep = ");
for(i=0; i<8; i++)
    printf("%d ", afterep[i]);

for(i=0; i<8; i++)
    k2[i] = k2[i] ^ afterep[i];

printf("\n\nAfter XOR operation with 2nd Key= ");
```

```
for(i=0; i<8; i++)
    printf("%d ", k2[i]);

row = to_digit(k2[0],k2[3]);
col = to_digit(k2[1],k2[2]);

s0 = s_matrix0[row][col];
to_binary(s0);
for(j=0; j<2; j++)
    s0_binary[j] = result[j];

row = to_digit(k2[4],k2[7]);
col = to_digit(k2[5],k2[6]);

s1 = s_matrix1[row][col];
to_binary(s1);
for(j=0; j<2; j++)
    s1_binary[j] = result[j];

for(j=0; j<2; j++)
    aftersboxestwo[j] = s0_binary[j];

for(i=0,j=2; i<2,j<4; i++,j++)
    aftersboxestwo[j] = s1_binary[i];

printf("\n\nAfter second S-Boxes= ");
for(i=0; i<4; i++)
    printf("%d ", aftersboxestwo[i]);

printf("\n\nP4 is defined as: ");
for(i=0; i<4; i++)
    printf("%d ",p4[i]);

for(i=0; i<4; i++)
{
    index = p4[i];
    afterp4[i] = aftersboxestwo[index - 1];
```

```
    }
    afterp4[i] = '\0';

    printf("\nAfter P4= ");
    for(i=0; i<4; i++)
        printf("%d ", afterp4[i]);

    for(i=0; i<4; i++)
        afterp4[i] = afterp4[i] ^ leftafterone[i];

    printf("\n\nAfter XOR operation with left nibble of after first
part= ");
    for(i=0; i<4; i++)
        printf("%d ", afterp4[i]);

    for(i=0; i<4; i++)
        aftertwo[i] = afterp4[i];

    for(i=0,j=4; i<4,j<8; i++,j++)
        aftertwo[j] = rightafterone[i];

    aftertwo[j] = '\0';

    printf("\n\nAfter second part= ");
    for(i=0; i<8; i++)
        printf("%d ", aftertwo[i]);


    printf("\n\nInverse Initial permutation is defined as: ");
    for(i=0; i<8; i++)
        printf("%d ",ipinverse[i]);

    for(i=0; i<8; i++)
    {
        index = ipinverse[i];
        afteripinverse[i] = aftertwo[index - 1];
    }
```

```
    afteripinverse[j] = '\0';

    printf("\n\n--->8-bit Ciphertext will be= ");
    for(i=0; i<8; i++)
        printf("%d ", afteripinverse[i]);
}
```

Output:



Program -6: Write a Java program to implement AES algorithm.

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class AESExample {

    // Method to generate a secret key
    public static SecretKey generateKey(int n) throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(n);
        SecretKey secretKey = keyGenerator.generateKey();
        return secretKey;
    }

    // Method to encrypt a plaintext using a secret key
    public static String encrypt(String plainText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    // Method to decrypt a ciphertext using a secret key
    public static String decrypt(String cipherText, SecretKey secretKey) throws Exception
{
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(cipherText));
        return new String(decryptedBytes);
    }
```

```
    // Main method to demonstrate encryption and decryption
    public static void main(String[] args) {
        try {
            String plainText = "Hello, this is a secret message!";
            SecretKey secretKey = generateKey(128);

            String encryptedText = encrypt(plainText, secretKey);
            System.out.println("Encrypted Text: " + encryptedText);

            String decryptedText = decrypt(encryptedText, secretKey);
            System.out.println("Decrypted Text: " + decryptedText);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

Encrypted Text: BnZW2bnsIvqHzqXUO4UsZrXh2mLEwgjPQd3t9IbNQ+A=

Decrypted Text: Hello, this is a secret message!

Program -7: Write a Java program to implement RSA algorithm logic.

```
import java.io.BufferedReader;
importjava.io.InputStreamReader;
importjava.math.*;
import java.util.Random;
import java.util.Scanner;
```

```
public class RSA {
static Scanner sc = new Scanner(System.in);
public static void main(String[] args) {
// TODO code application logic here
System.out.print("Enter a Prime number:");
BigInteger p = sc.nextBigInteger(); // Here's one prime number.
System.out.print("Enter another prime number:");
BigInteger q=sc.nextBigInteger(); // another prime number.
BigInteger n=p.multiply(q);
BigInteger n2=p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
BigInteger e=generateE( n2);
BigInteger d=e.modInverse(n2);//Here's the multiplicative inverse
System.out.println("Encryptionkeys are: "+ e + ", "+ n);
System.out.println("Decryptionkeysare:"+d+","+n);
}

Public static BigIntegergenerateE(BigIntegerfiofn) {
int y, intGCD;
BigInteger e;
BigInteger gcd;
Random x=new Random();
do {
y = x.nextInt(fiofn.intValue()-1);
String z = Integer.toString(y);
e= new BigInteger(z);
gcd = fiofn.gcd(e);
intGCD = gcd.intValue();
}
while(y <= 2 ||intGCD != 1);
```

```
return e;
}
}
```

Output:

Enter a Prime number: 5

Enter another prime number: 11

Encryption keys are: 33, 55

Decryption keys are: 17, 55

Program-8: Implement Diffie-Hellman key exchange mechanism using HTML and Java script.

THEORY:

- Diffie Hellman (DH) key exchange algorithm is a way for securely exchanging cryptographic keys over internet(Public channel). Keys are not actually exchanged – they are jointly derived.
- If 1st Person and 2nd Person want to communicate with each other, they first agree between them a large prime number **P**, and a generator (or base) **G** (where 0 < G < P).
- 1st Person select a secret integer "**a**" (his private key) and then calculates $x=G^a \bmod P$ (which is his public key). 2nd Person chooses his private key **"b'**, and calculates his public key $y=G^b \bmod P$
- 1st Person and 2nd Person exchange values of x and y.
- 1st Person and 2nd Person compute symmetric keys by using

$$K_a=y^a \bmod P \quad \text{and} \quad k_b= x^b \bmod P$$

- Example:

Step 1 : 1st Person and 2nd Person get public numbers P = 23,G = 9

Step 2: 1st Person selected a private key a = 4 and 2nd Person selected a private key b = 3

Step 3: 1st Person and 2nd Person compute public values

1st Person : $x = (9^4 \bmod 23) = (6561 \bmod 23) = 6$

2nd Person: $y = (9^3 \bmod 23) = (729 \bmod 23) = 16$

Step 4: 1st Person and 2nd Person exchange public numbers

Step 5: 1st Person receives public key y =16 and 2nd Person receives public key x = 6

Step 6: 1st Person and 2nd Person compute symmetric keys

1st Person : $k_a = y^a \bmod p = 65536 \bmod 23 = 9$

2nd Person: $k_b = x^b \bmod p = 216 \bmod 23 = 9$

Step 7: 9 is the shared secret

HTML File: dh.html

```
<!DOCTYPE HTML>
<html>
 <head>
        <meta charset="utf-8"/>
        <meta name="viewport" content="width=device-width, initial-scale=1.0"/>

        <link rel='stylesheet' type='text/css'
href='http://fonts.googleapis.com/css?family=Roboto:100,100italic,200,300,400,400italic,700
%7COpen+Sans:300italic,400,300,700'>
        <link rel="stylesheet" type="text/css" href="./css/foundation.min.css" />
        <link rel="stylesheet" type="text/css" href="./css/style.css" />

        <title>Diffie Hellman Implementation</title>
    </head>
    <body>
        <div class="row small center">
            <h1>Diffie Hellman</h1>
            <h4>Implementation</h4>
        </div>
        <div class="row small">
            <dl class="tabs" data-tab>
                <dd class="active"><a href="#generate-shared-key">Get Public
Key</a></dd>
                <dd><a href="#generate-secret-key">Get Shared
Secret</a></dd>
                <dd><a href="#bruteforce">Crypta Analysis</a></dd>
            </dl>
        </div>
        <div class="tabs-content">
            <div class="content active" id="generate-shared-key">
            <div class="row small">
```

```
                                    <div class="large-4 columns">
                                        <label>Prime
                                        <input type="number" placeholder="" id="shared-prime" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                                        </label>
                                    </div>
                                    <div class="large-4 columns">
                                        <label>Base
                                            <input type="number" placeholder="" id="shared-base" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                                        </label>
                                    </div>
                                    <div class="large-4 columns">
                                        <label>Integer
                                        <input type="number" placeholder="" id="shared-integer" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                                        </label>
                                    </div>
                                </div>
                                <div class="row small">
                                    <div class="large-12 columns">
                                        <label>Public key
                                            <input type="number" placeholder="" id="shared-key" disabled="true" />
                                        </label>
                                    </div>
                                </div>
                            </div>
                            <div class="content" id="generate-secret-key">
                            <div class="row small">
                                    <div class="large-4 columns">
                                        <label>Prime
                                        <input type="number" placeholder="" id="secret-prime" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
```

```
                    </label>
                </div>
                <div class="large-4 columns">
                    <label>Integer
                        <input type="number" placeholder="" id="secret-integer" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                    </label>
                </div>
                <div class="large-4 columns">
                    <label>Public Key Pair
                    <input type="number" placeholder="" id="secret-shared" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                    </label>
                </div>
            </div>
            <div class="row small">
                <div class="large-12 columns">
                    <label>Secret key
                        <input type="number" placeholder="" id="secret-key" disabled="true" />
                    </label>
                </div>
            </div>
        </div>
        <div class="content" id="bruteforce">
        <div class="row small">
            <div class="large-3 columns">
                <label>Prime
                <input type="number" placeholder="" id="bruteforce-prime" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                </label>
            </div>
            <div class="large-3 columns">
```

```
                    <label>Base
                        <input type="number" placeholder="" id="bruteforce-base" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                    </label>
                </div>
                <div class="large-3 columns">
                    <label>Public Key A
                    <input type="number" placeholder="" id="bruteforce-shared-a" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                    </label>
                </div>
                <div class="large-3 columns">
                    <label>Public Key B
                    <input type="number" placeholder="" id="bruteforce-shared-b" onkeypress='return event.charCode >= 48 && event.charCode <= 57' />
                    </label>
                </div>
            </div>
            <div class="row small">
                <div class="large-6 columns">
                    <label>Integer A
                        <input type="number" placeholder="" id="bruteforce-integer-a" disabled="true" />
                    </label>
                </div>
                <div class="large-6 columns">
                    <label>Integer B
                        <input type="number" placeholder="" id="bruteforce-integer-b" disabled="true" />
                    </label>
                </div>
            </div>
            <div class="row small">
                <div class="large-12 columns">
```

```
                                        <label>Secret key
                                            <input type="number" placeholder=""
id="bruteforce-key" disabled="true" />
                                        </label>
                                    </div>
                                </div>
                                <div class="row small">
                                    <div class="large-12 columns">
                                            <a id="bruteforce-button" class="button
expand">Bruteforce</a>
                                        </div>
                                </div>
                            </div>
                    </div
                    <script src="js/jquery.min.js"></script>
                    <script src="js/foundation.min.js"></script>
                    <script src="js/diffie-hellman.js"></script>
                    <script src="js/apps.js"></script>
            </body>
</html>
```

Javascript: diffehellman.js

```
var diffieHellman = {
        getPublicKeyDiffieHellman: function(prime, base, integer){
                return Math.pow(base, integer) % prime;
        },
        getSharedKeyDiffieHellman: function(prime, integer, public_key){
                return Math.pow(public_key, integer) % prime;
        },
        cryptaAnalysisDiffieHellman: function(prime, base, public_key_a, public_key_b, limit){
                limit = (typeof limit === "undefined") ? 1000 : limit;
```

```
        var attempt = -1;
        var integer_a = 0;
        var integer_b = 0;

        while(attempt != public_key_a && integer_a < limit){
            attempt = this.getPublicKeyDiffieHellman(prime,base,integer_a++);
        }
        attempt = -1;
        while(attempt != public_key_b && integer_b < limit){
            attempt = this.getPublicKeyDiffieHellman(prime,base,integer_b++);
        }

        integer_a = integer_a - 1;
        integer_b = integer_b - 1;

        return {
            integer: {
                a:integer_a,
                b:integer_b
            },
            secret: this.getSharedKeyDiffieHellman(prime, integer_a, public_key_b)
        };
    }
}
```

Output:

```
The value of P: 11
The value of G: 25
The private key (a) for Alice: 6
The private key (b) for Bob: 8
Secret key for Alice is: 5
Secret key for Bob is: 5
```

Program-9: Calculate the message digest of a text using the SHA-512 algorithm in Java.

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class SHA512Digest {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the text from the user
        System.out.println("Enter the text to calculate its SHA-512 digest:");
        String text = scanner.nextLine();

        // Calculate the SHA-512 digest
        String sha512Digest = calculateSHA512(text);

        // Display the result
        System.out.println("SHA-512 Digest: " + sha512Digest);
    }

    private static String calculateSHA512(String text) {
        try {
            // Create a MessageDigest instance for SHA-512
            MessageDigest md = MessageDigest.getInstance("SHA-512");

            // Digest the input text and get the byte array
            byte[] digestBytes = md.digest(text.getBytes(StandardCharsets.UTF_8));
```

```
        // Convert the byte array to a hexadecimal string
        StringBuilder sb = new StringBuilder();
        for (byte b : digestBytes) {
            sb.append(String.format("%02x", b));
        }

        return sb.toString();

    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException("SHA-512 algorithm not found", e);
    }
  }
}
```

Program-10: Calculate the message digest of a text using the MD5 algorithm in Java.

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class MD5Digest {

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Input the text from the user
    System.out.println("Enter the text to calculate its MD5 digest:");
    String text = scanner.nextLine();
```

```
        // Calculate the MD5 digest
        String md5Digest = calculateMD5(text);

        // Display the result
        System.out.println("MD5 Digest: " + md5Digest);
    }

    private static String calculateMD5(String text) {
        try {
            // Create a MessageDigest instance for MD5
            MessageDigest md = MessageDigest.getInstance("MD5");

            // Digest the input text and get the byte array
            byte[] digestBytes = md.digest(text.getBytes(StandardCharsets.UTF_8));

            // Convert the byte array to a hexadecimal string
            StringBuilder sb = new StringBuilder();
            for (byte b : digestBytes) {
                sb.append(String.format("%02x", b));
            }

            return sb.toString();

        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("MD5 algorithm not found", e);
        }
    }
}
```