St. Vincent Pallotti College of Engineering & Technology, Nagpur
Department of Computer Engineering
Session 2024-25
CNS Practical Details
7th Semester (A & B)

Practical 4:

Aim: Implement the following regarding Modern Block Cipher components.

1. **WAP that splits an *n*-bit word into two words, each of *n*/2 bits.**

```
split (word [0 … n], n)
{
        i ← 1
        while (i ≤ n / 2)
        {
                rightWord [i] ← word [i]
                leftWord [i] ← word [i + n / 2]
                i ← i + 1
        }
        return (rightWord, leftWord)
}
```

2. **WAP that combines two *n/2* bits word into *n*-bit word.**

```
combine (rightWord [1 … m], leftWord [1 … m], m)
{
        i ← 1
        while (i ≤ m)
        {
                word [i] ← rigthWord [i]
                word [i + m] ← leftWord [i]
                i ← i + 1
        }
        return (word[1 … n])                    // n = 2m
}
```

3. **WAP that swaps the left and right halves of an *n*-bit word.**

```
swap (word [1 … n], n)
{
        i ← 1
        while (i ≤ n / 2)
        {
                temp [i + n / 2] ← word [i]
                temp [i] ← word [i + n / 2]
                i ← i + 1
        }
        return (temp[0 … n])
}
```

4. **WAP that circular shifts an *n*-bit word, *k* bits to the left or right based on the first parameter passed to the routine.**

```
circularShift (shift, word [1 … n], n, k)
{
        i ← 1
        while (i ≤ k)
        {
                if (shift = left)
                        word ← circularShiftLeft (word, n)
                else
                        word ← circularShiftRight (word, n)
                i ← i + 1
        }
        return (word[1 … n])
}
circularShiftLeft (word [1 … n], n)
{
        temp ← word [n]
        j ← n − 1
        while (j ≥ 0)
        {
                word [j +1] ← word [j]
                j ← j − 1
        }
        word [1] ← temp
        return (word[0 … n])
}
circularShiftRight (word [0 … n], n)
{
        temp ← word [1]
        j ← 1
        while (j ≤ n)
        {
                word [j −1] ← word [j]
                j ← j + 1
        }
        word [n] ← temp
        return (word[0 … n])
}
```

5. **WAP to show the mapping for P-box.**

```
P-box (inputBits [1 … n], Table [1 … m], n, m)
{
        i ← 1
        while (i ≤ m)
        {
                outputBits[i] ← inputBits[Table[i]]
                i ← i + 1
        }
        return (outputBits[1 … n])
}
```

6. **WAP to implement a Linear S-Box in which I/O is defined by a table.**

```
S-box (inputBits [1 … n], Table [1 … n], n, m)
{
        index ← binaryToDecimal (inputBits)
        value ← Table [index]
        outputBits ← decimalToBinary (value)
        return (outputBits [0 … m])
}
```

Even Roll Nos of all the batches will execute: 1,3,5

Odd Roll Nos of all the batches will execute: 2,4,6

Compulsory program for all the students:

WAP to implement at least two rounds of Feistel Cipher structure.

```
FeistelRound (inputBits [1 … n], roundKey[1 … n], n)
{
        (tempLeft, tempRight) ← split (word, n)
        tempLeft ← tempLeft ⊕ function (tempRight, roundKey)
        (tempLeft, tempRight) ← swap (tempLeft, tempRight)
        outputBits ← combine (tempLeft, tempRight)
        return (outputBits [1 … n])
}
```

Prof. Reema Roychaudhary
Practical In-charge