```
Void Programa(){
        String textoACifrar = “x texto”;
        String vectorInicializacion = {…};
        AlgoritmoIDEA(textoACifrar, vectorInicializacion);
}

string AlgoritmoIDEA(string plaintext, string[] initializationVector) {
Int valorParaLlave;
List<string> Llaves;
Llaves = Generate Key(valorParaLlave);
string cipherBlock;
string resultado;
bool completado = false;
String xorResult = XOR(plaintext, initializationVector.ToString());
Int counter = 0;
if(counter == 0){
        Byte[] valores = plaintext.toByteArray();
        cipherBlock = GenerateBlockCipher(valores,  cipherBlock);
}
Else if(counter == 1){
        Byte[] valores = cipherBlock.toByteArray();
        cipherBlock = GenerateBlockCipher(valores,  cipherBlock);
        completado = true;

}
    If(counter < 3){
        If(completado){
            AlgoritmoIDEA(plaintext, xorResult);
            }
            Else{
            Counter++;
            AlgoritmoIDEA(plaintext, xorResult);
            }
    }
Return cipherBlock;
}

Public string  GenerateBlockCipher(byte[] valores, string cipherBlock, Lista<string> Llaves){
    Foreach(value in plaintext) {
        cipherBlock += AlgoritmoSDES(value, Llaves[1], Llaves[2]);
        }
        Return cipherBlock;
}
```

```
List<string> Generate Key (int value) {
    String ValorBinario = ConvertirABinario(value);
    String K1 = “”;
    String K1Aux = ””;
    String K2 = “”;
    String K2Aux = ””;

    K1 = Permutacion10(ValorBinario);
    K1 = LeftShift(K1.Substring(4,4);
    K1Aux = LeftShift(K1.Substring(0,4);
    K1 = Permutacion8(K1+K1Aux);
    K2 = LeftShift(K1.Substring(4,4);
    K2Aux = LeftShift(K1.Substring(0,4);
    K2 = Permutacion8(K2+K2Aux);
    List<string> Llaves;
    Llaves.Add(K1);
    Llaves.Add(K2);
}

//métodos asumidos
String permutacionInicial();
String ExpandirYPermutar();
String permutacion10();
String permutacion8();
String permutacion4();
String swapBox();
String swap();
String permutacionInicialInversa;



String AlgoritmoSDES (Byte valorCifrar, string K1, string K2) {
      string valorBinario = ConvertirABinario(valorCifrar);
      string nuevoValor = “”;

      string key = “”, swap = “”;
      key = K1;
      nuevoValor = PermutacionInicial(valorBinario);
      swap = nuevoValor;
      int counter = 1;
      while(counter <= 2){
          string parte1PermutacionInicial = nuevoValor.substring(0,4);
          string parte2PermutacionInicial = nuevoValor.substring(4,4);
          string nuevoValorAux = ExpandirYPermutar(swap.subtring(4,4);
```

```
            nuevoValor = XOR(key, nuevoValorAux);
            nuevoValorAux = SwapBox(nuevoValor);
            nuevoValor = Permutar4(nuevoValorAux);
            nuevoValorAux = XOR(nuevoValor, parte1PermutacionInicial)
            nuevoValor = Swap(nuevoValorAux,  parte2PermutacionInicial);
            swap = nuevoValor.substring(4,4);
            key = K2;
            counter++;
        }
        String permutacionInicialI = nuevoValor.substring(0,4);
        nuevoValor = permutacionInversa(nuevoValorAux + permutacionInicial);
        return nuevoValor;
}
```