**Code – BLOCK SIZE CALCULATE**

**public static int BlockSizeCalculate(string passPhrase,ref string ER)**

{

var key = passPhrase;

key = key?.Trim();

if (key?.All(x => x == '0') == true)

{

return -1;

}

else if (key?.Length != 32)

{

return -2;

}

//// Convert to Binary

byte[] bkey = new byte[32];

for (int i = 0; i < key.Length; i++)

{

bkey[i] = (byte)key[i];

}

byte[] K1 = bkey.Take(2).ToArray();

byte[] K2 = bkey.Skip(2).Take(2).ToArray();

byte[] K3 = bkey.Skip(4).Take(2).ToArray();

byte[] K4 = bkey.Skip(6).Take(2).ToArray();

byte[] K5 = bkey.Skip(8).Take(2).ToArray();

byte[] K6 = bkey.Skip(10).Take(2).ToArray();

byte[] K7 = bkey.Skip(12).Take(2).ToArray();

byte[] K8 = bkey.Skip(14).Take(2).ToArray();

byte[] K9 = bkey.Skip(16).Take(2).ToArray();

byte[] K10 = bkey.Skip(18).Take(2).ToArray();

byte[] K11 = bkey.Skip(20).Take(2).ToArray();

byte[] K12 = bkey.Skip(22).Take(2).ToArray();

byte[] K13 = bkey.Skip(24).Take(2).ToArray();

byte[] K14 = bkey.Skip(26).Take(2).ToArray();

byte[] K15 = bkey.Skip(28).Take(2).ToArray();

byte[] K16 = bkey.Skip(30).Take(2).ToArray();

// Take Select k1,k3,k5,k5,k7,k9,k11,k13,k15 take complement of last 8 bits

byte[] KC1 = K1.Take(2).ToArray().Concat(K1.Skip(2).Take(2).Select(x => (byte)~x).ToArray()).ToArray();

byte[] KC3 = K1.Take(2).ToArray().Concat(K3.Skip(2).Take(2).Select(x => (byte)~x).ToArray()).ToArray();

byte[] KC5 = K1.Take(2).ToArray().Concat(K5.Skip(2).Take(2).Select(x => (byte)~x).ToArray()).ToArray();

byte[] KC7 = K1.Take(2).ToArray().Concat(K7.Skip(2).Take(2).Select(x => (byte)~x).ToArray()).ToArray();

byte[] KC9 = K1.Take(2).ToArray().Concat(K9.Skip(2).Take(2).Select(x => (byte)~x).ToArray()).ToArray();

byte[] KC11 = K1.Take(2).ToArray().Concat(K11.Skip(2).Take(2).Select(x => (byte)~x).ToArray()).ToArray();

byte[] KC13 = K1.Take(2).ToArray().Concat(K13.Skip(2).Take(2).Select(x => (byte)~x).ToArray()).ToArray();

byte[] KC15 = K1.Take(2).ToArray().Concat(K15.Skip(2).Take(2).Select(x => (byte)~x).ToArray()).ToArray();

// Take select k2,k4,k6,k8,k10,k12,k14,k16 take complement of left first 8 bits

byte[] KC2 = K2.Take(2).Select(x => (byte)~x).ToArray().Concat(K2.Skip(2).Take(2).ToArray()).ToArray();

byte[] KC4 = K4.Take(2).Select(x => (byte)~x).ToArray().Concat(K4.Skip(2).Take(2).ToArray()).ToArray();

byte[] KC6 = K6.Take(2).Select(x => (byte)~x).ToArray().Concat(K6.Skip(2).Take(2).ToArray()).ToArray();

byte[] KC8 = K8.Take(2).Select(x => (byte)~x).ToArray().Concat(K8.Skip(2).Take(2).ToArray()).ToArray();

byte[] KC10 = K10.Take(2).Select(x => (byte)~x).ToArray().Concat(K8.Skip(2).Take(2).ToArray()).ToArray();

byte[] KC12 = K12.Take(2).Select(x => (byte)~x).ToArray().Concat(K8.Skip(2).Take(2).ToArray()).ToArray();

byte[] KC14 = K14.Take(2).Select(x => (byte)~x).ToArray().Concat(K8.Skip(2).Take(2).ToArray()).ToArray();

byte[] KC16 = K16.Take(2).Select(x => (byte)~x).ToArray().Concat(K8.Skip(2).Take(2).ToArray()).ToArray();

byte[] KCX1 = new byte[2];

byte[] KCX3 = new byte[2];

byte[] KCX5 = new byte[2];

byte[] KCX7 = new byte[2];

byte[] KCX9 = new byte[2];

byte[] KCX11 = new byte[2];

byte[] KCX13 = new byte[2];

byte[] KCX15 = new byte[2];

byte[] KCX2 = new byte[2];

byte[] KCX4 = new byte[2];

byte[] KCX6 = new byte[2];

byte[] KCX8 = new byte[2];

byte[] KCX10 = new byte[2];

byte[] KCX12 = new byte[2];

byte[] KCX14 = new byte[2];

byte[] KCX16 = new byte[2];

// Take the XOR of chunk kc1+kc16 Ans will replace with kc16

for (int i = 0; i < KC1.Length; i++)

{

KC16[i] ^= KC1[i];

KCX16[i] = KC16[i];

}

// Take the XOR of chunk kc2+kc15 Ans will replace with kc15

for (int i = 0; i < KC2.Length; i++)

{

KC15[i] ^= KC2[i];

KCX15[i] = KC15[i];

}

// Take the XOR of chunk kc3+kc14 Ans will replace with kc14

for (int i = 0; i < KC3.Length; i++)

{

KC14[i] ^= KC3[i];

KCX14[i] = KC14[i];

}

// Take the XOR of chunk kc4+kc13 Ans will replace with kc13

for (int i = 0; i < KC4.Length; i++)

{

KC13[i] ^= KC4[i];

KCX13[i] = KC13[i];

}

// Now take the new XORed kc5 with kc12 and replce with kc12

for (int i = 0; i < KC5.Length; i++)

{

KC12[i] ^= KC5[i];

KCX12[i] = KC12[i];

}

// Now take the new XORed kc6 with kc11 and replce with kc11

for (int i = 0; i < KC6.Length; i++)

{

KC11[i] ^= KC6[i];

KCX11[i] = KC11[i];

}

// Now take the new XORed kc7 with kc10 and replce with kc10

for (int i = 0; i < KC7.Length; i++)

{

KC10[i] ^= KC7[i];

KCX10[i] = KC10[i];

}

// Now take the new XORed kc8 with kc9 and replce with kc9

for (int i = 0; i < KC8.Length; i++)

{

KC9[i] ^= KC8[i];

KCX9[i] = KC9[i];

}

// Take the XOR of chunk kc1+kc16 Ans will replace with kc16

for (int i = 0; i < KC9.Length; i++)

{

KC1[i] ^= KC9[i];

KCX1[i] = KC1[i];

}

// Take the XOR of chunk kc2+kc15 Ans will replace with kc15

for (int i = 0; i < KC10.Length; i++)

{

KC2[i] ^= KC10[i];

KCX2[i] = KC2[i];

}

// Take the XOR of chunk kc3+kc14 Ans will replace with kc14

for (int i = 0; i < KC11.Length; i++)

{

KC3[i] ^= KC11[i];

KCX3[i] = KC3[i];

}

// Take the XOR of chunk kc4+kc13 Ans will replace with kc13

for (int i = 0; i < KC12.Length; i++)

{

KC4[i] ^= KC12[i];

KCX4[i] = KC4[i];

}

// Now take the new XORed kc5 with kc12 and replce with kc12

for (int i = 0; i < KC13.Length; i++)

{

KC5[i] ^= KC13[i];

KCX5[i] = KC5[i];

}

// Now take the new XORed kc6 with kc11 and replce with kc11

for (int i = 0; i < KC14.Length; i++)

{

KC6[i] ^= KC14[i];

KCX6[i] = KC6[i];

}

// Now take the new XORed kc7 with kc10 and replce with kc10

for (int i = 0; i < KC15.Length; i++)

{

KC7[i] ^= KC15[i];

KCX7[i] = KC7[i];

}

// Now take the new XORed kc8 with kc9 and replce with kc9

for (int i = 0; i < KC16.Length; i++)

{

KC8[i] ^= KC16[i];

KCX8[i] = KC8[i];

}

// Take the decimal of new sting “kcx”

int v = BitConverter.ToInt16(KCX1, 0);

var kk = string.Join("", KCX1.ToList().Select(x => Convert.ToString(x, 2).PadLeft(8, '0')));

var KCXD1 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX1, 0)));

var KCXD2 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX2, 0)));

var KCXD3 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX3, 0)));

var KCXD4 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX4, 0)));

var KCXD5 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX5, 0)));

var KCXD6 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX6, 0)));

var KCXD7 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX7, 0)));

var KCXD8 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX8, 0)));

var KCXD9 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX9, 0)));

var KCXD10 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX10, 0)));

var KCXD11 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX11, 0)));

var KCXD12 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX12, 0)));

var KCXD13 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX13, 0)));

var KCXD14 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX14, 0)));

var KCXD15 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX15, 0)));

var KCXD16 = Convert.ToUInt32(Math.Abs(BitConverter.ToInt16(KCX16, 0)));

// Take the XOR of (1)+(16)= 2 bit block 1

for (int i = 0; i < KCX1.Length; i++)

{

KCX1[i] ^= KCX16[i];

KCX1[i] = KCX1[i];

}

// Take the XOR of (2)+(15)= 2 bit block 1

for (int i = 0; i < KCX2.Length; i++)

{

KCX2[i] ^= KCX15[i];

KCX2[i] = KCX2[i];

}

// Take the XOR of (3)+(14)= 2 bit block 1

for (int i = 0; i < KCX3.Length; i++)

{

KCX3[i] ^= KCX14[i];

KCX3[i] = KCX3[i];

}

// Take the XOR of (4)+(13)= 2 bit block 1

for (int i = 0; i < KCX4.Length; i++)

{

KCX4[i] ^= KCX13[i];

KCX4[i] = KCX4[i];

}

// Take the XOR of (5)+(12)= 2 bit block 1

for (int i = 0; i < KCX5.Length; i++)

{

KCX5[i] ^= KCX12[i];

KCX5[i] = KCX5[i];

}

// Take the XOR of (6)+(11)= 2 bit block 1

for (int i = 0; i < KCX6.Length; i++)

{

KCX6[i] ^= KCX11[i];

KCX6[i] = KCX6[i];

}

// Take the XOR of (7)+(10)= 2 bit block 1

for (int i = 0; i < KCX7.Length; i++)

{

KCX7[i] ^= KCX10[i];

KCX7[i] = KCX7[i];

}

// Take the XOR of (8)+(9)= 2 bit block 1

for (int i = 0; i < KCX8.Length; i++)

{

KCX8[i] ^= KCX9[i];

KCX8[i] = KCX8[i];

}

//Take XOR for 8 to 4 pairs

// Take the XOR of (1)+(8)= 2 bit block 1

for (int i = 0; i < KCX1.Length; i++)

{

KCX1[i] ^= KCX8[i];

KCX1[i] = KCX1[i];

}

// Take the XOR of (2)+(7)= 2 bit block 1

for (int i = 0; i < KCX2.Length; i++)

{

KCX2[i] ^= KCX7[i];

KCX2[i] = KCX2[i];

}

// Take the XOR of (3)+(6)= 2 bit block 1

for (int i = 0; i < KCX3.Length; i++)

{

KCX3[i] ^= KCX6[i];

KCX3[i] = KCX3[i];

}

// Take the XOR of (4)+(5)= 2 bit block 1

for (int i = 0; i < KCX4.Length; i++)

{

KCX4[i] ^= KCX5[i];

KCX4[i] = KCX4[i];

}

var k = Math.Log(Convert.ToUInt16(KCXD1), 2);

var Ans1 = Math.Log(Convert.ToUInt16(KCXD1), 2);

var Ans2 = Math.Log(Convert.ToUInt16(KCXD2), 2);

var Ans3 = Math.Log(Convert.ToUInt16(KCXD3), 2);

var Ans4 = Math.Log(Convert.ToUInt16(KCXD4), 2);

var Ans5 = Math.Log(Convert.ToUInt16(KCXD5), 2);

var Ans6 = Math.Log(Convert.ToUInt16(KCXD6), 2);

var Ans7 = Math.Log(Convert.ToUInt16(KCXD7), 2);

var Ans8 = Math.Log(Convert.ToUInt16(KCXD8), 2);

var Ans9 = Math.Log(Convert.ToUInt16(KCXD9), 2);

var Ans10 = Math.Log(Convert.ToUInt16(KCXD10), 2);

var Ans11 = Math.Log(Convert.ToUInt16(KCXD11), 2);

var Ans12 = Math.Log(Convert.ToUInt16(KCXD12), 2);

var Ans13 = Math.Log(Convert.ToUInt16(KCXD13), 2);

var Ans14 = Math.Log(Convert.ToUInt16(KCXD14), 2);

var Ans15 = Math.Log(Convert.ToUInt16(KCXD15), 2);

var Ans16 = Math.Log(Convert.ToUInt16(KCXD16), 2);

//Now we will Convert To Decimal

var KCXDL1 = Convert.ToDecimal(Ans1);

var KCXDL2 = Convert.ToDecimal(Ans2);

var KCXDL3 = Convert.ToDecimal(Ans3);

var KCXDL4 = Convert.ToDecimal(Ans4);

var KCXDL5 = Convert.ToDecimal(Ans5);

var KCXDL6 = Convert.ToDecimal(Ans6);

var KCXDL7 = Convert.ToDecimal(Ans7);

var KCXDL8 = Convert.ToDecimal(Ans8);

var KCXDL9 = Convert.ToDecimal(Ans9);

var KCXDL10 = Convert.ToDecimal(Ans10);

var KCXDL11 = Convert.ToDecimal(Ans11);

var KCXDL12 = Convert.ToDecimal(Ans12);

var KCXDL13 = Convert.ToDecimal(Ans13);

var KCXDL14 = Convert.ToDecimal(Ans14);

var KCXDL15 = Convert.ToDecimal(Ans15);

var KCXDL16 = Convert.ToDecimal(Ans16);

byte[] bKCXDL1 = new byte[2];

for (int i = 0; i < bKCXDL1.Length; i++)

{

bKCXDL1[i] = (byte)KCXDL1.ToString()[i];

}

byte[] bKCXDL2 = new byte[2];

for (int i = 0; i < bKCXDL2.Length; i++)

{

bKCXDL2[i] = (byte)KCXDL2.ToString()[i];

}

byte[] bKCXDL3 = new byte[2];

for (int i = 0; i < bKCXDL3.Length; i++)

{

bKCXDL3[i] = (byte)KCXDL3.ToString()[i];

}

byte[] bKCXDL4 = new byte[2];

for (int i = 0; i < bKCXDL4.Length; i++)

{

bKCXDL4[i] = (byte)KCXDL4.ToString()[i];

}

byte[] bKCXDL5 = new byte[2];

for (int i = 0; i < bKCXDL5.Length; i++)

{

bKCXDL5[i] = (byte)KCXDL5.ToString()[i];

}

byte[] bKCXDL6 = new byte[2];

for (int i = 0; i < bKCXDL6.Length; i++)

{

bKCXDL6[i] = (byte)KCXDL6.ToString()[i];

}

byte[] bKCXDL7 = new byte[2];

for (int i = 0; i < bKCXDL7.Length; i++)

{

bKCXDL7[i] = (byte)KCXDL7.ToString()[i];

}

byte[] bKCXDL8 = new byte[2];

for (int i = 0; i < bKCXDL8.Length; i++)

{

bKCXDL8[i] = (byte)KCXDL8.ToString()[i];

}

byte[] bKCXDL9 = new byte[2];

for (int i = 0; i < bKCXDL9.Length; i++)

{

bKCXDL9[i] = (byte)KCXDL9.ToString()[i];

}

byte[] bKCXDL10 = new byte[2];

for (int i = 0; i < bKCXDL10.Length; i++)

{

bKCXDL10[i] = (byte)KCXDL10.ToString()[i];

}

byte[] bKCXDL11 = new byte[2];

for (int i = 0; i < bKCXDL11.Length; i++)

{

bKCXDL11[i] = (byte)KCXDL11.ToString()[i];

}

byte[] bKCXDL12 = new byte[2];

for (int i = 0; i < bKCXDL12.Length; i++)

{

bKCXDL12[i] = (byte)KCXDL12.ToString()[i];

}

byte[] bKCXDL13 = new byte[2];

for (int i = 0; i < bKCXDL13.Length; i++)

{

bKCXDL13[i] = (byte)KCXDL13.ToString()[i];

}

byte[] bKCXDL14 = new byte[2];

for (int i = 0; i < bKCXDL14.Length; i++)

{

bKCXDL14[i] = (byte)KCXDL14.ToString()[i];

}

byte[] bKCXDL15 = new byte[2];

for (int i = 0; i < bKCXDL15.Length; i++)

{

bKCXDL15[i] = (byte)KCXDL15.ToString()[i];

}

byte[] bKCXDL16 = new byte[2];

for (int i = 0; i < bKCXDL16.Length; i++)

{

bKCXDL16[i] = (byte)KCXDL16.ToString()[i];

}

// Take the XOR of (1)+(16)= 2 bit block 1

for (int i = 0; i < bKCXDL1.Length; i++)

{

bKCXDL1[i] ^= bKCXDL4[i];

bKCXDL1[i] = bKCXDL1[i];

}

// Take the XOR of (2)+(15)= 2 bit block 1

for (int i = 0; i < bKCXDL2.Length; i++)

{

bKCXDL2[i] ^= bKCXDL3[i];

bKCXDL2[i] = bKCXDL2[i];

}

//// Take the XOR of (3)+(14)= 2 bit block 1

for (int i = 0; i < bKCXDL14.Length; i++)

{

bKCXDL3[i] ^= bKCXDL14[i];

}

// Take the XOR of (4)+(13)= 2 bit block 1

for (int i = 0; i < bKCXDL4.Length; i++)

{

bKCXDL4[i] ^= bKCXDL13[i];

}

// Take the XOR of (5)+(12)= 2 bit block 1

for (int i = 0; i < bKCXDL5.Length; i++)

{

bKCXDL5[i] ^= bKCXDL12[i];

}

// Take the XOR of (6)+(11)= 2 bit block 1

for (int i = 0; i < bKCXDL6.Length; i++)

{

bKCXDL6[i] ^= bKCXDL11[i];

}

// Take the XOR of (7)+(10)= 2 bit block 1

for (int i = 0; i < bKCXDL7.Length; i++)

{

bKCXDL7[i] ^= bKCXDL10[i];

}

// Take the XOR of (8)+(9)= 2 bit block 1

for (int i = 0; i < bKCXDL8.Length; i++)

{

bKCXDL8[i] ^= bKCXDL9[i];

}

//Take XOR for 8 to 4 pairs

// Take the XOR of (1)+(8)= 2 bit block 1

for (int i = 0; i < bKCXDL1.Length; i++)

{

bKCXDL1[i] ^= bKCXDL8[i];

}

// Take the XOR of (2)+(7)= 2 bit block 1

for (int i = 0; i < bKCXDL2.Length; i++)

{

bKCXDL2[i] ^= bKCXDL7[i];

}

// Take the XOR of (3)+(6)= 2 bit block 1

for (int i = 0; i < bKCXDL3.Length; i++)

{

bKCXDL3[i] ^= bKCXDL6[i];

}

// Take the XOR of (4)+(5)= 2 bit block 1

for (int i = 0; i < bKCXDL4.Length; i++)

{

bKCXDL4[i] ^= bKCXDL5[i];

}

//Take XOR for 4 to 2 pairs

// Take the XOR of (1)+(4)= 2 bit block 1

for (int i = 0; i < KCX1.Length; i++)

{

KCX1[i] ^= KCX4[i];

KCX1[i] = KCX1[i];

}

//// Take the XOR of (2)+(3)= 2 bit block 1

for (int i = 0; i < KCX2.Length; i++)

{

KCX2[i] ^= KCX3[i];

KCX2[i] = KCX2[i];

}

//Take XOR for 2 to 1 pairs

// Take the XOR of (1)+(2)= 2 bit block 1

for (int i = 0; i < KCX1.Length; i++)

{

KCX1[i] ^= KCX2[i];

KCX1[i] = KCX1[i];

}

var FA = Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL1, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL2, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL3, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL4, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL5, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL6, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL7, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL8, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL9, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL10, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL11, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL12, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL13, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL14, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL15, 0)));

FA = FA + Convert.ToUInt64(Math.Abs(BitConverter.ToInt16(bKCXDL16, 0)));

ER = FA.ToString();

var ln = FA.ToString().Length;

string lst3 = "0";

if (ln > 3)

{

lst3 = FA.ToString().Substring(ln - 3);

}

UInt64 result = 0;

UInt64 modNum = 0;

if (lst3 != "0" )

{

modNum = Convert.ToUInt64(lst3);

result = FA % modNum;

}

// while (result >= 512) result -256;

while (result > 512) result = result - 256;

if (result < 256 && result > 0)

{

result = 512 - result;

}

int fin1 = (next8((int)result) == 0) ? (int)result : next8((int)result);

return (int)fin1;

}

public static int next8(int n)

{

int bits = n & 7; // give us the distance to the previous 8

if (bits == 0) return 0;// ("Number already divisible by 8");

return n + (8 - bits);

}

**PERMUTATION & SUBSTITUTION**