1. Basic

Hint: "key: **a0-d3-57-17-e2-17-98-82-ae-42-0b-df-2a-80-ec-d0-1b-f2-2e-62-67-96-f3-ba**"

Check name and password:

```
            return;
        }
        byte[] bytes = Encoding.UTF8.GetBytes(this.textBox1.Text);
        byte[] array = Form1.FromHex(this.textBox2.Text);
        byte[] array2 = Form1.Encode(bytes, Encoding.UTF8.GetBytes("c-sharp"));
        for (int i = 0; i < array.Length; i++)
        {
            if (array2[i] != array[i])
            {
                MessageBox.Show("Invalid name or key!", "Error");
                return;
            }
        }
        MessageBox.Show("C++++ is C#!", "Correct");
    }
```

- Compare array to array2
- array: password
- array2: the result of the Encode function with the string bytes (name) and the key ('c-sharp')

Function byte[] Encode(byte[] data, byte[] key)

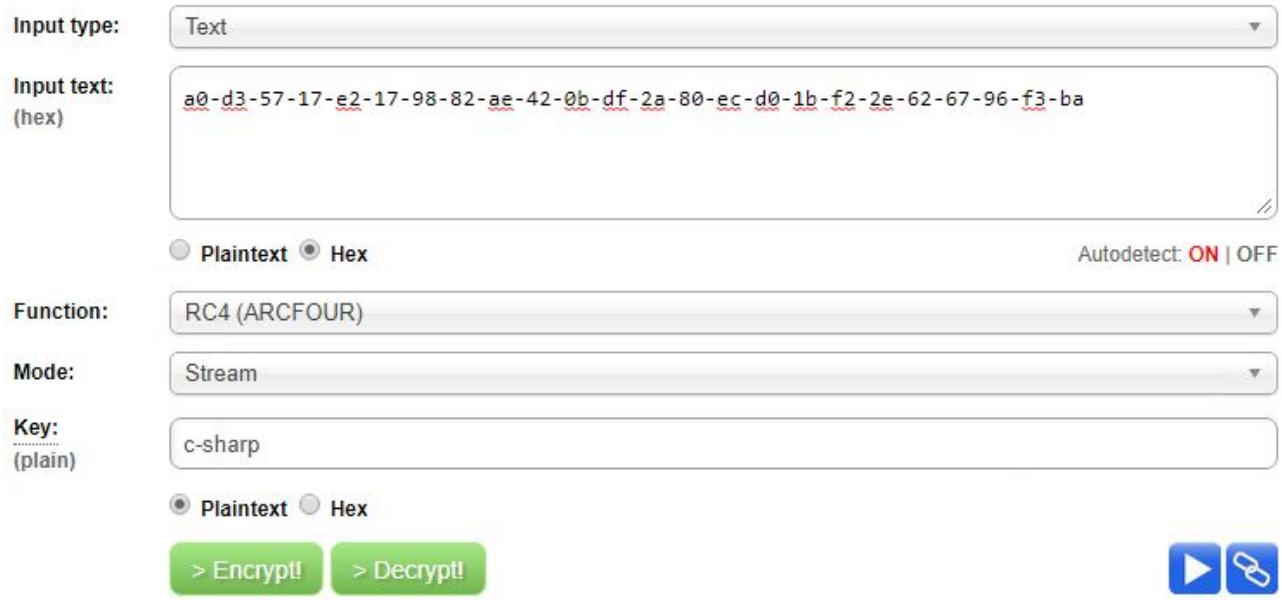
- **data**: input name (bytes variable)
- **key**: 'c-sharp' ('\x63\x2d\x73\x68\x61\x72\x70')
- **encode: RC4**

```
// Token: 0x06000005 RID: 5 RVA: 0x00002170 File Offset: 0x00000370
public static byte[] Encode(byte[] data, byte[] key)
{
    int[] array = new int[256];
    for (int i = 0; i < 256; i++)
    {
        array[i] = i;
    }
    int[] array2 = new int[256];
    if (key.Length == 256)
    {
        Buffer.BlockCopy(key, 0, array2, 0, key.Length);
    }
    else
    {
        for (int j = 0; j < 256; j++)
        {
            array2[j] = (int)key[j % key.Length];
        }
    }
    int num = 0;
    int k;
```

```
    int num = 0;
    int k;
    for (k = 0; k < 256; k++)
    {
        num = (num + array[k] + array2[k]) % 256;
        int num2 = array[k];
        array[k] = array[num];
        array[num] = num2;
    }
    num = (k = 0);
    byte[] array3 = new byte[data.Length];
    for (int l = 0; l < data.Length; l++)
    {
        k = (k + 1) % 256;
        num = (num + array[k]) % 256;
        int num3 = array[k];
        array[k] = array[num];
        array[num] = num3;
        int num4 = array[(array[k] + array[num]) % 256];
        array3[l] = Convert.ToByte((int)data[l] ^ num4);
    }
    return array3;
}
```

Solution:

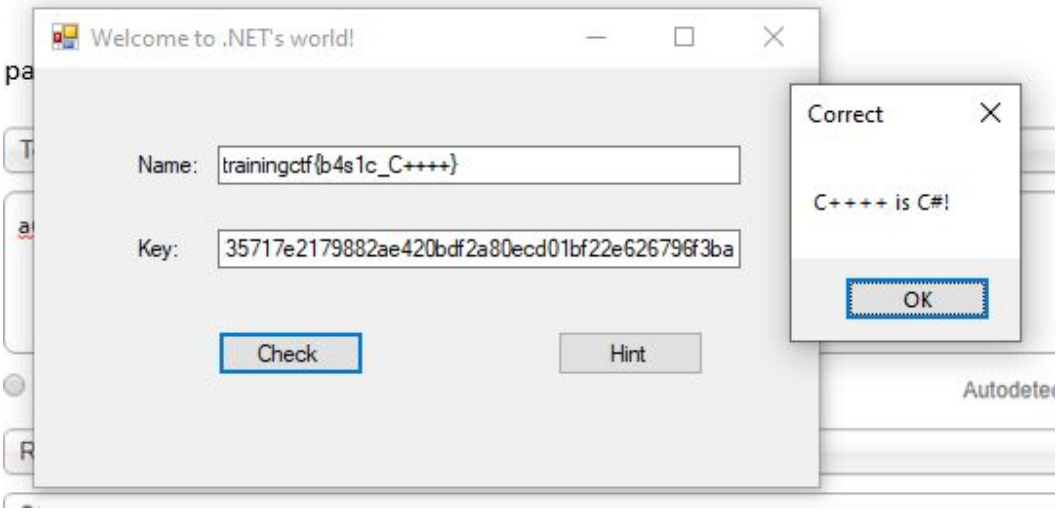
- **Using the password hint to decode rc4 with the key**



- **We have the result:**

- Hex: **74 72 61 69 6E 69 6E 67 63 74 66 7B 62 34 73 31 63 5F 43 2B 2B 2B 2B 7D**
- Text: **trainingctf{b4s1c_C++++}**

Run and check the result:



2. Find_Imposter

The “main” program:

```
// Token: 0x06000004 RID: 4 RVA: 0x00002BB0 File Offset: 0x00000DB0
private void button1_Click(object USuFj7d5SoYLp2yfO29dT2BrrYFQH4w4MTq4fyR5ou9nQ6UeQRlHJAe, EventArgs
  naX1jgvP1QM5G8Wi2x5ojAZSvkLGxgfjCPYNrFiz6yVqSXlu5fp8FECDgpNrxj)
{
    string s = Form1.CreateHash(this.n5MMxvTtkbZkOeDHzxJX5OeJppn0Ew40OaeM1H7nOo7X3IhXnUosMAC4mn5BLOUtT3R1l.Text);
    byte[] bytes = Encoding.UTF8.GetBytes(s);
    byte[] bytes2 = Encoding.UTF8.GetBytes("among-us");
    byte[] array = Form1.Encode(bytes, bytes2);
    byte[] array2 = Form1.FromHex("40-72-b1-25-9e-ff-83-f3-07-c4-e8-d6-8a-a6-0c-e0-ef-9f-a6-3f-e2-fc-0b-81-2a-47-
      dd-8b-1a-a3-4c-32");
    int num = 0;
    if (num >= 32)
    {
        return;
    }
    if (array[num] != array2[num])
    {
        MessageBox.Show("Try again!", "Defeat");
        return;
    }
    MessageBox.Show("Your flag is: trainingctf{" +
      this.n5MMxvTtkbZkOeDHzxJX5OeJppn0Ew40OaeM1H7nOo7X3IhXnUosMAC4mn5BLOUtT3R1l.Text + "_is_imposter!}", "Victory");
}
```

- **this.n5MMxvTtkbZkOeDHzxJX5OeJppn0Ew40OaeM1H7nOo7X3IhXnUosMAC4mn5BLOUtT3R1l.Text:** the input text from the textbox

Function string CreateHash(string)

- **The main function:**

```
// Token: 0x0600010D RID: 269 RVA: 0x00002E8C File Offset: 0x0000108C
public static string j4mMyqi176FiQDogGzESz4Om9j8Fuf1LjWlyLODE1cUDmjDaHIHK64qUPnW2X(string A_0)
{
    string result;
    using (MD5 md = MD5.Create())
    {
        byte[] bytes = Encoding.ASCII.GetBytes(A_0);
        byte[] array = md.ComputeHash(bytes);
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < array.Length; i++)
        {
            stringBuilder.Append(array[i].ToString("X2"));
        }
        result = stringBuilder.ToString();
    }
    return result;
}
```

- **Hash function: MD5 encryption**

Variable {s}: the MD5 hashing value of input text

Variable {array}: the result of the Encode function with the string {bytes} and the key (‘among-us’)

Function byte[] Encode(byte[], byte[])

- **The main function:**

```
// Token: 0x0600010C RID: 268 RVA: 0x00002C48 File Offset: 0x00000E48
public static byte[] Un4h993EKLUUSX7j44sFonUMI51xa96Y8ex1XWfscxs318xakeC2x7o8iTLnNILG(byte[] A_0, byte[] A_1)
{
    int[] array = new int[256];
    for (int i = 0; i < 256; i++)
    {
        array[i] = i;
    }
    int[] array2 = new int[256];
    if (A_1.Length == 256)
    {
        Buffer.BlockCopy(A_1, 0, array2, 0, A_1.Length);
    }
    else
    {
        for (int j = 0; j < 256; j++)
        {
            array2[j] = (int)A_1[j % A_1.Length];
        }
    }
    int num = 0;
    int k;
    for (k = 0; k < 256; k++)
    {
        num = (num + array[k] + array2[k]) % 256;
        int num2 = array[k];
        array[k] = array[num];
        array[num] = num2;
    }
    num = (k = 0);
    byte[] array3 = new byte[A_0.Length];
    for (int l = 0; l < A_0.Length; l++)
    {
        k = (k + 1) % 256;
        num = (num + array[k]) % 256;
        int num3 = array[k];
        array[k] = array[num];
        array[num] = num3;
        int num4 = array[(array[k] + array[num]) % 256];
        array3[l] = Convert.ToByte((int)A_0[l] ^ num4);
    }
    return array3;
}
```

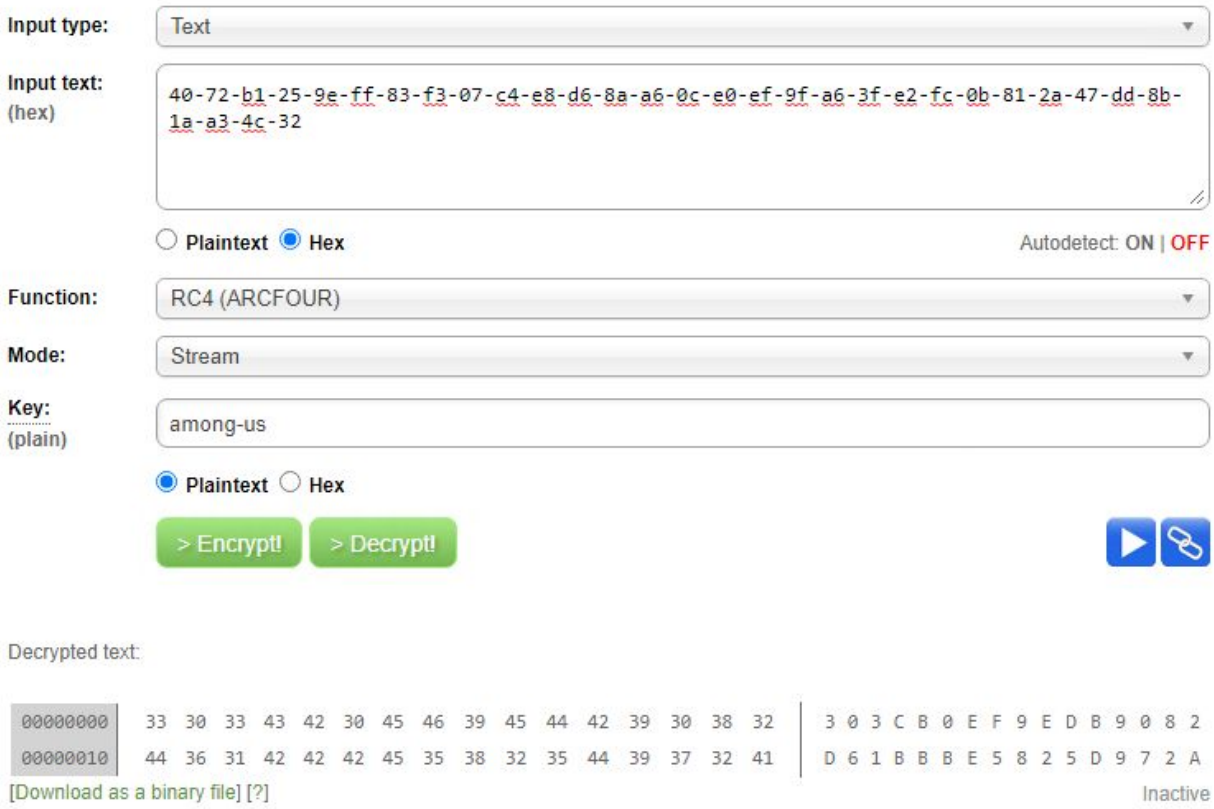
- **A_0: input text**
- **A_1: key {bytes2} ('among-us')**
- **Encode: RC4**

Checking part:

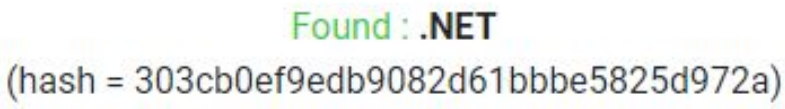
- Compare array[0] to array2[0]
- Variable {array2}: [40, 72, b1, 25, 9e, ff, 83, f3, 07, c4, e8, d6, 8a, a6, 0c, e0, ef, 9f, a6, 3f, e2, fc, 0b, 81, 2a, 47, dd, 8b, 1a, a3, 4c, 32]

Solution:

- Like the previous task, then we are decrypting the text {array2} by using the existing key {bytes2} **('among-us')** in RC4 algorithm



- Then, we had the MD5 hashing value ("303CB0EF9EDB9082D61BBBE5825D972A") which is the value of variable {s}
- Finally, we just have to decrypt the hashing value 303CB0EF9EDB9082D61BBBE5825D972A



- The input needed to find is **.NET**

Run and check the result