Implement Secure Communications

Abstract

In this assignment, I implemented two applications (windows application and web application) that simulate a secure communication between two servers. We consider two different parties named Alice and Bob and implement a windows application program with C# that shows how they can communicate to each other in a safe way. For securing their communication, we use cryptography concepts like Diffie-Hellman algorithm, generate prime number and cyclic group and a generator for the cyclic group. Then each party generates its own public and private key based on this information and then they send to each other their public key through an unsecure channel like internet. (Implemented in part2 of my program). In this phase, both of them are able to generate their shared key which is used in BBS (that is a pseudo-random number generator) to generate their secret keys. Finally, they use KES cipher algorithm (implemented in assignment1) to encrypt their data based on this key and then transfer to other party so that the communication will be very safe. In part2, we focus on implementing a safe communication between two different servers trying to send data to each other. In fact, we use the implemented concepts and algorithms in part1 to make communication between two servers safe

1. Introduction

After understanding the concept of symmetric and asymmetric ciphering for encrypting data, now we want to investigate how we can send this encrypted data in a secure way. Many channels that are widely used like internet are very insecure. So, we need some concepts like public-key exchange protocol. The first method that was used widely for communicating data through insecure channels was Diffie-Hellman key exchange. It is worth mentioning that although in the real word communications this algorithm uses a very large number as prime number, I restricted the length of prime number in my program. In the following section, I will explain the process of implementing this algorithm in detail.

2. Design and Implementation

2.1 Part I

I developed a program in C# that helps Alice and Bob send data to each other in a secure way. First of all, they should agree on The Diffie-Hellman key parameters like prime number (p), Cyclic group and generator (g). There are different approaches to create cyclic group. my program generates a random prime number with 3 or 4 digits to decrease computational burden on CPU. (But in the real word applications, very large Prime number will be selected). The next step is finding the smallest primitive root of this prime number.

It is worth mentioning that the primitive root of a prime number p is an integer "i" in range of [2, n-1] such that the values of i^x(mod n) where x is in the range[0, n-2] are different. So, if the program finds a primitive root for the selected "p", we can call group: {1, 2, …, p-1} as our cyclic group with generator "g" and it is closed to multiplication.

If there is no primitive root for the selected prime number, program returns -1 for parameter "g", which means we cannot form a linear cyclic group of {1, 2, …, p-1} for this prime number (p) and we should select another prime number and repeat the process of finding primitive root.

It is worth mentioning that based on cyclic group's definition q=(p-1)/2

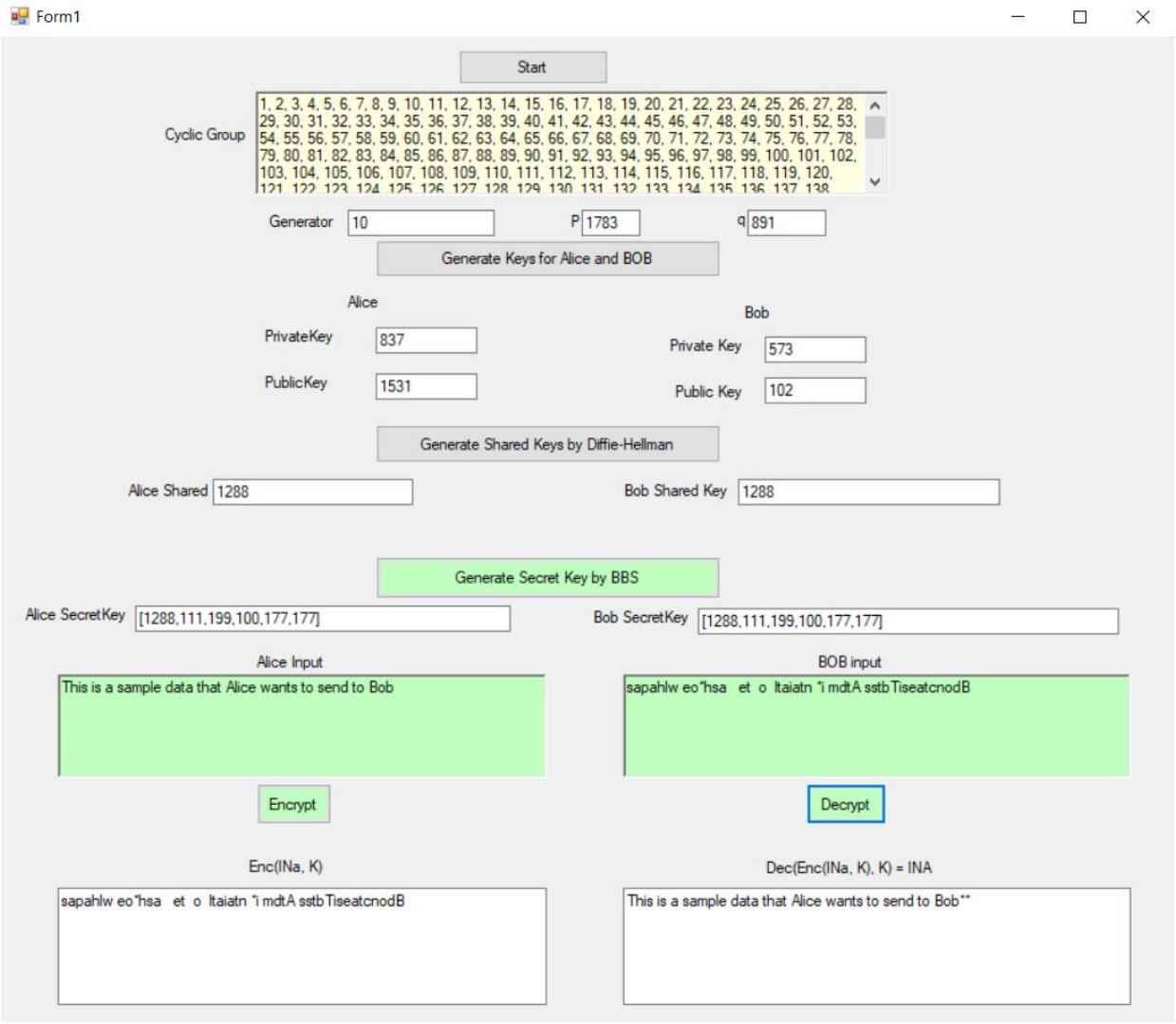

Figure 1

As we can see in the Figure1, my program generated P=1783, q=861. And the smallest primitive root for P is 10 which is called "g".

Therefore, we can form the cyclic group as {1, 2, 3, …, 1782}.

Now we can easily calculate public key and private key based on the Diffie-Hellman algorithm.

The next step in Diffie-Hellman algorithm is exchanging public key between Alice and Bob. so that they will be able to calculate their shared key which will be the same for both of them.

To implement this, we defined 2 different classes named Alice and Bob. Alice calls Bob's Send_PublicKey function, and Bob calls Alice's Send_PublicKey function:

```
1 reference | 0 changes | 0 authors, 0 changes
private void DiffieHellman_Click(object sender, EventArgs e)
{
    Alice.SharedKey =CyclicGroup.power(Bob.Send_PublicKey(), Alice.PrivateKey,Prime);

    Bob.SharedKey = CyclicGroup.power(Alice.Send_PublicKey(), Bob.PrivateKey,Prime);

    Alicesharedkey.Text = Alice.SharedKey.ToString();
    bobShared.Text = Bob.SharedKey.ToString();
}
```

Note: the method, power(a,b,c), gets 3 parameters a,b,c and returns a^b%n.

After this, Alice and Bob need a pseudo-random number generator (CSPRNG) like BBS to generate their Secret Key. In fact, BBS function gets their shared key as a seed and generates an array of decimal numbers or bits as a secret key. Alice and Bob have decided the length of their secret key to be 5. (It can be more than 5).

```
1 reference | 0 changes | 0 authors, 0 changes
public static int[] BBShub_GenerateKey(int seed, int p)
{
    int q = (p-1)/2;
    int n = p * q;

    int x0 = seed;
    int[] key = new int[5];
    int[] X = new int[5];


    X[0] = seed;
    key[0] = seed % 2;

    for (int i = 1; i < key.Length; i++)
    {

        X[i] = CyclicGroup.power((X[i - 1]), 2, n);
        if (X[i] < 0) X[i] = X[i] + n;


        key[i] = X[i] % 2;
    }
    return X;
}
```

As you see the result of BBS algorithm can be an array of binaries (int[] key) or an array of decimal numbers (int[] X = new int[5]). If you want to use AES ciphering to encrypt and decrypt data, you can use the array of binary numbers. But since I decided to use KES encryption and decryption algorithm (that I implemented in assignment1). I used an array of decimal numbers generated by BBS as secret key. As it is clear in Figure 1, Alice can enter any string data and encrypt it and we call the result (Enc (INa, K)) which is generated by processes mentioned above.

Now if Bob enters Enc (INa, K) as input and applies the KES decryption function using his own secret key, he can easily decrypt the file transferred by Alice. The decrypted text called Dec (Enc (INa, K), K) = INA, that will be as same as Alice's input.

2.2 Part 2

In this part, we want to implement a real-world communication between two different parties named Alice and Bob. I created a project for Alice and another project for Bob. But their codes are the same. As you see in Figure2, after running Alice's project, you should go in "properties" and then in "web" and dedicate the port 44360 to Alice.

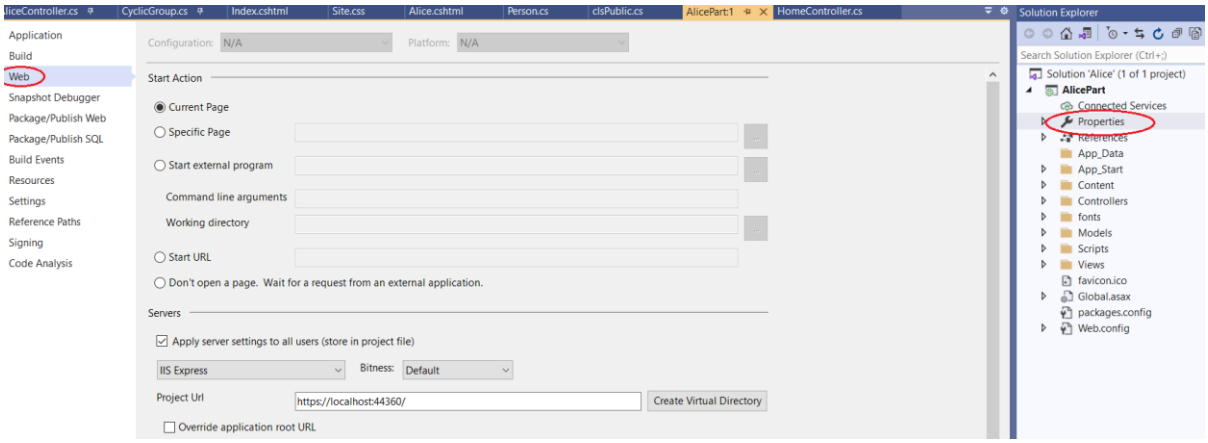

Figure 2

And then run the Bob project and dedicate port 44300 to Bob. (Figure3)

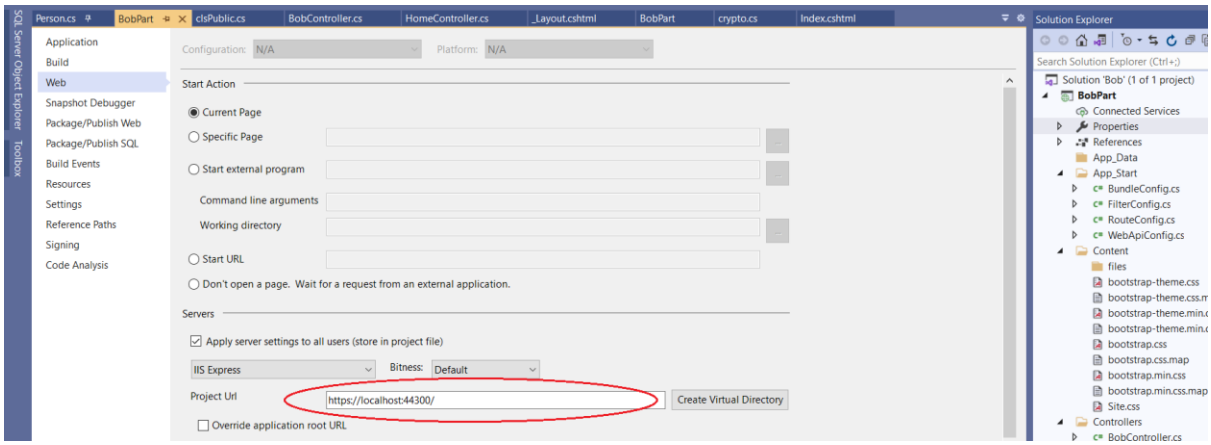

Figure 3

So, in this way, we can simulate communicating between two servers.

I implemented program in Asp.net MVC and used web Api for sending and receiving data between severs. I considered a public class named clsPublic and tried to set general parameters like prime number, cyclic group, generator of cyclic group in this class.

When Alice and Bob run their programs, they need to compromise on the same Prime numbers and consequently same cyclic group and same generator. So, when each one runs, send its own prime number to the other one, if the other one has already generated prime number, change its prime number to this one so that their Prime number will be the same.

Finally, both Alice and Bob will have the same prime number, generator and cyclic group. (Figure4)

```
//---------------------------------------------------------------
//-------------------- Send Prime To Bob ------------------------
//---------------------------------------------------------------
try
{
    using (HttpClient client = new HttpClient())
    {
        client.BaseAddress = new Uri("https://localhost:44300/");
        var T = client.GetAsync("api/GetPrime?id=" + clsPublic.Prime.ToString());
        T.Wait();
        if (T.Result.StatusCode == System.Net.HttpStatusCode.OK)
        {
            var TData = T.Result.Content.ReadAsStringAsync();
            TData.Wait();
            string ReturnPrime = TData.Result.Substring(1, TData.Result.Length - 2);
            if (ReturnPrime != clsPublic.Prime.ToString())
                clsPublic.Prime = Convert.ToInt32(ReturnPrime);
        }
    }
}
```

Figure4

After that, each party starts to generate its own private and public key. In this phase, based on the Diffie-Hellman algorithm, Alice and Bob should send their public keys to each other.

In figure 5, you can see how Alice asks Bob for his public through web api.

```
[HttpPost]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult ExchangeKey(string mystr)
{
    //string AlicePublicKey = "123";
    string BobKey = "";
    int BobPublicKey;
    string alice_secret_key = "[";

    using (HttpClient client = new HttpClient())
    {
        client.BaseAddress = new Uri("https://localhost:44300/");
        var T = client.GetAsync("api/GetPublicKey?id=" + clsPublic.PublicKeyAlice);
        T.Wait();
        if (T.Result.StatusCode == System.Net.HttpStatusCode.OK)
        {
            var TData = T.Result.Content.ReadAsStringAsync();
            TData.Wait();
            BobKey = TData.Result.Substring(1, TData.Result.Length - 2);
            clsPublic.PublicKeyBob = Convert.ToInt32(BobKey);
            Session["BobPublicKey"] = BobKey;
        }
    }
```

Figure 5

On the other side, Bob asks for Alice's public key in the same way. Now, both of them have all necessary parameters for generating shared key. After generating shared key, as I

explained in part1, they apply BBS algorithm which uses shared key as a seed and returns an array of decimal or binary numbers as secret key.

Now each one has this secret key and is able to encrypt and decrypt message. In this program, I decided to use KES algorithm for encryption and decryption. It uses the secret key as key and the length of key that both parties comprised on, is 5. In fact, the BBS algorithm generate an array of decimals with the length of 5.

When Alice encrypts the file and sends it to Bob with web Api, Bob sends a confirmation message to Alice that shows data has transferred successfully. (Figure 6)

```
[HttpPost]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult Encryptfile()
{
    string FinalCipher = clsCipher.EncodeTextWithKey();
    Session["FinalCipher"] = FinalCipher;


    using (HttpClient client = new HttpClient())
    {
        client.BaseAddress = new Uri("https://localhost:44300/");
        //sending Encrypted data to Bob
        var T = client.GetAsync("api/UploadEncodeText?encodetext=" + FinalCipher);
        T.Wait();
        if (T.Result.StatusCode == System.Net.HttpStatusCode.OK)
        {
            var TData = T.Result.Content.ReadAsStringAsync();
            TData.Wait();
            string BobDecodeText = TData.Result.Substring(1, TData.Result.Length - 2);
            Session["BobDecodeText"] = "I recieved Alice's File";// confirmation from Bob
        }
    }

    return View("Index");
}
```

Figure6

Now if Bob clicks the button "Show File Received from Alice", he will be able to download both encrypted and decrypted file and see the content. (Figure 7)

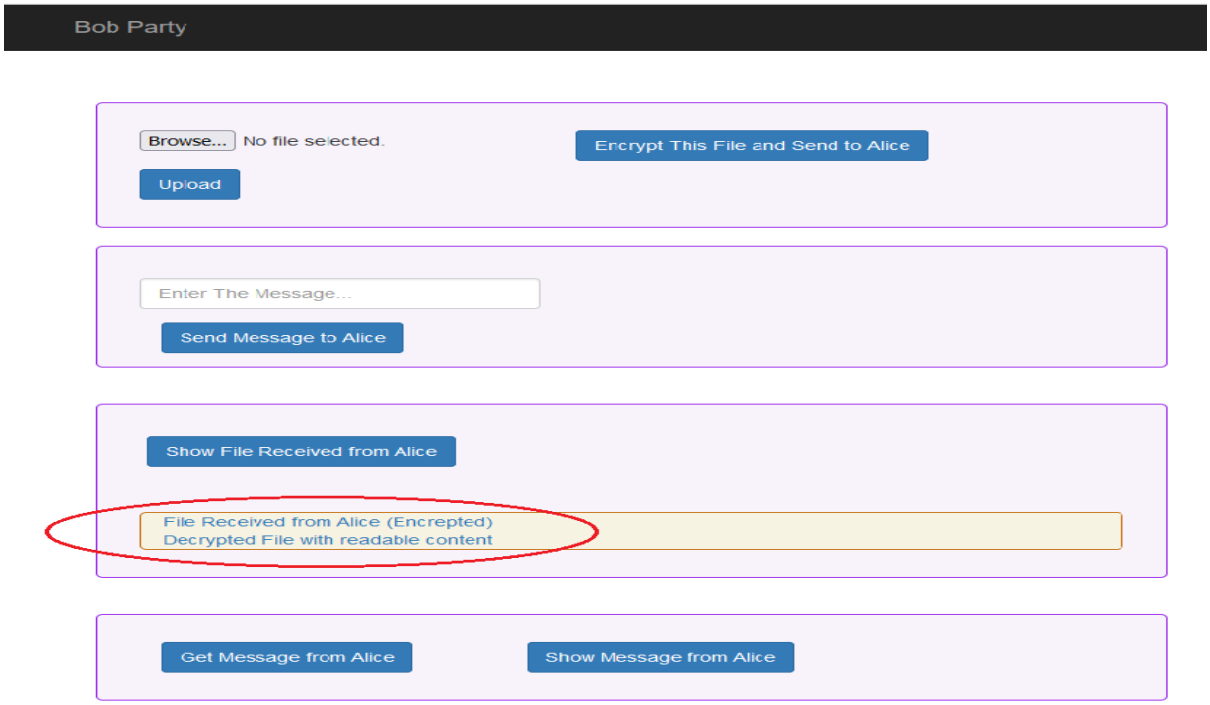

Figure 7

In addition to sending file, Alice and Bob can send messages to each other in a secure way as mentioned above.

3. Test Results:

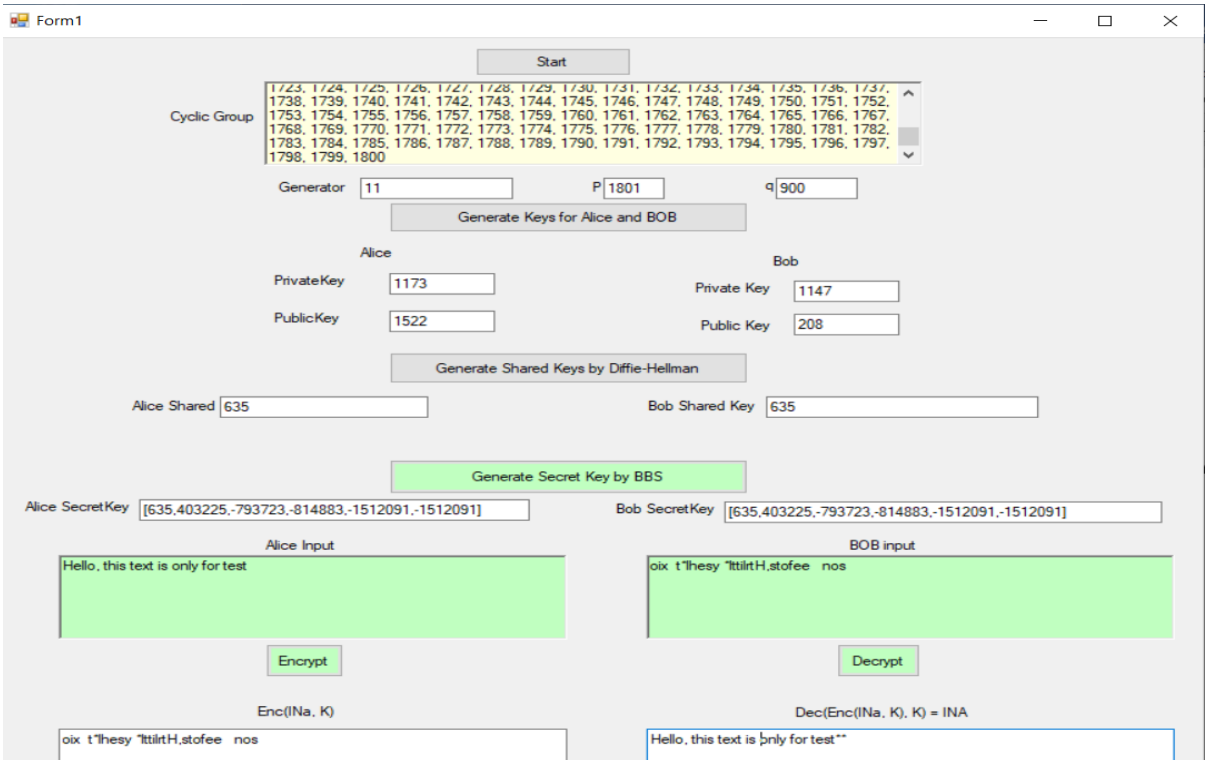

Figure 8

As you see in figure 8, this is the result of part1 of assignment.

Step 1) By clicking button "Start", cyclic group, prime number(p), q, generator of cyclic group(g) will be generated.

Step 2) When we click the button labeled "Generate key for Alice and Bob", Alice and Bob generate their private and public key.

Step 3) When we click the button named "Generate secret key by BBS" Alice and Bob generate their secret key, which will be shown in the textbox.

Step 4) Now Alice can enter an input and encrypt it and show the result in the box which is called Enc (INa, K)

Step 5) If Bob enter Enc (INa, K) as an input, after clicking Decrypt button, Dec (Enc (INa, K), K) = INA will be shown.

In part2, Alice and Bob, will be run on different ports, so they can send text file and messages to each other. Figure 9, is the result of running Alice application on port 44360:

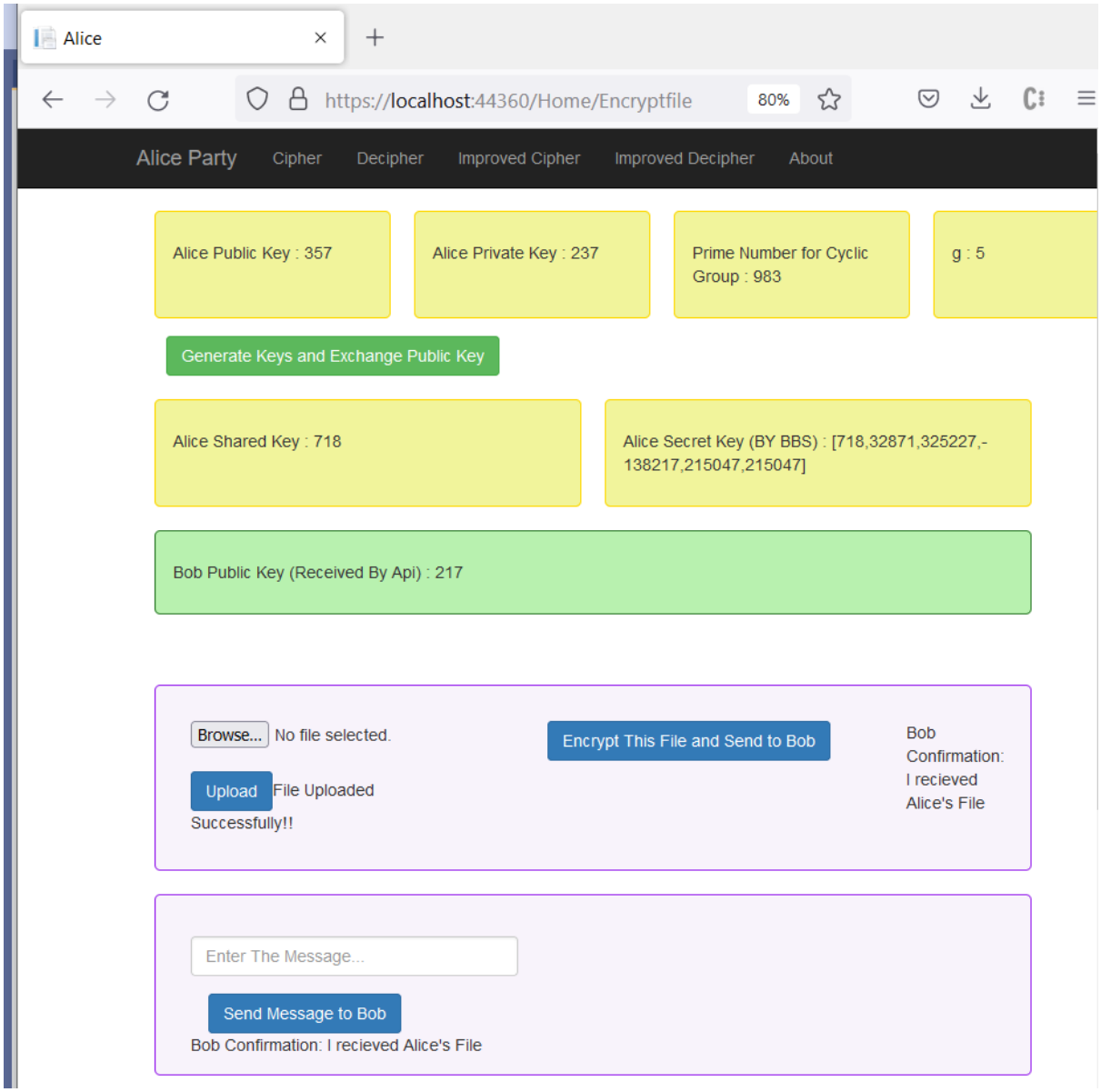

Figure9

And Figure10, is the result of running Bob's application on port 44300.

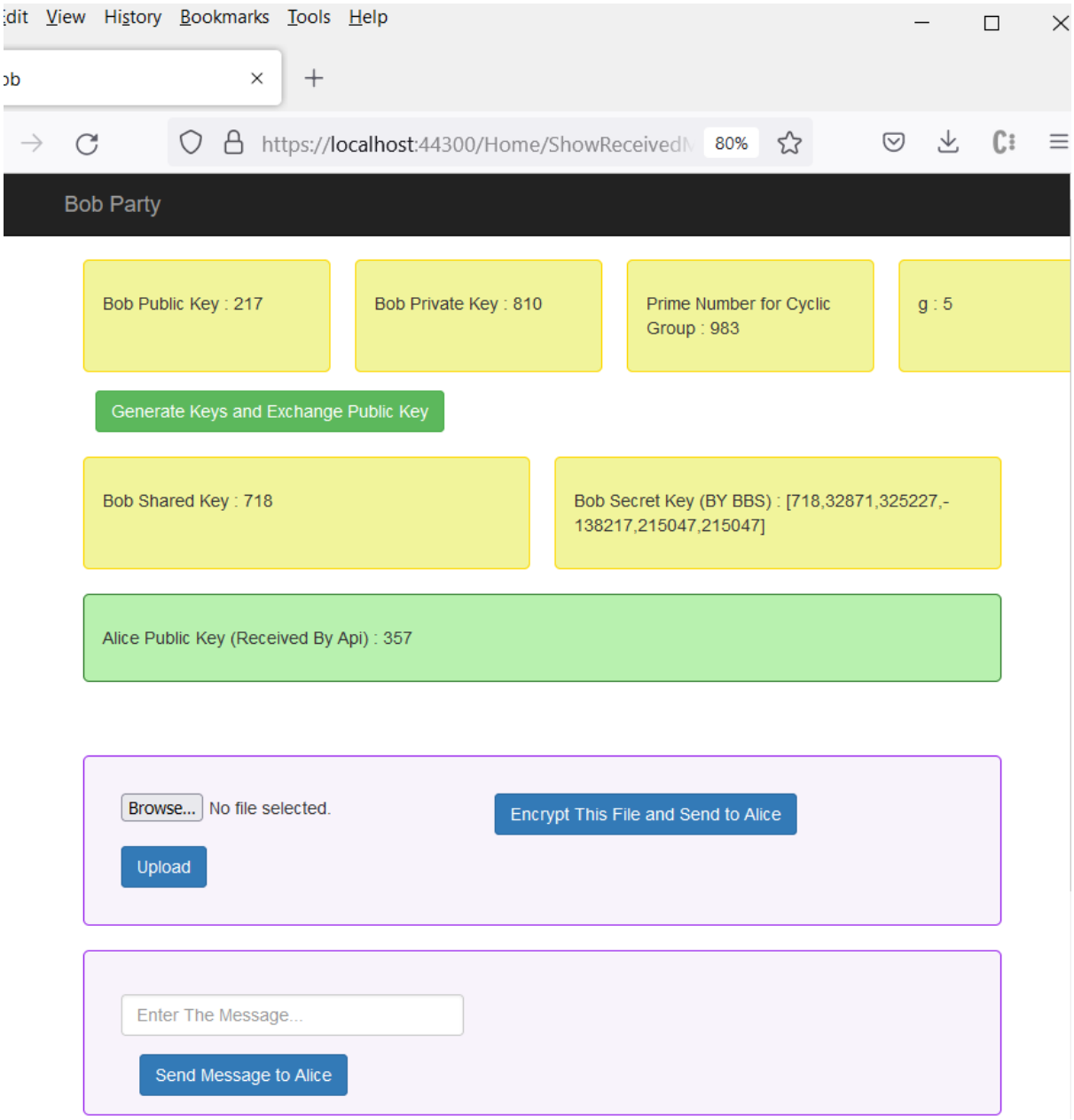

Figure 10

First of all, both parties should click the first button "Generate keys and Exchange public key". the generated keys will be shown in the output. And the most important thing is that we are able to exchange public keys.

Now if Alice browses a text file, upload it and send to Bob, Bob will send her back a confirmation message.

Now, as you can see in the Figure 11, if Bob clicks the button "Show File Received from Alice", it will be possible for him to download the Encrypted file sent by Alice.

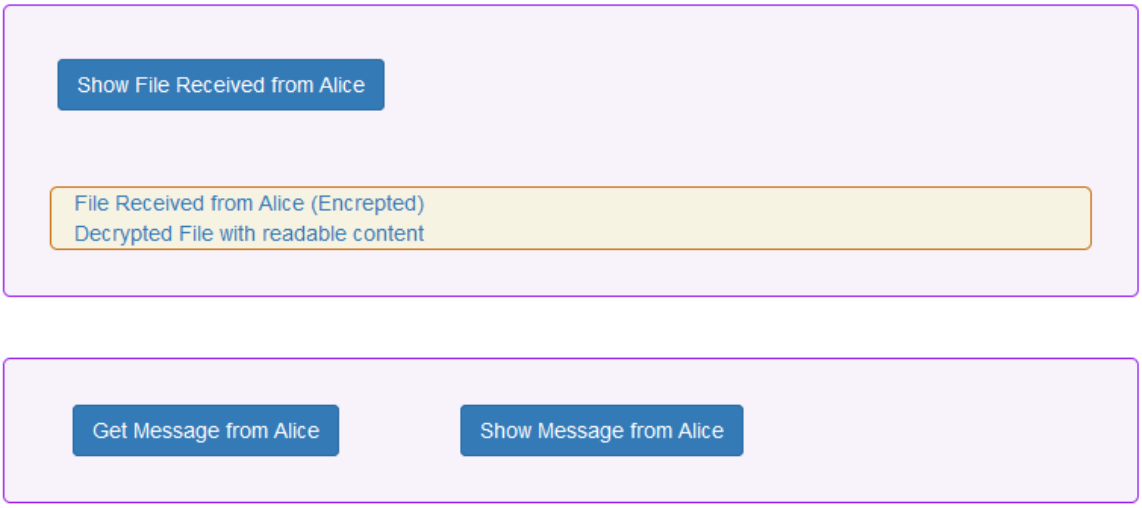

Figure 11

The process can also work for Bob, which means Bob will be able to encrypt a file or message and send it to Alice.

It is worth mentioning that since Alice and Bob are web applications, and we send and receive data by web Api, for seeing all changes in UI, you should click buttons that gets data from server and refresh the page.

4. Discussion:

In part1, when I considered prime number large, the performance of CPU and finally my program decreased. So, I limited the length of prime number to 3 or 4 digits.

In part2, when Alice and Bob want to communicate data, every change will be shown in UI, provided that the page gets refreshed by clicking necessary buttons. For example, when you press the button "Generate Keys and Exchange" in either Alice or Bob, data will be exchanged between them, but in the other page will not be shown until the page gets refreshed by clicking the same button "Generate Keys and Exchange".

5. Conclusion

By implementing Diffie-Helman algorithm we figured out that it is computationally intensive and if we consider prime number very large (which should be very large in real-world programs), it will put many burdens on CPU. Another disadvantage of this exchange protocol is that, it is only practical for symmetric key exchange. Finally, it does not authenticate processes that ask for public key. Therefore, this protocol needs to get improved and more efficient.

References

W. Stallings: "Cryptography and Network Security" 8th Ed., Prentice Hall