```
1  using System;
2  using System.IO;
3  using System.Net;
4  using System.Net.Sockets;
5  using System.Security.Cryptography;
6  using System.Threading;
7  using System.Text.RegularExpressions;
8
9  using static SmokeScreen.Modules.Cryptography;
10 using static SmokeScreen.Modules.Common;
11 using System.Windows.Forms;
12
13 namespace SmokeScreen
14 {
15     public class AsynchronousClient
16     {
17         public AsynchronousClient()
18         {
19
20         }
21
22         private delegate void InvokeDelegate();
23         private TextBox _logBox;
24         private readonly ManualResetEvent getConnected = new ManualResetEvent
             (false);
25         private readonly ManualResetEvent getSented = new ManualResetEvent
             (false);
26         private readonly ManualResetEvent getReceived = new ManualResetEvent
             (false);
27         private string response = string.Empty;
28
29         public string SymmetricKeyExchange(Algorithm alg, out string
             initialPublicKey)
30         {
31             string symmetricKey;
32
33             try
34             {
35                 using (Socket client = new Socket(IpAddress.AddressFamily,
                     SocketType.Stream, ProtocolType.Tcp))
36                 {
37                     client.BeginConnect(new IPEndPoint(IpAddress, PortNumber),
                      new AsyncCallback(ConnectCallback), client);
38                     getConnected.WaitOne();
39
40                     using (ECDiffieHellmanCng initializor = new
                      ECDiffieHellmanCng())
41                     {
42                         initializor.KeyDerivationFunction =
                      ECDiffieHellmanKeyDerivationFunction.Hash;
43                         initializor.HashAlgorithm = CngAlgorithm.Sha256;
44                         initialPublicKey = Convert.ToBase64String
```

```
                    (initializor.PublicKey.ToByteArray());
45                      Send(client, TransactionFormat("exchange",
                    initialPublicKey, "", alg.ToString(), ""));
46                      getSented.WaitOne();
47
48                      Receive(client, false);
49                      getReceived.WaitOne();
50
51                      Log(response);
52
53                      MatchCollection matchCollection =
                    transactionRegex.Matches(response);
54                      if (matchCollection.Count != 0)
55                      {
56                          GroupCollection groupCollection = matchCollection
                    [0].Groups;
57                          byte[] serverPublicKey = Convert.FromBase64String
                    (groupCollection[3].Value);
58                          byte[] symKeyBytes = initializor.DeriveKeyMaterial
                    (CngKey.Import(serverPublicKey,
                    CngKeyBlobFormat.EccPublicBlob));
59                          symmetricKey = Convert.ToBase64String(symKeyBytes);
60                      }
61                      else
62                      {
63                          symmetricKey = "";
64                      }
65
66                      Log($"symmetricKey = {symmetricKey}");
67
68                  }
69                  client.Shutdown(SocketShutdown.Both);
70              }
71
72              getConnected.Reset();
73              getSented.Reset();
74              getReceived.Reset();
75
76              return symmetricKey;
77          }
78          catch (Exception exception)
79          {
80              Log(exception.Message);
81              initialPublicKey = string.Empty;
82              return string.Empty;
83          }
84      }
85
86      public int CreateAccount(Algorithm algorithm, string symmetricKey, string
            publicKey, string username, string password)
87      {
88          int truth = -1;
```

```
89              try
90              {
91                  using (Socket client = new Socket(IpAddress.AddressFamily,
                      SocketType.Stream, ProtocolType.Tcp))
92                  {
93                      client.BeginConnect(new IPEndPoint(IpAddress, PortNumber),
                        new AsyncCallback(ConnectCallback), client);
94                      getConnected.WaitOne();
95
96                      string content = CreateAccountFormat(algorithm, symmetricKey,
                        username, password, out string IV);
97
98                      Send(client, TransactionFormat("createAccount", publicKey,
                        IV, algorithm.ToString(), content));
99                      getSented.WaitOne();
100
101                     Receive(client, true, symmetricKey);
102                     getReceived.WaitOne();
103                     Log(response);
104
105                     client.Shutdown(SocketShutdown.Both);
106                 }
107
108                 truth = IsAccountCreated(response);
109
110                 getConnected.Reset();
111                 getSented.Reset();
112                 getReceived.Reset();
113             }
114             catch (Exception exception)
115             {
116                 Log(exception.Message);
117                 return truth;
118             }
119
120             return truth;
121         }
122
123         public int Authenicate(Algorithm algorithm, string symmetricKey, string
              publicKey, string username, string password)
124         {
125             int truth = -1;
126             try
127             {
128                 using (Socket client = new Socket(IpAddress.AddressFamily,
                      SocketType.Stream, ProtocolType.Tcp))
129                 {
130                     client.BeginConnect(new IPEndPoint(IpAddress, PortNumber),
                        new AsyncCallback(ConnectCallback), client);
131                     getConnected.WaitOne();
132
133                     string content = AuthenicateFormat(algorithm, symmetricKey,
```

```
                    username, password, out string IV);
134
135                 Send(client, TransactionFormat("authenticate", publicKey, IV,
                      algorithm.ToString(), content));
136                 getSented.WaitOne();
137
138                 Receive(client, true, symmetricKey);
139                 getReceived.WaitOne();
140
141                 Log(GetMessage(response));
142
143                 client.Shutdown(SocketShutdown.Both);
144             }
145
146             truth = IsAuthenticated(response);
147
148             getConnected.Reset();
149             getSented.Reset();
150             getReceived.Reset();
151         }
152         catch (Exception exception)
153         {
154             Log(exception.Message);
155             return truth;
156         }
157
158         return truth;
159     }
160
161     public void SendMessage(Algorithm algorithm, string symmetricKey, string
          publicKey, string message)
162     {
163         string truth = string.Empty;
164         try
165         {
166             using (Socket client = new Socket(IpAddress.AddressFamily,
                      SocketType.Stream, ProtocolType.Tcp))
167             {
168                 client.BeginConnect(new IPEndPoint(IpAddress, PortNumber),
                      new AsyncCallback(ConnectCallback), client);
169                 getConnected.WaitOne();
170
171                 string content = MessageFormat(algorithm, symmetricKey,
                      message, out string IV);
172
173                 Send(client, TransactionFormat("message", publicKey, IV,
                      algorithm.ToString(), content));
174                 getSented.WaitOne();
175
176                 Receive(client, true, symmetricKey);
177                 getReceived.WaitOne();
178
```

```
179                     Log(GetMessage(response));
180
181                     client.Shutdown(SocketShutdown.Both);
182                 }
183                 getConnected.Reset();
184                 getSented.Reset();
185                 getReceived.Reset();
186             }
187             catch (Exception exception)
188             {
189                 Log(exception.Message);
190             }
191         }
192
193         public void SetLogBox(TextBox logBox)
194         {
195             _logBox = logBox;
196         }
197
198         public void SendFile(Algorithm algorithm, Files file, string
              symmetricKey, string publicKey)
199         {
200             string logMessage;
201
202             try
203             {
204                 using (Socket client = new Socket(IpAddress.AddressFamily,
                      SocketType.Stream, ProtocolType.Tcp))
205                 {
206                     client.BeginConnect(new IPEndPoint(IpAddress, PortNumber),
                       new AsyncCallback(ConnectCallback), client);
207                     getConnected.WaitOne();
208
209                     string fileContent = ReadFile(file);
210
211                     string content = MessageFormat(algorithm, symmetricKey,
                       fileContent, out string IV);
212
213                     Log(content);
214
215                     Send(client, TransactionFormat($"sendfile{file.ToString()}",
                       publicKey, IV, algorithm.ToString(), content));
216                     getSented.WaitOne();
217
218                     Receive(client, true, symmetricKey);
219                     getReceived.WaitOne();
220
221                     MatchCollection matchCollection = messageRegex.Matches
                       (response);
222                     if (matchCollection.Count != 0)
223                     {
224                         GroupCollection groupCollection = matchCollection
```

```
                    [0].Groups;

                        string fileInput = groupCollection[1].Value;

                        logMessage = $"Recieved file {file.ToString()}.txt from client";

                        SaveFile(file, fileInput);
                    }
                    else
                    {
                        logMessage = "Unable to Access the selected file.";
                    }

                    Log($"{logMessage} {GetMessage(response)}");

                    client.Shutdown(SocketShutdown.Both);
                }
                getConnected.Reset();
                getSented.Reset();
                getReceived.Reset();
            }
            catch (Exception exception)
            {
                Log(exception.Message);
            }
        }

        public void RequestFile(Algorithm algorithm, Files file, string symmetricKey, string publicKey)
        {
            string logMessage;
            try
            {
                using (Socket client = new Socket(IpAddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp))
                {
                    client.BeginConnect(new IPEndPoint(IpAddress, PortNumber), new AsyncCallback(ConnectCallback), client);
                    getConnected.WaitOne();

                    string fileRequest = MessageFormat(algorithm, symmetricKey, file.ToString(), out string IV);

                    Send(client, TransactionFormat($"requestfile", publicKey, IV, algorithm.ToString(), fileRequest));
                    getSented.WaitOne();

                    Receive(client, true, symmetricKey);
                    getReceived.WaitOne();

                    MatchCollection matchCollection = messageRegex.Matches
```

```
                    (response);
270                 if (matchCollection.Count != 0)
271                 {
272                     GroupCollection groupCollection = matchCollection
                    [0].Groups;
273
274                     string fileInput = groupCollection[1].Value;
275
276                     logMessage = $"Recieved file {file.ToString()} from
                    client";
277
278                     SaveFile(file, fileInput);
279                 }
280                 else
281                 {
282                     logMessage = "Unable to Access the selected file.";
283                 }
284
285                 Log($"{logMessage} {GetMessage(response)}");
286
287                 client.Shutdown(SocketShutdown.Both);
288             }
289             getConnected.Reset();
290             getSented.Reset();
291             getReceived.Reset();
292         }
293         catch (Exception exception)
294         {
295             Log(exception.Message);
296         }
297     }
298
299     private void ConnectCallback(IAsyncResult asyncResult)
300     {
301         try
302         {
303             Socket client = (Socket)asyncResult.AsyncState;
304             client.EndConnect(asyncResult);
305             Log($"Socket connected to {client.RemoteEndPoint.ToString()}");
306             getConnected.Set();
307         }
308         catch (Exception exception)
309         {
310             Log(exception.Message);
311             getConnected.Set();
312         }
313     }
314
315     private void Send(Socket worker, string data)
316     {
317         byte[] byteArray = encoding.GetBytes(data);
318         worker.BeginSend(byteArray, 0, byteArray.Length, 0, new AsyncCallback
```

```
                (SendCallback), worker);
319         }
320
321         private void SendCallback(IAsyncResult asyncResult)
322         {
323             try
324             {
325                 Socket worker = (Socket)asyncResult.AsyncState;
326                 int byteCount = worker.EndSend(asyncResult);
327                 Console.WriteLine($"Recieved {byteCount.ToString()} bytes from
                      server.");
328                 getSented.Set();
329             }
330             catch (Exception exception)
331             {
332                 Log(exception.Message);
333                 getSented.Set();
334             }
335         }
336
337         private void Receive(Socket worker, bool decrypt, string symmetricKey =
              "")
338         {
339             try
340             {
341                 StateObject stateObject = new StateObject { workSocket = worker,
                      decrypt = decrypt, symmetricKey = symmetricKey };
342                 worker.BeginReceive(stateObject.buffer, 0,
                      StateObject.BufferSize, 0, new AsyncCallback(ReceiveCallback),
                      stateObject);
343             }
344             catch (Exception exception)
345             {
346                 Log(exception.Message);
347             }
348         }
349
350         private void ReceiveCallback(IAsyncResult aysncResult)
351         {
352             try
353             {
354                 StateObject stateObject = (StateObject)aysncResult.AsyncState;
355                 Socket worker = stateObject.workSocket;
356
357                 int byteCount = worker.EndReceive(aysncResult);
358
359                 if (byteCount > 0)
360                 {
361                     stateObject.stringBuilder.Append(encoding.GetString
                          (stateObject.buffer, 0, byteCount));
362                     worker.BeginReceive(stateObject.buffer, 0,
                          StateObject.BufferSize, 0, new AsyncCallback
```

```
                    (ReceiveCallback), stateObject);
363                 }
364                 else
365                 {
366                     if (stateObject.stringBuilder.Length > 1)
367                     {
368                         if (stateObject.decrypt)
369                         {
370                             MatchCollection matchCollection =
                    transactionRegex.Matches(stateObject.stringBuilder.ToString
                    ());
371                             if (matchCollection.Count != 0)
372                             {
373                                 GroupCollection groupCollection = matchCollection
                    [0].Groups;
374
375                                 string token = groupCollection[3].Value;
376                                 string algorithm = groupCollection[4].Value;
377                                 string content = groupCollection[5].Value;
378
379                                 response = Decrypt(ConvertStringToAlgorithm
                    (algorithm), stateObject.symmetricKey, token, content);
380                             }
381                             else
382                             {
383                                 response = stateObject.stringBuilder.ToString();
384                             }
385                         }
386                         else
387                         {
388                             response = stateObject.stringBuilder.ToString();
389                         }
390                     }
391                     getReceived.Set();
392                 }
393             }
394             catch (Exception exception)
395             {
396                 Log(exception.Message);
397                 getReceived.Set();
398             }
399         }
400
401         public Transaction GetTransaction(string data)
402         {
403             MatchCollection matchCollection = transactionRegex.Matches(data);
404
405             if (matchCollection.Count != 0)
406             {
407                 GroupCollection groupCollection = matchCollection[0].Groups;
408                 return new Transaction(groupCollection[1].Value, groupCollection
                      [2].Value, "", groupCollection[3].Value);
```

```
409             }
410
411             return new Transaction();
412         }
413
414         public class Transaction
415         {
416             public Transaction()
417             {
418                 Type = string.Empty;
419                 Key = string.Empty;
420                 Token = string.Empty;
421                 Content = string.Empty;
422             }
423
424             public Transaction(string type, string key, string token, string
                  content)
425             {
426                 Type = type;
427                 Key = key;
428                 Token = token;
429                 Content = content;
430             }
431
432             public string Type { get; }
433
434             public string Key { get; }
435
436             public string Token { get; }
437
438             public string Content { get; }
439
440         }
441
442         private int IsAuthenticated(string content)
443         {
444             MatchCollection matchCollection = messageRegex.Matches(content);
445
446             if (matchCollection.Count != 0)
447             {
448                 GroupCollection groupCollection = matchCollection[0].Groups;
449                 string reply = groupCollection[1].Value;
450                 if (reply == "Authorized")
451                     return 1;
452                 else if (reply == "InvalidUsername")
453                     return 2;
454                 else if (reply == "InvalidPassword")
455                     return 3;
456                 else if (reply == "InvalidFormatAuthorize")
457                     return 4;
458             }
459             return 0;
```

```
460         }
461
462         private int IsAccountCreated(string content)
463         {
464             MatchCollection matchCollection = messageRegex.Matches(content);
465
466             if (matchCollection.Count != 0)
467             {
468                 GroupCollection groupCollection = matchCollection[0].Groups;
469                 string reply = groupCollection[1].Value;
470
471                 if (reply == "CreatedAccount")
472                     return 1;
473                 else if (reply == "InvalidUsernameLength")
474                     return 2;
475                 else if (reply == "InvalidUsernameExists")
476                     return 3;
477                 else if (reply == "FailedAccount")
478                     return 4;
479                 else if (reply == "InvalidFormatCreateAccount")
480                     return 5;
481             }
482             return 0;
483         }
484
485         private string GetMessage(string content)
486         {
487             MatchCollection matchCollection = messageRegex.Matches(content);
488
489             if (matchCollection.Count != 0)
490             {
491                 GroupCollection groupCollection = matchCollection[0].Groups;
492                 return groupCollection[1].Value;
493             }
494             else
495             {
496                 return "Unable to Read Message. This could be due to tampering
                      across the web or exceptions.";
497             }
498         }
499
500         private string AuthenicateFormat(Algorithm algorithm, string key, string
              username, string password, out string IV)
501         {
502             return Encrypt(algorithm, key, "{{Username='" + username + "'}
                  {Password='" + password + "'}}", out IV);
503         }
504
505         private string CreateAccountFormat(Algorithm algorithm, string key,
              string username, string password, out string IV)
506         {
507             return Encrypt(algorithm, key, "{{Username='" + username + "'}
```

```
                {Password='" + password + "'}}", out IV);
508         }
509
510         private string MessageFormat(Algorithm algorithm, string symmetricKey,
              string message, out string IV)
511         {
512             return Encrypt(algorithm, symmetricKey, "{{Message = '" + message +
                  "'}}", out IV);
513         }
514
515         private string ReadFile(Files file)
516         {
517             string fileContent;
518             if (file == Files.Sales)
519             {
520                 fileContent = File.ReadAllText(@"..\..\Static\FilesOut
                      \Sales.txt");
521             }
522             else if (file == Files.Maps)
523             {
524                 fileContent = File.ReadAllText(@"..\..\Static\FilesOut
                      \Maps.txt");
525             }
526             else if (file == Files.Budget)
527             {
528                 fileContent = File.ReadAllText(@"..\..\Static\FilesOut
                      \Budget.txt");
529             }
530             else
531             {
532                 fileContent = File.ReadAllText(@"..\..\Static\FilesOut
                      \Error.txt");
533             }
534             return fileContent;
535         }
536
537         private void SaveFile(Files file, string data)
538         {
539             if (file == Files.Sales)
540             {
541                 File.WriteAllText(@"..\..\Static\FilesIn\Sales.txt", data);
542             }
543             else if (file == Files.Maps)
544             {
545                 File.WriteAllText(@"..\..\Static\FilesIn\Maps.txt", data);
546             }
547             else if (file == Files.Budget)
548             {
549                 File.WriteAllText(@"..\..\Static\FilesIn\Budget.txt", data);
550             }
551             else
552             {
```

```
553                 File.WriteAllText(@"..\..\Static\FilesIn\Error.txt", data);
554             }
555         }
556
557         private void Log(string text)
558         {
559             if (_logBox != null)
560             {
561                 _logBox.BeginInvoke(new InvokeDelegate(() => _logBox.Text =
                      text));
562             }
563             Console.WriteLine(text);
564         }
565
566     }
567
568 }
569
```