```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Text.RegularExpressions;
using static SmokeScreen.Modules.Cryptography;

namespace SmokeScreen.Modules
{
    public class Common
    {
        public static readonly int PortNumber = 11000;
        public static readonly int MaxPendingConnections = 100;
        public static readonly IPAddress IpAddress = IPAddress.Parse
          ("127.0.0.1");
        public static readonly Encoding encoding = Encoding.Unicode;

        // Group 0 is all, Group 1 is type, Group 2 is key, Group 3 is token,
          Group 4 is content
        public static readonly Regex transactionRegex1 = new Regex(
            @"<transaction\s+type='([\w\s]*)'\s+key='(.*)'\s+token='(.*)'\s*>(.*)
              <\/transaction>",
            RegexOptions.Compiled | RegexOptions.IgnoreCase);

        // Group 0 is all, Group 1 is type, Group 2 is key, Group 3 is token,
          Group 4 is alg, Group 5 is content
        public static readonly Regex transactionRegex = new Regex(
            @"<transaction\s+type='([\w\s]*)'\s+key='(.*)'\s+token='(.*)'\s
              +alg='([\w\s]*)'\s*>(.*)<\/transaction>",
            RegexOptions.Compiled | RegexOptions.IgnoreCase);

        // Group 0 is all, Group 1 is username, Group 2 is password
        public static readonly Regex createAccountRegex = new Regex(
            @"\s*\{\s*\{\s*Username\s*=\s*'(.*)'\s*.*\}\s*\{\s*Password\s*=
              \s*'(.*)'\s*\}\s*\}",
            RegexOptions.Compiled | RegexOptions.IgnoreCase);

        // Group 0 is all, Group 1 is username, Group 2 is password
        public static readonly Regex authenicationRegex = new Regex(
            @"\s*\{\s*\{\s*Username\s*=\s*'(.*)'\s*.*\}\s*\{\s*Password\s*=
              \s*'(.*)'\s*\}\s*\}",
            RegexOptions.Compiled | RegexOptions.IgnoreCase);

        // Group 0 is all, Group 1 is message
        public static readonly Regex messageRegex = new Regex(
            @"\s*\{\s*\{\s*Message\s*=\s*'(.*)'\s*.*\}\s*\}",
            RegexOptions.Compiled | RegexOptions.IgnoreCase);

        /// <summary>
        /// Client Specific Data Class
        /// </summary>
        public class ClientObject
```

```
46          {
47              public ClientObject(string publicKey, string symmetricKey)
48              {
49                  PublicKey = publicKey;
50                  SymmetricKey = symmetricKey;
51              }
52
53              public bool Authenticated = false;
54
55              public string PublicKey { get; }
56
57              public string SymmetricKey { get; }
58
59          }
60
61          public class StateObject
62          {
63              public const int BufferSize = 256;
64
65              public byte[] buffer = new byte[BufferSize];
66
67              public Socket workSocket = null;
68
69              public StringBuilder stringBuilder = new StringBuilder();
70
71              public bool decrypt = false;
72
73              public string symmetricKey = string.Empty;
74          }
75
76          public static string TransactionFormat(string type, string key = "",
              string token = "", string alg = "", string content = "")
77          {
78              return $"<transaction type='{type}' key='{key}' token='{token}'
                  alg='{alg}'>{content}</transaction>";
79          }
80
81          public static string Decrypt(Algorithm algorithm, string symmetricKey,
              string token, string content)
82          {
83              if (algorithm == Algorithm.AES)
84              {
85                  content = AES.Decrypt(symmetricKey, content, token);
86              }
87              else if (algorithm == Algorithm.RIJ)
88              {
89                  content = RIJ.Decrypt(symmetricKey, content, token);
90              }
91              else if (algorithm == Algorithm.DES)
92              {
93                  content = TDES.Decrypt(symmetricKey, content, token);
94              }
```

```
 95              else
 96              {
 97                  content = string.Empty;
 98              }
 99              return content;
100          }
101
102          public static string Encrypt(Algorithm algorithm, string symmetricKey,
               string message, out string IV)
103          {
104              if (algorithm == Algorithm.AES)
105              {
106                  return AES.Encrypt(symmetricKey, message, out IV);
107              }
108              else if (algorithm == Algorithm.RIJ)
109              {
110                  return RIJ.Encrypt(symmetricKey, message, out IV);
111              }
112              else if (algorithm == Algorithm.DES)
113              {
114                  return TDES.Encrypt(symmetricKey, message, out IV);
115              }
116              else
117              {
118                  IV = string.Empty;
119                  return "Algorithm Issue";
120              }
121          }
122
123          public static Algorithm ConvertStringToAlgorithm(string algorithm)
124          {
125              try
126              {
127                  Enum.TryParse(algorithm, out Algorithm selectedAlgorithm);
128                  return selectedAlgorithm;
129              }
130              catch
131              {
132                  return Algorithm.AES;
133              }
134          }
135
136          public enum Files
137          {
138              Sales = 0,
139              Maps = 1,
140              Budget = 2
141          }
142
143      }
144  }
145
```