```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

using static SmokeScreen.Modules.Logging;
using static SmokeScreen.Modules.Common;

namespace SmokeScreen.Modules
{
    public static class Cryptography
    {
        //Reference: Micrsoft Docs System.Security.Cryptography -> AesCryptoServiceProvider Class
        public static class AES
        {
            public static string Encrypt(string symmetricKey, string message, out string IV)
            {
                using (Aes aes = new AesCryptoServiceProvider())
                {
                    //Accepts 16, 24, or 32 byte Keysize - [Sha 256 hash is 32 bytes]
                    aes.Key = Convert.FromBase64String(symmetricKey);
                    IV = Convert.ToBase64String(aes.IV);

                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, aes.CreateEncryptor(), CryptoStreamMode.Write))
                        {
                            byte[] plaintext = encoding.GetBytes(message);
                            cryptoStream.Write(plaintext, 0, plaintext.Length);
                        }
                        message = Convert.ToBase64String(memoryStream.ToArray());
                    }
                }
                return message;
            }

            public static string Decrypt(string symmetricKey, string message, string IV)
            {
                using (Aes aes = new AesCryptoServiceProvider())
                {
                    aes.Key = Convert.FromBase64String(symmetricKey);
                    aes.IV = Convert.FromBase64String(IV);

                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        using (CryptoStream cryptoStream = new CryptoStream
                            (memoryStream, aes.CreateDecryptor(), 
                            CryptoStreamMode.Write))
                        {
                            byte[] cipherText = Convert.FromBase64String
                            (message);
                            cryptoStream.Write(cipherText, 0, cipherText.Length);
                        }
                        message = encoding.GetString(memoryStream.ToArray());
                    }
                }
                return message;
            }
        }

        //Reference: Micrsoft Docs System.Security.Cryptography -> 
          RijndaelManaged Class
        public static class RIJ
        {
            public static string Encrypt(string symmetricKey, string message, out 
               string IV)
            {
                using (RijndaelManaged rij = new RijndaelManaged())
                {
                    //Converts to 24 Bytes
                    byte[] symKey = Convert.FromBase64String(symmetricKey);
                    Array.Resize(ref symKey, 8);

                    rij.Key = Convert.FromBase64String(symmetricKey);
                    IV = Convert.ToBase64String(rij.IV);

                    rij.Padding = PaddingMode.Zeros;

                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        using (CryptoStream cryptoStream = new CryptoStream
                            (memoryStream, rij.CreateEncryptor(), 
                            CryptoStreamMode.Write))
                        {
                            byte[] plainText = encoding.GetBytes(message);
                            cryptoStream.Write(plainText, 0, plainText.Length);
                        }
                        message = Convert.ToBase64String(memoryStream.ToArray());
                    }
                }
                return message;
            }

            public static string Decrypt(string symmetricKey, string message, 
              string IV)
            {
                using (RijndaelManaged rij = new RijndaelManaged())
                {
```

```csharp
                    rij.Key = Convert.FromBase64String(symmetricKey);
                    rij.IV = Convert.FromBase64String(IV);

                    rij.Padding = PaddingMode.Zeros;

                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, rij.CreateDecryptor(), CryptoStreamMode.Write))
                        {
                            byte[] cipherText = Convert.FromBase64String(message);
                            cryptoStream.Write(cipherText, 0, cipherText.Length);
                        }
                        message = encoding.GetString(memoryStream.ToArray());
                    }
                }
                return message;
            }
        }

        //Reference: Micrsoft Docs System.Security.Cryptography -> RC2CryptoServiceProvider Class
        public static class RC2
        {
            public static string Encrypt(string symmetricKey, string message, out string IV)
            {
                using (RC2CryptoServiceProvider arc2 = new RC2CryptoServiceProvider())
                {
                    //Converts to 8 Bytes
                    byte[] symKey = Convert.FromBase64String(symmetricKey);
                    Array.Resize(ref symKey, 8);

                    //Accepts 8 byte Keysize
                    arc2.Key = symKey;
                    IV = Convert.ToBase64String(arc2.IV);

                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, arc2.CreateEncryptor(), CryptoStreamMode.Write))
                        {
                            byte[] plainText = encoding.GetBytes(message);
                            cryptoStream.Write(plainText, 0, plainText.Length);
                        }
                        message = Convert.ToBase64String(memoryStream.ToArray());
                    }
                }
                return message;
            }

            public static string Decrypt(string symmetricKey, string message, string IV)
            {
                using (RijndaelManaged rij = new RijndaelManaged())
                {
                    rij.Key = Convert.FromBase64String(symmetricKey);
                    rij.IV = Convert.FromBase64String(IV);

                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, rij.CreateDecryptor(), CryptoStreamMode.Read))
                        {
                            byte[] cipherText = Convert.FromBase64String(message);
                            cryptoStream.Write(cipherText, 0, cipherText.Length);
                        }
                        message = encoding.GetString(memoryStream.ToArray());
                    }
                }
                return message;
            }
        }

        public static class TDES
        {
            public static string Encrypt(string symmetricKey, string message, out string IV)
            {
                using (TripleDESCryptoServiceProvider des = new TripleDESCryptoServiceProvider())
                {
                    //Converts to 24 Bytes
                    byte[] symKey = Convert.FromBase64String(symmetricKey);
                    Array.Resize(ref symKey, 24);

                    //Accepts 16, 24 byte Keysizes
                    des.Key = symKey;
                    IV = Convert.ToBase64String(des.IV);

                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, des.CreateEncryptor(), CryptoStreamMode.Write))
                        {
                            byte[] plainText = encoding.GetBytes(message);
                            cryptoStream.Write(plainText, 0, plainText.Length);
                        }
                        message = Convert.ToBase64String(memoryStream.ToArray());
                    }
                }
                return message;
            }

            public static string Decrypt(string symmetricKey, string message, string IV)
            {
                using (TripleDESCryptoServiceProvider des = new TripleDESCryptoServiceProvider())
                {
                    //Converts to 24 Bytes
                    byte[] symKey = Convert.FromBase64String(symmetricKey);
                    Array.Resize(ref symKey, 24);

                    //24 Byte Key
                    des.Key = symKey;
                    des.IV = Convert.FromBase64String(IV);

                    using (MemoryStream memoryStream = new MemoryStream())
                    {
                        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, des.CreateDecryptor(), CryptoStreamMode.Write))
                        {
                            byte[] cipherText = Convert.FromBase64String(message);
                            cryptoStream.Write(cipherText, 0, cipherText.Length);
                        }
                        message = encoding.GetString(memoryStream.ToArray());
                    }
                }
                return message;
            }
        }

        /// <summary>
        /// Used https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.sha256managed for
        /// Example of How to Implement a Sha256 Hash with C#
        /// </summary>
        public static class Sha256Hash
        {
            public static string Generate(string password)
            {
                if (string.IsNullOrEmpty(password))
                {
                    return string.Empty;
                }
                else
                {
                    byte[] hash;

                    using (SHA256 generator = SHA256.Create())
                    {
                        hash = generator.ComputeHash(encoding.GetBytes(password));
                    }

                    return Convert.ToBase64String(hash);
                }
            }

            public static bool Compare(string password1, string password2)
            {
                if (string.IsNullOrEmpty(password1) || string.IsNullOrEmpty(password2))
                {
                    return false;
                }

                if (password1 == password2)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }

        public enum Algorithm
        {
            AES = 0,
            DES = 1,
            RIJ = 2
        }

    }
}
```