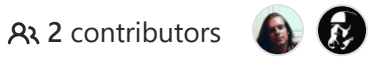


```
1  using System;
2  using Renci.SshNet.Abstractions;
3  using Renci.SshNet.Common;
4  using Renci.SshNet.Messages.Transport;
5  using Renci.SshNet.Security.Chaos.NaCl;
6  using Renci.SshNet.Security.Chaos.NaCl.Internal.Ed25519Ref10;
7
8  namespace Renci.SshNet.Security
9  {
10     internal class KeyExchangeECCurve25519 : KeyExchangeEC
11     {
12         private byte[] _privateKey;
13
14         /// <summary>
15         /// Gets algorithm name.
16         /// </summary>
17         public override string Name
18         {
19             get { return "curve25519-sha256"; }
20         }
21
22         /// <summary>
23         /// Gets the size, in bits, of the computed hash code.
24         /// </summary>
25         /// <value>
26         /// The size, in bits, of the computed hash code.
27         /// </value>
28         protected override int HashSize
29         {
```

```
30              get { return 256; }
31          }
32
33          /// <summary>
34          /// Starts key exchange algorithm
35          /// </summary>
36          /// <param name="session">The session.</param>
37          /// <param name="message">Key exchange init message.</param>
38          public override void Start(Session session, KeyExchangeInitMessage message)
39          {
40              base.Start(session, message);
41
42              Session.RegisterMessage("SSH_MSG_KEX_ECDH_REPLY");
43
44              Session.KeyExchangeEcdhReplyMessageReceived += Session_KeyExchangeEcdhReplyMessageRece
45
46              var basepoint = new byte[MontgomeryCurve25519.PublicKeySizeInBytes];
47              basepoint[0] = 9;
48
49              var rnd = new Random();
50              _privateKey = new byte[MontgomeryCurve25519.PrivateKeySizeInBytes];
51              rnd.NextBytes(_privateKey);
52
53              _clientExchangeValue = new byte[MontgomeryCurve25519.PublicKeySizeInBytes];
54              MontgomeryOperations.scalarmult(_clientExchangeValue, 0, _privateKey, 0, basepoint, 0)
55
56              SendMessage(new KeyExchangeEcdhInitMessage(_clientExchangeValue));
57          }
58
59          /// <summary>
60          /// Finishes key exchange algorithm.
61          /// </summary>
62          public override void Finish()
63          {
64              base.Finish();
65
66              Session.KeyExchangeEcdhReplyMessageReceived -= Session_KeyExchangeEcdhReplyMessageRece
67          }
68
69          /// <summary>
70          /// Hashes the specified data bytes.
71          /// </summary>
72          /// <param name="hashData">The hash data.</param>
73          /// <returns>
74          /// Hashed bytes
75          /// </returns>
76          protected override byte[] Hash(byte[] hashData)
77          {
78              using (var sha256 = CryptoAbstraction.CreateSHA256())
```

```
79              {
80                  return sha256.ComputeHash(hashData, 0, hashData.Length);
81              }
82          }
83
84          private void Session_KeyExchangeEcdhReplyMessageReceived(object sender, MessageEventArgs<K
85          {
86              var message = e.Message;
87
88              //  Unregister message once received
89              Session.UnRegisterMessage("SSH_MSG_KEX_ECDH_REPLY");
90
91              HandleServerEcdhReply(message.KS, message.QS, message.Signature);
92
93              //  When SSH_MSG_KEXDH_REPLY received key exchange is completed
94              Finish();
95          }
96
97          /// <summary>
98          /// Handles the server DH reply message.
99          /// </summary>
100         /// <param name="hostKey">The host key.</param>
101         /// <param name="serverExchangeValue">The server exchange value.</param>
102         /// <param name="signature">The signature.</param>
103         private void HandleServerEcdhReply(byte[] hostKey, byte[] serverExchangeValue, byte[] sign
104         {
105             _serverExchangeValue = serverExchangeValue;
106             _hostKey = hostKey;
107             _signature = signature;
108
109             var sharedKey = new byte[MontgomeryCurve25519.PublicKeySizeInBytes];
110             MontgomeryOperations.scalarmult(sharedKey, 0, _privateKey, 0, serverExchangeValue, 0);
111             SharedKey = sharedKey.ToBigInteger2().ToByteArray().Reverse();
112         }
113     }
114 }
```