Custom Single Sign-on(SSO) implementation between SharePoint and Non SharePoint Portal

Every organization will have a lot of end user facing portals, some of which may even be partner sites. The users of these portals need to login into these portals to make complete use of the various features and functionalities of the portal, including making online purchases. It is painstaking for users to supply their credentials for every portal, especially if the user wants to switch from one portal to another. This is felt especially when switching to partner sites or other portals of the same organization, Here I am just suggesting high level approach , if need exact solution, please reach to me and I would be more than happy to provide all source files

SharePoint does support the Single Sing On feature with Claim based and Federation Module But you ask me if you have requirement where any non-SharePoint portal can have Single Sign-on then it is not supported or you have do lots of analysis (there is no easy way to do it) and spent a time to share claim between Portal, here I am describing the custom and Simple way to achieve Single sign on , Custom SSO Auth provide all features included

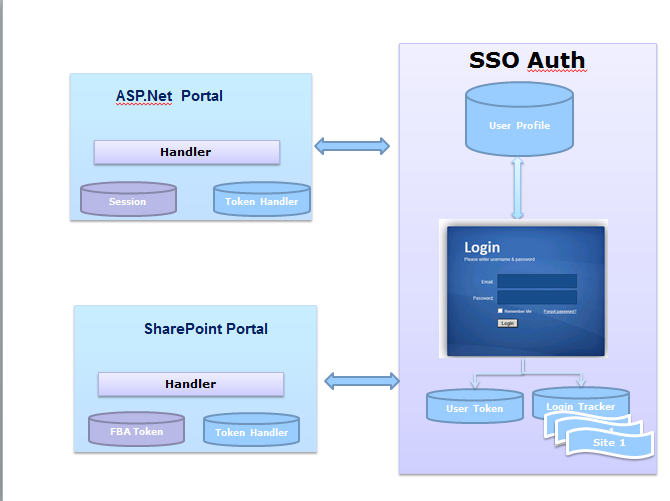
* Single Sign-On (SSO) is the feature that enables a user to receive a single challenge for credentials within a certain realm, and use the same token for access to other applications which require user authentication.
* Platform Agnostic : Custom SSO Auth is the custom way to support SSO across the technologies
* Easy To Integrate: Any technologies based Portal can be integrated with SSO Auth
* Center Login Box: Center Login box hosted on SSO Auth will be take care of Authenticating the user and creating the Token.
* Standard Patterns: SSO Auth does support Standard Patterns which will be needed to follow in order to integrate.
* User Token: User Token can be easily to be extent to contain more user profile Property which can be shared to all sites at the time when authentication has been requested

Top Features

* + Authenticate only once and use multiple portals or partner sites or resources.
  + Improved User Productivity
  + Ease of Administration

Prerequisite Requirements

1. FBA Enabled Site with custom membership & Provider
2. ASP.NET Application for SSO Auth- to store/share token
3. One ASP.net application to check the SSO.



To Test the SSO.

This Custom SSO is dependent on the patterns which every portal need to follow.

1. When user try to open your portal, redirect to [http://localhost:2013/login.aspx?getdirecttoken=true&tokenUrl=http%3a%2f%2fsp2010%3a2012%2f\_layouts%2fCMP\_Custom.FBAComponents%2fanonlogin.ashx](http://localhost:2013/login.aspx?getdirecttoken=true&tokenUrl=http://sp2010:2012/_layouts/CMP_Custom.FBAComponents/anonlogin.ashx)
   1. It contain two querystring one is
      1. Getdirecttoken
      2. tokenurl – url where SSO Auth will send Token
2. If user is not logged in then it will show asp.net login box on SSO Auth
3. Once user enter the username password which can be validate your single user profile source (you can implement LDAP user profile/custom user profile)
4. Then it will create Token and redirect user to Token URL

|  |
| --- |
| if ((!string.IsNullOrEmpty(Username)) && (!string.IsNullOrEmpty(Request[PASSWORD])))  {  UPS ups = new UPS();  if (ups.AuthenticateUser(Request[INPUTUSERNAME], Request[PASSWORD]))  {  //get the Token Property  User userProfiles = ups.GetUserProfile(Username, null);  //encrept the userporifle to send to portal  string token = CMPCryptor.Encrypt(userProfiles);  //create the cookie / session to track user  nvc = new NameValueCollection(2);  nvc.Add("UserName", token);  nvc.Add(SSO\_COOKIE\_NAME, Request[PORTALIDINPUTTAG]);  Utility.CreateCookieWithKeys(CookieType.Basic\_Server, nvc);  //Need to save UserToken  //prepare the response  string url = Request[PORTALTOKENURL] + "?token=" + HttpUtility.UrlEncode(token); ;  //string url = "login.aspx?token=" + HttpUtility.UrlEncode(token);  Response.Redirect(url);  }  else  {  //need to make errorcode enum class  Response.Redirect(Request[PORTALTOKENURL] + "?status=#32456");  } |

|  |
| --- |
| Encrypt |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Security.Cryptography;  using System.IO;  using System.Globalization;  using System.Xml.Serialization;  using System.Xml;  using System.Xml.Linq;  namespace CMP.Cryptor  {  public class CMPCryptor  {  #region Member Variables  private static string passPhrase = "a#s5pr@se";  private static string saltValue = "s@1t45^lue";  private static string hashAlgorithm = "SHA1"; // can be "MD5"  private static int passwordIterations = 2; // can be any number  private static string initVector = "@1B2c3D4e5F6g7H#"; // must be 16 bytes  private static int keySize = 256; // can be 192 or 128  #endregion  #region Public Methods  /// <summary>  ///  /// </summary>  /// <param name="user"></param>  /// <returns></returns>  public static string Encrypt(User user)  {  MemoryStream memoryStream = null;  XmlSerializer xs = null;  XmlTextWriter xmlTextWriter = null;  XDocument xDoc = null;  string encStr = string.Empty;  try  {  if (user != null)  {  memoryStream = new MemoryStream();  xs = new XmlSerializer(typeof(User));  xmlTextWriter = new XmlTextWriter(memoryStream, new UTF8Encoding(false));  xs.Serialize(xmlTextWriter, user);  string result = Encoding.UTF8.GetString(memoryStream.ToArray());  xDoc = XDocument.Parse(result.Trim());  encStr = GetEncryptedString(xDoc.ToString());  }  return encStr;  }  catch  {  throw;  }  finally  {  memoryStream.Close();  memoryStream.Dispose();  memoryStream = null;  xs = null;  xmlTextWriter.Close();  xmlTextWriter = null;  xDoc = null;  }  }  /// <summary>  ///  /// </summary>  /// <param name="encryptedText"></param>  /// <returns></returns>  public static User Decrypt(string encryptedText)  {  User userObj = null;  XmlSerializer serializer = null;  byte[] bytes = null;  string decryptedString = string.Empty;  MemoryStream stream = null;  try  {  serializer = new XmlSerializer(typeof(User));  decryptedString = GetDecryptedString(encryptedText);  bytes = Encoding.ASCII.GetBytes(decryptedString);  stream = new MemoryStream(bytes);  userObj = (User)serializer.Deserialize(stream);  return userObj;  }  catch  {  throw;  }  finally  {  stream.Close();  stream.Dispose();  bytes = null;  serializer = null;  userObj = null;  }  }  #endregion  #region Private Methods  /// <summary>  /// Encrypts a specified string  /// </summary>  /// <param name="plainText">The string to be encrypted</param>  /// <returns>The encrypted string</returns>  private static string GetEncryptedString(string plainText)  {  string cipherText = string.Empty;  try  {  byte[] initVectorBytes = Encoding.ASCII.GetBytes(initVector);  byte[] saltValueBytes = Encoding.ASCII.GetBytes(saltValue);  byte[] plainTextBytes = Encoding.UTF8.GetBytes(plainText);  // Create a password, from which the key will be derived.  // This password will be generated from the specified passphrase and  // salt value. The password will be created using the specified hash  // algorithm. Password creation can be done in several iterations.  PasswordDeriveBytes password = new PasswordDeriveBytes(passPhrase, saltValueBytes, hashAlgorithm, passwordIterations);  // Use the password to generate pseudo-random bytes for the encryption  // key. Specify the size of the key in bytes (instead of bits).  byte[] keyBytes = password.GetBytes(keySize / 8);  // Create uninitialized Rijndael encryption object.  RijndaelManaged symmetricKey = new RijndaelManaged();  // It is reasonable to set encryption mode to Cipher Block Chaining  // (CBC). Use default options for other symmetric key parameters.  symmetricKey.Mode = CipherMode.CBC;  // Generate encryptor from the existing key bytes and initialization  // vector. Key size will be defined based on the number of the key  // bytes.  ICryptoTransform encryptor = symmetricKey.CreateEncryptor(keyBytes, initVectorBytes);  // Define memory stream which will be used to hold encrypted data.  using (MemoryStream memoryStream = new MemoryStream())  {  // Define cryptographic stream (always use Write mode for encryption).  using (CryptoStream cryptoStream = new CryptoStream(memoryStream, encryptor, CryptoStreamMode.Write))  {  // Start encrypting.  cryptoStream.Write(plainTextBytes, 0, plainTextBytes.Length);  // Finish encrypting.  cryptoStream.FlushFinalBlock();  // Convert our encrypted data from a memory stream into a byte array.  byte[] cipherTextBytes = memoryStream.ToArray();  // Convert encrypted data into a base64-encoded string.  cipherText = Convert.ToBase64String(cipherTextBytes);  }  }  }  catch  {  // throwing actual exception to caller function  throw;  }  // Return encrypted string.  return cipherText;  }  /// <summary>  /// Decrypts a specified string  /// </summary>  /// <param name="plainText">The string to be decrypted</param>  /// <returns>The decrypted string</returns>  private static string GetDecryptedString(string cipherText)  {  string plainText = string.Empty;  try  {  byte[] initVectorBytes = Encoding.ASCII.GetBytes(initVector);  byte[] saltValueBytes = Encoding.ASCII.GetBytes(saltValue);  byte[] cipherTextBytes = Convert.FromBase64String(cipherText);  // Create a password, from which the key will be  // derived. This password will be generated from the specified  // passphrase and salt value. The password will be created using  // the specified hash algorithm. Password creation can be done in  // several iterations.  PasswordDeriveBytes password = new PasswordDeriveBytes(passPhrase, saltValueBytes, hashAlgorithm, passwordIterations);  // Use the password to generate pseudo-random bytes for the encryption  // key. Specify the size of the key in bytes (instead of bits).  byte[] keyBytes = password.GetBytes(keySize / 8);  // Create uninitialized Rijndael encryption object.  RijndaelManaged symmetricKey = new RijndaelManaged();  // It is reasonable to set encryption mode to Cipher Block Chaining  // (CBC). Use default options for other symmetric key parameters.  symmetricKey.Mode = CipherMode.CBC;  // Generate decryptor from the existing key bytes and initialization  // vector. Key size will be defined based on the number of the key  // bytes.  ICryptoTransform decryptor = symmetricKey.CreateDecryptor(keyBytes, initVectorBytes);  // Define memory stream which will be used to hold encrypted data.  using (MemoryStream memoryStream = new MemoryStream(cipherTextBytes))  {  // Define cryptographic stream (always use Read mode for encryption).  using (CryptoStream cryptoStream = new CryptoStream(memoryStream, decryptor, CryptoStreamMode.Read))  {  // Allocate the buffer long enough to hold ciphertext;  // plaintext is never longer than ciphertext.  byte[] plainTextBytes = new byte[cipherTextBytes.Length];  // Start decrypting.  int decryptedByteCount = cryptoStream.Read(plainTextBytes, 0, plainTextBytes.Length);  // Convert decrypted data into a string.  plainText = Encoding.UTF8.GetString(plainTextBytes, 0, decryptedByteCount);  }  }  }  catch  {  // throwing actual exception to caller function  throw;  }  return plainText;  }  /// <summary>  /// Convert Byte Array to String  /// </summary>  /// <param name="buffer">Array of Bytes</param>  /// <returns>String of Bytes</returns>  private static string byteToString(byte[] buffer)  {  StringBuilder sbBinary = new StringBuilder();  foreach (byte byteChar in buffer)  {  sbBinary.Append(byteChar.ToString("X2", CultureInfo.InvariantCulture)); // hex format  }  return sbBinary.ToString();  }  #endregion  }  } |

1. Then it will redirect to you SharePoint portal where you have to create Custom FBA Claim based token after decrypting the token

|  |
| --- |
|  |
| SecurityToken tk = null;  string url = HttpContext.Current.Request.Url.Scheme + "://" + HttpContext.Current.Request.Url.Authority;  //Create claims based authentication token  tk = SPSecurityContext.SecurityTokenForFormsAuthentication(new Uri(url), membershipProvider, roleProvider, strusername, String.Empty);    if (tk != null)  {  SPFederationAuthenticationModule fam = SPFederationAuthenticationModule.Current;  fam.SetPrincipalAndWriteSessionToken(tk);  HttpContext.Current.Response.Redirect("/");  } |

1. Now same patterns you need to implement in any other technologies as well here we are talking about FBA so it can be that particular session cookie as per technologies.
2. When next portal go to SSO Auth for token, it will return token as above and you need to use same algorithm to decrypt it.

