**Module Description**

**MODULES**

1. User Module.
2. Client module.
3. CSP Module.
4. De-duplication module.

**Modules Description**

**User** **Module**

In this module covers, user registration, forgot password, change password, user profile detail and user can able to update is profile information.

Registration:

Registration screen is used to register the user information for using application. Once registration completed user get the registration key for accessing module for cloud. Registered user can able to view shared client file and download the based on user key. And also he can upload his file and share to some other person.

Forgot Password:

In this screen used to retrieve your account. If the registered user forgot his credential this screen will help to get the account detail.

Change Password:

In this screen used to change password for the registered user. When the user logged in he can use the screen and change his own password.

User Profile:

In this screen used to view his detail. When logged in to the system, user can view his detail. User can modify his/her detail for their convenient.

**Client Module**

In this module, a client (nothing like a user) makes use of provider’s resource to store, retrieve and share data with multiple user.

In the module having File upload, file sharing, list of file he/she upload and view file detail screen. And client can able to delete his/her uploaded file.

File Upload:

In this screen client can able to upload file to cloud storage. We use cryptography for securing the data in cloud storage. Here we are restricting duplication file can’t upload using file checksum.

File Sharing:

In this screen client can able to share his/her file to multiple users. They can able to view and download.

File List:

In this screen client can view his/her uploaded file list. This screen shown entire list for his/her uploaded file.

File Detail:

In this screen client can able to view file detail and download option for the file. Client can able to delete his/her upload unwanted files.

**CSP Module:**

In this module CSP can view all the user details, client uploads details, clients detail and clients activities regarding the Secure Client Side De-duplication Scheme in Cloud Storage Environments.

In the module covers user list and file list screens.

User list:

In this screen admin can able to view list of users registered in the Secure Client Side De-duplication Scheme in Cloud Storage Environments. Admin can view individual user detail.

File list:

In this screen admin can able to view list of files uploaded by registered user in the Secure Client Side De-duplication Scheme in Cloud Storage Environments. Admin can able to view file detail and he/she have a permission to delete uploaded files.

**De-duplication module:**

In this module the clients will upload files that can be stored in cloud database and will built secure clients who can view the file from the database based on the de-duplicate Scheme Hence we Provide a highest Performing Secure Client With Normalized Cloud.

This is inbuilt module for file upload module. In this module we use cryptography to store and retrieving data from cloud storage environment. We use two separate algorithm for encryption and decryption for meta data and file content data.

1. Symmetric – AES algorithm for encryption and decryption for uploaded file content.

1. Asymmetric - Elliptic Curve Cryptography ( Ex: ECDiffieHellmanCng) for encryption and decryption for uploaded file meta data.

In this module we restrict duplicated file upload to cloud storage environment.

SHA512 – Checksum used to check file already exist in the storage area. SHA512 generate Unicode characters once for the file content it may not change if the file can’t modified.

1. AES – Symmetric Encryption / Decryption:

public static class SymmetricEncryption

{

public static byte[] Encrypt(byte[] data, string EncryptionKey)

{

byte[] encryptBytes;

using (Aes encryptor = Aes.Create())

{

Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, new byte[] { 0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });

encryptor.Key = pdb.GetBytes(32);

encryptor.IV = pdb.GetBytes(16);

using (MemoryStream ms = new MemoryStream())

{

using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateEncryptor(), CryptoStreamMode.Write))

{

cs.Write(data, 0, data.Length);

cs.Close();

}

encryptBytes = ms.ToArray();

}

}

return encryptBytes;

}

public static byte[] Decrypt(byte[] data, string DecryptionKey)

{

byte[] cipherBytes;

using (Aes encryptor = Aes.Create())

{

Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(DecryptionKey, new byte[] { 0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });

encryptor.Key = pdb.GetBytes(32);

encryptor.IV = pdb.GetBytes(16);

using (MemoryStream ms = new MemoryStream())

{

using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateDecryptor(), CryptoStreamMode.Write))

{

cs.Write(data, 0, data.Length);

cs.Close();

}

cipherBytes = ms.ToArray();

}

}

return cipherBytes;

}

}

1. Asymmetric - Elliptic Curve Cryptography Encryption/ Decryption

public static class EllipticAsymmetric

{

public static void KeyGenerator(out byte[] SecretA, out byte[] SecretB)

{

SecretA = null;

SecretB = null;

byte[] SecretAPublic;

byte[] SecretBPublic;

ECDiffieHellmanCng A = new ECDiffieHellmanCng();

A.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;

A.HashAlgorithm = CngAlgorithm.Sha256;

SecretAPublic = A.PublicKey.ToByteArray();

ECDiffieHellmanCng B = new ECDiffieHellmanCng();

B.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;

B.HashAlgorithm = CngAlgorithm.Sha256;

SecretBPublic = B.PublicKey.ToByteArray();

SecretB = B.DeriveKeyMaterial(CngKey.Import(SecretAPublic, CngKeyBlobFormat.EccPublicBlob));

CngKey k = CngKey.Import(SecretBPublic, CngKeyBlobFormat.EccPublicBlob);

SecretA = A.DeriveKeyMaterial(CngKey.Import(SecretBPublic, CngKeyBlobFormat.EccPublicBlob));

}

public static byte[] Encrypte(byte[] key, string secretMessage)

{

byte[] encryptedMessage = null;

UnicodeEncoding ByteConverter = new UnicodeEncoding();

string EncryptionKey = ByteConverter.GetString(key);

using (Aes aes = new AesCryptoServiceProvider())

{

Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, new byte[] { 0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });

aes.Key = pdb.GetBytes(32);

aes.IV = pdb.GetBytes(16);

// Encrypt the message

using (MemoryStream ciphertext = new MemoryStream())

using (CryptoStream cs = new CryptoStream(ciphertext, aes.CreateEncryptor(), CryptoStreamMode.Write))

{

byte[] plaintextMessage = Encoding.UTF8.GetBytes(secretMessage);

cs.Write(plaintextMessage, 0, plaintextMessage.Length);

cs.Close();

encryptedMessage = ciphertext.ToArray();

}

}

return encryptedMessage;

}

public static string Decrypte(byte[] encryptedMessage, byte[] key)

{

string decryptedMessage = string.Empty;

UnicodeEncoding ByteConverter = new UnicodeEncoding();

string DecriptionKey = ByteConverter.GetString(key);

using (Aes aes = new AesCryptoServiceProvider())

{

Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(DecriptionKey, new byte[] { 0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });

aes.Key = pdb.GetBytes(32);

aes.IV = pdb.GetBytes(16);

// Decrypt the message

using (MemoryStream plaintext = new MemoryStream())

{

using (CryptoStream cs = new CryptoStream(plaintext, aes.CreateDecryptor(), CryptoStreamMode.Write))

{

cs.Write(encryptedMessage, 0, encryptedMessage.Length);

cs.Close();

decryptedMessage = Encoding.UTF8.GetString(plaintext.ToArray());

}

}

}

return decryptedMessage;

}

}

1. File Checksum SHA512

public static class CheckSumGenerator

{

public static string GetCheckSum(Stream file, int length, out byte[] data)

{

data = GetByteFromStream(file, length);

HashAlgorithm SHA512 = new SHA512Managed();

byte[] hash = SHA512.ComputeHash(data);

return Convert.ToBase64String(hash.ToArray());

}

public static byte[] GetByteFromStream(Stream file, int length)

{

byte[] imgByte = null;

using (BinaryReader breader = new BinaryReader(file))

{

imgByte = breader.ReadBytes(length);

}

return imgByte;

}

}