1. AccessToken Generation Technique:

1) Compute SHA256 hash of plain password.
2) Generate Timestamp.
3) Concatenate the output of step 1 and step 2 (Step 1 output + Step 2 output)
4) Compute SHA256 hash of Step 3 output.
5) Concatenate the output of step 3 and PwdSecretKey (Step 3 output + PwdSecretKey)
6) Compute SHA256 hash of Step 5 output.

Note: Username, Password, and PwdSecretKey will be shared separately via Email/SMS.

Timestamp Generation (C#):

```
string TS = DateTime.UtcNow.ToString("yyyyMMddHHmmss");
```

Hash Generation Method(C#):

```
public string ComputeSha256Hash(string pstrInputString)
{
 string lstrcode = pstrInputString;
  // Create a SHA256
 using (SHA256 sha256Hash = SHA256.Create())
 {
  // ComputeHash - returns byte array
  byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(lstrcode));
  string hashString = string.Empty;
  foreach (byte x in bytes)
  {
     hashString += String.Format("{0:x2}", x);
   }
  return hashString;
 }
}
```

2. Advanced Encryption Standard (AES) for Encryption:

- **Cipher Mode Operation :** Galois/Counter Mode (GCM) with No Padding
- **Cryptographic Key :** 256 bits*
- **IVector :** First 12 byte of cryptographic key (Secret Key)
- **GCM Tag Length :** 16 Bytes

Note: AES secret key will be shared separately via Email/SMS.

AES Encryption Logic (Programming Language: C#):

```
using Org.BouncyCastle.Crypto.Engines;

using Org.BouncyCastle.Crypto.Modes;

using Org.BouncyCastle.Crypto.Parameters;

using System.Security.Cryptography;

-----------------------------------------------------------------
var key = <Dynamic Key 32 character>
var secretkey = Encoding.UTF8.GetBytes(key);
var iv = Encoding.UTF8.GetBytes(Encoding.UTF8.GetString(secretkey).Substring(0, 12));

-----------------------------------------------------------------

public string AESEncrypt(string PlainText, byte[] key, byte[] iv)
    {

        string EncryptedStr = string.Empty;

        try

        {

           byte[] plainBytes = Encoding.UTF8.GetBytes(PlainText);
           GcmBlockCipher cipher = new GcmBlockCipher(new AesEngine());    //GCM Cipher

           AeadParameters parameters =

                    new AeadParameters(new KeyParameter(key), 128, iv, null);

           cipher.Init(true, parameters);

           byte[] encryptedBytes = new byte[cipher.GetOutputSize(plainBytes.Length)];
           Int32 retLen = cipher.ProcessBytes
                     (plainBytes, 0, plainBytes.Length, encryptedBytes, 0);

           cipher.DoFinal(encryptedBytes, retLen);

           EncryptedStr = Convert.ToBase64String(encryptedBytes, Base64FormattingOptions.None);
        }

        catch (Exception ex) {  }

        return EncryptedStr;

    }
```

Function Definition:

public string AESEncrypt(string PlainText, byte[] key, byte[] iv)

- Public string: The function returns a string.
- AESEncrypt: The name of the function.
- string PlainText: The plaintext string to be encrypted.
- byte[] key: The encryption key as a byte array.
- byte[] iv: The initialization vector (IV) as a byte array.

Variable Initialization:

- sR: A string variable initialized to an empty string to store the result.

Convert PlainText to Byte Array:

byte[] plainBytes = Encoding.UTF8.GetBytes(PlainText);

GCM Cipher Initialization:

GcmBlockCipher cipher = new GcmBlockCipher(new AesEngine());

- GcmBlockCipher: Uses Galois/Counter Mode (GCM) for AES.
- new AesEngine(): Specifies the AES encryption engine.

Set Parameters:

AeadParameters parameters = new AeadParameters(new KeyParameter(key), 128, iv,null);

- AeadParameters: Parameters for authenticated encryption with associated data (AEAD).
- new KeyParameter(key): The encryption key.
- 128: The MAC size in bits.
- iv: The initialization vector.
- null: Additional associated data (AAD), not used here.

Initialize Cipher:

cipher.Init(true, parameters);

- true: Specifies encryption mode (false would be decryption).
- parameters: The parameters for the cipher.

Encrypt Plain Bytes:

byte[] encryptedBytes = new byte[cipher.GetOutputSize(plainBytes.Length)]; Int32 retLen = cipher.ProcessBytes(plainBytes, 0, plainBytes.Length,

encryptedBytes, 0);

cipher.DoFinal(encryptedBytes, retLen);

- encryptedBytes: Byte array to hold the encrypted data.
- cipher.GetOutputSize(plainBytes.Length): Gets the size of the output buffer needed.
- cipher.ProcessBytes: Processes the input bytes.
- cipher.DoFinal: Completes the encryption process.

Convert Encrypted Bytes to Base64 String:

```
sR = Convert.ToBase64String(encryptedBytes, Base64FormattingOptions.None);
```

- Convert.ToBase64String: Converts the encrypted byte array to a Base64 string.
- Base64FormattingOptions.None: Specifies no line breaks in the output string.

AES Decryption Logic (Programming Language: C#):

```
public string AESDecrypt(string EncryptedText, byte[] key, byte[] iv)
    {
        string DecryptedStr = string.Empty;
        try
        {
           byte[] encryptedBytes = Convert.FromBase64String(EncryptedText);
           GcmBlockCipher cipher = new GcmBlockCipher(new AesEngine());
           AeadParameters parameters =
                   new AeadParameters(new KeyParameter(key), 128, iv, null);
           //ParametersWithIV parameters = new ParametersWithIV(new KeyParameter(key), iv);
           cipher.Init(false, parameters);
           byte[] plainBytes = new byte[cipher.GetOutputSize(encryptedBytes.Length)];
           Int32 retLen = cipher.ProcessBytes
                     (encryptedBytes, 0, encryptedBytes.Length, plainBytes, 0);
           cipher.DoFinal(plainBytes, retLen);
           DecryptedStr = Encoding.UTF8.GetString(plainBytes).TrimEnd("\r\n\0".ToCharArray());
        }
        catch (Exception ex) { }
        return DecryptedStr;
    }
```

Note: Kindly modify the above programming logic according to your own implementation.

------End of Document-------