HASH GENERATION DOCUMENT

DOCUMENT VERSION HISTORY

Version	Description	Author	Approved by	Remark
1.1	HMAC generation algorithm	Vishal Mishra	Rakesh Shukla	

CONTENTS

HASH GENERATION

This Method takes input data dictionary, Secure Secret Key, Hash Type and reference string for Generated Hash by this Method. If this Method fails in processing it returns false. If Method returns true then the calculated Hash string value will be set to the reference variable (strHash variable in the below described method)

Input Parameters:

dictInput:

Input key Value Pair Dictionary has all the input fields except Secure Secret Hash and Hash Type.

strSecretKey:

Secure Secret Key Provided by PinePG

strHashType:

Hashing Algorithm Type

strHash:

This Parameter is passed as reference. It will be set as the Generated Hash Value.

Return Parameter:

DataType as bool (true/false), It will be true in case of success else it will be false.

```
/// <summary>
/// This Method is used to Generate Hash.
/// If Method returns true, strHash will contain Generated Hash
value.
/// </summary>
/// <param name="dictInput">
/// Input Data Dictionary of Message to be hashed
/// </param>
/// <param name="strSecretKey">
/// SecureSecret Provided By PinePG
/// </param>
/// <param name="strHashType">
/// Hash Generation Algorithm
/// </param>
/// <param name="strHash">- Out Param
/// Generated Hash Value.
/// </param>
/// <returns>
/// If true then Success, else Fail
/// </returns>
private bool GenerateHash(Dictionary<string, string> dictInput,
string strSecretKey, string strHashType, ref string strHash)
{
bool bReturn = false;
try
{

    if ((dictInput == null )|| (dictInput.Count == 0 )||
    (string.IsNullOrEmpty(strSecretKey)) ||
(string.IsNullOrEmpty(strHashType)))
    {
        return bReturn;
    }

    SortedList<string, string> sortdLstRequestFields = new
SortedList<string, string>();

    foreach (KeyValuePair<string, string> keyValPair in dictInput)
    {
            sortdLstRequestFields.Add(keyValPair.Key,
keyValPair.Value);
    }
    //Convert Secret Key to required format
    byte[] convertedHashKey = new byte[strSecretKey.Length / 2];
    for (int i = 0; i < strSecretKey.Length / 2; i++)
    {
        convertedHashKey[i] =
(byte)Int32.Parse(strSecretKey.Substring(i * 2, 2),
        System.Globalization.NumberStyles.HexNumber);
    }

    // Build string from collection.
    StringBuilder sbMessage = new StringBuilder();
    foreach (KeyValuePair<string, string> kvp in
sortdLstRequestFields)
    {
            sbMessage.Append(kvp.Key + "=" + kvp.Value + "&");
    }
    sbMessage.Remove(sbMessage.Length - 1, 1);

    strHash = "";
```

```
    //generate hash
    if (strHashType.ToUpper().Equals("SHA256"))
    {
        using (HMACSHA256 hasherSHA256 = new
HMACSHA256(convertedHashKey))
        {
            byte[] hashValue = hasherSHA256.ComputeHash(
            Encoding.UTF8.GetBytes(sbMessage.ToString()));
            foreach (byte b in hashValue)
            {
                strHash += b.ToString("X2");
            }
        }
        bReturn = true;

    }
    else if (strHashType.ToUpper().Equals("MD5"))
    {

        using (HMACMD5 hasher = new HMACMD5(convertedHashKey))
        {
            byte[] hashValue = hasher.ComputeHash(
            Encoding.UTF8.GetBytes(sbMessage.ToString()));
            foreach (byte b in hashValue)
            {
                strHash += b.ToString("X2");
            }
        }
        bReturn = true;
    }

    if (string.IsNullOrEmpty(strHash))
    {
        bReturn =  false;
    }

}
catch (Exception ex)
{
    bReturn = false;
}
return bReturn;
}
```