PoP Token Builder Library
C# / .NET Core 3.1

Build/Install PoP Token Builder Library

1) Open up a Developer Command Window
2) Run the following command

```
msbuild.exe PopTokenBuilderSolution.sln /t:Clean,Build /p:Configuration=Release
```

Implementation Details

The T-Mobile PoP Token Builder library follows the following logic for creating the PoP token.

- Sets up the edts (external data to sign) / ehts (external headers to sign) claims in PoP token using the specified ehts key-value map. The library uses SHA256 algorithm for calculating the edts and then the final edts value is encoded using Base64 URL encoding.
- Signs the PoP token using the specified RSA private key.
- Creates the PoP token with 2 minutes of validity period.
- Current PoP token builder libraries support RSA PKCS8 format key for signing and validating the PoP tokens.

Determining the ehts Key Name

- For HTTP request URI, "uri" should be used as ehts key name, `PopEhtsKeyEnum.Uri.GetDescription()`.
- For "uri" ehts value, the URI and query string of the request URL should be put in the ehts key-value map. Example:
    - If the URL is `https://api.t-mobile.com/commerce/v1/orders?account-number=0000000000` then only `/commerce/v1/orders?account-number=0000000000` should be used as ehts value.
    - The query parameter values part of "uri" ehts value should not be in URL encoded format.

- For HTTP method, "http-method" should be used as ehts key name, `PopEhtsKeyEnum.HttpMethod.GetDescription()`.
- For HTTP request headers, the header name should be used as ehts key name.
- For HTTP request body, "body" should be used as ehts key name, `PopEhtsKeyEnum.Body.GetDescription()`.
- For code sample, see test "`PopTokenBuilder_Build_ValidPopToken_Success_Test`"

Supported Key Format

The PoP token builder library currently supports PKCS8 key format.

Using Non Encrypted Keys:

Below commands shows how to create private and public keys in PKCS8 format:

```
# Create a 2048 bit Private RSA key in PKCS1 format
openssl genrsa -out private-key-pkcs1.pem 2048

# Convert the Private RSA key to PKCS8 format.
openssl pkcs8 -topk8 -inform PEM -in private-key-pkcs1.pem -outform PEM -nocrypt -out
private-key-pkcs8.pem

# Create a Public RSA key in PKCS8 format
openssl rsa -in private-key-pkcs8.pem -outform PEM -pubout -out public-key.pem

# Convert the keys from PEM format to XML format
# By hand: https://superdry.apphb.com/tools/online-rsa-key-converter
# Via code: https://gist.github.com/misaxi/4642030
```

Building the PoP Token Using Private Key PEM XML String

The following Unit Test shows how to build the PoP token using private key PEM XML string.

```
[TestClass]
public class PopTokenBuilderUnitTest
{

string privateKeyPemRsa;
string privateKeyXmlRsa;
string audience;
string issuer;

[TestInitialize]
public void TestInitialize()
{
    // Private Key
    var privateKeyPemRsaStringBuilder = new StringBuilder();
    privateKeyPemRsaStringBuilder.AppendLine("-----BEGIN PRIVATE KEY-----");
privateKeyPemRsaStringBuilder.AppendLine("MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQDZPQF1bZdaY80nB/xK+8KdApRhv/nlaIuHR6jaoEih0YAz5N3
cd++zH5BY+HX9Muc8pAbmtypMs5o2HIcS12Qm+H998eXWcL8rVUJURLqZpObWY0egPomOqowwZGHFhbTG02bRZWjANIFTQj2BukOi/6m9o4GvlhGrGSdjXhskrFmWPTO/gT6Dv3SJ8eLd
KeQU1AV17JB4vreJgWs1ZRPtdwPNc+ef3UzS8w710ykjSGSxDWF7SaUqn8KUAL1DMmBgD1bectIYsT2646KkABSayYGAe+6UjdYAN2OLYvEbScGdDDtZgQxIu1GDASoFn1xh5iJ5o7sq5
```

```
1zQM/dBhLWbAgMBAAECggEATOwDJjeGDmWkcRusxEhdYwdUz0ARFqBsN5yyN6fl0BbE1JtHzBdT8xNMI5TnAp8RrjFOmEdnXP2Dr4Fuesd2GS6IxmnvPn1x08A+2mPzxw/TBTrmU+GRB8
lwFnqU/EIZ/wVANQk5jEWLPZYI/XSdGox46EOLWkdDPliLz+20oskVdMF0JwC0/pwFuleUUv5ryPZ40c4OP/VfXDcjYY66GWHKgWUVmp9CxMqWwfFbwKqjZh+s3vMEqGjwwZ7PNRylFTk
yeaIu9KcMaoQy5VZodyzaf3cVofkW9Ue2Ty6Rrn+37W+3ZUYmFRLLAatoDm4K4m2zbS8T4zWJwB3Qpma3kQKBgQDtW5G+xVIwv7cMXPkXpkmM9VYDb72XlwxbiPS20oywh7e8rWM3ikC+
I0tnJKmd/Ek7iacNvVsdoBm0rEE/KwCUCiBklAFIT5EMuzBdshX8hgusvdBHxaGFrMFyXMFWvof/dAolHRtZxfeOrRLlq1Q8QjEhu2TtDHIHkO31MF/61QKBgQDqTOfxwxUV2TPNOKxLC
uViXCnRIQC0dpsXFMELCvz5EspgfCPC86WLOL2zoxfJN0wSkyUTONjvEXeLv3cdrc6fJs2cuGZHiuKfhLwlLdLoYoWju3uE/ucYuPdkTGKJYkeAT4jXDtHQezWbSw9daxrFN9DThpiiin
bd6SpVEQpGrwKBgEajacyMjNlVNPTI9pjDNEBvAxoitxnWKidTqwB9yMEAov3T3CM7UelEN7yKfLA31NOTM9Qy9lrquru6R/C0q4djPCCyq28JvvE7BDneNgzhF7hhBQtXFariru+KCz/
1lCPCNQK2lt0wvWwItgcD5h3I1TZkvrSNb6Iwz6CYtPfBAoGAaKDjSxEEz3bpLRHLzs8U1DG38s28FNqKM2pvMlE72rZLbX7CMYLAQEWcYSXJr29kJz9SZR0Tst6n9d4QgU5mYKfhVcT6
16Prw7RwmGG4N1IXv6Avbpqt9FpVD5MUxaj/qQrbXr4db+41aB3CxMLZd4yPUoZejucqYbqHzukHH70CgYA/9osJo6W+MRTYWkDonnj37HivtmbHxNukxU9UmlNuxXJDn8D9w/M5b4PTi
0isg/jZ9oJ/gqnJwDuXmIHebk2Qi0EHXPaRZE0ECATwWHt1MzqPHGifU3OZ1S5XVtEjXghFTa9A1T+JObhayAdNYCbPxw3OGu4FvOR3eWFQNkfA4g==");
    privateKeyPemRsaStringBuilder.AppendLine("-----END PRIVATE KEY-----");
    privateKeyPemRsa = privateKeyPemRsaStringBuilder.ToString();

    // Converted private key from PEM format to XML format
    privateKeyXmlRsa =
"<RSAKeyValue><Modulus>2T0BdW2XWmPNJwf8SvvCnQKUYb/55WiLh0eo2qBIodGAM+Td3Hfvsx+QWPh1/TLnPKQG5rcqTLOaNhyHEtdkJvh/ffHl1nC/K1VCVES6maTm1mNHoD6Jjq
qMMGRhxYW0xtNm0WVowDSBU0I9gbpDov+pvaOBr5YRqxknY14bJKxZlj0zv4E+g790ifHi3SnkFNQFdeyQeL63iYFrNWUT7XcDzXPnn91M0vMO9dMpI0hksQ1he0mlKp/ClAC9QzJgYA9
W3nLSGLE9uuOipAAUmsmBgHvulI3WADdji2LxG0nBnQw7WYEMSLtRgwEqBZ9cYeYieaO7Kudc0DP3QYS1mw==</Modulus><Exponent>AQAB</Exponent><P>7VuRvsVSML+3DFz5F6
ZJjPVWA2+9l5cMW4j0ttKMsIe3vK1jN4pAviNLZySpnfxJO4mnDb1bHaAZtKxBPysAlAogZJQBSE+RDLswXbIV/IYLrL3QR8WhhazBclzBVr6H/3QKJR0bWcX3jq0S5atUPEIxIbtk7Qx
yB5Dt9TBf+tU=</P><Q>6kzn8cMVFdkzzTisSwrlYlwp0SEAtHabFxTBCwr8+RLKYHwjwvOlizi9s6MXyTdMEpMlEzjY7xF3i793Ha3OnybNnLhmR4rin4S8JS3S6GKFo7t7hP7nGLj3Z
ExiiWJHgE+I1w7R0Hs1m0sPXWsaxTfQ04aYoop23ekqVREKRq8=</Q><DP>RqNpzIyM2VU09Mj2mMM0QG8DGiK3GdYqJ1OrAH3IwQCi/dPcIztR6UQ3vIp8sDfU05Mz1DL2Wuq6u7pH8L
Srh2M8ILKrbwm+8TsEOd42DOEXuGEFC1cVquKu74oLP/WUI8I1AraW3TC9bAi2BwPmHcjVNmS+tI1vojDPoJi098E=</DP><DQ>aKDjSxEEz3bpLRHLzs8U1DG38s28FNqKM2pvMlE72r
ZLbX7CMYLAQEWcYSXJr29kJz9SZR0Tst6n9d4QgU5mYKfhVcT616Prw7RwmGG4N1IXv6Avbpqt9FpVD5MUxaj/qQrbXr4db+41aB3CxMLZd4yPUoZejucqYbqHzukHH70=</DQ><Inver
seQ>P/aLCaOlvjEU2FpA6J549+x4r7Zmx8TbpMVPVJpTbsVyQ5/A/cPzOW+D04tIrIP42faCf4KpycA7l5iB3m5NkItBB1z2kWRNBAgE8Fh7dTM6jxxon1NzmdUuV1bRI14IRU2vQNU/i
Tm4WsgHTWAmz8cNzhruBbzkd3lhUDZHwOI=</InverseQ><D>TOwDJjeGDmWkcRusxEhdYwdUz0ARFqBsN5yyN6fl0BbE1JtHzBdT8xNMI5TnAp8RrjFOmEdnXP2Dr4Fuesd2GS6Ixmnv
Pn1x08A+2mPzxw/TBTrmU+GRB8lwFnqU/EIZ/wVANQk5jEWLPZYI/XSdGox46EOLWkdDPliLz+20oskVdMF0JwC0/pwFuleUUv5ryPZ40c4OP/VfXDcjYY66GWHKgWUVmp9CxMqWwfFbw
KqjZh+s3vMEqGjwwZ7PNRylFTkyeaIu9KcMaoQy5VZodyzaf3cVofkW9Ue2Ty6Rrn+37W+3ZUYmFRLLAatoDm4K4m2zbS8T4zWJwB3Qpma3kQ==</D></RSAKeyValue>";

    // ClientId
    audience = "JYM89zuJAf3D1N0omc0VzaoehUW5Inn3";
    issuer = "JYM89zuJAf3D1N0omc0VzaoehUW5Inn3";
}

[TestMethod]
public void PopTokenBuilder_Build_ValidPopToken_Success_Test()
{
    // Arrange
    var keyValuePairDictionary = new Dictionary<string, string>();
    keyValuePairDictionary.Add(PopEhtsKeyEnum.ContentType.GetDescription(), PopEhtsKeyEnum.ApplicationJson.GetDescription());
    keyValuePairDictionary.Add(PopEhtsKeyEnum.Authorization.GetDescription(), "Bearer UtKV75JJbVAewOrkHMXhLbiQ11SS");
    keyValuePairDictionary.Add(PopEhtsKeyEnum.Uri.GetDescription(), "/commerce/v1/orders");
    keyValuePairDictionary.Add(PopEhtsKeyEnum.HttpMethod.GetDescription(), PopEhtsKeyEnum.Post.GetDescription());
    keyValuePairDictionary.Add(PopEhtsKeyEnum.Body.GetDescription(), "{\"orderId\": 100, \"product\": \"Mobile Phone\"}");
    var hashMapKeyValuePair = HashMapKeyValuePair.Set<string, string>(keyValuePairDictionary);
    var popTokenBuilder = new PopTokenBuilder(audience, issuer);

    // Act
    var popToken = popTokenBuilder.SetEhtsKeyValueMap(hashMapKeyValuePair)
                                  .SignWith(privateKeyXmlRsa)
                                  .Build();
    // Assert
    Assert.IsTrue(!string.IsNullOrEmpty(popToken));
}
```