**Proof Of Possession (PoP) Token Builder using .NET Standard 2.1**

**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().

**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

**Building the PoP Token Using Private Key PEM or XML String**

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

The first unit shows how to create a POP token for an OAuth2 call for an access token.

The following unit test shows how to create a POP token for an API call after you got the access token above.

[TestClass]

public class PopTokenBuilderUnitTest

{

string \_publicRsaKeyPem;

string \_publicRsaKeyXml;

string \_privateRsaKeyPem;

string \_privateRsaKeyXml;

string audience;

string issuer;

[TestInitialize]

public void TestInitialize()

{

// 1) Create RSA public/private keys (one time)

// Download OpenSSL for Windows

// https://sourceforge.net/projects/gnuwin32/postdownload

// # 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

// private-key-pkcs8.pem

// public-key.pem

// Private Key

var privateKeyPemRsaStringBuilder = new StringBuilder();

privateKeyPemRsaStringBuilder.AppendLine("-----BEGIN PRIVATE KEY-----");

privateKeyPemRsaStringBuilder.AppendLine("");

privateKeyPemRsaStringBuilder.AppendLine("-----END PRIVATE KEY-----");

\_privateRsaKeyPem = privateKeyPemRsaStringBuilder.ToString();

// (Optional) Private Key converted from PEM format to XML format  
 // By hand https://superdry.apphb.com/tools/online-rsa-key-converter

\_privateRsaKeyXml = "<RSAKeyValue><Modulus>n2da7FfnjpYVNWKtS0KMck8M50hG7VEPu/desMPsWuZTnd5XUsSCf3/++qE8EpybX4RZYMY8SqiEVGvDtzYUVWeWhLzB6YxzHkzWu3sK+5KalgOStHSRPCrAgdjcdPgRi4AhAt5aRd+8WVSJHM6c0n50OLgsrijzbj9aWYABNu2uQLiVNqYxkuEV0e+wJYR0XlSNbE9AjG4kwZw+JCeBvUH62sqg9xTDTL2DingWZC2qYq/jU25U5wMxskbuvYUCzNA4PV8fA9vJhWE5/MgRDcYWY7ajr8S3JqYoad82AOn6LBt8/4G4WQZ0z38tytWeg1/wGbOC4MktIaaEMWbAdw==</Modulus><Exponent>AQAB</Exponent><P>zbMFsT57lnHsEDlr3K5h7nIhB+CttlgBPz3wympiJ51zKfjh3Tl99L9XSNg/mpgxcTdBfzCsjMKr71OO9TGp/rl93a4PxlwaTBzUhMGbCSuoYCxVzMIAywuFSDJvnuk5IAhVlWV7Nsi+2wz1wTfWBIXqWubZjiKzqYf5QW4+lg8=</P><Q>xmIvRzMQgqGFT0l0O9O/9pJjC7qk1riqSczClOoYwG183OlOa3xguHyDw6gC19ggsKtyBKDT+mgY2ppdnTbbJk1TIDdCagD3T3iaBiEad6m7Fs+HRdJO9sVOx7r8l9ocggKJuPBL7AscYvIIosmC+GV5sbUxIX71b01mNiYFVxk=</Q><DP>RuMP7iIDQzhlSr4PHtD1rM+l9GoIU1OGsn2tEoSQ6OgIvQkpBSz/7C1YbiEf4i3atBJ/vs5OWH/p8qMQHA2OcNsJtjB6/TfWVC6HSmzR+doSv3nn45Vj4pVIzDWdY90ps5FLtR1w1dNeemy/8GNGnO5tcgAmLyZkVeMnEdZlOR8=</DP><DQ>WB+VUNNmKiEFzsqaT1kolKdCSBuIzbkKK+5BIVU72X7JUHhy1VxSuqDVBzzCxo7DNrdx1ox6nWlQYQrhOsz7XHBM1Kq3Xc9ADJVOFhruXumOqftV47YgTY4oCKEPQ4Un1Li75OMZVqk42tsY6vcIrr6k6EPMp0x2ShLfrH4HMUE=</DQ><InverseQ>ip2YXm8yrDpWOSeL5fnqtA0zFnLc28Bxc47RRcy3jjMPQ9ADfRXfa087Te+WzG0p1wZJWSpTINQjTX+BdUMpmicgU7iX/QDDAVuvKImbb9TBCO0D9OZ+fnogq03MwerZyTuws2pS5BEytgdlcTYG+w+prDZi0ll8U+EQgWeaFUQ=</InverseQ><D>XOpJBITE08dF+4VWUA0tgp/zfIkT1tcuXbl2d4Dsr5ucV+Q3cGZdTuaUARGky5B/vLCPzKogkMAjynW6cnvSZGnqQdspCPK2U44kiMnTAAtXkmPoysk7sx+UcNuwvXmv+GmqVFq5sgsVZdixx5njrYrKQhmQ6b+zDateBddoXdRH+N9RrU5lwzqhwPnswO79cjPkHd5+3H/2dirNXa5VNK0ykdGd6f0V5aesDcZwl/96VGgOX9T23Ghf4gNt2JoAcp4wKwz2u0AUgM4sJP13FXbfRhB61c9aBjldzoTVpNZofI7xADxjVWl4HRdFB+5e3xGTbDbRU/Vl/4RWpO2c0Q==</D></RSAKeyValue>";

// Public Key

var publicKeyPemRsaStringBuilder = new StringBuilder();

publicKeyPemRsaStringBuilder.AppendLine("-----BEGIN PUBLIC KEY-----");

publicKeyPemRsaStringBuilder.AppendLine("MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAn2da7FfnjpYVNWKtS0KMck8M50hG7VEPu/desMPsWuZTnd5XUsSCf3/++qE8EpybX4RZYMY8SqiEVGvDtzYUVWeWhLzB6YxzHkzWu3sK+5KalgOStHSRPCrAgdjcdPgRi4AhAt5aRd+8WVSJHM6c0n50OLgsrijzbj9aWYABNu2uQLiVNqYxkuEV0e+wJYR0XlSNbE9AjG4kwZw+JCeBvUH62sqg9xTDTL2DingWZC2qYq/jU25U5wMxskbuvYUCzNA4PV8fA9vJhWE5/MgRDcYWY7ajr8S3JqYoad82AOn6LBt8/4G4WQZ0z38tytWeg1/wGbOC4MktIaaEMWbAdwIDAQAB");

publicKeyPemRsaStringBuilder.AppendLine("-----END PUBLIC KEY-----");

\_publicRsaKeyPem = publicKeyPemRsaStringBuilder.ToString();

// (Optional) Public Key converted from PEM format to XML format  
 // By hand https://superdry.apphb.com/tools/online-rsa-key-converter

\_publicRsaKeyXml = "<RSAKeyValue><Modulus>n2da7FfnjpYVNWKtS0KMck8M50hG7VEPu/desMPsWuZTnd5XUsSCf3/++qE8EpybX4RZYMY8SqiEVGvDtzYUVWeWhLzB6YxzHkzWu3sK+5KalgOStHSRPCrAgdjcdPgRi4AhAt5aRd+8WVSJHM6c0n50OLgsrijzbj9aWYABNu2uQLiVNqYxkuEV0e+wJYR0XlSNbE9AjG4kwZw+JCeBvUH62sqg9xTDTL2DingWZC2qYq/jU25U5wMxskbuvYUCzNA4PV8fA9vJhWE5/MgRDcYWY7ajr8S3JqYoad82AOn6LBt8/4G4WQZ0z38tytWeg1/wGbOC4MktIaaEMWbAdw==</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>";

// 3) Setup Audience/Issuer

audience = "123";

issuer = "abc";

}

/// <summary>

/// Example 1a: Calling oAuth2 server to get bearer token with poptoken (Pem Format)

/// </summary>

[TestMethod]

public void PopTokenBuilder\_Build\_ValidateToken\_OAuth2Example\_PEMFormat\_Success\_Test()

{

// Arrange

var keyValuePairDictionary = new Dictionary<string, string>

{

{ PopEhtsKeyEnum.Authorization.GetDescription(), "Basic UtKV75JJbVAewOrkHMXhLbiQ11SS" },

{ PopEhtsKeyEnum.Uri.GetDescription(), "/oauth2/v6/tokens" },

{ PopEhtsKeyEnum.HttpMethod.GetDescription(), PopEhtsKeyEnum.Post.GetDescription() },

};

var hashMapKeyValuePair = HashMapKeyValuePair.Set<string, string>(keyValuePairDictionary);

var popTokenBuilder = new PopTokenBuilder(audience, issuer);

// Act

var popToken = popTokenBuilder.SetEhtsKeyValueMap(hashMapKeyValuePair)

.SignWith(\_privateRsaKeyPem) // PEM format

.Build();

var publicRsaSecurityKey = PopTokenBuilderUtils.CreateRsaSecurityKey(\_publicRsaKeyPem); // PEM format

var tokenValidationResult = PopTokenBuilderUtils.ValidateToken(popToken, issuer, audience, publicRsaSecurityKey);

// Assert

Assert.IsNotNull(popToken);

Assert.IsNotNull(tokenValidationResult);

Assert.IsTrue(tokenValidationResult.IsValid);

Assert.IsTrue(tokenValidationResult.Claims.Count == 9);

}

/// <summary>

/// Example 1a: Calling oAuth2 server to get bearer token with poptoken (Xml Format - Optional)

/// </summary>

[TestMethod]

public void PopTokenBuilder\_Build\_ValidateToken\_OAuth2Example\_XMLFormat\_Success\_Test()

{

// Arrange

var keyValuePairDictionary = new Dictionary<string, string>

{

{ PopEhtsKeyEnum.Authorization.GetDescription(), "Basic UtKV75JJbVAewOrkHMXhLbiQ11SS" },

{ PopEhtsKeyEnum.Uri.GetDescription(), "/oauth2/v6/tokens" },

{ PopEhtsKeyEnum.HttpMethod.GetDescription(), PopEhtsKeyEnum.Post.GetDescription() },

};

var hashMapKeyValuePair = HashMapKeyValuePair.Set<string, string>(keyValuePairDictionary);

var popTokenBuilder = new PopTokenBuilder(audience, issuer);

// Act

var popToken = popTokenBuilder.SetEhtsKeyValueMap(hashMapKeyValuePair)

.SignWith(\_privateRsaKeyXml) // XML format

.Build();

var publicRsaSecurityKey = PopTokenBuilderUtils.CreateRsaSecurityKey(\_publicRsaKeyXml); // XML format

var tokenValidationResult = PopTokenBuilderUtils.ValidateToken(popToken, issuer, audience, publicRsaSecurityKey);

// Assert

Assert.IsNotNull(popToken);

Assert.IsNotNull(tokenValidationResult);

Assert.IsTrue(tokenValidationResult.IsValid);

Assert.IsTrue(tokenValidationResult.Claims.Count == 9);

}

/// <summary>

/// Example 2a: Using bearer token to call WebApi endpoint with poptoken (Pem Format)

/// </summary>

[TestMethod]

public void PopTokenBuilder\_Build\_ValidateToken\_ApiExample\_PEMFormat\_Success\_Test()

{

// Arrange

var keyValuePairDictionary = new Dictionary<string, string>

{

{ PopEhtsKeyEnum.ContentType.GetDescription(), PopEhtsKeyEnum.ApplicationJson.GetDescription() },

{ PopEhtsKeyEnum.CacheControl.GetDescription(), PopEhtsKeyEnum.NoCache.GetDescription() },

{ PopEhtsKeyEnum.Authorization.GetDescription(), "Bearer UtKV75JJbVAewOrkHMXhLbiQ11SS" },

{ PopEhtsKeyEnum.Uri.GetDescription(), "/commerce/v1/orders" },

{ PopEhtsKeyEnum.HttpMethod.GetDescription(), PopEhtsKeyEnum.Post.GetDescription() },

{ 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(\_privateRsaKeyPem) // Pem format

.Build();

var publicRsaSecurityKey = PopTokenBuilderUtils.CreateRsaSecurityKey(\_publicRsaKeyPem); // Pem format

var tokenValidationResult = PopTokenBuilderUtils.ValidateToken(popToken, issuer, audience, publicRsaSecurityKey);

// Assert

Assert.IsNotNull(popToken);

Assert.IsNotNull(tokenValidationResult);

Assert.IsTrue(tokenValidationResult.IsValid);

Assert.IsTrue(tokenValidationResult.Claims.Count == 9);

}

/// <summary>

/// Example 2b: Using bearer token to call WebApi endpoint with poptoken (Xml Format - Optional)

/// </summary>

[TestMethod]

public void PopTokenBuilder\_Build\_ValidateToken\_ApiExample\_XmlFormat\_Success\_Test()

{

// Arrange

var keyValuePairDictionary = new Dictionary<string, string>

{

{ PopEhtsKeyEnum.ContentType.GetDescription(), PopEhtsKeyEnum.ApplicationJson.GetDescription() },

{ PopEhtsKeyEnum.CacheControl.GetDescription(), PopEhtsKeyEnum.NoCache.GetDescription() },

{ PopEhtsKeyEnum.Authorization.GetDescription(), "Bearer UtKV75JJbVAewOrkHMXhLbiQ11SS" },

{ PopEhtsKeyEnum.Uri.GetDescription(), "/commerce/v1/orders" },

{ PopEhtsKeyEnum.HttpMethod.GetDescription(), PopEhtsKeyEnum.Post.GetDescription() },

{ 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(\_privateRsaKeyXml) // XML format

.Build();

var publicRsaSecurityKey = PopTokenBuilderUtils.CreateRsaSecurityKey(\_publicRsaKeyXml); // XML format

var tokenValidationResult = PopTokenBuilderUtils.ValidateToken(popToken, issuer, audience, publicRsaSecurityKey);

//Assert

Assert.IsNotNull(popToken);

Assert.IsNotNull(tokenValidationResult);

Assert.IsTrue(tokenValidationResult.IsValid);

Assert.IsTrue(tokenValidationResult.Claims.Count == 9);

}

.