**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**

// Download and install OpenSSL for Windows

// # 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: private-key-pkcs8.pem

var privateKeyPemRsa = File.ReadAllText("private-key-pkcs8.pem")

.Replace(Environment.NewLine, "")

.Replace("KEY-----", "KEY-----" + Environment.NewLine)

.Replace("-----END", Environment.NewLine + "-----END");

// Public Key: public-key.pem

var publicKeyPemRsa = File.ReadAllText("public-key.pem")

.Replace(Environment.NewLine, "")

.Replace("KEY-----", "KEY-----" + Environment.NewLine)

.Replace("-----END", Environment.NewLine + "-----END");

// ClientId

var audience = "JYM89zuJAf3D1N0omc0VzaoehUW5Inn2";

var issuer = audience;

var keyValuePairDictionary = new Dictionary<string, string>

{

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

// Create PopToken using private RSA Key

var popToken = popTokenBuilder.SetEhtsKeyValueMap(hashMapKeyValuePair)

.SignWith(privateKeyPemRsa)

.Build();

Assert.IsTrue(!string.IsNullOrEmpty(popToken));

// Validate PopToken using public RSA Key

var publicRsaSecurityKey = PopTokenBuilderUtils.CreateRsaSecurityKey(publicKeyPemRsa);

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

Assert.IsNotNull(tokenValidationResult);

Assert.IsTrue(tokenValidationResult.IsValid);

Assert.IsTrue(tokenValidationResult.Claims.Count > 0);