Integrated Marketplace
Two-Factor Authentication Specifications

9/8/2017
Southwest Power Pool
## Revision History

<table>
<thead>
<tr>
<th>Date or Version Number</th>
<th>Author</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1.0 7/26/2016</td>
<td>Eric Wyles</td>
<td>Version 1</td>
</tr>
</tbody>
</table>
| Version 1.1 4/15/2017  | Eric Wyles | - Updated screenshots.  
                          |             | - Clarified requirements for Lowercase URI path and Created Timestamp.  
                          |             | - Added logging recommendations.  
                          |             | - Added unit test data section.  |
| Version 1.2 07/05/2017 | Eric Wyles | - Changed references to User ID in the API Token generation documentation to Screen Name for clarity and accuracy.  |
| Version 1.3 08/30/2017 | Eric Wyles | - Updates to “Interactive Two-Factor Login” section to include requirements related to user IP address.  |
Table of Contents

Revision History .................................................................................................................. 1
Introduction .......................................................................................................................... 3
Prerequisites .......................................................................................................................... 4
X509 Client Certificates ....................................................................................................... 5
Interactive Two-Factor Login ............................................................................................... 6
Programmatic Two-Factor Authentication ............................................................................. 7
Generating an API Token for a request .................................................................................. 8
Requirements for a successful API Token authentication .................................................... 11
Sample Code ......................................................................................................................... 12
Java 7 Sample ....................................................................................................................... 13
Python Sample ...................................................................................................................... 15
Perl Example ......................................................................................................................... 16
C# Sample ............................................................................................................................. 17
Introduction

The SPP Integrated Marketplace User Interfaces (UIs) and Application Programming Interfaces (APIs) are protected by Two-Factor Authentication. In order to authenticate, users or applications accessing the UIs and APIs must present a valid x509 certificate and present a second authentication factor.

This document explains the required authentication factors and provides technical details necessary for application developers to create client code.
Prerequisites

Before authenticating with SPP Integrated Marketplace UIs and APIs, the user or application developer must have available these prerequisites.

For interactive access to UIs:
- An x509 client certificate issued by a Certificate Authority (CA) that is trusted by SPP.
- The email address that is configured for the user in the Integrated Marketplace Portal.
- A web browser that is configured to allow first party cookies from the following domains:
  - .spp.org (Production)
  - .itespp.org (MTE)

For programmatic access to APIs:
- An x509 client certificate issued by a Certificate Authority (CA) that is trusted by SPP.
- The Screen Name that was automatically issued to the user when they were created in the Integrated Marketplace Portal.
- An API Key generated for the user in the Integrated Marketplace Portal.
X509 Client Certificates

The first authentication factor that is required to access SPP UIs and APIs is a valid X509 client certificate that is issued by a Certificate Authority (CA) which is trusted by SPP. This authentication factor is always required, for both interactive access to UIs and programmatic access to APIs.

SPP trusts client certificates issued by the following CAs:

- /C=US/ST=MN/L=Minneapolis/O=Open Access Technology International Inc/CN=OATI WebCARES Issuing CA

- /C=US/ST=MN/L=Minneapolis/O=Open Access Technology International Inc/CN=webCARES Issuing CA 2013

Once a client certificate has been obtained, the Local Security Administrator (LSA) for the user must configure the certificate information within the SPP Integrated Marketplace Portal before it can be used for authentication.
Interactive Two-Factor Login

Users that are interacting with SPP UIs in a web browser will be required present a valid x509 client certificate (see previous section). Once a valid certificate has been presented, the user is required to log in through the Integrated Marketplace login page. An example login page is shown below:

Login ID is the user’s email address, as configured by the LSA in the Integrated Marketplace portal. The initial password is also the user’s email address. The password must be changed at first login.

After entering their email address and password and clicking Login, the user will be authenticated to the user interface.

Once the user is authenticated to the Marketplace Portal, the user’s client IP address (as seen from SPP) must remain consistent during the session for authentication to work properly. Network configuration/routing should be verified to ensure that users are connecting to SPP in a consistent way when navigating from one SPP hosted application to another.

For example, if the user logs into the Marketplace Portal over an Internet connection but their connection to the Markets User Interface is over SPP Net, authentication to the Markets User Interface will fail because IP address validation will fail.
Similarly, if some connections are being routed through a proxy before arriving at SPP but other connections are not, authentications may fail. Network routing and configuration must be validated to ensure that network connectivity to the Integrated Marketplace UIs is configured consistently across all applications.

---

**Programmatic Two-Factor Authentication**

Client application code must generate a unique API token for each request to Integrated Marketplace APIs. The API Token is an HMAC SHA512 signed hash that is generated new for each request. To generate an API Token, you will need the following information:

- **API Key** – The Base64 Encoded API Key for the user, issued through the portal. Contact your LSA to obtain your API Key.
  - Example: 6Z56TvN2euFNGKeK0ccessK0tMUA9EiRIpcB1M4aTrI3V/CDVFQx0mbj7vHn6C4p5twjJ3/MYM J3qNJLY6OCw==

- **Screen Name** – The automatically generated screen name for the user, issued through the portal. Contact your LSA to obtain your Screen Name. For reference, this is available by navigating to the “Manage Users” screen, finding the correct user, and looking in the “Screen Name” column.
  - NOTE: This is not the email address.
    - Example: AU01 (as shown in the screenshot below, first column)

![Screen Name Table](image)

- **Service URL** – The URL of the API that is being called.
  - NOTE: This is the URL of the actual service where the requests are being sent (the destination of the HTTP post, in the case of SOAP services).

In most cases, for SOAP services, this is the same as the WSDL address (without the ?wsdl) on the end. However, if you are loading the WSDL from a local or offline location, or otherwise sending the request to a location that doesn’t match the location of the WSDL, the URL where the service is actually being invoked over HTTP is the correct value for the Service URL.
Generating an API Token for a request

A new API token must be generated for each request to an Integrated Marketplace API. Any retry logic should also regenerate a token, because in the case of connectivity failures, it may not be possible to determine if the API token was received by SPP and already marked as used.

To generate an API token, the client must perform the following steps:

1. Create a **Request Nonce** for the message. A nonce is a unique identifier for the message. A nonce must meet the following characteristics:
   a. Must be unique. A given nonce must not have been used before for this user.
   b. Must be 128 characters or less.

   A UUID is recommended for this value.
   
   Example: 461b94ad-6c59-469e-a6c0-bbd03897111b

2. Extract the lowercase ‘path’ portion of the target Service URL. This is the portion of the URI after the host (and port), but before any query parameters (up to but not including the first question mark in the Service URL). This is the **Service URI Path**.

   Example: For ‘https://api.spp.org:443/markets/SomeWebService?parm=value’ the lowercase uri path is ‘/markets/somewebservice’

3. Capture the current UTC time as the **Request Created Timestamp** in the format yyyy-MM-dd’T’HH:mm:ss’Z’. Where:
   a. yyyy = 4 digit year
   b. MM = 2 digit month (January=01, February=02, etc)
   c. dd = 2 digit day of month (01-31)
   d. HH = 2 digit hour (00-23)
   e. mm = 2 digit minute of hour (00-59)
   f. ss = 2 digit second (00-50)

   Example: 2016-04-22T21:03:00Z

NOTE: The Request Created Timestamp must:
   • Be exactly 20 characters.
   • Match the format yyyy-MM-dd’T’HH:mm:ss’Z’ where ‘T’ is the literal character T and ‘Z’ is the literal character Z.
   • Represent a moment in time expressed in UTC time.

4. Concatenate the Request Nonce, Request Created Timestamp, Lowercase Screen Name, and Lowercase Service URI Path to create a string that will be signed to create the API Token. Pseudo code:

   ```
   stringToSign = requestNonce + requestCreatedTimestamp + lowercase(screenName) + lowercase(serviceUriPath)
   ```
Example: 461b94ad-6c59-469e-a6c0-bbd03897111b2016-04-22T21:03:00Zau01/markets/somewebservice

**NOTE:** It is recommended for client applications to log this value for troubleshooting purposes in the case of failed authentications.

5. Sign the string to create an **Hmac Sha 512 hash**.
   a. Base 64 decode the API Key and use the binary API Key to create a signed hash of the string created in Step #4. Base 64 encode the resulting hash to create the API Token

   Pseudo code:
   ```java
   decodedApiKey = Base64Decode(apiKey)
   token = HmacSha512(decodedApiKey, stringToSign)
   hmacHash = Base64Encode(token)
   ```

   Example:
   `/PpMsilxDgf8vfqlozXF55U/Vy9AngjnCw+3OXRZAXhocv0W4An0JJlUw2rrK1N5vvKFck7b1dc+PwkX7KNcHg==`

   **NOTE:** It is recommended for client applications to log this value for troubleshooting purposes in the case of failed authentications.

6. Concatenate the **Request Created Timestamp**, **Request Nonce**, and **Hmac Sha 512 Hash**, separated by hyphens, to form the **API Token**

   Example:
   ```
   apiToken = requestCreated + "-" + requestNonce + "-" + hmacHash
   ```

7. Transmit the Token with the request

   The API Token with the request as an HTTP header. The following table shows the required header name and an example value. The API Key and Screen Name must never be transmitted directly in the request headers, only the header shown below should be used.

<table>
<thead>
<tr>
<th>Header Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-SPP-API-Token</td>
<td>The concatenated token, made up of the Request Created Timestamp, Request Nonce, and Hmac Sha512 Hash, concatenated together and separated by hyphens.</td>
<td><code>2016-04-22T21:03:00Z-461b94ad-6c59-469e-a6c0-bbd03897111b-/PpMsilxDgf8vfqlozXF55U/Vy9AngjnCw+3OXRZAXhocv0W4An0JJlUw2rrK1N5vvKFck7b1dc+PwkX7KNcHg==</code></td>
</tr>
</tbody>
</table>

**Note:** The examples used in this section are the exact inputs that will create the X-SPP-API-Token in the table above. They are shown again below. These inputs can be used to validate that client code
is producing the correct output. Client code, when given these inputs, should produce the exact X-SPP-API-Token shown in the table above.

- **Screen Name**: AU01
- **Base 64 API Key**: 6Z56TvN2euFNGKeK0ccsKuG0tMU9EiRldpB1M4aTrl3V/CDVFX0mbj7vHn6C4p5t
  wjJ/MYMJ3qNJLY6OCw==
- **Nonce**: 461b94ad-6c59-469e-a6c0-bbd03897111b
- **Created Timestamp**: 2016-04-22T21:03:00Z
- **Service URL**: https://api.spp.org:443/markets/SomeWebService?parm=value
**Requirements for a successful API Token authentication**

When the API Token, Created Timestamp, and Request Nonce are transmitted, along with x509 client certificate, the following criteria must be met for a successful authentication:

1. The Screen Name used in the token must be the Screen Name that matches the client certificate that was used.

2. The Service URI Path used in the token must match the Service URI Path where the request is sent.

3. The Nonce used in the token must not have been used before for this user.

4. The Request Created Timestamp used in the token must be no more than 3 minutes in the past or 5 minutes in the future. It is important that the system time be accurate on the computer that is generating tokens. If the system time is more than 3-5 minutes off, the receiving system may interpret the tokens as expired or not yet active.

5. The token must be signed by the same API Key that is associate with the Screen Name and certificate used in the request.
   a. **NOTE:** The API Key and Screen Name are automatically generated by the Marketplace Portal. A user’s Screen Name and API Key will be different for each environment. Application code should make these values configurable per environment (Production will require different values than MTE).
Sample Code

The examples in this section demonstrate how to generate a valid API Token in several programming languages. The input variables are hard-coded in the examples to be consistent with the values shown in Section 4 of this document. Each sample should produce the same Token from Section 4.

The examples in this section are not production ready and should be considered only as an educational tool. In the interest of brevity, they are missing logging, exception handling, generation of unique nonces and timestamps, reading from configuration, etc. Also in these examples, the API Token is printed to the console. In practice, it would be sent in an HTTP header called X-SPP-API-Token.
Java 7 Sample

NOTE: The apache commons codec library is used for Base64 encoding/decoding functionality. Other options may be available (for example, Java 8 includes java.util.Base64).

Code:

```java
package org.spp.sample;

import java.net.URL;
import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;

public class SampleApp {
    public static void main(String[] args) throws Exception {
        // the encoded api key (obtain from LSA)
        String encodedApiKey = "6Z56TvN2euFNGKekG0ccsKuG0tMUA9EiR1pCB1M4aTr13V/CDVFQx0mbj7vHn6C4p5twjlJ/MYMJ3qN3LY6OCw==";

        // the screen name (obtain from LSA)
        String screenName = "AU01";

        // the url of the service to call
        URL endpointUrl = new URL(serviceUrl);
        String uriPath = endpointUrl.getPath();

        // hard coding a nonce here... should be a unique UUID and current timestamp each time
        String nonce = "461b94ad-6c59-469e-a6c0-bbd0389711b";
        String createdTimestamp = "2016-04-22T21:03:00Z";

        // concatenate values together to create stringToSign
        String stringToSign = nonce + createdTimestamp + screenName.toLowerCase() + uriPath.toLowerCase();

        // create the hmac hash
        String hmacHash = signData(encodedApiKey, stringToSign);

        // create the token
        String token = String.format("%s-%s-%s", createdTimestamp, nonce, hmacHash);

        // print the value that would be sent as http header
        System.out.println(String.format("X-SPP-API-Token: %s", token));
    }
}
```
```java
public static String signData(String encodedApiKey, String data) throws Exception {
    String SIGNING_ALGORITHM = "HmacSHA512";
    String ENCODING = "UTF-8";

    // turn the base64 key back into a byte array key
    Base64 b64 = new Base64();
    byte[] decodedKeyBytes = b64.decode(encodedApiKey);
    SecretKey apiKey = new SecretKeySpec(decodedKeyBytes, 0,
    decodedKeyBytes.length, SIGNING_ALGORITHM);

    Mac mac = Mac.getInstance(SIGNING_ALGORITHM);
    mac.init(apiKey);

    byte[] rawHmac = mac.doFinal(data.getBytes(ENCODING));

    return Base64.encodeBase64String(rawHmac);
}
```

Output:

```
X-SPP-API-Token: 2016-04-22T21:03:00Z-461b94ad-6c59-469e-a6c0-bbd03897111b-/PpMsli1xDgf8vfq1ozXF55U/Vy9AngjnCw+30XRZAXhocv0W4An0JthUw2rrK1N5vKFCk7b1dc+PwkX7KNcHg==
```
Python Sample

Code:

```python
import hmac
import hashlib
import base64
import datetime
import sys
import uuid
from urlparse import urlparse

# the encoded api key (obtain from LSA)
apiKey = "6Z56TvN2euFNGKeKGOcC5Kv0U07a4Trl3V/CDVFQx0mbj7Vh6C4pStwjLJ/MYM3qNJLY60Cw=="

# the screen name (obtain from LSA)
screenName = "AU01"

# the url of the service to call

# hard coding a nonce here... should be a unique UUID and current
timestamp each time
nonce = "461b94ad-6c59-469e-a6c0-bbd0389711b"
createdTimestamp = "2016-04-22T21:03:00Z"

# parse the url
parsedUrl = urlparse(url);

# concatenate values together to create stringToSign
stringToSign = nonce + createdTimestamp + screenName.lower() + parsedUrl.path.lower()

# decode the base64 api key to get the original binary key
decodedApiKey = base64.b64decode(apiKey)

# create the hmac hash
hmacBinary = hmac.new(decodedApiKey, stringToSign, hashlib.sha512)
hmacEncoded = base64.b64encode(hmacBinary.digest())

# print the values that would be sent as http headers
print 'X-SPP-API-Token: ' + token
```

Output:

```
X-SPP-API-Token: 2016-04-22T21:03:00Z-461b94ad-6c59-469e-a6c0-bbd0389711b-/PpMsF1xgj8vFq9oZK55 UFy9ANgjnCw+30XRAHocv0W4An0J31Uw2rrK1N5vKFCk7b1dC+PwKX7KNCg=
```
Perl Example

Code:

use Digest::SHA qw(hmac_sha512_base64);
use URI::URL;

# the encoded api key (obtain from LSA)
$apiKey = "6Z56TvN2euFNGKeKG0ccsKuG0tMUA9EiR1pCB1M4aTr73V/CDVFQx0mbj7vHn6C4p5twjlJ/MYM3qN3LY60Cw==";

# the screen name (obtain from LSA)
$screnName = "AU01";

# the url of the service to call

# hard coding a nonce here... should be a unique UUID and current
# timestamp each time
$nonce = "461b94ad-6c59-469e-a6c0-bbd03897111b";
$createdTimestamp = "2016-04-22T21:03:00Z";

# concatenate values together to create stringToSign
$stringToSign = $nonce . $createdTimestamp . lc($screnName) . lc($url->path);

# decode the base64 api key to get the original binary key
$decodedApiKey = MIME::Base64::decode($apiKey);

# create the hmac hash
$hmacHash = hmac_sha512_base64($stringToSign, $decodedApiKey);

# perl does not properly pad base 64 encoded values to the proper length
while (length($hmacHash) % 4) {
    $hmacHash .= '=';
}

# create the token
$token = $createdTimestamp . "-" . $nonce . "-" . $hmacHash;

# print the value that would be sent as http header
print "X-SPP-API-Token: $token\n";

Output:

X-SPP-API-Token: 2016-04-22T21:03:00Z-461b94ad-6c59-469e-a6c0-bbd03897111b-/PpMsilxDgf8vFqIoXZ55U/Vy9AngjnhCw+3OXRzAxcv0W4An0JJ1Uw2rK1N5vvKFCk7blc+PwX7KNCg=

Integrated Marketplace Two-Factor Authentication
Specifications

16
C# Sample

Code:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // the encoded api key (obtain from LSA)
            String encodedApiKey = "6Z56TvN2euFNGkekG0ccsKuG0tMUAA9EiR1pcB1M4aTrl3V/CDVFQx0mbj7vHn6C4p5twj1J/MYM3qNJLY6OCw==";

            // the screen name (obtain from LSA)
            String screenName = "AU01";

            // the url of the service to call
            String urlPath = url.AbsolutePath;

            // hard coding a nonce here... should be a unique UUID and current timestamp each time
            String nonce = "461b94ad-6c59-469e-a6c0-bbd03897111b";
            String createdTimestamp = "2016-04-22T21:03:00Z";

            // concatenate values together to create stringToSign
            String stringToSign = nonce + createdTimestamp + screenName.ToLower() + url.AbsolutePath.ToLower();

            // decode the base64 api key to get the original binary key
            byte[] decodedApiKey = Convert.FromBase64String(encodedApiKey);

            // sign the data
            System.Text.UTF8Encoding encoding = new System.Text.UTF8Encoding();
            byte[] keyBytes = encoding.GetBytes(decodedApiKey);
            HMACSHA512 hmacsha512 = new HMACSHA512(decodedApiKey);
            byte[] messageBytes = encoding.GetBytes(stringToSign);
            byte[] hashmessage = hmacsha512.ComputeHash(messageBytes);
            String hmacHash = Convert.ToBase64String(hashmessage);

            String token = createdTimestamp + "-" + nonce + "-" + hmacHash;

            Console.WriteLine("X-SPP-API-Token: " + token);
        }
    }
}
```
Output:

X-SPP-API-Token: 2016-04-22T21:03:00Z-461b94ad-6c59-469e-a6c0-bbd03897111b-/PpMsilxDgf8vfq1ozXF55U/Vy9AngjnCw+30XRZAXhocvOw4An0JJlUw2rrK1N5vvKFck7b1dc+PwkX7KNcHg
==
The table below provides several examples of token generation. These examples can be used by application developers to test their application code. Each example uses the following parameters to generate tokens for a different Service URL:

- **Screen Name**: EX123456789
- **Api Key**: 6Z5TvN2euFNGKe6Gc8c5KuG08U9AEIRIpBC1M4aTrl3V/CDVFQx0mbj7vHn6C4p5twjJl3/MYMJ3qJLY60Cw==
- **Nonce**: 1265fb1e-bbc3-453a-be40-9e2a808cbaae
- **Created Timestamp**: 2017-04-15T17:08:57Z

In practice, the Screen Name and Api Key will be configurable in the client application, and the Nonce and Created Timestamp would be generated new for each request. The table below shows several different hypothetical Service URLs and shows the intermediate values (Lowercase URI Path and Token Before Signing) as well as the resulting token that would be transmitted in the X-SPP-API-Token header.

Application developers may use the values above and the table below to create unit tests that validate client application code before integrating with SPP.

<table>
<thead>
<tr>
<th>Service URL</th>
<th>Lowercase URI path</th>
<th>Token Before Signing</th>
<th>X-SPP-API-Token Header</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://api.spp.org/test-ws">https://api.spp.org/test-ws</a></td>
<td>/test-ws</td>
<td>1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
<td>2017-04-15T17:08:57Z-1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
</tr>
<tr>
<td><a href="https://api.spp.org:443/test-ws">https://api.spp.org:443/test-ws</a></td>
<td>/test-ws</td>
<td>1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
<td>2017-04-15T17:08:57Z-1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
</tr>
<tr>
<td><a href="https://api.spp.org/test-ws">https://api.spp.org/test-ws</a></td>
<td>/test-ws/</td>
<td>1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
<td>2017-04-15T17:08:57Z-1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
</tr>
<tr>
<td><a href="https://api.spp.org:443/test-ws">https://api.spp.org:443/test-ws</a></td>
<td>/test-ws/</td>
<td>1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
<td>2017-04-15T17:08:57Z-1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
</tr>
<tr>
<td><a href="https://api.spp.org/test-ws?wsdl">https://api.spp.org/test-ws?wsdl</a></td>
<td>/test-ws/</td>
<td>1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
<td>2017-04-15T17:08:57Z-1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
</tr>
<tr>
<td><a href="https://api.spp.org:443/test-ws?wsdl">https://api.spp.org:443/test-ws?wsdl</a></td>
<td>/test-ws/</td>
<td>1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
<td>2017-04-15T17:08:57Z-1265fb1e-bbc3-453a-be40-9e2a808cbaae-wafGmKc3UU++g0ssD6ZEcC0XARhItd5SbJUpwRhda2drzu0/7XP-s8jo15tn30LFFzv1mjmrmBADZGCLEH14K6g==</td>
</tr>
<tr>
<td>Request URL</td>
<td>Response URL</td>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>
|---------------------------------------------------------------------------|-------------------------------------------------------------------------------|-----------------------------------------------------------------------------

The tables above summarize the responses to the requests, providing insight into the services offered by the Southwest Power Pool, Inc. for two-factor authentication. The responses include URLs, parameters, and service descriptions tailored to facilitate the authentication process.
<table>
<thead>
<tr>
<th>URL</th>
<th>Path</th>
<th>Timestamp</th>
<th>MD5 Hash</th>
</tr>
</thead>
</table>