WSAA Manual del Desarrollador

Publicación 20.2.19

soporte-ws-testing@afip.gob.ar

20.2.19

Índice general

1. Introducción 1

2. Herramientas útiles para elprogramador3
2.1. OpenSSL....3
2.2. SoapUI....3
2.3. WireShark....3
2.4. Fiddler....3

3. Arquitectura general de los web services deAFIP5
3.1. Webservices SOAP denegocio(WSN)....5
3.2. WebService de Autenticación y Autorización(WSAA)....5
3.3. Flujogeneral....5

4. Obtención de certificado digital paratesting/homologación7

5. Creación del CMS firmado usandoOpenSSL9
5.1. Solicitud de ticketdeacceso....9
5.2. CreacióndelCMS....9

6. Usando SoapUI para probar los webservices11
6.1. Creación delosproyectos....11
6.2. Obtención de un ticket de accesodelWSAA....13
6.3. Análisis del token recibidodelWSAA....14
6.4. Enviar request al web servicedenegocio....15

7. Un cliente de WSAA enPowerShell17
7.1. Descripción....17
7.2. Declaracióndeargumentos....17
7.3. Inicializacióndelscript....18
7.4. Armar el XML del ticketdeacceso....18
7.5. Firmar el CMS y encodearloenBase64....18
7.6. Invocar al método LoginCmsdelWSAA....19
7.7. Ejemplodeejecución....19

8. Un cliente de WSAA enPHP21
8.1. Descripción....21
8.2. Valoresdeconfiguración....21
8.3. Función para crearelTRA....21
8.4. Función para firmarelmensaje....22
8.5. Ejecución del método LoginCmsdelWSAA....22
8.6. Función para mostrar cómoseusa....22
8.7. Programaprincipal....23
8.8. Ejemplo de ejecución en líneadecomandos....23

9. Usando curl para probar los webservices25
9.1. Obtener un ticketdeacceso....25
9.2. Consumir el método*dummy*de*ws_sr_constancia_inscripcion*....25

10. FAQ(Preguntas/Respuestasfrecuentes)27
10.1. Cuáles son los ambientesdelWSAA....27
10.2. Certificado no emitido porACdeconfianza....27
10.3. Computador no autorizado a acceder a los serviciosdeAFIP....27
10.4. Computador no autorizado a accederalservicio....27
10.5. DN delsourceinválido....28
10.6. El CEE ya posee unTAvalido para el acceso alWSNsolicitado....28
10.7. El tiempo de expiración es inferior a lahoraactual....28
10.8. Firma inválida o algoritmonosoportado....28
10.9. GenerationTime en el futuro o más de 24 horasdeantiguedad....28

10.10. No se ha podido interpretar el XML contraelschema.....29

11. Linksdeinterés **31**

Introducción

El presente manual tiene como propósito guiar al programador que debe consumir el Web Service de Autenticación/Autorización (WSAA). Este manual es un complemento de “Especificación Técnica del Web Service de Au- tenticación y Autorización”.

Por comentarios y sugerencias de este manual, por favor contactar asoporte-ws-testing@afip.gob.ar

Herramientas útiles para el programador

En nuestra experiencia desarrollando y soportando aplicaciones relacionadas con web services y/o networkingen general, hemos encontrado que es muy práctico (o casi necesario) disponer de algunas herramientas específicas para el manejo de certificados digitales y criptografía en general, para capturar y analizar el tráfico de red y para probar/depurar web services SOAP.

En este sentido, podemos sugerir las siguientes herramientas, todas gratis, la mayoría disponibles para múltiples sistemas operativos. Todas estas herramientas son de libre disponibilidad y en nuestra experiencia y la de otros usuarios, han probado ser muy útiles, pero no asumimos ninguna responsabilidad sobre su funcionalidad ni estamos en condiciones de dar soporte ni contestar consultas sobre su uso.

2.1 OpenSSL

Para el manejo de certificados digitales y criptografía de clave pública: OpenSSL. Esta herramienta forma parte estándar de la mayoría del sistemas operativos Unix/GNU/Linux. En el caso de Windows, se puede obtener una versión binaria en:https://slproweb.com/products/Win32OpenSSL.html

2.2 SoapUI

Para probar/depurar web services SOAP, una herramienta que ha probado ser muy útil. Está disponible para ambientes Linux y Windows. Esta herramienta puede obtenerse en: www.soapui.org

2.3 WireShark

Para capturar/analizar tráfico de red en general, un popular “sniffer” es: WireShark. Esta herramienta suele formar parte de muchas distribuciones GNU/Linux, pero también puede obtenerse una versión para Windows en: www.wireshark.org

2.4 Fiddler

Para capturar/analizar tráfico HTTP en particular, una herramienta más limitada pero más simple de utilizar es: Fiddler. Esta herramienta sólo está disponible para entornos Windows y se puede descargar en: www.fiddler2.com/fiddler2

Arquitectura general de los web services de AFIP

3.1 Web services SOAP de negocio (WSN)

El intercambio de información entre los web services de negocio (WSN) de AFIP y los sistemas externos a la AFIP se implementa a través de Web Services SOAP (Simple Object Access Protocol) sobre transporte HTTPS (puerto 443). No se requiere el establecimiento de canales especiales de comunicaciones ni VPNs. SOAP es un protocolo estándar que define cómo dos diferentes procesos remotos pueden comunicarse por medio de intercambio de datos en formato XML.

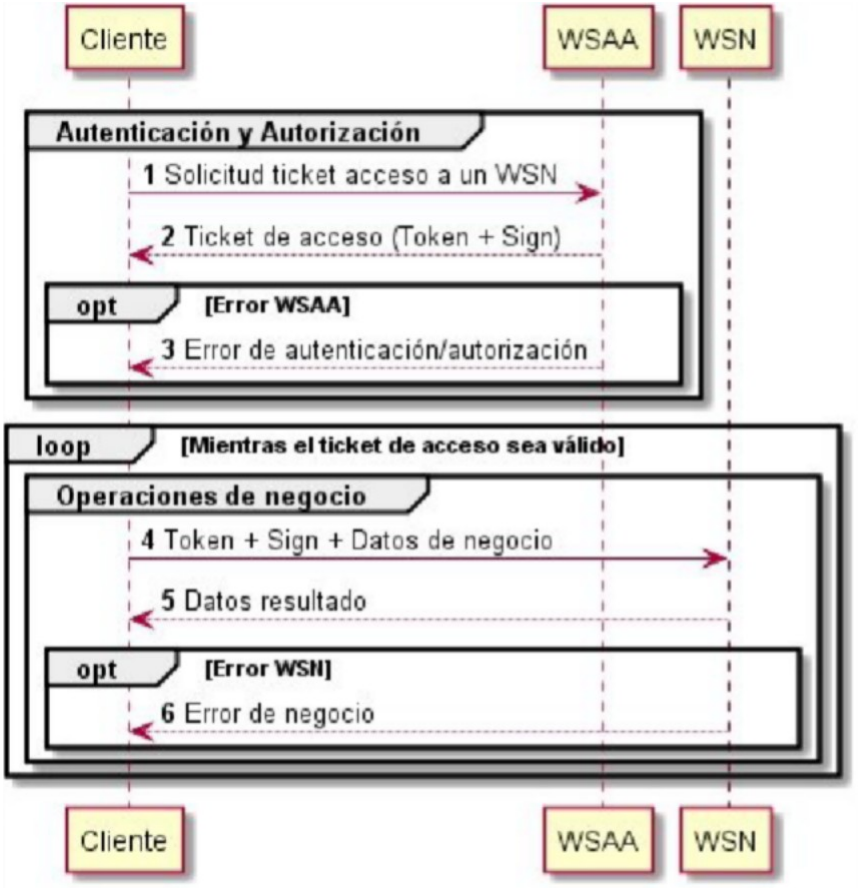
Ejemplos de WSN: Facturación electrónica, consulta al padrón de contribuyentes, etc.

3.2 Web Service de Autenticación y Autorización (WSAA)

Para que un sistema externo pueda comunicarse con un WSN, primero debe autenticarse y ser autorizado. Para eso existe el Web Service de Autenticación y Autorización (WSAA), que es un web service SOAP que autentica a las aplicaciones clientes y les concede permiso de acceso a cada WSN mediante el otorgamiento de un Ticket de Acceso (TA).

3.3 Flujo general

El diagrama siguiente muestra el flujo entre todos los servicios:



1. Para autenticarse, primero la aplicación cliente solicita un TA al WSAA. Dicha solicitud es un mensaje firmado usando un certificado digital. Este certificado digital identifica a la aplicación cliente y tiene asociada una lista de WSN autorizados a los cuales puede acceder.
2. El WSAA analiza la solicitud y otorga el TA. Cada TA es válido para acceder a un WSN en particular y tiene una validez limitada en el tiempo. La aplicación cliente será responsable de presentar al WSN el TA otorgado por el WSAA, de lo contrario el WSN rechazará su solicitud de acceso. El TA consiste en dos componentes, denominadas Token y Sign.
3. Si el WSAA rechaza la solictud, devolverá una explicación del motivo del rechazo.
4. El cliente del WSN tomará el Token y Sign, y los enviará junto con los datos de negocio en cada solicitud que le envíe al WSN. Cuando el TA expire en el tiempo, la aplicación cliente volverá a solicitar un nuevo TA.
5. El WSN responderá un resultado exitoso.
6. Si se produce algún error, el WSN responderá informando del error.

Obtención de certificado digital para testing/homologación

Para los desarrolladores que deseen implementar aplicaciones que consuman de los web services SOAP de testing/homologacion de AFIP, actualmente se dispone de la herramienta WSASS (Autogestion de certificados para Servicios Web en los ambientes de homologacion). La misma es una aplicacion online que permite al programador gestionar sus certificados para hacer pruebas.

Los instructivos son:

- Como adherirse al WSASS mediante el Administrador de Relaciones de Clave Fiscal. Descargar PDF:http://www.afip.gob.ar/ws/WSASS/WSASS_como_adherirse.pdf
- Manual del Usuario del WSASS. Descargar PDF:http://www.afip.gob.ar/ws/WSASS/WSASS_manual.pdf

NOTA

Los detalles de cómo usar la aplicación WSASS está fuera del alnace del presente documento. Se remite al lector a los instructivos indicados arriba.

Creación del CMS firmado usando OpenSSL

En esta sección mostraremos cómo crear un CMS firmado, usando OpenSSL en la línea de comandos. Vamos a asumir que ya tenemos los archivos:

- *MiCertificado2019.pem* (certificado de testing creado en la aplicación WSASS).
- *MiPrivada2019.key* (clave privada usada, correspondiente a la solicitud del certificado).
- *MiLoginTicketRequest.xml* (la solicitud de ticket de acceso).

5.1 Solicitud de ticket de acceso

La solicitud de ticket de acceso es un documento XML indicando el identificador del web service de negocio al que se solicita acceder.

Ejemplo: 'ws_sr_constancia_inscripcion' es el identificador del servicio de Consulta de la Constancia de Inscrip- ción de Padrón, que permite consultar los datos de la constancia de un contribuyente registrado en el Padrón de AFIP.

A continuación vemos el contenido del archivo *MiLoginTicketRequest.xml*:

```
<loginTicketRequest>
        <header>
                <uniqueId>190926</uniqueId>
                <generationTime>2019-09-26T10:09:20</generationTime>
                <expirationTime>2019-09-26T10:29:20</expirationTime>
        </header>
        <service>ws_sr_constancia_inscripcion</service>
</loginTicketRequest>
```

5.2 Creación del CMS

Para crear el CMS con OpenSSL hacemos:

```
D:\> openssl cms -sign -in MiLoginTicketRequest.xml
                -out MiLoginTicketRequest.xml.cms -signer MiCertificado2019.pem
                -inkey MiPrivada2019.key -nodetach -outform PEM
```

El archivo *MiLoginTicketRequest.xml.cms* creado contiene algo similar a:

```
-----BEGIN CMS-----
MIIG2AYJKoZIhvcNAQcCoIIGyTCCBsUCAQExDTALBglghkgBZQMEAgEwgf8GCSq
G
SIb3DQEHAaCB8QSB7jxsb2dpblRpY2tldFJlcXVlc3Q+PGhlYWRlcj48dW5pcXV
l
SWQ+MTkwOTI2PC91bmlxdWVJZD48Z2VuZXJhdGlvblRpbWU+MjAxOS0wOS0yNlQ
```

```
. . .
AIAwDQYIKoZIhvcNAwICAUAwBwYFKw4DAgcwDQYIKoZIhvcNAwICASgwDQYJKoZ
I
hvcNAQEBBQAEggEADDXstf5M89wT9IZnAz2N/Yn0r7aGhCQ30U9UAb6BkLfXta6
F
dbCqDbqO7ekUyO/GjSR9R6hMGsxn7lg9U+JZ+yCVfPENaZItOAOPnfJ3gWih7U5
c
loPLAIC1WvjSxoKCn81NG5ZKxNGmik9K2pKv86BdtN/1BABDFak5QO8WcTo2Wpt
c
```

NOTAS

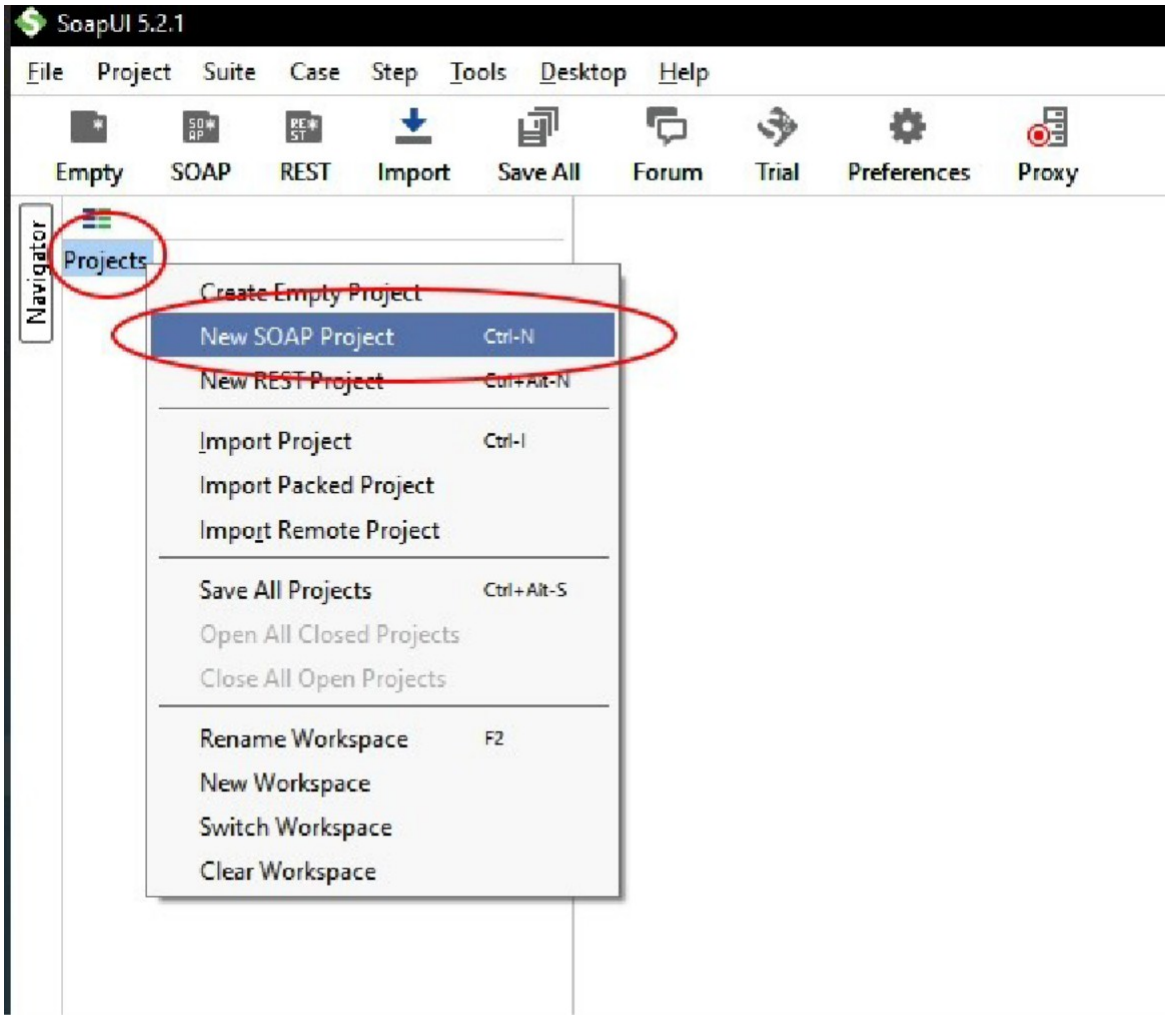
El texto comprendido entre las líneas *—–BEGIN CMS—–* y *—–END CMS—–* es el mensaje firmado que deberemos enviar al método *loginCms* del WSAA (en el parámetro *in0*).

Usando SoapUI para probar los web services

En esta sección mostraremos cómo usar la herramienta SoapUI para hacer pruebas contra los web services. A modo de ejemplo, usaremos SoapUI para probar el web service de negocio de consulta de constancia de inscripción (ws_sr_constancia_inscripcion).

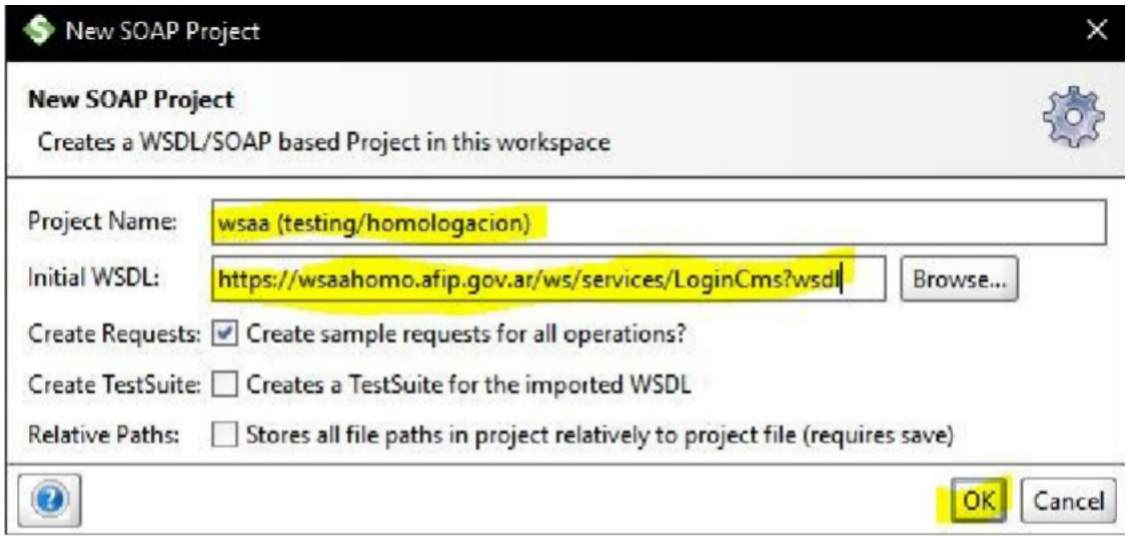
6.1 Creación de los proyectos

Empezaremos creando en el SoapUI un proyecto para cada servicio a usar. Primero vamos a crear un proyecto SOAP nuevo para el WSAA de testing/homologación:



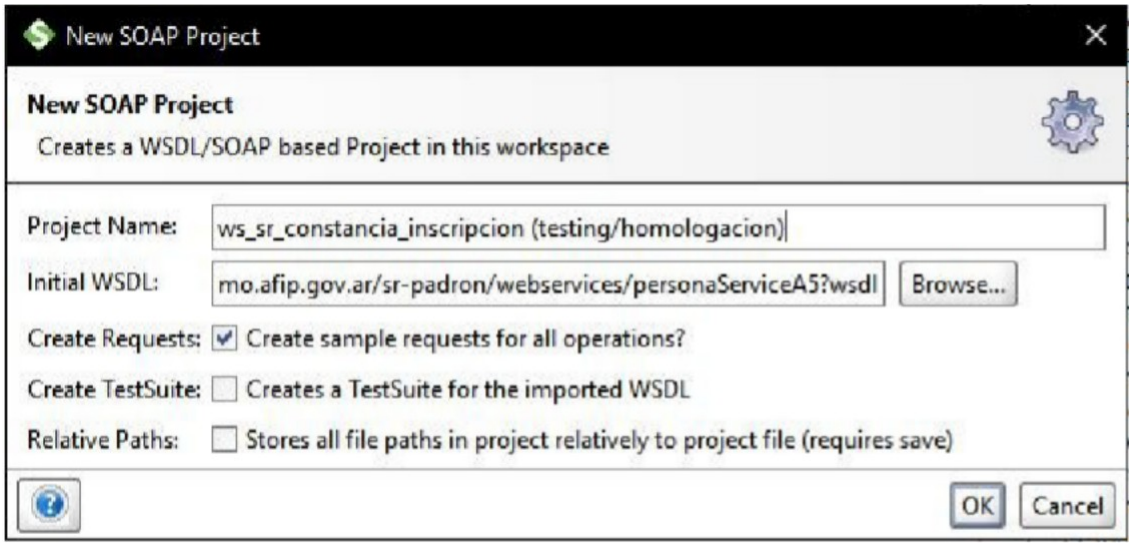
Indicaremos la URL del WSDL del WSAA y un nombre para nuestro proyecto. La URL a usar es:

```
https://wsaahomo.afip.gov.ar/ws/services/LoginCms?wsdl
```

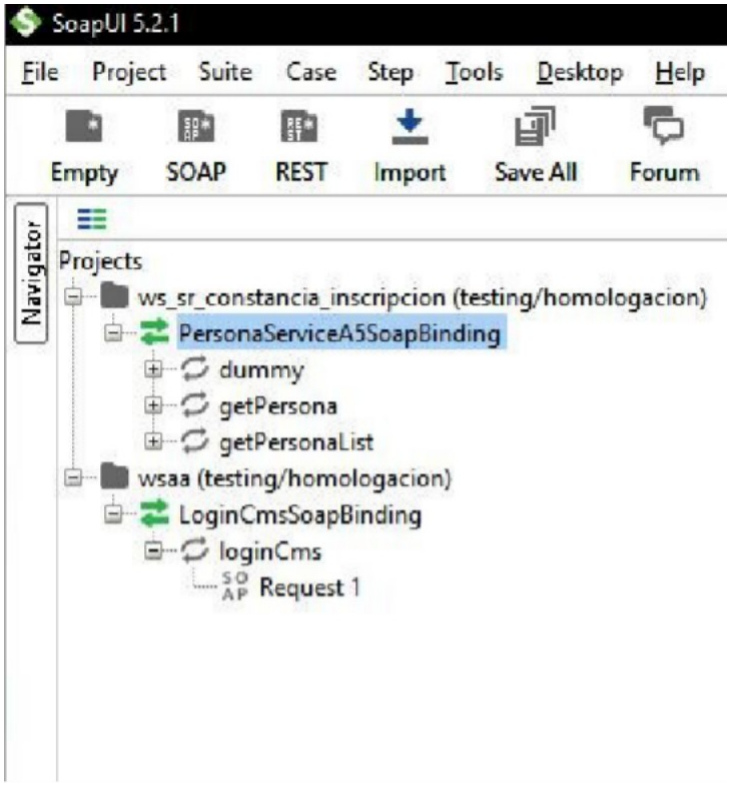


Similarmente creamos otro proyecto para *ws_sr_constancia_inscripcion*. La URL del WSDL a usar es:

```
https://awshomo.afip.gov.ar/sr-padron/webservices/personaServiceA5?wsdl
```



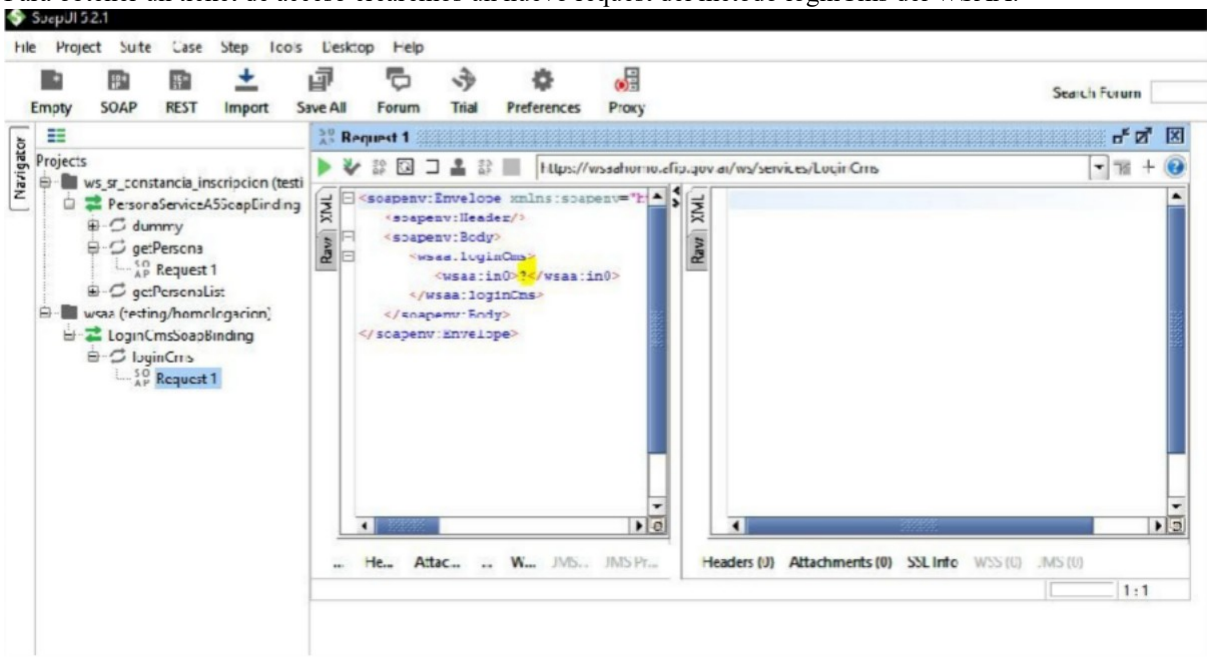
El navegador de proyectos se verá así:



Observar que en cada servicio se generaron ejemplos de requests para cada uno de los métodos en el servicio.

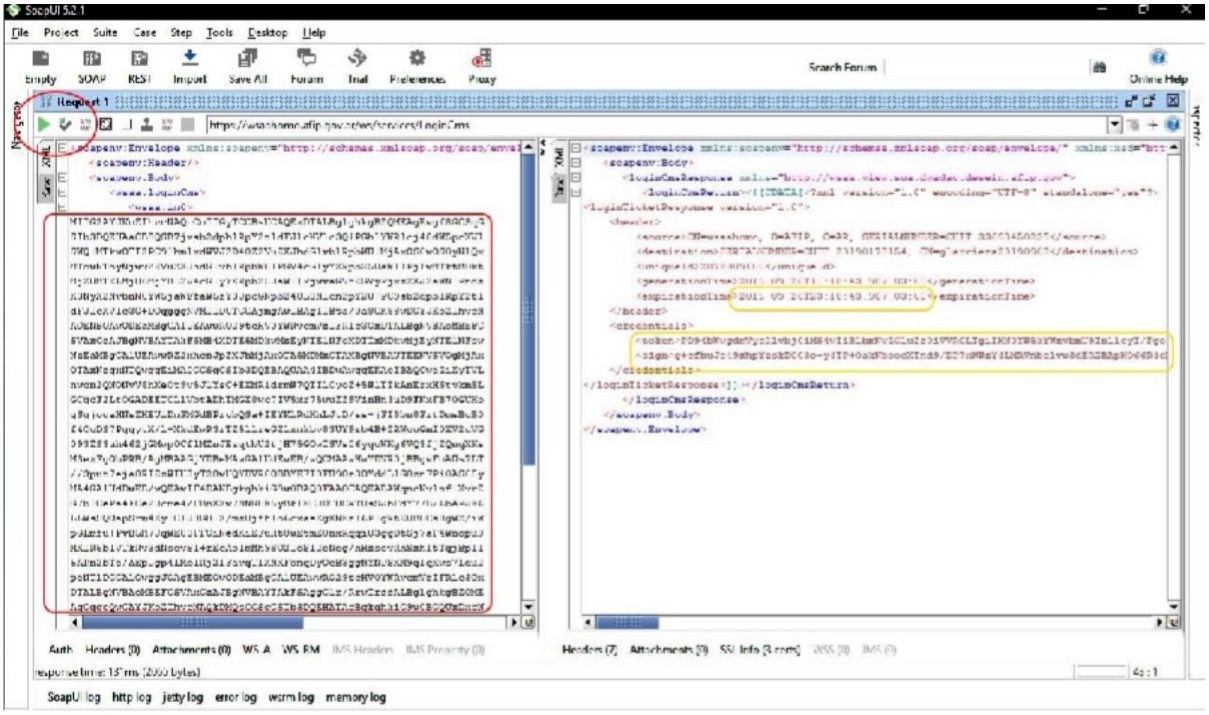
6.2 Obtención de un ticket de acceso del WSAA

Para obtener un ticket de acceso crearemos un nuevo request del método loginCms del WSAA:



Observar que el método loginCms tiene un sólo parámetro, el elemento *in0*, que va a ser un mensaje CMS firmado con una solicitud de acceso.

Vamos a utilizar el CMS que hemos creado antes con OpenSSL. Copiamos y pegamos el texto del CMS en el valor del elemento *in0* del request y pulsamos el botón verde para ejecutar el request:



La respuesta del WSAA aparece a la derecha. Podemos observar en el ticket de acceso recibido: La fecha de expiración, el token y el sign. Los valores de token y sign vamos a usarlos después para acceder al web service de negocio.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<loginTicketResponse version="1.0">
        <header>
                <source>CN=wsaahomo, O=AFIP, C=AR,
                        SERIALNUMBER=CUIT 33693450239</source>
                <destination>SERIALNUMBER=CUIT 20190178154,
                        CN=glarriera20190903</destination>
                <uniqueId>3866895167</uniqueId>
                <generationTime>2019-09-26T13:56:14.467-03:00</generationTime>
                <expirationTime>2019-09-27T01:56:14.467-03:00</expirationTime>
        </header>
        <credentials>
                <token>PD94bWwgdmVyc2lv . . . go8L3Nzbz4K</token>
                <sign>Urp5dbarIb8m5y . . . SEzSeon1W7ys=</sign>
         </credentials>
</loginTicketResponse>
```

6.3 Análisis del token recibido del WSAA

El token puede analizarse usando el comando *base64*. De esta forma puede verse el XML contenido. Por ejemplo:

```
$ cat token
PD94bWwgdmVyc2lvbj0i . . . bj4KPC9zc28+Cg==

$ base64 -d token
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sso version="2.0">
        <id src="CN=wsaahomo, O=AFIP, C=AR, SERIALNUMBER=CUIT 33693450239"
        dst="CN=wsfe, O=AFIP, C=AR" unique_id="1540869723" gen_time="1573830140"
        exp_time="1573873400"/>
        <operation type="login" value="granted">
                <login entity="33693450239" service="wsfe"
```

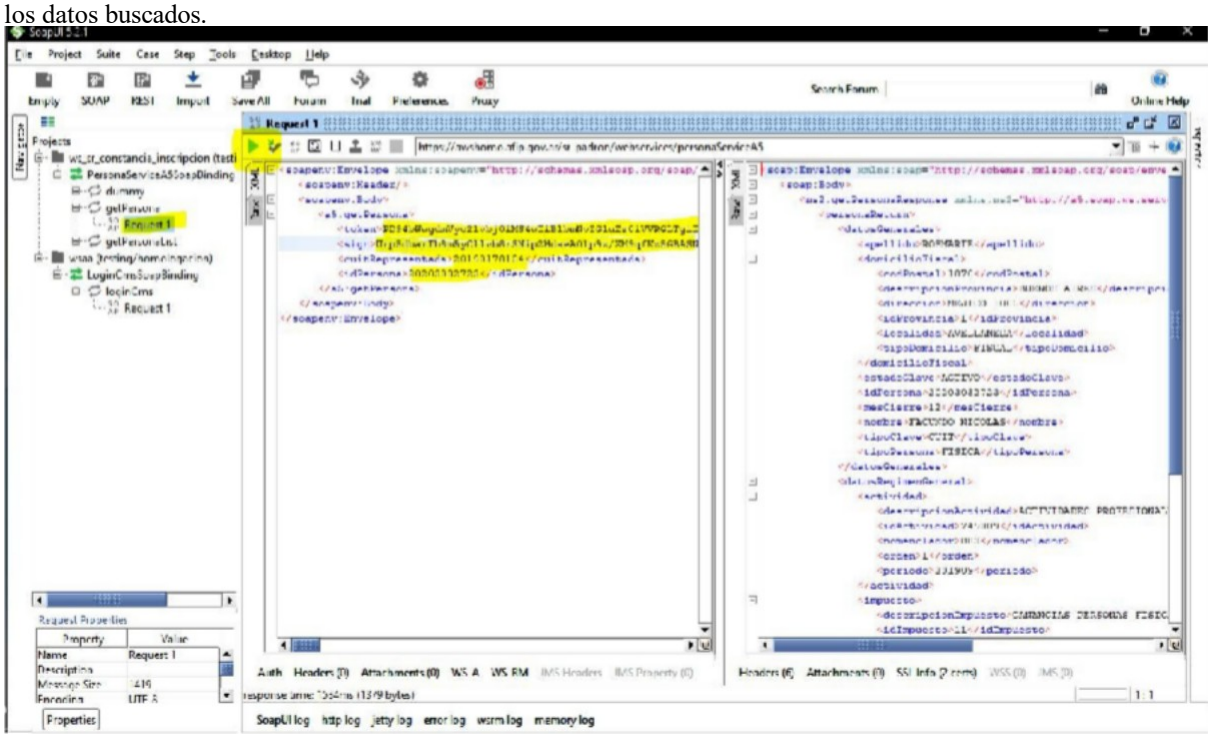
```
                uid="SERIALNUMBER=CUIT 20190178154, CN=glarriera20181029"
                authmethod="cms" regmethod="22">
                        <relations>
                                <relation key="20190178154" reltype="4"/>
                                <relation key="30333333313" reltype="4"/>
                        </relations>
                </login>
        </operation>
</sso>

$ date -d @1573873400
Sat Nov 16 00:03:20 ART 2019
```

Algunos datos que se pueden observar en el token:

- El servicio a acceder: “wsfe”.
- Las CUITs autorizadas a actuar como REPRESENTADAS: 20190178154 y 30333333313.
- La fecha de expiración (exp_time) mostrada en formato epoch.

6.4 Enviar request al web service de negocio

Vamos a hacer un request al método *getPersona* del servicio *ws_sr_constancia_inscripcion*. Copiamos y pegamos los valores de token y sign que habíamos obtenidos del WSAA, escribimos la CUIT que nos representa y la CUIT a buscar. Luego de ejecutar la consulta recibimos la respuesta de *ws_sr_constancia_inscripcion* con los datos buscados.



Un cliente de WSAA en PowerShell

Este ejemplo es un script Microsoft PowerShell para solicitar un ticket de acceso al WSAA. Puede ejecutarse en una línea de comandos PowerShell en Windows.

7.1 Descripción

```
# wsaa-cliente-noopenssl.ps1
# Autor: Gustavo Larriera (AFIP, 2019)
#
# Descripción:
#
#    Ejemplo de cliente del WSAA.
#    Consume el metodo LoginCms ejecutando desde la Powershell de
Windows. #   Muestra el login ticket response.
#
# Argumentos de linea de comandos:
#
#   $Certificado: Ruta del certificado firmante a usar (formato .pfx,
.p12) #      $Password: Clave del certificado
#   $ServicioId: ID de servicio a
acceder #    $OutXml: Archivo TRA a crear
#   $OutCms: Archivo CMS a crear
#   $WsaaWsdl: URL del WSDL del WSAA
```

7.2 Declaración de argumentos

```
[CmdletBinding()]
Param(
        [Parameter(Mandatory=$False)]
        [string]$Certificado="D:\wsaacliente-
        posh\MiCertificado.p12",

        [Parameter(Mandatory=$False)] [string]
        $Password="PasswordDeMiCertificado",

        [Parameter(Mandatory=$False)] [string]
        $ServicioId="wsfe",

        [Parameter(Mandatory=$False)] [string]
        $OutXml="LoginTicketRequest.xml",

        [Parameter(Mandatory=$False)] [string]
        $OutCms="LoginTicketRequest.xml.cms",
```

```
        [Parameter(Mandatory=$False)]
        [string]$WsaaWsdl = "https://wsaahomo.afip.gov.ar/ws/services/LoginCms?WSDL"
)
```

7.3 Inicialización del script

```
# Acción preferida si hay algún error
$ErrorActionPreference = "Stop"

# Usar biblioteca de criptografía de .NET Framework
Add-Type -Path "C:\Windows\Microsoft.NET\Framework64\v4.0.30319
        \System.Security.dll"
```

7.4 Armar el XML del ticket de acceso

```
$dtNow = Get-Date
$xmlTA = New-Object System.XML.XMLDocument
$xmlTA.LoadXml('<loginTicketRequest><header><uniqueId></uniqueId>
        <generationTime></generationTime><expirationTime></expirationTime>
        </header><service></service></loginTicketRequest>')
$xmlUniqueId = $xmlTA.SelectSingleNode("//uniqueId")
$xmlGenTime = $xmlTA.SelectSingleNode("//generationTime")
$xmlExpTime = $xmlTA.SelectSingleNode("//expirationTime")
$xmlService = $xmlTA.SelectSingleNode("//service")
$xmlGenTime.InnerText = $dtNow.AddMinutes(-10).ToString("s")
$xmlExpTime.InnerText = $dtNow.AddMinutes(+10).ToString("s")
$xmlUniqueId.InnerText = $dtNow.ToString("yyMMddHHMM")
$xmlService.InnerText = $ServicioId
$seqNr = Get-Date -UFormat "%Y%m%d%H%S"
$xmlTA.InnerXml | Out-File $seqNr-$OutXml -Encoding ASCII
```

7.5 Firmar el CMS y encodearlo en Base64

```
$cer = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2
$cer.Import($Certificado, $Password, 0)
$encoding = [System.Text.Encoding]::UTF8 [System.Byte[]]
$msgBytes = $encoding.GetBytes($xmlTA.InnerXml)
[System.Security.Cryptography.Pkcs.ContentInfo]$contentInfo = [System
        .Security.Cryptography.Pkcs.ContentInfo]::new($msgBytes)
[System.Security.Cryptography.Pkcs.SignedCms]$signedCms = [System.Security
        .Cryptography.Pkcs.SignedCms]::new($contentInfo)
[System.Security.Cryptography.Pkcs.CmsSigner]$cmsSigner = [System.Security
        .Cryptography.Pkcs.CmsSigner]::new([System.Security.Cryptography.Pkcs
        .SubjectIdentifierType]::IssuerAndSerialNumber, $cer)
$cmsSigner.IncludeOption = [System.Security.Cryptography.X509Certificates
        .X509IncludeOption]::EndCertOnly
$signedCms.ComputeSignature($cmsSigner) [System.Byte[]]
$encodedSignedCms = $signedCms.Encode()
$signedCmsBase64 = [System.Convert]::ToBase64String($encodedSignedCms)
```

7.6 Invocar al método LoginCms del WSAA

```
try
{
   $wsaa = New-WebServiceProxy -Uri $WsaaWsdl -ErrorAction Stop
   $wsaaResponse = $wsaa.loginCms($signedCmsBase64)
   $wsaaResponse > $seqNr-loginTicketResponse.xml
   $wsaaResponse
}
catch
{
   $errMsg = $_.Exception.Message
   $errMsg > $seqNr-loginTicketResponse-ERROR.xml
   $errMsg
}
```

7.7 Ejemplo de ejecución

Para este ejemplo usaremos un certificado *MiCertificado2019.pfx* en formato PKCS#12 (.pfx .p12), creado con OpenSSL (a partir del certificado *MiCertificado2019.pem* y la clave privada *MiPrivada2019.key*. OpenSSL va a solicitarnos que definamos una password para el archivo PFX (se ingresa 2 veces para confirmarla):

```
PS> openssl pkcs12 -export -out MiCertificado2019.pfx
                -inkey MiPrivada2019.key -in MiCertificado2019.pem
Enter Export Password: glarriera2019
Verifying - Enter Export Password: glarriera2019
PS>
```

NOTA

Estamos asumiendo que el certificado de testing *MiCertificado2019.pem* fué previamente creado y autoriza- do a acceder al servicio *ws_sr_constancia_inscripcion*, usando la aplicación WSASS.

Ahora ejecutaremos el script PowerShell para solicitar un ticket de acceso para el web service *ws_sr_constancia_inscripcion*:

```
PS> .\wsaa-cliente-noopenssl.ps1
                -Certificado "D:\wsaacliente-posh\MiCertificado2019.pfx"
                -Password "glarriera2019" -ServicioId "ws_sr_constancia_inscripcion"
```

La ejecución muestra la siguiente salida (el ticket de acceso otorgado):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<loginTicketResponse version="1.0">
<header>
<source>CN=wsaahomo, O=AFIP, C=AR, SERIALNUMBER=CUIT 33693450239</source>
<destination>SERIALNUMBER=CUIT 20190178154, CN=glarriera20190903</destination>
<uniqueId>3706381629</uniqueId>
<generationTime>2019-09-27T10:11:51.532-03:00</generationTime>
<expirationTime>2019-09-27T22:11:51.532-03:00</expirationTime>
</header>
<credentials>
        <token>PD94bWwgdmVyc2lQiIH . . . aW9uPgo8L3Nzbz4K</token>
        <sign>kKu+Spy+QwiFB3fZjNw . . . n8b9hi5MDl/WZg/MW0=</sign>
</credentials>
</loginTicketResponse>
```

Un cliente de WSAA en PHP

Este ejemplo es un script PHP para solicitar un ticket de acceso al WSAA. Puede ejecutarse en una línea de comandos de Linux o de Windows (si tiene PHP instalado).

8.1 Descripción

```
#!/usr/bin/php
<?php
# Autor: Gerardo Fisanotti (AFIP, 2007)
# Descripción: Obtener un ticket de acceso (TA) del WSAA
# Entrada:
#        WSDL, CERT, PRIVATEKEY, PASSPHRASE, SERVICE, URL
#        Ver las definiciones
abajo # Salida:
#        TA.xml: El ticket de acceso otorgado por el WSAA
```

8.2 Valores de configuración

Definir los valores adecuadamente para el entorno usado.

```
define ("WSDL", "wsaa.wsdl"); # WSDL correspondiente al WSAA
define ("CERT", "MiCertificado2019.pem"); # Certificado usado para firmar
define ("PRIVATEKEY", "MiPrivada2019.key"); # Clave privada del certificado
define ("PASSPHRASE", ""); # Password para firmar
define ("PROXY_HOST", "10.30.28.25"); # IP del proxy para salir a Internet
define ("PROXY_PORT", "80"); # Puerto del proxy
define ("URL", "https://wsaahomo.afip.gov.ar/ws/services/LoginCms");
```

8.3 Función para crear el TRA

```
function CreateTRA($SERVICE)
{
        $TRA = new SimpleXMLElement(
                '<?xml version="1.0" encoding="UTF-8"?>' .
                '<loginTicketRequest version="1.0">'.
                '</loginTicketRequest>');
        $TRA->addChild('header');
        $TRA->header->addChild('uniqueId',date('U'));
        $TRA->header->addChild('generationTime',date('c',date('U')-60));
        $TRA->header->addChild('expirationTime',date('c',date('U')+60));
        $TRA->addChild('service',$SERVICE);
```

```
        $TRA->asXML('TRA.xml');
}
```

8.4 Función para firmar el mensaje

```
# Esta función hace la firma usando como entrada el archivo TRA,
# el certificado y la privada. Genera un archivo intermedio TRA.tmp
# y le quita los encabezados MIME para obtener el resultado.
function SignTRA()
{
  $STATUS=openssl_pkcs7_sign("TRA.xml", "TRA.tmp", "file://".CERT,
        array("file://".PRIVATEKEY, PASSPHRASE),
        array(),
        !PKCS7_DETACHED
        );
  if (!$STATUS) {exit("ERROR generating PKCS#7 signature\n");}
  $inf=fopen("TRA.tmp", "r");
  $i=0;
  $CMS="";
  while (!feof($inf))
        {
          $buffer=fgets($inf);
          if ( $i++ >= 4 ) {$CMS.=$buffer;}
        }
  fclose($inf);
  unlink("TRA.tmp")
  ; return $CMS;
}
```

8.5 Ejecución del método LoginCms del WSAA

```
function CallWSAA($CMS)
{
$client=new SoapClient(WSDL, array( 'proxy_host'=> PROXY_HOST, 'proxy_port' =>
PROXY_PORT, 'soap_version'
=> SOAP_1_2,
'location'=> URL,
'trace'=> 1,
'exceptions'=> 0
));
$results=$client->loginCms(array('in0'=>$CMS)); file_put_contents("request-
loginCms.xml",$client-> getLastRequest()); file_put_contents("response-
loginCms.xml",$client-> getLastResponse()); if (is_soap_fault($results))
{exit("SOAP Fault: ".$results->faultcode."\n".$results->faultstring."\n");} return
$results->loginCmsReturn;
```

8.6 Función para mostrar cómo se usa

```
function ShowUsage($MyPath)
{
  printf("Uso  : %s Arg#1 Arg#2\n", $MyPath);
  printf("donde: Arg#1 debe ser el service name del WS de negocio.\n");
```

```
  printf("  Ej.: %s wsfe\n", $MyPath);
}
```

8.7 Programa principal

```
ini_set("soap.wsdl_cache_enabled", "0");
if (!file_exists(CERT)) {exit("Failed to open ".CERT."\n");}
if (!file_exists(PRIVATEKEY)) {exit("Failed to open ".PRIVATEKEY."\n");}
if (!file_exists(WSDL)) {exit("Failed to open ".WSDL."\n");}
if ( $argc < 2 ) {ShowUsage($argv[0]); exit();}
$SERVICE=$argv[1];
CreateTRA($SERVICE)
;
$CMS=SignTRA();
$TA=CallWSAA($CMS);
if (!file_put_contents("TA.xml", $TA)) {exit();}
```

8.8 Ejemplo de ejecución en línea de comandos

NOTA

Luego de la ejecución, el ticket de acceso otorgado queda grabado en el archivo *TA.xml*.

```
    $ php wsaa-client.php ws_sr_constancia_inscripcion

    $ cat TA.xml
    <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <loginTicketResponse version="1.0">
<header>
    <source>CN=wsaahomo, O=AFIP, C=AR, SERIALNUMBER=CUIT 33693450239</source>
    <destination>SERIALNUMBER=CUIT 20190178154, CN=glarriera20190903</destination>
    <uniqueId>2005999911</uniqueId>
    <generationTime>2019-09-27T11:22:03.231-03:00</generationTime>
    <expirationTime>2019-09-27T23:22:03.231-03:00</expirationTime>
</header>
<credentials>
    <token>PD94bWwgdmVyc2lv . . . jwvc3NvPgo=</token>
    <sign>Yl3y6QmdM9L0TEv . . . gesW4dJSUsk=</sign>
</credentials>
    </loginTicketResponse>
```

Usando curl para probar los web services

En esta sección mostraremos cómo usar la heramienta *curl* para hacer pruebas contra los web services.

9.1 Obtener un ticket de acceso

```
$ curl -k -v -X POST --header "Content-Type: text/xml;charset=UTF-8"
--header "SOAPAction:urn:LoginCms"
--header "User-Agent: Apache-HttpClient/4.1.1 (java 1.5)"
--data-binary @requestLoginCms.xml
--noproxy "*" https://wsaahomo.afip.gov.ar/ws/services/LoginCms
```

donde *requestLoginCms.xml* es un archivo que contiene:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsaa="http://wsaa.view.sua.dvadac.desein.afip.gov">
 <soapenv:Header/>
    <soapenv:Body>
       <wsaa:loginCms>
          <wsaa:in0>
MIIG2AYJKoZIhvcNAQcCoIIGyTCC
pGLmfu+PvB5M7JqWE83FFQiNedXiEFuHt8wEtmE0nxRqqiU3ggDtSj7aF4Wnopu
0
MXlR5b1VTkRv3dNsovS1+rEcAo1mMh9SO2IoSl2eNeg/hHzsovRhSxhItTqjBp1
1
fKXbVv4M16YW9biJH15bh9cBZMWAsDVQlRlF9CA1s8rTrbxQCh6F60Era9Z0Evj
I
ImhCkSwznb0yRVXjf9oFLZVwSgG+P2Y3Jk7nc7ZlWKInvZleE/jZgteZDcixY9B
h OJoyp6HQkA8ps8iwYE1lQry1lVYfu/Xl6S10qQ==
                </wsaa:in0>
```

9.2 Consumir el método *dummy* de *ws_sr_constancia_inscripcion*

Vamos a hacer un request al método *dummy* del servicio *ws_sr_constancia_inscripcion*. Luego de ejecutar la consulta recibimos la respuesta de *ws_sr_constancia_inscripcion* con los datos buscados.

```
$ curl -k -v -X POST --header "Content-Type: text/xml;charset=UTF-8"
--header "SOAPAction:"
--header "User-Agent: Apache-HttpClient/4.1.1 (java 1.5)"
--data-binary @requestDummy.xml
--noproxy "*" https://awshomo.afip.gov.ar/sr-padron/webservices/personaServiceA5
```

donde *requestDummy.xml* es un archivo que contiene:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:a5="http://a5.soap.ws.server.puc.sr/">
 <soapenv:Header/>
    <soapenv:Body>
       <a5:dummy/>
    </soapenv:Body>
</soapenv:Envelope>
```

FAQ (Preguntas/Respuestas frecuentes)

10.1 Cuáles son los ambientes del WSAA

Los ambientes disponibles para el WSAA (Webservice de Autenticacion y Autorizacion) son:

- Ambiente de Testing/Homologacion:https://wsaahomo.afip.gov.ar/ws/services/LoginCms?WSDL
- Ambiente de Produccion:https://wsaa.afip.gov.ar/ws/services/LoginCms?WSDL

10.2 Certificado no emitido por AC de confianza

El error "Certificado no emitido por AC de confianza" suele ocurrir cuando:

- Usted está usando un certificado no emitido por la Autoridad Certificante (AC) de la AFIP.
- Usted esta usando un certificado AFIP de produccion para acceder a un webservice de homologacion (o viceversa, usted esta usando un certificado AFIP de homologacion para acceder a un webservice de produccion). Para nuestro ambiente de produccion debe usar un certificado de produccion. Y similarmente, para nuestro ambiente de homologacion debe usar un certificado de homologacion.

10.3 Computador no autorizado a acceder a los servicios de AFIP

El error "Computador no autorizado a acceder a los servicios de AFIP"se puede dar por una de las siguientes dos condiciones:

- En el entorno de producción envía un request al WSAA utilizando un certificado válido para el entorno de testing.
- En el entorno de testing envía un request al WSAA utilizando un certificado válido para el entorno de producción.

10.4 Computador no autorizado a acceder al servicio

El error "Computador no autorizado a acceder al servicio" puede ocurrir por diversos motivos:

- Que el certificado usado para firmar el mensaje usted aun no lo ha autorizado a acceder al servicio.

En ambiente de testing la autorización se hace siguiendo los pasos del capítulo 6 del Manual del Usuario del WSASS:http://www.afip.gob.ar/ws/WSASS/WSASS_manual.pdf

En ambiente de producción se denomina "delegación" y se hace acorde al manual: http://www.afip.gob.ar/ws/WSAA/ADMINREL.DelegarWS.pdf

- Que usted está firmando el mensaje con otro certificado y no el que corresponde.

- Que en el elemento ‘service’ del ticket request usted no ha puesto el ID del servicio correcto.

10.5 DN del source inválido

Para solucionar este inconveniente, cuando genere el TRA, no incluya los campos source y destination. los mismos son opcionales. Es recomendable que elija esta opción para evitar inconvenientes en el futuro si cambian los DN de los certificados.

10.6 El CEE ya posee un TA valido para el acceso al WSN solicita- do

El error “El CEE ya posee un TA valido para el acceso al WSN solicitado” ocurre cuando se solicitan al WSAA varios tickets de acceso (TA), para un mismo servicio, durante un corto lapso.

ACTUALMENTE ESE LAPSO PREVENTIVO ES DE 10 MINUTOS EN EL WSAA DE TESTING Y 2 MINUTOS EN EL WSAA DE PRODUCCION. TENER EN CUENTA QUE ESTOS VALORES PUEDEN SER MODIFICADOS DINAMICAMENTE Y SIN AVISO PREVIO.

Si se solicitan mas de un TA (ticket de acceso) dentro del lapso de retencion, estando vigente el anterior, el servicio WSAA de AFIP puede rechazar el pedido devolviendo el error “El CEE ya posee un TA valido para el acceso al WSN solicitado”.

Como idea general: Su sistema deberia solicitar al WSAA un TA y luego acceder al webservice de negocio multiples veces mientras el TA siga vigente (alrededor de 12 horas). Cuando deja de estar vigente, ahora si, vuelve a solicitar un TA nuevo. La vigencia del TA es conocida mediante el campo ‘expirationTime’, valor recibido al obtener un TA exitoso.

10.7 El tiempo de expiración es inferior a la hora actual

Este error se da porque el valor del campo expirationTime del TRA (LoginTicketRequest.xml) es inferior a la hora actual. Algunas de las posibles causas pueden ser las siguientes:

- Sincronización de Clocks no realizada. La fecha y hora del computador que genera el TRA y recibe el TA deberá estar sincronizada. Dicha sincronización

se podrá realizar a través del protocolo NTP con el servidor “time.afip.gov.ar” u otro servidor que preste dicho servicio.

- El equipo donde corre el cliente tienen incorrectamente ajustada la zona horaria, la cual debería ser GMT-3.

10.8 Firma inválida o algoritmo no soportado

El error “Firma inválida o algoritmo no soportado” (cms.sign.invalid, cms.bad y cms.bad.base64) normalmente ocurren porque no se ha proporcionado correctamente el mensaje criptográfico firmado. Generalmente se esta tratando de usar un certificado inválido o expirado, se está pasando mal algún parámetro (servicio o tiempo de vida), o se está tratando de acceder al ambiente equivocado (el certificado es de producción y el servidor de homologación, o viceversa.)

10.9 GenerationTime en el futuro o más de 24 horas de antiguedad

Esto indica que hay un problema en el valor provisto por su sistema en el campo ‘generationTime’ del TRA (LoginTicketRequest.xml). Las causas posibles de este error:

- El formato de fecha/hora que usaron no cumple con el formato requerido. Un ejemplo del formato valido es: *2018-03-21T12:57:42*.
- El formato aunque correcto, indicaria una fecha posterior a la fecha actual. En este caso ustedes deben verificar que la computadora donde corre su sistema este con la hora sincronizada contra un servidor NTP.

Como recomendacion practica, le sugerimos tambien que en la programacion donde construyen el valor del gene- rationTime, tomen el valor de fecha del sistema y le resten algunos minutos para ajustar potenciales problemas de sincronizacion.

Si el campo generationTime del TRA (LoginTicketRequest.xml) está correctamente generado la causa del error puede ser alguna de las siguientes:

- Sincronización de Clocks no realizada. La fecha y hora del computador que genera el TRA y recibe el TA deberá estar sincronizada. Dicha sincronización se podrá realizar a través del protocolo NTP con el servidor “time.afip.gov.ar” u otro servidor que preste dicho servicio.
- El equipo donde corre el cliente tiene incorrectamente ajustada la zona horaria, probar con GMT-3.

10.10 No se ha podido interpretar el XML contra el schema

El error “No se ha podido interpretar el XML contra el SCHEMA” suele producirse por alguno de estos motivos:

- Formato XML incorrecto del TRA. IMPORTANTE: las mayúsculas y minúsculas del XML deben respetarse.
- Formato incorrecto de la fecha en los campos generationTime y expirationTime del TRA. Un ejemplo de fecha correcto es: *2018-01-29T13:52:57.467-03:00*.
- Que la fecha no sea válida (es decir, que no sea un 31 de Febrero o que la hora sea 25:00).
- Que el ID de servicio exceda 35 caracteres.
- El mensaje firmado fue mal generado.

Links de interés

- Especificación Técnica del WebService de Autenticación y Autorización (WSAA):

http://www.afip.gob.ar/ws/WSAA/Especificacion_Tecnica_WSAA_1.2.2.pdf

- Manual del Usuario del WSASS:

http://www.afip.gob.ar/ws/WSASS/WSASS_manual.pdf

- Ejemplos open source de clientes del WSAA (PHP, Java, .NET, PowerShell):

http://www.afip.gob.ar/ws/documentacion/wsaa.asp