

**PMS Interfész C# és PHP kliens segédlet**

Verzió 1.00

**Nemzeti Turisztikai Adatszolgáltató**

**Központ kialakítása**

Projektazonosító: 2017/S 249-525381

Készítette: Sagemcom Magyarország Kft.

2019.07.07.

# 

# Dokumentum Kontroll

* 1. Dokumentum adatok

|  |  |
| --- | --- |
| **Dokumentum címe** | PMS Interfész C# kliens segédlet |
| **Projekt neve** | Nemzeti Turisztikai Adatszolgáltató Központ kialakítása |
| **Szerző(k)** | Szmetankó Gábor, Vasi Sándor |
| **Felelős** | Fekete Tamás |
| **Cég** | Sagemcom Magyarország Kft. |
| **Elektronikus állomány neve** | NTAK\_PMS\_Interfesz\_CSharp\_es\_PHP\_Kliensek\_v1.00.docx |
| **Dokumentum verzió** | V1.00 |
| **Oldalszám** |  |
| **Státusz** |  |

* 1. Változáskövetés

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Verzió | Állapot | Dátum | Felelős Személy | Változás oka |
| *v1.00* |  | 2019.07.07 | Sagemcom | Első verzió |

Tartalom

[1. Dokumentum Kontroll 2](#_Toc14188361)

[1.1. Dokumentum adatok 2](#_Toc14188362)

[1.2. Változáskövetés 3](#_Toc14188363)

[2. Bevezető 5](#_Toc14188364)

[3. Példa Microsoft.NET alapú megvalósításra 6](#_Toc14188365)

[3.1. Aláírói illetve authentikációs kulcspár létrehozása 6](#_Toc14188366)

[3.2. Kulcspár alapján tanúsítvány kérelem létrehozása 7](#_Toc14188367)

[3.3. Kulcspárok és tanúsítványok elhelyezése jelszóval védett tárolóban 8](#_Toc14188368)

[3.4. SOAP végpontok kliens osztályainak létrehozása 9](#_Toc14188369)

[3.5. SoapFilter létrehozása 9](#_Toc14188370)

[3.6. Authentikációs tanúsítvány alkalmazása 10](#_Toc14188371)

[3.7. Aláírói tanúsítvány alkalmazása 11](#_Toc14188372)

[4. Példa PHP alapú megvalósításra 13](#_Toc14188373)

[4.1. Meglévő SOAP üzenet digitális aláírással történő ellátása 13](#_Toc14188374)

[4.2. Aláírt SOAP üzenet küldése 18](#_Toc14188375)

[5. Gyakori problémák, kérdések 20](#_Toc14188376)

[5.1. Az aláíró tanúsítvány nem tartalmazza a Digital Signature Key Usage extensiont, ezért az NTAK által kiállított tanúsítványok nem alkalmasak aláírások készítésére 20](#_Toc14188377)

[5.2. The underlying connection was closed: Could not establish trust relationship for the SSL/TLS secure channel. 21](#_Toc14188378)

[6. Mellékletek 23](#_Toc14188379)

[6.1. C# nyelven készült mintaalkalmazás 23](#_Toc14188380)

[6.2. PHP nyelven készült mintaalkalmazás 23](#_Toc14188381)

# Bevezető

A Nemzeti Turisztikai Adatszolgáltató Központ kialakítása részeként kiadott PMS Interfész leírás dokumentum az NTAK rendszer a PMS szoftverekkel való integrációjának előfeltételeit, az integráció PMS oldali konfigurációját, annak kommunikációs csatornáit és az adatküldés szabályait részletezte.

Jelen dokumentum .NET, illetve PHP alapú PMS rendszerek kialakításához nyújt részletesebb segítséget és választ ad az eddig felmerült gyakori kérdésekre. A dokumentum mellékleteként egy Visual Studio Solution is elérhető, amely végig viszi a fejlesztőt a sikeres csatlakozás és adatküldés lépésein.

Fontos megjegyezni, hogy a csatolt forráskód továbbra sem alkalmas élő rendszerben történő használatra. A hibakezelés megfelelő implementálása, illetve a kód megfelelő strukturálása a PMS fejlesztő feladata.

# Példa Microsoft.NET alapú megvalósításra

Az NTAK rendszerhez történő sikeres csatlakozás technikai előfeltétele, hogy egyrészt megfelelő authentikációs és aláírói tanúsítványok legyenek alkalmazva az adatküldés során, másrészt az üzenetváltás megfelelő SOAP adatstruktúrákat tartalmazzon. Ezen előfeltétel eléréséhez Microsoft.NET alapú megvalósítás esetén az alábbi lépések szükségesek

1. Aláírói illetve authentikációs kulcspár létrehozása (PEM)
2. Kulcspár alapján tanúsítvány kérelem létrehozása (CSR)
3. Tanúsítvány kérelem feltöltése az NTAK rendszerbe
4. NTAK rendszer által kiadott tanúsítványok letöltése (CER)
5. Kulcspárok és tanúsítványok elhelyezése jelszóval védett tárolóban (P12)
6. NTAK végpontok kliens osztályainak létrehozása WSDL állományok alapján
7. NTAK végpontokhoz történő HTTPS csatlakozás kialakítása az authentikációs tanúsítvány segítségével
8. NTAK végpontokon felküldött SOAP borítékok elektronikus aláírása az aláírói tanúsítvány alkalmazásával

Bár a fenti lépések biztosítása többféle komponens alkalmazása mentén lehetséges, jelen fejezetben az alábbi komponenseket használtuk fel:

1. BouncyCastle.NetCore komponens.   
   Tanúsítvány kérelem előállítására és a tanúsítvány és kulcspár jelszóval védett tárolóban történő elhelyezése
2. BCrypt.Net-Next komponens.  
   Vendég azonosító adatok egyirányú és visszafejthetetlen transzformációja
3. Microsoft.Web.Services3 komponens.  
   Felküldött SOAP borítékok elektronikus aláírása

A szükséges lépések végrehajtásához nyújtanak segítséget az alábbi alfejezetekben leírt kódrészletek, illetve a fentebb csatolt példa Visual Studio Solution.

* 1. Aláírói illetve authentikációs kulcspár létrehozása

A privát és publikus kulcsokat tartalmazó kulcspár létrehozásához a kulcs 4096-bites hosszát és a felhasznált RSA algoritmust kell megadnunk

// 4096-bit hosszú RSA kulcspár létrehozása  
var random = new SecureRandom();  
var parameters = new KeyGenerationParameters(random, 4096);  
var generator = new RsaKeyPairGenerator();  
generator.Init(parameters);  
var keypair = generator.GenerateKeyPair();

Az így létrejött kulcspárt ajánlott elmenteni későbbi felhasználásra

FileStream file = new FileStream("reg00008.pem", FileMode.Create, FileAccess.Write, FileShare.Read);  
StreamWriter streamWriter = new StreamWriter(file);  
PemWriter pemWriter = new PemWriter(streamWriter);  
pemWriter.WriteObject(keypair);  
streamWriter.Close();

Fontos, hogy a fenti generálást kétszer kell elvégezni, hogy létrejöjjön egy kulcspár az aláírásokhoz, illetve egy az authentikációhoz.

* 1. Kulcspár alapján tanúsítvány kérelem létrehozása

Tanúsítvány kérelem létrehozásához a tanúsítvány Alany (Subject) mezőjét kell helyesen kitölteni

string szallasRegisztraciosSzam = "reg00008";  
string szallasNev = "Vadvirág Panzió";  
string szallasIranyitoszam = "3325";  
string szallasTelepules = "Noszvaj";  
string szallashelySzolgaltatoAdoszam = "12345678";  
  
// Tanúsítvány kérelem létrehozása  
//  
// A subjectName mező értékét a szálláshely adatiaval kell feltölteni  
// Common Name (CN) – Az Alany neve: Szálláhely regisztrációs száma  
// Organization (O) – A Szervezet megnevezése: Szálláshely neve  
// Organization Identifier (OrgId) – OID: 2.5.4.97 – Szervezet azonosítója: Szálláshely szolgáltató adószáma  
// Country (C) – Ország azonosítója: Mindig Magyarország, azaz "HU" érték kerül kitöltésre  
// Locality Name(L) – Településnév: Szálláshely település  
// Postal Code – OID: 2.5.4.17 – Irányítószám: Szálláshely irányítószám  
  
var subjectName = $"CN={szallasRegisztraciosSzam}, O={szallasNev}, 2.5.4.97={szallashelySzolgaltatoAdoszam}, 2.5.4.17={szallasIranyitoszam}, L={szallasTelepules}, C=HU";  
var subject = new X509Name(subjectName);  
var factory = new Asn1SignatureFactory(PkcsObjectIdentifiers.Sha512WithRsaEncryption.Id, keypair.Private, random);

var request = new Pkcs10CertificationRequest(factory, subject, keypair.Public, null, keypair.Private);

Az így létrejött tanúsítvány kérelmeket kell állományba menteni és felküldeni az NTAK rendszer felé.

FileStream file = new FileStream("reg00008.csr", FileMode.Create, FileAccess.Write, FileShare.Read);  
StreamWriter streamWriter = new StreamWriter(file);  
PemWriter pemWriter = new PemWriter(streamWriter);  
pemWriter.WriteObject(request);  
streamWriter.Close();

* 1. Kulcspárok és tanúsítványok elhelyezése jelszóval védett tárolóban

Az NTAK rendszer által kibocsátott tanúsítványt és a korábban létrehozott kulcspár ajánlott jelszóval védett tárolóban elhelyezni és mind a tároló állomány mind a hozzá kapcsolódó jelszót megfelelően védeni

// Tanúsítvány felovasása  
X509CertificateParser parser = new X509CertificateParser();  
FileStream file = new FileStream("reg00008.cer", FileMode.Open, FileAccess.Read, FileShare.Read);  
X509Certificate certificate = parser.ReadCertificate(file);  
file.Close();  
  
// Kulcspár felovasása  
file = new FileStream("reg00008.pem", FileMode.Open, FileAccess.Read, FileShare.Read);  
StreamReader streamReader = new StreamReader(file);  
PemReader pemReader = new PemReader(streamReader);  
AsymmetricCipherKeyPair keypair = (AsymmetricCipherKeyPair) pemReader.ReadObject();  
streamReader.Close();  
  
// Kulcstároló létrehozása  
var random = new SecureRandom();  
string keyPassword = "1111";  
Pkcs12Store store = new Pkcs12StoreBuilder().Build();  
  
// Kulcstároló feltöltése  
X509CertificateEntry certificateEntry = new X509CertificateEntry(certificate);  
store.SetCertificateEntry("reg00008", certificateEntry);  
AsymmetricKeyEntry keyEntry = new AsymmetricKeyEntry(keypair.Private);  
store.SetKeyEntry("reg00008", keyEntry, new X509CertificateEntry[] { certificateEntry });  
  
// Kulcstároló mentése  
file = new FileStream("reg00008.p12", FileMode.Create, FileAccess.Write, FileShare.Read);  
store.Save(file, keyPassword.ToCharArray(), random);  
file.Close();

* 1. SOAP végpontok kliens osztályainak létrehozása

A SOAP végpontok WSDL definíciós állományai alapján létrehozhatjuk azokat az osztályokat és adathalmazokat, amelyek segítségével összeállíthatjuk a felküldendő SOAP borítékot.

wsdl.exe" /out:NTAK.cs /nologo /order /protocol:SOAP   
 /namespace:NTAK /sharetypes /enableDataBinding  
 esemenyvezerelt-adatkuldes.wsdl  
 napi-zaras-teszt.wsdl  
 napi-zaras.wsdl  
 napi-zaras-utemezes.wsdl

Előfordulhat, hogy az újabb .NET keretrendszer verziók nem a Microsoft.Web.Services3 komponensnek megfelelő WebServicesClientProtocol osztály alapján hozzák létre a kliens osztályokat, hanem SoapHttpClientProtocol alaposztály felhasználásával. Ebben az esetben cseréljük le a létrehozott NTAK.cs állomában az összes

System.Web.Services.Protocols.SoapHttpClientProtocol

alaposztályt a megfelelő

Microsoft.Web.Services3.WebServicesClientProtocol

alaposztályra

* 1. SoapFilter létrehozása

A WSE3 osztálykönyvtár hibája miatt nem enged amúgy helyesen kiállított tanúsítványokat alkalmazni digitális előírások előállítására. Ugyanez a probléma más programnyelvek és osztálykönyvtárak alkalmazása esetén nem fordul elő. Ahhoz, hogy a megfelelő digitális aláírással tudjuk ellátni a SOAP borítékot, ki kell egészítenünk a WSE3 megoldást egy egyedi SoapFilterrel, amely a kimenő üzeneteket ellátja a megfelelő digitális aláírással. Ehhez először létre kell hozni egy SecurityPolicyAssertiont:

class SigningPolicyAssertion : SecurityPolicyAssertion  
{  
 private Security security;  
  
 public SigningPolicyAssertion(Security security)  
 {  
 this.security = security;  
 }  
  
 public override SoapFilter CreateClientInputFilter(FilterCreationContext context)  
 {  
 return null;  
 }  
  
 public override SoapFilter CreateClientOutputFilter(FilterCreationContext context)  
 {  
 return new ClientOutputFilter(this, security);  
 }  
  
 public override SoapFilter CreateServiceInputFilter(FilterCreationContext context)  
 {  
 return null;  
 }  
  
 public override SoapFilter CreateServiceOutputFilter(FilterCreationContext context)  
 {  
 return null;  
 }  
}

Majd aktiválni azt az NTAK.cs fájlban található service konstruktorában:

public napiZarasTesztPortService() {  
 this.Url = "http://localhost:8081/ntak/";  
 // Aláíró policy hozzáadása a klienshez.  
 Policy policy = new Policy();  
 policy.Assertions.Add(new SigningPolicyAssertion(RequestSoapContext.Security));  
 this.SetPolicy(policy);  
}

* 1. Authentikációs tanúsítvány alkalmazása

SOAP végpontokkal történő HTTPS kommunikációs csatorna kialakításakor használjunk kliens oldali tanúsítványokat

// Authentikációs kulcs felolvasása  
String keyPath = "reg00008-authentication.p12";  
String keyPassword = "1111";  
X509Certificate2 certificate = new X509Certificate2(keyPath, keyPassword);  
  
// Kliens osztály felépítése  
var client = new NTAK.esemenyvezereltAdatkuldesPortService();  
client.Url = "https:// ... ";  
client.ClientCertificates.Add(certificate);

* 1. Aláírói tanúsítvány alkalmazása

SOAP üzenetek elektronikus aláírásához az aláírói tanúsítványokat kell alkalmaznunk a Timestamp és Body elemekre. Ezt a fentebbi SecurityPolicyAssertionben hivatkozott ClientOutputFilter végzi el.

internal class ClientOutputFilter : SoapFilter  
{  
 private readonly Security security;  
  
 public ClientOutputFilter(SigningPolicyAssertion parentAssertion, Security security)  
 {  
 this.security = security;  
 }  
 public override SoapFilterResult ProcessMessage(SoapEnvelope envelope)  
 {  
 // ID attribútum hozzáadása a Body elemhez. Ez a későbbi referencia hozzáadása miatt szükséges.  
 XmlAttribute idAttrib = envelope.CreateAttribute("wsu", "Id", "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd");  
 idAttrib.Value = "Body";  
 envelope.Body.Attributes.Append(idAttrib);  
  
 // Aláírói kulcs felolvasása  
 var keyPath = "alairo.p12";  
 var keyPassword = "1111";  
 X509Certificate2 certificate = new X509Certificate2(keyPath, keyPassword);  
  
 // WSSE Security elem létrehozása  
 var token = new X509SecurityToken(certificate);  
 security.Tokens.Add(token);  
 security.Timestamp.TtlInSeconds = 43200;  
 security.SerializeXml(envelope);  
  
 // Aláírás elkészítése  
 var signedXml = new SignedXmlWithId(envelope);  
 signedXml.SigningKey = certificate.GetRSAPrivateKey();  
 signedXml.SignedInfo.SignatureMethod = SignedXml.XmlDsigRSASHA384Url;  
 signedXml.SignedInfo.CanonicalizationMethod = SignedXml.XmlDsigExcC14NTransformUrl;  
  
 AddReference(signedXml, security.Timestamp.Id);  
 AddReference(signedXml, "Body");  
  
 KeyInfo keyInfo = new KeyInfo();  
 SecurityTokenReference tokenRef = new SecurityTokenReference(token);  
 keyInfo.AddClause(tokenRef);  
 signedXml.KeyInfo = keyInfo;  
  
 signedXml.ComputeSignature();  
 XmlElement signedElement = signedXml.GetXml();  
  
 // Aláírás beillesztése a SOAP Header WSSE Security elemébe  
 XPathNavigator signatureNavigator = signedElement.CreateNavigator();  
 XPathNavigator headerNavigator = envelope.Header.CreateNavigator();  
 XmlNamespaceManager namespaceManager = new XmlNamespaceManager(envelope.NameTable);  
 namespaceManager.AddNamespace("wsse", "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd");  
 XPathNavigator secNode = headerNavigator.SelectSingleNode("wsse:Security", namespaceManager);  
 secNode.AppendChild(signatureNavigator);  
  
 return SoapFilterResult.Continue;  
 }  
  
 private void AddReference(SignedXml signedXml, string uri)  
 {  
 Reference reference = new Reference();  
 reference.Uri = "#" + uri;  
 reference.DigestMethod = SignedXml.XmlDsigSHA384Url;  
 XmlDsigExcC14NTransform env = new XmlDsigExcC14NTransform();  
 reference.AddTransform(env);  
  
 signedXml.AddReference(reference);  
 }  
}

A fenti kódrészletben hivatkozott SignedXmlWithId forrása a következő:

// SignedXml osztály kiterjesztése, hogy a wsu:Id attribútumokat is tudja azonosító elemként kezelni.

internal class SignedXmlWithId : SignedXml  
{  
 public SignedXmlWithId(SoapEnvelope envelope) : base(envelope)  
 {  
 }  
  
 public override XmlElement GetIdElement(XmlDocument document, string idValue)  
 {  
 XmlElement idElem = base.GetIdElement(document, idValue);  
  
 if (idElem == null)  
 {  
 XmlNamespaceManager nsManager = new XmlNamespaceManager(document.NameTable);  
 nsManager.AddNamespace("wsu", "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd");  
  
 idElem = document.SelectSingleNode("//\*[@wsu:Id=\"" + idValue + "\"]", nsManager) as XmlElement;  
 }  
  
 return idElem;  
 }  
}

# Példa PHP alapú megvalósításra

Az NTAK rendszerhez történő sikeres csatlakozás technikai előfeltétele, hogy egyrészt megfelelő autentikációs és aláírói tanúsítványok legyenek alkalmazva az adatküldés során, másrészt az üzenetváltás megfelelő SOAP adatstruktúrákat tartalmazzon. Ezen előfeltétel eléréséhez PHP alapú megvalósítás esetén az alábbi lépések szükségesek

1. Aláírói illetve autentikációs kulcspár létrehozása (PEM)
2. Kulcspár alapján tanúsítvány kérelem létrehozása (CSR)
3. Tanúsítvány kérelem feltöltése az NTAK rendszerbe
4. NTAK rendszer által kiadott tanúsítványok letöltése (CER)
5. Kulcspárok és tanúsítványok elhelyezése PEM formátumú állományokban)
6. NTAK végpontok kliens osztályainak létrehozása WSDL állományok alapján
7. NTAK végpontokhoz történő HTTPS csatlakozás kialakítása az autentikációs tanúsítvány segítségével
8. NTAK végpontokon felküldött SOAP borítékok elektronikus aláírása az aláírói tanúsítvány alkalmazásával

Bár a fenti lépések biztosítása többféle komponens alkalmazása mentén lehetséges, jelen fejezetben az alábbi komponenseket használtuk fel:

1. Xmlseclibs (Rob Richards) - <https://github.com/robrichards/xmlseclibs>

SOAP borítékok digiális aláírására szolgáló eszköz

1. PHP cURL – <https://www.php.net/manual/en/book.curl.php>  
   HTTP hívások megvalósítására szolgáló függvénykönyvtár
   1. Meglévő SOAP üzenet digitális aláírással történő ellátása

|  |
| --- |
| <?php  const SECONDS\_TO\_EXPIRE = 43200;  const WSSECURITY\_UTILITY\_1\_0 = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd";  const WSSECURITY\_SECEXT\_1\_0 = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd";  const WSS\_X\_509\_TOKEN\_PROFILE\_1\_0 = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3";  const WSS\_SOAP\_MSG\_SEC\_1\_0\_BASE\_64\_BIN = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary";  require \_\_DIR\_\_ . '/vendor/autoload.php';  use RobRichards\XMLSecLibs\XMLSecurityDSig;  use RobRichards\XMLSecLibs\XMLSecurityKey;  // Aláírás előtt álló SOAP üzenet betöltése fájlból  $soapMessage = new DOMDocument();  $soapMessage->load('./napi-zaras-teszt-adat-request.xml');  $signingCertificateFile = './keys/sign.pem';  $privateKeyOfSigningCertificate = './keys/private-sign-nopw.pem';  // XML dokumentum aláírása  $resultDoc = signXML($soapMessage, $signingCertificateFile, $privateKeyOfSigningCertificate);  Save the signed XML  $resultDoc->save('signed.xml');  function signXML($doc, $signingCertificate, $privateKeyFile) {  print 'signing ...';  // SOAP header elem referenciája  $headerElement = $doc->getElementsByTagNameNS('http://schemas.xmlsoap.org/soap/envelope/', 'Header')->item(0);  // SOAP body elem referenciája  $bodyElement = $doc->getElementsByTagNameNS('http://schemas.xmlsoap.org/soap/envelope/', 'Body')->item(0);  // A BinarySecurityToken elemen egy ID attribútum beállítása  $binarySecurityTokenId = generateUniqueId('X509', 34);  // Security elem létrehozása a WSS-hez  $securityElement = createSecurityElement($doc, $signingCertificate, $binarySecurityTokenId);  // Az előzőleg létrehozott Security elem hozzáadása a meglévő SOAP Header elemhez.  $headerElement->appendChild($securityElement);  // Signature elem létrehozása az XMLSecLib lib segítségével  $signatureElement = new XMLSecurityDSig();  // Az előzőleg létrehozott Signature elemhez egy ID attribútum létrehozása  $signatureElement->sigNode->setAttribute("Id", generateUniqueId("SIG", 34));  // Canonical method beállítása a Signature elemen  $signatureElement->setCanonicalMethod(XMLSecurityDSig::EXC\_C14N);  $signatureElement->canonicalizeSignedInfo();  // A body elem számára referencia létrehozása, melyet majd alá kell írni  $signatureElement->addReference(  $bodyElement,  XMLSecurityDSig::SHA384,  null,  array('id\_name' => 'Id', 'prefix\_ns' => WSSECURITY\_UTILITY\_1\_0)  );  // Timestamp szekció létrehozása és hozzáfűzése a Security elemhez  $timestampElement = createTimestamp($doc);  $securityElement->appendChild($timestampElement);  // A timestamp elem számára referencia létrehozása, melyet majd alá kell írni  $signatureElement->addReference(  $timestampElement,  XMLSecurityDSig::SHA384,  null,  array('id\_name' => 'Id', 'prefix\_ns' => WSSECURITY\_UTILITY\_1\_0)  );  // Privát kulcs betöltése az aláíráshoz  $privateKey = new XMLSecurityKey(XMLSecurityKey::RSA\_SHA256, array('type' => 'private'));  $privateKey->loadKey($privateKeyFile, TRUE, FALSE);  // KeyInfo elem létrehozása  $signatureElement->appendToKeyInfo(createTokenReference($signatureElement->sigNode, $binarySecurityTokenId));  // SOAP üzenet megfelelő részeinek aláírása  $signatureElement->sign($privateKey, $securityElement);  // Signature elem hozzáfűzése a korábban létrehozott Security elemhez  $signatureElement->appendSignature($securityElement);  print " done\n";  return $doc;  }  // A könnyebb olvashatóság érdekében az aláírást végző funkció részfeladatait kiemeltük külön függvényekbe:  function createSecurityElement($document, $cert\_file, $binarySecurityTokenId) {  $security = $document->createElementNS(WSSECURITY\_SECEXT\_1\_0, "wsse:Security");  $security->setAttribute("xmlns:wsu", WSSECURITY\_UTILITY\_1\_0);  $binarySecurityToken = $document->createElement("wsse:BinarySecurityToken", getCertificateAsString($cert\_file));  $binarySecurityToken->setAttribute("EncodingType", WSS\_SOAP\_MSG\_SEC\_1\_0\_BASE\_64\_BIN);  $binarySecurityToken->setAttribute("ValueType", WSS\_X\_509\_TOKEN\_PROFILE\_1\_0);  $binarySecurityToken->setAttribute("wsu:Id", $binarySecurityTokenId);  $security->appendChild($binarySecurityToken);  return $security;  }  function getCertificateAsString($cert\_file) {  $cert\_file\_content = file\_get\_contents($cert\_file);  preg\_match\_all('/-----BEGIN CERTIFICATE-----([a-zA-Z0-9\n\+\/\=]+)-----END CERTIFICATE-----/', $cert\_file\_content, $cert\_array);  $cert = $cert\_array[1];  return trim(preg\_replace('/\s+/', '', implode('', $cert)));  }  function generateUniqueId($prefix, $valLength) {  $result = '';  $moduleLength = 40;  $steps = round(($valLength / $moduleLength) + 0.5);  for ($i = 0; $i < $steps; $i++) {  $result .= sha1(uniqid() . md5(rand() . uniqid()));  }  return $prefix . '-' . substr($result, 0, $valLength);  }  function createTimestamp($document) {  $timestamp = $document->createElementNS(WSSECURITY\_UTILITY\_1\_0, "wsu:Timestamp");  $currentTime = time();  $secondsToExpire = SECONDS\_TO\_EXPIRE;  $created = gmdate("Y-m-d\TH:i:s", $currentTime) . 'Z';  $expires = gmdate("Y-m-d\TH:i:s", $currentTime + $secondsToExpire) . 'Z';  $createdElement = $document->createElementNS(WSSECURITY\_UTILITY\_1\_0, "wsu:Created", $created);  $expiresElement = $document->createElementNS(WSSECURITY\_UTILITY\_1\_0, "wsu:Expires", $expires);  $timestamp->appendChild($createdElement);  $timestamp->appendChild($expiresElement);  return $timestamp;  }  function createTokenReference($document, $binarySecurityTokenId) {  $baseDoc = $document->ownerDocument;  $securityTokenReference = $baseDoc->createElementNS(WSSECURITY\_SECEXT\_1\_0, 'SecurityTokenReference');  $reference = $baseDoc->createElementNS(WSSECURITY\_SECEXT\_1\_0, 'wsse:Reference');  $reference->setAttribute('URI', "#$binarySecurityTokenId");  $reference->setAttribute('ValueType', WSS\_X\_509\_TOKEN\_PROFILE\_1\_0);  $securityTokenReference->appendChild($reference);  return $securityTokenReference;  }  ?> |

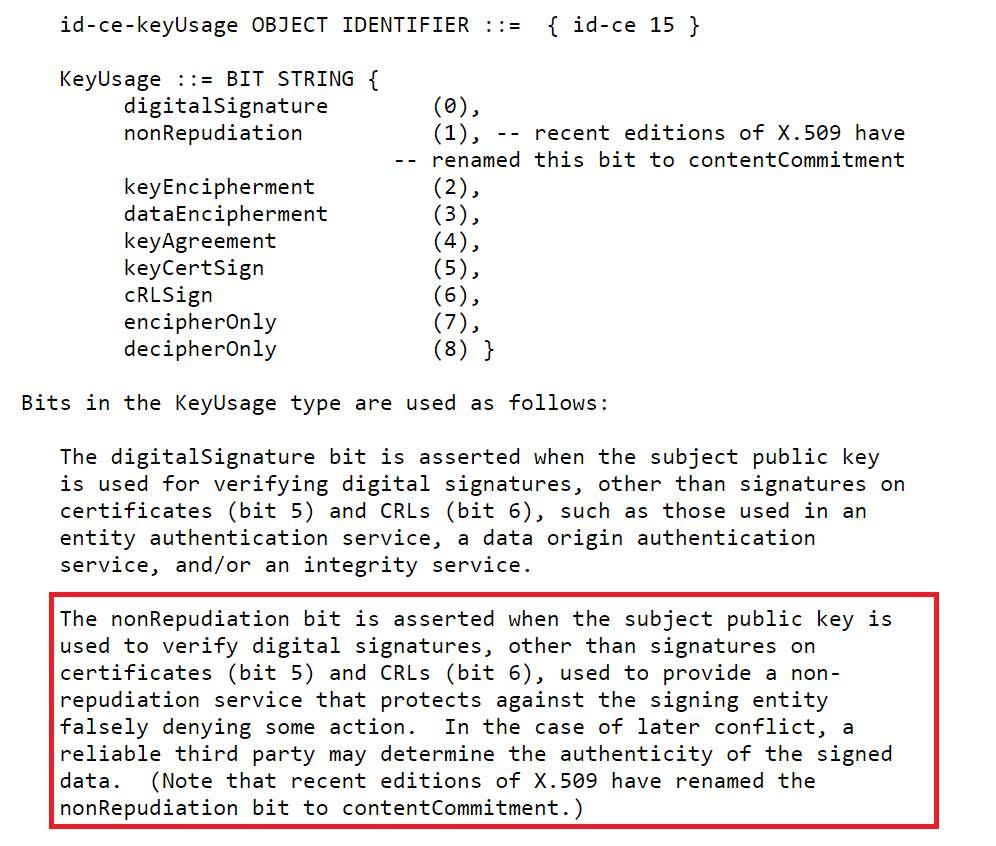
* 1. Aláírt SOAP üzenet küldése

|  |
| --- |
| <?php  $endPointUrl = 'https://ntak.ad.development/ntak/v6';  $soapMessage = file\_get\_contents(‘signed-message.xml’);  $authCertificate = './keys/auth.pem';  $privateKeyOfAuthCertificate = './keys/private-auth-nopw.pem';  $caCertificate = './keys/ntaktstsagemhu.crt';  $response = sendXML($endPointUrl, $soapMessage, $authCertificate ,$privateKeyOfAuthCertificate, $caCertificate);  print "response:\n$response\n";  function sendXML($endPointUrl, $payload, $authCertificate, $authKey, $caCertificate, $debug = false) {  print 'sending ...';  $curlHandler = curl\_init($endPointUrl);  curl\_setopt($curlHandler, CURLOPT\_VERBOSE, $debug);  // Autentikációs kulcshoz tartozó privát kulcs felolvasása:  curl\_setopt($curlHandler, CURLOPT\_SSLKEY, $authKey);  // curl\_setopt($curlHandler, CURLOPT\_SSLKEYPASSWD, ‘privat kulcs jelszava’);  // szerver tanusítványának felolvasása  curl\_setopt($curlHandler, CURLOPT\_CAINFO, $caCertificate);  curl\_setopt($curlHandler,CURLOPT\_SSL\_VERIFYPEER,0);  curl\_setopt($curlHandler, CURLOPT\_CAPATH, $caCertificate);  // Authentikációs tanusítvány felolvasása:  curl\_setopt($curlHandler, CURLOPT\_SSLCERT, $authCertificate);  // Activate HTTP Post  curl\_setopt($curlHandler, CURLOPT\_POST, true);  // Set the conent that needs to be sent  curl\_setopt($curlHandler, CURLOPT\_POSTFIELDS, $payload);  // the curl\_exec method call should return with the response instead of printing it to the stdout  curl\_setopt($curlHandler, CURLOPT\_RETURNTRANSFER, true);  // setting the header  curl\_setopt($curlHandler, CURLOPT\_HTTPHEADER, array('Content-Type: text/xml;charset=UTF-8'));  // send the request  $response = curl\_exec($curlHandler);  // print the error, if there is any  if (curl\_errno($curlHandler) <> 0) {  print "\nerr: " . curl\_error($curlHandler) . "\n";  }  curl\_close($curlHandler);  print " done\n";  return $response;  } |

# Gyakori problémák, kérdések

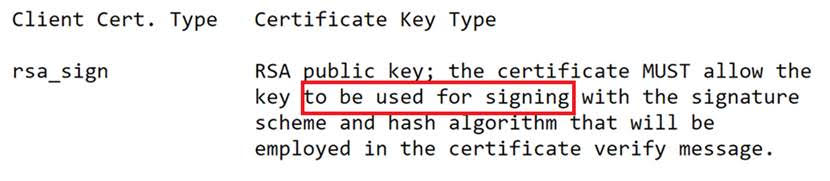
* 1. Az aláíró tanúsítvány nem tartalmazza a Digital Signature Key Usage extensiont, ezért az NTAK által kiállított tanúsítványok nem alkalmasak aláírások készítésére

A tanúsítvány KeyUsage attribútumának értelmezéséhez először érdemes magát az X.509 szabványt (https://tools.ietf.org/html/rfc5280) megismerni, legalábbis a vonatkozó részt:

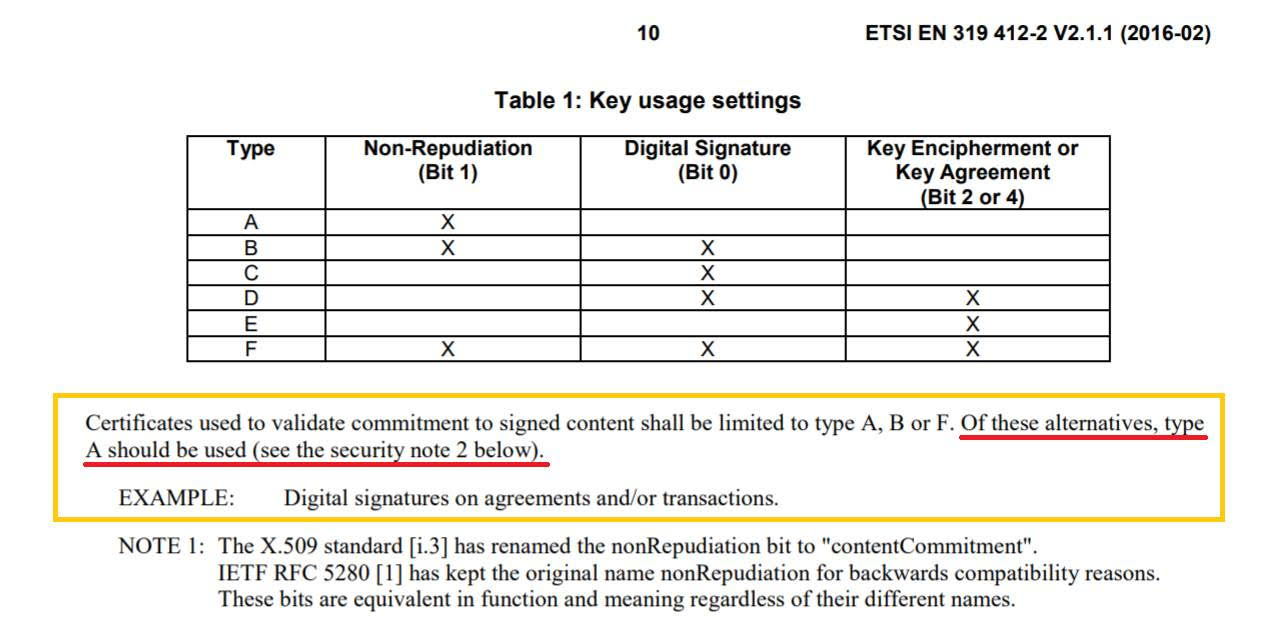


A nonRepudiation használatos aláírt adatok készítésére. Minősített tanúsítványok esetében ez a kötelező érték aláíró tanúsítványokban.

A digitalSignature KeyUsage definícióját tanulmányozva kiderül, hogy az viszont authentikációs szolgáltatásokban történő felhasználásra szolgál. Ezért is szerepel ez az érték az NTAK authentikációs tanúsítványokban és nem az aláíró tanúsítványokban. Segítségként még érdemes a TLS1.2 szabvány (https://tools.ietf.org/html/rfc5246#page-56) 7.4.6 Client Certificate fejezetét is megismerni, amiből egy részlet:



Továbbá a vonatkozó rész az ETSI szabványból (<https://www.etsi.org/deliver/etsi_en/319400_319499/31941202/02.01.01_60/en_31941202v020101p.pdf>):



A fentiek alapján látható, hogy az NTAK rendszerhez kiállított tanúsítványok mindenben megfelelnek a szabványokban előírt szabályoknak.

A problémát a WSE3 osztálykönyvtár hibája okozza, amely elvárja a digitalSignature KeyUsage értéket az aláíró tanúsítványban, amikor arra nincs is szükség.

* 1. The underlying connection was closed: Could not establish trust relationship for the SSL/TLS secure channel.

A teszt környezetben tapasztalt “The underlying connection was closed: Could not establish trust relationship for the SSL/TLS secure channel.” hibának az oka a teszt környezetben használt szerver tanúsítvány előállításának módjából ered. A teszt környezetben self-signed tanúsítványt használ a rendszer, így a PMS szoftverek alapból nem tekintik megbízhatónak ezt a tanúsítványt. A hibát úgy lehet kiküszöbölni, hogy a Trust Store-ban elhelyezzük a központi NTAK szerver tanúsítványát (a teljes certificate chain-t), ezáltal a szerver megbízhatóvá válik.

C# programok esetén ez legegyszerűbben a Windows tanúsítvány-kezelő alkalmazásával érhető el, amelyet a Start/Futtatás/certlm.msc beírásával tudunk elindítani. Itt a Trusted Root Certification Authorities mappában szerepelnie kell az NTAK-os tanúsítványnak. A tanúsítványt (chaint) a legegyszerűbben úgy szerezhetik meg, ha egy böngészőben betöltik az NTAK endpointot (<https://ntak.ad.development/ntak>), és onnan kimentik a tanúsítványt.

# Mellékletek

* 1. C# nyelven készült mintaalkalmazás



* 1. PHP nyelven készült mintaalkalmazás

