< IH Berichtauthenticatie met UZI-Pas

Datum: 1 februari 2019
Publicatie: V8.0.3.0

Inhoudsopgave

1 Inleiding

1.1 Doel en scope

Dit document beschrijft op welke wijze berichten tussen een goed beheerd zorgsysteem (GBZ) en het landelijk schakelpunt (LSP) worden uitgerust met authenticatietokens.

Dit document specificeert het authenticatietoken voor zorgverleners en hun medewerkers dat wordt gemaakt met hun UZI-pas. Daarnaast wordt het plaatsen van de digitale handtekening besproken (zie ook [IH tokens generiek]).

1.2 Doelgroep voor dit document

Dit document is bedoeld voor softwareontwikkelaars van XIS'en.

1.3 Documenthistorie

Versie	Datum	Omschrijving
v6.10.0.0	12-okt-2011	RfC 46142: SOAP Headers van tokens worden uitgebreid met soap:actor.
v6.11.0.0	12-okt-2012	Ongewijzigde herpublicatie als onderdeel van AORTA-Infrastructuur v6·11
v6.12.0.0	4-okt-2013	Bijlage technische context (gebruik UZI-pas) weer toegevoegd (zat in v6.0.5.0, maar verdwenen in v6.10)
V8.0.1.0	15-mei-2017	RfC 52477: Uitwisseling op basis van bouwstenen.
V8.0.1.0	15-mei-2017	RfC 65592: SHA1 encryptiemethoden verwijderd
V8.0.1.0	15-mei-2017	RfC 65898 en RfC 36422 SHA1 delen van AORTA_Auth_IH_Berichtauthenticatie_UZI.doc bijwerken met SHA256. Voor nieuwe leveranciers is het gebruik van SHA256 als DigestMethod en als onderdeel van SignatureMethode verplicht. Verwijzingen naar G2 generatie UZI-certificaat hierarchie verwijderd.
V8.0.1.0	15-mei-2017	RfC 71719: PKIO/UZI certificaten gaan over naar de Public G3/Private G1 generatie.
V8.0.1.0	15-mei-2017	RfC 76206: SSL verwijderen
V8.0.3.0	15-nov-2018	Opgenomen in publicatie 8.0.3.0

2 Het authenticatietoken

In dit hoofdstuk wordt concreet de inhoud van het authenticatietoken besproken die bij berichtauthenticatie met behulp van de UZI-pas wordt gebruikt.

2.1 Structuur

Het authenticatietoken die gebruikt wordt bij berichtauthenticatie met behulp van de UZI-pas voor het landelijk EPD heeft de volgende structuur:

```
<signedData xmlns="http://www.aortarelease.nl/805/"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
  wsu:Id="token_2.16.528.1.1007.3.3.1234567.1_0123456789">
  <authenticationData>
    <messageId>
      <root>2.16.528.1.1007.3.3.1234567.1</root>
      <extension>0123456789</extension>
    </messageId>
    <notBefore>20070128173600</notBefore>
    <notAfter>20070128174059</notAfter>
    <addressedParty>
      <root>2.16.840.1.113883.2.4.6.6</root>
      <extension>1</extension>
    </addressedParty>
  </authenticationData>
  <coSignedData>
    <triggerEventId>QURX_TE990011NL</triggerEventId>
    <patientId>
      <root>2.16.840.1.113883.2.4.6.3</root>
      <extension>012345672</extension>
    </patientId>
  </coSignedData>
</signedData>
```

2.2 Namespaces

Het authenticatietoken die gebruikt wordt bij berichtauthenticatie met behulp van de UZI-pas maakt gebruik van de volgende namespaces. De prefixen zijn niet normatief maar worden in dit document als voorbeelden gebruikt.

Tabel AORTA.STK.t3200 – Namespaces

Prefix	Namespace URI
-	http://www.aortarelease.nl/805/
-	http://www.w3.org/2000/09/xmldsig#
soap	http://schemas.xmlsoap.org/soap/envelope/
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd

In de voorbeelden wordt voor de eenvoud gewerkt zonder namespace-prefixes, dus:

```
<signedData xmlns="http://www.aortarelease.nl/805/..."
```

in plaats van:

```
<ao:signedData xmlns:ao="http://www.aortarelease.nl/805/..."
```

Namespace prefixes mogen echter wel gebruikt worden, dus ontvangers moeten tokens met prefixes goed kunnen verwerken. Bij het gebruik van prefixes is het wel van belang deze na ondertekening niet meer te veranderen, dit maakt de digitale handtekening ongeldig.

```
<signedData xmlns="http://www.aortarelease.nl/805/"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  wsu:Id="token_2.16.528.1.1007.3.3.1234567.1_0123456789">
```

Het token begint bij het `<signedData>` element. Het `<signedData>` element is geen HL7v3 element, maar gedefinieerd binnen deze specificatie.

De AORTA namespace `"http://www.aortarelease.nl/805/"` is verplicht en wordt als standaard namespace ingesteld voor de berichtauthenticatie met UZI-pas.

Een *Id* attribuut is verplicht (ook de hoofdletter I en kleine letter d). Voor het Id wordt de WS-Security 1.0 richtlijn van een Id in de wsu namespace gevolgd. De waarde van het wsu:Id attribuut moet uniek zijn in het hele XML bestand. De waarde moet *globaal uniek* zijn voor AORTA berichten, zodat bij samenvoegen van meerdere XML bestanden (in een HL7v3 batch of anderzins) de waarde uniek blijft.

2.3 Inhoud

```
<authenticationData>
```

De volgende paragrafen beschrijven de verschillende basis kenmerken en beveiligingsgerelateerde gegevens die het authenticatietoken onderscheiden, zoals in [IH tokens generiek] beschreven is.

2.3.1 Uniekheid

```
<messageId>
  <root>2.16.528.1.1007.3.3.1234567.1</root>
  <extension>0123456789</extension>
</messageId>
```

Een uniek gegeven, uitgegeven door de verzender van het bericht. Deze moet gelijk zijn aan het uiteindelijk gebruikte HL7v3 message.Id.

2.3.2 Geldigheid

```
<notBefore>20070128173600</notBefore>
```

De tijd waarop het authenticatietoken geldig wordt. Dit hoeft niet de tijd te zijn waarop het bericht is aangemaakt. Vaak zal het complete bericht pas later aangemaakt worden. In principe is het mogelijk *notBefore* in de toekomst te zetten, en het bericht na deze tijd pas te verzenden. Wordt een bericht ontvangen voor *notBefore* is aangevangen, dan **moet** dit bericht geweigerd worden.

De tijd moet doorgegeven worden als Coordinated Universal Time (UTC), volgens de ook in HL7v3 gebruikte ISO 8601 notatie. Deze notatie is YYYYMMDDHHMMSS, bijvoorbeeld "20080225134130" zonder tussenvoegsels als "T", ":" etc. Er wordt geen gebruik gemaakt van timezones, dus "20080225134130+1" is niet toegestaan. De verzender en ontvanger zijn verantwoordelijk voor het eventueel converteren van lokale tijd naar UTC, en dus daarbij rekening houden met zomer- en wintertijd. De precisie is in seconden.

```
<notAfter>20070128174059</notAfter>
```

De tijd waarop het authenticatietoken vervalt. Wordt een bericht ontvangen nadat *notAfter* is verstreken, dan **moet** dit bericht geweigerd worden. Deze tijd is als bovenstaande tijd geformatteerd. Het aanbevolen verschil tussen *notBefore* en *notAfter* is 5 minuten. Het maximaal toegestane verschil is 90 minuten. Dit maximum dient voor berichten die niet direct, maar bijvoorbeeld 's nachts verzonden worden, of kort voor de aanvang van een consult, zodat er iets ruimere mogelijkheden voor batchgewijze processen zijn. Het wordt sterk aanbevolen dat voor berichten die direct verzonden worden (dus terwijl de zorgverlener of medewerker achter diens computer zit) niet afgeweken wordt van de periode van vijf minuten. Het gaat immers om het voorkomen van misbruik van onderschepte tokens, en vijf minuten is meer dan voldoende om de hele keten van vraag tot antwoord te doorlopen.

De geldigheidsduur van een token (*notAfter* minus *notBefore*) mag niet langer dan 90 minuten zijn. Wordt een bericht ontvangen waarin deze maximale geldigheidsduur overschreden is, dan **moet** dat bericht geweigerd worden, ook al is het tijdstip notAfter nog niet verstreken.

Richtlijn: de beste geldigheidsduur van een token (*notAfter* minus *notBefore*) voor online bevraging is 5 minuten.

2.3.3 Ontvanger

```
<addressedParty>
  <root>2.16.840.1.113883.2.4.6.6</root>
  <extension>1</extension>
</addressedParty>
```

Het applicatie-id van het geadresseerde zorgsysteem, hier de ZIM. Het betreft hier het systeem waaraan het authenticatietoken gericht is; dat hoeft niet hetzelfde te zijn als het systeem waaraan het bericht zelf gericht is. De waarden in de elementen zijn (voorlopig) vaste waarden, omdat authenticatie alleen naar de ZIM geschiedt (behoudens testsituaties).

2.3.4 Attributen

```
<coSignedData>
```

Met de authenticatie meegetekende gegevens uit het bericht. Dit zijn kopieën van gegevens die ook in het bericht voorkomen. Deze zijn vooral wenselijk omdat de authenticatiegegevens op zich niet verhinderen dat het token met een ander bericht (met gelijk message-Id) wordt gebruikt.

```
<triggerEventId>QURX_TE990011NL</triggerEventId>
```

Merk op dat het trigger event geen verplicht element in het HL7v3-bericht zelf is. Het wordt echter altijd opgenomen in het token, omdat bij elke interactie een trigger event gespecificeerd is. De Trigger Event identificeert het *berichttype* uniek, en bepaalt dus de intentie van de verzender. Deze meesturen verhindert veel soorten aanvallen, bijv. een authenticatie van een medicatiequery kapen en pogen deze te hergebruiken voor het opvragen van een waarneemdossier. Het is ook mogelijk de interactionId te gebruiken: deze wijzigt echter tussen versies van berichten, het triggerEventId niet, zodat een nieuwe versie van het bericht geen wijzigingen voor het token tot gevolg heeft. De Trigger Events zijn te vinden in het vocab bestand 2.16.840.1.113883.1.6.xml.

```
<contextCode>
      <codeSystem>2.16.840.1.113883.2.4.3.111.15.1</codeSystem>
      <code>KZDI</code>
</contextCode>
```

In het geval er sprake is van een opvraag op basis van een context dient ook de contextCode meegetekend te worden. Dit is in principe alleen het geval bij de generiekeQueryZorggegevens. Bij deze generieke query is het trigger event niet meer voldoende om de intentie van de verzender aan te geven, aangezien deze altijd gelijk is. Door het toevoegen van de contextCode wordt deze intentie wel weer expliciet gemaakt. De codes zijn te vinden in het vocab bestand 2.16.840.1.113883.2.4.3.111.15.1.xml.

```
<patientId>
  <root>2.16.840.1.113883.2.4.6.3</root>
  <extension>012345672</extension>
</patientId>
```

Voor berichten die betrekking hebben op een enkele patiënt, wordt het burgerservicenummer van de patiënt opgenomen. Dit maakt ook weer vele aanvallen onmogelijk, namelijk gegevens van een andere patiënt proberen op te vragen. Dit geldt voor alle berichten die betrekking hebben op één en niet meer dan één patiënt; ook als het burgerservicenummer niet voorkomt in het bijbehorende schema. Het burgerservicenummer is te vinden in het patientId met root 2.16.840.1.113883.2.4.6.3.

Voor berichten die geen betrekking hebben op een persoon wordt het BSN weggelaten. Het hele coSignedData blok ziet er dan zo uit:

```
<coSignedData>
  <triggerEventId>QURX_TE990011NL</triggerEventId>
</coSignedData>
```

Tenslotte wordt het token afgesloten.

```
</signedData>
```

2.4 Algoritmes

Om de integriteit en onweerlegbaarheid van het authenticatietoken te waarborgen wordt een XML Signature geplaatst, zoals beschreven in [IH tokens generiek]. Na plaatsen van de XML Signature kan de ontvanger, met gebruikmaking van het persoonsgebonden UZI certificaat van de verzender, vaststellen dat het authenticatietoken ondertekend is met de privé sleutel behorend bij het gebruikte en meegezonden certificaat.

De XML Signature van het authenticatietoken die gebruikt wordt bij berichtauthenticatie met behulp van de UZI-pas maakt gebruik van de volgende algoritmes, zoals beschreven in [IH tokens generiek].

Voor het berekenen van de hashwaarde wordt SHA-256 gebruikt. Voor de digitale handtekening wordt RSA gebruikt.

2.5 Opbouw

2.5.1 De headers

Eerst wordt het token – het `<signedData>` element aangemaakt. Een voorbeeld:

```
<ao:authenticationTokens xmlns:ao="http://www.aortarelease.nl/805/"
soap:actor="http://www.aortarelease.nl/actor/zim" soap:mustUnderstand="1">
  <signedData ...>
  ... Het eerder gedefinieerd signedData blok ...
  </signedData>
</ao:authenticationTokens>
```

Vervolgens moet het XML Signature blok in een WSS 1.0 SOAP Header opgenomen worden. Een voorbeeld:

```
<wss:Security xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" soap:actor="http://www.aortarelease.nl/actor/zim" soap:mustUnderstand="1">
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo ...>
     ...
   </SignedInfo>
   <SignatureValue>
     ...
   </SignatureValue>
   <KeyInfo>
     ...
   </KeyInfo>
  </Signature>
</wss:Security>
```

Wanneer er met strings gewerkt wordt bij het zetten van de digitale handtekening, moet hier niet meer met de strings (signedData en SignedInfo) gemanipuleerd worden, maar ze moeten octet-voor-octet overgenomen worden in het bericht. Strikt genomen is het toegestaan wijzigingen aan te brengen die door canonicalisatie bij de ontvanger weer opgeheven worden, maar wanneer de digitale handtekening door middel van strings wordt opgebouwd, is het een foutgevoelige handeling.

Lange Base 64 waarden zijn afgekort. Wederom kan dit als strings worden behandeld, waarbij drie waarden vervangen moeten worden.

Deze drie waarden worden ingevuld:

- Neem het SignedInfo blok op.
- Neem de SignatureValue op.

- Neem certificaatgegevens in het KeyInfo blok op.

Canonicalisatie speelt alleen bij die XML code waarover een SHA digest wordt berekend of waarover RSA handtekeningen worden gezet. Het maken van de XML Signature uit strings levert dus twee blokken tekst op: de signedData en de Signature.

2.5.2 Plaats van het token

Het authenticatietoken wordt opgenomen in een SOAP Header.

Voor authenticatie doeleinden mag er niet meer dan één authenticatietoken voorkomen.

```
<soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <ao:authenticationTokens xmlns:ao="http://www.aortarelease.nl/805/"
    soap:actor="http://www.aortarelease.nl/actor/zim" soap:mustUnderstand="1">
      <signedData xmlns="http://www.aortarelease.nl/805/" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="token_2.16.528.1.1007.3.3.1234567.1_0123456789">
      <authenticationData>
        ...
      </authenticationData>
      <coSignedData>
        ...
      </coSignedData>
    </signedData>
  </ao:authenticationTokens>
    ...
</soap:Header>
```

Het token – het `<signedData>` element – wordt opgenomen in het `<authenticationTokens>` element van de `<soap:Header>`. Dit is voor het geval er (in de toekomst) meerdere authenticatietokens in een bericht zouden kunnen voorkomen. Op het `<authenticationTokens>` element **moet** een `soap:mustUnderstand="1"` vlag opgenomen worden, die aangeeft dat de ontvanger deze header **moet** verwerken. Bij het `<authenticationTokens>` element **mag** met een `soap:actor="http://www.aortarelease.nl/actor/zim"` worden aangegeven dat de ZIM deze header verwerkt.

Het declareren van de namespace `"http://www.aortarelease.nl/805/"` bij `<signedData>` is niet nodig: deze wordt overgeërfd van het bovenliggende element. Bij canonicalisatie zal bij de ontvanger de namespace teruggeplaatst worden voor controle van de digitale handtekening. Voor de duidelijkheid is de namespace declaratie hier tweemaal opgenomen.

2.5.3 Plaats van de digitale handtekening

De digitale handtekening wordt in een WS-Security 1.0 SOAP Header gezet. Op het `<wss:Security>` element **moet** een `soap:mustUnderstand="1"` vlag opgenomen worden, die aangeeft dat de ontvanger dit security element **moet** verwerken. Bij het `<authenticationTokens>` element **mag** met een `soap:actor="http://www.aortarelease.nl/actor/zim"` worden aangegeven dat de ZIM dit security element verwerkt.

Wanneer een bericht een authenticatietoken bevat, moet dat bericht precies één bijbehorende digitale handtekening bevatten.

De volgorde van beide headers (authenticatietoken en digitale handtekening) is niet relevant: zowel digitale handtekening vóór het token als token vóór de digitale handtekening zijn geldig.

```
<soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  ...
  <wss:Security xmlns:wss=
    "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    soap:actor="http://www.aortarelease.nl/actor/zim" soap:mustUnderstand="1">
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      ...
      </SignedInfo>
      <SignatureValue>...</SignatureValue>
      <KeyInfo>
        <X509Data>
        ...
        </X509Data>
      </KeyInfo>
    </Signature>
  </wss:Security>
</soap:Header>
```

3 Certificaten

3.1 Te gebruiken certificaat en attributen

De UZI-pas kent een aantal modellen:

Tabel AORTA.STK.t3210 – UZI pastype

Naam UZI-pastype	Codering Pastype
Zorgverlenerpas	Z
Medewerkerpas op naam	N
Medewerkerpas niet op naam	M
Servercertificaat	S

De pas die gebruikt wordt voor het ondertekenen van een authenticatietoken moet een zorgverlenerpas of een medewerkerpas op naam zijn. Hoewel het pastype gecodeerd is opgenomen in het authenticiteitcertificaat (in het `subjectAltName` attribuut), dient een applicatie op basis van de uitgevende CA vast te stellen wat het pastype van de UZI-pas is.

De signature wordt gezet met de sleutel voor authenticiteit (keyUsage=digitalSignature, hexadecimaal 0x80).

De attributen in het authenticiteitcertificaat worden gegeven in de vorm van een Distinguished Name *(*DN), zie [IH tokens generiek].

De waarden van deze attributen voor de relevante UZI-passen zijn:

Tabel AORTA.STK.t3220 – DN attributen van zorgverlenerspas

Attribuut	Omschrijving	Waarde
CN	Issuer.commonName	SHA-2 generatie (G21): UZI-register Zorgverlener CA G21 Generatie Public G3: UZI-register Zorgverlener CA G3
O	Issuer.organisationName	SHA-2 generatie (G21): agentschap Centraal Informatiepunt Beroepen Gezondheidszorg Public G3/Private G1 generatie: CIBG
C	Issuer.countryName	NL

Tabel AORTA.STK.t3230 – DN attributen van medewerkerpas op naam

Attribuut	Omschrijving	Waarde
CN	Issuer.commonName	SHA-2 generatie (G21): UZI-register Medewerker op naam CA G21 Generatie Public G3: UZI-register Medewerker op naam CA G3
O	Issuer.organisationName	SHA-2 generatie (G21): agentschap Centraal Informatiepunt Beroepen Gezondheidszorg Public G3/Private G1 generatie: CIBG
C	Issuer.countryName	NL

Om de digitale handtekening bij het LSP te verifiëren, moet de ontvanger over de bijbehorende publieke sleutel beschikken, zie [IH tokens generiek].

Voor verificatie is gekozen een verwijzing naar het certificaat mee te zenden; de ontvanger moet deze dan met bijvoorbeeld het LDAP protocol ophalen in de directory van het UZI-register.

Zie voor de verdere beschrijving van de passen [UZI pas].

Noot: uiteraard mogen in het testtraject alleen UZI-testpassen gebruikt worden. Het gebruik hiervan wordt verder niet uitgewerkt in deze handleiding. De werking is identiek.

4 Token afhandeling

4.1 Verificatie met het bericht

Bij ontvangst van een bericht met authenticatietoken moet het *ontvangende systeem* het token en de digitale handtekening daarover controleren (zie [IH tokens generiek]) en controleren of het token overeenstemt met het bericht. Deze twee onderwerpen van controle worden hieronder beschreven.

Bij ontvangst van een bericht haalt het ontvangende systeem het token uit de SOAP header. Het ontvangende systeem controleert of:

- Het gebruikte certificaat geldig is, zie paragraaf 3.1Te gebruiken certificaat en attributen;
- De messageId in het token overeen komt met de gebruikte HL7v3 message.Id in het bericht, zie paragraaf 2.3.1 Uniekheid;
- Het bericht ontvangen is binnen de geldigheidsperiode van het token, zie paragraaf 2.3.2 Geldigheid;
- Het token bestemd is voor het geadresseerde zorgsysteem, zie paragraaf 2.3.3 Ontvanger;
- In het token het patientId staat, en dit patientId overeenkomt met het gebruikte HL7v3 patientId (indien in bericht aanwezig), zie paragraaf 2.3.4 Attributen;
- In het token het triggerEventId staat, zie paragraaf 2.3.4 Attributen

Als aan één van bovenstaande condities niet is voldaan, wordt het bericht geweigerd en wordt een SOAP foutmelding aan het verzendende systeem gegeven, zie foutafhandeling in [IH tokens generiek]. Als wel aan alle condities is voldaan, wordt het HL7v3 bericht verder verwerkt.

Als er geen authenticatietoken in de SOAP header is aangetroffen *kan* het HL7v3 bericht wel verder worden verwerkt, op voorwaarde dat voor de betreffende interactie het vertrouwensniveau "laag" is toegestaan.

Naast de controle van het token moeten bij het bericht een aantal gegevens worden vastgehouden die verderop worden gebruikt voor inhoudelijke controle van het bericht:

- Authenticatietoken aanwezig (ja/nee);
- UZI-nummer van de afzender van het bericht (afkomstig uit het certificaat van de afzender);
- URA-nummer van het systeem dat het bericht heeft verstuurd (afkomstig uit het servercertificaat van dat systeem, die onderdeel is van gegevens van de TLS sessie).

Bijlage A Referenties

Referentie	Document	Versie
[AETSDK]	Software Developers Kit beschikbaar gesteld door AET Europe B.V. Geeft toelichting bij de PKCS#11 stappen die nodig zijn voor het ondertekenen van een authenticatie 'challenge'. http://www.uziregister.nl	
[IH tokens generiek]	Implementatiehandleiding security tokens generiek	8.0.3.0
[EXCC14N]	Exclusive XML Canonicalization, Version 1.0, W3C Recommendation	1.0 18-juli-2002
[MIME]	Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies http://www.ietf.org/rfc/rfc2045.txt	
[UZI pas]	CA Model, Pasmodel, Certificaat- en CRL-profielen Zorg CSP. http://www.uziregister.nl/ https://www.uziregister.nl/overhetuziregister/hierarchie-productieomgeving	7.0 16 jan 2014

Bijlage B Technische context (gebruik UZI-pas)

B.1 Inleiding

Voor realisatie van de cryptografische beveiligingsfuncties wordt binnen AORTA gebruik gemaakt van de Private Key Infrastructuur (PKI) die geboden wordt door het UZI-register. Het UZI-register verstrekt smartcards (UZI-passen) aan zorgverleners en medewerkers van zorgaanbieders. De UZI-pas bevat X.509 certificaten voor authenticatie, vertrouwelijkheid en elektronische handtekeningen en de bijbehorende private keys. Bij zogenaamde Medewerkerpassen niet op naam ontbreekt het handtekeningcertificaat, maar deze pas is binnen AORTA niet bruikbaar.

De UZI-pas is een smartcard. Deze bevat een NXP microprocessor, operating systeem (NXP JCOP) en de nodige (Java Card) programmatuur om kleine programma's te kunnen draaien. Zie [UZITS] voor details. De smartcard ondersteunt het RSA algoritme voor public/private key cryptografie, het SHA-1 en SHA-256 algoritme voor hashing en AES algoritmen voor symmetrische versleuteling.

De smartcard kan benaderd worden via de meegeleverde SafeSign middleware van AET. Zie voor actuele versies van de software:

http://www.uziregister.nl/uzipas/gebruikuzipas/paszonderabonneenummer/

http://www.uziregister.nl/ondersteuning/handleidingeneninstallatie/

Het UZI-register ondersteunt deze middleware op Windows; Linux (SuSe en RedHat distributie) en MacOS X operating systeem[1].

B.2 Application Programming Interfaces voor benadering UZI-pas

De AET middleware biedt twee interfaces om de smartcard te benaderen:

1. De [PKCS#11] interface. Dit is een leveranciers onafhankelijke standaard API -ook wel Cryptoki genoemd- voor cryptografische modules om cryptografische informatie te benaderen (certificaten, sleutels) en om cryptografische functies aan te roepen die op deze modules worden uitgevoerd (zoals het ondertekenen van data). Deze interface is platformonafhankelijk en wordt voor de UZI-pas ondersteund op Windows, Linux en MacOS.
2. [MSCrypto] MS CryptoAPI is een microsoftstandaard en biedt een API om cryptografische functies aan te roepen die door een zogenaamde Cryptographic Service Provider worden uitgevoerd.

Naast verschil in platformondersteuning is een belangrijk verschil het abstractieniveau:

- PKCS#11 is een API die specifiek is ontwikkeld voor smartcards en andere hardware modules. Deze biedt tot in detail inzicht in de stappen die uitgevoerd worden en de objecten op de UZI-pas die benaderd worden. Dit geeft de ontwikkelaar meer controle over het gebruik van de UZI-pas.
- MS CryptoAPI is een interface op een hoger abstractieniveau en is niet 'smartcard aware'. Er worden crypto functies aangeroepen, maar de programmeur hoeft

[1] AET Europe B.V. –de leverancier van SafeSign- ondersteunt meer operating systemen dan het UZI-register.

daarbij niet te weten dat sommige functies in specifieke hardware uitgevoerd worden (zoals het signen van data op de UZI-pas zelf).

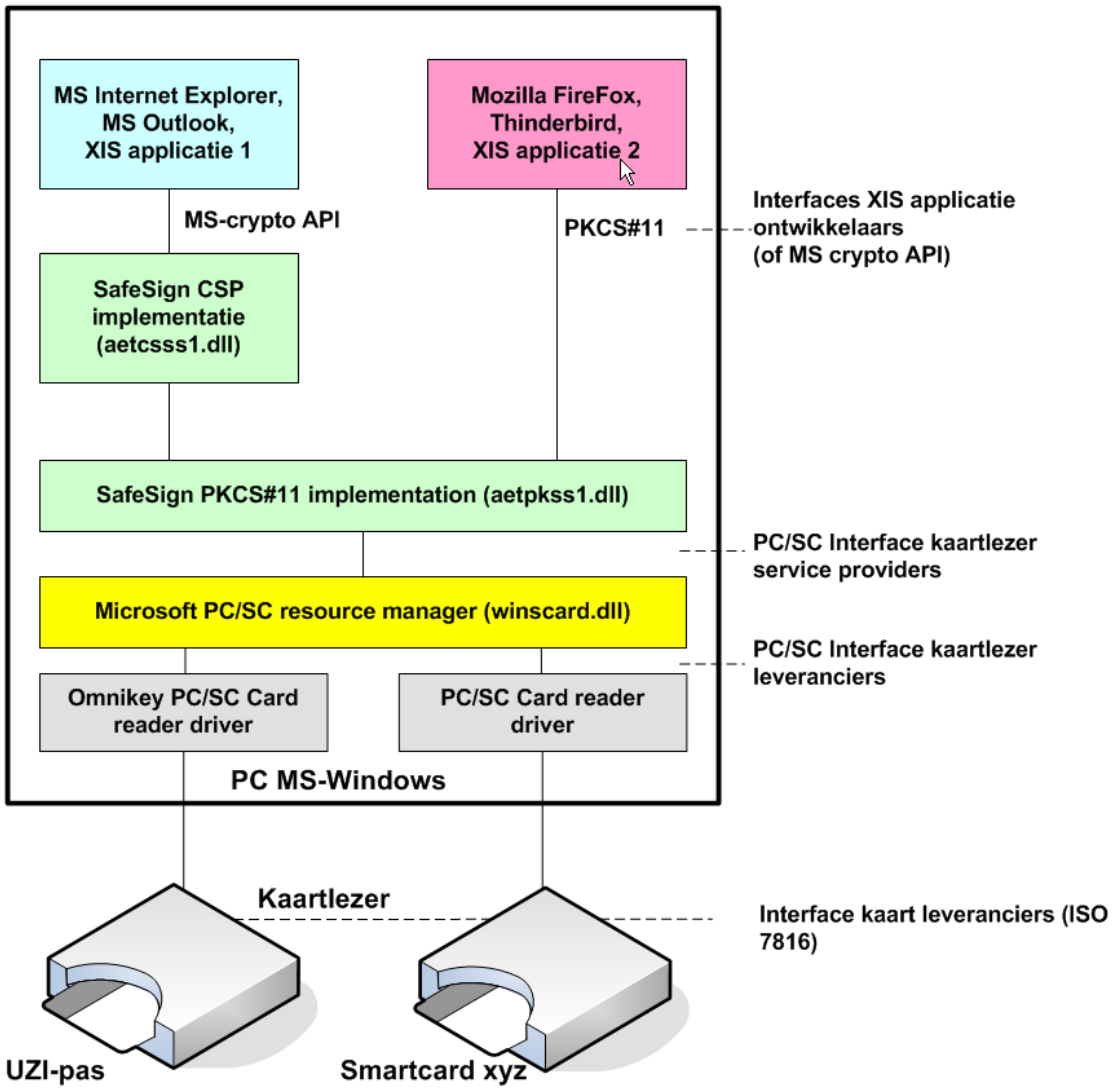
Het gebruik van de PKCS#11 interface heeft de voorkeur vanwege de volgende overwegingen:

- Open standaard die breed gedragen is in de industrie;
- Platform (Operating System) onafhankelijk;
- Performance is beter en beter te optimaliseren;
- Meer controle over het exacte aanroepen van de UZI-pas vanuit de toepassing.

Dit is de reden dat in de voorbeelden alleen PKCS#11 functies zijn genoemd.

B.3 Software architectuur voor benadering UZI-pas

Onderstaand figuur geeft schematisch en globaal weer welke software componenten gebruikt worden bij het aanroepen van de UZI-pas vanuit een XIS-applicatie. Beide interfaces zijn hierbij weergegeven.



Figuur 1: Software architectuur bij gebruik UZI-pas op MS-Windows PC

Toelichting standaard interfaces

Rechts in de figuur zijn de belangrijkste standaard interfaces aangegeven. Deze interfaces realiseren de volgende zaken:

- Generieke interfaces voor XIS-applicatie ontwikkelaars: PKCS#11 en MS Crypto API (Cryptographic Service Provider). Deze standaarden schermen de technische complexiteit van de UZI-pas af en maakt de XIS-applicatie onafhankelijk van de kaartlezer, hardware van de UZI-pas en het operating system op de UZI-pas;
- De PC/SC interface voor middleware en kaartlezer software. Deze standaard maakt het gebruik van de UZI-pas mogelijk met kaartlezers van diverse leveranciers (Omnikey, Gemplus, etc.) en van verschillende fysieke interfaces met de PC (USB, PCMCIA, Serieel). De PC/SC resource managers (winscard.dll) zijn componenten van het operating system en zorgt dat meerdere applicaties logisch gezien tegelijkertijd met 1 UZI-pas kunnen communiceren. Dit is hieronder apart toegelicht;
- De ISO 7816 standaard. Deze maakt het mogelijk om in de toekomst smartcards te gebruiken van andere leveranciers (hardware chip en card operating system) onafhankelijk van de in gebruik zijnde kaartlezers en applicaties.

Toelichting PC/SC resource management

Met de SafeSign middleware is het mogelijk dat meerdere client-applicaties logisch gezien 'tegelijkertijd' gebruik maken van de UZI-pas, bijvoorbeeld Outlook, IE, FireFox en één of meer lokale XIS-clients.

Op het laagste niveau kan de UZI-pas slechts communiceren met één applicatie omdat het (zoals de meeste smartcards) een ‘single-threaded device’ is dat sequentieel alle communicatie afhandelt. Dit hoeft geen probleem te zijn want bij printers of database records kan er uiteindelijk ook maar één applicatie tegelijk gebruik maken van de resource. De PC/SC resource manager van MS-Windows dwingt af dat iedere applicatie afzonderlijk de UZI-pas 'claimt' en de communicatie afhandelt. Op dat moment blokkeren de andere toepassingen totdat de communicerende toepassing klaar is. Vervolgens kan een andere toepassing communiceren met de UZI-pas. De SafeSign middleware communiceert dus altijd via de PC/SC resource manager met de UZI-pas.

Ondanks het resource management vanuit het OS en de middleware zijn er wel een aantal richtlijnen vanuit de PC/SC standaard voor de applicaties[2]. Ook als er op hoog abstractieniveau ontwikkeld wordt –door gebruik te maken van third party libraries- is het van groot belang om de onderliggende PC/SC architectuur goed te begrijpen. Tevens dient men te verifiëren dat de applicatierichtlijnen uit de PC/SC standaard ingevuld zijn, samengevat:

- Communiceer alleen met de pas als het nodig is;
- Geef geen reset naar de smartcard;
- Claim geen exclusieve toegang tot de smartcard behalve als dat noodzakelijk is;
- Beëindig connecties bij afsluiten van de applicatie[3].

[2] Deel 7 bevat de *Application Domain and Developer Design Considerations.* Paragraaf 3.3 gaat over *Device Sharing and Control*. Zie www.pcscworkgroup.com.

[3] De middleware zorgt ervoor dat PKCS#11 of Crypto API aanroepen niet onnodig gecombineerd worden, maar dat kleine atomaire transacties uitgevoerd worden met de smartcard.

B.4 Gebruik van UZI-pas voor authenticatie

Op de website van het UZI-register is een [AETSDK] beschikbaar die toelichting geeft bij de stappen die nodig zijn voor het ondertekenen van een authenticatie challenge met behulp van PKCS#11 functies.

Deze stappen zijn bij tokenauthenticatie identiek aan de huidige authenticatie met TLS voor zover het gaat om het aanroepen van de UZI-pas. Het verschil zit in de "challenge/data/digestvalue/hashwaarde" die ondertekend wordt. Bij tokenauthenticatie is dat de hash-waarde van het security token en bij implementatie van een TLS client is dat de hash-waarde die conform de TLS specificaties wordt berekend tijdens de TLS handshake fase.

De stappen voor het gebruik van de UZI-pas zijn:

- Laad de PKCS #11 library
- Initialiseer de PKCS #11 library (C_Initialize)
- Zoek een token[4] (C_GetSlotList, C_WaitForSlotEvent)
- Open een session met het token (C_OpenSession)
- Zoek het authenticatiecertificaat op het token (C_FindObjects...)
- Log in met PIN (C_Login)
- Zoek de bijbehorende sleutel (C_FindObjects...)
- Teken de challenge/digestvalue (C_Sign...)
- Log uit (C_Logout)
- Beëindig sessie (C_CloseSession)
- Sluit PKCS #11 af (C_Finalize)

Voor een verdere toelichting wordt verwezen naar de [AETSDK] die een presentatie bevat waarin iedere PKCS#11 functie aanroep is toegelicht. Daarnaast bevat de SDK ook voorbeeldcode.

[4] In de PKCS#11 context is een 'token' een hardware token: een smartcard.

Bijlage C Token implementatie

Deze bijlage met een voorbeeld van het tekenen gaat uit van het gebruik van SHA-1.

C.1 Maak een signedData blok

Het token moet aangemaakt worden en voorzien van de gegevens uit het bericht.

Het volgende is een voorbeeld van een token:

```
<signedData xmlns="http://www.aortarelease.nl/805/"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="_2.16.528.1.1007.3.3.1234567.1_0123456789">
<authenticationData>
<messageId>
<root>2.16.528.1.1007.3.3.1234567.1</root>
<extension>0123456789</extension>
</messageId>
<notBefore>20070128173600</notBefore>
<notAfter>20070128174059</notAfter>
<addressedParty>
<root>2.16.840.1.113883.2.4.6.6</root>
<extension>1</extension>
</addressedParty>
</authenticationData>
<coSignedData>
<triggerEventId>QURX_TE990011NL</triggerEventId>
<patientId>
<root>2.16.840.1.113883.2.4.6.3</root>
<extension>012345672</extension>
</patientId>
</coSignedData>
</signedData>
```

De rode tekstwaarden worden vervangen door de waarden voor het betreffende bericht. De groene teksten zijn 'vaste' teksten (voor communicatie met de ZIM). De tekst kan aangemaakt worden in ASCII of UTF-8. Gebruik wederom bij voorkeur een lange string zonder regeleinden.

C.2 Maak het signedData blok canoniek

Het signedData blok moet canoniek gemaakt worden volgens [EXCC14N]. Bovenstaand voorbeeld is canoniek (m.u.v. regeleinden voor de leesbaarheid). Voor de canonieke vorm zijn de volgende zaken van belang:

- Het Id attribuut moet double quoted zijn.
- De namespaces moeten voor het wsu:Id attribuut staan. Deze specificatie vermijdt het gebruik van attributen zoveel mogelijk om de canonicalisatie makkelijker te maken: in dit ene geval is dit niet mogelijk. Let op dat namespaces en wsu:Id gescheiden zijn met een enkele spatie: dit is in het voorbeeld hierboven niet meer zichtbaar.
- De namespaces moeten alfabetisch gerangschikt zijn.
- Alle regeleinden moeten met linefeed worden gedaan. Door de standaardtools voor bewerking van platte tekst zal op Unix-systemen een linefeed gebruikt worden, op MacIntosh een carriage return en op Windows carriage return + linefeed. Eenvoudiger is alle whitespace (regeleinden, tabs, spaties) weg te laten uit de signedData. In de voorbeeldbestanden is dat ook gedaan. Voor de leesbaarheid worden hier wel nieuwe regels gebruikt. (Maar: alle berekeningen zijn gedaan zonder whitespace, dus voor het narekenen van bijvoorbeeld een

SHA-1 digest moeten de regeleinden wel verwijderd worden.) In het algemeen geldt dat whitespace tussen de verschillende <tag>s significant is: XML is immers een documentformaat, waar whitespace een belangrijk deel van de opmaak is. Whitespace tussen de tags is dus toegestaan: een conformant XML processor zal deze ook nooit "zomaar" wijzigen. Veel XML-toepassingen als de DOM en XSLT maken het mogelijk expliciet aan te geven hoe met whitespace omgegaan moet worden. Bij verwerking van digitale handtekeningen is het altijd nodig "preserveWhitespace" of vergelijkbare attributen op "True" te zetten. Exclusive Canonicalization zal niets met de whitespace tussen tags doen (wél met whitespace binnenin tags).

- De SHA-1 word berekend over het signedData element, dus ook geen whitespace of newline voor of na het element meetekenen.
- Gebruik geen whitespace binnenin de tags, met uitzondering van een enkele spatie voor de attributen.
- Sla signedData op als ASCII of UTF-8. Dat is mogelijk omdat alle gegevens getallen of codes zijn (en niet bijv. namen van patiënten). Eigenlijk moeten bestanden die getekend worden in UTF-8 encoding zijn. De eerste 128 karakters (0 t/m 127) van UTF-8 hebben echter dezelfde hexadecimale waarden als ASCII. Er mogen beslist geen karakters in voorkomen uit de range 128-256 van b.v. ISO-Latin of Windows-1292: deze zijn niet hetzelfde in UTF-8. Voor het authenticatietoken zijn ook geen andere waarden nodig dan de eerste 128 ASCII karakters.
- Geen Byte Order Mark (BOM) toevoegen. UTF-8 staat het gebruik van een BOM toe. Applicaties kunnen deze gebruiken om te bepalen of de byte-ordering Little Endian of Big Endian is. Canonieke XML staat echter geen BOM toe – dat is ook geen probleem, aangezien UTF-8 verplicht is. Bij het gebruik van UTF-8 strings is het bij het wegschrijven dus van belang geen BOM te schrijven. Het gebruik van een BOM kan gedetecteerd worden door het bestand waarin de signedData is weggeschreven te openen: als de eerste drie octets EF BB BF zijn, is dit de BOM, en is het bestand niet correct. Als de eerste octets anders zijn, is er geen BOM gebruikt.

Of (in plaats van het bovenstaande)

- Maak het signedData blok canoniek volgens Exclusive XML Canonicalization met een geschikte library.

C.3 Bereken de SHA-1 of SHA-256 digest van het signedData blok

Van de signedData moet een SHA-1 of SHA-256 digest berekend worden. Deze digest – een verzameling octets – wordt gecodeerd met Base 64.

C.4 Maak het SignedInfo blok

XML Signature maakt gebruik van een SignedInfo blok voor de digitale handtekening. Dit ziet er als volgt uit:

```
<SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
</CanonicalizationMethod>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
</SignatureMethod>
<Reference URI="#_2.16.528.1.1007.3.3.1234567.1_0123456789">
<Transforms>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></Transform>
</Transforms>
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
```

```
<DigestValue>xx5...Gus2g=</DigestValue>
</Reference>
</SignedInfo>
```

Wederom is in de voorbeeldbestanden een string gebruikt zonder linefeeds, maar zijn deze hier voor de leesbaarheid toegevoegd.

Dit hele blok tekst kan als ASCII of UTF-8 string in de code gebruikt worden. Er zijn twee delen, wederom in rood, die vervangen moeten worden.

In de Reference moet de juiste Id worden toegevoegd: dit is de waarde van het Id-attribuut uit het signedData element.

In DigestValue wordt de in de vorige paragraaf berekende SHA-1 digest, in Base 64 vorm, geplaatst.

De SignedInfo strings moeten een SignedInfo blok opleveren in canonieke vorm. Het bovenstaande blok is in canonieke vorm, wanneer tenminste alle regeleinden verwijderd worden. Verder zijn nog twee zaken van belang:

- Lege tags: <tag att="1"/> moeten voorzien worden van een expliciete eind-tag: <tag att="1"></tag>
- De namespace van XML Signature moet opgenomen zijn in het SignedInfo element. Bij genereren van een XML Signature uit een toolkit zal deze namespace vaak in het omliggende <Signature> element opgenomen zijn. Bij gebruik van [EXCC14N] wordt deze bij het canoniek maken van <SignedInfo> hier toegevoegd.

C.5 Selecteer het authenticatiecertificaat en de bijbehorende private key

De volgende cryptografische objecten op de UZI-pas zijn noodzakelijk bij berichtauthenticatie:

1. Het authenticatiecertificaat.
2. De private key behorende bij het authenticatiecertificaat. Deze key wordt gebruikt om de DigestValue te ondertekenen. Deze bewerking wordt op de UZI-pas zelf uitgevoerd.

Globaal zijn de volgende stappen nodig.

1. Zoek het authenticatiecertificaat:
   - Normaalgesproken met de PKCS#11 functie C_FindObjects;
   - Certificaten zijn uit elkaar te houden op basis van de X.509 keyUsage;

Er zijn ook hulpfuncties beschikbaar in de [AETSDK] waarmee eenvoudig het juiste certificaat is te selecteren, waaronder C_FindAuthenticationCertificate.

2. Zoek de bij het authenticatiecertificaat behorende private key:
   - Deze sleutel kan bij het certificaat worden gezocht op basis van het CKA_ID attribuut

- Normaalgesproken met C_FindObjects...

Er zijn ook hulpfuncties beschikbaar in de [AETSDK] die eenvoudige selectie van de juiste private key bij een bepaald certificaat mogelijk maken: C_FindCertificatePrivateKey.

In de [AETSDK] is een nadere toelichten te vinden van bovenstaande (hulp)functies.

Zie verder:
http://www.uziregister.nl/veelgesteldevragen/certificaten/hoegebruikikvandedriecertificatendejuisteomteauthenticeren.asp

C.6 Bereken de "RSA with SHA-1" waarde over SignedInfo

Bereken de "RSA with SHA-1" signature value over het SignedInfo blok met het certificaat en plaats deze in een <Signature> element. Zorg ervoor dat de signature in Big Endian vorm wordt gemaakt. Bij netwerkcommunicatie wordt altijd Big Endian gebruikt. O.a. Windows gebruikt normaal Little Endian. Bewerk de resulterende octets volgens Base 64:

```
<SignatureValue>DpB...GW7A==</SignatureValue>
```

Base 64 staat wel whitespace zoals spaties en regeleinden toe, dus ook met regeleinden is de waarde geldig.
Base64 moet gebuikt worden zoals gespecificeerd in [MIME].

Wanneer SHA-256 wordt gebruikt, wordt "RSA with SHA-256" als type voor de handtekeningwaarde gekozen.

C.7 Neem het juiste certificaat op

De volgende string wordt aangemaakt voor de certificaatverwijzing. Uiteraard wordt de juiste referentie naar het gebruikte certificaat ingevoegd in het voorbeeld.

```
<KeyInfo>
<wss:SecurityTokenReference>
<ds:X509Data xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:X509IssuerSerial>
<ds:X509IssuerName>CN=TEST UZI-register Zorgverlener CA G21, O=agentschap Centraal
Informatiepunt Beroepen Gezondheidszorg, C=NL</ds:X509IssuerName>
<ds:X509SerialNumber>359...195</ds:X509SerialNumber>
</ds:X509IssuerSerial>
</ds:X509Data>
</wss:SecurityTokenReference>
</KeyInfo>
```

Uitgangspunt is dat de WSS namespace gedeclareerd is zoals hierboven opgenomen. Afhankelijk van de gebruikte pas moet de eerste rode string vervangen worden door de betreffende CA (zie AORTA.STK.t3220 en AORTA.STK.t3230). De tweede rode string wordt vervangen door het serial number van de pas (decimaal).