

Implementatiehandleiding Tokenauthenticatie en Elektronische Handtekening



Betere zorg door betere informatie

postadres: Postbus 19121, 2500 CC Den Haag
bezoekadres: Oude Middenweg 55, 2491 AC Den Haag
telefoon: (070) 317 34 50; fax: (070) 320 74 37; e-mail: servicedesk@infoEPD.nl
www.nictiz.nl

Versie : 6.0.3.0
Datum : 18 september 2009

Inhoudsopgave

1. Inleiding	3
1.1 Doel en doelgroep	3
1.2 Versie, status en wijzigingshistorie	3
1.2.1 Versie.....	3
1.2.2 Status	3
1.2.3 Wijzigingshistorie.....	3
1.3 Achtergrond.....	4
1.4 Reikwijdte	4
1.5 Structuur.....	4
1.6 Samenhang met andere documenten	4
2. Uitgangspunten	6
2.1 Normatieve referenties.....	6
2.2 Informatieve referenties.....	7
2.3 Afkortingen en begrippen	7
3. Functionele context.....	8
3.1 Beveiliging vanaf het werkstation	8
3.2 Beveiliging vanaf de perimeter	9
3.3 Uitgesteld opvragen.....	9
4. Technische context (gebruik UZI-pas)	10
4.1 Inleiding.....	10
4.2 Application Programming Interfaces voor benadering UZI- pas.....	10
4.3 Software architectuur voor benadering UZI-pas	11
4.3.1 Toelichting standaard interfaces	12
4.3.2 Toelichting PC/SC resource management	12
4.4 Gebruik van UZI-pas voor authenticatie	13
5. Technologieën	14
5.1 XML Signature	14
5.2 XML Canonicalization	14
5.3 WS-Security	15
5.4 WS-Security X.509 Profile	15
5.5 Basic Security Profile	16
6. Ontwerpoverwegingen tokenauthenticatie.....	17
6.1 X.509 Strong One-way Authentication.....	17
6.2 Uitbreiding op het token	17
6.3 Canonicalisatie.....	18
7. Het authenticatietoken	19
7.1 Inhoud van het token.....	19
7.2 Verificatie met het bericht	25
8. Digitale handtekening over het authenticatietoken	27

8.1	Inhoud van de handtekening	27
9.	Certificaten	30
9.1.1	Te gebruiken certificaat	30
9.1.2	Certificaat meezenden	31
9.1.3	Certificaatverwijzingen	32
10.	Het maken van de XML Signature	35
10.1.1	Maak een signedData blok	35
10.1.2	Maak het signedData blok canoniek	35
10.1.3	Bereken de SHA-1 digest van het signedData blok.....	36
10.1.4	Maak het SignedInfo blok	36
10.1.5	Selecteer het authenticatiecertificaat en de bijbehorende private key	37
10.1.6	Bereken de "RSA with SHA-1" waarde over SignedInfo.....	38
10.1.7	Neem het juiste certificaat op	38
10.1.8	Maak de headers	39
11.	Berichtenverkeer met token en handtekening	40
11.1	Plaats van het token	40
11.2	Plaats van de digitale handtekening	40
11.3	WSDL en Endpoints.....	41
11.4	Foutafhandeling	41
11.5	Relatie met pincode, sessies en SSL.....	42

1. Inleiding

Dit hoofdstuk is niet normatief.

1.1 Doel en doelgroep

Dit document heeft tot doel het gebruik van digitale handtekeningen in de zorgsector nader te specificeren. De term "digitale handtekeningen" wordt gebruikt voor de techniek van het ondertekenen van digitale gegevens met cryptografische technieken. Dit staat los van specifieke toepassingen. Digitale handtekeningen worden ingezet voor twee toepassingen: vaststellen van de identiteit van een inzender/aanmaker van een bericht (authenticatie) en wettelijk geldige elektronische handtekeningen. De technische aspecten van deze toepassingen worden ook in deze handleiding beschreven.

Dit document is vooral bedoeld voor softwareontwikkelaars van zorgapplicaties en zorg-infrastructurele applicaties, die op grond van de HL7v3 communicatiestandaard en op grond van dit document berichten willen ondertekenen.

De lezer wordt verondersteld kennis te hebben van [XML], [Namespaces] en [SOAP]. Lezing van de [IH Berichttransport], en overige relevante delen van de AORTA-documentatie in samenhang met dit document wordt ten zeerste aanbevolen.

1.2 Versie, status en wijzigingshistorie

1.2.1 Versie

Dit is versie 6.0.3.0 van het document "Implementatiehandleiding Tokenauthenticatie en Elektronische Handtekening".

1.2.2 Status

De status van deze versie is "Definitief".

1.2.3 Wijzigingshistorie

Wijzigingen in versie 6.0.3.0 ten opzichte van versie 1.1:

- Issue 2586: Overdrachts berichten zonder BSN: De eisen zijn zo aangepast dat voor ieder patiëntgerelateerd bericht het BSN in het token opgenomen **moet** worden.
- Issue 2587: Indirect versturen en destination: Bij indirect versturen werd ten onrechte geëist dat addressedParty in token == applicatie-id in receiver. Nu is de eis dat addressedParty de ZIM is.
- Issue 2588: Complete lijst triggers: De lijst met trigger events voor verificatie met bericht was niet compleet, deze is aangepast in de handleiding en de schematron.
2588: Complete lijst triggers
- Fouten in de waarden van hashes in voorbeelden aangepast.
- Eisen versoepeld voor mogelijk maken extern gastgebruik.

- RFC 24246: Het corrigeren van de Trigger Event (van belang voor token authenticatie) van de Interactie: *Find Act Reference Registry Entries, Query (QUMT_IN020011NL)*. De trigger event was foutief opgenomen in v6 van de publicatie. Zie paragraaf 7.2

1.3 Achtergrond

Dit document is geschreven in het kader van het AORTA-programma van Nictiz. Het ministerie van VWS, CIBG en Nictiz werken samen met partijen in het veld aan de invoering van een landelijk elektronisch patiëntendossier (EPD).

1.4 Reikwijdte

Dit document specificeert het gebruik van digitale handtekeningen voor het landelijk EPD. In de basisinfrastructuur en binnen zorgtoepassingen wordt specifiek beschreven wanneer en waarvoor deze techniek wordt ingezet.

1.5 Structuur

Dit document begint met twee inleidende hoofdstukken.

In hoofdstuk 3 wordt de functionele context beschreven: de systemen waarmee zorgverleners werken.

Hoofdstuk 4 beschrijft de technische context, met name gebruik van de UZI pas voor authenticatie.

Hoofdstuk 5 beschrijft de gebruikte XML- en Web Services technologieën.

Hoofdstuk 6 noemt de ontwerpoverwegingen bij het totstandkomen van tokenauthenticatie.

Hoofdstuk 7 gaat in op de structuur van het mee te sturen authenticatietoken.

Hoofdstuk 8 beschrijft de digitale handtekening die over het authenticatietoken gezet wordt.

Hoofdstuk 9 gaat in op de rol van de certificaten en de relatie tussen certificaat en digitale handtekening.

Hoofdstuk 10 gaat in op het proces van het samenstellen van de XML fragmenten authenticatietoken en digitale handtekening.

Hoofdstuk 11 gaat in op de plaats van de XML fragmenten in SOAP headers, en de berichten die daarmee verzonden worden.

1.6 Samenhang met andere documenten

Dit document maakt deel uit van de documentatieset AORTA-basisinfrastructuur. Die documentatieset, nader beschreven het [Documentatieoverzicht], definieert de

basisinfrastructuur voor berichtenuitwisseling in de Zorg. Dit document is een van de implementatiehandleidingen, die zijn afgeleid van het architectuurontwerp en daarvan met name de technische architectuur.

Dit document hangt nauw samen met de [IH Berichttransport] en de [IH Berichtwrappers].

2. Uitgangspunten

Dit hoofdstuk is niet normatief.

2.1 Normatieve referenties

De onderstaande documenten zijn beschouwd als leidend voor dit document:

Identificatie	Titel	Bron	Versie Datum
[AETSDK]	Deze Software Developers Kit is beschikbaar gesteld door AET Europe B.V. en geeft toelichting geeft bij de PKCS#11 stappen die nodig zijn voor het ondertekenen van een authenticatie 'challenge'. Naast een presentatie waarin iedere PKCS#11 functie aanroep is toegelicht bevat de SDK ook PKCS#11 voorbeeldcode.	www.uziregister.nl	
[BSP]	Basic Security Profile Version 1.0, Final, 2007-03-30	www.ws-i.org	1.0 2007-03-30
[EXCC14N]	Exclusive XML Canonicalization, Version 1.0, W3C Recommendation, 18 July 2002	www.w3.org	1.0 18 July 2002
[HL7v3 ballot7]	HL7 Version 3 Standard, ballot #7	www.hl7.org	
[LDAPUTF8]	Lightweight Directory Access Protocol (v3) :UTF-8 String Representation of Distinguished Names, IETF, December 1997	www.ietf.org/rfc/rfc2253.txt	December 1997
[MIME]	Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies	www.ietf.org/rfc/rfc2045.txt	
[MSCrypto]	Microsoft Cryptography API is an application programming interface included with Microsoft Windows operating systems that provides services to enable developers to secure Windows-based applications using cryptography.	msdn2.microsoft.com/en-us/library/aa380256.aspx	
[Namespaces]	Namespaces in XML 1.0 (Second Edition)	www.w3.org/TR/xml-names/	1.0
[PKCS#11]	Public-Key Cryptography Standards#11 :Cryptographic Token Interface Standard. This standard specifies an API, called Cryptoki, to devices which hold cryptographic information and perform cryptographic functions.	http://www.rsa.com/rsalabs/node.asp?id=2133	
[SOAP]	Simple Object Access Protocol (SOAP) 1.1	http://www.w3.org/TR/SOAP	
[UZIPAS]	CA model, Pasmodel, Certificaat- en CRL-profielen, Agentschap CIBG, v. 2.2, november 2007	www.uziregister.nl	2.2 november 2007
[UZITS]	Toegepaste standaarden, Landelijk Operationeel UZI-register, 1.2,4 januari 2006, CIBG	www.uziregister.nl	1.2 4 januari 2006
[WSDL]	Web Services Description Language (WSDL) 1.1, W3C Note, 15 March 2001	http://www.w3.org/TR/wsdl	15 March 2001

Identificatie	Titel	Bron	Versie Datum
[WSS]	Web Services Security: SOAP Message Security 1.0, OASIS Standard Specification, March 2004	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf	March 2004
[WSX509]	Web Services Security X.509 Certificate Token Profile 1.0, OASIS Standard Specification, March 2004	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf	March 2004
[X509]	Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks, Recommendation X.509, 08-2005	http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-200508-I!!PDF-E&type=items	08-2005
[X509CRL]	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	http://www.ietf.org/rfc/rfc3280.txt	
[XML]	Extensible Markup Language (XML) 1.0, W3C Recommendation, Fourth Edition, 16 August 2007	http://www.w3.org/TR/xml	16 August 2007
[XMLSIG]	XML-Signature Syntax and Processing, W3C Recommendation, 12 February 2002	http://www.w3.org/TR/xmlsig-core/	12 February 2002

2.2 Informatieve referenties

De onderstaande documenten hebben gediend als bron voor dit document:

Identificatie	Titel	Bron	Versie Datum
[Documentatieoverzicht]	Documentatieoverzicht AORTA-basisinfrastructuur	Nictiz	6.0.1.0 16 april 2009
[Verklarende woordenlijst]	Verklarende woordenlijst AORTA	Nictiz	6.0.0.0 31 oktober 2008
[IH berichtwrappers]	Implementatiehandleiding berichtwrappers	Nictiz	6.0.0.0 31 oktober 2008
[IH berichttransport]	Implementatiehandleiding berichttransport	Nictiz	6.0.0.0 31 oktober 2008

2.3 Afkortingen en begrippen

Alle afkortingen en begrippen die relevant zijn voor het voorliggende document zijn opgenomen in de [[Verklarende woordenlijst](#)].

3. Functionele context

De meest basale vorm van beveiliging van gegevensopvraging zonder tokenauthenticatie met AORTA is het volgende scenario. Na dit scenario worden twee use cases beschreven die met tokenauthenticatie ondersteund moeten worden.

3.1 Beveiliging vanaf het werkstation

Een arts krijgt een patiënt op consult. De arts wil de medicatiehistorie van de patiënt opvragen. De arts neemt zijn UZI-pas en doet deze in de paslezer. De applicatie van de arts vraagt om diens pincode, en zet vervolgens een SSL/TLS sessie met wederzijdse authenticatie op met het LSP. De arts authenticceert zich daarbij met behulp van de private sleutel op de UZI-pas die gebruikt wordt bij het opzetten van de verbinding. Het LSP authenticceert zich met behulp van de private sleutel van die hoort bij het servercertificaat van het LSP. Over deze geauthenticceerde en beveiligde SSL/TLS sessie kan de arts vervolgens de medicatiehistorie opvragen, en eventuele vervolgacties doen zoals een elektronisch voorschrift verzenden. Na afronden van deze werkzaamheden wordt de beveiligde verbinding verbroken.

Dit scenario kent de volgende problemen:

- 1) In grotere organisaties met een complexere infrastructuur is het niet wenselijk dat een SSL/TLS verbinding vanaf een werkstation wordt opgezet. Deze verbinding is "ondoorzichtig" voor alle servers en firewalls binnen het GBZ. Dit is in strijd met een beveiligingsvoorschrift dat veel organisaties hanteren, nl. dat alle in- en uitgaand dataverkeer aan de "grens" van het netwerk wordt gecontroleerd op virussen en ander malafide verkeer.
- 2) Veel organisaties hebben een beveiligingsbeleid dat verbiedt om een directe verbinding naar buiten te maken vanaf een werkstation. De overweging daarbij is, dat werkstations moeilijker zijn te beveiligen dan serversystemen die logisch en fysiek goed beveiligd kunnen worden.
- 3) Bij grotere organisaties is er vaak sprake van meerdere infrastructuurcomponenten. Een HL7v3 bericht wordt veelal pas samengesteld op een speciale communicatieserver, en niet op het werkstation van de arts. Dit levert een complicatie op als de SSL/TLS sessie vanaf het werkstation wordt opgezet. In dat geval moet het HL7v3 bericht van de communicatieserver eerst "terug" naar het werkstation om via de SSL/TLS sessie verzonden te kunnen worden.
- 4) Artsen willen meestal de gegevens niet opvragen op het moment dat de patiënt tegenover hen zit, maar willen dat de gegevens dan al gereedstaan.

Er is een oplossing voor de eerste drie van deze problemen, "Authentication Forwarding", die hier niet verder beschreven wordt. Aangezien deze oplossing complex is, vraagt tevens oproept ten aanzien van de beveiliging, in de praktijk vaak tot technische problemen leidt en het vierde probleem niet ondervangt is er behoefte aan een andere oplossing.

De volgende scenario's worden daartoe ondersteund door deze specificatie.

Noot: vraagstukken over mandatering en gastgebruik zijn niet opgenomen in deze specificatie. Waar dus staat "de arts vraagt op" kan dat ook diens gemandateerde medewerker zijn.

3.2 Beveiliging vanaf de perimeter

Een patiënt wordt na een ongeval bij de EHBO binnengebracht. Een arts verricht een diagnose en besluit medicatie voor te schrijven. Voordat dit voorschrift daadwerkelijk verricht wordt, wil de arts de medicatiehistorie van de patiënt inzien. De arts vraagt de medicatiehistorie op. De applicatie van de arts voegt daarbij een authenticatietoken toe aan de vraag. De applicatie vraagt de arts om diens UZI-pas en pincode, en tekent het authenticatietoken. De handtekening en token worden doorgegeven aan andere applicaties binnen het GBZ. Deze worden daarna verzonden met een HL7v3 opvraagbericht over de beveiligde SSL/TLS sessie tussen het GBZ en het LSP.

In dit geval zet een systeem aan de perimeter (de "rand" van de ICT infrastructuur van het GBZ) bijvoorbeeld een communicatieserver, de SSL/TLS sessie op. Deze sessie is opgezet met het servercertificaat van het GBZ. Het LSP verifieert het token met de handtekening, en concludeert daaruit dat het opvraagbericht inderdaad is aangemaakt door een bevoegde zorgverlener, voert de opvraging uit en retourneert de medicatiehistorie (via het GBZ) naar de arts.

3.3 Uitgesteld opvragen

Een patiënt heeft een gepland consult bij een arts. De arts wil normaliter niet tijdens een consult gegevens opvragen via de ZIM bij andere GBZ'en: de tijd per consult is vaak zeer beperkt, en een wachttijd pleegt teveel inbreuk op het werkpatroon. De arts wil dus vooraf de benodigde gegevens op halen. De medewerker logt bij aanvang van de dienst van de arts in en haalt alle geplande afspraken op. Het systeem zet de benodigde gegevensopvragingen per consult klaar, maakt de authenticatietokens, vraagt de arts indien dat nog niet is gebeurd om diens pincode, en tekent de tokens. Een half uur voor aanvang van het consult haalt het systeem de benodigde gegevens bij de ZIM op, zodat de gegevens enerzijds zo recent mogelijk zijn, en anderzijds klaarstaan bij aanvang van ieder consult.

4. Technische context (gebruik UZI-pas)

4.1 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 Philips microprocessor, operating systeem (IBM 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 algoritme voor hashing en 3DES 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/technischesupport/installeren/kaartlezeruzi-pas/>.

Het UZI-register ondersteunt deze middleware op Windows; Linux (SuSe en RedHat distributie) en MacOS X operating systeem¹.

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

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

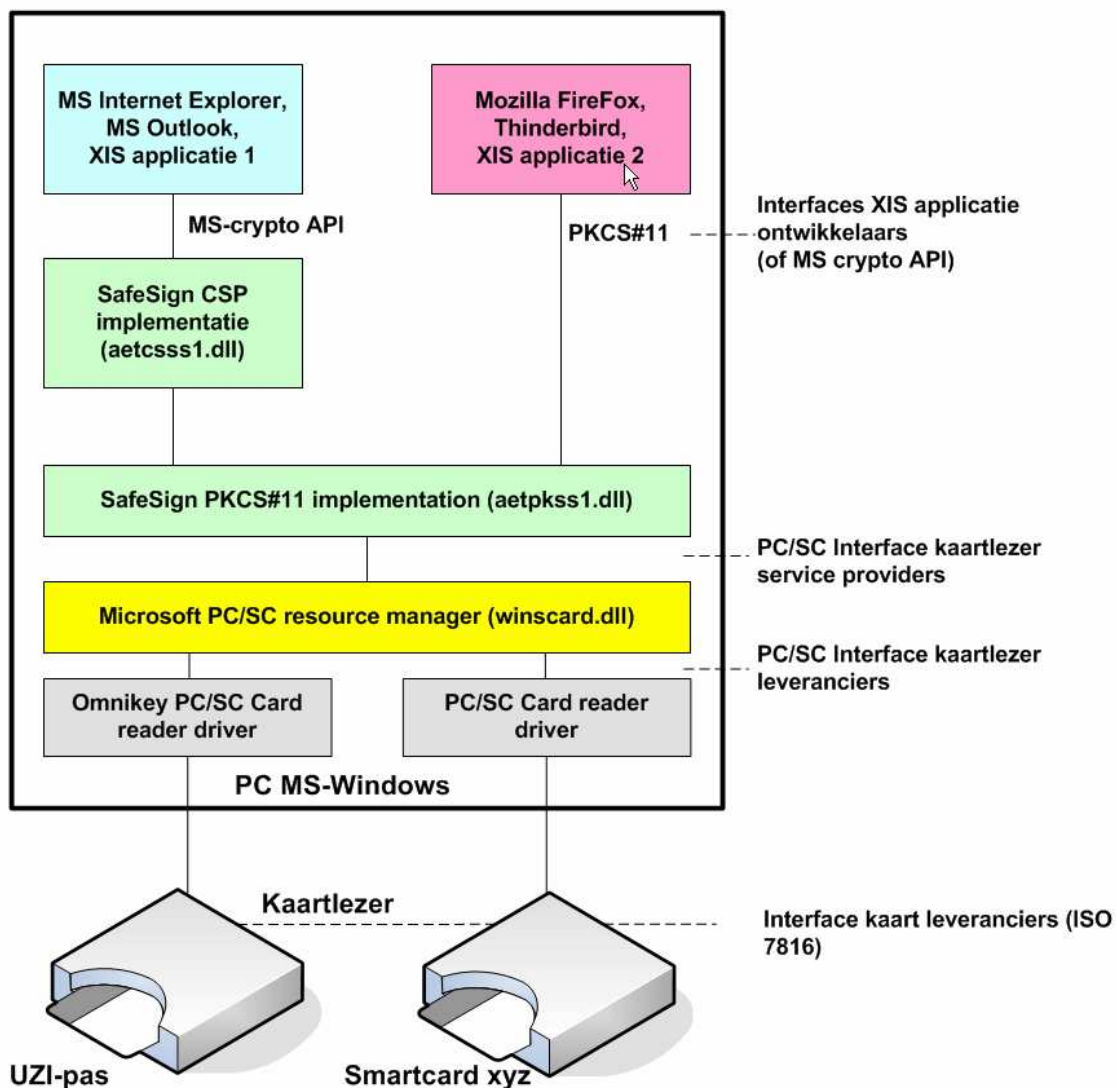
Het gebruik van de PKCS#11 interface heeft de voorkeur vanwege de volgende overwegingen:

- 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.

4.3 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

4.3.1 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.

4.3.2 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 applicaties². 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 applicatie³.

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.

4.4 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 SSL 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 SSL/TLS client is dat de hash-waarde die conform de SSL/TLS specificaties wordt berekend tijdens de SSL/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 token⁴ (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.

5. Technologieën

Dit hoofdstuk is niet normatief.

Overzicht van de gebruikte technologieën bij het digitaal ondertekenen.

5.1 XML Signature

XML Signature beschrijft de mogelijkheid om XML te signeren. Een deel van de XML wordt gesigneerd (als tekst) en de signature wordt als apart XML bestand gegenereerd of toegevoegd aan het XML document als Signature element.

5.2 XML Canonicalization

XML Canonicalization (kortweg C14N genaamd) is een methode om ervoor te zorgen dat XML altijd in dezelfde vorm aanwezig is. Met XML is het mogelijk dat software "onderweg" de vorm aanpast. Zo is in XML bijvoorbeeld per definitie `<tag/>` equivalent aan `<tag></tag>`. Voor een signature is dit echter niet hetzelfde. Met C14N wordt XML in een canonieke vorm gebracht. Daarna kan dan op eenduidige wijze de signature berekend worden. De ontvanger kan dan ook vóór het verifiëren van de ondertekening de canonieke vorm genereren, zodat het juiste – canonieke – document wordt vergeleken met de signature.

Er zijn twee vormen van canoniek maken: Inclusive en Exclusive. De tweede vorm is de meest gebruikte voor Web Services. Het verschil tussen Inclusive en Exclusive Canonicalization zit in de wijze waarop namespace declaraties worden meegenomen. Bij Inclusive Canonicalization worden namespace declaraties uit een inkapselend document gewoon meegenomen: dat betekent dat de signature verandert wanneer een document ingekapseld wordt. Dat maakt Inclusive Canonicalization ongeschikt voor ingekapselde documenten, zoals een HL7v3 bericht in een SOAP Envelope. Daarmee is Exclusive Canonicalization de beste optie voor AORTA.

Voorbeeld van een bericht vóór het canoniek maken:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <MFMT_IN002101 xmlns="urn:hl7-org:v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:hl7-org:v3 ../schemas/MFMT_IN002101.xsd">
      <id extension="34234" root="2.16.528.1.1007.3.2.700222.1"/>
      <creationTime value="20040417161000"/>
      <versionCode code="NictizEd2005-Okt"/>
      <interactionId extension="MFMT_IN002101" root="2.16.840.1.113883.1.6"/>
      <profileId root="2.16.840.1.113883.2.4.3.11.1" extension="608"/>
      <processingCode code="P"/>
      <processingModeCode code="T"/>
      <acceptAckCode code="AL"/>
    </MFMT_IN002101>
  </soap:Body>
</soap:Envelope>
```

Hetzelfde bericht canoniek, exclusive without comments:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <MFMT_IN002101 xmlns="urn:hl7-org:v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:hl7-
org:v3 ../schemas/MFMT_IN002101.xsd">
```

```

        <id extension="34234" root="2.16.528.1.1007.3.2.700222.1"></id>
        <creationTime value="20040417161000"></creationTime>
        <versionCode code="NictizEd2005-Okt"></versionCode>
        <interactionId extension="MFMT_IN002101"
root="2.16.840.1.113883.1.6"></interactionId>
        <profileId extension="608"
root="2.16.840.1.113883.2.4.3.11.1"></profileId>
        <processingCode code="P"></processingCode>
        <processingModeCode code="T"></processingModeCode>
        <acceptAckCode code="AL"></acceptAckCode>

```

Te zien valt hoe de XML declaratie is verwijderd, attributen alfabetisch gesorteerd zijn (b.v. op profileId) (let wel: namespace declaraties komen voor de andere attributen) en lege elementen een eind-tag krijgen. (Het origineel kent geen afgebroken geregeleindes voor lange XML elementen, zoals hierboven door Word wel is gedaan.)

Whitespace – spaties, tabs, geregeleindes – zijn significant in XML. XML is immers begonnen als opmaaktaal voor documenten, en daar is whitespace tussen tags relevant. Canoniek maken laat de whitespace dus ongemoeid. Newlines moeten echter als een linefeed worden opgenomen. Een ontvanger mag voor het verifiëren van de signature de whitespace dan ook niet veranderen.

5.3 WS-Security

WS-Security is een standaard voor het uitwisselen van beveiligingsgegevens in SOAP Headers. De technologie is een onderdeel van de Web Services stack, zoals SOAP en WSDL ook zijn. Er zijn meerdere implementaties van, onder andere in Microsoft .NET en Java.

De WS-Security standaard voorziet in het verzenden van digitale security tokens in een XML bericht in een SOAP Header. Er kunnen drie typen tokens (in WSS is een "token" een XML structuur met beveiligingsgegevens) aangehecht worden (het woord "token" wordt in WSS dus in een iets andere betekenis gebruikt dan 'authenticatietoken' in het verdere document):

- tekst, zoals een username
- binaire tokens, b.v. een signature
- XML gebaseerde tokens.

De tweede variant is de meest relevante, deze hebben we nodig voor X.509 certificaten en encryptiesleutels. Binaire gegevens moeten worden gecodeerd omdat ze niet zonder meer in XML gezet kunnen worden. Er kunnen diverse coderingen gebruikt worden. Base 64 is default. De nadelen van Base 64 (toename in lengte, extra rekentijd) zijn bij kortere tokens niet relevant. Er zijn meerdere profielen voor specifieke tokensoorten, waarvan de meest relevante X.509 is.

Er zijn twee versies van WS-Security, 1.0 verschenen in 2004 en 1.1, verschenen in 2006.

[Ontwerpkeuze] Omdat er onvoldoende brede ondersteuning bestaat voor WS-Security 1.1 is gekozen voor gebruik van WS-Security 1.0.

5.4 WS-Security X.509 Profile

Dit Profile beschrijft het gebruik van X.509 certificate tokens met WS-Security. Dit profile definieert enkele opties voor het invoegen van een X.509 certificaat en gerelateerde gegevens. Hiervan zijn voor ons relevant:

- X.509 certificaat in binaire vorm.
- Verwijzing naar X.509 certificaat aan de hand van de naam van de uitgevende CA en het serienummer van het certificaat binnen die CA.

[Ontwerpkeuze] Er is in aansluiting op de voorgaande paragraaf gekozen voor WS-Security X.509 Profile 1.0.

5.5 Basic Security Profile

[BSP] beschrijft verdere aanvullingen en inperkingen op gebruik van XML Signature, Canonicalization, WS-Security en de bijbehorende Profiles. Net als het Basic Profile voor SOAP/WSDL, wat Nictiz volgt, is het een set richtlijnen gebaseerd op praktijkervaringen met (in)interoperabiliteit.

6. Ontwerpoverwegingen tokenauthenticatie

Dit hoofdstuk is niet normatief.

6.1 X.509 Strong One-way Authentication

De opzet van het token is gebaseerd op X.509 Strong Authentication, zoals beschreven in [X509] paragraaf 18.2.2.1 One-way authentication.

Deze vorm van authenticatie wordt daar beschreven als:

$$C^A\{t^A, r^A, ID^B, \text{sgnData}\}$$

Deel	Verklaring
A	De partij die zich wil authenticeren bij B.
B	De partij die authenticatie van A wenst.
C^A	Het X.509 certificaat dat A gebruikt om zich te authenticeren.
$\{\}$	Met het certificaat ondertekende gegevens.
t^A	Een timestamp (eventueel bestaande uit twee tijdstippen, aanmaak en expiratie).
r^A	Een door A gegenereerd random nummer, de nonce.
ID^B	Een adres of identiteitsteken van B.
sgnData	Eventuele overige gegevens die meegetekend worden.

Omdat A de gegevens met diens private sleutel ondertekent, kan B, na ophalen van het certificaat van A, vaststellen dat de gegevens inderdaad van A afkomstig zijn. Wanneer B de nonce r^A bewaart tot de expiratietijd, kan B ook vaststellen dat er geen sprake is van een "replay attack".

6.2 Uitbreiding op het token

De vraag is ook wat er precies geauthenticeerd wordt. Het UZI-register definieert authenticatie als "Een proces waarbij iemands identiteit bevestigd kan worden of waarmee de integriteit en de herkomst van de aangeboden gegevens gecontroleerd kunnen worden." Bij het verzenden van een HL7v3 bericht is overduidelijk geen sprake van de bevestiging van iemands persoonlijke identiteit (dat is het geval als een pasdrager zich persoonlijk met zijn pas komt identificeren), maar van het vaststellen van de herkomst van gegevens. Uiteindelijk zijn die gegevens het hele HL7v3 bericht, met de SOAP Envelope en Headers. Bij authenticatie gaat het echter niet om het zetten van een digitale handtekening onder het hele bericht, maar om het aspect van controle van herkomst.

Bij het opzetten van een SSL/TLS verbinding met de UZI-pas, wordt met de UZI-pas een beveiligde tunnel gecreëerd tussen het werkstation en het LSP. Daarbij wordt dan redelijkerwijs aangenomen dat de gegevens die vervolgens over die tunnel binnenkomen, ook daadwerkelijk van de persoon afkomstig zijn die die UZI-pas bezit.

Bij tokenauthenticatie is de relatie tussen UZI-pas en gegevens losser: het is immers mogelijk een token te tekenen, en pas veel later te verzenden. Het gevaar bestaat dat door manipulatie het token wordt gebruikt met andere gegevens. Daarom is gezocht naar een nauwere relatie tussen het token en het verzonden bericht, zodat tokenauthenticatie een voldoende vertrouwen geeft dat het bericht dat ontvangen wordt inderdaad

afkomstig is van de pasdrager. Deze relatie tussen bericht en token is gelegd door de volgende gegevens in het token op te nemen:

- een aanduiding van het berichttype;
- het burgerservicenummer van de patiënt (voor patiëntgerelateerde berichten).

Met deze extra gegevens is het voor een onderschepper alleen mogelijk eenzelfde bericht voor dezelfde patiënt op het beoogde tijdstip naar dezelfde ontvanger te zenden als het oorspronkelijke bericht. Dat is een voldoende borging voor de authenticatie. Wanneer grotere zekerheid nodig is dat de gegevens niet gewijzigd worden, zoals bij een elektronisch medicatievoorschrift, dient een andere techniek ingezet te worden.

6.3 Canonicalisatie

XML Canonicalization (Exclusive Canonicalization in het geval van tokenauthenticatie) is een van de bekende performancebottlenecks bij het gebruik van digitale handtekeningen bij XML. Het is echter een noodzakelijk kwaad.

Maatregelen die getroffen zijn om de performance impact te beperken, zijn:

- beperkt gebruik van namespaces
- geen gebruik van lege elementen
- een eenvoudige structuur voor het token en de handtekeningstructuur
- beperkt gebruik van attributen, waar mogelijk elementen in plaats van attributen.

7. Het authenticatietoken

7.1 Inhoud van het token

In deze paragraaf wordt de inhoud van het authenticatietoken besproken. Dit staat op zich los van de inhoud en plaats van de (eventuele) digitale handtekening. De vraag die hier speelt is: "van welke gegevens moet ik de aanmaker ondubbelzinnig vast kunnen stellen", niet de vraag "hoe stel ik de aanmaker ondubbelzinning vast". De term authenticatietoken wordt gebruikt voor de verzameling gegevens die de inzender van een bericht ondertekent. (Let wel: het woord "token" kan ook breder gebruikt worden – b.v. voor certificaten of signatures.)

Digitale handtekeningen verschillen fundamenteel van papieren handtekeningen in die zin, dat ze niet door het gebruikte medium aan elkaar zijn verbonden. Een traditionele handtekening staat op hetzelfde papier als de ondertekende gegevens. Digitale handtekeningen zijn logisch gekoppeld aan de ondertekende gegevens, waarbij de gebruikte cryptografie de koppeling beveiligd. Je kan iemand een e-mail sturen, en die ondertekenen, en de digitale handtekening op een usb-stick met de post verzenden. Het hele "bij elkaar horen" van papier en handtekening is hier niet geldig - hoe de handtekening ook getransporteerd wordt, de geldigheid en relatie tot het getekende is altijd vast te stellen. XML digital signatures ondersteunen zowel in één structuur gekoppelde handtekeningen als "losse" handtekeningen. Het "bij elkaar horen" van handtekening en authenticatietoken is dus anders dan bij papieren handtekeningen.

Het authenticatietoken voor het landelijk EPD ziet er als volgt uit:

```
<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>20050128173600</notBefore>
    <notAfter>20050128174059</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 te tekenen onderdelen van X.509 zijn dus als volgt geserialiseerd:

Deel	XML
t ^A	<notBefore>20050128173600</notBefore> <notAfter>20050128174059</notAfter>

r ^A	<pre><messageId> <root>2.16.528.1.1007.3.3.1234567.1</root> <extension>0123456789</extension> </messageId></pre>
ID ^B	<pre><addressedParty> <root>2.16.840.1.113883.2.4.6.6</root> <extension>1</extension> </addressedParty></pre>
sgnData	<pre><coSignedData>...</coSignedData></pre>

Noot:

De voorbeelden in deze handleiding bevatten geregeleiden en inspringingen. Die zijn toegevoegd ten behoeve van de leesbaarheid. De voorbeeldberichten waarvan ze afgeleid zijn bevatten echter een lange signedData string zonder whitespace of geregeleiden tussen de elementen.

Noot:

in alle 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 niet meer te veranderen, dit beïnvloedt de handtekening.

```
<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">
```

Hier begint het authenticatietoken. Het signedData element is geen HL7v3 element, maar gedefinieerd binnen deze specificatie. De AORTA namespace is verplicht, de waarde (hier: <http://www.aortarelease.nl/805/>) wordt pas definitief vastgesteld bij de release.

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 anderszins) de waarde uniek blijft. De aanbevolen methode is:

- begin met de string "token" en een underscore "_"
- voeg hier de root van het HL7v3 message.id aan toe
- voeg hier een underscore "_" aan toe
- voeg hier de extension van het HL7v3 message.id aan toe
- bijvoorbeeld: token_2.16.528.1.1007.3.3.1234567.1_0123456789

Deze methode garandeert dat iedere waarde uniek is, ook wanneer meerdere HL7v3 berichten in een XML bestand worden samengevoegd. (De methode kan niet verplicht worden, omdat de extension van een message.id niet numeriek hoeft te zijn, en het dus mogelijk is dat deze karakters bevat die in een XML Id niet toegestaan zijn. In dat geval dient een leverancier er zelf voor te zorgen een waarde te genereren die uniek is voor een AORTA bericht. Hierbij dient dan een UUID gebruikt te worden. Bij het gebruik van andere vormen is er een kans, hoe klein ook, dat een Id samenvalt met een Id gemaakt volgens een andere methode van een andere leverancier.) Bij gebruik van globaal unieke waarden is geen logica nodig bij samenvoegen van HL7v3 berichten om de uniekheid van id's te borgen. (Noot: samenvoegen van HL7v3 berichten in een enkel XML bestand is op

dit moment niet aan de orde voor berichten waar deze vorm van authenticatie gebruikt wordt. Het uniek maken van het Id is dan ook enkel een maatregel om dit eventueel later mogelijk te maken.) Een extra voordeel van globaal unieke Id's is dat het altijd voor iedere handtekening mogelijk is vast te stellen welk token er bedoeld wordt. Het Id moet uniek zijn voor AORTA berichten. (Dat geldt niet voor duplicaten op transportniveau, waarbij een ontvanger duplicaatdetectie doet en dubbele berichten verwijdert. In dergelijke duplicaten mag (en moet) dezelfde Id voorkomen.) Op hoger niveau (HL7v3-berichten) zijn dergelijke duplicaten echter een en hetzelfde bericht.

Voor digitale handtekeningen is het essentieel dat het bericht niet wijzigt. Veel wijzigingen worden ondervangen door XML Canonicalization. Whitespace valt daar echter niet onder. Dat is geen probleem: een XML parser zal nooit zomaar whitespace tussen elementen verwijderen (XML is immers opgezet als document-formaat). XML editors doen dit soms wel: een "pretty-printed" ondertekend bericht zal niet meer valideren. Daarom is het aan te bevelen de hele handtekening niet "pretty-printed" op te nemen in het bericht.

```
<authenticationData>
```

Het begin van de gegevens voor de alternatieve authenticatie.

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

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

EIS: Wordt binnen de expiratietijd een nonce een tweede maal ontvangen, dan **moet** dat bericht geweigerd worden.

Een uitzondering is duplicaten ten behoeve van betrouwbaar transport: een eis aan dit type duplicaten is dat er slechts een enkele van verwerkt wordt. Duplicaatdetectie en –verwijdering zal dan ook plaats moeten vinden voordat de uniciteit van de nonce gecontroleerd wordt om aan bovenstaande regel te kunnen voldoen. Wanneer hetzelfde bericht nogmaals verzonden wordt, maar niet als duplicaat, dan dient een nieuwe nonce gebruikt te worden. Bijvoorbeeld wanneer een query niet binnen de time-out beantwoord is, en de query wordt opnieuw gesteld, dan moet een nieuwe nonce gegenereerd worden. (Het is mogelijk bij een onbeantwoorde query hetzelfde token nogmaals in te zenden. Wanneer de communicatie mislukt is doordat de query nooit bij het LSP is aangekomen, dan zal het (duplicaat) token probleemloos verwerkt worden. Is de oorspronkelijke query echter wel bij het LSP aangekomen en het probleem later opgetreden, dan zal het token geweigerd worden. Wanneer het niet duidelijk is waar de communicatie mislukt is, is dit herhalen dus wel toegestaan mits het verzendende systeem rekening houdt met een foutmelding van het LSP.)

```
<notBefore>20050128173600</notBefore>
```

De tijd waarop het authenticatieblok 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>20050128174059</notAfter>
```

De tijd waarop het authenticatieblok vervalst. 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.

Eis: 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.

Tokenauthenticatie maakt het mogelijk een token aan te maken, te tekenen en klaar te zetten voor latere verzending. Wanneer het token bij de ontvanger aankomt tussen *notBefore* en *notAfter*, kan deze niet zien wanneer het token daadwerkelijk is aangemaakt. Dit pre-signen is dus technisch goed mogelijk, maar het introduceert ook een nieuw risico: klaarstaande getekende tokens kunnen gestolen en misbruikt worden. Omdat het token door opnemen van BSN en *triggerEvent* (zie hieronder) een sterke relatie heeft met het bericht is dit risico zeer beperkt. Bovendien is door controle op de relatie tussen gebruikte UZI-pas en het servercertificaat van het GBZ misbruik alleen mogelijk als de inbreker opnieuw toegang weet te krijgen tot een systeem waarop de eigenaar van de UZI-pas mag werken op het moment dat het token geldig is. Wel geldt dat er geen mechanisme is om eenmaal ondertekende tokens ongeldig te verklaren (anders dan het laten verlopen zonder te versturen). Met andere woorden, als een ondertekend token gestolen wordt, kan niemand voorkomen dat de dief het gebruikt. Het wordt daarom aanbevolen spaarzaam om te gaan met de mogelijkheid van pre-signen.

Wanneer een zorgverlener een query klaarzet voor een later moment, bijvoorbeeld voor een later consult, kan het gebeuren dat de communicatie mislukt binnen de gestelde periode. In dat geval kan een automatisch proces de query niet opnieuw stellen na het verlopen van *notAfter*: er is immers geen pas om een nieuw token mee aan te maken. Hoewel dit soort gevallen zeldzaam zullen zijn bij een goed functionerend XIS van de zorgverlener, zal in deze gevallen de zorgverlener de gegevens alsnog bij aanvang van het consult op moeten halen met een nieuw token.

Het is mogelijk tokens aan te maken voor later gebruik. Een token wordt bijvoorbeeld aangemaakt op 28 april 2008, met als waarden "20080528180000" in notBefore en "20080528181500" in notAfter. Het is dan geldig een maand na aanmaken, gedurende een kwartier. Het aanmaken van dergelijke tokens is geen probleem. Het signen van een dergelijk token kan gebeuren vlak voor het token verzonden wordt. Het signen kan echter ook eerder gebeuren, bijvoorbeeld op het moment van aanmaken. Na dergelijk pre-signen is het token klaar voor gebruik. Een inbreker kan het bijvoorbeeld ook gebruiken om (op het juiste moment) gegevens op te vragen. Daarom is het zaak pre-signen beperkt te houden. De volgende aanbevelingen gelden:

- teken de tokens zo kort mogelijk voor het daadwerkelijk gebruik, gebruik dus geen pre-signing als er functioneel geen aanleiding toe is,
- beveilig reeds getekende tokens goed.

```
<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).

```
</authenticationData>
```

Einde van de gegevens voor de authenticatie conform de X.509 specificatie voor éézijdige authenticatie.

```
<coSignedData>
```

Met de authenticatie meegetekende gegevens uit het bericht. Dit zijn kopieën van gegevens die ook in het bericht voorkomen. Deze zijn met name 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>
```

De TriggerEventId wordt altijd meegetekend. Deze identificeert het *berichttype* uniek, en bepaalt dus de intentie van de verzender. Deze meesturen verhindert veel soorten aanvallen, bijvoorbeeld 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.

```
<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".

Let op: in HL7v3 XML berichten komt BSN soms voor in verschillende elementtypen. De volgende vormen komen allen voor in berichten:

elementnaam/elementnaam@attribuutnaam
patient/id@extension
Patient/id@extension
patientId/value@extension
patientID/value@extension
subjectID/value@extension
person.id/value@extension
IdentifiedPerson/id@extension

Voor de verzender hoeft dit niet zo'n probleem te zijn: deze stopt het BSN in het token en is klaar. De ontvanger moet echter het BSN in het token verifiëren tegen het token in het bericht, en moet dus zorgen dat er per berichttype het BSN wordt opgehaald uit het juiste element. Een BSN is altijd in de vorm van een oid (Object Identifier), en dat leidt in HL7v3 XML altijd tot een element met twee attributen, een root met de waarde 2.16.840.1.113883.2.4.6.3 en een extension met het BSN er in.

Het BSN moet altijd opgenomen worden voor berichten die betrekking hebben op een persoon, ook als het schema toestaat het BSN weg te laten. Dat leidt tot de volgende regel: BSN wordt opgenomen in het token voor alle berichttypen die betrekking hebben op één en niet meer dan één persoon, ook als het schema het opnemen van het BSN optioneel maakt of het schema het BSN helemaal niet toestaat. Bijvoorbeeld bij het afmelden van patiëntgegevens bij de verwijzindex is het BSN optioneel, niet verplicht. Voor berichttypen waar nooit een BSN in zit, wordt het BSN niet opgenomen in het token. Een ander voorbeeld: bij beheersoverdrachtberichten bevat het schema geen burgerservicenummer. In dit geval kan de ontvangende partij met het burgerservicenummer uit het token controleren of het patiëntstuk waarvan het beheer overgedragen wordt, inderdaad hoort bij de patiënt waarvan het burgerservicenummer in het token zit. Uiteraard is in de enkele gevallen waar het BSN niet bekend is (bijvoorbeeld berichten voor opvragen BSN) het BSN niet verplicht in het token.

Eis: Voor berichten die betrekking hebben op op één en niet meer dan één persoon waarvan het BSN bekend is, **moet** het BSN in het token zitten.

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

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

Tenslotte wordt het token afgesloten.

```
</coSignedData>  
</signedData>
```

7.2 Verificatie met het bericht

Het is belangrijk vast te stellen dat de velden in het authenticatietoken overeenstemmen met die in het bericht. Wanneer dit niet zou gebeuren, kan een kwaadwillende met een gestolen token nog steeds gegevens opvragen van b.v. ieder willekeurig burgerservicenummer.

Eis: Bij gebruik van een authenticatietoken moet de ontvanger de onderstaande controles uitvoeren en een bericht dat hier niet aan voldoet, weigeren.

Eis: De ontvangende partij **moet** verifiëren of het BSN in het token past bij het verzonden bericht; ofwel door een match met het BSN in het token; ofwel doordat de gegevens in het bericht daadwerkelijk betrekking hebben op de persoon wiens BSN in het token zit.

Voor die berichten waar het token wel een BSN bevat, en het bericht niet, kan de verificatie niet beschreven worden met XPath; de ontvanger zal deze op andere wijze moeten implementeren. De volgende tabel geeft de XPath expressies waar een bestand met een authenticatietoken aan moet voldoen, met daaropvolgend de tekstuele foutmelding. Het is tevens als Schematron beschikbaar. De context node is het /soap:Envelope[1]/soap:Header[1]/ao:authenticationTokens[1]/ao:signedData[1] element. De daadwerkelijke controle hoeft uiteraard niet met XPath of Schematron gedaan te worden, zolang de controles equivalent zijn aan de onderstaande.

```
"ao:authenticationData/ao:messageId/ao:root = //hl7:interactionId/../../hl7:id/@root"  
Root in messageId is niet gelijk aan message.id root  
"ao:authenticationData/ao:messageId/ao:extension =  
//hl7:interactionId/../../hl7:id/@extension"  
Extension in messageId is niet gelijk aan message.id root  
"ao:authenticationData/ao:notBefore &lt; ao:authenticationData/ao:notAfter "  
Datum notBefore moet voor notAfter liggen  
"ao:authenticationData/ao:addressedParty/ao:root = '2.16.840.1.113883.2.4.6.6'"  
Root in addressedParty is niet gelijk aan 2.16.840.1.113883.2.4.6.6  
"ao:authenticationData/ao:addressedParty/ao:extension = '1'"  
Extension in addressedParty is niet gelijk aan 1  
"(ao:coSignedData/ao:triggerEventId = 'COMT_TE800100' and //hl7:COMT_IN800100) or  
(ao:coSignedData/ao:triggerEventId = 'COMT_TE800110' and //hl7:COMT_IN800110) or  
(ao:coSignedData/ao:triggerEventId = 'COMT_TE800120' and //hl7:COMT_IN800120) or  
(ao:coSignedData/ao:triggerEventId = 'COMT_TE800200' and //hl7:COMT_IN800200) or  
(ao:coSignedData/ao:triggerEventId = 'COMT_TE800300' and //hl7:COMT_IN800300) or  
(ao:coSignedData/ao:triggerEventId = 'COMT_TE800300' and //hl7:COMT_IN800400) or  
(ao:coSignedData/ao:triggerEventId = 'MFMT_TE002101' and //hl7:MFMT_IN002101) or  
(ao:coSignedData/ao:triggerEventId = 'MFMT_TE002102' and //hl7:MFMT_IN002102) or  
(ao:coSignedData/ao:triggerEventId = 'MFMT_TE002103' and //hl7:MFMT_IN002103) or  
(ao:coSignedData/ao:triggerEventId = 'PORX_TE990011NL' and //hl7:PORX_IN924000NL)  
or  
(ao:coSignedData/ao:triggerEventId = 'PORX_TE990001NL' and //hl7:PORX_IN932000NL)  
or  
(ao:coSignedData/ao:triggerEventId = 'QUMT_TE020010' and //hl7:QUMT_IN020010) or  
(ao:coSignedData/ao:triggerEventId = 'QUMT_TE020010' and //hl7:QUMT_IN020011NL) or  
(ao:coSignedData/ao:triggerEventId = 'QUPC_TE990001NL' and //hl7:QUPC_IN990001NL)  
or  
(ao:coSignedData/ao:triggerEventId = 'QURX_TE990001NL' and //hl7:QURX_IN990001NL)  
or  
(ao:coSignedData/ao:triggerEventId = 'QURX_TE990011NL' and //hl7:QURX_IN990011NL)  
or  
(ao:coSignedData/ao:triggerEventId = 'REPC_TE000006NL' and //hl7:REPC_IN000015NL)  
or
```

```

(ao:coSignedData/ao:triggerEventId = 'REPC_TE000012NL' and //hl7:REPC_IN000023NL)
or
(ao:coSignedData/ao:triggerEventId = 'REPC_TE990003NL' and //hl7:REPC_IN990003NL)
or
(ao:coSignedData/ao:triggerEventId = 'PRPM_TE908100NL' and //hl7:PRPM_IN908100NL)
or
(ao:coSignedData/ao:triggerEventId = 'PRPM_TE908200NL' and //hl7:PRPM_IN908200NL)
">
Trigger Event Id stemt niet overeen met interactie
<!-- In HL7 staat BSN in element met root '2.16.840.1.113883.2.4.6.3' -->
"((ao:coSignedData/ao:patientId/ao:extension =
//node() [@root='2.16.840.1.113883.2.4.6.3']/@extension)
or
not(//node() [@root='2.16.840.1.113883.2.4.6.3']/@extension))"
BSN in token stemt niet overeen met BSN in payload
" test="not(ao:coSignedData/ao:patientId/ao:extension !=
//node() [@root='2.16.840.1.113883.2.4.6.3']/@extension)"
Onjuiste root patient id in token

```

Noot: de tabel met TriggerEventId's is niet noodzakelijk up to date na verschijnen van dit document (zeker niet bij tussentijdse deelreleases van zorgtoepassingen). Per ondersteunde zorgtoepassing zal gekeken moeten worden welke TriggerEventId's nodig zijn. Trigger events van interacties met vertrouwensniveau "laag" zijn niet opgenomen, hiervoor is geen tokenauthenticatie nodig.

8. Digitale handtekening over het authenticatietoken

8.1 Inhoud van de handtekening

Over een (deel van een) XML document kan een XML Signature geplaatst worden. Na plaatsen van de XML Signature kan de ontvanger, met gebruikmaking van het persoonsgebonden UZI certificaat van de verzender en de CA certificaten zoals verstrekt door het UZI-register, onomstotelijk vaststellen dat het getekende (deel van een) XML document ondertekend is met de private sleutel behorend bij het gebruikte certificaat.

De XML Signature ziet er als volgt uit (lange Base 64 strings zijn afgekort):

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#" />
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="#token_2.16.528.1.1007.3.3.1234567.1_0123456789">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>BX/zd6w2FH4gqtVNNk7jJ9mAPwI=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>XbgER...QOZQA=</SignatureValue>
  <KeyInfo>
    ...
  </KeyInfo>
</Signature>
```

Eerst volgt een bespreking punt voor punt. Als een XML Signature library wordt gebruikt, zal de XML Signature grotendeels worden gegenereerd door die library. Voor de gebruikers van een dergelijke library, is dit schrijven slechts informatief. De programmeur zal de signature dan niet zelf hoeven opbouwen. Wanneer er geen XML Signature library gebruikt wordt, kan de XML Signature als tekst strings opgebouwd worden. De bespreking hoe dat kan volgt later.

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
```

De XML Signature begint met een Signature element en een verwijzing naar de XML Signature namespace.

```
<SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
```

Vervolgens komt het SignedInfo blok. Dit blok bevat de gegevens die daadwerkelijk getekend worden.

```
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
```

Het SignedInfo blok zelf moet in de juiste canonieke vorm gebracht worden. Voor de digitale handtekening in AORTA dient hier altijd Exclusive XML Canonicalization gebruikt te worden [EXCC14N]. Deze vorm van canoniek maken gaat goed om met namespaces van XML documenten die ingebed zijn in andere documenten. Aangezien het token is ingebed in een HL7v3 document, dat weer is ingebed in een SOAP envelop, is het gebruik van Exclusive XML Canonicalization verplicht.

```
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
```

De methode van tekenen. Voor de digitale handtekening in AORTA wordt altijd gebruik gemaakt van een RSA handtekening over een SHA-1 digest. (Zie hieronder voor details.) Andere waarden zijn niet toegestaan. Migratie naar SHA-2 is een aandachtspunt voor de toekomst.

```
<Reference URI="#token_2.16.528.1.1007.3.3.1234567.1_0123456789">
```

Een referentie naar het stuk XML dat getekend is. In dit geval betreft het getekende deel het signedData element dat het token omsluit. Dit signedData element is van een Id voorzien. De waarde van dit Id wordt hier overgenomen in een relatieve URI (dus voorzien van een "#" als prefix). De relatieve URI betekent dat de Id verwijst naar een deel XML in hetzelfde document als het Reference element (en de XML Signature). Dat is wenselijk, omdat bij absolute URI's de locatie van de XML (op een specifieke server op internet, of op een specifiek pad) vastligt. Het betekent wel dat bij "uitpakken" van token en handtekening de relatie tussen beiden vastgelegd zal moeten worden. Dus als de XML Signature en het HL7v3 bestand met token apart opgeslagen worden, moet ergens vastgelegd worden bij welk HL7v3 bestand de XML Signature hoort.

```
<Transforms>
  <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
</Transforms>
```

Over het token (het signedData blok) wordt ook een canonicalisatieslag gedaan, ook weer Exclusive XML Canonicalization.

```
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

Van het canonieke signedData blok wordt een SHA-1 digest berekend. Alleen SHA-1 is toegestaan. De digest – en canonicalisatie – is over het signedData blok inclusief de signedData tags.

```
<DigestValue>g42...GAw</DigestValue>
```

De digest over het signedData blok wordt opgeslagen, gecodeerd met Base 64. Base 64 is een methode om hexadecimale gegevens, zoals een digest of signature, om te zetten in een teken dat bestaat uit een beperkte set leesbare tekens. Dat is essentieel voor het opnemen in XML, omdat niet ieder willekeurig octet zomaar in XML voor mag komen. Na Base 64 codering is dit geen probleem. Base 64 is een eenduidige code, eenvoudig terug om te zetten in de oorspronkelijke hexadecimale vorm.

```
</Reference>
</SignedInfo>
```

Einde van de Reference en het SignedInfo blok.

```
<SignatureValue>a04IUxC...9yvunA</SignatureValue>
```

Over het SignedInfo blok wordt een SHA-1 digest berekend waarover een RSA handtekening gezet wordt. De waarde van de RSA handtekening wordt met Base 64 gecodeerd en opgenomen in de XML Signature. De handtekening is dus niet over het token zelf, maar over een SHA-1 digest van een SignedInfo blok, waarin een SHA-1 digest van het token.

```
<KeyInfo>  
  ...  
</KeyInfo>
```

Er worden gegevens van het gebruikte certificaat opgenomen. Dit wordt in de volgende paragraaf nader toegelicht.

```
</Signature>
```

Einde van de XML Signature.

9. Certificaten

9.1.1 Te gebruiken certificaat

De UZI-pas kent een aantal modellen:

Naam UZI-pastype	Codering Pastype
Individuele zorgverlenerpas ⁵	I
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, type I, Z of N. Hoewel het pastype gecodeerd is opgenomen in het certificaat (in het subjectAltName attribuut), dient een applicatie op basis van de uitgevende CA vast te stellen of het om een vertrouwde UZI-pas gaat.

Op deze pastypen staan drie verschillende certificaten:

Naam	Key Usage omschrijving	Key usage hexadecimaal
authenticiteitcertificaat	digitalSignature	0x80
handtekeningcertificaat	NonRepudiation	0x40
vertrouwelijkheids-certificaat	keyEncipherment, dataEncipherment, keyAgreement	0x38 (OR'ed 0x20, 0x10, 0x08)

Voor het tekenen van het authenticatietoken mag alleen het authenticiteitcertificaat gebruikt worden. Dit certificaat bevat een 1024 bits RSA publieke sleutel. De bijbehorende private sleutel die op de UZI-pas staat, wordt gebruikt om de handtekening te genereren.

Om de handtekening te verifiëren, moet de ontvanger over de bijbehorende publieke sleutel beschikken. Dit kan op twee manieren:

- 1) door het certificaat met de publieke sleutel mee te zenden;
- 2) door een verwijzing naar het certificaat mee te zenden en deze met het LDAP protocol op te halen in de directory van het UZI-register.

Deze opties zijn in de volgende paragrafen uitgewerkt. Voor tokenauthenticatie is gekozen voor de tweede variant, het meezenden van een verwijzing. De paragraaf over het meezenden van certificaten is behouden voor toekomstig gebruik.

Zie voor de verdere beschrijving van de passen [UZIPAS].

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.

⁵ Het UZI-register heeft in het nieuwe CPS van november 2007 aangegeven dat de I pas zal worden uitgefaseerd. Hij wordt hier nog genoemd als historische referentie.

9.1.2 Certificaat meezenden

```
<wss:Security
  xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
  secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
  utility-1.0.xsd"
  soap:mustUnderstand="1">
  <wss:BinarySecurityToken
    wsu:Id="signing-cert"
    ValueType=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
    token-profile-1.0#X509v3
    EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
    message-security-1.0#Base64Binary">MIIFd...PZaIvdgXOQ==
  </wss:BinarySecurityToken>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo ...>...</SignedInfo>
    <SignatureValue>...</SignatureValue>
    <KeyInfo>
      <wss:SecurityTokenReference>
        <wss:Reference URI="#signing-cert"
          ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
          profile-1.0#X509v3" />
        </wss:SecurityTokenReference>
      </KeyInfo>
    </Signature>
  </wss:Security>
```

Een bespreking per punt:

```
<wss:Security
  xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
  secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
  utility-1.0.xsd"
  soap:mustUnderstand="1">
```

Bovenstaand is de WSS 1.0 Security header.

```
<wss:BinarySecurityToken
  wsu:Id="signing-cert"
  ValueType=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
  token-profile-1.0#X509v3
  EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
  message-security-1.0#Base64Binary">MIIFd...PZaIvdgXOQ==
</wss:BinarySecurityToken>
```

Bij het opnemen van het certificaat wordt een BinarySecurityToken element opgenomen met het hele certificaat in Base 64 codering. De wsu:Id moet matchen met de referentie later, maar de invulling is verder vrij. Wel moet deze waarde uniek zijn binnen het gehele document, aangezien het een ID is. De waarden voor ValueType en EncodingType zijn vast en verplicht.

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo ...>...</SignedInfo>
  <SignatureValue>...</SignatureValue>
```

De Signature zoals in het vorige hoofdstuk.

```
<KeyInfo>
  <wss:SecurityTokenReference>
```

De KeyInfo, met een verwijzing naar het certificaat.

```
<wss:Reference URI="#signing-cert"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
profile-1.0#X509v3" />
```

De verwijzing in de URI moet kloppen met de wsu:Id in het BinarySecurityToken element. ValueType is vast en verplicht.

```
</wss:SecurityTokenReference>
</KeyInfo>
</Signature>
</wss:Security>
```

Afsluiting van de elementen.

9.1.3 Certificaatverwijzingen

Bij het opnemen van een verwijzing naar het certificaat, wordt de volgende constructie opgenomen.

```
<KeyInfo>
  <wss:SecurityTokenReference>
    <X509Data xmlns="http://www.w3.org/2000/09/xmldsig#">
      <X509IssuerSerial>
        <X509IssuerName>CN=TEST UZI-register Zorgverlener CA,O=agentschap
Centraal Informatiepunt Beroepen Gezondheidszorg,C=NL
        </X509IssuerName>
        <X509SerialNumber>35972415477696508790773831356241160195
        </X509SerialNumber>
      </X509IssuerSerial>
    </X509Data>
  </wss:SecurityTokenReference>
</KeyInfo>
```

Een bespreking per punt:

```
<KeyInfo>
<wss:SecurityTokenReference>
```

Het begin van de certificaatverwijzing. Het <KeyInfo> blok zit in de in het vorige hoofdstuk besproken <Signature>. <KeyInfo> valt in de eerder gedeclareerde namespace van XML Signatures, <SecurityTokenReference> in de WSS 1.0 namespace.

```
<X509Data xmlns="http://www.w3.org/2000/09/xmldsig#">
```

De certificaatverwijzing zelf zit weer in de namespace van XML Signatures.

```
<X509IssuerSerial>
```

Begin van de certificaatgegevens.

```
<X509IssuerName>CN=TEST UZI-register Zorgverlener CA, O=agentschap Centraal
Informatiepunt Beroepen Gezondheidszorg,C=NL
</X509IssuerName>
```

De DN (Distinguished Name) van de CA (Certificate Authority). De attributen die in een DN in een authenticatietoken zitten zijn CN (CommonName), O (OrganizationName) en C

(CountryName), in die volgorde. De attributen worden van de waarden gescheiden door een = teken, van elkaar gescheiden met komma's en opgenomen zonder spaties, met uitzondering van de spaties in de waarden zelf, zoals beschreven in [LDAPUTF8].

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

Zorgverlenerpas		
issuer.countryName	C	NL
issuer.organisationName	O	agentschap Centraal Informatiepunt Beroepen Gezondheidszorg
issuer.commonName	CN	UZI-register Zorgverlener CA

String:

CN=UZI-register Zorgverlener CA,O=agentschap Centraal Informatiepunt Beroepen Gezondheidszorg,C=NL

(Zonder regeleinde, met een enkele spatie waar het regeleinde staat.)

LET OP: de issuer.commonName verschilt per 'generatie' UZI-passen. Inmiddels is er een ook een 'UZI-register Zorgverlener CA G2' en 'UZI-register Medewerker op naam CA G2'. G1 en G2 (en in de toekomst G3) UZI-passen kunnen bij een zorgaanbieder naast elkaar gebruikt worden. Daarom dient de Issuer DN dynamisch afgeleid te worden uit het gebruikte authenticatiecertificaat.

Medewerkerpas op naam		
issuer.countryName	C	NL
issuer.organisationName	O	agentschap Centraal Informatiepunt Beroepen Gezondheidszorg
issuer.commonName	CN	UZI-register Medewerker op naam CA

String:

CN=UZI-register Medewerker op naam CA,O=agentschap Centraal Informatiepunt Beroepen Gezondheidszorg,C=NL

(Zonder regeleinde, met een enkele spatie waar het regeleinde staat.)

```
<X509SerialNumber>35972415477696508790773831356241160195</X509SerialNumber>
```

Het decimale serialNumber van de pas. Dit is gegarandeerd uniek voor een bepaalde CA. Het certificate.serialNumber is conform [X509CRL] opgenomen in de certificaten, hetgeen betekent dat het een positieve integer is.

```
</X509IssuerSerial>
</X509Data>
</wss:SecurityTokenReference>
</KeyInfo>
```

Afsluiting van de elementen.

WSS 1.0 biedt de mogelijkheid de certificaatverwijzing mee te tekenen:

```
<Signature xmlns="...">
```

```
<SignedInfo>...
  <Reference URI="#token_..."></Reference>
  <Reference URI="#keyinfo"></Reference>
</SignedInfo>
<SignatureValue>HFLP...</SignatureValue>
<KeyInfo Id="keyinfo">
  <wsse:SecurityTokenReference>
    <X509Data>
      <X509IssuerSerial>
        <X509IssuerName>...</X509IssuerName>
        <X509SerialNumber>...</X509SerialNumber>
      </X509IssuerSerial>
    </X509Data>
  </wsse:SecurityTokenReference>
</KeyInfo>
</Signature>
```

Dit wordt bij tokenauthenticatie **niet** gedaan. Er mag dus maar één Reference element in SignedInfo voorkomen, wat verwijst naar het authenticatietoken.

10. Het maken van de XML Signature

Wanneer de XML Signature vanuit tekst gemaakt wordt, kan de volgende methode gevolgd worden.

10.1.1 Maak een signedData blok

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

Het volgende is een voorbeeldtoken:

```
<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>20050128173600</notBefore>
<notAfter>20050128174059</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.

10.1.2 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 wordt 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.

10.1.3 Bereken de SHA-1 digest van het signedData blok.

Van de signedData moet een SHA-1 digest berekend worden. Deze digest – een verzameling octets – wordt gecodeerd met Base 64. In het voorbeeld (mits alle linefeeds worden weggehaald) is de waarde van de digest, gecodeerd met Base 64: "xsaInPxc9r8Vf0ZY6iIbj7eJ6Is=" (zonder de aanhalingstekens).

10.1.4 Maak het SignedInfo blok

XML Signature maakt gebruik van een SignedInfo blok voor de 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>xsaInPxc9r8Vf0ZY6iIbj7eJ6Is=</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. Hier gelden weer alle regels uit 7.8.2. Het bovenstaande blok is in canonieke vorm, wanneer tenminste alle regeleinden verwijderd worden. Buiten de richtlijnen van 7.8.2 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.

10.1.5 Selecteer het authenticatiecertificaat en de bijbehorende private key

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

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/veelgestelde vragen/certificaten/hoegebruikikvandedriecertificatendejuisteomteauthenticeren.asp>

10.1.6 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. In het voorbeeld, mits regeleinden verwijderd worden, levert dit de waarde:

```
<SignatureValue>bgERa3Utevm52UWmjapFUvS2ygvOUmTknXd0AvZqWIGDKqyRAh4L01aSt2WmJXia2+6
sSli99fHPRekZSU9iqmwhmTVu8QDv6NiThB5wEJJFdpseCJiKemqEvbrIDG2pxmnV3IcmYluahcBiTAN/wX
4AoMT4KtB9BxF5FQOZQA=</SignatureValue>
```

Waarbij alweer de regeleinden weggedacht moeten worden. (Base 64 staat wel whitespace zoals spaties en regeleinden toe, dus ook met regeleinden is de waarde geldig.)

Base64 moet gebruikt worden zoals gespecificeerd in [MIME].

10.1.7 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>
<X509Data xmlns="http://www.w3.org/2000/09/xmldsig#">
<X509IssuerSerial>
<X509IssuerName>CN=TEST UZI-register Zorgverlener CA,O=agentschap Centraal
Informatiepunt Beroepen Gezondheidszorg,C=NL</X509IssuerName>
<X509SerialNumber>35972415477696508790773831356241160195</X509SerialNumber>
</X509IssuerSerial>
</X509Data>
</wss:SecurityTokenReference>
</KeyInfo>
```

Uitgangspunt is dat de WSS namespace gedeclareerd is zoals onder opgenomen. Bij een zorgverlenerpas moet de eerste rode string vervangen worden door "UZI-register Zorgverlener CA" bij een medewerkerpas op naam door "UZI-register Medewerker op naam CA". De tweede rode string wordt vervangen door het serial number van de pas (decimaal).

10.1.8 Maak de headers

Tenslotte moet het hele 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:mustUnderstand="1">  
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">  
<SignedInfo ...>  
... Het boven gedefinieerde SignedInfo blok ...  
</SignedInfo>  
<SignatureValue>  
... De boven gedefinieerde Signature ...  
</SignatureValue>  
<KeyInfo>  
... De boven gedefinieerde certificaatverwijzing ...  
</KeyInfo>  
</Signature>  
</wss:Security>
```

Hier moet beslist niet meer met de strings (signedData, SignedInfo) gemanipuleerd worden, maar ze moeten octet-voor-octet overgenomen worden in het bericht. Heel 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 op.

Bij de Signature spelen geen canonicalisatie issues: iedere vorm van well-formed XML mag dus. Canonicalisatie speelt alleen bij die XML code waarover een SHA-1 digest wordt berekend of waarover RSA handtekeningen worden gezet, en dat gebeurt op dit blok niet meer. Het maken van de XML Signature uit strings levert dus twee blokken tekst op: de signedData en de Signature.

Ook de header met het authenticatietoken wordt aangemaakt:

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

Plaats deze beide SOAP Headers tenslotte in de SOAP Envelope zoals in het volgende hoofdstuk beschreven.

11. Berichtenverkeer met token en handtekening

11.1 Plaats van het token

Het authenticatietoken wordt opgenomen in een SOAP Header.

Eis: Voor authenticatiedoeleinden 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:mustUnderstand="1">
    <signedData xmlns="http://www.aortarelease.nl/805/"
      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 een <authenticationTokens> element in een 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.

Het declareren van de namespace <http://www.aortarelease.nl/805/> bij <signedData> is niet nodig: deze kan ook overgeërfd worden 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.

11.2 Plaats van de digitale handtekening

De handtekening wordt in een WS-Security 1.0 SOAP Header gezet.

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

De volgorde van beide headers (token en handtekening) is niet relevant: zowel handtekening vóór het token als token vóór de 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:mustUnderstand="1">
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        ...
      </SignedInfo>
    </Signature>
  </wss:Security>
  ...
</soap:Header>
```

```

    </SignedInfo>
    <SignatureValue>Wuwn...5e4=</SignatureValue>
    <KeyInfo>
      <X509Data>
        ...
      </X509Data>
    </KeyInfo>
  </Signature>
</wss:Security>
</soap:Header>

```

11.3 WSDL en Endpoints

De bestaande WSDL wordt gebruikt voor tokenauthenticatie. Er wordt niet gekozen voor aparte WSDL eindpunten voor tokenauthenticatie. De voornaamste reden is dat er meerdere SOAP headers denkbaar zijn waarvoor aparte eindpunten potentieel wel makkelijk zijn: bijvoorbeeld bij inzet van WS-ReliableMessaging. Dan kunnen combinaties optreden als:

- noch WSS, noch WSRM,
- WSS zonder WSRM
- geen WSS, wel WSRM
- zowel WSS als WSRM.

Uiteraard kan dit bij een derde of vierde modus snel oplopen tot zeer veel eindpunten.

Een dergelijke opzet met aparte eindpunten voor SOAP Headers is in Web Services ook niet gebruikelijk.

De WSDL is wel anders door het op kunnen treden van andere SOAP Faults (zie hieronder). De Faults zijn echter niet expliciet in de WSDL opgenomen, dus daar verandert ook niets aan.

11.4 Foutafhandeling

De volgende SOAP Faults worden onderkend en geretourneerd waar van toepassing. (Deze tabel is overgenomen uit WSS 1.0).

Omschrijving (faultstring)	Faultcode
An unsupported token was provided	wss:UnsupportedSecurityToken
An unsupported signature or encryption algorithm was used	wss:UnsupportedAlgorithm
An error was discovered processing the <wss:Security> header.	wss:InvalidSecurity
An invalid security token was provided	wss:InvalidSecurityToken
The security token could not be authenticated or authorized	wss:FailedAuthentication
The signature or decryption was invalid	wss:FailedCheck
Referenced security token could not be retrieved	wss:SecurityTokenUnavailable

De volgende foutmeldingen worden daar aan toegevoegd.

Omschrijving (faultstring)	Faultcode
Authenticatietoken en bericht stemmen niet overeen	ao:AuthTokenMessageMismatch
Authenticatietoken is niet valide of compleet	ao:AuthTokenInvalid
Authenticatietoken buiten geldigheidsduur ontvangen	ao:ExpirationTimeError
Nonce is reeds gebruikt	ao:NonceRejected

Om te voorkomen dat deze informatie misbruikt wordt om de beste aanval te bepalen, dient eerst de handtekening geverifieerd te worden, en pas daarna op de "AORTA" fouten gecontroleerd te worden.

Een applicatie dient er ook rekening mee te houden dat er fouten door een ontvanger op HTTP-niveau worden afgehandeld, met name door het retourneren van de HTTP statuscode 403 "Forbidden" of 404 "Not Found".

11.5 Relatie met pincode, sessies en SSL

De communicatie met de UZI-pas en gebruik pin is in wezen niet anders dan bij AORTA zonder gebruik authenticatietokens. Er is geen sessie tussen het werkstation en het LSP op basis van de UZI-pas meer, maar er kan wel een sessie zijn tussen het werkstation en de kaartlezer (een sessie is hier de mogelijkheid opnieuw gegevens te versleutelen met de UZI-pas zonder opnieuw een pincode in te hoeven geven). De principes blijven hetzelfde: een nieuwe pincode moet gevraagd worden nadat de UZI-pas een bepaalde periode niet gebruikt is (nu 15 minuten) en nadat een gebruiker de UZI-pas uit de kaartlezer heeft verwijderd (in de toekomst komt de optie om de pas binnen een bepaalde tijd terug te doen zonder de kaartlezersessie te beëindigen).

Bij tokenauthenticatie hoeft er geen directe SSL/TLS sessie tussen het werkstation en het LSP te zijn. Uiteraard mogen de gegevens tussen het werkstation en het LSP alleen beveiligd verzonden worden. Dat leidt tot de volgende eisen:

EIS: Gegevens die met authenticatietokens worden verzonden moeten binnen een GBZ op dergelijke wijze getransporteerd worden dat onbevoegden er redelijkerwijs geen toegang toe kunnen verkrijgen.

EIS: Gegevens die met authenticatietokens worden verzonden moeten tussen GBZ en LSP via een SSL/TLS verbinding verzonden worden die is opgezet met behulp van het servercertificaat van het GBZ.

EIS: Bij gegevens die met authenticatietokens worden verzonden moet de ontvanger controleren of de URA van het servercertificaat waarmee de SSL/TLS verbinding is opgezet, dezelfde is als de URA behorend bij de pas waarmee het token is ondertekend. Deze eis geldt niet bij extern gastgebruik.