|  |
| --- |
| NSI |
| Seal.Net |
| Version 2.0 |
|  |
| **Ivan Overgaard** |
| **11/29/2012** |

|  |
| --- |
|  |

**Revisionshistorik:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Dato** | **Ændring** | **Ansvarlig** |
| 0.8 | 29-11-2012 | Oprettet | IO |
| 1.0 | 04-04-2013 | redigeret | IO |

Indhold

[Revisionshistorik: 2](#_Toc352679649)

[Indledning 4](#_Toc352679650)

[Historik 4](#_Toc352679651)

[Systemkrav 4](#_Toc352679652)

[Leverancer 4](#_Toc352679653)

[Seal.Net Api’et 5](#_Toc352679654)

[Designvalg 5](#_Toc352679655)

[Objekter 5](#_Toc352679656)

[Brug af Seal.Net Api 9](#_Toc352679657)

[Klienteksempler 9](#_Toc352679658)

[Direkte kald 10](#_Toc352679659)

[Føderalt 11](#_Toc352679660)

[Brug af NemId: 12](#_Toc352679661)

[Secure browser logon 13](#_Toc352679662)

[SOSI Gateway 15](#_Toc352679663)

[Eksempler på generering af objekter fra proxy. 17](#_Toc352679664)

[Assertion 17](#_Toc352679665)

[Security 18](#_Toc352679666)

[Header 19](#_Toc352679667)

[MakeAssertionForSTS 20](#_Toc352679668)

# Indledning

Den Gode WebService (DGWS) er en profil for webservices, som bygger på flere WebService standarder fra WS\* stakken.   
Det er ikke en triviel opgave at designe en klient eller en webservice implementering, der overholder DGWS profilen.   
Seal.Net Api’ets formål er, at sænke den tærskel der uvægerligt er forbundet med at med at udvikle software der overholder Den Gode WebService. Seal.Net indpakker alle DGWS specifikke detaljer abstraherer alle typer fra XML til objektform. API’et tager sig af validering og signering aktuelle klasser.

API’et er leveret som.Net assemblies og kan benyttes med forskellige sprogbindinger herunder f.eks. VB.NET, C#, etc.

## Historik

Den Gode WebService er specificeret i tre versioner 1.0, 1.0.1 og 1.1. Ingen af versionerne er kompatible og der er tidligere udviklet individuelle Api’er til at understøtte disse versioner.

Dette Api understøtter alle nuværende versioner af DGWS i samme implementering for hhv. klient og service.

Der er dog væsentlige designforskelle mellem dette API og tidligere versioner.

# Systemkrav

Api’et er bygget til WCF (Windows Communication Foundation), WIF (Windows Identity Foundation), og kæver som minimum .Net version 3.5.

Læseren af denne guide forudsættes at have indsigt i C#, WCF, WIF og XML.

Denne dokumentation refererer til .Net 4.5. Enkelte interfaces er forskellige i .net 3.5 til .net 4.5.

F.eks. benyttes System.ServiceModel.Security.IWSTrustChannelContract interfacet i 4.5 dette interface er i .Net 3.5 Microsoft.IdentityModel.Protocols.WSTrust.IWSTrustChannelContract.

# Leverancer

Seal.Net leveres som to assemblies. En der assembly der indeholder datatyperne for til Seal og en der indeholder logikken Der leveres to versioner. En som benyttes i .Net version 3.5-4.0 og en version som benyttes fra og med .Net version 4.5.

Årsagen til at der leveres to versioner er, at WIF først er med i det samlede .Net framework fra .Net 4.5. Før version 4.5 leveredes WIF som en medfølgende assembly. Der er små forskelle i snitfladen mellem den medfølgende assembly og den endlige version. Derfor er der udviklet to versioner af Seal.Net. som fortløbende vil blive omtalt som hhv. Seal.net35 og Seal.net45.

Seal.net35 kræver en at en version tilgængelig af ”Microsoft.IdentityModel.dll” som følger med leverancen.

# Seal.Net Api’et

## Designvalg

Seal.net er bygget oven på WCF og WIF

Ovnnævnte Api’er er valgt da de er supporteret af Microsoft og indgår som standard i..Net frameworket fra .Net version 3.5. Yderligere giver WCF udpræget mulighed for at benytte aspekt orienteret udvikling.

Grundet inkompatibilitet mellem Saml2.0 og Seal Saml har det været nødvendigt i enkelte tilfælde at gå uden om WCF.

I en WSDL der beskriver en snitflade til Den Gode Webservice (DGWS) indgår skemaer der beskriver de specifikke DGWS klasser. Når der genereres en proxy til hhv. klient eller server, dannes disse klasser på typestærk form i den autogenererede proxy. Eksempler på genrerede klasser er Security, Assertion og Header.

Dette Api er designet benytte disse klasser i videst mulig omfang.

Designet tager yderlige højde for at alle klienter kan konfigureres både via kode og konfigurationsfiler

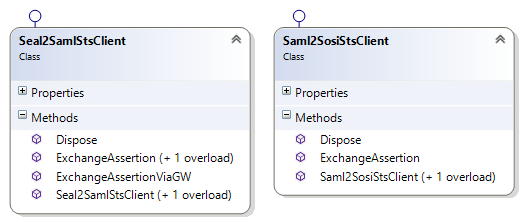
Ordforklaringer:

* SealAssertion:  
  Dækker over typerne som er specificeret i Seal. Det er ikke helt kompatible med Saml2 standarden.
* Saml2Assertion:  
  Er microsoft’s implementering af Saml2 som er en del af WIF.
* DGWSHeader:  
  Den Gode Webservice specificerer en ekstra headertype. DGWSHeader implementerer denne typer.

## Objekter

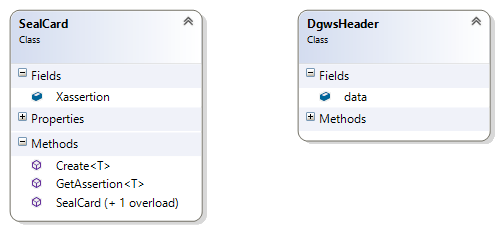
Følgende figurer illustrerer de klasser der indgår Seal.Net inddelt i grupper:

Klienter:



Seal2SalmStsClient koverterer en SealAssertion til en Saml2Assertion enten via direkte kald til sts eller via SOSI Gateway.

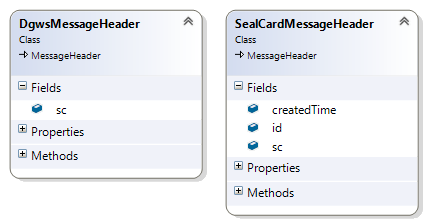
Kort



SealCard indpakker svaret fra en STS (assertion) og giver mulighed for benytte data i fremtidige kald.

DgwsHeader indpakker tilsvarende Medcom-header data

MessageHeaders



SealCardMessageHeader benyttes til at udstille et SealCard som en WCF MessageHeader

DgwsMessageHeader udstiller tilsvarende et DgwsHeader som en WCF MeassageHeader.

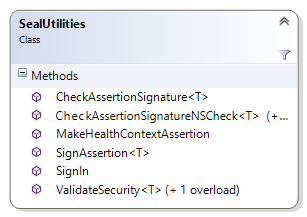
Endpointbehaviors



SealSigningEndpointBehavior tilpasser request XML header med manglende attributter og underskriver request v.h.a. et associeret certifikat i clientCredentials. Yderligere valideres underskriften af response.

SealEndpointBehavior implementerer DGWS.

Hjælpefunktioner



Indeholder forskellige funktioner f.eks. til at signere og validere en signatur.

# Brug af Seal.Net Api

I dette afsnit beskrives overordnet hvordan Seal.net benyttes til at opbygge en klient eller en service applikation.

## Klienteksempler

En klient til en WebService af typen Den Gode Webservice benyttes som en hver anden webservice.

I Visual Studio adderes en Service-Reference til den pågældende WSDL. Herefter genereres en proxy indeholdende alle datatyper for webservicen.

Det er også muligt at benytte WcfUtil.exe hvis ikke er tilstrækkelig.

Herefter kan der være flere klient-scenarier:

1. Direkte kald af Service (ikke føderalt)  
   Her oprettes et kort lokalt og Service Udbyderen kaldes med det oprettede kort. Det er så op til Service Udbyderen at autentificere brugeren,
2. Føderalt   
   Et kort oprettes lokalt valideres via et kald til en STS, som returnerer et kort der er digitalt underskrevet. Dette kort benyttes til fremtidige kald af webservices. Service Udbyderen skal nu kun autentificere STS’en.
3. Via NemId  
   Hvis en bruger allerede er logget på et system via NemLogin. Kan dette login benyttes til at kalde en webservice.  
   Til NemLogin er associeret en SamlToken. Denne token skal veksles til et IDKort til Den Gode Webservice. Når IDKortet er modtaget benyttes det til fremtidige kald
4. SOSI Gateway

Til efterfølgende eksempler benyttes nogle metoder til at oprette instanser af datatyper.

*MakeAssertion*, *MakeAssertionForSTS*, *MakeHeader* og *MakeSecurity*.

Eksemplerne viser et kald til en *FKM* webservice. Metodekadet er *GetMedicineCard\_20120101*

### Direkte kald

1. var client = new MedicineCardPortTypeClient("localFMK");
2. var ass = SealUtilities.SignAssertion( MakeAssertion(), g.global.cert );
3. client.GetMedicineCard\_20120101(MakeSecurity(ass), MakeHeader());
4. Der oprettes en instans af FMK klientklassen der benytter den navngivne konfiguration.
5. Der initieres en instans af *Assertion* som evt. underskrives
6. Assertion indsættes i et Security element. Der genereres en DGWSHeader og service kaldes

Følgende konfiguration er benyttes i ovenstående eksempel.

<endpoint address="http://localhost:1010/FMK"

binding="customBinding"

bindingConfiguration="Soap11Http"

behaviorConfiguration="sealbehavior"

contract="dk.nsi.fmk.MedicineCardPortType"

name="localFMK" />

<customBinding>

<binding name="Soap11Http">

<textMessageEncoding messageVersion="Soap11WSAddressing10" writeEncoding="utf-8" />

<httpTransport />

</binding>

</customBinding>

<behavior name="sealbehavior" >

<sbhe/>

</behavior>

<behaviorExtensions>

<add name="sbhe" type="dk.nsi.seal.SealBehaviorExtentionElement, Seal" />

</behaviorExtensions>

Som det ses er Binding er en customBinding uden sikkerhed og der benyttes SealBehaviorExtentionElement til at aktivere SealEndpointBehavior

### Føderalt

I et føderalt login skal en bruger først logges på føderationen

1. var rsc = SealCard.Create(MakeAssertionForSTS( g.global.NsiLge1));
2. SealCard sc = SealUtilities.SignIn(rsc, "http://www.ribeamt.dk/EPJ", "http://pan.certifikat.dk/sts/services/SecurityTokenService");
3. var client = new MedicineCardPortTypeClient("localFMK");
4. client.GetMedicineCard\_20120101( MakeSecurity(sc.GetAssertion<Assertion>()), MakeHeader() );
5. Der initieres et SealCard vha. en underskrevet Assertion.
6. STS kaldes med kortet. Der returneres et nyt SealCard underskrevet af STS
7. Der oprettes en instans af Service proxy klientklassen, i dette tilfælde FMK.
8. Der oprettes et Security element der indeholder den Assertion der er underskrevet af STS, samt et MedcomHeader element. Herefter kaldes Servicen.

Konfigurationen.

<endpoint address="http://localhost:1010/FMK"

binding="customBinding"

bindingConfiguration="Soap11Http"

behaviorConfiguration="sealbehavior"

contract="dk.nsi.fmk.MedicineCardPortType"

name="localFMK" />

<customBinding>

<binding name="Soap11Http">

<textMessageEncoding messageVersion="Soap11WSAddressing10" writeEncoding="utf-8" />

<httpTransport />

</binding>

</customBinding>

<behavior name="sealbehavior" >

<sbhe/>

</behavior>

<behaviorExtensions>

<add name="sbhe" type="dk.nsi.seal.SealBehaviorExtentionElement, Seal" />

</behaviorExtensions>

### Brug af NemId:

Efterfølgende kode viser hvordan en NemidLogin Saml2Assertion konverteres til en DGWSAssertion. For herefter at oprette en en Security instans der indeholder denne Assertion og kalde en FMK service.

1. Saml2Assertion nemidAssertion = MakeNemIdAssertion(g.global.cert);
2. SealCard sc;
3. using (var stsClient = new Saml2SosiStsClient("NemId")) {
4. stsClient.ChannelFactory.Credentials.ClientCertificate.Certificate =   
    g.global.NsiLge1;
5. sc = stsClient.ExchangeAssertion(  
    nemidAssertion,

SealUtilities.MakeHealthContextAssertion(  
 "Test Sundhed.dk",   
 g.global.cert.SubjectName.Name,   
 "Sygdom.dk",   
 "5GXFR"  
 ),  
 “<http://sosi.dk>”  
 );

}

1. var client = new MedicineCardPortTypeClient("localFMK");
2. client.GetMedicineCard\_20120101(   
    MakeSecurity( sc.GetAssertion<Assertion>() ),  
    MakeHeader()  
   );

I ”eksempler på generering af proxyklasser” vises implementeringen af MakeSecurity og MakeHeader som benyttes i ovenstående eksempel.

1. nemidAssertion forventes at være genereret i en tidligere longon scenarie
2. Klienten til den STS som veksler kortet initieres
3. Certifikatet til at underskrive requestet loades..
4. STS kaldes og der returneres et kort
5. En klient til FMK oprettes
6. Service kaldes med Assertion og DGWSHeader

Konfiguration:

.Net 4.5

<endpoint address="http://test1.ekstern-test.nspop.dk:8080/sts/services/OIOSaml2Sosi"

binding="basicHttpBinding"

behaviorConfiguration="SealSigning"

contract="System.ServiceModel.Security.IWSTrustChannelContract"

name="NemId" />

.Net 3.5

<endpoint address="http://test1.ekstern-test.nspop.dk:8080/sts/services/OIOSaml2Sosi"

binding="basicHttpBinding"

behaviorConfiguration="SealSigning"

contract="Microsoft.IdentityModel.Protocols.WSTrust.IWSTrustChannelContract"

name="NemId" />

<behavior name="SealSigning">

<SealSigningBE/>

</behavior>

<endpoint address="http://localhost:1010/FMK"

binding="customBinding"

bindingConfiguration="Soap11Http"

behaviorConfiguration="sealbehavior"

contract="dk.nsi.fmk.MedicineCardPortType"

name="localFMK" />

<customBinding>

<binding name="Soap11Http">

<textMessageEncoding messageVersion="Soap11WSAddressing10" writeEncoding="utf-8" />

<httpTransport />

</binding>

</customBinding>

<behavior name="sealbehavior" >

<sbhe/>

</behavior>

<behaviorExtensions>

<add name="sbhe" type="dk.nsi.seal.SealBehaviorExtentionElement, Seal" />

<add name="SealSigningBE" type="dk.nsi.seal.SealSigningBehaviorExtentionElement, Seal"/>

</behaviorExtensions>

### Secure browser logon

Efterfølgende kode og konfiguration viser hvordan der oprettes en krypteret Assertion som kan benyttes til SBO.

1. var rsc = SealCard.Create(MakeAssertionForSTS(g.global.NsiLge1));
2. var sc = SealUtilities.SignIn(rsc, "http://www.ribeamt.dk/EPJ", Properties.Settings.Default.SecurityTokenService);
3. using (var stsClient = new Seal2SamlStsClient("Seal2EncSaml")){
4. stsClient.ChannelFactory.Credentials.ClientCertificate.Certificate = g.global.NsiLge1;
5. var d = stsClient.ExchangeAssertion(sc, "http://sundhed.dk/") as GenericXmlSecurityToken;
6. var elm = d.TokenXml;}

1. Et SealCard opretts fra en local oprettet og signeret Assertion.

2. Et nyt SealCard oprettes underskrevet af STS.

3. En proxy til STS der veksler SealCard oprettes

4. ClientCredentials sættes.

5. STS til konvertering af assertion kaldes.

6. Det genererede krypterede kort hentes som XML

Konfiguration:

<endpoint address="http://test1.ekstern-test.nspop.dk:8080/sts/services/Sosi2OIOSaml"

binding="customBinding"

bindingConfiguration="Soap11Http"

behaviorConfiguration="SealSigning"

contract="System.ServiceModel.Security.IWSTrustChannelContract"

name="Seal2EncSaml" />

<customBinding>

<binding name="Soap11Http">

<textMessageEncoding messageVersion="Soap11WSAddressing10" writeEncoding="utf-8" />

<httpTransport />

</binding>

</customBinding>

<behavior name="SealSigning">

<SealSigningBE/>

</behavior>

<behaviorExtensions>

<add name="SealSigningBE" type="dk.nsi.seal.SealSigningBehaviorExtentionElement, Seal"/>

</behaviorExtensions>

### SOSI Gateway

Nå SOSI Gateway skal benyttes skal der ført oprettes en service reference SOSI Gateway. Hvis ikke der allerede er oprettet oprettes også en service reference til servicen, i nedenstående eksempel FMK.

Følgende kode opretter et login på SOSI Gateway.

using GW = SealTest.SosiGWReference;

public Assertion assertion;

public Header header;

public void DoLogin(X509Certificate2 cert){

1. var GWClient = new GW.SosiGWFacadeClient();
2. GW.Security sec = MakeSecurity(MakeAssertionForSTS());
3. var dig = GWClient.requestIdCardDigestForSigning(sec, "whatever");
4. var csp = (RSACryptoServiceProvider)cert.PrivateKey;
5. var sha1 = new SHA1Managed();
6. byte[] hash = sha1.ComputeHash(dig.DigestValue);
7. var rb = new GW.signIdCardRequestBody
8. {
   1. SignatureValue = csp.SignHash(hash, CryptoConfig.MapNameToOID("SHA1")),
   2. KeyInfo = new GW.KeyInfo()
   3. {
      1. Item = new GW.X509Data() { Item = cert.Export(X509ContentType.Cert) }
   4. }
9. };
10. var res = GWClient.signIdCard(sec, rb);
11. if (res != GW.signIdCardResponse.ok)
12. {
    1. throw new Exception("Gateway logon error");
13. }
14. header = MakeHeader();
15. assertion = SealCard.Create(sec.Assertion).
    * 1. GetAssertion<Assertion>(typeof(GW.AssertionType).Name);

}

1. DoLogin(global.NsiLge1);
2. var client = new MedicineCardPortTypeClient("SosiGWFMK");
3. client.GetMedicineCard\_20120101(MakeSecurity(assertion), header);

1) opretter en proxy til SOSI Gateway

2)Opretter en lokal assertion

3)Efterspørger en digest i SOSI Gateway.

4-9) Digest signeres

10) Der oprettes et kort i SOSI Gateway

14-15) Header og assertion konverteres til korrekt type og gemmes til senere brug.

16) Login kaldes

17) FMK klient oprettes.

18)Service kaldes

Konfiguration:

SOSIGW

<endpoint address="http://test1.ekstern-test.nspop.dk:8080/sosigw/service/sosigw"

binding="basicHttpBinding"

contract="SosiGWReference.SosiGWFacade" />

FMK

<endpoint address="https://triforkfaellestest.lms.trifork.com/fmk12/ws/MedicineCard"

binding="customBinding"

bindingConfiguration="Soap11Http"

behaviorConfiguration="AddressingBehavior"

contract="dk.nsi.fmk.MedicineCardPortType"

name="SosiGWFMK" />

<customBinding>

<binding name="Soap11Http">

<textMessageEncoding messageVersion="Soap11WSAddressing10" writeEncoding="utf-8" />

<httpTransport />

</binding>

</customBinding>

<behavior name="AddressingBehavior">

<clientVia viaUri="http://test1.ekstern-test.nspop.dk:8080/sosigw/proxy/soap-request" />

</behavior>

# 

### SOSI Gateway SBO

En token til SBO kan hentes fra SOSI Gateway på nedenstående måde. Det forudsættes at der er logget på SOSI Gateway og dermed er assertion initieret.

Endpoint refererer til STS og ClientVia refererer til SOSI Gateway.

using (var stsClient = new Seal2SamlStsClient("GWFetchCard"))

using (var scope = new OperationContextScope((IContextChannel)stsClient.Channel.Channel))

{

OperationContext.Current.OutgoingMessageHeaders.Add(new SealCardMessageHeader( SealCard.Create(assertion)));

var d = stsClient.ExchangeAssertionViaGW( "http://sundhed.dk/") as GenericXmlSecurityToken;

var elm = d.TokenXml;

}

Konfiguration:

<endpoint address="http://test1.ekstern-test.nspop.dk:8080/sts/services/Sosi2OIOSaml"

binding="customBinding"

behaviorConfiguration="AddressingBehavior"

bindingConfiguration="Soap11Http"

contract="System.ServiceModel.Security.IWSTrustChannelContract"

name="GWFetchCard" />

<customBinding>

<binding name="Soap11Http">

<textMessageEncoding messageVersion="Soap11WSAddressing10" writeEncoding="utf-8" />

<httpTransport />

</binding>

</customBinding>

<behavior name="AddressingBehavior">

<clientVia viaUri="http://test1.ekstern-test.nspop.dk:8080/sosigw/proxy/soap-request" />

</behavior>

# Eksempler på generering af objekter fra proxy.

Nedenstående eksempler viske initieringer at klasser der er autogenereret ud fra en DGWS WSDL.

## Assertion

Nedenstående eksempel opretter en Assertion hvor NotBefore sættes til at være 5 minutter gammel fra start for at sikre tiden i forhold til den modtagne server.

Det er et DGWS kort version 1.0.1 niveau 4.

static Assertion MakeAssertionForSTS( X509Certificate2 certificate )

{

var vnow = DateTimeEx.UtcNowRound - TimeSpan.FromMinutes(5);

var later = (vnow + TimeSpan.FromHours(8)).ToString("u").Replace(' ', 'T');

var ass= new Assertion

{

IssueInstant = vnow,

id = "IDCard",

Version = 2.0m,

Issuer = "WinPLC",

Conditions = new Conditions

{

NotBefore = vnow,

NotOnOrAfter = vnow + TimeSpan.FromHours(8)

},

Subject = new Subject

{

NameID = new NameID

{

Format = "http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/03/18/CPR\_PersonCivilRegistrationIdentifier.xsd",

Value = "2203333571"

},

SubjectConfirmation = new SubjectConfirmation

{

ConfirmationMethod = ConfirmationMethod.urnoasisnamestcSAML20cmholderofkey,

SubjectConfirmationData = new SubjectConfirmationData

{

Item = new proxy.KeyInfo

{

Item = "OCESSignature"

}

}

}

},

AttributeStatement = new AttributeStatement[]

{

new AttributeStatement

{

id = AttributeStatementID.IDCardData,

Attribute = new proxy.Attribute[]

{

new proxy.Attribute{ Name = "sosi:IDCardID", AttributeValue = Guid.NewGuid().ToString("D")},

new proxy.Attribute{ Name = "sosi:IDCardVersion", AttributeValue = "1.0.1"},

new proxy.Attribute{ Name = "sosi:IDCardType", AttributeValue = "user"},

new proxy.Attribute{ Name = "sosi:AuthenticationLevel", AttributeValue = "4"},

}

},

new AttributeStatement

{

id = AttributeStatementID.UserLog,

Attribute = new proxy.Attribute[]

{

new proxy.Attribute{ Name = "medcom:UserCivilRegistrationNumber", AttributeValue = "2203333571"},

new proxy.Attribute{ Name = "medcom:UserGivenName", AttributeValue = "Amaja Christiansen"},

new proxy.Attribute{ Name = "medcom:UserSurName", AttributeValue = "Person"},

new proxy.Attribute{ Name = "medcom:UserEmailAddress", AttributeValue = "jso@trifork.com"},

new proxy.Attribute{ Name = "medcom:UserRole", AttributeValue = "1234"},

new proxy.Attribute{ Name = "medcom:UserAuthorizationCode", AttributeValue = "NS101"}

}

},

new AttributeStatement

{

id = AttributeStatementID.SystemLog,

Attribute = new proxy.Attribute[]

{

new proxy.Attribute{ Name = "medcom:ITSystemName", AttributeValue = "Sygdom.dk"},

new proxy.Attribute{ Name = "medcom:CareProviderID", AttributeValue = "25520041", NameFormat="medcom:cvrnumber"},

new proxy.Attribute{ Name = "medcom:CareProviderName", AttributeValue = "TRIFORK SERVICES A/S // CVR:25520041"},

}

}

}

};

return certificate == null ? ass : SealUtilities.SignAssertion(ass, certificate);

}

## Security

Nedenstående eksempel opretter en Security element Timestamp sættes til at være 5 minutter gammel.

static Security MakeSecurity( Assertion assertion)

{

return new Security

{

id = Guid.NewGuid().ToString("D"),

Timestamp = new Timestamp { Created = DateTime.Now - TimeSpan.FromMinutes(5) },

Assertion = assertion

};

}

## Header

static Header MakeHeader()

{

return new Header

{

SecurityLevel = 3,

TimeOut = TimeOut.Item1440,

TimeOutSpecified = true,

Linking = new Linking

{

FlowID = Guid.NewGuid().ToString("D"),

MessageID = Guid.NewGuid().ToString("D")

},

FlowStatus = FlowStatus.flow\_running,

FlowStatusSpecified = true,

Priority = Priority.RUTINE,

RequireNonRepudiationReceipt = RequireNonRepudiationReceipt.yes

};

}

## MakeAssertionForSTS

static Assertion MakeAssertionForSTS()

{

var vnow = DateTime.UtcNow - TimeSpan.FromMinutes(5);

var later = (vnow + TimeSpan.FromHours(8)).ToString("u").Replace(' ', 'T');

return new Assertion

{

IssueInstant = vnow,

id = "IDCard",

Version = 2.0m,

Issuer = "WinPLC",

Conditions = new Conditions

{

NotBefore = vnow,

NotOnOrAfter = vnow + TimeSpan.FromHours(8)

},

Subject = new Subject

{

NameID = new NameID

{

Format = "http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/03/18/CPR\_PersonCivilRegistrationIdentifier.xsd",

Value = "2203333571"

},

SubjectConfirmation = new SubjectConfirmation

{

ConfirmationMethod = ConfirmationMethod.urnoasisnamestcSAML20cmholderofkey,

SubjectConfirmationData = new SubjectConfirmationData

{

Item = new proxy.KeyInfo

{

Item = "OCESSignature"

}

}

}

},

AttributeStatement = new AttributeStatement[]

{

new AttributeStatement

{

id = AttributeStatementID.IDCardData,

Attribute = new proxy.Attribute[]

{

new proxy.Attribute{ Name = "sosi:IDCardID", AttributeValue = Guid.NewGuid().ToString("D")},

new proxy.Attribute{ Name = "sosi:IDCardVersion", AttributeValue = "1.0.1"},

new proxy.Attribute{ Name = "sosi:IDCardType", AttributeValue = "user"},

new proxy.Attribute{ Name = "sosi:AuthenticationLevel", AttributeValue = "4"},

//new proxy.Attribute{ Name = "sosi:OCESCertHash", AttributeValue = g.global.cprCertMap["2203333571"].GetCertHashString()}

}

},

new AttributeStatement

{

id = AttributeStatementID.UserLog,

Attribute = new proxy.Attribute[]

{

new proxy.Attribute{ Name = "medcom:UserCivilRegistrationNumber", AttributeValue = "2203333571"},

new proxy.Attribute{ Name = "medcom:UserGivenName", AttributeValue = "Amaja Christiansen"},

new proxy.Attribute{ Name = "medcom:UserSurName", AttributeValue = "Person"},

new proxy.Attribute{ Name = "medcom:UserEmailAddress", AttributeValue = "jso@trifork.com"},

new proxy.Attribute{ Name = "medcom:UserRole", AttributeValue = "1234"},

new proxy.Attribute{ Name = "medcom:UserAuthorizationCode", AttributeValue = "NS101"}

}

},

new AttributeStatement

{

id = AttributeStatementID.SystemLog,

Attribute = new proxy.Attribute[]

{

new proxy.Attribute{ Name = "medcom:ITSystemName", AttributeValue = "Sygdom.dk"},

new proxy.Attribute{ Name = "medcom:CareProviderID", AttributeValue = "25520041", NameFormat="medcom:cvrnumber"},

new proxy.Attribute{ Name = "medcom:CareProviderName", AttributeValue = "TRIFORK SERVICES A/S // CVR:25520041"},

}

}

}

};

}

# Referencer

|  |  |
| --- | --- |
| **Forkortelse i teksten** | **Henvisning** |
| [SAML2.0] | https://www.oasis-open.org/committees/tc\_home.php?wg\_abbrev=security |
| [OIOSAML] | http://digitaliser.dk/resource/2377872 |
| [OIOIDWS] | http://digitaliser.dk/resource/526486 |