|  |
| --- |
| 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 |
| 3.0.0 | 03-01-2017 | Fjernet .net 3.5 referancer | FMO |
| 4.0.0 | 26-06-2017 | Tilføjet IdCards og OioSamlFactory | KRO og SKS |

Indhold

[Revisionshistorik: 2](#_Toc487881346)

[Indledning 4](#_Toc487881347)

[Historik 4](#_Toc487881348)

[Systemkrav 4](#_Toc487881349)

[Leverancer 4](#_Toc487881350)

[Seal.Net Api’et 5](#_Toc487881351)

[Designvalg 5](#_Toc487881352)

[Ordforklaringer 5](#_Toc487881353)

[Objekter 5](#_Toc487881354)

[Brug af Seal.Net Api 12](#_Toc487881355)

[Klienteksempler 12](#_Toc487881356)

[Direkte kald 13](#_Toc487881357)

[Føderalt 15](#_Toc487881358)

[Brug af NemId: 17](#_Toc487881359)

[Secure browser logon 18](#_Toc487881360)

[SOSI Gateway 20](#_Toc487881361)

[SOSI Gateway SBO 22](#_Toc487881362)

[Eksempler ved brug af OioSamlFactory 23](#_Toc487881363)

[Brug af NemId: 23](#_Toc487881364)

[Eksempler på generering af objekter fra proxy 24](#_Toc487881365)

[CallingSystem og User 24](#_Toc487881366)

[Security 25](#_Toc487881367)

[Header 25](#_Toc487881368)

[Make NemId Assertion 26](#_Toc487881369)

[Referencer 27](#_Toc487881370)

# 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.

Seal.Net kan ligeledes også installeres som NuGet pakke.

## 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 kræver I nyeste version som minimum .Net version 4.6.2.

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

# Leverancer

Seal.Net leveres som to assemblies. En der assembly der indeholder datatyperne for til Seal og en der indeholder logikken.

# Seal.Net Api’et

## Designvalg

Seal.net er bygget oven på WCF og WIF

Ovennævnte 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 genererede 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.

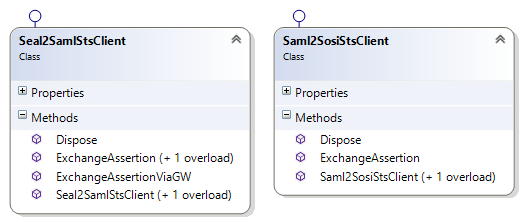
##### DGWSHeader

Den Gode Webservice specificerer en ekstra headertype. DGWSHeader implementerer denne typer.

## Objekter

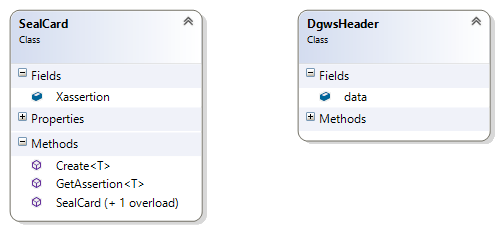
Følgende figurer illustrerer de klasser der indgår Seal.Net inddelt i grupper:

##### Klienter



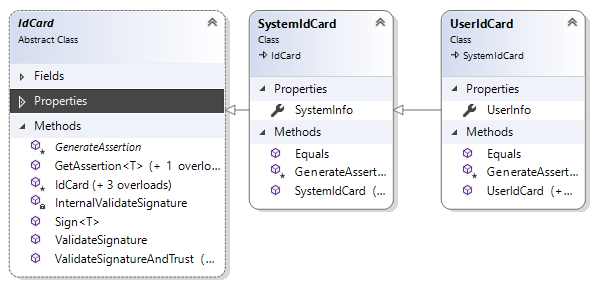
Seal2SalmStsClient koverterer en SealAssertion til en Saml2Assertion enten via direkte kald til sts eller via SOSI Gateway.

##### Kort



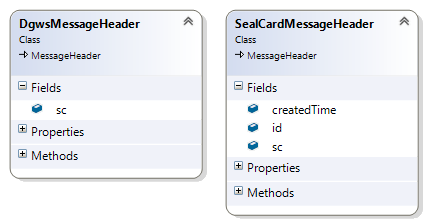
SealCard indpakker svaret fra en STS (assertion) og giver mulighed for benytte data i fremtidige kald. SealCard er deprecated og brugen af IdCards (nederst) er derfor at foretrække.

DgwsHeader indpakker tilsvarende Medcom-header data



Nye kort er bleven tilføjet som er IdCards. IdCard er en abstract klasse så man bruger enten et SystemIdCard eller UserIdCard. System eller UseridCard bruges hhv. for at representere et system eller en bruger.

##### MessageHeaders



SealCardMessageHeader benyttes til at udstille et SealCard som en WCF MessageHeader

DgwsMessageHeader udstiller tilsvarende et DgwsHeader som en WCF MeassageHeader.

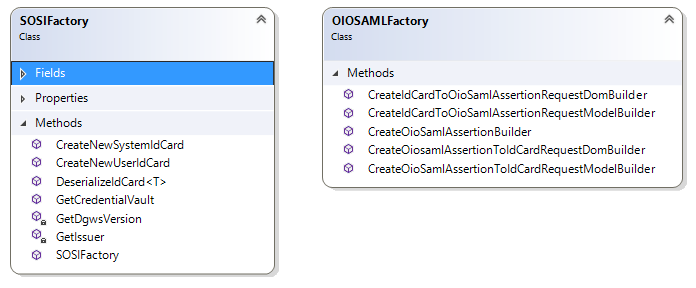
##### Endpointbehaviors



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.

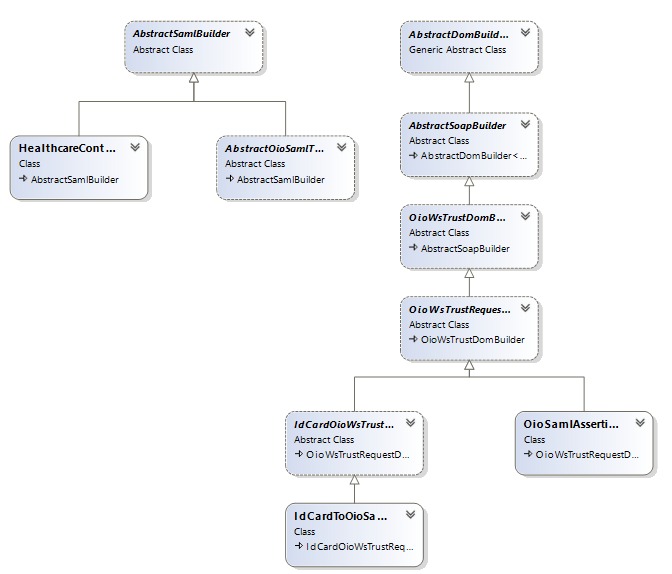
##### Factories



SosiFactory kan generere nye IdCards.

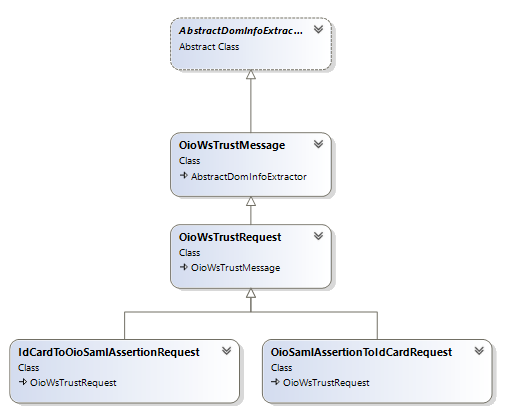
OioSamlFactory kan generere nye Requests. Normalt laver man en DOM builder som genererer selve XML’en og bagefter en ModelBuilder for at lave selve Requesten. Requesten kan så blive sendt ved hjælp af SealUtilities SignIn funktion.

##### DomBuilders



Dombuilders er et hierarki, som bliver brugt til at generere XML kode. Til venstre er der 3 klasser der håndterer bygning af selve assertions, mens det højre klasse hierarki bliver brugt til at generere selve Soap beskeden uden Assertion.

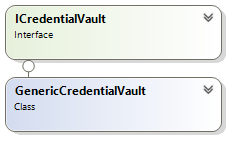
##### Requests



Requests er selve modeller over XML’en. De har diverse convenience methods til at hente informationer direkte ud fra XML'en.

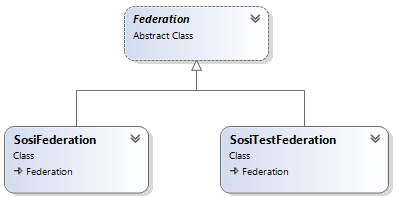
##### Vaults

Vi har implementeret et generelt vault system som bliver brugt til at gemme certifikater i. GenericCredentialVault bruger en standart .Net CertificateStore til at gemme certifikater i. Der kan laves nye CredentialVaults som kan bruges i SosiFactory og andre steder i Seal.Net ved at nedarve fra ICredentialVault.

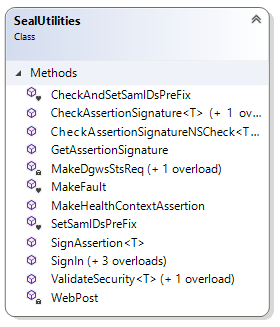


##### Federation

Federations bliver brugt når man skal validere certfikatet fra den service man snakker med. SosiFederation neders vil validere servicens certifikat op mod OCES2 root certifikatet for at sikre, at servicen er authoriseret. SosiTestFederation vil validere op mod OCES2 test root certifikat. Hvis ingen Federation bliver brugt, skal root certifikatet af den pågældende services certifikat (eller selve certifikatet) ligge i den brugte CredentialVault.



##### Hjælpefunktioner



Indeholder forskellige funktioner f.eks. til at signere og validere en signatur. Derudover kan man sende requests fra OioSamlFactory afsted uden brug af clienter.

# Brug af Seal.Net Api

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

## Konfiguration

Der er to måder at konfigurere certifikat validering på i Seal.Net. Begge kan aktiveres/deaktiveres i App.config eller Web.config.

##### CheckTrust

CheckTrust specificerer om certifikatet skal valideres op mod et root certifikat og om chainen skal valideres. Hvis en federation er specificeret så er det valideringen mod den federation som bliver aktiveret-deaktiveret. Skal slås fra ved self signed certifikater.

<appSettings>

<add key="CheckTrust" value="false"/>

</appSettings>

CheckTrust er per default True.

##### CheckCrl

CheckCrl specificerer om certifikatet skal checkes op mod en Certificate Revocation List. Skal slås fra ved self signed certifikater.

<appSettings>

<add key="CheckCrl" value="false"/>

</appSettings>

CheckCrl er per default False.

## Klienteksempler

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

I Visual Studio tilføjes 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 Visual Studio 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.

*MakeHeader* og *MakeSecurity*.

Disse metoder opretter instanser af de før nævnte klasser; *Security* og *Header*.

Der benyttes desuden variablerne *callingSystem* og *user*, som kan ses under ” eksempler på generering af proxyklasser”.

Eksemplerne viser et kald til en *FKM* webservice. Metodekaldet er *GetMedicineCard\_2015\_06\_01*

### Direkte kald

1. var client = new MedicineCardPortTypeClient("MedicineCardPort");
2. GenericCredentialVault vault = new GenericCredentialVault("FMKTestStore");

vault.SetSystemCredentials(user.Certificate);

CredentialVaultSignatureProvider sigProvider

= new CredentialVaultSignatureProvider(vault);

SOSIFactory factory = new SOSIFactory(null, sigProvider);

1. UserInfo userInfo = new UserInfo(user.Cpr, user.GivenName, user.SurName, user.Email, user.Occupation, user.Role, user.AuthCode);

var idCardRequest = factory.CreateNewUserIdCard(callingSystem.SystemName, userInfo, new CareProvider(callingSystem.CareProviderIdFormat, callingSystem.CareProviderId, callingSystem.CareProviderName), AuthenticationLevel.MocesTrustedUser, "", "", user.Certificate, "");

1. idCardRequest.Sign<Assertion>(factory.SignatureProvider);
2. var response = client.GetMedicineCard\_2015\_06\_01(

new GetMedicineCardRequest\_2015\_06\_01

{

Security = SecurityHeaderUtil.MakeSecurityUsingDgwsTypes(idCardRequest),

Header = MakeHeader(),

WhitelistingHeader = MakeWhitelistingHeader(),

GetMedicineCardRequest = new GetMedicineCardRequestType

{

PersonIdentifier = new PersonIdentifierType

{

source = "CPR",

Value = "1802602810"

}

}

});

1. Der oprettes en instans af FMK klientklassen der benytter den navngivne konfiguration.
2. Der initieres en instans *SOSIFactory* til at generere et *IdCard*
3. Et *UserIdCard* genereres vha. factory’en.
4. Id-kortet signeres.
5. Servicen kaldes.

Følgende konfiguration er benyttes i ovenstående eksempel.

<endpoint

address="https://test2.fmk.netic.dk/fmk12/ws/MedicineCard"

behaviorConfiguration="sealbehavior"

binding="basicHttpBinding"

bindingConfiguration="MedicineCardBinding"

contract="MedicinCard.MedicineCardPortType"

name="MedicineCardPort"

/>

<basicHttpBinding>

<binding name="MedicineCardBinding">

<security mode="Transport"/>

</binding>

</basicHttpBinding>

<behavior name="sealbehavior" >

<sbhe/>

</behavior>

<behaviorExtensions>

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

</behaviorExtensions>

Som det ses er benyttes der 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 factory = CreateFactory();

factory.GetCredentialVault().SetSystemCredentials(user.Certificate);

UserInfo userInfo = new UserInfo(user.Cpr, user.GivenName, user.SurName, user.Email, user.Occupation, user.Role, user.AuthCode);

var idCardRequest = factory.CreateNewUserIdCard(system.SystemName, userInfo, new CareProvider(system.CareProviderIdFormat, system.CareProviderId, system.CareProviderName), AuthenticationLevel.MocesTrustedUser, "", "", user.Certificate, "");

idCardRequest.Sign<Assertion>(factory.SignatureProvider);

1. var idc = SealUtilities.SignIn(idCardRequest, system.SystemName, "http://test2.ekstern-test.nspop.dk:8080/sts/services/NewSecurityTokenService");
2. var client = new MedicineCardPortTypeClient("MedicineCardPort");
3. var response = client.GetMedicineCard\_2015\_06\_01(

new GetMedicineCardRequest\_2015\_06\_01

{

Security = SecurityHeaderUtil.MakeSecurityUsingDgwsTypes(idc),

Header = requestHeader,

WhitelistingHeader = makeWhitelistingHeader,

GetMedicineCardRequest = new GetMedicineCardRequestType

{

PersonIdentifier = new PersonIdentifierType

{

source = "CPR",

Value = requestCpr

}

}

});

1. Der initieres et signeret *IdCard* vha. en *SOSIFactory* ud fra *user* og *callingSystem*
2. STS kaldes med kortet. Der returneres et nyt IdCard underskrevet af STS
3. Der oprettes en instans af Service proxy klientklassen, i dette tilfælde FMK.
4. Der oprettes et Security element der indeholder den Assertion der er underskrevet af STS, samt et MedcomHeader element. Herefter kaldes Servicen.

Konfigurationen er det samme som ved Direkte kald i forrige afsnit.

<endpoint

address="https://test2.fmk.netic.dk/fmk12/ws/MedicineCard"

behaviorConfiguration="sealbehavior"

binding="basicHttpBinding"

bindingConfiguration="MedicineCardBinding"

contract="MedicinCard.MedicineCardPortType"

name="MedicineCardPort"

/>

<basicHttpBinding>

<binding name="MedicineCardBinding">

<security mode="Transport"/>

</binding>

</basicHttpBinding>

<behavior name="sealbehavior" >

<sbhe/>

</behavior>

<behaviorExtensions>

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

</behaviorExtensions>

### Secure browser logon

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

1. SOSIFactory factory = CreateFactory();

UserInfo userInfo = new UserInfo(user.Cpr, user.GivenName, user.SurName, user.Email, user.Occupation, user.Role, user.AuthCode);

var idCardRequest = factory.CreateNewUserIdCard(system.SystemName, userInfo, new CareProvider(system.CareProviderIdFormat, system.CareProviderId, system.CareProviderName), AuthenticationLevel.MocesTrustedUser, "", "", user.Certificate, "");

idCardRequest.Sign<Assertion>(factory.SignatureProvider);

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

1. Et IdCard oprettes fra *SOSIFactory*.

2. Et nyt IdCard oprettes underskrevet af STS.

3. En proxy til STS der veksler IdCard 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;

private static Assertion LoginToGateway(IdCard idc, X509Certificate2 certificate)

{

// Convert the dgwsType Assertion into a sosi gateway Assertion.

var assertion = idc.GetAssertion<GW.AssertionType>();

var security = new GW.Security

{

Timestamp = new GW.Timestamp { Created = DateTimeEx.UtcNowRound - TimeSpan.FromMinutes(5) },

Assertion = assertion

};

using (var gwClient = new GW.SosiGWFacadeClient())

{

// Get an digest form the sosi gateway that should be signed

var dig = gwClient.requestIdCardDigestForSigning(security, "whatever");

// Create SHA1 hash of digest

var sha1Managed = new SHA1Managed();

var computeHash = sha1Managed.ComputeHash(dig.DigestValue);

// Get the private key

var privateKey = (RSACryptoServiceProvider) certificate.PrivateKey;

// Calculate the needed signature

var signatureValue = privateKey.SignHash(computeHash, CryptoConfig.MapNameToOID("SHA1"));

var cardRequestBody = new GW.signIdCardRequestBody

{

SignatureValue = signatureValue,

KeyInfo = new GW.KeyInfo

{

Item = new GW.X509Data {Item = certificate.Export(X509ContentType.Cert)}

}

};

// The unsigned id-card in the cache is combined with the signature and the certificate and is sent to STS.

// The STS signed card is saved in the sosiGw cache and is used for future calls through the proxy

var res = gwClient.signIdCard(security, cardRequestBody);

if (res != GW.signIdCardResponse.ok)

{

throw new Exception("Gateway logon error");

}

// Convert the GW Assertion to a dgwsType Assertion for later use.

idc.Xassertion = SerializerUtil.Serialize(security.Assertion).Root;

return idc.GetAssertion<Assertion>(typeof(GW.AssertionType).Name);

}

1. var factory = CreateFactory();

var idCardRequest = factory.CreateNewUserIdCard(system.SystemName, userInfo, new CareProvider(system.CareProviderIdFormat, system.CareProviderId, system.CareProviderName), AuthenticationLevel.MocesTrustedUser, "", "", user.Certificate, "");

1. LoginToGateway(idCardRequest, global.NsiLge1);
2. var client = new MedicineCardPortTypeClient("SosiGWFMK");
3. var response = client.GetMedicineCard\_2015\_06\_01(

new GetMedicineCardRequest\_2015\_06\_01

{

Security = SecurityHeaderUtil.MakeSecurityUsingDgwsTypes(idc),

Header = requestHeader,

WhitelistingHeader = makeWhitelistingHeader,

GetMedicineCardRequest = new GetMedicineCardRequestType

{

PersonIdentifier = new PersonIdentifierType

{

source = "CPR",

Value = requestCpr

}

}

});

1. Opretter en SosiFactory og IdCardRequest

2. Kalder LoginToGateway som har inline kommentar

3. FMK klient oprettes.

4. Service kaldes

Konfiguration:

SOSIGW

<endpoint address="http://test2.ekstern-test.nspop.dk:8080/sosigw/service/sosigw"

binding="basicHttpBinding"

contract="SosiGwService.SosiGWFacade"

name="SosiGWSoapBinding"/>

FMK

<endpoint address="https://test2.fmk.netic.dk/fmk12/ws/MedicineCard"

behaviorConfiguration="AddressingBehavior"

binding="customBinding"

bindingConfiguration="Soap11Http"

contract="MedicinCard.MedicineCardPortType"

name="SosiGWFMK"/>

<customBinding>

<binding name="Soap11Http">

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

<httpTransport />

</binding>

</customBinding>

<behavior name="AddressingBehavior">

<clientVia viaUri="http://test2.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))

{

var factory = CreateFactory();

OperationContext.Current.OutgoingMessageHeaders.Add(new IdCardMessageHeader( factory.DeserializeIdCard(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://test2.ekstern-test.nspop.dk:8080/sosigw/proxy/soap-request"/>

</behavior>

## Eksempler ved brug af OioSamlFactory

### Brug af NemId:

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

1. var callingSystem = new CallingSystem(

systemName: "Seal.Net.Fmk.Demo",

careProviderId: "30808460",

careProviderName: "TRIFORK SERVICES A/S",

careProviderIdFormat:

dk.nsi.seal.dgwstypes.SubjectIdentifierType.medcomcvrnumber,

certificate: new X509Certificate2("Resources/certificates/Statens\_Serum\_Institut\_FOCES.p12", "Test1234"));

var user = new User(

cpr: "1802602810",

givenName: "Stine",

surName: "Svendsen",

email: "stineSvendsen@example.com",

role: "læge",

authCode: "ZXCVB",

occupation: "Overlæge",

certificate:

new X509Certificate2("Resources/certificates/MOCES\_cpr\_gyldig.p12", "Test1234"));

1. var userVault = new GenericCredentialVault("UserVault");

userVault.SetSystemCredentials(user.Certificate);

// This nemidAssertion is expected to have been generated at an earlier login.

1. var nemidAssertion = NemIdAssertionBuilder.MakeNemIdAssertion(user.Certificate, system.Certificate, system, user);
2. var assertion = new OioSamlAssertion(samlAssertion);
3. var factory = new OIOSAMLFactory();
4. var domBuilder = factory.CreateOiosamlAssertionToIdCardRequestDomBuilder();

domBuilder.SigningVault = userVault;

domBuilder.OioSamlAssertion = assertion;

domBuilder.ItSystemName = system.SystemName;

domBuilder.UserAuthorizationCode = user.AuthCode;

domBuilder.UserEducationCode = null;

domBuilder.UserGivenName = user.GivenName;

domBuilder.UserSurName = user.SurName;

1. var requestDoc = domBuilder.Build();
2. var request = factory.CreateOioSamlAssertionToIdCardRequestModelBuilder() .Build(requestDoc);
3. var idCard = SealUtilities.SignIn(request, "http://test2.ekstern-test.nspop.dk:8080/sts/services/OIOSaml2Sosi");
4. var client = new MedicineCardPortTypeClient(\_fmkEndpoint);
5. using (new OperationContextScope(client.InnerChannel))

{

var response = client.GetMedicineCard\_2015\_06\_01(

new GetMedicineCardRequest\_2015\_06\_01

{

Security = SecurityHeaderUtil.MakeSecurityUsingDgwsTypes(idCard),

Header = MakeHeader(),

WhitelistingHeader = MakeWhitelistingHeader(),

GetMedicineCardRequest = new GetMedicineCardRequestType

{

PersonIdentifier = new PersonIdentifierType

{

source = "CPR",

Value = "2603558084"

}

}

});

Her kan man se brugen af OioSamlFactory til at udveksle en Nemid assertion med FMK.

1. Generer bruger og system information for nem tilgængelighed.
2. Opret ny CredentialVault med bruger certifikat.
3. Opret en ny Saml2Assertion ud fra informationerne.
4. Opret OioSamlAssertion ud fra NemidAssertion
5. Opret ny OioSamlFactory
6. Opret ny OiosamlAssertionToIdCardRequestDomBuilder og initialiser med korrekte informationer.
7. Byg dokumentet
8. Opret OioSamlAssertionToIdCardRequestModelBuilder og byg requesten.
9. Brug Sealutilities.Signin til at sende requesten til serveren. Returnerer nyt IdCard signed fra STS
10. Opret ny klient til FMK kommunikation
11. Kald FMK.

# Eksempler på generering af objekter fra proxy

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

## CallingSystem og User

var callingSystem = new CallingSystem(

systemName: "Seal.Net.Fmk.Demo",

careProviderId: "30808460",

careProviderName: "TRIFORK SERVICES A/S",

careProviderIdFormat: dk.nsi.seal.dgwstypes.SubjectIdentifierType.medcomcvrnumber,

certificate: new X509Certificate2("Resources/certificates/Statens\_Serum\_Institut\_FOCES.p12", "Test1234"));

var user = new User(

cpr: "1802602810",

givenName: "Stine",

surName: "Svendsen",

email: "stineSvendsen@example.com",

role: "læge",

authCode: "ZXCVB",

occupation: "Overlæge",

certificate: new X509Certificate2("Resources/certificates/MOCES\_cpr\_gyldig.p12", "Test1234"));

## 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

};

}

## Make NemId Assertion

public static Saml2Assertion MakeNemIdAssertion(

X509Certificate2 userCertificate,

X509Certificate2 signingCertificate,

CallingSystem system,

User user

)

{

var ass = new Saml2Assertion(new Saml2NameIdentifier("https://saml.test-nemlog-in.dk/"))

{

Conditions = new Saml2Conditions

{

NotOnOrAfter = DateTime.Now + TimeSpan.FromHours(8),

NotBefore = DateTime.Now

},

Subject = new Saml2Subject(new Saml2NameIdentifier(userCertificate.SubjectName.Name))

};

ass.Subject.SubjectConfirmations.Add(

new Saml2SubjectConfirmation(new Uri("urn:oasis:names:tc:SAML:2.0:cm:bearer"))

{

SubjectConfirmationData = new Saml2SubjectConfirmationData

{

NotOnOrAfter = DateTime.Now + TimeSpan.FromHours(8),

Recipient = new Uri("https://staging.fmk-online.dk/fmk/saml/SAMLAssertionConsumer")

}

});

IList<Saml2Attribute> q = new List<Saml2Attribute>();

// Spec

q.Add(new Saml2Attribute(OioSamlAttributes.SpecVersion, SpecVersion) {NameFormat = BasicNameFormat});

// User

q.Add(new Saml2Attribute(OioSamlAttributes.CommonName, user.GivenName) {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.Surname, user.SurName) {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.Email, user.Email) {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.CprNumber, user.Cpr) {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.AssuranceLevel, "4") {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.UserCertificate, Convert.ToBase64String(userCertificate.RawData)) {NameFormat = BasicNameFormat});

// Organization

q.Add(new Saml2Attribute(OioSamlAttributes.CvrNumber, system.CareProviderId) {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.OrganizationName, system.CareProviderName) {NameFormat = BasicNameFormat});

// Certificate

var subjectSerialNumber = userCertificate.SubjectName.Name;

q.Add(new Saml2Attribute(OioSamlAttributes.CertificateSerial, userCertificate.GetSerialNumberString()) {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.CertificateIssuer, userCertificate.IssuerName.Name) {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.Uid, ExtractUidNumber(subjectSerialNumber)) {NameFormat = BasicNameFormat});

q.Add(new Saml2Attribute(OioSamlAttributes.RidNumber, ExtractRidNumber(subjectSerialNumber)) {NameFormat = BasicNameFormat});

ass.Statements.Add(new Saml2AttributeStatement(q));

ass.Statements.Add(

new Saml2AuthenticationStatement(

new Saml2AuthenticationContext(new Uri("element:urn:oasis:names:tc:SAML:2.0:ac:classes:X509")),

DateTime.Now));

ass.SigningCredentials = new X509SigningCredentials(signingCertificate, SignedXml.XmlDsigRSASHA1Url, SignedXml.XmlDsigSHA1Url);

return ass;

}

# 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 |