GLOBETEAM

Danmarks Miljøportal (DMP)

Vejledning til fagsystemejere omkring tilkobling af .NET-baseret web service

Version 1.4

Indledning

Denne vejledning beskriver hvordan man claims-enabler en .Net-baseret web service (WS) til brug med DMP's føderale brugerstyringsløsning ved brug af Microsoft Windows Identity Foundation (WIF).

Vejledningen skal læses i konteksten af den overordnede vejledning "Vejledning til fagsystemejere omkring forløbet for tilkobling af en applikation".

Vejledningen forudsætter at Microsoft Windows Identity Foundation (WIF) benyttes til claims-enabling af web service'n. Denne vejledning er således ikke brugbar, såfremt man ønsker at anvende et andet API til claims-enabling af web services.

Indhold

Indledning 2
Indhold 3
Forudsætninger for tilkobling af en .Net-baseret web service 4
WIF 4
.NET Framework 3.5 SP1 4
Certifikaterne er din største fejlkilde 4
Tilkoblingen af en .Net-baseret WS 5
Sådan kobles WS'en op mod DMP 6
Trin 1: Anskaffelse af det nødvendige certifikat 6
Applikationens krypteringscertifikat 6
Trin 2: Konfiguration af webapplikations web.config-fil 6
Information omkring sikkerhedsrelateret logning 6
Step-by-step eksempel på tilkoblingen af de .NET-baserede samples 7
Appendiks A. Tracing 15
Appendiks B. Beskrivelse af WS'ens web.config-fil 17
Appendiks C. Brug af ActAs til at kalde på vegne af en bruger 19
Appendiks D. Web Services føderation - login mod organisationens lokale AD FS 2.0 21
Appendiks E: Brugen af WCF 3.0-baserede klienter til kald af web services på DMPs Identify STS uden brug af Windows Identity Foundation 23

Forudsætninger for tilkobling af en .Net-baseret web service

Såfremt de involverede .Net-udviklere ikke allerede er fortrolige med claims og claims-enabling af web services på .Net-platformen kan det anbefales at studere Microsofts onlinebog "A Guide to Claims–based Identity and Access Control" fra Patterns & Practices-serien (http://msdn.microsoft.com/en-us/library/ff423674.aspx).

WIF

Som nævnt i indledningen er vejledningen bygget på en forudsætning om at claims-enablingen sker ved brug af Windows Identity Foundation (WIF).

WIF tager rent praktisk form af en hotfix og kan hentes på Download Center (http://www.microsoft.com/downloads/details.aspx?FamilyID=eb9c345f-e830-40b8-a5fe-ae7a864c4d76&displaylang=en). Der findes også en version af WIF til Windows 2003, der kan hentes på http://www.microsoft.com/downloads/details.aspx?FamilyID=be4db6a0-b76d-446d-810c-ea3c25b3969a&displaylang=en.

WIF kan afvikles på Windows Server 201, Windows Server 2008 R2, Windows Server 2008 Service Pack 2, Windows 2003 SP2, Windows 7 og Windows Vista Service Pack 2. WIF til Windows 2008 forudsætter at der allerede er installeret IIS 7.0 og Microsoft .NET Framework 3.5 eller højere på platformen, mens WIF til Windows 2003 forudsætter at der er IIS 6.0 og Microsoft .NET Framework 3.5 eller højere på platformen.

Det er et krav at WS'en er bygget i .Net 3.5 SP1 samt IIS 7.0 (dog IIS 6.0 ved anvendelse af Windows Server 2003).

Det er i øvrigt også et krav at web service'n er bygget i WCF.

Både ASP.NET og WCF skal således være registreret i IIS.

.NET Framework 3.5 SP1

Det er et krav at anvende mindst .NET Framework 3.5 SP1. Windows Identity Foundation kræver .NET Framework 3.5 SP1.

Certifikaterne er din største fejlkilde

Når man ser bort fra deciderede kodefejl i selve WS'en og WS-klienten, så er brugen og konfigureringen af certifikater den absolut største kilde til fejl i forbindelse med claims-enabling af en løsning.

Derfor anbefales det på det kraftigste at at udviklere såvel som infrastrukturfolk får "Appendix D: Digital Certificates" (http://msdn.microsoft.com/en-us/library/ff359106.aspx) som pligtlæsning, idet langt størsteparten af de fejl og problemer, der forekommer i relation til claims-enabling, viser sig at udspringe i og omkring certifikaterne.

Tilkoblingen af en .Net-baseret WS

Sikkerheden for Web Services på DMPs Identify STS-løsning er baseret på følgende standarder:

- WS-Trust 1.3
- Message-baseret sikkerhed (dvs. HTTP-transport med message-baseret kryptering og signering)
- SAML 1.1 eller SAML 2.0 tokens. Det skal aftales i forvejen med DMP, hvilken type tokens, der anvendes.
- Der anvendes symmetriske nøgler til kryptering
- Key size er 256
- Algorithm suite er Basic256
- Security Header Layout er Lax
- Der kan autentificeres med Identify STS'en med User Name. Brugernavn og adgangskode skal mappe op mod en bruger oprettet i Danmarks Miljøportals administratorløsning
- Der kan autentificeres mod Identify STS'en med x509. X509 certifikatet skal udstedes af DMP og mappe op mod en bruger oprettet i Danmarks Miljøportals administratorløsning
- Der anvendes ikke service certificate negotiation mod Identify STS'en
- Der er frit valg om der anvendes service certificate negotation mod Web Services
- Der er frit valg om der anvendes Secure Conversation mod Web Services.

End points for DMP's STS er angivet nedenfor:

User name authentication

- http://log-in.miljoeportal.dk/runtime/services/trust/14/username (produktionsmiljø)
- http://log-in.test.miljoeportal.dk/runtime/services/trust/14/username (testmiljø)

X509 authentication

- http://log-in.miljoeportal.dk/runtime/services/trust/14/certificate (produktionsmiljø)
- http://log-in.test.miljoeportal.dk/runtime/services/trust/14/certificate (testmiljø)

MEX

- https://log-in.miljoeportal.dk/runtime/services/trust/mex (produktionsmiljø)
- https://log-in.test.miljoeportal.dk/runtime/services/trust/mex (testmiljø)

Bemærk at samples'ne som udgangspunkt er bygget til brug med testsystemet. De vil dog naturligvis kunne tilpasses til at bruge produktionssystemet blot ved at udskifte URL'erne (og evt. certifikater) på end points'ne.

Det anbefales så vidt muligt, at udviklerne tager udgangspunkt i de medfølgende samples, når der oprettes nye WS'er. De mange opsætningsmuligheder vil ellers hurtigt kunne risikere at gøre det til en uoverskuelig opgave at konfigurere alt det ovenstående i hånden.

Sådan kobles WS'en op mod DMP

Der er kun en enkelt fysisk forudsætning for tilkoblingen af en .Net-baseret web service. Der skal foreligge et X.509-certifikat fra en offentlig CA. Dette X.509-certifikat skal benyttes til encryption på WS'en.

Hver enkelt web service, der kobles op mod DMPs Safewhere*Identify STS skal således være udstyret med et sådant unikt certifikat.

Hvis der benyttes x509 til også at autentificere mod Safewhere*Identify kræver det dog tillige at der er udstedt et certifikat fra DMPs egen PKI.

Trin 1: Anskaffelse af det nødvendige certifikat

Der skal erhverves et certifikat til hver WS:

- Et certifikat til kryptering af beskeder mellem DMP og web servicen. Dette certifikat er ikke at forveksle med SSL certifikatet, idet certifikatet benyttes til at gøre det muligt for STS'en at kryptere beskeder, som kun servicen kan læse.

Som nævnt i den overordnede vejledning ("Vejledning til fagsystemejere omkring forløbet for tilkobling af en applikation") skal encryption certifikatet overleveres til DMP uden den private nøgle. Dvs. DMP skal have certifikatets public key-del.

Applikationens krypteringscertifikat

Krypteringscertfikatet skal erhverves hos en offentlig og anerkendt CA. Vi anbefaler især http://www.godaddy.com, da de er godkendte i alle større browsere og er markant billigere end de store spillere såsom Verisign og Thawte. Vi anbefaler tilsvarende at dette erhverves som et almindeligt SSL-certifikat.

Det skal som nævnt ikke bruges til SSL, men SSL-certifikaterne er generelt billigst, og de dækker de behov et certifikat skal opfylde for at kunne benyttes som krypteringscertifikat. Modsat et certifikat der benyttes til SSL-trafik, så skal dette certifikat ikke være udstedt til et bestemt common name. Common name er ligegyldigt for krypteringscertifikatet. Vi anbefaler dog stadig, at der vælges et sigende common name for certifikatet (f.eks. CN=<applikationsnavn>-encryption.<organisationsnavn>.dk).

Trin 2: Konfiguration af webapplikations web.config-fil

Webapplikationens web.config-fil skal tilpasses på en række punkter før .NET-applikationen fungerer efter hensigten. Appendiks B rummer en gennemgang af, hvor web.config-filen skal tilpasses.

Information omkring sikkerhedsrelateret logning

Claims er konsistente og valide repræsentationer af brugeren, der er logget ind via STS'en, som i vort tilfælde er DMP's føderale brugerstyring. Claims overføres til applikationen ved hjælp af sikre offentlige standarder, der garanterer for at overleveringen af claims til applikationen er sikker og pålidelig.

Hvis applikationen benytter sig af sikkerhedsrelateret logning kan brugerens claims således fint benyttes i denne sammenhæng. Der er med andre ord ikke nogen forskel på logning i et føderalt scenarie relativt til et traditionelt brugerlog-in. Det er dog naturligvis vitalt at man tilsikrer at logningen som minimum indbefatter de claims, der sikrer at der kan ske en entydig identifikation af

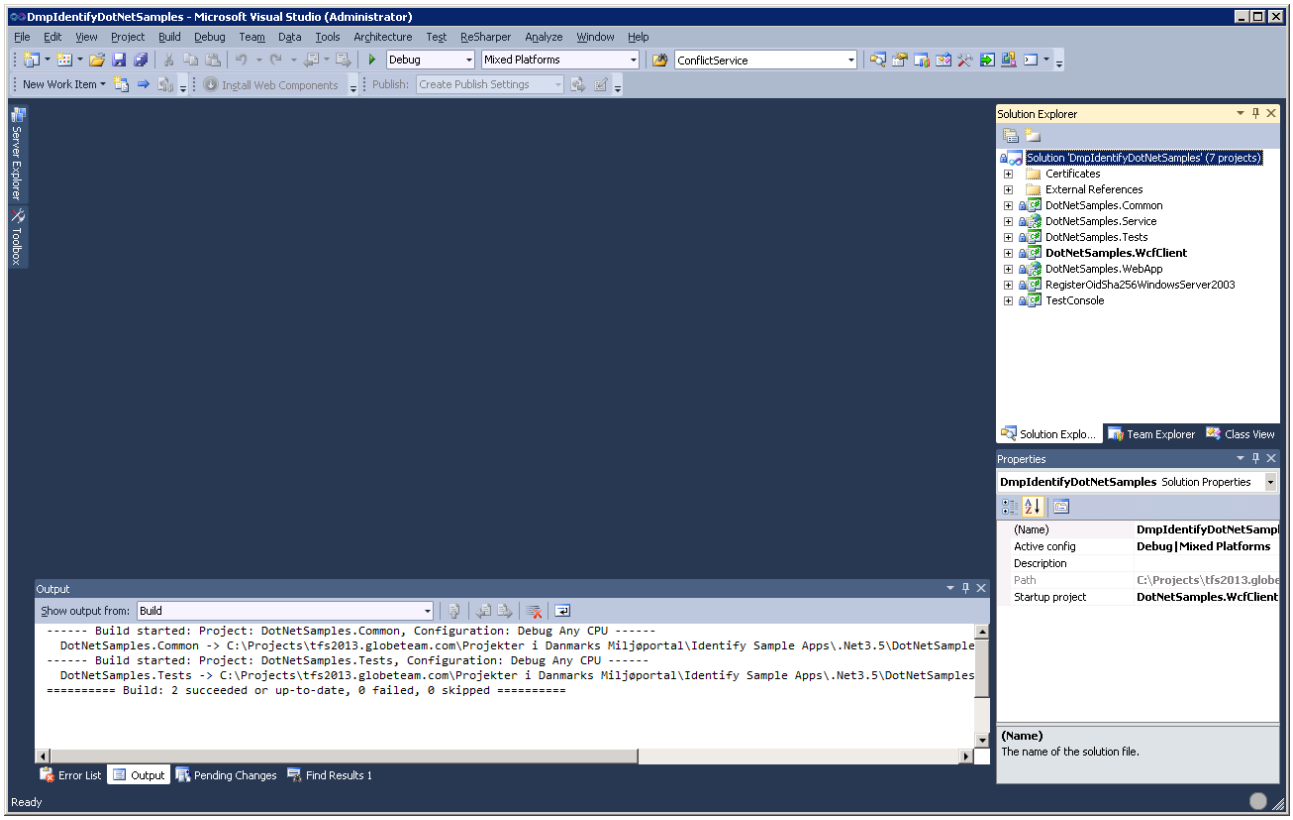
brugeren til et hvilke som helst senere-kommende tidspunkt (dvs. logningen skal indbefatte en unik identifier såvel som de nødvendige informationer, der gør det muligt at finde tilbage til hvem denne unikke bruger var).

Step-by-step eksempel på tilkoblingen af de .NET-baserede samples

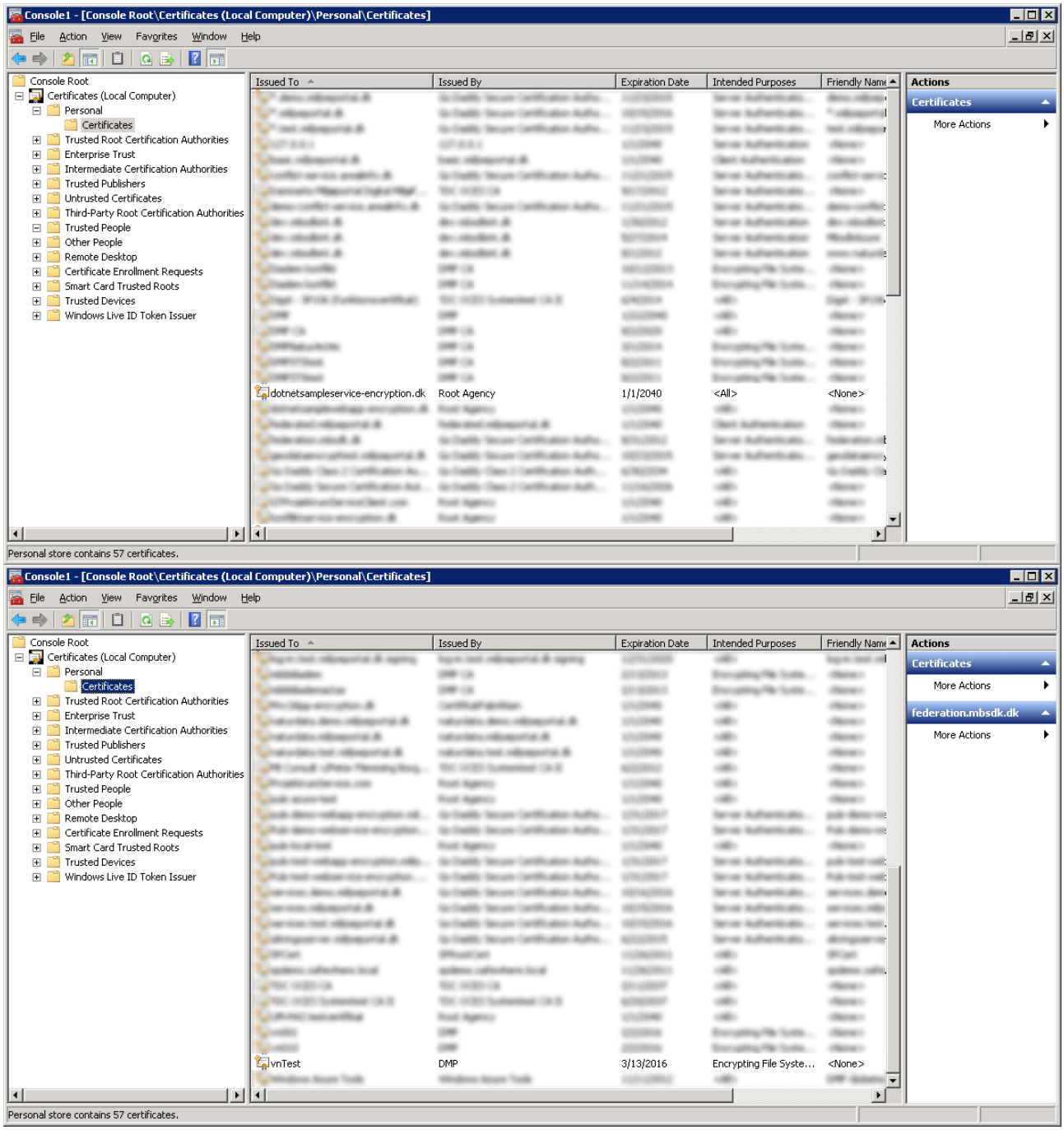
Følgende step-by-step guide tager udgangspunkt i de medfølgende samples i solution'en "DmpIdentifyDotNetSamples.sln" og beskriver, hvordan du får sample .NET web servicen op at køre lokalt og får kaldt servicen via de medfølgende unit tests.

Når disse trin er udført vil du have en kørende .NET-baseret WS samt en unit test (dvs. i al praksis en minimal WS-klient), der rekvirerer et token fra DMPs Identify STS og påhæfter det et kald til WS'en.

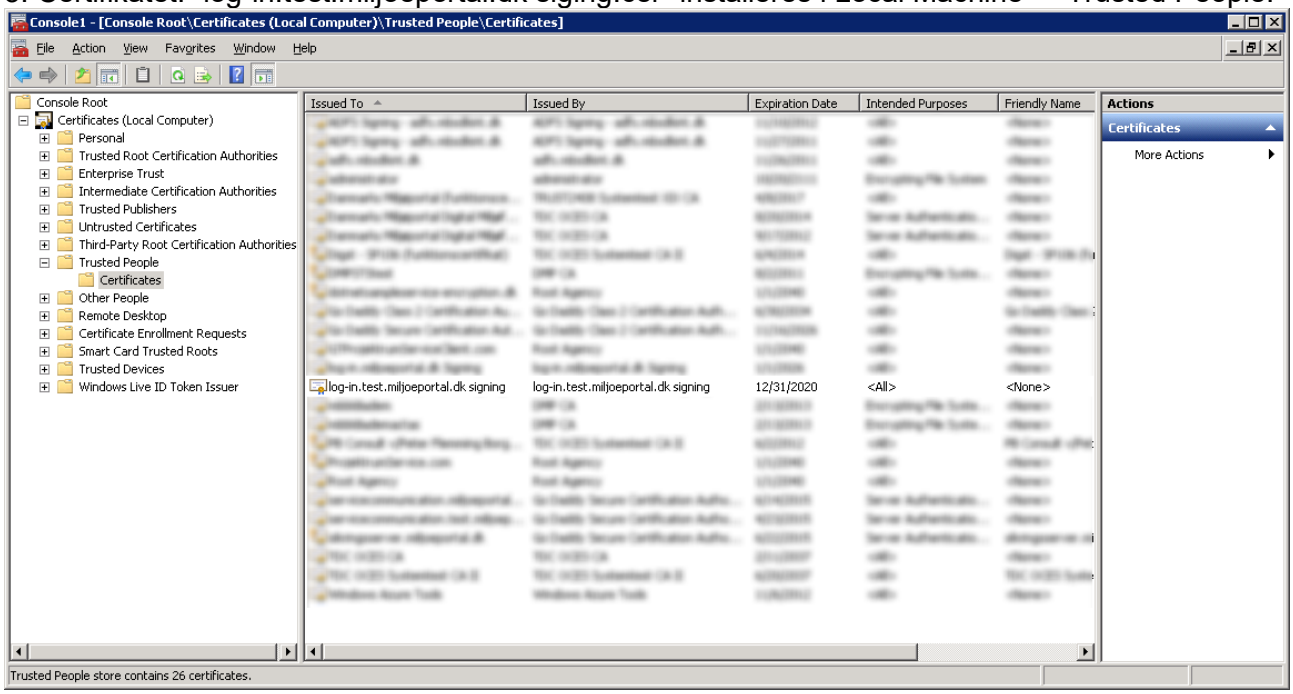
1. Åbn solution'en i Visual Studio 2010.



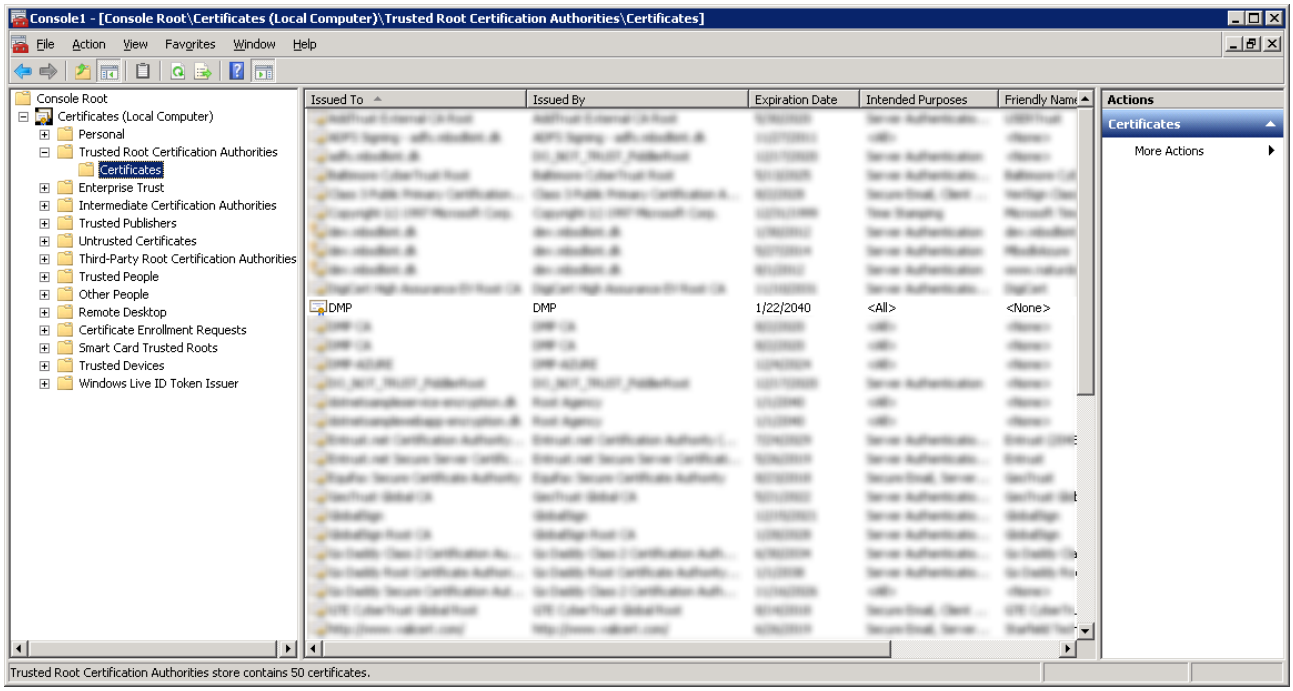
2. Certifikaterne “dotnetsampleservice-encryption.dk.pfx” og “vnTest.pfx” installeres i Local Machine -> Personal.



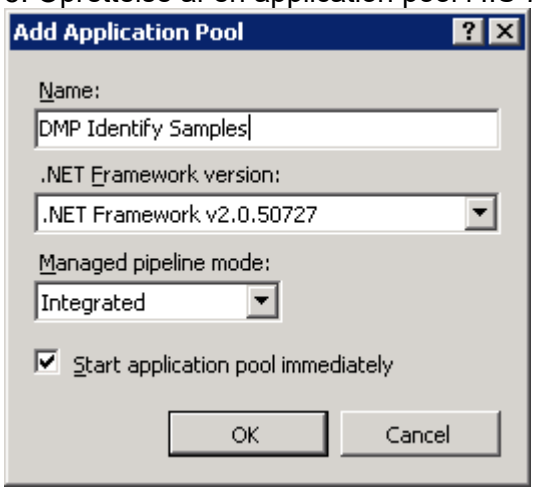
3. Certifikatet: “log-in.test.miljoeportal.dk siging.cer” installeres i Local Machine -> Trusted People.



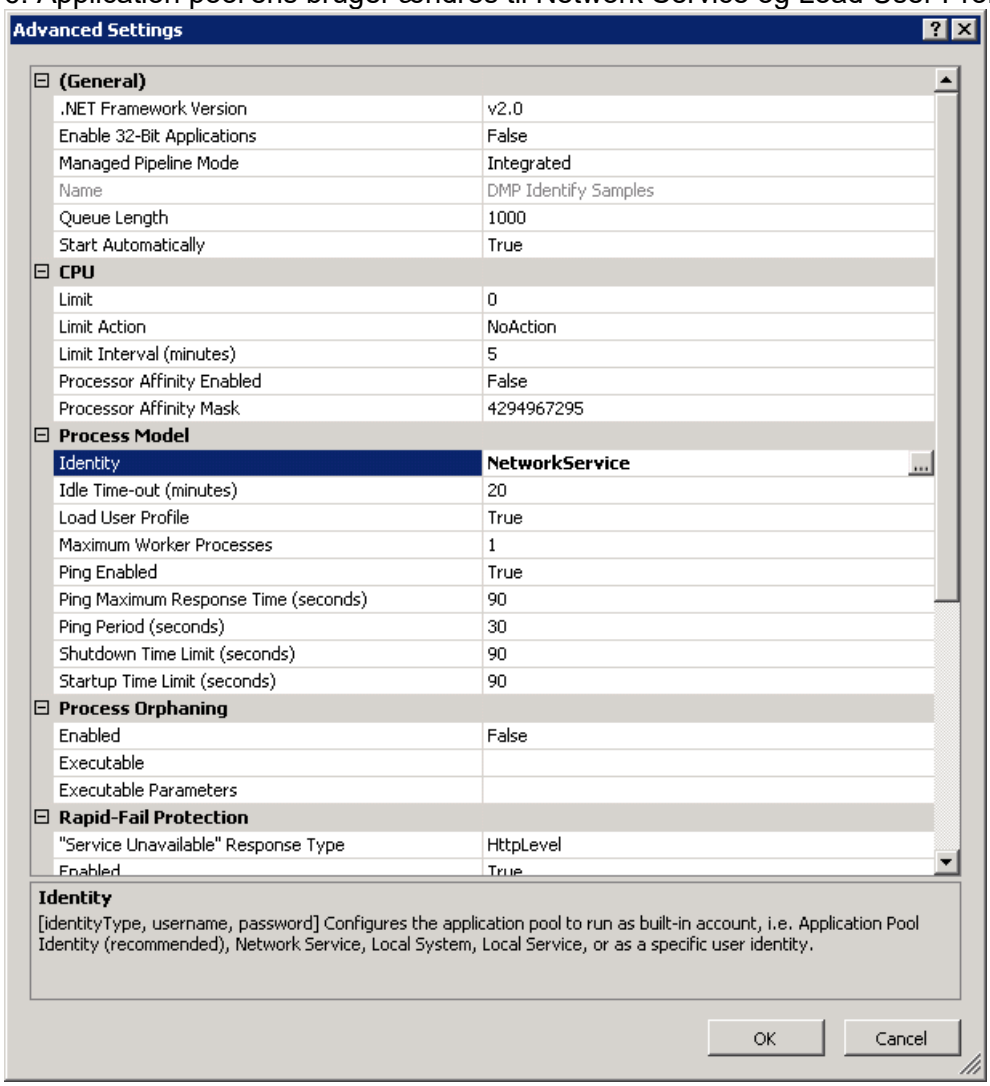
4. Certifikaterne ”DMP CA.cer” installeres i Local Computer -> Trusted Root Certification Authorities.



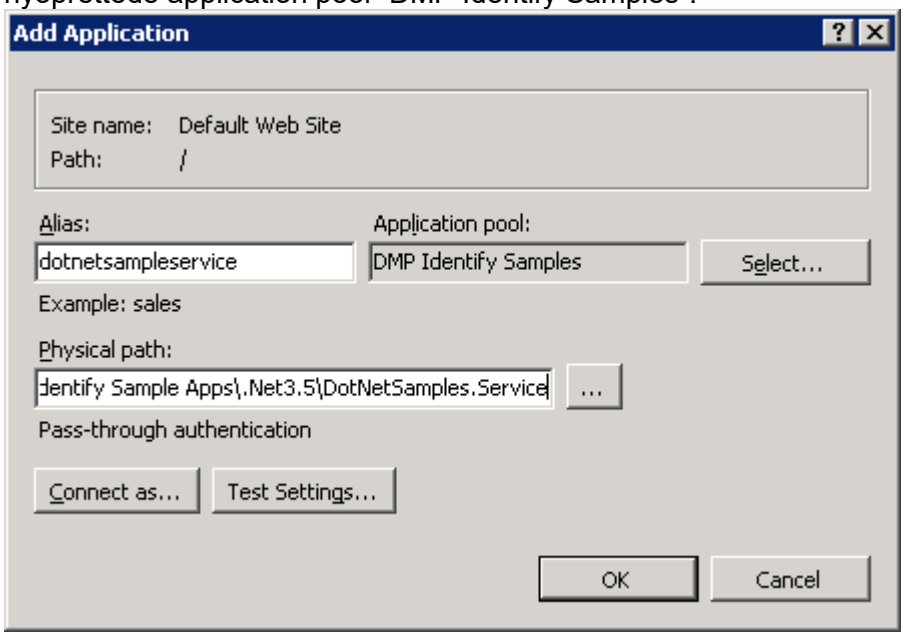
5. Oprettelse af en application pool i IIS 7 ved navn ”DMP Identify Samples”.



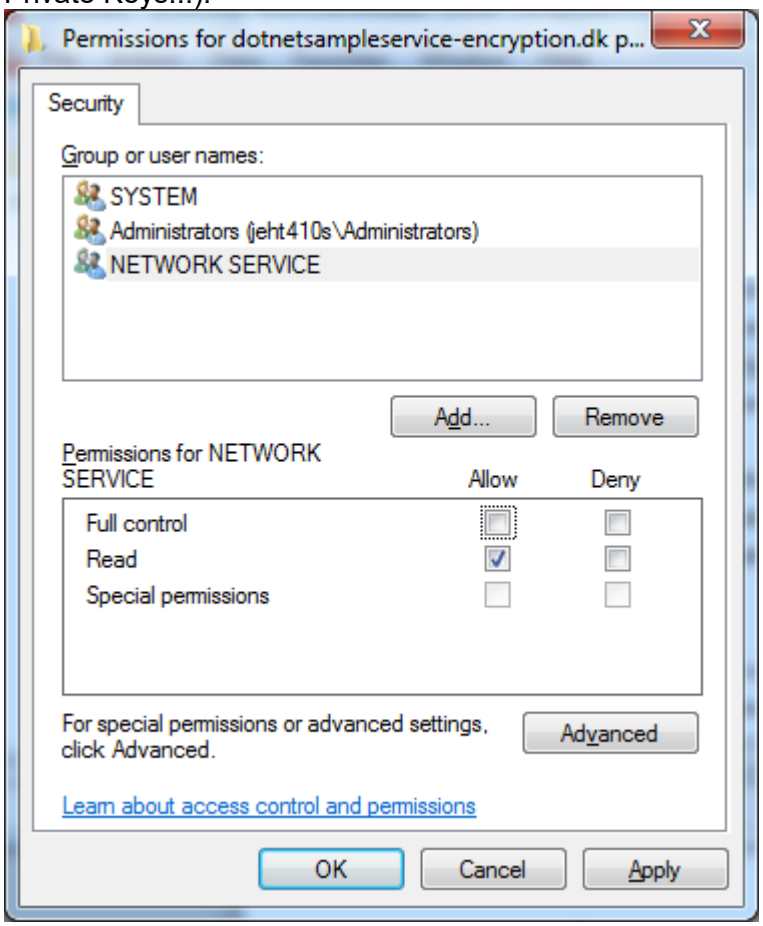
6. Application pool’ens bruger ændres til Network Service og Load User Profile sættes til True.



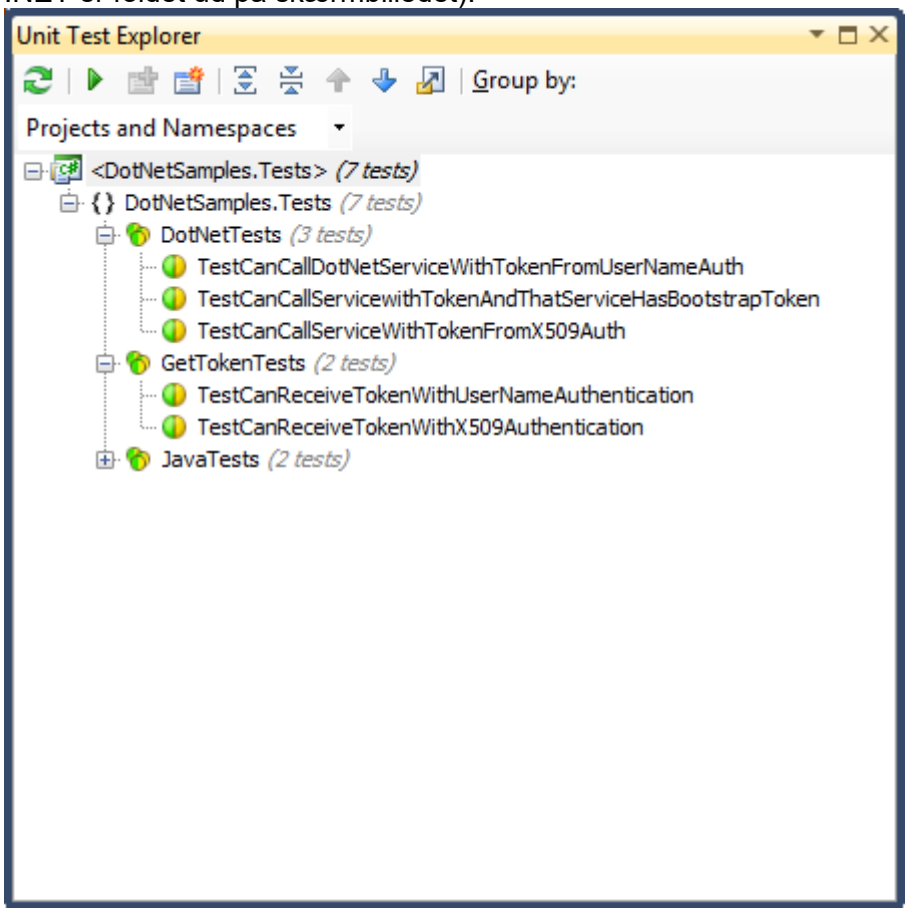
7. Der oprettes en applikation i IIS 7 ved navn ”dotnetsampleservice”. Denne skal pege ned på mappen ”DotNetSamples.Service” i de medfølgende samples. Applikationen tildeles den nyoprettede application pool ”DMP Identify Samples”.



8. Network Service tildeles Read-rettighed til private key på dotnetsampleservice-encryption.dk-certifikatet (højreklik på certifikatet under Local Machine -> Personal og vælg All Tasks -> Manage Private Keys...).



9. Solution'en kompileres og de medfølgende .Net unit tests for .NET køres (bemærk, unit tests for .NET er foldet ud på skærmbilledet).



Appendiks A. Tracing

Tracing er et must have-værktøj, når man konfigurerer .NET services baseret på WCF. Med tracing slået til på servicen vil du således meget hurtigere kunne opdage og fejlfinde en evt. fejlkonfiguration af servicen.

Tracing tager form af en .e2e-fil.

Du kan slå tracing til ved at indkommentere følgende sektion i Web.config på servicen:

```
<!-- Enable this section to enable system.servicemodel diagnostics
    <diagnostics>
      <messageLogging maxMessagesToLog="30000" logEntireMessage="true"
logMessagesAtServiceLevel="true" logMalformedMessages="true"
logMessagesAtTransportLevel="true">
      </messageLogging>
    </diagnostics>
    -->
```

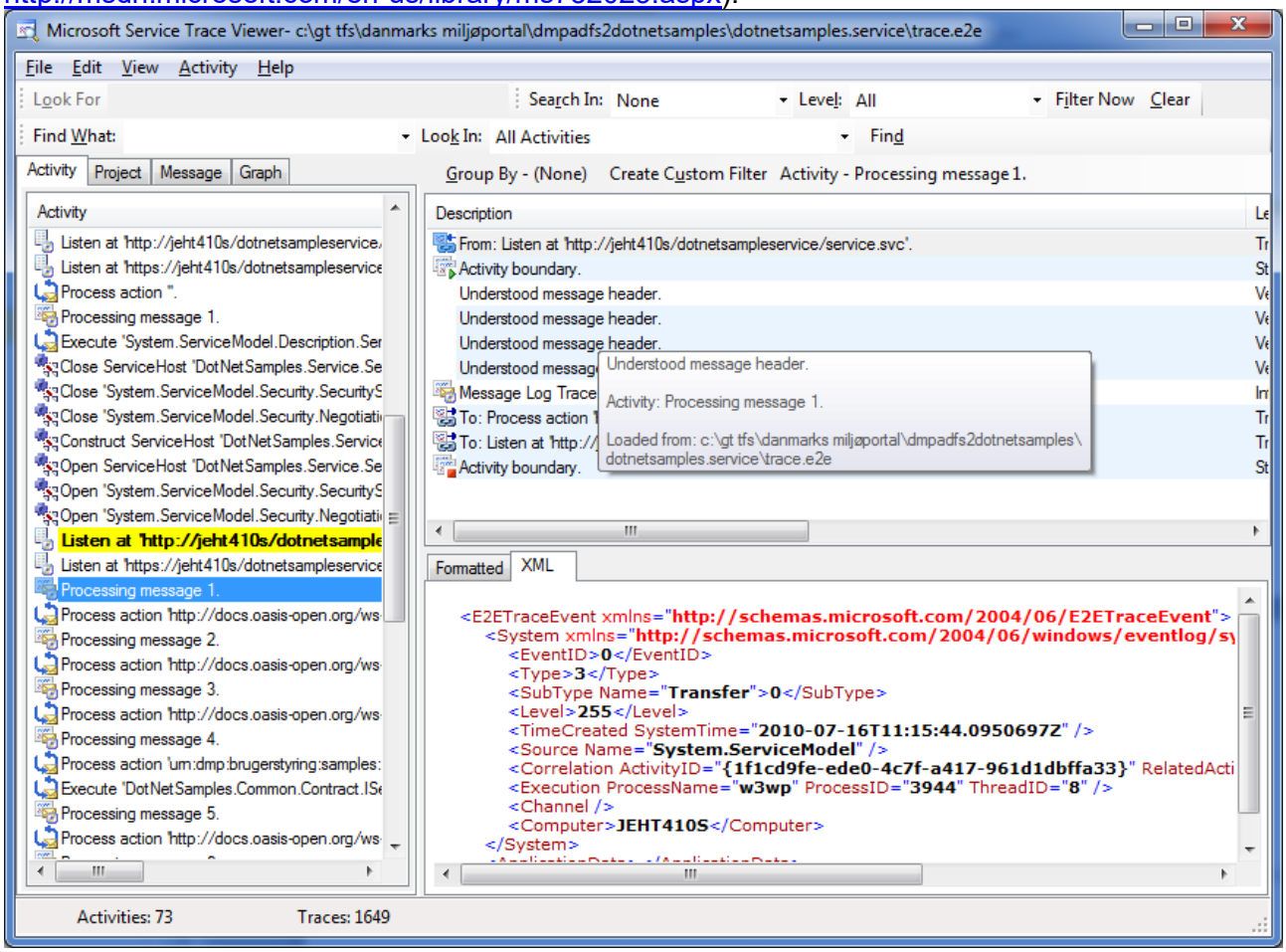
Og følgende sektion:

```
<!-- Enable this section to enable full tracing
  <system.diagnostics>
    <sources>
      <source name="System.IdentityModel" switchValue="Verbose"
logKnownPii="true">
        <listeners>
          <add name="xml"/>
        </listeners>
      </source>
      <source name="System.ServiceModel.MessageLogging">
        <listeners>
          <add name="xml"/>
        </listeners>
      </source>
      <source name="System.ServiceModel" switchValue="Verbose, ActivityTracing"
propagateActivity="true">
        <listeners>
          <add name="xml"/>
        </listeners>
      </source>
      <source name="Microsoft.IdentityModel" switchValue="Verbose">
        <listeners>
          <add name="xml"/>
        </listeners>
      </source>
      <source name="Domstolene.JFS.Brugerstyring" switchValue="Verbose">
        <listeners>
          <add name="xml"/>
        </listeners>
      </source>
    </sources>
    <sharedListeners>
```

```
      <add name="xml" type="System.Diagnostics.XmlWriterTraceListener"
initializeData="trace.e2e"/>
    </sharedListeners>
    <trace autoflush="true"/>
  </system.diagnostics>
  -->
```

Bemærk, at dette først fungerer, når application pool-brugeren er blevet tildelt Write-adgang til den mappe, som servicen kører i.

Du kan fx åbne trace.e2e-filen i svctraceviewer.exe (som er tilgængelig på http://msdn.microsoft.com/en-us/library/ms732023.aspx):



Appendiks B. Beskrivelse af WS'ens web.config-fil

Dette appendix indeholder en uddybning af indholdet af WS'ens web.config-fil.

Bemærk at sektioner der ikke skal ændres for at bygge en ny service er udeladt af hensyn til overblikket.

```
<?xml version="1.0"?>
<configuration>

  <microsoft.identityModel>
    <service saveBootstrapTokens="true">
      <audienceUris>
        <!-- Den præcise URL servicen er
registreret på -->
        <add
value="http://localhost/dotnetsampleservice/se
rvice.svc"/>
      </audienceUris>
  </microsoft.identityModel>

  <system.serviceModel>

    <services>
      <service
behaviorConfiguration="MyServiceBehavior"
name="DotNetSamples.Service.ServiceImpl">
        <endpoint address=""

binding="ws2007FederationHttpBinding"

bindingConfiguration="MyServiceBinding"
                  contract="DotNetSamples.Common.Contract.IServiceContract">
          <identity>
            <dns value="dotnetsampleservice-encryption.dk"/>
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
      </service>
    </services>

    <behaviors>
      <serviceBehaviors>
        <behavior name="MyServiceBehavior">
```

1) Der angives, at servicen skal opbevare token'et i hukommelsen når den kaldes, så det kan sendes videre til andre services via ActAs. Denne værdi kan udelades hvis servicen ikke laver act as
2) Der angives servicens præcise URL i audienceUris

Her angives din services datakontrakt og entry point

```
 <serviceCredentials>
            <!-- Identifikation af servicens encryption cert -->
            <serviceCertificate
storeLocation="LocalMachine"
                                    storeName="My"

x509FindType="FindBySubjectName"

findValue="dotnetsampleservice-encryption.dk"/>

          </serviceCredentials>
        </behavior>
      </serviceBehaviors>
    </behaviors>

      </system.serviceModel>


</configuration>
```

Her angives servicens encryption certifikat.

Appendiks C. Brug af ActAs til at kalde på vegne af en bruger

ActAs-teknologien er en del af WS-Trust specifikation.

ActAs gør det muligt at forespørge på et token til et servicekald, hvor man bruger et eksisterende token til at veksle til det nye token.

Det primære scenarie for ActAs er, når:

1) En klient kalder service A med et token.
2) Service A ønsker at kalde en service B og bibeholde brugerens identitet mod service B. Service A kan dog ikke bruge det samme token, da dette er udstedt udelukkende til brug med service A.
3) Service A forespørger i stedet på et nyt token til service B , hvor token'et til Service A sendes med som ”ActAs”

Understøttelse af ActAs kræver først og fremmest, at DMP har åbnet op for at netop din service må lave ActAs samt angivet de services, den må bruge det overfor.

Der skal altså laves en aftale med DMP om, at din service gerne må lave ActAs samt hvilke services, der gerne må udføres laves ActAs til. Når det er på plads er det blot nødvendigt at udføre en mindre ændring i koden, der laver forespørgslen for token’et.

Koden der snupper et token ser sådan her ud (pseudo):

```
// Lav channel
var channel = (WSTrustChannel) channelFactory.CreateChannel();

...

// Send request (RST) og modtag response (RSTR)
var securityToken = channel.Issue(requestSecurityToken);

return securityToken;

...
CloseChannel(channel);
```

For at requeste et token med ActAs bruges følgende samme kode med med den angivne mindre ændring:

```
// Lav channel
var channel = (WSTrustChannel) channelFactory.
   CreateChannelActingAs(securityTokenSomSkalBrugesTilActAs);

...

// Send request (RST) og modtag response (RSTR)
var securityToken = channel.Issue(requestSecurityToken);

return securityToken;

...
CloseChannel(channel);
```

Forskellen er altså blot en enkelt kodelinje. Selve kaldet til servicen foregår præcis som var det et token forespurgt på ”almindelig” vis.

I samples’ne er der tilføjet en operation på .NET WS'en, der viser hvordan man får fat i det aktuelle token, der er sendt til servicen. Dette token har man altså "ved hånden", når der skal bruges ActAs:

```
/// <summary>
/// Operation der verificerer at bootstrap token'et er tilgængeligt
/// </summary>
public void AssertBootstrapTokenIsAvailable()
{
   var identity = GetClaimsIdentity();

      var token = identity.BootstrapToken;

      if (token == null)
           throw new ApplicationException("Bootstrap token cannot be null");
}
```

Windows Identity Foundation “husker” som hovedregel ikke token’et, så det kan tilgås i servicen som ovenfor. For at token'et skal være tilgængeligt skal der udføres følgende ændring i web.config-filen:

```
<microsoft.identityModel>
   <service saveBootstrapTokens="true">
          ...
   </service>
</microsoft.identityModel>
```

Appendiks D. Web Services føderation - login mod organisationens lokale AD FS 2.0

Web services føderation omhandler i al sin enkelthed, at login'et mod DMP sker med et token der er rekvireret hos en partnerorganisation frem for at der anvendes brugernavn/adgangskode eller x509 authentication direkte mod DMP.

Den primære forudsætning for at bruge web services føderation, er at den kalder der anvender web servicen er logget på domænet i partnerorganisationens domæne. Et fødereret web services login består af følgende trin.

1) Klient henter "lokalt" token fra partnerorganisationens AD FS 2.0-server
2) Klient veksler det "lokale" token til et DMP token på DMPs Identify STS server
3) Klient kalder service med DMP token
4) Service svarer klient

Afvigelsen ift. et direkte login mod DMP ligger således i punkt 1 og 2 hvor klienten bruger et lokalt token som kreditiver når DMP token'et skal trækkes.

I de medfølgende samples ligger der under unit tests en kodestump der kan hente et lokalt token fra en AD FS 2.0 server. Dette svarer altså til punkt 1 i ovenstående punktopstilling.

```
public void TestCanGetLocalIdPToken()
{
     // Indsæt adresse på lokal AD FS 2.0
     // Bemærk, dette skal erstattes med din lokale AD FS 2.0 server
     // og denne AD FS 2.0 server skal have end pointet
     // "adfs/services/trust/13/windows" enablet
     var stsAddress = new EndpointAddress(
     new Uri("http://adfs.c0d3.dk/runtime/services/trust/13/windows"),
          EndpointIdentity.CreateDnsIdentity(
          "http://adfs.c0d3.dk/runtime/services/trust"));

     // Det end point på STS'en der modtager token'et fra den lokale IdP
     var dmpIssuedTokenAddress = Constants.StsAddressIssuedToken;

     // Hent lokalt security token vha. Windows Authentication,
     // da vi er logget på domænet med den lokale IdP
     var localToken = WsTrustClient.RequestSecurityTokenWithWindowsAuth(
          stsAddress,
          dmpIssuedTokenAddress);

     Assert.That(localToken, Is.Not.Null);
}
```

Med det lokale token i hånden kan klienten veksle token'et til et DMP token (punkt 2), som vist her. Bemærk, at koden for at hente det lokale token er vist sammen med koden der veksler det til et DMP token for overblikkets skyld.

```
public void TestCanExchangeLocalIdPTokenForToken()
{
     // Indsæt adresse på lokal AD FS 2.0
     var stsAddress = new EndpointAddress(
     new Uri("http://adfs.c0d3.dk/runtime/services/trust/13/windows"),
     EndpointIdentity.CreateDnsIdentity(
          "http://adfs.c0d3.dk/runtime/services/trust"));

     // Det end point på STS'en der modtager token'et fra den lokale IdP
     var dmpIssuedTokenAddress = Constants.StsAddressIssuedToken;

     // Hent lokalt security token vha. Windows Authentication,
     // da vi er logget på domænet med den lokale IdP
     var localToken = WsTrustClient.RequestSecurityTokenWithWindowsAuth(
          stsAddress,
          dmpIssuedTokenAddress);

     // Udveksl token'et til et DMP token, som efterfølgende kan bruges til et
     // servicekald
     var securityToken = WsTrustClient.RequestSecurityTokenWithIssuedTokenAuth(
          dmpIssuedTokenAddress,
          Constants.ServiceAddress,
          stsAddress,
          localToken,
          Constants.StsCertificate);

     Assert.That(securityToken, Is.Not.Null);
}
```

Med DMP token'et i hånden kan klienten kalde en service på normal vis – præcis som hvis token'et var hentet via et direkte login mod DMP (punkt 3 og 4).

```
// Kald service
var result = TestCommon.TestCanCallServiceWithIssuedToken(securityToken);
```

Appendiks E: Brugen af WCF 3.0-baserede klienter til kald af web services på DMPs Identify STS uden brug af Windows Identity Foundation

Klienter der anvender Windows operativsystemerne tidligere end Windows Vista har ikke mulighed for at bruge Windows Identity Foundation-baseret klientkode og her må der anvendes traditionelle WCF-klienter.

Denne sektion gennemgår hvordan disse klienter laves og klienten i sample-koden gennemgås.

Selve klienten er placeret i projektet ”DotNetSamples.WcfClient”, som er en del af de medfølgende samples til .NET. Der anvendes ws2007HttpBindings og ws2007HttpBindings i klienten. Både scenariet for direkte login såvel som partnerlogin er implementeret i klienten.

Bindings til hhv. Safewhere*Identify og services konstrueres programmatisk for at have fuld control over brugen af secure conversation (.NET-baserede services) og ingen secure conversation (Java Metro-baserede services).

Følgende kodestump konstruerer en binding til en service der anvender direkte login hos DMP:

```
// Lav binding til STS'en
var stsBinding = new WS2007HttpBinding(SecurityMode.Message, false);
stsBinding.Security.Message.ClientCredentialType =
MessageCredentialType.UserName;
stsBinding.Security.Message.EstablishSecurityContext = false;
stsBinding.Security.Message.NegotiateServiceCredential = false;

// Lav binding til servicen
var serviceBinding = new
WS2007FederationHttpBinding(WSFederationHttpSecurityMode.Message, false);
serviceBinding.Security.Message.AlgorithmSuite =
SecurityAlgorithmSuite.Basic256;
serviceBinding.Security.Message.IssuedKeyType = SecurityKeyType.SymmetricKey;
serviceBinding.Security.Message.IssuedTokenType = "http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1";
serviceBinding.Security.Message.NegotiateServiceCredential = false;
serviceBinding.Security.Message.IssuerAddress = Constants.StsAddressUserName;
serviceBinding.Security.Message.IssuerBinding = stsBinding;
serviceBinding.Security.Message.IssuerMetadataAddress = new
EndpointAddress("http://log-
in.test.miljoeportal.dk/runtime/services/trust/mex");
```

Følgende kondestump konstruerer en binding til en service der anvender fødereret login hos en partnerorganisation.

```
// Lav binding til STS'en
var partnerStsBinding = new WS2007HttpBinding(SecurityMode.Message, false);
partnerStsBinding.Security.Message.ClientCredentialType =
MessageCredentialType.Windows;
partnerStsBinding.Security.Message.EstablishSecurityContext = false;
partnerStsBinding.Security.Message.NegotiateServiceCredential = true;

// Lav binding til DMP STS'en
var stsBinding = new
WS2007FederationHttpBinding(WSFederationHttpSecurityMode.Message, false);
stsBinding.Security.Message.AlgorithmSuite =
SecurityAlgorithmSuite.Basic256Sha256;
stsBinding.Security.Message.IssuedKeyType = SecurityKeyType.SymmetricKey;
stsBinding.Security.Message.IssuedTokenType = "http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1";
stsBinding.Security.Message.NegotiateServiceCredential = false;
stsBinding.Security.Message.IssuerAddress = Constants.StsAddressPartnerOrg;
stsBinding.Security.Message.IssuerBinding = partnerStsBinding;
stsBinding.Security.Message.IssuerMetadataAddress = new
EndpointAddress("http://adfs.c0d3.dk/runtime/services/trust/mex");

// Lav binding til servicen
var serviceBinding = new
WS2007FederationHttpBinding(WSFederationHttpSecurityMode.Message, false);
serviceBinding.Security.Message.AlgorithmSuite =
SecurityAlgorithmSuite.Basic256;
serviceBinding.Security.Message.IssuedKeyType = SecurityKeyType.SymmetricKey;
serviceBinding.Security.Message.IssuedTokenType = "http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1";
serviceBinding.Security.Message.NegotiateServiceCredential = false;
serviceBinding.Security.Message.IssuerAddress =
Constants.StsAddressIssuedToken;
serviceBinding.Security.Message.IssuerBinding =
WsTrustClient.CreateWsFederationBindingWithoutSecureConversation(stsBinding);
serviceBinding.Security.Message.IssuerMetadataAddress = new
EndpointAddress("http://log-
in.test.miljoeportal.dk/runtime/services/trust/mex");
```

Selve klienten bruger således en af de to ovenstående bindings afhængigt af om der logges på direkte hos DMP med en brugerkonto oprettet i DMP eller via partnerorg login lokalt hos en partnerorganisation.

```
// For direkte login mod DMP
var tempBinding = CreateServiceBindingDirectLogin();

// For fødereret login
//var serviceBinding = CreateServiceBindingFederatedLogin();

// Deaktiver secure conversation hvis der kaldes en Java service
//var serviceBinding =
WsTrustClient.CreateWsFederationBindingWithoutSecureConversation(tempBinding);
```

Ovenstående opretter en binding til den konkrete services der skal kaldes og der vælges enten secure conversation eller ingen secure conversation. .NET services taler fint secure conversation mens der på Java Metro i skrivende stund er kompatibilitetsproblemer mellem .NET og Java. Derfor slukkes secure conversation mod Java services.

```
// Hvis der kaldes en .NET service anvendes der secure conversation
var serviceBinding = tempBinding;

var cf = new ChannelFactory<JupiterWrite>(serviceBinding, ServiceAddress);
var credentials = cf.Credentials;

if (credentials == null)
   throw new InvalidOperationException("ClientCredentials not present. Client
is not correctly configured.");

// Til DMP login
credentials.UserName.UserName = UserName;
credentials.UserName.Password = Password;

// Til fødereret login
credentials.Windows.ClientCredential.UserName = PartnerorgUserName;
credentials.Windows.ClientCredential.Password = PartnerorgPassword;
```

Der sættes credentials på channel factory'en hvor der anvendes "UserName" auth til direkte login mod DMP og "Windows" auth til login hos partnerorganisationer. Bemærk, at for det lokale login hos partnerorganisationerne er det ikke nødvendigt at angive windows auth, da brugeren allerede vil være logget på domænet.

```
// Indlæs certifikat til hhv. AD FS 2.0
var identifyServiceCertificate =
LoadX509CertificateFromConfigFile("StsIdentifyServiceCertificate");

// Indlæs certifikat til service
var serviceCertificate =
LoadX509CertificateFromConfigFile("DotNetServiceCertificate");

// Til fødereret login skal her angives service communication certifikatet for
partnerorg sts'en
```

```
var partnerOrgCertificate =
LoadX509CertificateFromConfigFile("PartnerorgAdfs20ServiceCertificateBase64Test
");

credentials.ServiceCertificate.ScopedCertificates.Add(Constants.StsAddressPartn
erOrg.Uri, partnerOrgCertificate);
```

```
// Sæt public keys på hhv. AD FS 2.0 channel og Service channel da negotiation
er slået fra
credentials.ServiceCertificate.ScopedCertificates.Add(Constants.StsAddressIssue
dToken.Uri, adfs20ServiceCertificate);
credentials.ServiceCertificate.ScopedCertificates.Add(Constants.StsAddressUserN
ame.Uri, adfs20ServiceCertificate);
credentials.ServiceCertificate.ScopedCertificates.Add(ServiceUri,
serviceCertificate);

// Deaktivering af certifikatvalidering på servicecertifikatet. Må IKKE være
aktiveret i et produktionsmiljø,
// da der her skal anvendes offentligt udstedte certifikater, som kan valideres

credentials.ServiceCertificate.Authentication.CertificateValidationMode =
X509CertificateValidationMode.None;
credentials.ServiceCertificate.Authentication.RevocationMode =
X509RevocationMode.NoCheck;

var client = cf.CreateChannel();

try
{
   // Gør noget med klienten...
   client.stuff();

   // Luk channel
   ((ICommunicationObject)client).Close();
}
catch(Exception ex)
{
   ((ICommunicationObject)client).Abort();
   throw ex;
}
```