Teknisk beskrivning av app-växling

Nedan stycke beskriver vad som krävs för app-växling i det nya underskriftsflödet. Notera dock att detta flöde är ett komplement till den underskrift som idag kan göras i Internet Explorer via NetID. Åtgärder som anges nedan är därmed enbart krav för de som ämnar nyttja detta nya underskriftsflöde för att tillgängliggöra Webcert i fler webbläsare.

Bakgrund

Den nya underskriftfunktionen som kommer bli tillgänglig i Webcert kommer använda s.k. *app-växling* för att öppna det program där användaren anger sin PIN-kod. Detta är en ny autentiseringsklient som tagits fram inom SITHS. Vidare i texten kommer denna benämnas `SITHS eID Autentieringsklient`

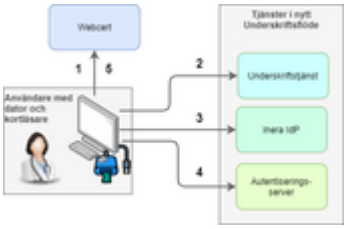
Om man ser till användarflödet så är de flesta redan vana vid denna typ av *app-växling,* då det vanligtvis sker på samma sätt när man använder *BankID* eller *Mobilt BankID*. Dvs användarens webbläsare är ansvarig för att öppna det program där PIN-koden skall anges.

För att detta skall fungera ur ett tekniskt perspektiv är det således ett krav att webbläsaren har rätt behörighet för att utföra denna åtgärd, dvs öppna ett annat program.

För de som använder Webcert fristående, eller där integrationen sker genom att öppna Webcert i webbläsarens normalläge är detta vanligtvis inte ett problem, utan det är främst de som integrerar genom att öppna Webcert i en *inbäddad webbläsare* som måste kontrollera funktionalitet och rättigheter.

Användarflöde

Ur ett användarperspektiv så är det följande som sker.



1:

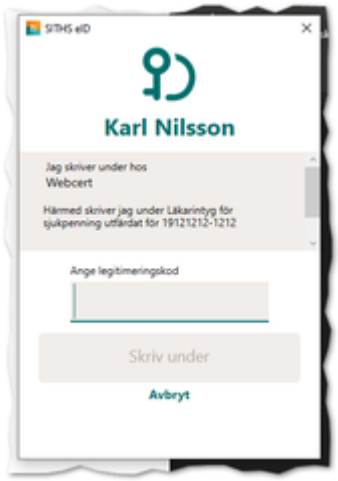
Användaren har loggat in i Webcert med SITHS-kort och har skrivit ett intyg som är klart att signera.

Användaren klickar på `Signera intyget`

2:

Webcert dirigerar användarens webbläsare till Underskriftstjänsten med information om den data som ska signeras.

Ingen dialog visas för användaren (webbläsaren visar vit bakgrund).

3: Underskriftstjänsten dirigerar användarens webbläsare till Ineras IdP med begäran om att legitimera användaren för underskrift. IdP'n visar en dialog och försöker samtidigt starta `SITHS eID Autentiseringsklient`på användarens dator. Om Autentiseringsklienten inte startar automatiskt kan användaren klicka `TBD`för att försöka starta den manuellt.	
4: `SITHS eID Autentiseringsklient`startas och ansluter till Autentiseringsservern. En dialog visas för användaren med information om vad som ska signeras. Användaren anger sin **PIN-kod för legitimering (PIN1)** och klickar `Skriv under`. Autentiseringsklienten slutför legitimeringen av användaren och skickar informationen till Autentiseringsservern.	

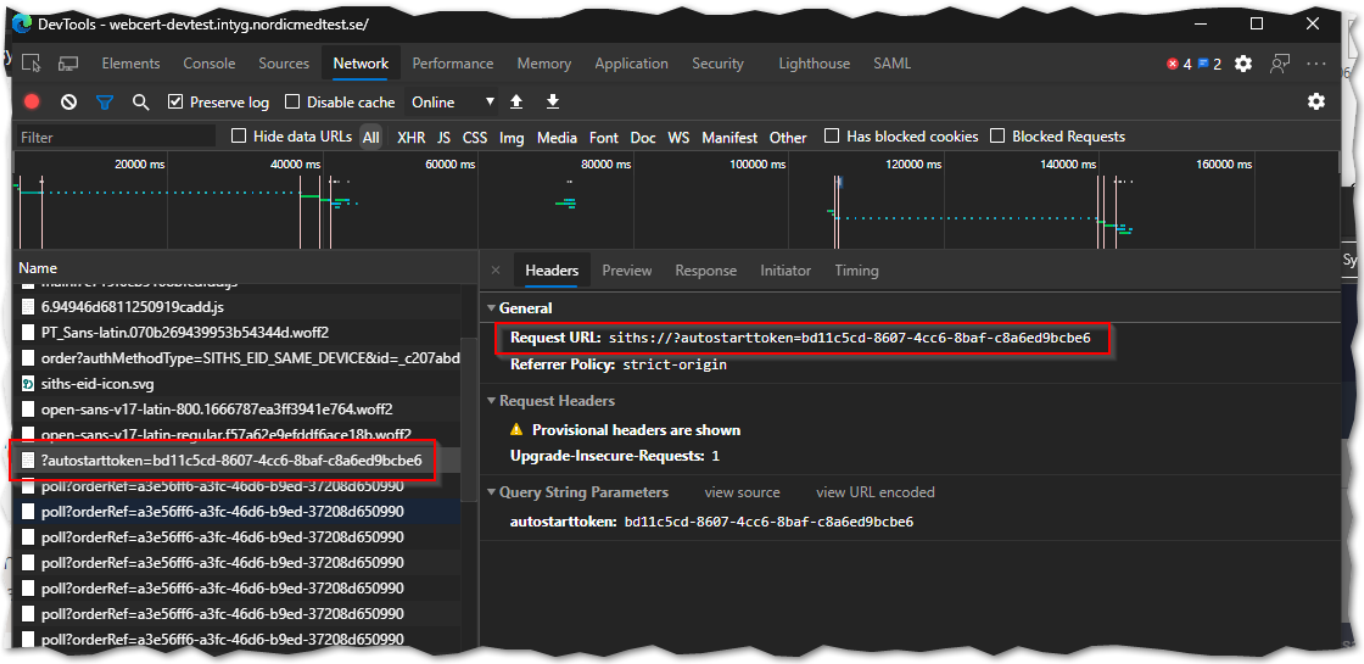
	5: Användarens webbläsare dirigeras tillbaka till Webcert via Underskriftstjänsten. Webcert visar dialog med signerat intyg, eller felmeddelande om signering misslyckades.	

Teknisk beskrivning av app-växling

För att steg 3 och 4 i ovan flöde skall fungera så krävs det som sagt att webbläsaren har rättighet att starta det aktuella programmet.

Programmet startas genom att webbläsaren försöker öppna en länk som börjar på `siths://`. Detta är ett eget protokoll som är unikt för `SITHS eID Autentiseringsklient`. Vid installation så registrerar programmet att när någon försöker öppna en länk som börjar med `siths://`så är det just detta program som skall startas.

Om man använder Devtools i webbläsaren så kan man se hur länken anropas:



Det är i detta steg som det kan ta stopp i en inbäddad webbläsare. Dessa kan ha begränsade rättigheter vilka hindrar de från att öppna externa program. På grund av detta behöver de som använder inbäddade webbläsare och som tänker använda det nya underskriftsflödet, kontrollera med sina leverantörer att detta steg kommer att fungera.

Exempelkod för inbäddad webbläsare

Följande ger en proof-of-concept på hur man kan tillåta SITHS-protokollet att exekvera/app-växla vid användning av `CefSharp`. Givetvis skiljer sig implementationen mellan alla olika journalsystem, så nedan exempel är enbart tänkt som en fingervisning på hur det kan göras i just denna implementationen av inbäddad webbläsare.

Det viktigaste i exemplet är den anpassade `ResourceRequestHandler` som tillåter protokollet `siths` men inget annat protokoll. Denna funktion anropas för alla protokoll som är okända, innan de tillåts.

```
class CustomResourceRequestHandler : ResourceRequestHandler
{
    protected override bool OnProtocolExecution(IWebBrowser
chromiumWebBrowser, IBrowser browser, IFrame frame, IRequest request)
    {
        Uri uri = new Uri(request.Url);
        if (uri.Scheme == "siths")
        {
            return true;
        }
        return false;
    }
}
// Här skapas en factory-klass som returnerar custom implementation av
ResourceRequestHandlerFactory.
public class CustomResourceRequestHandlerFactory :
IResourceRequestHandlerFactory
{
    public bool HasHandlers => true;    public IResourceRequestHandler
GetResourceRequestHandler(IWebBrowser chromiumWebBrowser,
    IBrowser browser, IFrame frame, IRequest request, bool
isNavigation, bool isDownload, string requestInitiator, ref bool
disableDefaultHandling)
    {
        return new CustomResourceRequestHandler();
    }
}

// och slutligen pekas denna factory-klass ut för browser-instansen.
var browser = new ChromiumWebBrowser("https://enrollment.preacctest.
ineratest.org")
{
    ResourceRequestHandlerFactory = new
CustomResourceRequestHandlerFactory()
};
panel.Controls.Add(browser);
```