NTNU
Norwegian University of
Science and Technology

A SAML 2.0 Authentication Middleware for ASP.NET Core

Jessy Kyalo Musyimi

Master in Information Security
Submission date: June 2018
Supervisor: Basel Katt, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Preface

This report represents the work done in partial fulfillment of my Master in Information Security, at the Norwegian University of Science and Technology (NTNU), department of Information Security and Communication Technology, in the spring semester of 2018.

The idea came when I was working for a Norwegian online pharmacy. Patient identity and prescription information was managed and protected by a SAML identity provider. The online pharmacy application had to be integrated with the SAML identity provider, in order to retrieve and show patient prescriptions on the online shop. To minimize the work required to integrate with a SAML identity provider, I searched for an open source solution to no avail, and ended up buying a SAML component that costed around 6000 USD.

The reader should be familiar with the SAML framework and programming. An understanding of the C# programming language, and the .NET Core framework would be helpful. An introduction to the SAML 2.0 standard and the ASP.NET Core framework is given.

01-06-2018

Acknowledgment

To my wife Jeddidah, thank you for your love and support. Thank you for all the nights, mornings and days I was away. Above all, thanks for taking care of our daughter Joan. Thank you for being my best friend. I owe you everything.

To my family, particularly my mother Serah, thank you for your support, prayers and words of encouragement during tough times. Thank you for your unwavering support and belief in me. I would never be the person I am today without you.

I owe great thanks to my supervisor Dr. Basel Katt, for the discussions, guidance and feedback. Thank you for being available to me whenever I needed you, not forgetting how quickly you replied my emails.

J.K.M

Abstract

The modern society is becoming more and more depended on information systems to run its critical services. Public infrastructure facilities, including the health services, commercial airlines and nuclear power plants depend on functional information systems to deliver secure and quality services to the society [1].

One way of building information systems is the use of web-based Internet applications. Web applications are software programs that run on a web server, and are accessed through a web browser [2]. They are accessible from any device or computer that is connected to the Internet. Considering the sensitivity and nature of personal information web applications store and give access to this days, they have to be built with security in mind. This includes, but not only limited to an effective authentication and authorization mechanism [3]. Effective authentication in web applications can be achieved using web application authentication protocols such as SAML and others [4].

Integrating a web application with a SAML identity provider is complex and time consuming for software developers [5] [6]. It requires a deep knowledge and understanding of XML, XML signatures and x509 certificates for encryption, decryption and signing of protocol messages [7] [8].

ASP.NET Core is the new framework developed by Microsoft for implementing web applications. At the moment, there are no free, open source SAML 2.0 libraries for ASP.NET Core. This thesis looks at how the SAML 2.0 authentication framework can be implemented in ASP.NET Core based web applications. It explores a way of making SAML 2.0 implementation friendly to software developers, by creating an open source, easy to configure, reusable, and flexible SAML 2.0 based authentication middleware for ASP.NET Core.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

ACS Assertion Consumer Service. 25, 26, 37, 41, 44, 60, 65, 75

CLR Common Language Run-time. 5

FCL .NET Framework Class Library. 5

HTTP Hypertext Transfer Protocol. 1, 7

IDP Identity Provider. 23, 24, 26, 32, 37, 40, 41, 52, 55

JSON JavaScript Object Notation. 9, 32, 53, 74

LINQ Language Integrated Query. 5

SaaS Software as a Service. 2

SAML Security Assertion Markup Language. iii, 1, 6, 15–18, 21, 23, 24, 27

SAML 2.0 Security Assertion Markup Language version 2.0. iii, 3, 4, 15, 16, 27, 31, 32, 52, 53, 55, 59

SLO Single Logout. 32, 37, 75, 82

SP Service Provider. 23, 24, 26, 31, 32, 37, 52

SSO Single Sign-On. 5, 15, 16, 23, 24, 26, 81, 82

URL Uniform Resource Locator. 19

USD US dollars. 3

XML eXtensible Markup Language. iii, 2, 6

Glossary

Assertion A package of information about an identity issued by a SAML identity provider. 26

Binding Mapping from SAML request-response exchange, into the standard communication protocols like HTTP-GET, HTTP-POST and SOAP. 19

Identity Information about a person/subject. 62

Identity Provider An authority that can issues a SAML security assertion. 1, 15

Profile The combination of SAML protocols, Bindings and Assertions in order to satisfy a particular use case and enhance interoperability in applications. 23

RelayState A mechanism, whereby state data can be round tripped to an identity provider and back to the service provider or vice versa, as a parameter in the request-response exchange. 31, 38, 76

SAML Artifact A reference used for transferring authentication messages between a service provider and an identity provider. 21

SAML Protocol Defines rules that govern how an identity provider and a service provider, will handle different SAML requests and responses. 17

SAMLart A URL parameter for transferring a SAML artifact value between a service provider and an identity provider. 21, 37, 46

SAMLRequest An authentication request (samlp:AuthnRequest) or a logout request (samlp:LogoutRequest) generated by either a service provider or an identity provider. 2, 64, 77

SAMLResponse A response to an authentication (samlp:Response) or logout request (samlp:LogoutResponse). 2, 14, 65, 78

Service Provider Any entity that is in a position to accept a SAML security assertion, and use it for access control to provided services. 1, 15

1 Introduction

1.1 SAML

Security Assertion Markup Language is an open standard for exchanging authentication and authorization information between a Service Provider and an Identity Provider. The authentication information is exchanged in form of a security assertion, regarding a given identity, also known as a subject and typically a person . A service provider is any entity that is in a position to accept a security assertion, and use it for access control to provided services. An identity provider is the authority that issues the security assertion [7] [9].

1.2 ASP.NET Core

ASP.NET Core is the new framework created by Microsoft for building web applications. It is cross-platform and it runs on top of Microsoft .NET run-time, which can be installed on Windows, Mac or linux systems. This means that web applications created in ASP.NET Core can be hosted on either Windows or Linux web servers [10].

1.3 ASP.NET Core middleware

An ASP.NET Core middleware, is a software that can be plugged into a web application's HTTP pipeline, to process HTTP requests and responses. It is set up when the web application starts, has access to all the incoming HTTP requests, can perform work on them and choose to pass them to the next component/middleware in the pipeline. ASP.NET Core middleware is suited for web application's cross cutting concerns such as logging or authentication [11]. This project will utilize the ASP.NET Core middleware functionality, to create a SAML authentication handler for ASP.NET Core.

1.4 Keywords

Authentication, authorization, access control, web application security

1.5 Problem description

To software developers, SAML is hard to work with [5] [6]. It requires deep knowledge and understanding of XML, XML signatures and x509 certificates for encryption, decryption and signing of protocol messages. This makes integrating a web application with a SAML identity provider, complex and time consuming, especially if a developer has to write all from scratch [7] [12] [13].

Some identity providers have created SAML toolkits, which are tied to their solutions and therefore lacking reusability and flexibility. Other third-party companies have created proprietary SAML libraries that are quite expensive to use [14]. Different full .NET framework open source projects have tried to solve this by creating SAML libraries, that support only a subset of the SAML supported bindings, basically the HTTP redirect binding. This is due to lack of funding and support among other issues. Currently, there are no open source SAML libraries supporting ASP.NET Core or the HTTP Artifact binding [12] [7].

Clearly, some work needs to be done in order to bring SAML support into ASP.NET Core. The Danish Government, through Digitaliserdk, created an open source SAML library for the full .NET framework. The library is called OIOSAML.Net and it implements the SAML standard, in the context of the full .NET framework. The library does not support ASP.NET Core [15].

Is it therefore possible to fork this repository, port it into ASP.NET Core and, further build a full SAML authentication middleware for ASP.NET Core, that is reusable, flexible and easy to configure by software developers? One of the main challenges here will be initiating SAMLRequests, from the service provider, via the middleware and forwarding them to an identity provider. Another major challenge will be receiving and handling SAMLResponses from an identity provider, through the middleware and forwarding the result to the service provider application.

1.6 Justification, motivation and benefits

Despite the recent growth of new and developer friendly authentication standards like the OpenID Connect [8], SAML is still widely used for enterprise identity management. According to a survey by onelogin conducted in 2014, 67% of Software as a Service (SaaS) providers used SAML for SSO, while 19% were planning to implement SAML based services within the next 12 months [16].

SAML is relatively complex and time consuming for developers to implement.

A simple search in stackoverflow for SAML related questions gave over 9000 questions, of which a good number had not been answered [5].

ComponentSpace recently released a SAML 2.0 software component for ASP.NET Core. The cost for an enterprise licence with source code is barely 6000 USD with an additional 1099 USD for a subscription license [14]. Clearly, this is a lot of money for small and medium-sized enterprises who would like to use SAML as an authentication protocol in their applications.

When implementing SAML based authentication solutions, many developers are not willing to spend the time required to understand SAML or become experts. What they would like is a simple, free (open source), easy to adopt software package, which can be with a few lines of code configured to integrate with any SAML based identity provider [6] [8] [13]. With our solution, the software developer is free to focus their time and resources into developing features that deliver direct value to their organization.

1.7 Research questions

On the basis of the problem description given above, the following research questions are defined:

1. How to initiate and handle authentication requests?
2. How to receive and handle authentication responses?
3. How can the middleware, securely manage user sessions?
4. How to initiate and handle logout requests?
5. How to receive and handle logout responses?
6. How should the middleware be designed to offer, a reusable and flexible configuration, for any identity provider specific, or internal service provider runtime variables?
7. How to monitor, troubleshoot and diagnostically trace events within the middleware?

1.8 Planned contributions

The main contribution of this research is the created SAML 2.0 authentication middleware for ASP.NET Core (Saml2.Authentication.Core). This is available as a software package in github (`https://github.com/jkmu/Saml2.Authentication.Core`). The software is:

1. Open source under the Mozilla Public license [17]

2. Developer friendly (easy to adopt, configurable with a few lines of code)
3. Flexible and reusable (works to integrate with any SAML 2.0 identity provider using the HTTP Redirect binding and the HTTP Artifact binding)
4. Further work can be done to;
    - Support other SAML 2.0 bindings and profiles.
    - Enhance performance and usability.
    - Support multiple SAML 2.0 identity providers

    .

2 Related work

This chapter discusses related work and introduces the .NET and ASP.NET Core frameworks for web applications development. It explores authentication in ASP.NET Core, while introducing background materials on creating custom authentication handlers. The chapter ends with a review of Single Sign-On (SSO) in the context of the SAML 2.0 framework and its components.

2.1 Background

2.1.1 .NET Framework

The .NET framework is a software development platform developed by Microsoft. It runs only on Windows machines, providing tools and technologies needed for developing Windows, Windows phone, Windows Server, Azure and web applications. The framework provides the compile time and run-time(execution environment for a managed program) necessary for building applications in any .NET based programming language [18]. Some of the main features of the .NET run-time are; automatic memory management, type safety, delegates and lambdas, generic types, Language Integrated Query (LINQ) and async programming [19]. The .NET framework has two main components; the Common Language Run-time (CLR) and .NET Framework Class Library (FCL). The CLR provides a run-time environment and services required for running programs, while the FCL contains a library of classes, interfaces and types that gives a developer access to the underlying system functionality [20].

2.1.2 .NET Core and .NET standard

.NET Core is a cross-platform implementation (port) of the .NET framework. Applications build in .NET Core can run on Windows, macOs, linux machines including embedded and IOT devices. It is completely open source under the MIT licence. .NET Core run-time is flexible in the sense that it can be deployed together and inside the application, or it can be installed on the machine running the application. It is build on the same basis as the .NET Framework CLR [21].

.NET Standard is an interface specification, describing which features and

APIs are available in .NET implementations. Its purpose is to maintain uniformity and a standard in the .NET Base classes, giving developers the possibility of building software libraries that are flexible and reusable in all .NET implementations. Different versions of the .NET standard show how many APIs are available, with the latest version being .NET standard 2.0. Every .NET implementation (.NET/.NET Core) must advertise which .NET Standard it supports. [22].

Both the .NET framework and .NET Core support C-Sharp, F-Sharp and Visual Basic programming languages [22].

2.2 ASP.NET Core

As mentioned earlier, ASP.NET Core is a cross-platform and open source framework for building web applications, created and maintained by Microsoft. ASP.NET Core is made to run on the .NET Core, .NET or the mono run-time. It originated from ASP.NET which is a framework for building web applications that run on the full .NET framework. ASP.NET Core based applications are capable of running on Windows, Mac and Linux platforms [18] [10].

2.2.1 Porting a full .NET framework library (OIOSAML.NET) into .NET Core

As described in the problem description of this plan, SAML framework is XML based [12]. All protocol messages such as assertions, protocols, bindings and profiles are defined using a XML schema [23] [24]. Their integrity and confidentiality is guaranteed using XML signatures and XML encryption [25].

The full .NET framework uses the following APIs to create and verify XML digital signatures and XML encryption [15].

- System.Xml
- System.Security.Cryptography.X509Certificates
- System.Security.Cryptography
- System.Security.Cryptography.Xml
- System.Security.Cryptography.Xml.SignedXml

These APIs have been ported over to .NET Core 2.0 with the System.Security.Cryptography.Xml being the latest and the last API to be ported. This means that as of now, .NET Core supports signing of XML documents [26] [27] [28].

Microsoft recommends the following 5 steps when porting existing .NET frame-

work libraries into .NET Core [29]

1. Identify all the library dependencies and assert if they are also supported in .NET Core
2. Update the target framework of the library to .NET Framework 4.6.2
3. With the .NET Portability Analyzer tool, analyze the library assembly [30]. The generated report can help find out which APIs the library is using that are not or are supported by .NET Core.
4. Port the library tests first
5. Port the library

2.2.2 Session management in ASP.NET Core web applications

Web applications use the HTTP protocol to transfer web pages from a web server to a client through a web browser. The web browser opens a connection to the web server, makes a request, and waits for a response. Web servers do not store any state between consecutive requests, making the protocol stateless [31]. Since there is no storage between subsequent requests, there is no way of remembering what the web server transferred to a client in the past, or what activities the user carried out in the previous request [32].

The intention of session management in web applications is to create an association between an authenticated user and a session [32]. Sessions represent long lasting authenticated requests and responses, between the same user (browser), and a web server, starting when the user signs in to the web server, until when the user explicitly signs-out, or the session expires [32]. Different user related variables (e.g. user-is-authenticated, username etc) can be added to the session, and these will be remembered by the web server as long as the session is valid.

There are multiple ways of implementing session management in web applications (HTTP). Some of these include the use of HTTP cookies, URL parameters and hidden form fields [33]. This project focuses on using HTTP cookies as the main method of keeping user sessions within the SAML authentication middleware and the service provider's application.

HTTP cookies are pieces of text data that the web server sends to the client (browser) as part of the response. These are meant for the browser to store them on disk and send them as part of the next request to the web server. In this way, the web server can tell which HTTP requests belong to the same user (session). The web server will then read the contents of the cookie and populate the request

state/session. [34]

Microsoft offers the cookie authentication middleware as a way of managing user sessions in ASP.NET Core web applications. This is implemented as part of the
Microsoft.AspNetCore.Authentication.Cookies package [35]. The middleware is initiated during the start-up of a web application. The following is an example code from Microsoft showing how to initiate the cookie middleware.

```
1        services.AddAuthentication("Cookies").AddCookie();
```

Listing 1: Adding cookie middleware

ASP.NET Core 2.0 allows the use of multiple cookie instances, with each instance creating a new authentication handler, having the capability of authenticating the same user on different levels, and also using different identity providers. The above code will initiate the cookie middleware with a named cookie authentication handler (authentication scheme) called "Cookies". The handler will be responsible for storing the outcome of the authentication in a cookie called "Cookies" [36]. Signing in a user and creating the session cookie is accomplished by calling the cookie middleware extension method **SignInAsync** and passing the authentication scheme name e.g. "Cookies" from the above example and a **ClaimsPrincipal** (user data/claims/variables). The cookie is encrypted and attached to the current response [36]. To sign out a user, the extension method **SignOutAsync** is called with the authentication scheme name as a parameter.

```
        // creates the session cookie
        await HttpContext.SignInAsync("Cookies",
                              new ClaimsPrincipal(claimsIdentity));

        // removes the authentication session cookie
        await HttpContext.SignOutAsync("Cookies");
```

Listing 2: Extension methods to create(sign-in) and destroy (sign-out) session cookie

2.2.3 Configuring ASP.NET Core applications

In the full .NET framework, ASP.NET web applications use XML based configuration files. An XML configuration file called the **web.config** is placed at the root of the application. The **web.config** file has a section called **AppSettings** where application configuration variables can be added as key value pairs [37].

ASP.NET Core web applications, do not utilize **web.config** files for application configuration, but can read application configuration values from multiple external sources [37]. Each external source is implemented as a configuration provider for a given file format or source.

The following configuration sources are supported in ASP.NET Core [38].

1. File formats (JSON, XML and INI). This is the most commonly used configuration provider and it uses external json/xml/ini files placed in the application root.
2. Command-line arguments
3. Environment variables
4. In-Memory .NET objects
5. An encrypted user store
6. Azure key vault
7. Custom created or installed providers

ASP.NET Core's in-build configuration API is used to read stored configuration values from the configuration files as key value pairs [37].

2.2.4 Diagnostics, tracing and logging

When troubleshooting or debugging applications, tracing makes a fundamental technique as it provides developers with an insight on what is happening under the covers of their application. Tracing records relevant events during application run-time and keeps them for analysis [39]. This is helpful when testing or troubleshooting a piece of code as it can identify system errors, pinpoint exceptions, warnings and code behaviour. [40]. Combined with logging, information collected when tracing can also be stored for later offline analysis [39].

Chiarreta and Lattanzi identifies logging as an important part of a web application which is difficulty to implement. This is demonstrated by the fact that there are more that 1,900 different logging related frameworks available in the .NET package manager nuget [37].

Modern web applications utilize software packages from different vendors and authors. These can use different logging frameworks, making logging in applications complicated. Microsoft in the full .NET framework implemented the Common.Logging library as a means of solving this problem, by providing an abstraction and enabling switching between different logging frameworks e.g. log4net, NLog, Serilog [41] [37].

With ASP.NET Core, the Common.Logging library is not required anymore.The framework offers the same behaviour out of the box [37]. This is implemented as a built in logging API, supporting different logging providers (frameworks) and offering the possibility of adding custom third party logging frameworks. Logging providers can be configured to show a log message, e.g a console logger or store message in a file or database [42].

To create logs, the **ILogger** interface is injected into a class constructor. With the help of the in built dependency injection in ASP.NET Core, this will create an **ILogger** object of a given category name. The category name in this context will be the class name. The **ILogger** interface expose different logging methods, depending on the required log level. A log level defines the degree of importance of a given log message [42]. The different log levels supported by ASP.NET Core are [42].

1. **Trace.** The lowest level indented for log messages useful only to developers. Should never enabled in production as they may contain sensitive information.
2. **Debug.** Intended for log messages that are useful during development and

debugging sessions

3. **Information.** For general log messages useful for tracing flow in the application
4. **Warning.** Warning log messages are used for indicating any error of which application did not stop but automatically recovered. These point to actions that needs more investigation.
5. **Error.** The error log level is indented for log massages indicating failure during a given operation.
6. **Critical.** Service level log messages that lead to service shutdown or prevent the service from starting

2.2.5 ASP.NET Core middleware

To create the SAML authentication middleware, it is important to understand the role ASP.NET Core middlware play in handling requests and responses in the application pipeline.

A middleware is a reusable class or a method that uses a request delegate to build the request pipeline. The Run, Map and Use methods are used to configure request delegates. When the application receives an incoming HTTP request, the request is passed through each middleware, (request delegate) in the pipeline. The main function of each middleware is to execute a given function or block of code on the request and determine whether to allow the the request to enter the next middleware in the pipeline or not. This is called short circuiting [11].

The figure below shows the SAML 2.0 authentication middleware placement in the context of other middlewares in an ASP.NET Core request-response pipeline [11].

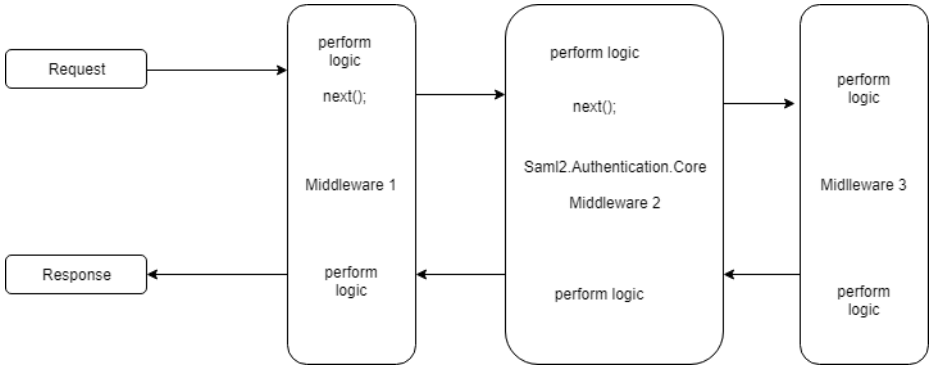

Figure 1: SAML 2.0 authentication middleware in the context of ASP.NET Core request pipeline

2.2.6 Handling authentication request-responses

In ASP.NET Core 2.0, authentiation middlewares are implemented as authentication handlers. The **AuthenticationHandler<TOptions>** class in the **Microsoft.AspNetCore.Authentication** name space is the base class under which new custom authentication handlers can be derived. The class contains the constructors, properties and methods necessary for processing each HTTP request in the HTTP pipeline [43].

The class is abstract, takes in one parameter **Options**, which represents the base Options model for all authentication middlewares. The **AuthenticationOptions** class is used to pass authentication middleware configuration values, into the authentication handler. These can be for example, the **AuthenticationScheme** property, which sets the logical name of the authentication middleware. This enables the possibility of using the same authentication middleware, more than once in the HTTP pipeline, with different authentication scheme names [44] [43].

The base authentication handler class offers different methods which should be implemented by any custom authentication handler. Some of these methods are discussed below.

```
1 public abstract class AuthenticationHandler<TOptions>
2     : Microsoft.AspNetCore.Authentication.IAuthenticationHandler
      ↪  where TOptions
3     : AuthenticationSchemeOptionsnew()
```

Listing 3: Base AutheticationHandler class

HandleChallengeAsync()

This method should be implemented, to deal with unauthorized requests, and user login challenge. It should contain logic that performs some kind of authentication checks, and returns the appropriate response, either by modifying the HTTP response headers, redirecting to a login page, or creating an authentication request and redirecting to an external identity provider for authentication as implemented in this project [45].

The method takes in one parameter of type **AuthenticationProperties**. This is used to temporarily store state values, during an authentication session. The state values should include a redirect URL. This denotes the endpoint where the authentication handler will transfer control to after a successful authentication challenge. State values are not limited to redirect URLs, but any data that the consumer of the middleware needs to be restored, after a successful or failed authentication attempt [46].

IAuthenticationRequestHandler interface

The IAuthenticationRequestHandler interface contains one method

```
public System.Threading.Tasks.Task<bool> HandleRequestAsync();
```

which is called on every request coming in through the HTTP pipeline. It returns true if HTTP request processing should stop, or false if the HTTP request should be passed on to the next middleware in the HTTP pipeline [47]. This is useful especially for authentication handlers that redirect to external identity providers for authentication, as they will need a way of receiving the authentication response, handling it and reporting to the HTTP pipeline that the request has been handled.

One way of doing this, is by listening to HTTP requests to a particular application path or endpoint. If the current HTTP request path equals to a pre-configured

value, then the authentication handler should handle the request, otherwise the handler should return false which forwards the HTTP request into the next middleware.

The method **HandleRequestAsync** should be implemented to handle different requests-response messages, based on different pre-configured application paths; for example handling SAMLResponses through pre-configured Assertion Consumer Service or Single Logout URLs [47].

2.3 SAML 2.0

SAML 2.0 is a general framework for transferring identity and security information about a user, also known as a subject, between a service provider and a SAML 2.0 based identity provider. [7]. Oasis, describes three major uses of SAML; Web Single Sign-On (SSO), Attribute-Based Authorization and Securing Web Services. This thesis looks at SAML 2.0 in the context of providing Web SSO.

2.3.1 Web Single Sign-On (SSO)

In access control, single-sign-on is the reuse of an existing session by an identity provider, for a given user. The session is usually set when the user tries to access resources, on any service provider, protected by the same identity provider. If a user tries to access resources on another service provider, protected by the same identity provider, and on the same browser, the identity provider will redirect back to the service provider, with an assertion from the existing session, as long as it is valid. The user will not have to re-supply their login credentials [7] [9].

Identity federation is the sharing of a users identity information between multiple identity providers. The same user identity information is used to identify, and give a user access to resources on different systems (service providers). The identity providers have to agree on which subset of identity information or attributes to federate [7] [9].

As described earlier, web SSO makes it possible for a user to authenticate on website A, using an Identity Provider A, and without an additional authentication, the user is allowed to access protected resources at website B, that uses identity provider A. A website here refers to a Service Provider.

SAML 2.0 web SSO, works by allowing the communication of the initial authentication assertion, that was issued to the user after authenticating in website A, to website B. If website B, trusts the origin of the assertion, website B, can then choose to sign in the user as if they were directly authenticated [7].

The figure below shows the basic web SSO model [7].

The identity provider authenticates users, which are subsequently recognized and allowed to access restricted resources at the service provider [7].

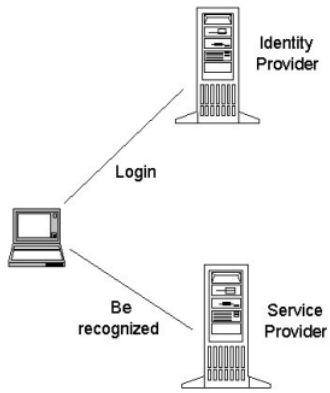

Figure 2: Basic SSO model

2.3.2 SAML 2.0 Components

The SAML 2.0 framework is defined by different components which are, assertions, protocols, bindings and profiles [48].

Assertions

SAML assertions represent packages of information about an identity, issued by an identity provider and transferred to the service provider as a response to an authentication request. Three types of assertion statements are defined in the framework; authentication, attribute and authorization decision. The three types can be created by any SAML identity provider. [48].

The authentication assertion statement, is generated by an identity provider, to show that a given subject was authenticated using a particular method at a given time. The attribute assertion statement shows that the given attributes are associated with the specified authenticated subject, while authorization decision statements show, whether requests to access a given resource at the service provider has been granted or not [7]

Structure

SAML assertions are XMLDocuments, and their structure is generic. Inside the XMLDocument, different XMLElements describe the authentication, attribute, authorization decision or any other custom defined statements.

The figure below from OASIS represents the high level structure of a SAML authentication assertion [7]

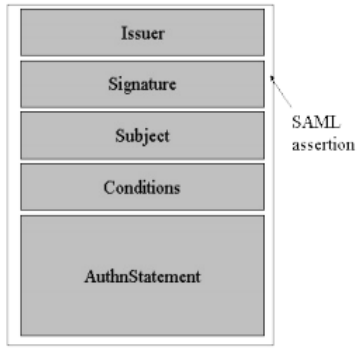

Figure 3: Structure of a SAML assertion

Protocols

SAML protocols define how an identity provider and a service provider, will handle different SAML requests and responses. Service providers send authentication (login) and logout requests to the identity provider and expect responses for the same in return [23].

Other SAML requests, that a service provider can send to an identity provider include; requests to authenticate a given principal and return an assertion, requests that a name identifier be registered, requests to terminate the use of a given identifier, requests to retrieve SAML Protocol messages by means of an artifact, requests to destroy related sessions, and requests for a name identifier mapping [7].

The following protocols are supported by the SAML standard; Assertion Query and Request Protocol, Name Identifier Management Protocol, Authentication Request Protocol, Artifact Protocol, Name Identifier Mapping Protocol and Single

Logout Protocol [23]. This project focuses on implementing the Authentication Request Protocol, Artifact Protocol and the Single Logout Protocol in the authentication middleware. Further work to support other protocols is outside the scope. The protocols are elaborated below, but their concrete usage and examples are given in the results chapter.

Authentication Request Protocol

The Authentication Request Protocol defines an authentication request <AuthnRequest>, which is issued by a service provider to an identity provider, requesting for authentication of a given subject. The identity provider responds with a <Response>, containing one or more assertions of the authenticated subject [23].

Artifact Protocol

The Artifact Protocol provides a way of transferring authentication assertions by reference, also referred to as an artifact. During authentication, the identity provider creates the assertion, and assigns a reference/artifact to it. The SAML response to the service provider contains the artifact, which can be used to obtain the actual assertion through a back channel. [23].

Single Logout Protocol

The Single logout Protocol defines a request, issued by a service provider to an identity provider, allowing the destruction of all sessions associated with a given subject. The logout request can be initiated by either a service provider or an identity provider [23].

Bindings

SAML protocol Bindings, represent mappings from SAML request-response exchanges, into the standard communication protocols like HTTP-GET, HTTP-POST and SOAP. These define how SAML protocol messages can be communicated within HTTP-GET (redirect), HTTP-POST or SOAP messages. The aim of SAML bindings is to ensure that implemented SAML software can operate using standard messaging and communication protocols.

Some bindings define a **RelayState** mechanism, whereby state data can be round tripped to the identity provider and back as a parameter in the request-response exchange. This is necessary to preserve the application state before authentication, and to restore the state after authentication [7]

The standard defines the following bindings; SOAP Binding, HTTP-Redirect, HTTP-POST, HTTP-Artifact and URI Binding. This project looks at the HTTP-

Redirect and HTTP-Artifact bindings as implemented in the authentication middleware [7].

HTTP Redirect Binding (urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect)

The HTTP-Redirect Binding transmits SAML protocol messages, within browser URL parameters. SAML or the HTTP Protocol does not enforce a limitation on the resulting request URL length, but standard web browsers do. For example, the Microsoft Internet Explorer browser has a limit of 2,083 characters. According to Boutell, long URLs over 2,000 characters are discouraged as they will not work in all browsers [7] [49].

To carry SAML XML messages on a URL, special encoding have to be applied on the message. More complex messages should be transmitted using the HTTP POST or HTTP Artifact bindings [7].

State data can be transmitted within the protocol message as a RelayState parameter in the HTTP redirect binding. SAML enforces a maximum size of 80 bytes, and requires that the integrity of the state be protected by the creator of the state. The creator is either a service provider or an identity provider. The standard requires that, any SAML message that has RelayState within, the responder must return the same RelaySate as received in the corresponding response using a binding that supports RelayState [7].

SAML protocol messages withing the HTTP Redirect binding are encoded using URL encoding techniques and transmitted using the HTTP GET method. The DEFLATE compression encoding is one of the URL encoding techniques used. This is identified as urn:oasis:names:tc:SAML:2.0:bindings:URL-Encoding:DEFLATE [7].

The following steps builds a signed authentication request URL using the DEFLATE encoding technique [7].

1. The XML request (without signature) is deflate encoded, then URL encoded and added to a parameter SAMLRequest.
2. The RelayState, if available is DEFLATE encoded and URL encoded and added as a parameter.
3. The signing algorithm is URL encoded and added as parameter.
4. The signature of the result is calculated using the correct hashing algorithm.
5. The signature is converted into a base 64 string, URL encoded, and added as a parameter.

The resulting URL would look like `{destination}?SAMLrequest=request&`

`RelayState=relaystate&SigAlg=alg&Signature=signature`

The diagram below from OASIS models the communication between a service and identity provider, using the HTTP Redirect binding. The user agent represents the user's browser, while the SAML Requester or SAML responder can either be a service or an identity provider [7].

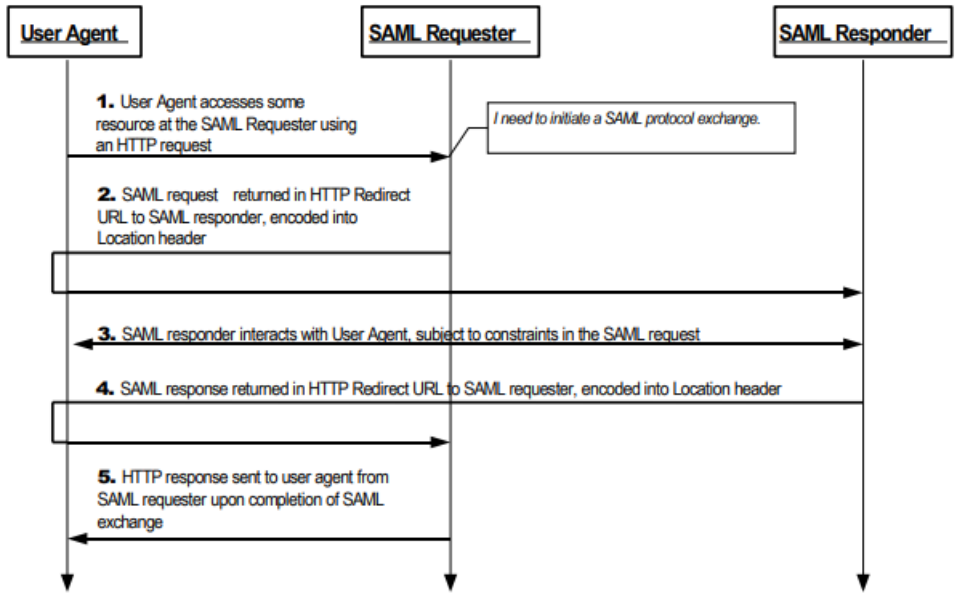

Figure 4: Communication model using HTTP Redirect binding

HTTP Artifact binding (urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact)

The HTTP Artifact binding represents a way of transmitting the SAML request, the SAML response, or both by a reference; also know as an artifact. The SAML requester creates the actual SAML message, assigns it a reference and stores it locally. The SAML requester then transmits the artifact to the SAML responder through either HTTP Redirect or HTTP POST binding. On receiving the artifact, the responder can then retrieve the actual message by the artifact using a back channel binding, such as the HTTP SOAP binding. The HTTP Artifact binding therefore uses multiple bindings to transmit request and response messages [7].

The authentication middleware created in this project supports transmitting the SAML request, using the HTTP Redirect binding, and receiving the SAML Response via HTTP Redirect or the HTTP Artifact binding.

The HTTP Artifact binding is recommended for cases where the service and identity provider must communicate via a HTTP user agent (browser), but not transmit the whole message through the front channel. This can be due to technical or security reasons. To resolve the actual message, an **<samlp:ArtifactResolve>** request is sent to the requester through a pre-configured communication path, basically an endpoint [7].

The binding supports transmitting of a RelayState together with the SAML Artifact, in the same manner as the HTTP Redirect binding. When transmitting using the HTTP Redirect binding, the artifact value is URL encoded, and added into a parameter known as SAMLart. If RelayState is present, it is URL encoded and added to the query, in a parameter called **RelayState**. If transmitting using the HTTP POST Binding, the SAMLart is form-encoded and placed in a form named SAMLart with an additional hidden field RelayState. The action of the form is set to the recipient's assertion consumer service URL, with the method set to **POST** [7].

The following diagram from OASIS shows the system model for request-response SAML communication using the HTTP Artifact binding [7].

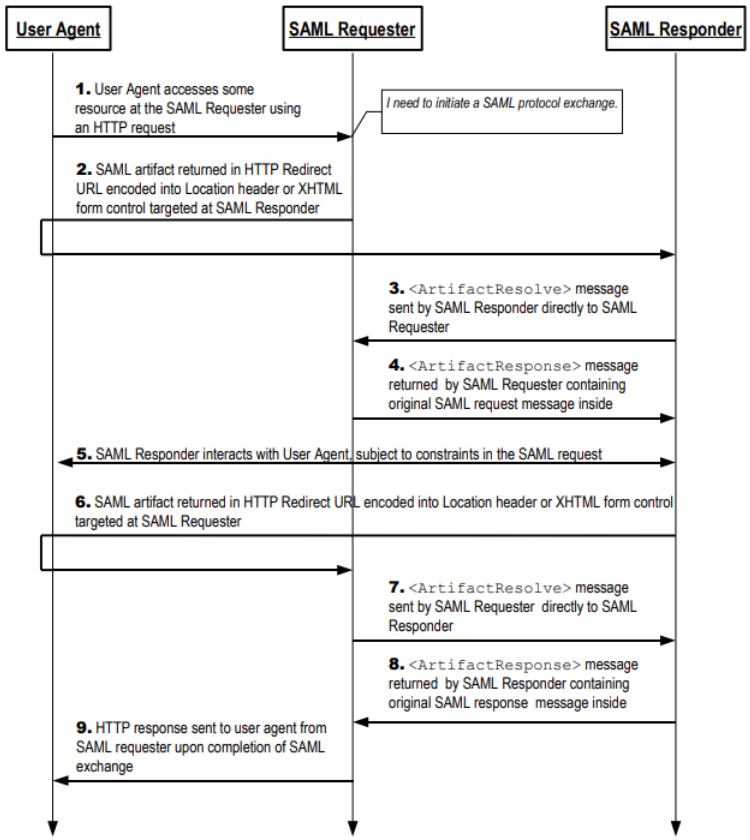

Figure 5: Communication model using HTTP Artifact binding

Profiles

SAML profiles specify how SAML components are combined to work together in a particular application enhancing interoperability [50]. Described below are some of the SAML profiles that are implemented in this project.

Web Browser SSO Profile

The Web Browser SSO Profile combines the Authentication Request Protocol with the HTTP Redirect, HTTP POST and the HTTP Artifact bindings to support Single Sign-On in web browsers [50].

The Profile defines four types of communication models grouped into twos. These are the **push or pull** defined by how SAML assertions, are delivered to the Service Provider, while the **Idp or SP initiated** models are defined by how the communication flow is initiated. The pull model involves using the Artifact binding to send a message by a reference an allowing the receiver to 'pull' the actual message related to the reference. The push model is used to deliver SAML messages through HTTP redirects or HTTP POST [7].

IDP and SP initiated communication flows can be combined with the different SAML bindings to give six different use cases. Two of the actual use cases for this project are discussed below [7].

SP initiated: Redirect → POST binding This use case assumes an unauthenticated user trying to browse restricted resources in service provider www.abc.com which has it identities provided by identity provider www.xyz.com. The website www.abc.com will therefore create an <AuthnRequest> message and deliver it to www.xyz.com through HTTP Redirect while the identity provider in response will prepare and deliver a SAML response through the HTTP POST binding [7].

The figure below from OASIS illustrates the communication [7].

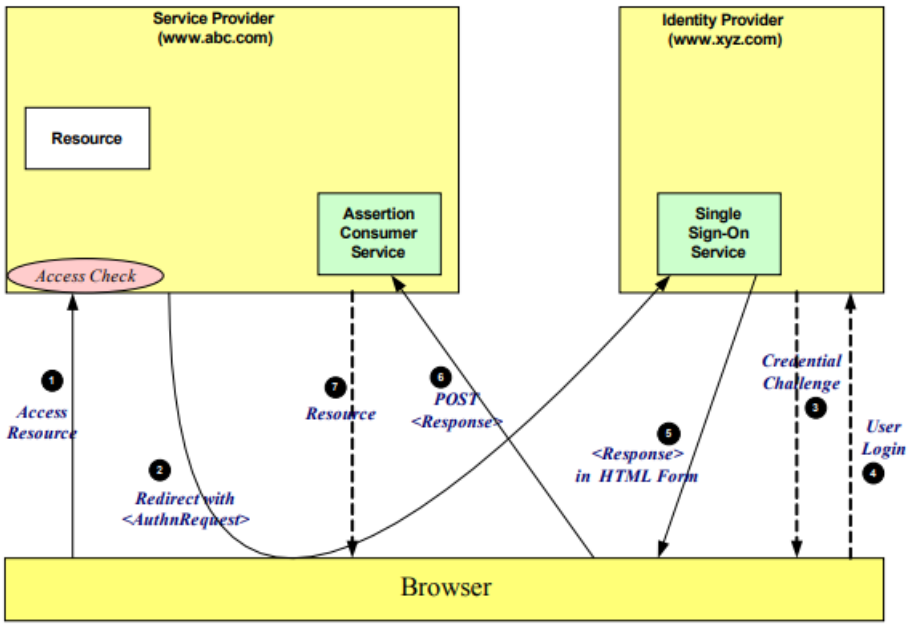

Figure 6: SP initiated: Redirect → POST binding

Description of the process [7].

1. Anonymous user tries to access a restricted resource
2. SP creates and sends an <AuthnRequest> message and redirects to IDP
3. The SSO service decides if user has to login and challenges the user for credentials
4. User provides valid credentials
5. The SSO service POSTS a SAML response back to the browser
6. The browser issues an HTTP POST containing the SAML Response to the

Assertion Consumer Service (ACS)

7. ACS validates the digital signature on the SAML Response and issues an HTTP Redirect to the browser allowing the user to browse the protected resource

SP initiated: Redirect → Artifact binding This use case assumes an unauthenticated user trying to browse a restricted resource in service provider www.abc.com which has its identities managed by identity provider www.xyz.com. The service provider www.abc.com will therefore create an <AuthnRequest> message and deliver it to www.xyz.com through the HTTP Redirect binding. The identity provider www.xyz.com will in turn respond with a SAML Artifact through an HTTP POST message. The service provider can use the SAML Artifact to resolve the actual SAML response from the identity provider [7].

The figure below from OASIS illustrates this use case [7].

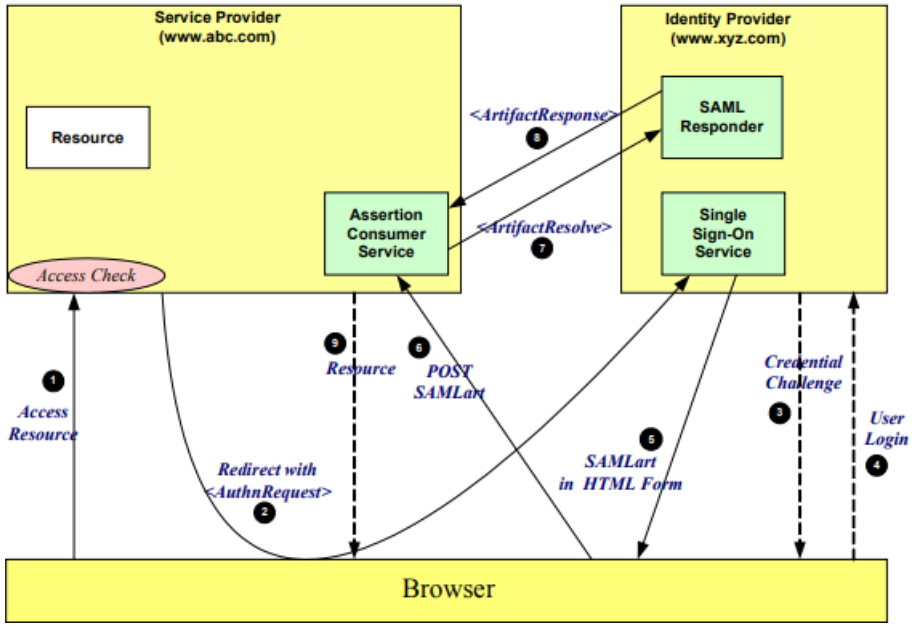

Figure 7: SP initiated: Redirect → Artifact binding

Description of the process [7].

1. Anonymous user tries to access a restricted resource.

2. SP creates and sends an <AuthnRequest> message and redirects to IDP.
3. The SSO service decides if user has to login and challenges the user for credentials.
4. User provides valid credentials.
5. The SSO service generates an assertion and an artifact for the user, and sends the artifact to the browser.
6. Browser POSTS the SAMLart to the ACS and extracts the source ID from the artifact to find the identity of the SAML responder (www.xyz.com).
7. The ACS sends an <ArtifactResolve> message to www.xyz.com.
8. www.xyz responds with an <ArtifactResponse> message containing the previously generated Assertion and establishes a session for the user.
9. The ACS redirects the browser to the protected resource with session cookies for access control.

Single Logout Profile

The Single Logout Profile defines how the Single Logout protocol is combined with other bindings (SOAP, HTTP Redirect, HTTP POST and HTTP Artifact) to support the destruction of SAML sessions for a particular subject [50].

Artifact Resolution Profile

The Artifact resolution Profile specifies the combination of the Artifact Resolution Protocol and other bindings like the SOAP binding [50].

3 Research methods

This chapter discusses the research methods used in the thesis.

3.1 Exploratory case study as the research method

To be able to answer the above research questions, an exploratory case study was used as the main research methodology. This was combined with the waterfall model for software development. According to Yin, case studies are the preferred research method strategy when "how" research questions are formulated as it is in this project [51].

Case study is a robust method of investigation that allows researchers to explore, and understand complex issues. It gives the ability to combine both qualitative and quantitative data, explaining the process and outcome of an experiment [52]. This thesis aims at investigating and understanding how to build a SAML 2.0 authentication middleware for the ASP.NET Core web application development framework. The projected is scoped to implement the HTTP Redirect and HTTP Artifact bindings of the SAML framework.

The outcome of the experiment was a software application named Saml2.Authentication.Core and source code lives in github (`https://github.com/jkmu/Saml2.Authentication.Core`).

To validate the results, a demo web application that referenced the authentication middleware software package (Saml2.Authentication.Core) as a dependency was created. The demo web application was then configured using the authentication middleware, to use a real world SAML identity provider salesforce.com (`https://www.salesforce.com`), for authentication and managing identities.

3.2 Literature review

The study focused on related work in regard to implementation of SAML authentication in ASP.NET Core web applications. It was of importance to understand how SAML 2.0 authentication works, and how SAML 2.0 can be implemented into a middleware useful in ASP.NET Core web applications development. OASIS [53], which is the body behind SAML 2.0 has released a lot of literature and

technical documentation for SAML 2.0. This literature was studied, combined with the OIOSAML.NET library [15] in order to understand; the inner workings of the SAML 2.0 framework, the .NET framework classes that implement the different aspects of the framework, and the corresponding classes in the .NET Core framework for easier porting.

The Microsoft documentation for ASP.NET Core was also studied in order to answer the other research questions, namely; session management, configuration of identity provider's information and tracing for troubleshooting. The stack overflow forum was utilized for searching of examples and asking software development related questions, combined with standard search engines such as www.google.com and others. Other search engines that are more related to science like the www.scholar.google.com, Springerlink (link.springer.com), Science Direct (www.sciencedirect.com) and IEEEPlore (http://ieeexplore.ieee.org) were also used.

Since there was limited literature on creating authentication middlewares/handlers for ASP.NET Core, as it is not a task software developers do often, the Microsoft source code for ASP.NET Security (https://github.com/aspnet/Security/tree/dev/src) was studied. This gave different examples of authentication handlers created by Microsoft for Facebook, Google authentication etc.

3.3 Software development

3.3.1 Methodology

When developing software, the choice of a good software development methodology is important for success. A good software development methodology forces discipline on the software developer, helping them to increase the overall understanding of the problem, which will in turn improves the quality of the product [54]

To develop the middleware, the waterfall method of software development life-cycle was used. This is one of the oldest methods of software development, proposed by Winston W. Royce in the 1970 [55]. The model is comprised of several phases, one after the other and in a sequential manner, with the output from one phase being the input into the next phase. The phases are analysis, design, implementation, testing and maintenance [55].

The waterfall model was suitable for this project because, it is simple, easy to understand, the project is short, and the requirements are clear. Since only one

person was working on the project, it was easy to set milestones while completing tasks from one stage/phase to the next.

3.3.2 Tools

This section shortly describes the software development tools used to create the SAML authentication middleware.

Programming language

C#

C# is an object oriented language for building different application types for the .NET framework. The language is type-safe with encapsulation, inheritance and polymorphism concepts built in. Its syntax is similar to C, C++ or Java programming languages making it easy for developers to switch to any of these languages with ease [56].

Integrated Development Environment (IDE)

An IDE is an application that helps to manage the development of other applications, combining all the required tools and features into one application. Some of the tools offered by an IDE include code editors, compilers, debuggers, automation tools e.t.c [57]

Visual Studio 2017

Created by Microsoft, Visual Studio provides all the tools necessary for building mobile, Windows, web and cloud applications with the possibility of installing only the components needed for the particular application type. The IDE contains code editing features, with an in built debugger and a code testing framework [58].

Resharper

Resharper is a code inspection extension for Visual Studio with the aim of improving code quality. Resharper analyzes code quality on the fly and offers suggestions where the code can be improved. It also enforces programming language specific code styling and formatting [59]

Source control

Also known as version control, or revision control, it is a component that manages changes in digital artifacts including documents or computer software as it is in this context. It helps software developers to work simultaneously on the same files, while helping to resolve any change, merging conflicts and keeping the

history or version of all changes to a given file [60].

Git

Git is a distributed version control system and it is one of the worlds most used [61]. It is a decentralized version control system (DVCS) as it keeps a full history of all the changes in each developers machine on the contrary to a centralized version control systems which only keeps a single copy with the full history on a centralized server [62].

Github

Github is a web based repository hosting service using git. It hosts both open source and business repositories and it will be used to host the software source code. [63]

Git extensions

Git extensions is a toolkit that is installed in Windows to work with git. It integrates with Windows explorer, Visual Studio and provides a user interface for git. Some of the git commands on git extensions include clone, commit, push pull and merge [64].

Nuget package manager

Project planning

A project planning tool was required to be able to plan and deliver the planned milestones effectively and efficiently. A trello kanban board was used to create, organize and prioritize the different tasks and user stories. The board kept a back log of user stories and tasks. It was configured to show the overall status and progress of the project, based on the status of the user stories in different columns (To do, On Going, Testing and Done) [65]

4 Saml2.Authentication.Core

In this chapter we will discuss the middleware that was developed in this project. We start with listing the requirements, then we look at how parts of the middleware were ported from OIOSAML.NET to ASP.NET Core, and the different components making up the middleware. The chapter ends with a discussion of how the components are put together, work together as a whole, to provide a fully functioning SAML 2.0 authentication middleware for ASP.NET Core.

Saml2.Authentication.Core is the authentication middleware that was created during the thesis. It is a software application implementing the SAML 2.0 authentication protocol in the ASP.NET Core framework for web application development. The software brings SAML into the world of ASP.NET Core, by creating an authentication middleware/handler, that supports the HTTP Redirect and HTTP Artifact SAML bindings.

The middleware can easily be added into any ASP.NET Core web application (SP) as a dependency, and configured into the HTTP request processing pipeline with minimal effort. When used, the middleware takes over user challenging for authentication, creating SAML authentication requests (<AuthnRequest>) and redirecting to the configured identity provider for authentication.

After the user enters valid credentials in the identity provider's login form, the middleware is capable of receiving the authentication response by HTTP Redirect or HTTP artifact binding, validating the SAMLResponse, getting the assertion, reading identity information (subject, username, name etc) from the validated assertion, setting a local session cookie, restoring the RelayState, and redirecting back to the service provider's requested page, or resource while maintaining the integrity and confidentiality of the exchanged identity, and the communication it self.

4.1 Requirements and specification

As mentioned earlier, the waterfall method of software development was used when creating the software. During the analysis phase, the following specifications and requirements were uncovered.

#	**Requirement/Specification**
1.	Should have support for HTTP Redirect Binding
2.	Should have support for HTTP Artifact binding through SAMLResponse
3.	Should have support for SP-initiated Single Sign-On
4.	Should have support for SP-initiated SLO
5.	Should be easily configurable into the ASP.NET Core 2 HTTP pipeline
6.	Should be able of keeping user sessions in a secure way using ASP.NET Core 2 cookie authentication
7.	Should support adding configuration values for IDP and SP using JSON and ASP.NET Core 2 configuration options
8.	Should be able to trigger the middleware/authentication handler using challenge/challenge result
9.	Should be able to send user to the page they were in application after authentication using RelayState
10.	Should validate SAML requests and responses using digital signatures according the SAML 2.0 specification (protect message integrity)
11.	Should stop/detect automated/replay requests and attacks
12.	Should be able to ensure that contents of SAML messages are only accessible by the intended recipient (confidentiality)

Table 1: Requirements and specifications

4.2 Porting OIOSAML.NET library into ASP.NET Core

To reduce the amount of work in creating the middleware, parts of the open source, full .NET framework, SAML class library from Digitaliserdk was used as the base. These parts include, the core SAML protocol types, schema types and assertion validation types [15] [66]. Since these types where written targeting the full .NET framework, they had to be ported into the .NET Core framework.

The .NET API Portability Analyzer tool [30] was used to analyze these types. The aim was to identify the full .NET APIs they used, and their portability to .NET Core. The table below gives a summary of the results. The full API portability report is given in appendix A.

Submission Id	07113ced-343d-4ecc-bdc3-5d4b50a7f384
Description	Summary of .NET Core API portability analysis
Targets	.NET Core,.NET Framework,.NET Standard

Assembly name	Target Framework	.NET Core	.NET Standard
dk.nita.saml20 (OIOSAML.NET)	.NETFramework,Version=v4.7.1	81,85	70,53

Table 2: Summary of .NET Core API portability analysis

The report from the portability analysis (appendix A) gives information about supported and not supported APIs in .NET Core. The tool is supposed to be capable of giving some hints on how to fix the APIs flagged as not supported. In the case of partially porting OIOSAML.NET, the tool only managed to give recommendation for only about 1% of the used types.

From table 2 we can see that the OIOSAML.NET library targets the .NET framework version 4.7.1, and that, the analyzed target types to port where 81% portable to .NET Core, and 70% portable to .NET Standard. Theoretically, this meant that 81% of the code in the core SAML protocol types, schema types and assertion validation types, could be used in making the middleware. This did not end up being the case, as making the 19% changes to work led to changing more of the existing code. This transformed into roughly 40% of the code in these types, being used as the base for the SAML 2.0 authentication middleware.

4.3 Architecture

Figure 8 shows the architecture of the authentication middleware as placed within the ASP.NET Core HTTP pipeline. The middleware receives and handles only SAML authentication and logout HTTP requests. Other requests are passed on to the next middleware in the pipeline. There are three main components that make up the middleware; HandleChallengeAsync, SignOutAsync and HandleRequestAsync. Their functions are described below, while their concrete implementations are discussed in section 4.5.

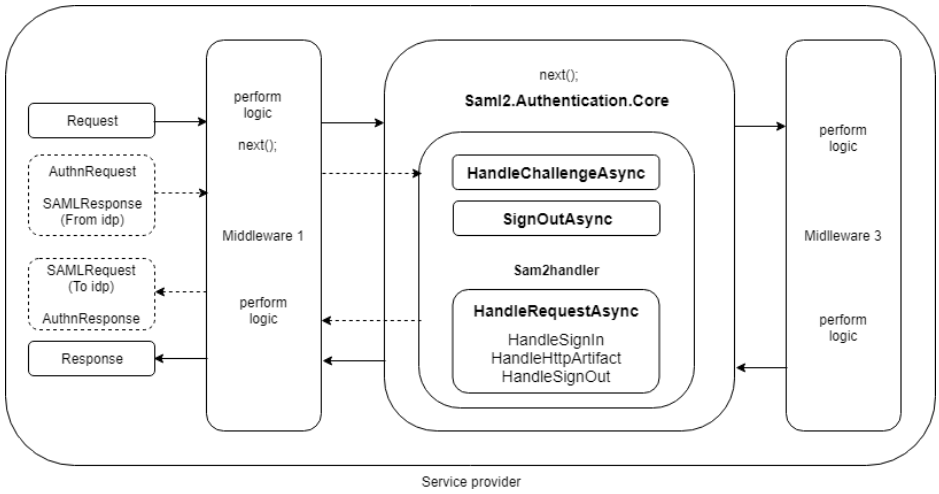

Figure 8: Architecture diagram

AuthnRequest Authentication requests initiated by the service provider.

SAMLResponse Authentication response from identity provider to service provider.

SAMLRequest Authentication request from service provider to identity provider (external redirect).

AuthnResponse Authentication response from middleware to service provider (internal redirect).

HandleChallengeAsync Component for handling service provider authentication requests.

SignOutAsync Component for handling service provider logout requests.

HandleRequestAsync Component for handling authentication (HTTP redirect, HTTP Artifact) and logout SAMLResponse from identity provider.

4.4 Solution structure

The solution is made up of two projects; the middleware (**Saml2.Authentication.Core**), and a demo web application (**DemoWebApp**) which was used for testing the middlware. The classes making up the middleware are logically grouped into folders of related functionality. These includes, **Validation**; for classes handling validation of requests and responses, **Bindings**; for all binding related functionality, **Factories**; for classes responsible for creating authentication requests and responses, **Utils**; for utilities like digital signature, encryption and serialization helpers, **Extensions**; for classes that extend standard Microsoft types (strings, StringBuilder, ClaimsPrincipal, HttpRequest, HttpResponse) among others.

The figure below represent the solution as seen from Microsoft Visual Studio 2017 IDE. Appendix V shows the complete class diagram.

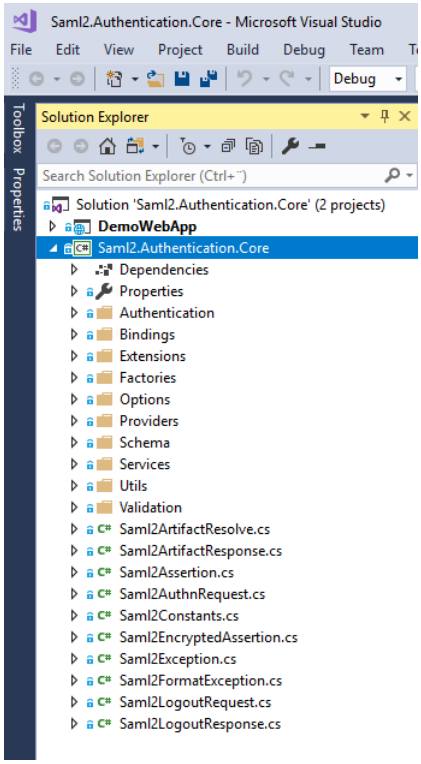

Figure 9: Solution structure

4.5 The SAML 2.0 authentication handler

The **Saml2Handler** is the core of the middleware. It is an ASP.NET Core authentication handler that coordinates the communication between the client (SP), and the IDP. It contains all the necessary methods and dependencies required for handling authentication challenges, authentication request-response exchanges, logout requests, and logout responses. To do the work, the handler depends on other services, factories and binding classes that are injected in to the handler using the inversion of control mechanism. This is provided by the ASP.NET Core framework by default. Some of the dependencies are described below. Their full implementations are given in 8.2.

IOptionsMonitor<Saml2Options> An instance of the Saml2Options object, providing default middleware configuration options, including the Assertion Consumer Service and Single Logout URLs among other configuration options (Appendix F).

ILoggerFactory An instance of the LoggerFactory used for configuring the logging system.

UrlEncoder Used for URL character encoding.

ISystemClock An abstraction of the system clock.

ISamlService An instance of the SamlService which contains methods for creating SAML request URLs and handling SAMLResponses (Appendix J).

IHttpRedirectBinding An instance of the HttpRedirectBinding object containing HTTP Redirect binding related implementations. These includes, building authentication request URL and handling HTTP Redirect binding SAMLResponse (Appendix P).

IHttpArtifactBinding An instance of the HttpArtifactBinding object containing implementations to handle HTTP Artifact binding. This includes but is not limited to; getting SAMLart from SAMLResponse and resolving the assertion from the ArtifactResolutionService (Appendix O).

ISaml2ClaimFactory An instance of the Saml2ClaimFactory which receives a validated assertion, reads the transferred identity information and transforms the identity values into a list of claims (Appendix M).

The following sub sections gives an insight into the functionality provided by the authentication handler. The methods making up the handler are described. The full implementation of the handler is given in appendix B.

4.5.1 Handling SAMLResponse

The **HandleRequestAsync** method is called on every HTTP Request into the web application. It gives a way of performing logic on every request and determining if the HTTP Request should be handled by the middleware or not. The method returns true if the request has been handled and false if not, allowing the HTTP pipeline to forward the request to the next middleware in the HTTP pipeline [11]. Basically, this method handles SAMLResponses from the identity provider. These can either be authentication request responses (HTTP Redirect or Artifact binding) or logout responses. It uses three concrete methods to handle sign-in with HTTP Redirect, signout and sign-in using HTTP Artifact binding. These are described further down.

```
1         public async Task<bool> HandleRequestAsync()
2         {
3             if (await HandleSignIn())
4                 return true;
5
6             if (await HandleSignOut())
7                 return true;
8             return await HandleHttpArtifact();
9         }
```

Listing 4: HandleRequestAsync method

4.5.2 Handling logout requests

The **SignOutAsync** method handles SP-initiated logout requests. It takes in one parameter of **AuthenticationProperties** type, passed from the client. It contains the redirectURI where the middleware will redirect to after a successful authentication, among other RelayState values. The method calls the SamlService to create the logout request URL with the following parameters;

LogoutRequestId A unique identifier which should corresponds to the expected

InResponseTo value of the logoutResponse. The LogoutRequestId is kept in a protected cookie.

SessionIndex A value indicating the session on the identity provider that the service provider requests to be destroyed. This value is obtained from the previous assertion.

Subject A value indicating the session owner. This value is also obtained from the previous assertion.

RelayState A value indicating the state which should be restored after logout. This value is the AuthenticationProperties parameter which is confidentially protected using the data protection API [67].

The method transfers control to the Identity Provider by setting the HTTP Response to redirect to the created logout request URL. This triggers the logout on the identity provider.

```
public Task SignOutAsync(AuthenticationProperties properties)
{
    _logger.LogDebug($"Entering {nameof(SignOutAsync)}",
                         properties);

    var logoutRequestId = CreateUniqueId();
    var cookieOptions = Options.RequestIdCookie
                             .Build(Context, Clock.UtcNow);
    Response.Cookies.Append(Options.RequestIdCookie.Name,
        Options.StringDataFormat.Protect(logoutRequestId),
        cookieOptions);

    var relayState = Options.StateDataFormat.Protect(properties);
    var sessionIndex = Context.User.GetSessionIndex();
    var subject = Context.User.GetSubject();

    var logoutRequestUrl = _samlService.GetLogoutRequest(
        logoutRequestId, sessionIndex, subject, relayState);

    _logger.LogDebug(
        $"Method={nameof(SignOutAsync)}.
        Redirecting to saml identity provider for SLO.
        Url={logoutRequestUrl}");

    Context.Response.Redirect(logoutRequestUrl, true);
    return Task.CompletedTask;
}
```

Listing 5: SignOutAsync method

4.5.3 Handling authentication requests

The **HandleChallengeAsync** method is the equivalent of the **SignOutAsync** method, but for handling authentication requests using the HTTP Redirect binding. It is

triggered by the client when user tries to access a restricted resource (how to trigger authentication is discussed further down). The method has one parameter of type **AuthenticationProperties** containing RelayState. This includes a redirectURI to redirect to, after authentication, among other state values. It uses the SamlService instance to get the actual authentication request URL with the following parameters;

AuthnRequestId A unique identifier which should corresponds to the InResponseTo value of the expected SAMLResponse. The AuthnRequestId is kept in a protected cookie.

RelayState A value indicating the state of the service provider that should be restored after authentication. This value is the **AuthenticationProperties** parameter which is confidentially protected using the data protection API [67].

AssertionConsumerServiceUrl Represents the URL for the Assertion Consumer Service, capable of receiving and handling a SAMLResponse from the Identity Provider. This value is added into the <AunthnRequest> as configured in the identity provider.

The method transfers control to the Identity Provider by setting the HTTP Response to redirect to the created authentication request URL. This triggers the login form on the identity provider.

```
protected override Task HandleChallengeAsync(
    AuthenticationProperties properties)
{
    _logger.LogDebug($"Entering {nameof(HandleChallengeAsync)}",
        properties);

    var authnRequestId = CreateUniqueId();

    var deleteCookieOptions = Options.RequestIdCookie
        .Build(Context, Clock.UtcNow);
    Response.DeleteAllRequestIdCookies(
                Context.Request, deleteCookieOptions);

    var cookieOptions = Options.RequestIdCookie
```

```
                            .Build(Context, Clock.UtcNow);

    Response.Cookies.Append(Options.RequestIdCookie.Name,
        Options.StringDataFormat.Protect(authnRequestId),
        cookieOptions);

    var relayState = Options.StateDataFormat.Protect(properties);
    var requestUrl = _samlService
                    .GetAuthnRequest(
                    authnRequestId,
                    relayState,
                    $"{Request.GetBaseUrl()}
                        /{Options.AssertionConsumerServiceUrl}");

    _logger.LogDebug(
        $"Method={nameof(HandleChallengeAsync)}.
        Redirecting to saml identity provider for SSO.
        Url={requestUrl}");

   Context.Response.Redirect(requestUrl, true);
    return Task.CompletedTask;
}
```

Listing 6: HandleChallengeAsync method

4.5.4 Handling logout response

The **HandleSignOut()** method receives and handles SP-initiated single logout responses from the identity provider.

It is a part of the **HandleRequestAsync** method, and it is called on every request into the web application. It compares the configured SingleLogoutServiceUrl and the HTTP Request path to decide whether to handle the HTTP request or not. It gets the SAMLResponse from the HTTP Request including the round tripped RelayState; gets the original LogoutRequestId from cookie and validates it against the InResponseTo value of the logout response; validates the SAMLResponse and uses the cookie middleware to logout the user from the ser-

vice provider. The RelayState is unprotected and the middleware returns control to the provided redirectURI.

```
private async Task<bool> HandleSignOut()
{
    if (!Request.Path.Value.EndsWith(
        Options.SingleLogoutServiceUrl,
        StringComparison.OrdinalIgnoreCase))
        return false;

    _logger.LogDebug($"Entering {nameof(HandleSignOut)}");

    if (!_httpRedirectBinding.IsValid(Context.Request))
        return false;

    var uri = new Uri(Context.Request.GetEncodedUrl());

    var response = _httpRedirectBinding
                        .GetResponse(Context.Request);
    var authenticationProperties =
        Options.StateDataFormat.Unprotect(response.RelayState)
        ?? new AuthenticationProperties();

    var initialLogoutRequestId = GetRequestId();
    if (!_samlService.IsLogoutResponseValid(
                            uri, initialLogoutRequestId))
        return false;

    await Context.SignOutAsync(
                    Options.SignOutScheme,
                    authenticationProperties);

    var cookieOptions = Options.RequestIdCookie
                                .Build(Context, Clock.UtcNow);

    Context.Response.DeleteAllRequestIdCookies(
                                Context.Request, cookieOptions);

     _logger.LogDebug(
        $"Method={nameof(HandleSignOut)}.
```

```
38          Received and handled sp initiated logout response.
39          Redirecting to {redirectUrl}");
40
41      var redirectUrl = GetRedirectUrl(authenticationProperties);
42      Context.Response.Redirect(redirectUrl, true);
43      return true;
44  }
```

Listing 7: HandleSignOut method

4.5.5 Handling SAML authentication response (SAMLResponse)

HTTP Redirect

The **HandleSignin()** method handles SAMLResponse for authentication requests. These are the HTTP Requests with the configured Assertion Consumer Service URL as the request path. This is compared with the configured value and any other request whose path does not match the configured AssertionConsumerServiceUrl value is ignored.

It uses the SamlService (**HandleHttpRedirectResponse** method) to; get the validated SAMLResponse; get and validate the assertion; read the transferred identity information as claims, and use the claims to sign-in the user for the configured SignInScheme. This creates a session cookie with a user principal containing the given claims. The RelayState is unprotected and the middleware returns control to the provided redirectURI (J). The service provider can then read the session cookie to get the transferred identity information from the identity provider as claims and use the session cookie for access control.

```
1  private async Task<bool> HandleSignIn()
2  {
3      if (!Request.Path.Value.EndsWith(
4          Options.AssertionConsumerServiceUrl,
5          StringComparison.OrdinalIgnoreCase))
6          return false;
7
8      _logger.LogDebug($"Entering {nameof(HandleSignIn)}");
```

```
9
10      if (!_httpRedirectBinding.IsValid(Context.Request))
11          return false;
12
13      var initialAuthnRequestId = GetRequestId();
14      var result = _httpRedirectBinding
15                      .GetResponse(Context.Request);
16
17      var base64EncodedSamlResponse = result.Response;
18      var assertion = _samlService.HandleHttpRedirectResponse(
19          base64EncodedSamlResponse, initialAuthnRequestId);
20
21      var authenticationProperties =
22          Options.StateDataFormat.Unprotect(result.RelayState)
23          ?? new AuthenticationProperties();
24
25      await SignIn(assertion, authenticationProperties);
26
27      var cookieOptions = Options.RequestIdCookie
28                              .Build(Context, Clock.UtcNow);
29      Response.DeleteAllRequestIdCookies(
30                          Context.Request, cookieOptions);
31
32      _logger.LogDebug(
33          $"Method={nameof(HandleSignIn)}.
34          Received and handled SSO redirect response.
35          Redirecting to {redirectUrl}");
36
37      var redirectUrl = GetRedirectUrl(authenticationProperties);
38      Context.Response.Redirect(redirectUrl, true);
39      return true;
40  }
```

Listing 8: HandleSignin method

HTTP Artifact

The middleware supports the HTTP Artifact binding by SAMLResponse (SP initiated: Redirect → Artifact binding) as described in sub section (2.3.2). The method **HandleHttpArtifact()** handles SAMLResponses for authentication requests, and ignores all HTTP Requests whose request path is not equal to the configured AssertionConsumerServiceUrl.

It receives the artifact (SAMLart) and uses the **HandleHttpArtifactResponse** method of the SamlService (J) to resolve the artifact by sending an <ArtifactResolve> message to the source identity provider. The response is an <ArtifactResponse> message containing the respective assertion. The transferred identity information is read from the validated assertion by the Saml2ClaimFactory (M), and transformed into claims which are used to create a new ClaimsPrincipal. The **ClaimsPrincipal** is signed-in to a session cookie. The RelayState is unprotected and the redirectURI from the state is read.

The middleware transfers control to the service provider by setting the HTTP Response to redirect to the given redirectURI. As described in the **HandleSignin** method, the service provider can read identity information of the subject from the user principal claims and use the created session cookie for access control.

```
1  private async Task<bool> HandleHttpArtifact()
2  {
3      if (!Request.Path.Value.EndsWith(
4          Options.AssertionConsumerServiceUrl,
5          StringComparison.OrdinalIgnoreCase))
6          return false;
7
8      _logger.LogDebug($"Entering {nameof(HandleHttpArtifact)}");
9
10     if (!_httpArtifactBinding.IsValid(Context.Request))
11         return false;
12
13     var initialAuthnRequestId = GetRequestId(); //TODO validate
       ↪  inResponseTo
14     var assertion = _samlService
15                     .HandleHttpArtifactResponse(Context.Request);
16
17     var relayState = _httpArtifactBinding
18                                 .GetRelayState(Context.Request);
```

```
19      var authenticationProperties =
20          Options.StateDataFormat.Unprotect(relayState)
21          ?? new AuthenticationProperties();
22
23      await SignIn(assertion, authenticationProperties);
24
25      var cookieOptions = Options.RequestIdCookie
26                                 .Build(Context, Clock.UtcNow);
27      Response.DeleteAllRequestIdCookies(
28                                 Context.Request, cookieOptions);
29
30      _logger.LogDebug(
31          $"Method={nameof(HandleHttpArtifact)}.
32          Received and handled SSO artifact response.
33              Redirecting to {redirectUrl}");
34
35      var redirectUrl = GetRedirectUrl(authenticationProperties);
36      Context.Response.Redirect(redirectUrl, true);
37      return true;
38  }
```

Listing 9: HandleHttpArtifact method

The cookies responsible for keeping the unique request identifiers are deleted after the SAMLResponse is received.

4.5.6 Handling sign-in (creating session cookie)

The **SignIn** method is a helper used by the **HandleSignIn()** and **HandleHttpArtifact()** methods to read identity information from the assertion and transform it into claims. It also creates a new ClaimsPrincipal using the claims, and signs-in the created principal into a session cookie.

```
1  private async Task SignIn(
2      Saml2Assertion assertion,
3      AuthenticationProperties authenticationProperties)
4  {
5      var claims = _claimFactory.Create(assertion);
6      var identity = new ClaimsIdentity(claims, Scheme.Name);
7      var principal = new ClaimsPrincipal(identity);
8
9      await Context.SignInAsync(
10         Options.SignInScheme,
11             principal,
12               authenticationProperties);
13 }
```

Listing 10: Signin method

4.5.7 Unique SAML request identifiers

The helper method **CreateUniqueId** is used to create unique identifiers for each SAML request. These are necessary for the validation of received SAMLResponses. The value is validated against the InResponseTo value of the SAMLResponse, and the message is discarded if these do not match. This helps in protection against automated attacks, and the integrity of the request-response message exchange.

```
1  private static string CreateUniqueId(int length = 32)
2  {
3      var bytes = new byte[length];
4      using (var randomNumberGenerator
5                      = RandomNumberGenerator.Create())
6      {
7          randomNumberGenerator.GetBytes(bytes);
8          var hex = new StringBuilder(bytes.Length * 2);
9          foreach (var b in bytes)
10             hex.AppendFormat("{0:x2}", b);
11
12         return hex.ToString();
13     }
14 }
```

Listing 11: CreateUniqueId method

4.5.8 Creating SAML 2.0 messages

The middleware creates three specific SAML 2.0 protocol messages during authentication and signout. These include authentication requests, logout requests and logout responses. Appendix L shows the **Saml2MessageFactory** class with the functions responsible for creating the protocol messages. Calling the function **GetXml()** on the resulting objects automatically creates the XML version of the request by serializing the values into XML strings.

4.6 Configuration

4.6.1 AuthenticationBuilder

The **AuthenticationBuilder** class gives an extension method **AddScheme** which is used to configure custom authentication handlers into the HTTP pipeline as authentication schemes. An authentication scheme represents a unique identifier for the current authentication middleware. This adds the custom authentication middleware into the ASP.NET Core authentication pipeline [68].

Listing 12 shows how the **Saml2Handler** is added into the application's authentication middleware pipeline. The complete implementation is given in appendix D.

```
public static AuthenticationBuilder AddSaml(
    this AuthenticationBuilder builder,
    string authenticationScheme,
    string displayName,
    Action<Saml2Options> configureOptions)
{
    builder.Services.TryAddEnumerable(

    ServiceDescriptor.Singleton<
            IPostConfigureOptions<Saml2Options>,
                    Saml2PostConfigureOptions>());

    return builder.AddScheme<Saml2Options, Saml2Handler>(
            authenticationScheme, displayName, configureOptions);
}
```

Listing 12: Configuring the authenticationbuilder

The **IPostConfigureOptions** offers a way of adding configuration values to the authentication handler after all other configurations have occurred. This is useful, especially when some configuration values have to be instantiated before use. Objects like the state **DataProtectionProvider** instance fall into this category [69]. Appendix E shows the post configuration of the **DataProtectionProvider** which is used in the authentication middleware to protect the confidentiality of cookies and RelaysSate.

4.6.2 Startup services

The middleware requires several services to be instantiated during startup. These services are used by the authentication handler to perform specific functions. The authentication handler gets instances of these services from the dependency injection (DI) container which is configured during startup. DI is supported by default in ASP.NET Core and the framework exposes an **IServiceCollection** extension which is used to add services into the inbuilt DI container [70].

```
1  private static void AddRequiredServices(
2      this IServiceCollection services)
3  {
4      services.AddOptions();
5      services.TryAddSingleton(resolver =>
6          resolver.GetRequiredService<
7                          IOptions<Saml2Configuration>>().Value);
8
9      services.TryAddTransient<ISaml2Validator, Saml2Validator>();
10     services.TryAddTransient<ISaml2ClaimFactory,
11                                              Saml2ClaimFactory>();
12     services.TryAddTransient<ISamlProvider, SamlProvider>();
13     services.TryAddTransient<ISaml2MessageFactory,
14                                          Saml2MessageFactory>();
15     services.TryAddTransient<ISignatureProviderFactory,
16                                      SignatureProviderFactory>();
17
18     services.TryAddTransient<IHttpRedirectBinding,
19                                          HttpRedirectBinding>();
20     services.TryAddTransient<IHttpArtifactBinding,
21                                          HttpArtifactBinding>();
22     services.TryAddTransient<ISamlService, SamlService>();
23 }
```

Listing 13: Configuring start-up services

Listing 13 shows how the required services are added into the DI container.

Other dependencies that need to be configured during startup include but are not limited to signing certificates. The middleware requires signing certificates

for both SP and IDP during runtime. These are used for digitally signing, encrypting and verifying protocol messages as required by the SAML 2.0 specifications [7]. Three different ways of reading and configuring the signing certificates are implemented. These can be added as a file in the file system, a thumbprint or certificate name from the certificate store. Appendix C gives a detailed overview of how start-up services are configured in the middleware.

4.6.3 Bringing it all together

The above sections have given and described the different components making up the authentication middleware. This section describes how these components are put together, work together as a whole, to provide a fully functioning SAML 2.0 authentication middleware for ASP.NET Core. It explains, the minimal configuration required for the middleware as used in the **DemoWebApp**.

ASP.NET Core applications use an application start-up class called **Startup**. This is the entry point of any AS.NET Core based web application, and it runs once when the application starts. The class is made of of two methods **ConfigureServices** which is used to configure application services, and the **Configure** method, which is used to configure the application request processing pipeline (HTTP pipeline) [71].

The middleware is configured within the **ConfigureServices** method with a few lines of code. Listing 14, shows how to configure the middleware in **Startup**. A brief description for what each line does is given.

```
1  public void ConfigureServices(IServiceCollection services)
2  {
3      // Omitted code
4
5      // Add Saml2.Authentication.Core configuration values
6      services.Configure<Saml2Configuration>(
7                              Configuration.GetSection("Saml2"));
8
9      // Add Saml2.Authentication.Core services to the container
10     services.AddSaml();
11
12     // Add signing certificates by thumbprint
13     services.AddSigningCertificates(Configuration
14     ["Saml2:ServiceProviderConfiguration:
15                                  SigningCertificateThumprint"],
```

```
16      Configuration
17       ["Saml2:IdentityProviderConfiguration:
18                                  SigningCertificateThumprint"]);
19
20      // Add Saml2.Authentication.Core authentication handler to
21      // ASP.NET Core authentication middlewares with a named
         ↪ SignInScheme
22      // and AuthenticationScheme.
23      // Add and configure session cookie.
24      services.AddAuthentication()
25          .AddCookie()
26          .AddSaml(options =>
27          {
28              options.SignInScheme = "SignInSchemeName";
29              options.AuthenticationScheme =
30                                      "AutthenticationSchemeName";
31          });
32
33      // Omitted code
34  }
```

Listing 14: Configuring Saml2.Authentication.Core in startup

In listing 14, we see that the SAML configuration values are read from a JSON configuration file, with a section called "Saml2". This is usually done in the **appsettings.json** file. The settings used during testing of the **DemoWebApp** are given in listing 15. See appendices G, H and I for all the possible SAML 2.0 configuration variables.

```
1   {
2      "Saml2": {
3       "ForceAuth": "false",
4       "IsPassive": "false",
5       "IdentityProviderConfiguration": {
6         "EntityId": "https://saml2login.my.salesforce.com",
7         "Name": "samllogin.my.salesforce.com",
8         "SigningCertificateThumprint":
```

```
                "a69d709f32f14b484f1a46b3d514b42388a0c3f6",
      "SingleSignOnService":
        "https://saml2login.my.salesforce.com/idp/endpoint
                                              /HttpRedirect",
      "SingleSignOutService":
        "https://saml2login.my.salesforce.com/services/auth/
                                              idp/saml2/logout",
      "ProtocolBinding":
            "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
    },
    "ServiceProviderConfiguration": {
      "EntityId": "https://localhost:44344",
      "Name": "Saml2.auth",
      "SigningCertificateThumprint":
            "83331e94db6a841ed921ff86e41624a6eb78c8d1"
    }
  }
}
```

Listing 15: Appsettings configuration example

4.7 Triggering authentication

The SAML 2.0 authentication middleware can be invoked by creating an action (**IActionResult**), that returns an instance of the **ChallengeResult** class. The authentication scheme name and properties (**AuthenticationProperties**) should also be supplied. This is called "challenging" the middleware. When challenged, the middleware triggers the SAML 2.0 authentication handler (**HandleChallengeAsync** method) as described in section 4.5 to handle the challenge. This will create a SAML authentication request (AuthnRequest) and redirect to the configured IDP for authentication [72].

The listing below gives an example of code "challenging" the middleware.

```
1  public IActionResult ExternalLogin(string returnUrl = null)
2  {
3      var provider = "AuthenticationSchemeName";
4
5      // Request a redirect to the external login provider.
6      var redirectUrl = Url.Action(nameof(ExternalLoginCallback),
7                        "Account", new { returnUrl });
8
9      var properties = _signInManager
10                     .ConfigureExternalAuthenticationProperties(
11                         provider,
12                             redirectUrl);
13
14     return Challenge(properties, provider);
15 }
```

Listing 16: Triggering authentication

The action in listing 16 defines a redirectUrl variable which is added into the authentication properties object. The object is passed into to the middleware and it represents the RelayState value that is protected and round tripped to the identity provider. The redirectUrl is the callback action that the middleware will redirect to after authentication.

An example callback action is given below.

```
public async Task<IActionResult> ExternalLoginCallback(
    string returnUrl = null)
{
    var info = await _signInManager.GetExternalLoginInfoAsync();
    if (info == null)
    {
        return RedirectToAction(nameof(Login));
    }

    // Sign in the user with this external login provider.
    var result = await _signInManager
                .ExternalLoginSignInAsync(
                info.LoginProvider,
                info.ProviderKey,
                isPersistent: false,
                bypassTwoFactor: true);

    if (result.Succeeded)
    {
        _logger.LogInformation("User logged in with
                               {Name} provider.",
                                info.LoginProvider);
        return RedirectToLocal(returnUrl);
    }
}
```

Listing 17: External authentication callback

From the example in listing 16, the middleware redirects to the **ExternalLoginCallback** action after authentication. The line

```
var info = await _signInManager.GetExternalLoginInfoAsync();
```

reads the session cookie that was set by the authentication middleware. This contains a user principal object with the transferred identity information from the identity provider. This example goes on to sign-in the received user principal in to a new cookie which is used for access control.

4.8 Triggering signout

Triggering signout is done the same way as triggering authentication, except that the action triggering signout calls the **HttpContext.SignOutAsync** method. The authentication scheme name to signout from and the **AuthenticationProperties** object [73] are required. This triggers the **SignOutAsync** method of the authentication handler which creates a logout request for the currently logged-in subject and redirects to the identity provider. The following example shows an action that triggers signout using the SAM 2.0 authentication middleware.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task Logout()
{
    var redirectUrl = Url.Action(
                        nameof(ExternalLogoutCallback),
                            "Account");

    var properties = _signInManager
                .ConfigureExternalAuthenticationProperties(
                                    "AuthenticationSchemeName",
                                        redirectUrl);

    await HttpContext.SignOutAsync(
                        "AuthenticationSchemeName",
                                    properties);
}
```

Listing 18: Triggering signout

4.9 Ethical and legal considerations

A part of this project uses classes ported from the open source project OIOSAML.Net 2.0 made by digitaliser.dk [15]. The OIOSAML.Net 2.0 [15] library is released under the Mozilla Public Licence. The SAML 2.0 authentication middleware for ASP.NET Core is also released under the same licence [66]. This license allows the author to grant modification rights to any contributor of the project. It allows for use, modification, sub licensing, distribution of both source code and the modified source code [17].

5 Results

This chapter analyses the results captured from testing the created authentication middleware to verify the research questions, requirements and specifications formulated during the project start. The results are based on an evaluation of the authentication middleware using a demo ASP.NET web application (**DemoWebApp**), configured to use a real world SAML identity provider (`https://www.salesforce.com`) for authentication and managing identities.

5.1 Configuration

Salesforce offers the possibility of creating and configuring it as a SAML 2.0 identity provider. This allows third party web applications to give users access to resources using identities created and maintained by Salesforce, and transferred to the web application using the SAML 2.0 framework. For this to work, configuration has to be done at `salesforce.com`, to enable Salesforce as a SAML identity provider, and also configure the service provider as an application connected to the identity provider [74].

5.1.1 Configuring salesforce as a SAML identity provider

To configure the identity provider, the domain name for the identity provider (`https://saml2login.my.salesforce.com/`) was created and a self-signed certificate with the SHA-256 signature algorithm was generated. These were added to the identity provider's configuration settings in salesforce.com [74].

Figure 10 shows the configured Salesforce identity provider with entity id (issuer) `https://saml2login.my.salesforce.com`. Appendix S shows the identity provider's metadata.

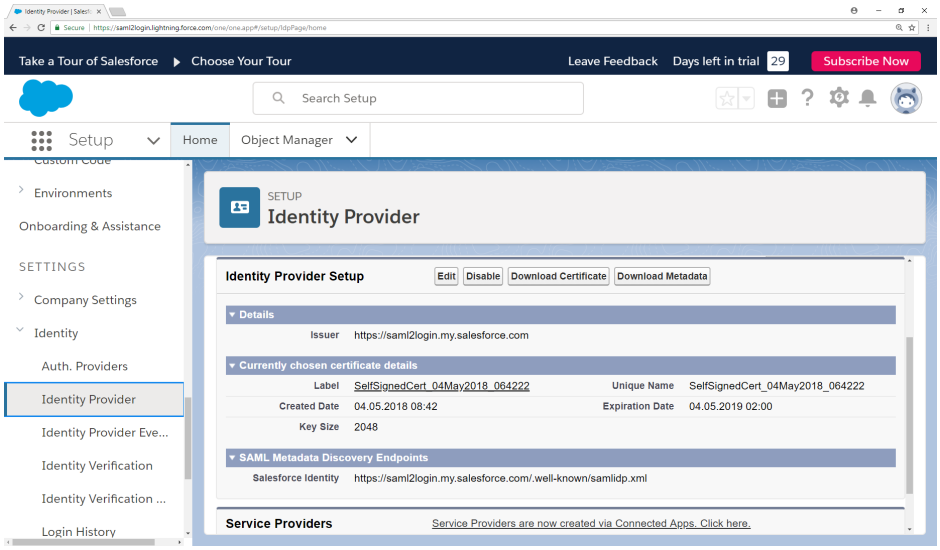

Figure 10: Identity provider configuration in Salesforce

5.1.2 Configuring DemoWebApp as the service provider

The **DemoWebApp** was added in Salesforce as a service provider application, and connected to the configured SAML identity provider. All the required SAML 2.0 configuration values for service providers were added in the web app settings. Some of these are described below [75].

Start Url The web application's start URL.

Entity Id The web application's unique identifier.

ACS URL The web application's Assertion Consumer Service URL.

Single Logout URL The web application's single logout endpoint.

Issuer The SAML identity provider's entity identifier.

Figure 11 shows the **DemoWebApp** application's configuration as a service provider that is connected to the identity provider (issuer) with entity id `https://saml2login.my.salesforce.com`.

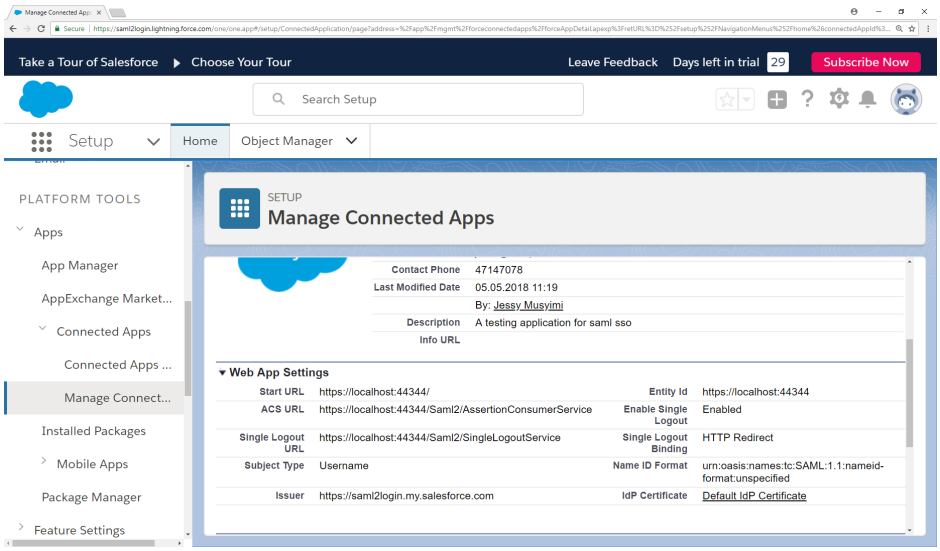

Figure 11: Service provider configuration in Salesforce

After configuring the **DemoWebApp** as a service provider in Salesforce, it was also necessary to configure the **DemoWebApp** application to use Salesforce as the SAML identity provider. This was done in the application settings file as seen in listing 15.

5.1.3 Managing identities

An identity with the username jkmu@XXXXXX.com, and federation id 123456789 was added and connected to the service provider **DemoWebApp**. This was done in the user management portal at Salesforce.

The figure below shows the Identity information of the added user.

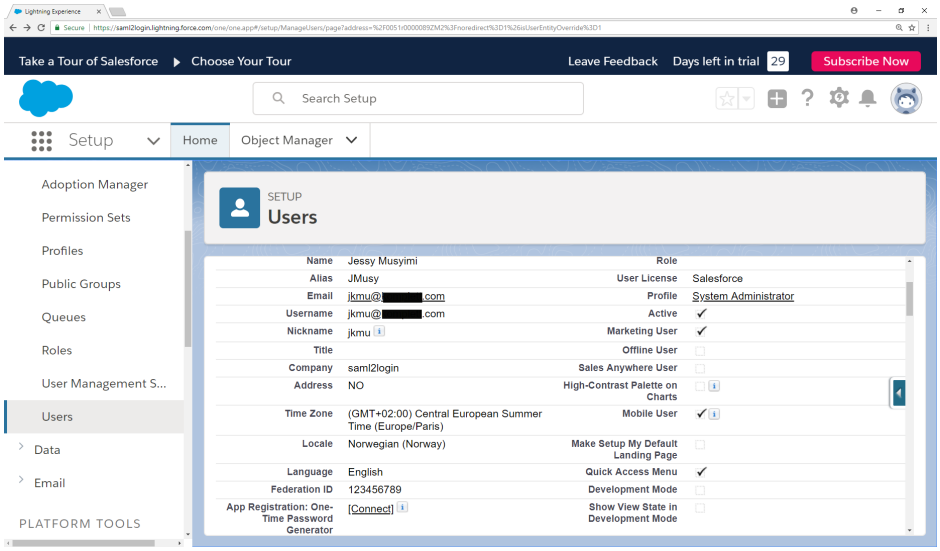

Figure 12: Users in salesforce

5.2 Verification of results

The results were verified using case studies which where designed to cover the overall SAML 2.0 signin and signout using the authentication middleware, as configured in the **DemoWebApp**. The aim was to verify the fulfillment of the research questions identified in section 1.7. For each case study, the middleware was configured to use the HTTP redirect binding. A brief discussion of how the middleware solves each of the research question is given together with the achieved test results.

5.2.1 Case study 1: Initiating authentication (SAMLRequest)

This case study verifies research question 1. The service provider initiates authentication using an action that returns an instance of the **ChallengeResult** class. The class is instantiated with the authentication schene name given to the authentication middleware, and a set of given properties. The **HandleChallengeAsync** component receives the challenge and handles it by creating an authentication request and redirecting to the external identity provider for login . This is discussed in sections 4.5.3 and 4.7.

Results

Triggering authentication using the middleware created an authentication request (AuthnRequest) with a unique identifier as shown in listing 19.

Listing 19: AuthnRequest

```
<?xml version="1.0"?>
<q1:AuthnRequest
    ID="b3dc5923fff09491d9f66ec6000c"
    Version="2.0"
    IssueInstant="2018-05-05T15:55:03.7250138Z"
    Destination=
    "https://saml2login.my.salesforce.com/../HttpRedirect"
        ForceAuthn="false"
        IsPassive="false"
        ProtocolBinding=
        "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
        AssertionConsumerServiceURL=
        "https://localhost:44344/Saml2/AssertionConsumerService"
```

```
        xmlns:q1="urn:oasis:names:tc:SAML:2.0:protocol">
    <Issuer xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
        https://localhost:44344
    </Issuer>
    <Conditions xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
        <AudienceRestriction>
          <Audience>https://localhost:44344</Audience>
        </AudienceRestriction>
    </Conditions>
</q1:AuthnRequest>
```

The middleware created an authentication request URL by concatenating the SAMLRequest (AuthnRequest), **RelayState**, **SignAlg** (signing algorithm), **Signature** and the **destination URL** (single sign-on endpoint of the identity provider). The **SAMLRequest** and the **RelayState** were deflate encoded and URL encoded. The **SignAlg** was URL encoded while the signature was converted into a base 64 string and URL encoded [7].

Figure 13 shows the created authentication request URL.


```
https://saml2login.my.salesforce.com/idp/endpoint/HttpRedirect SAMLRequest =
lJLdauMwEIXvF/oORve15N/EwknJblka6EJIsnvROOUaNwJbSjRy2L595SRNlkIKCzKCmTlnvvGofvjbtdEBHGprJiSJGXmY3n2r9wmf9X5r
lrDvAXOOf5yQTaZkUaVZOzSsyqtEVU1ZgiwZY7IR5SjPynBXucqLkapkUyWVqOQ4fJsmV1lKoj8ffdLQJ5oj9jA36IXxIcSS8TOrwlknBS8K
zrJ4lBYsycYvJHoMENoIf1Rvvd8hpxRF16atfdUm7t5iFC1gY52EWNqOarWjYNTOauPpU1AsQWkHOpPo51BOHG9CGtEiDCwLgagPcIksnPVW
2va7Nkqb1wnpneFWoEZuRAfIveSr2a9nHmbhm1MR8qf1enF/7TRDBDdA/7AG+w7cCtxBS/i9fL6OOVop2q1Fz/M8y3O6Gsait6QkChszyPfJ
1Oi7Mz8J64yi+viz3Un7tVB8NCbTG4Q1PbmdnAOfOkM9/qf7oA76Wa8OGAnLsGKn5ZA7p/5J3ka5VJztLoHPfjW9kg5PnH5649N3AAAA//8D
AA== RelayState =
BMHJcoMgAADQL3JGY3A5ohgXXAI6TeDCKC41dmLdaunX9z2/R4lTnMMOVuVwkHyZSDm/EZu8o5acYXs6cybaR2TQb/W8kD+HDe83ofOlOreu
Go4tzLSxug6UFKOBR3Igud4t4SaWzuwok5Pvom7CXJ+zxsD1p938fHiC3nOTElyCpFQvFyoo0+daxd2keRoE5kKcsA/m9Jot4RqJB9r6Fjfu
ba2Xbb/xPAeuyMrdk4q+NAtw3KUqCMpwplFcjHasCZ2SfwAAAP//AwA= SigAlg = http://www.w3.org/2001/04/xmldsig-
more#rsa-sha256 Signature =
mV1dT4yC43xiuV81d69XEmiOtlMm4lh4raAVz46K4SdwNMotoq4Wa1IFB8KomPqcVKJCQX93iBNtOqYj9DScT9UIb5PiOFIuGyFNndrUdSbf
VC/iDwajcCNU4x4X1hKSZsQrHHUaGsOuOpKtaoZrWW79daVSGNlvTnWXGUo3tfquyWsXhIOfkt8LHyaofajr0F77UgDyRjYRAtBgbXMDNMBX
Lkxir3J8arql6TjZ+5iVkgtx4kj5YTK1Vtjw3lOoXOdlmtE/YRbRpjkCqJGzHIm3vyow+wVkc3t77BIyJA7L1KEFY3eOq2HsoDCK5SeZBGhp
hg9RvJf7jQkQRjKZVg==
```


Figure 13: Authentication request URL

Setting the created authentication request URL as the location to redirect to in the ASP.NET Core HTTP response pipeline resulted to a redirect to the identity provider's login page [76].

5.2.2 Case study 2: Handling authentication response (SAMLResponse)

This case study verifies research question 2. Authentication responses are handled by the **HandleRequestAsync** component 4.5.1. This component is called on

every HTTP request that comes through the middleware. To determine whether to handle the request, the component has a function **HandleSignIn** (8) which listens to HTTP requests with the configured Assertion Consumer Service URL as the request path. A match means that this is an authentication response. The method initiates verification of the response, reading of the identity data from the received assertion and signs in the user into a local session cookie.

Results

Upon entering the user credentials at the identity providers login page, the middleware received an authentication response in an embedded html form. The form contained the roundtripped RelayState value as added in the authentication request, and a base 64 encoded SAMLResponse. Appendix T shows the actual SAMLResponse as received while listing 20 shows the received RelayState, whose value is equivalent to the value in the authentication request URL in figure 13.

```
BMHJcoMgAADQL3JGY3A5ohgXXAI6TeDCKC41dmLdaunX9z2/
R4lTnMMOVuVwkHyZSDm/EZu8o5acYXs6cybaR2TQb/W8kD+
HDe83ofOlOreuGo4tzLSxug6UFKOBR3Igud4t4SaWzuwok5Pvom7CXJ+
zxsD1p938fHiC3nOTElyCpFQvFyoo0+daxd2keRoE5kKcsA/
m9Jot4RqJB9r6Fjfuba2Xbb/xPAeuyMrdk4q+NAtw3KUqCMpwplFcjHasCZ2SfwAAAP/
/AwA=
```

Listing 20: RelayState from authentication response

The SAMLResponse was then decoded and validated. The result was the authentication response in form of an XML assertion. Appendix U, shows the decoded SAMLResponse while listing 21 shows the actual assertion.

Listing 21: Validated assertion

```
<saml:Assertion
 ↪  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 ↪  ID="_f580b191d80f081b766398a5738360aa1525535780073"
IssueInstant="2018-05-05T15:56:20.073Z" Version="2.0">
<saml:Issuer
 ↪  Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
https://saml2login.my.salesforce.com</saml:Issuer>
```

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod
 ↪  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
<ds:SignatureMethod
 ↪  Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
<ds:Reference
 ↪  URI="#_f580b191d80f081b766398a5738360aa1525535780073">
<ds:Transforms>
<ds:Transform Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
<ds:Transform
 ↪  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ec:InclusiveNamespaces
 ↪  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"
 ↪  PrefixList="ds saml xs xsi" />
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod
 ↪  Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<ds:DigestValue>YKJC6DmbJUociDCUODISdHw9RO8=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
                <ds:SignatureValue>
HlSyOlGGSBys3VGdpj7iNmO+PZpN9OwN/vhPA4wZse5ItWdXF+V4mnLrK8L+
RXBHYwg3wWfzuY9ZJdyX7nwZwI9jTMWEvREkbQS4SIvCFa89pNMZs7qiOfLf
jRfz9B9OLeFdeNabw3A67ouVsC8Ss4KABkPNz1ZNydcBSqSv2AKEUKPLueGz
RBBKhDoDV7LvYczZ/uNkRE1QPPzU926spRyU4ASBdj1bDS1bhIB9e7XePE7I
IsGF9ZC7ASC7SVoGibRsRLk8wlZzn+yiFkFU9K2YeeN79Grh5XJZ4geo7EtHKv5R
AAe3+ObU/scQAZLispNmY/5ky20fwSVosArpvw==
                </ds:SignatureValue>
                <ds:KeyInfo>
                        <ds:X509Data>
                <ds:X509Certificate>
MIIErDCCA5SgAwIBAgIOAWMp4yXaAAAAAAcFp9YwDQYJKoZIhvcNAQELBQAwgZAx
KDAmBgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAxOF8wNjQyMjIxGDAWBgNV
BAsMDzAwRDFyMDAwMDAyQ3YzcjEXMBUGA1UECgwOU2FsZXNmb3JjZS5jb20xFjAU
BgNVBAcMDVNhbiBGcmFuY2lzY28xCzAJBgNVBAgMAkNBMQwwCgYDVQQGEwNVU0Ew
HhcNMTgwNTA0MDY0MjIyWhcNMTkwNTA0MDAwMDAwWjCBkDEoMCYGA1UEAwwfU2Vs
ZlNpZ25lZENlcnRfMDRNYXkyMDE4XzA2NDIyMjEYMBYGA1UECwwPMDBEMXIw
```

```
MDAwMDJDdjNyMRcwFQYDVQQKDA5TYWxlc2ZvcmNlLmNvbTEWMBQGA1UEBwwNU2F
uIEZyYW5jaXNjbzELMAkGA1UECAwCQ0ExDDAKBgNVBAYTA1VTQTCCASIwDQYJKo
ZIhvcNAQEBBQADggEPADCCAQoCggEBAKIF3ftUDKu5JKhFxkhRwCg9jZaPqJyXQ
0Q9vuaQ+CDLn+Zya4DZWaOEMIVPaL5b6ArvrT9W0QGbDNWRdyoHeL2iaWHuKU+
kFe+pvwe1H4sSiqioRRnK8KY4xxmDiajIPISEgcPUnONNd39zWvar6Q1/Iwytz
QvOQr9YvH5nvGmgcjeqXbNuAhOUfaCn/Qt+ssiCuMY+WRr+iDsYOfMyZbj75+2
Eb03N+HZ3C+xJ5YsseNzgPr5zj6ktH93llOhcO5C/OMl/l7HdFmFgkNMKGDKYA
4qlNMIlMqDrRrSC8bhcxnGJ/7L40kdw8CWg+XqKTRwVtu7377K+uo8AMuBCCEE
CAwEAAaOCAQAwgf0wHQYDVR0OBBYEFPIqUh1nbrZhKxSjg5z2LyiGMVqgMA8GA
1UdEwEB/wQFMAMBAf8wgcoGA1UdIwSBwjCBv4AU8ipSHWdutmErFKODnPYvKIY
xWqChgZakgZMwgZAxKDAmBgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAx
OF8wNjQyMjIxGDAWBgNVBAsMDzAwRDFyMDAwMDAyQ3YzcjEXMBUGA1UECgwOU
2FsZXNmb3JjZS5jb20xFjAUBgNVBAcMDVNhbiBGcmFuY2lzY28xCzAJBgNVBA
gMAkNBMQswCgYDVQQGEwNVU0GCDgFjKeMl2gAAAAAHBafWMA0GCSqGSIb3DQE
BCwUAA4IBAQCeTzfDa72aXQ60wrWCYuEzB75Kj/3BaAAHSjb22jbcNVQPncEP
phoP+baMEHDlUf5XNDZx5octD3hrKwGe14wcmr8LaBFrRDwwomVepP84wuz0
Hoqi+qrvTpFZyhDsvyyHEKTaNDNeQtkzbc3VCqsUW5KBYIXukys26h1pmQDm
HcMmkjMErY1yYAZu7mhM965ZDT90W1x8spWT/p82p/Ak6mjW/GcxPQnM4j9Nv
0JucOf1SfNjTAIDTgonPNf2PoNUmgLgXcan6S38bZBCRWfdC4f2GohfiOuScD
O+4pMFJ9yv/zm2ldh/j0cFe2fhknbCu2zXeWCnFgx31kE5
ZzSj</ds:X509Certificate>
                        </ds:X509Data>
                </ds:KeyInfo>
        </ds:Signature>
<saml:Subject>
<saml:NameID
Format=
"urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
jkmu@XXXXXXXX.com</saml:NameID>
<saml:SubjectConfirmation
 ↪  Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
<saml:SubjectConfirmationData
InResponseTo="b3dc5923fff09491d9f66ec6000c"
 ↪  NotOnOrAfter="2018-05-05T16:01:20.073Z"
Recipient=
"https://localhost:44344/Saml2/AssertionConsumerService" />
</saml:SubjectConfirmation>
</saml:Subject>
<saml:Conditions NotBefore="2018-05-05T15:55:50.073Z"
 ↪  NotOnOrAfter="2018-05-05T16:01:20.073Z">
```

```
<saml:AudienceRestriction>
        <saml:Audience>https://localhost:44344</saml:Audience>
</saml:AudienceRestriction>
</saml:Conditions>
<saml:AuthnStatement AuthnInstant="2018-05-05T15:56:20.073Z">
<saml:AuthnContext>
<saml:AuthnContextClassRef>
    urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified
</saml:AuthnContextClassRef>
</saml:AuthnContext>
</saml:AuthnStatement>
<saml:AttributeStatement>
<saml:Attribute
Name="userId"
NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
<saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:anyType">0051r0000089ZM2</saml:AttributeValue>
</saml:Attribute>
<saml:Attribute
Name="username"
NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
<saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 ↪  xsi:type="xs:anyType">jkmu@XXXXXXXX.com</saml:AttributeValue>
</saml:Attribute>
<saml:Attribute
Name="email"
NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
<saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:anyType">jkmu@XXXXXXXX.com</saml:AttributeValue>
</saml:Attribute>
<saml:Attribute
Name="is_portal_user"
NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
```

```
<saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:anyType">false</saml:AttributeValue>
</saml:Attribute>
</saml:AttributeStatement>
</saml:Assertion>
```

From the listing above, we can see that the identity information of the user was transferred as attribute statements in the assertion. The identity information included userId, username and email. This information was used to create a user principal object in ASP.NET Core, by transforming the attribute statements into claims. The **ClaimsPrincipal** (user) was then signed-in to create a session cookie (2).

5.2.3 Case study 3: Maintaining session

This case study verifies research question 3. The identity information received from the identity provider needs to be persisted into a session for access control. Case study 2 describes how the middleware creates a user from the received identity information. This is persisted into a local session cookie by the **SignIn** method (4.5.6).

Results

Figure 14 is an example of a callback endpoint where the authentication middleware returned control after a successful authentication. It shows how the service provider can read the transferred identity information as claims on the authenticated user principal. Note that, the claims have the same identity information as the attribute statements in the received assertion in listing 21.

Based on the individual requirements of the service provider, the developer can choose to continue using the already set session cookie, or, read the identity information from the session cookie, create a local identity, probably with extra information, connect these identities and sign-in the new user principal.

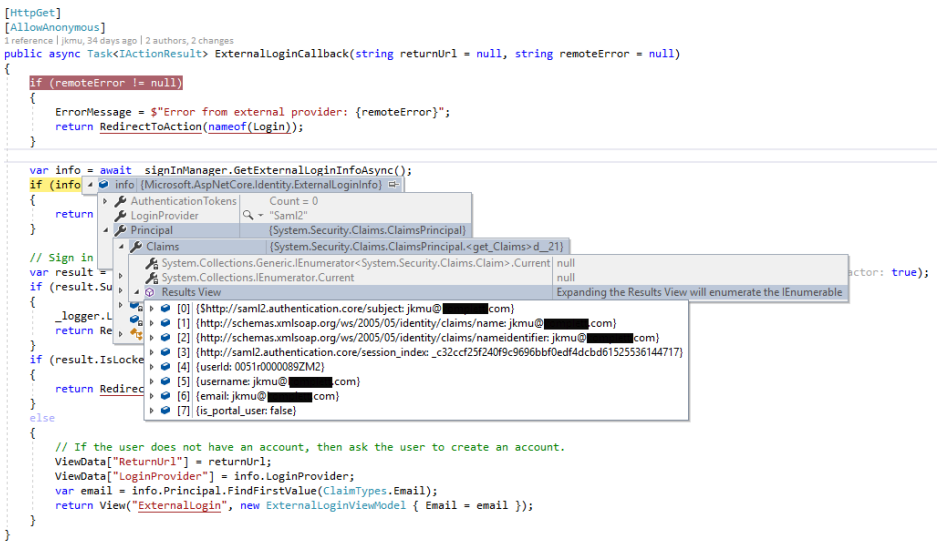

Figure 14: User principal and claims

5.2.4 Case study 4: SP-initiated logout

This case study verifies research question 4. Service provider logout requests are initiated by calling the **HttpContext.SignOutAsync** method from ASP.NET Core, with the authentication handler's scheme name (4.8). This triggers the **SignOutAsync** component of the authentication middleware which creates a logout request and transfers control to the identity provider for logout (4.5.2).

Results

After clicking logout on the **DemoWebApp** application, the SignOutAsync method of the authentication handler was triggered. This created a **LogoutRequest** XML with a unique logout identifier. The current session identifier (SessionIndex) and subject to logout was added to the request. The resulting XML string was deflate encoded, URL encoded, signed (same as the authentication request) and added as a SAMLRequest parameter to the identity provider's single logout endpoint. The resulting logout request URL was set as the location to redirect to in the ASP.NET Core HTTP response [76]. The full logout request URL is given in figure 15, while listing 22 shows the **LogoutRequest** XML message.

Listing 22: LogoutRequest message

```
<?xml version="1.0"?>
<q1:LogoutRequest
ID=" eb45015c9e787c7ad186626dbb90de"
Version="2.0"
IssueInstant="2018-05-05T16:30:52.0315094Z"
Destination="https://saml2login.my.salesforce.com/../logout"
Reason="urn:oasis:names:tc:SAML:2.0:logout:user"
    xmlns:q1="urn:oasis:names:tc:SAML:2.0:protocol">
<Issuer xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
    https://localhost:44344
</Issuer>
<NameID xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
    jkmu@XXXXXXXX.com
</NameID>
<q1:SessionIndex>
_f404918957093a3e80346cf72542aadc1525537793123
</q1:SessionIndex>
```

```
</q1:LogoutRequest>
```

```
https://saml2login.my.salesforce.com/services/auth/idp/saml2/logout SAMLRequest =
lJK7btwwEEX7AP4HQn2Wbz0IeZOA2whwUtiGizQBRY28iilxVOMZzt+HWsGFXRgIwGow91zemalvXkdPXmDGIUzXGd+x7GZ/9aU+c3MbnsIS
7+C8AEbSHK4zaJVmXLsKirJwhe14meci79q2Yh1oKAvopZCdznNlS1Y53fJWOFkUrOhV32bk8c1IJCPSIC7QTBjtFFOJ8fIr0+k98NxIZnRq
klyzSv3KyCF9YphsvKiPMZ7QUIp29MKHp2HajX93aD1gH2YHOxdGijC/DA6Q2iUe6dCdtnbqL7kycgcWV9gyTyZYHNBMdgQOOZn77z9uTXI3
W69ZEisjaVITmjP/XHKaQwwu+CyNkZD6knHetJ8LLSaXNV+2f8vng7P+GDAapaRSNd1oG/lnIjSH/yT/eR6Xb89hPHmIcR1TTTfOxkxrvwdc
V9RMHbzuf/eKqYqXlS5YJa2EkkmVu74QWglrO8e1ODotuJJcyJp+1KdLWmvvTmn/DwAA//8DAA== RelayState =
BMHJcoIwAADQL+qMLFE4InHKYlgkhIZLRwixEqLDGuLX9z2fw8hJ1WMBk3ZqEAOW1M4eUHFej2VIJsRtlaUfERHxrXWcPF1dXZsfMydE3uOU
e6Beccy21ui+CrX2t6ZmWRlawb0f2CEJKtFKJrd+/PMl3OgCLip267ka8+UUm2bPNwasHf3yDptZiBXd3da+pPKpFUpk4ahX/paFmDsj8rqT
133wtSANCW9DCwJO3n2iOIMbcCQlnV4TXGea2Lw5rkT8AwAA//8DAA== SigAlg = http://www.w3.org/2001/04/xmldsig-
more#rsa-sha256 Signature =
h71UoAjyf3dBSUXquvnKOSXkmdAwpIoMVz82met8HOBPa+JI9HG72tF747PnEOhy19y3jmELuT6Qsmc1lLPVlVCBj1UhhsJsZQFMXLtaEgzU
UStPzzjdmdeTVU+ZKVUJkUqmogT5t3MVwTs16Z7UW+3PqulQFIKrhd3D8zE3k5dGR1AW84oLWFmerlRfPUuws7D6g6pEHarqy6wubNAHc8Ml
iI6wqJy1LuhsHVFU981H6ZuGQg8Il/Ks+Xrp5kUEG4nAofPkvtL179DbWqH5zIj4C4wNgYexnd11DsmqeRW1vqZxTI41B1U1AU5uMsg6n2rH
5TbhT9AAToUItMlG3w==
```

Figure 15: LogutRequest URL

5.2.5 Case study 5: Handling SP-initiated logout response

This case study verifies research question 5. As described in section 4.5.4, the **HandleSignOut()** method handles logout responses. Being a part of the **HandleRequestAsync** component which is called on every request, it checks for requests that target the single logout service and disregards the rest. The method destroys the session cookie for valid responses by calling the ASP.NET Core method **HttpContext.SignOutAsync** with the specified SignOutScheme of the middleware (7).

Results

The logout request in case study 4 triggered a redirect to the identity provider, and a redirect back from the identity provider with a SAMLResponse (**LogoutResponse**). The middleware validated the LogoutResponse, removed the local session cookie and redirected back to the **DemoWebApp** application. Figure 16 shows the received logout response while listing 22 shows the validated logout response message.

```
SAMLResponse =
fZJBa9wwEIX/itE9tixbsiXWGOJCIZBe6m0OvRRpPN0YZGnxyKH991V2WZpAidBBg+a9N3zS7vb34otXXGmOYWB1yVmBAeI0h+PAvh++3PTs
dr8ju/iTeYrHuKVvSKcYCIusDGTOVwPb1mCipZlMsAuSSWDGu69PRpTcnNaYIkTPigekNAebzmEvKZ3IVJWPYP1LpGTatmnbasyOohrzBB4v
kSOurzMgKx4fBvazgamzqgOnGs5rrjoOqrVSTbYXQqqmlkLKpped0J3KmnCd+BAH5izkRsirnSYtQbgaUPO+dk4KrRunUINWHBwowZ3s80gd
tIJb1/X5kP2INnwMlGxIAxO87m+4zPtQK9MKIOWpO/2DFc9XqBkBuyAOZ+36jtzn4CwRrm+w2P4K6OOkfDzOoVz+lGQ90q+4ApYQl131LuT6
aGOyaaOP1X2csHi2fsPP8+ncbS78JlxZtb9k/LOt/vc19n8B RelayState =
BMHbkoIgAADQL9qZokB6FFHJrdFCJXlp3LyQ7uZlU9Cv7xynogEOdf2G44IlDH53dMGGZS35141hTRs3Z8olUhTPiyUJ6xUf1O7H56RiqZAX
a03DjfcjlMeMS8obUiJy9h4C9sqj+OPUkJlCl7Do41q+egqryXKPKND51ynUvS8kBOn+Gh6DUfssa+3uuQHgxiflfQ/pn7+OXrSrpAuu5YGg
oppyqAYyd/ywTXJyh90cXfstPsX2cnz6wdsJQR5dkDD4AwAA//8DAA== SigAlg = http://www.w3.org/2000/09/xmldsig#rsa-sha1
Signature =
WtiIOqORS8kMxOdlkDSgx5wkvI7xGOBv1k/KqnJQO7mWfsYWANOD96diNDsfm2tTZvJBRgGbCMIeHEOnmJOyZWhAlSDjnrhN5fJrK7e5iQsX
BGq6dAEcseIWlyGQO6wX8/iuBGON5gTLAqhj97+VmjVZqII/rabt2UMmlODx2cPPhztJ7TAMCWbi+Y1dMYIicUvZ43cE4tuM7mcMVkmfYvOH
gIBYwDW2x+Cy6G7T4sG5Qww2R+yITeMY/iK+/q2OGYhS4l3kdAfnnNP43V3xPQHETvuCCE0sU9lNFCQQ7YqLAIDi0xTqkMCbgo1AFqEKiCmU
EeVlVv9wbsr1344JAQ==
```

Figure 16: LogoutResponse

```
<?xml version="1.0" encoding="UTF-8"?>
<samlp:LogoutResponse
 ↪  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
Destination="https://localhost:44344/Saml2/SingleLogoutService"
ID="_3cd7a67cb630010670c64a56da8225631525538572976"
InResponseTo="bac6dacccc4dd95c2b1ce9081bb52"
IssueInstant="2018-05-05T16:42:52.979Z"
Version="2.0">
        <saml:Issuer
```

```
            xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
        https://saml2login.my.salesforce.com</saml:Issuer>
        <samlp:Status>
        <samlp:StatusCode
        Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
        </samlp:Status>
</samlp:LogoutResponse>
```

Listing 23: Validated LogoutResponse message

5.2.6 Case study 6: Middleware configuration

This case study verifies research question 6. The middleware implements extension methods for adding it into the application's middleware pipeline (12), and instantiating the required services during start up (13). These are simplified into simple single functions which are called in the application's startup class (14). Runtime application variables (service and identity provider configuration values) are added as JSON values in a file called **appsettings.json** placed at the root of the application (15).

Results

See section 5.1 for discussion on how the **DemoWebApp** was configured during result verification.

5.2.7 Case study 7: Monitoring and tracing

This case study verifies research question 7. The middleware offers the possibility of tracing specific events within the middleware using logs. Debug log level messages are added to the authentication handler when entering and leaving different functions. These can be persisted to the file system or database based on the service provider requirements.

6 Discussion

6.1 Threat analysis

This section discusses some of the threats associated with SAML-based systems. These are studied before and authors in [25] list a comprehensive list of possible threats. In this thesis, we base our analysis on the aforementioned list.

6.1.1 Denial of service (DOS)

SAML-based systems are susceptible to denial of service attacks whereby an attacker can issue user agent redirects into the identity or service provider target endpoints. Handling SAML requests requires computing resources to parse XML, validate digital signatures and decrypt XML messages. An attacker can potentially generate "fake" requests faster than the system can handle causing the system to drown [25].

Oasis recommends several countermeasures for denial of service attacks in SAML-based systems [25]. These include;

- Requiring client authentication using client certificates which are traceable in case of DOS attacks.
- Requiring the use of signed SAML requests which increases the work required by the attacker to generate and embed digital signatures to the requests.
- Restricting access to the Assertion Consumer Service (ACS) and Single Logout (SLO) endpoints to known service and identity providers.

Other threats associated with SAML-based systems are given below [25].

#	Threat	Description	Countermeasures
1.	Stolen Assertion	An eavesdropper copying a real user's assertion and using it to impersonate the user	Confidentiality protection 6.2.1, reject expired assertions, short values for NotBefore and NotAfter attributes
2.	Man In the Middle Attack	A malicious site impersonating the user	Validate the **Recipient** value in the assertion
3.	Forged Assertion	Malicious altering of an assertion	Ensure message integrity by verifying the message signature
4.	Modification or exposure of state information	Altering of RelayState information	Ensure message confidentiality and integrity of RelayState using encryption. Se 2.3.2 for more information

Table 3: Other threats associated with SAML-based systems

6.2 Security requirements

This section looks at some of the security requirements identified in section 4.1, and discusses the security controls implemented in the middleware to address them.

6.2.1 Confidentiality

Confidentiality is the ability to ensure that the contents of SAML messages send from either an identity provider to a service provider or vice versa are only accessible by the indented recipient. This requires the protection of the message while in transit and also on the message level [25].

Oasis recommends the use of secure network protocols such as TLS and IP Security protocols (IPSec) between service and identity providers to ensure the confidentiality of messages in transit [25].

On the message level, the confidentiality of SAML messages can be ensured by the use of XML Encryption of selected XML elements within the message [25]. The authentication middleware supports this by allowing identity providers to issue encrypted assertions. The middleware is capable of reading the encrypted assertion and decrypting it using the service provider's private key.

Listing 24 shows how the middleware reads and decrypts the contents of an encrypted assertion.

```
public XmlElement GetDecriptedAssertion(
        XmlElement xmlElement,
        AsymmetricAlgorithm privateKey)
{
    var encryptedList =
        xmlElement.GetElementsByTagName(
        EncryptedAssertion.ELEMENT_NAME,
        Saml2Constants.ASSERTION);
    var assertion = (XmlElement) encryptedList[0];
    if (assertion == null)
        throw new Saml2Exception("Missing assertion");

    var encryptedAssertion = new Saml2EncryptedAssertion(
                                    (RSA) privateKey);
    encryptedAssertion.LoadXml(assertion);
    encryptedAssertion.Decrypt();

    return encryptedAssertion.Assertion.DocumentElement;
}
```

Listing 24: Decrypting an encrypted assertion

6.2.2 Message Integrity

Message integrity is the ability to prove that a received SAML message has not been altered under transport. The integrity of SAML messages should be ensured when in transit using secure network protocols such as TLS and IPSec. To protect message integrity on message level, XML Signature is used to create a signature of the original SAML message. The signature is transported together with the original message, giving the receiver the ability to verify and confirm that the original message was not altered [25].

The middleware protects and verifies the integrity of both SAMLRequests and SAMLResponses. SAMLRequests are protected by creating and embedding the signature of the original SAMLRequest in the SAMLRequest, according to the

SAML specifications. See page 19 and case study 5.2.1 for more details.

The integrity of a received SAMLResponse is confirmed by verifying the signature that is embedded on the SAML message. Messages whose integrity cannot be verified are discarded. It is possible to configure the middleware to not check message signatures in **appsettings** using the provided **OmitAssertionSignatureCheck** configuration value.

Listing 25 shows how the middleware verifies the signature of a received assertion.

```
1  public Saml2Assertion GetValidatedAssertion(
2      XmlElement assertionElement,
3      AsymmetricAlgorithm key,
4      string audience,
5      bool omitAssertionSignatureCheck = false)
6  {
7      var keys = new List<AsymmetricAlgorithm> { key };
8      var assertion = new Saml2Assertion(assertionElement, keys,
9      AssertionProfile.Core, new List<string> { audience }, false);
10     if (!omitAssertionSignatureCheck)
11     {
12         //TODO: This is checked automaticaly if autovalidation is
           ↪ on
13         if (!assertion.CheckSignature(keys))
14         {
15             throw new Saml2Exception(
16                         "Invalid signature in assertion");
17         }
18     }
19
20     if (assertion.IsExpired())
21     {
22         throw new Saml2Exception("Assertion is expired");
23     }
24
25     return assertion;
26 }
```

Note that, it is OPTIONAL to check and confirm the integrity and confiden-

Listing 25: Assertion signature check

tiality of SAML requests and responses according to the SAML specifications [7]. Service providers and identity providers should choose this depending on the security requirements of their SAML systems.

6.2.3 Replay attacks

Replay attacks are caused by re-submission of forms containing SAML responses, in order to gain access to protected resources. Oasis recommends the use of one-time-use values in assertions that can be validated by the service provider [25].

The middleware protects against replay attacks by creating and embedding a unique request identifier on each authentication or logout request. The value is kept on an encrypted local cookie. This value is expected to be echoed back in the SAML response inside the **InResponseTo** XML attribute. On receiving any SAML response, the value of the unique request identifier cookie is read, and compared to the **InResponseTo** attribute value. Messages whose values do not match are discarded. Listing 26 shows how the middleware checks for replay attacks in SAML responses.

```
1  public void CheckReplayAttack(
2      XmlElement element,
3      string originalSamlRequestId)
4  {
5      var inResponseToAttribute =
6              element.Attributes["InResponseTo"];
7      if (inResponseToAttribute == null)
8      {
9          throw new Saml2Exception(
10         "Received a response message that did
11         not contain an InResponseTo attribute");
12     }
13
14     var inResponseTo = inResponseToAttribute.Value;
15     if (string.IsNullOrEmpty(originalSamlRequestId)
16        || string.IsNullOrEmpty(inResponseTo))
17     {
18         throw new Saml2Exception(
```

```
19                 "Empty protocol message id is not allowed.");
20         }
21
22         if (!inResponseTo.Equals(originalSamlRequestId,
23             StringComparison.OrdinalIgnoreCase))
24         {
25             throw new Saml2Exception("Replay attack.");
26         }
27     }
```

Listing 26: Replay attack check

6.2.4 Privacy consideration

SAML transfers the identity information about an authenticated subject, from an identity provider to a service provider as SAML statements with associated attributes. It is possible that at times the subject, the service provider or the identity provider would like to keep the accessibility of some of this identity information restricted. Example of such information might be attributes about the subject's health or finances [25].

In this situation, Oasis recommends that, the privacy laws and regulations in the respective jurisdiction should be considered when deploying SAML-based systems. The identity provider and service provider should address any privacy concerns within their systems [25].

6.3 Advantages and disadvantages of the thesis

This section discusses the advantages and disadvantages of using the created SAML 2.0 authentication middleware for ASP.NET Core. Also given, are the benefits of using SAML 2.0 as an authentication framework.

6.3.1 Advantages

Usability The middleware is easy to use. It requires only a few lines of code to add it into the HTTP request processing pipeline.

Flexibility The middleware is reusable and flexible, it is designed to support any service and identity provider that implements the SAML 2.0 framework.

Save development time and costs The middleware will save software developers and enterprises the time needed to integrate with SAML identity providers, helping them to focus on developing features that give direct returns to their organizations.

Extendable Support for other SAML binding combinations (use cases) can be added with ease.

6.3.2 Disadvantages

Testing The middleware has not been thoroughly tested for features, performance and security. It is possible that the current implementation contains bugs and security problems. It is also not known the number of concurrent authentication requests the middleware can service in a busy environment.

Partial support for SAML binding combinations The middleware supports only two SAML binding combinations (SP initiated: Redirect → POST binding, SP initiated: Redirect → Artifact binding) and SP initiated logout with the HTTP redirect binding.

6.4 Benefits of SAML 2.0

Geyer, in the saml.xml.org journal gives several advantages of using SAML as an authentication protocol. Some of these are discussed below.

Platform neutrality SAML separates security and authentication from service applications, making security and application logic independent of each other [77].

Improved online user experience SAML Single Sign-On alows users to enter their credentials once at a trusted identity provider, and use them to access online services on different service providers [77].

Reduced administrative costs for service providers The cost of maintaining identities is handled by the identity provider [77].

Transfer of risk The risks around maintaining and properly protecting identity information is handled by the identity provider [77].

7 Conclusion

Saml2.Authentication.Core is a fork of the OIOSAML.Net repository. The SAML schema C# classes were copied, ported and used as the base for creating the Saml2.Authentication.Core middleware. The middleware is reusable; it can be used to integrate any service provider, with any real world SAML-based identity provider, using a few lines of code. It is flexible; it is possible to configure it to support variable service and identity provider configurations.

The middleware is capable of handling both authentication and logout request-response messages. It transfers identity information from an identity provider to a service provider through a session cookie. The information in the session cookie can be used to create and signin a local user or grant access to local resources depending on the requirements of the service provider.

The middleware supports;

- Service provider initiated authentication requests with;
  - HTTP redirect binding SAMLRequest and HTTP POST binding SAML-Response.
  - HTTP redirect binding SAMLRequest and HTTP artifact binding SAML-Response.
- Single Logout Profile
  - Service provider initiated logout

The middleware does not support identity provider initiated logouts. This has negative consequences in Single Sign-On environments as the service provider will not respond to Single Logout requests from the identity provider.

7.1 Future Work

This thesis represents a fairly working solution for bringing SAML 2.0 authentication framework into ASP.NET Core. The solution needs core software development improvements to make it production ready. Below are interesting areas for further work that were out of scope for this study:

- Code refactoring to restructure the existing code to improve readability and reduce complexity.
- Thorough manual and automated testing to find bugs.
- Testing with other SAML-based identity providers.
- Adding support for other SAML binding combinations.
- Packaging and releasing the software as a nuget package.

Bibliography

[1] Government, U. *Computer-Based National Information Systems: Technology and Public Policy Issues*, chapter Chapter 10 Society's Dependence on Information Systems. U.S. Government Printing Office, SEPTEMBER 1981. Available at http://govinfo.library.unt.edu/ota/Ota_5/DATA/1981/8109.PDF. Retrieved 10.12.2017.

[2] Christensson, P. FEBRUARY 2014. Web application definition. *techterms.com*. Available at https://techterms.com/definition/web_application. Retrieved 10.12.2017.

[3] Microsoft. JUNE 2003. Design guidelines for secure web applications. *Microsoft Developer Network*. Available at https://msdn.microsoft.com/en-us/library/ff648647.aspx/#c04618429_007. Retrieved 10.12.2017.

[4] University, H. Behind the login screen: Understanding web authentication protocols. Available at https://iam.harvard.edu/resources/behind-login-screen. Retrieved 10.12.2017.

[5] stackoverflow. Results found containing saml. Available at https://stackoverflow.com/search?q=SAML. Retrieved 10.10.2017.

[6] I hate saml. Available at https://systoilet.wordpress.com/2010/09/29/i-hate-saml/. Retrieved 14.12.2017.

[7] OASIS. *Security Assertion Markup Language (SAML) 2.0 Technical Overview*, FEBRUARY 2005. Available at https://www.oasis-open.org/committees/download.php/11511/sstc-saml-tech-overview-2.0-draft-03.pdf. Retrieved 10.12.2017.

[8] Schwartz, M. is openid connect on the roadmap? *http://shibboleth.net*. Available at http://shibboleth.net/pipermail/dev/2012-November/001117.html. Retrieved 14.12.2017.

[9] Geyer, C. OCTOBER 2007. Saml specifications. *SAML Wiki*. Available at http://saml.xml.org/saml-specifications. Retrieved 10.12.2017.

[10] Roth, D., Anderson, R., & Luttin, S. MARCH 2017. Introduction to asp.net core. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/aspnet/core/. Retrieved 10.12.2017.

[11] Anderson, R. & Smith, S. OCTOBER 2017. Asp.net core middleware fundamentals. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware?tabs=aspnetcore2x. Retrieved 10.12.2017.

[12] OASIS. *OASIS Security Services TC*, JANUARY 2006. Available at https://www.oasis-open.org/committees/security/faq.php. Retrieved 10.12.2017.

[13] Brunner, L. April 2016. Developer-friendly saml single sign on support. *Stormpath.com*. Available at https://stormpath.com/blog/developer-friendly-saml-single-sign-on. Retrieved 14.12.2017.

[14] ComponentSpace. Purchase. Available at https://www.componentspace.com/Purchase-SAMLv20-Core.aspx. Retrieved 10.12.2017.

[15] Møller, K. V. SEPTEMBER 2017. Oiosaml.net 2.0. *Digitalisér dk*. Available at https://www.digitaliser.dk/resource/3732425. Retrieved 10.12.2017.

[16] onelogin. FEBRUARY 2014. 97% of saas vendors backing saml-based single sign-on. Available at https://www.onelogin.com/press-center/press-releases/97-percent-of-saas-vendors-backing-saml-based-single-sign-on. Retrieved 10.12.2017.

[17] Mozilla. Mozilla public license version 1.1. Available at https://www.mozilla.org/en-US/MPL/1.1/. Retrieved 12.11.2017.

[18] GitBook. Asp.net core from the ground up. Available at https://www.gitbook.com/book/openlearningportal/all-about-asp-net-core/details. Retrieved 10.11.2017.

[19] Microsoft. MAY 2017. Tour of .net. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/dotnet/standard/tour. Retrieved 10.11.2017.

[20] Microsoft. MARCH 2017. .net framework class library overview. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/dotnet/standard/class-library-overview. Retrieved 12.11.2017.

[21] Microsoft. JUNE 2016. .net core guide. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/dotnet/core/index. Retrieved 12.11.2017.

[22] Microsoft. AUGUST 2017. .net standard. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/dotnet/standard/net-standard. Retrieved 12.11.2017.

[23] Geyer, C. DECEMBER 2007. Protocols. *SAML Wiki*. Available at http://saml.xml.org/protocols. Retrieved 18.11.2017.

[24] (W3C), W. W. W. C. Xml schema part 1: Structures second edition. Available at https://www.w3.org/TR/xmlschema-1/. Retrieved 10.12.2017.

[25] OASIS. *Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0*, MARCH 2005. Available at http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf. Retrieved 10.12.2017.

[26] Microsoft. .net api browser. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/dotnet/api/?view=netcore-2.0. Retrieved 12.11.2017.

[27] Microsoft. System.security.cryptography.xml. *Nuget*. Available at https://www.nuget.org/packages/System.Security.Cryptography.Xml. Retrieved 12.11.2017.

[28] Microsoft. dotnet/corefx. *Github*. Available at https://github.com/dotnet/corefx/issues/4278. Retrieved 12.11.2017.

[29] Microsoft. JUNE 2016. Porting to .net core from .net framework. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/dotnet/core/porting/index. Retrieved 12.11.2017.

[30] Microsoft. Microsoft/dotnet-apiport. *Github*. Available at https://github.com/Microsoft/dotnet-apiport/. Retrieved 22.11.2017.

[31] Mozilla. Http. *MDN Web Docs*. Available at https://developer.mozilla.org/en-US/docs/Web/HTTP. Retrieved 12.11.2017.

[32] Uniface. Session management for web applications. Available at https://unifaceinfo.com/docs/0905/Uniface_Library_HTML/ulibrary/webSecurity_SessionManagement_AA38E5F6DE8733B3877A5598ADA532FC.html. Retrieved 10.12.2017.

[33] OWASP. Session management implementation. Available at https://www.owasp.org/index.php/Session_Management_Cheat_Sheet#Session_Management_Implementation. Retrieved 05.12.2017.

[34] Mozilla. Http cookies. *MDN Web Docs*. Available at https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. Retrieved 12.11.2017.

[35] Microsoft. Microsoft.aspnetcore.authentication.cookies. *Nuget*. Available at https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.Cookies/. Retrieved 12.11.2017.

[36] Anderson, R. & Latham, L. NOVEMBER 2017. Using cookie authentication without asp.net core identity. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/aspnet/core/security/authentication/cookie?tabs=aspnetcore2x. Retrieved 12.11.2017.

[37] Chiaretta, S. & Lattanzi, U. *ASP.NET Core Succinctly*, chapter 1, 3, 4. Syncfusion, Inc, 2017.

[38] Rick Anderson, Mark Michaelis, S. S. D. R. & Latham, L. JANUARY 2017. Configure an asp.net core app. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/index?tabs=basicconfiguration. Retrieved 12.11.2017.

[39] Johan Kraft, A. W. & Kienle, H. Trace recording for embedded systems: Lessons learned from five industrial projects. Available at http://www.es.mdh.se/pdf_publications/1911.pdf. Retrieved 07.11.2017.

[40] Levy, T. JANUARY 2016. An introduction to log-driven development. *logz.io*. Available at https://logz.io/blog/log-driven-development/. Retrieved 05.12.2017.

[41] Microsoft. net-commons/common-logging. *Github*. Available at https://github.com/net-commons/common-logging. Retrieved 22.11.2017.

[42] Smith, S. & Dykstra, T. 2017. Introduction to logging in asp.net core. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging/?tabs=aspnetcore2x. Retrieved 12.11.2017.

[43] Microsoft. *AuthenticationHandler<TOptions> Class*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication.authenticationhandler-1?view=aspnetcore-2.0.

[44] Microsoft. *AuthenticationOptions.AuthenticationScheme Property*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.authenticationoptions.authenticationscheme?view=aspnetcore-1.1#Microsoft_AspNetCore_Builder_AuthenticationOptions_AuthenticationScheme.

[45] Microsoft. *AuthenticationHandler<TOptions>.HandleChallengeAsync(AuthenticationProperties) Method*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication.authenticationhandler-1.handlechallengeasync?view=aspnetcore-2.0.

[46] Microsoft. *AuthenticationProperties Class*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.authentication.authenticationproperties?view=aspnetcore-2.0.

[47] Microsoft. *IAuthenticationRequestHandler.HandleRequestAsync Method*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication.iauthenticationrequesthandler.handlerequestasync?view=aspnetcore-2.0#Microsoft_AspNetCore_Authentication_IAuthenticationRequestHandler_HandleRequestAsync.

[48] Geyer, C. DECEMBER 2007. Assertions. *SAML Wiki*. Available at http://saml.xml.org/assertions. Retrieved 18.11.2017.

[49] October 2006. What is the maximum length of a url? *Boutell.com*. URL: https://boutell.com/newfaq/misc/urllength.html.

[50] Carol, G. 2007. Profiles. *http://saml.xml.org*. URL: http://saml.xml.org/components-profiles.

[51] Yin, R. K. *CASE STUDY RESEARCH Design and Methods*, volume 5 of *Applied Social Research Methods Series*, chapter 1-2, 1–23. SAGE Publications, International Educational and Professional Publisher Thousand Oaks London New Deihl, second edition, 2013.

[52] Zainal, Z. June 2007. Case study as a research method. Faculty of Management and Human Resource Development Universiti Teknologi Malaysia.

[53] Thomas Hardjono, Chair Hal Lockhart, S. C. *OASIS Security Services (SAML) TC*. Available at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security. Retrieved 10.12.2017.

[54] Myers, G. J. *The Art of Software Testing*. John Wiley & Sons, 2004. ISBN:0471469122.

[55] Bassil, Y. A simulation model for the waterfall software development life cycle. *International Journal of Engineering & Technology*. Available at https://arxiv.org/ftp/arxiv/papers/1205/1205.6904.pdf. Retrieved 05.12.2017.

[56] Microsoft. Introduction to the c# language and the .net framework. *Microsoft Docs*. Available at https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. Retrieved 15.11.2017.

[57] VERACODE. What is an integrated development environment (ide)? *APPSEC KNOWLEDGE BASE*. Available at https://www.veracode.com/security/integrated-development-environments. Retrieved 05.12.2017.

[58] Microsoft. Visual studio ide. Available at https://www.visualstudio.com/vs/. Retrieved 15.11.2017.

[59] Jetbrains. Resharper. Available at https://www.jetbrains.com/resharper/. Retrieved 15.11.2017.

[60] Sink, E. Version control by example. Available at http://alliancevoc.ac.ug/wp-content/uploads/2017/01/VersionControlByExample.pdf. Retrieved 05.12.2017.

[61] Git. git. Available at https://git-scm.com/. Retrieved 05.12.2017.

[62] Lionetti, G. What is version control: centralized vs. dvcs. *Atlassian blog*. Available at https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs. Retrieved 05.12.2017.

[63] Github. Available at https://github.com/. Retrieved 05.12.2017.

[64] Git extensions. Available at http://git-extensions-documentation.readthedocs.io/en/latest/git_extensions.html. Retrieved 05.12.2017.

[65] Kanban board. Available at https://trello.com/b/C4Awm5lK/kanban-board. Retrieved 05.12.2017.

[66] dk, D. dk.nita.saml20. *Nuget*. Available at https://www.nuget.org/packages/dk.nita.saml20/. Retrieved 04.12.2017.

[67] Microsoft. October 2016. Asp.net core data protection. *Microsoft docs*. URL: https://docs.microsoft.com/en-us/aspnet/core/security/data-protection/introduction?view=aspnetcore-2.1.

[68] Microsoft. Authenticationbuilder class. *Microsoft docs*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication.authenticationbuilder?view=aspnetcore-2.0.

[69] Microsoft. November 2017. Options pattern in asp.net core. *Microsoft docs*. URL: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options?view=aspnetcore-2.1#ipostconfigureoptions.

[70] Microsoft. Servicecollection class. *Microsoft docs*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.servicecollection?view=aspnetcore-2.0.

[71] Microsoft. April 2018. Application startup in asp.net core. *Microsoft docs*. URL: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-2.1.

[72] Microsoft. Controllerbase.challenge method. *Microsoft docs*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.controllerbase.challenge?view=aspnetcore-2.0.

[73] Microsoft. Authenticationhttpcontextextensions.signoutasync method. *Microsoft docs*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication.authenticationhttpcontextextensions.signoutasync?view=aspnetcore-2.0.

[74] Salesforce. Enable salesforce as an identity provider. *help.salesforce.com*. URL: https://help.salesforce.com/articleView?id=identity_provider_enable.htm&type=5.

[75] Salesforce. Create a connected app. *help.salesforce.com*. URL: https://help.salesforce.com/articleView?id=connected_app_create.htm&type=5.

[76] Microsoft. Httpresponse.redirect method. *Microsoft docs*. URL: https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.httpresponse.redirect?view=aspnetcore-2.1.

[77] Carol, G. 2007. Advantages of saml. *http://saml.xml.org*. URL: http://saml.xml.org/advantages-saml.

8 Materials

8.1 Source code

Github repository: https://github.com/jkmu/Saml2.Authentication.Core

8.2 Project planning

Project planning tool: https://trello.com/b/2cnwgqV3

Appendices

A .NET Core API port analysis in the OIOSAML.NET assembly

Target types and member support
Not supported
T:System.IdentityModel.Selectors.X509CertificateValidator
T:System.ServiceModel.BasicHttpBinding
T:System.ServiceModel.BasicHttpSecurityMode
T:System.ServiceModel.ChannelFactory‘1
T:System.ServiceModel.Channels.Binding
T:System.ServiceModel.Channels.BodyWriter
T:System.ServiceModel.Channels.HttpRequestMessageProperty
T:System.ServiceModel.Channels.IRequestChannel
T:System.ServiceModel.Channels.Message
T:System.ServiceModel.Channels.MessageHeaders
T:System.ServiceModel.Channels.MessageProperties
T:System.ServiceModel.Channels.MessageVersion
T:System.ServiceModel.EndpointAddress
T:System.ServiceModel.HttpBindingBase
T:System.ServiceModel.ICommunicationObject
T:System.Web.Caching.Cache
T:System.Web.Caching.CacheDependency
T:System.Web.Configuration.CustomErrorsMode
T:System.Web.Configuration.CustomErrorsSection

T:System.Web.Configuration.WebConfigurationManager

T:System.Web.HttpContext

T:System.Web.HttpCookie

T:System.Web.HttpCookieCollection

T:System.Web.HttpRequest

T:System.Web.HttpResponse

T:System.Web.IHttpHandler

T:System.Web.Security.FormsAuthentication

T:System.Web.SessionState.IRequiresSessionState

T:System.Web.UI.AttributeCollection

T:System.Web.UI.ClientScriptManager

T:System.Web.UI.Control

T:System.Web.UI.ControlCollection

T:System.Web.UI.CssStyleCollection

T:System.Web.UI.HtmlControls.HtmlControl

T:System.Web.UI.HtmlControls.HtmlHead

T:System.Web.UI.HtmlControls.HtmlInputControl

T:System.Web.UI.HtmlControls.HtmlInputHidden

T:System.Web.UI.HtmlControls.HtmlLink

T:System.Web.UI.HtmlControls.HtmlMeta

T:System.Web.UI.HtmlTextWriterStyle

T:System.Web.UI.LiteralControl

T:System.Web.UI.Page

T:System.Web.UI.WebControls.HyperLink

T:System.Web.UI.WebControls.Label

T:System.Web.UI.WebControls.Panel

T:System.Web.UI.WebControls.WebControl

T:System.Web.UI.WebResourceAttribute

Supported: 2.0+

T:System.Configuration.Configuration

T:System.Configuration.ConfigurationElement

T:System.Configuration.ConfigurationErrorsException

T:System.Configuration.ConfigurationManager

T:System.Configuration.ConfigurationSaveMode

T:System.Configuration.ConfigurationSection

T:System.Configuration.ConfigurationSectionCollection

T:System.Configuration.SectionInformation

T:System.Security.Cryptography.Xml.CipherData

T:System.Security.Cryptography.Xml.DSAKeyValue

T:System.Security.Cryptography.Xml.EncryptedData

T:System.Security.Cryptography.Xml.EncryptedKey

T:System.Security.Cryptography.Xml.EncryptedType

T:System.Security.Cryptography.Xml.EncryptedXml

T:System.Security.Cryptography.Xml.EncryptionMethod

T:System.Security.Cryptography.Xml.KeyInfo

T:System.Security.Cryptography.Xml.KeyInfoClause

T:System.Security.Cryptography.Xml.KeyInfoEncryptedKey

T:System.Security.Cryptography.Xml.KeyInfoX509Data

T:System.Security.Cryptography.Xml.Reference

T:System.Security.Cryptography.Xml.RSAKeyValue

T:System.Security.Cryptography.Xml.SignedInfo

T:System.Security.Cryptography.Xml.SignedXml

T:System.Security.Cryptography.Xml.Transform

T:System.Security.Cryptography.Xml.XmlDsigEnvelopedSignatureTransform

T:System.Security.Cryptography.Xml.XmlDsigExcC14NTransform

B Saml2Handler

```
using System;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;
using System.Text.Encodings.Web;
using System.Threading.Tasks;
using dk.nita.saml20;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Http.Extensions;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Saml2.Authentication.Core.Bindings;
using Saml2.Authentication.Core.Extensions;
using Saml2.Authentication.Core.Factories;
using Saml2.Authentication.Core.Options;
using Saml2.Authentication.Core.Services;

namespace Saml2.Authentication.Core.Authentication
{
    /// <inheritdoc>
    ///     <cref>AuthenticationHandler</cref>
    /// </inheritdoc>
    /// <summary>
    /// Saml2Handler class. Handles communication between the
    ↪ service provider and the identity provider.
    /// Handles authentication challeges, authentication
    ↪ responses, Logout requests and logout responses
    /// </summary>
    public class Saml2Handler
        : AuthenticationHandler<Saml2Options>,
            IAuthenticationRequestHandler,
            IAuthenticationSignOutHandler
    {
```

```
32          private readonly ISaml2ClaimFactory _claimFactory;
33          private readonly IHttpArtifactBinding
             ↪  _httpArtifactBinding;
34          private readonly IHttpRedirectBinding
             ↪  _httpRedirectBinding;
35          private readonly ILogger _logger;
36          private readonly ISamlService _samlService;
37
38          public Saml2Handler(
39              IOptionsMonitor<Saml2Options> options,
40              ILoggerFactory logger,
41              UrlEncoder encoder,
42              ISystemClock clock,
43              ISamlService samlService,
44              IHttpRedirectBinding httpRedirectBinding,
45              IHttpArtifactBinding httpArtifactBinding,
46              ISaml2ClaimFactory claimFactory)
47              : base(options, logger, encoder, clock)
48          {
49              _logger = logger.CreateLogger(typeof(Saml2Handler));
50              _samlService = samlService;
51              _httpRedirectBinding = httpRedirectBinding;
52              _httpArtifactBinding = httpArtifactBinding;
53              _claimFactory = claimFactory;
54          }
55
56          /// <inheritdoc />
57          /// <summary>
58          /// Called on every request.
59          /// Handles authentication and signout response requests.
60          /// </summary>
61          /// <returns>True or false if request is
             ↪  handled</returns>
62          public async Task<bool> HandleRequestAsync()
63          {
64              if (await HandleSignIn())
65                  return true;
66
67              if (await HandleSignOut())
68                  return true;
```

```
69              return await HandleHttpArtifact();
70          }
71
72          /// <inheritdoc />
73          /// <summary>
74          /// Handles service provider signout requests.
75          /// Creates a logout request and redirects to identity
            ↪ provider.
76          /// </summary>
77          /// <param name="properties"></param>
78          /// <returns></returns>
79          public Task SignOutAsync(AuthenticationProperties
            ↪ properties)
80          {
81              _logger.LogDebug($"Entering {nameof(SignOutAsync)}",
                ↪ properties);
82
83              var logoutRequestId = CreateUniqueId();
84              var cookieOptions = Options.RequestIdCookie
85                                        .Build(Context,
                                          ↪ Clock.UtcNow);
86              Response.Cookies.Append(Options.RequestIdCookie.Name,
87                      Options.StringDataFormat
88                          .Protect(logoutRequestId),
89                              cookieOptions);
90
91              var relayState =
                ↪ Options.StateDataFormat.Protect(properties);
92              var sessionIndex = Context.User.GetSessionIndex();
93              var subject = Context.User.GetSubject();
94
95              var logoutRequestUrl = _samlService
96                      .GetLogoutRequest(logoutRequestId,
97                              sessionIndex, subject, relayState);
98
99              _logger.LogDebug(
100                 $"Method={nameof(SignOutAsync)}.
101                 Redirecting to saml identity provider for SLO.
102                 Url={logoutRequestUrl}");
103
```

```
104             Context.Response.Redirect(logoutRequestUrl, true);
105             return Task.CompletedTask;
106         }
107 
108         protected override Task<AuthenticateResult>
            ↪ HandleAuthenticateAsync()
109         {
110             return Task.FromResult(AuthenticateResult.Fail("Not
                ↪ supported"));
111         }
112 
113         /// <inheritdoc />
114         /// <summary>
115         /// Handles service provider authentication requests.
116         /// Creates <AuthnRequest></AuthnRequest> and redirects
            ↪ to identity provider
117         /// </summary>
118         /// <param name="properties"></param>
119         /// <returns>Task.CompletedTask</returns>
120         protected override Task HandleChallengeAsync(
121             AuthenticationProperties properties)
122         {
123             _logger.LogDebug(
124                 $"Entering {nameof(HandleChallengeAsync)}",
                    ↪ properties);
125 
126             var authnRequestId = CreateUniqueId();
127 
128             var deleteCookieOptions = Options.RequestIdCookie
129                         .Build(Context, Clock.UtcNow);
130             Response.DeleteAllRequestIdCookies(
131                     Context.Request, deleteCookieOptions);
132 
133             var cookieOptions = Options.RequestIdCookie
134                                     .Build(Context,
                                        ↪ Clock.UtcNow);
135             Response.Cookies.Append(
136                         Options.RequestIdCookie.Name,
137                             Options.StringDataFormat
138                                 .Protect(authnRequestId),
```

```
                                    cookieOptions);

            var relayState =
             ↪  Options.StateDataFormat.Protect(properties);
            var requestUrl = _samlService.GetAuthnRequest(
                authnRequestId, relayState,
                $"{Request.GetBaseUrl()}/
                    {Options.AssertionConsumerServiceUrl}");

            _logger.LogDebug(
                $"Method={nameof(HandleChallengeAsync)}.
                Redirecting to saml identity provider for SSO.
                Url={requestUrl}");

            Context.Response.Redirect(requestUrl, true);
            return Task.CompletedTask;
        }

        /// <summary>
        /// Handles sigout response requests from identity
         ↪  provider
        /// </summary>
        /// <returns>True/false</returns>
        private async Task<bool> HandleSignOut()
        {
            if (!Request.Path.Value.EndsWith(
                    Options.SingleLogoutServiceUrl,
                    StringComparison.OrdinalIgnoreCase))
                return false;

            _logger.LogDebug($"Entering
             ↪  {nameof(HandleSignOut)}");

            if (!_httpRedirectBinding.IsValid(Context.Request))
                return false;

            var uri = new Uri(Context.Request.GetEncodedUrl());
            //idp initiated logout. BUG:Context.User and cookies
             ↪  are not populated
            if (_httpRedirectBinding
```

```
                .IsLogoutRequest(Context.Request))
        {
            var logoutReponse =
             ↪ _samlService.GetLogoutReponse(uri);
            if (logoutReponse.StatusCode
                != Saml2Constants.StatusCodes.Success
                || Context.User.Identity.IsAuthenticated)
                return false;

            var relayState = _httpRedirectBinding.
                                GetCompressedRelayState(
                                        Context.Request);
            var url = _samlService
                        .GetLogoutResponseUrl(logoutReponse,
                         ↪ relayState);

            await Context.SignOutAsync(
                    Options.SignOutScheme, new
                     ↪ AuthenticationProperties());

            Context.Response.Redirect(url, true);
            return true;
        }

        //sp initiated logout
        var response =
         ↪ _httpRedirectBinding.GetResponse(Context.Request);
        var authenticationProperties =
            Options.StateDataFormat.Unprotect(
                response.RelayState) ?? new
                 ↪ AuthenticationProperties();

        var initialLogoutRequestId = GetRequestId();
        if (!_samlService.IsLogoutResponseValid(
                            uri, initialLogoutRequestId))
            return false;

        await Context.SignOutAsync(
                    Options.SignOutScheme,
                     ↪ authenticationProperties);
```

```
209
210             var cookieOptions = Options.RequestIdCookie
211                                        .Build(Context,
                                           ↪  Clock.UtcNow);
212
213             Context.Response
214                 .DeleteAllRequestIdCookies(Context.Request,
                     ↪  cookieOptions);
215
216             var redirectUrl =
                 ↪  GetRedirectUrl(authenticationProperties);
217
218             _logger.LogDebug(
219                 $"Method={nameof(HandleSignOut)}.
220                 Received and handled sp initiated logout
                     ↪  response.
221                 Redirecting to {redirectUrl}");
222
223             Context.Response.Redirect(redirectUrl, true);
224             return true;
225         }
226
227         /// <summary>
228         /// Handles authentication response requests from idp
229         /// HTTP Redirect binding
230         /// </summary>
231         /// <returns>True/false</returns>
232         private async Task<bool> HandleSignIn()
233         {
234             if (!Request.Path.Value.EndsWith(
235                 Options.AssertionConsumerServiceUrl,
236                 StringComparison.OrdinalIgnoreCase))
237                 return false;
238
239             _logger.LogDebug($"Entering {nameof(HandleSignIn)}");
240
241             if (!_httpRedirectBinding.IsValid(Context.Request))
242                 return false;
243
244             var initialAuthnRequestId = GetRequestId();
```

```
245             var result =
                ↪  _httpRedirectBinding.GetResponse(Context.Request);
246             var base64EncodedSamlResponse = result.Response;
247             var assertion =
                ↪  _samlService.HandleHttpRedirectResponse(
248                             base64EncodedSamlResponse,
                                ↪  initialAuthnRequestId);
249
250             var authenticationProperties =
251                 Options.StateDataFormat.Unprotect(
252                     result.RelayState) ?? new
                        ↪  AuthenticationProperties();
253             await SignIn(assertion, authenticationProperties);
254
255             var cookieOptions = Options.RequestIdCookie
256                                        .Build(Context,
                                           ↪  Clock.UtcNow);
257             Response.DeleteAllRequestIdCookies(Context.Request,
                ↪  cookieOptions);
258
259             var redirectUrl =
                ↪  GetRedirectUrl(authenticationProperties);
260
261             _logger.LogDebug(
262                 $"Method={nameof(HandleSignIn)}.
263                 Received and handled SSO redirect response.
264                 Redirecting to {redirectUrl}");
265
266             Context.Response.Redirect(redirectUrl, true);
267             return true;
268         }
269
270         /// <summary>
271         /// Handles authentication request responses from idp
272         /// HTTP Artifact binding
273         /// </summary>
274         /// <returns>True/false</returns>
275         private async Task<bool> HandleHttpArtifact()
276         {
277             if (!Request.Path.Value.EndsWith(
```

```
278             Options.AssertionConsumerServiceUrl,
279             StringComparison.OrdinalIgnoreCase))
280                 return false;
281
282             _logger.LogDebug($"Entering
                ↪ {nameof(HandleHttpArtifact)}");
283
284             if (!_httpArtifactBinding.IsValid(Context.Request))
285                 return false;
286
287             //TODO validate inResponseTo
288             var initialAuthnRequestId = GetRequestId();
289
290             var assertion = _samlService
291                     .HandleHttpArtifactResponse(Context.Request);
292
293             var relayState = _httpArtifactBinding
294                                 .GetRelayState(Context.Request);
295             var authenticationProperties =
296                 Options.StateDataFormat
297                     .Unprotect(relayState) ?? new
                        ↪ AuthenticationProperties();
298
299             await SignIn(assertion, authenticationProperties);
300
301             var cookieOptions = Options.RequestIdCookie
302                                        .Build(Context,
                                           ↪ Clock.UtcNow);
303             Response.DeleteAllRequestIdCookies(Context.Request,
                ↪ cookieOptions);
304
305             var redirectUrl =
                ↪ GetRedirectUrl(authenticationProperties);
306
307             _logger.LogDebug(
308                 $"Method={nameof(HandleHttpArtifact)}.
309                 Received and handled SSO artifact response.
310                 Redirecting to {redirectUrl}");
311
312             Context.Response.Redirect(redirectUrl, true);
```

```
313             return true;
314         }
315 
316         /// <summary>
317         /// Gets identity information from assertion and signs in
                ↪ user
318         /// </summary>
319         /// <param name="assertion">Saml2Assertion</param>
320         /// <param name="authenticationProperties">
321         /// AuthenticationProperties</param>
322         /// <returns></returns>
323         private async Task SignIn(
324             Saml2Assertion assertion,
325             AuthenticationProperties authenticationProperties)
326         {
327             var claims = _claimFactory.Create(assertion);
328             var identity = new ClaimsIdentity(claims,
                    ↪ Scheme.Name);
329             var principal = new ClaimsPrincipal(identity);
330 
331             await Context.SignInAsync(
332                             Options.SignInScheme,
333                             principal,
334                             authenticationProperties);
335         }
336 
337         /// <summary>
338         /// Creates a unique cryptorandom id
339         /// </summary>
340         /// <param name="length"></param>
341         /// <returns></returns>
342         private static string CreateUniqueId(int length = 32)
343         {
344             var bytes = new byte[length];
345             using (var randomNumberGenerator =
                    ↪ RandomNumberGenerator.Create())
346             {
347                 randomNumberGenerator.GetBytes(bytes);
348                 var hex = new StringBuilder(bytes.Length * 2);
349                 foreach (var b in bytes)
```

```
350                     hex.AppendFormat("{0:x2}", b);
351
352                 return hex.ToString();
353             }
354         }
355
356         /// <summary>
357         /// Gets the requestId from cookie
358         /// </summary>
359         /// <returns>RequestId</returns>
360         private string GetRequestId()
361         {
362             var requestIdCookie = Request.GetRequestIdCookie();
363             if (string.IsNullOrEmpty(requestIdCookie))
364                 throw new ArgumentNullException(
365                                     nameof(requestIdCookie));
366
367             return Options.StringDataFormat
368                                 .Unprotect(requestIdCookie);
369         }
370
371         /// <summary>
372         /// Gets the redirect url from authentication properties
373         /// </summary>
374         /// <param name="authenticationProperties">
375         /// AuthenticationProperties</param>
376         /// <returns>RedirectUrl</returns>
377         private string GetRedirectUrl(
378             AuthenticationProperties authenticationProperties)
379         {
380             return authenticationProperties.RedirectUri
381                     .IsNotNullOrEmpty()
382                 ? authenticationProperties.RedirectUri
383                 : Options.DefaultRedirectUrl;
384         }
385     }
386 }
```

C Saml2ServiceCollectionExtensions

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Options;
using Saml2.Authentication.Core.Bindings;
using Saml2.Authentication.Core.Bindings.SignatureProviders;
using Saml2.Authentication.Core.Extensions;
using Saml2.Authentication.Core.Factories;
using Saml2.Authentication.Core.Options;
using Saml2.Authentication.Core.Providers;
using Saml2.Authentication.Core.Services;
using Saml2.Authentication.Core.Validation;

namespace Microsoft.Extensions.DependencyInjection
{
    public static class Saml2ServiceCollectionExtensions
    {
        public static IServiceCollection AddSaml(
            this IServiceCollection services)
        {
            if (services == null)
                throw new
                 ↪  ArgumentNullException(nameof(services));

            services.AddRequiredServices();
            return services;
        }

        private static void AddRequiredServices(
            this IServiceCollection services)
        {
            services.AddOptions();
```

```
33              services.TryAddSingleton(resolver =>
34                  resolver.GetRequiredService<IOptions<
35                  Saml2Configuration>>().Value);
36
37              services.TryAddTransient<ISaml2Validator,
                ↪  Saml2Validator>();
38              services.TryAddTransient<ISaml2ClaimFactory,
                ↪  Saml2ClaimFactory>();
39              services.TryAddTransient<ISamlProvider,
                ↪  SamlProvider>();
40              services.TryAddTransient<ISaml2MessageFactory,
41                                       Saml2MessageFactory>();
42              services.TryAddTransient<ISignatureProviderFactory,
43                                       SignatureProviderFactory>();
44              services.TryAddTransient<IHttpRedirectBinding,
45                                       HttpRedirectBinding>();
46              services.TryAddTransient<IHttpArtifactBinding,
47                                       HttpArtifactBinding>();
48              services.TryAddTransient<ISamlService,
                ↪  SamlService>();
49          }
50
51          /// <summary>
52          ///     Add signing certificates by thumpprint
53          /// </summary>
54          /// <param name="services"></param>
55          /// <param name="serviceProviderCertificateThumbprint">
56          /// ServiceProvider's certificate thumbprint</param>
57          /// <param name="identityProviderCertificateThumbprint">
58          /// IdentityProvider's certificate thumbprint</param>
59          /// <returns></returns>
60          public static IServiceCollection AddSigningCertificates(
61              this IServiceCollection services,
62              string serviceProviderCertificateThumbprint,
63              string identityProviderCertificateThumbprint)
64          {
65              if (services == null)
66                  throw new
                    ↪  ArgumentNullException(nameof(services));
67
```

```
68              return AddSigningCertificates(services,
69                  GetSigningCertificates(
70                      X509FindType.FindByThumbprint,
71                      serviceProviderCertificateThumbprint
72                          .TrimSpecialCharacters(),
73                      identityProviderCertificateThumbprint
74                          .TrimSpecialCharacters()));
75          }
76
77          public static IServiceCollection AddSigningCertificates(
78              this IServiceCollection services, X509FindType
                ↪ findType,
79              string serviceProviderCertificateName,
80              string identityProviderCertificateName)
81          {
82              if (services == null)
83                  throw new
                    ↪ ArgumentNullException(nameof(services));
84
85              return AddSigningCertificates(services,
86                  GetSigningCertificates(
87                      findType,
88                      identityProviderCertificateName,
89                      serviceProviderCertificateName));
90          }
91
92          /// <summary>
93          ///     Add signing certificates from file
94          /// </summary>
95          /// <param name="services"></param>
96          /// <param name="serviceProviderCertificateFileName">
97          /// ServiceProvider's certificate filename</param>
98          /// <param name="identityProviderCertificateFileName">
99          /// IdentityProvider's certificate filename</param>
100         /// <returns></returns>
101         public static IServiceCollection
            ↪ AddSigningCertificatesFromFile(
102             this IServiceCollection services,
103             string serviceProviderCertificateFileName,
104             string identityProviderCertificateFileName)
```

```
105         {
106             if (services == null)
107                 throw new
                    ↪ ArgumentNullException(nameof(services));
108
109             if (string.IsNullOrEmpty(
110                     serviceProviderCertificateFileName))
111                 throw new ArgumentNullException(
112                     nameof(serviceProviderCertificateFileName));
113
114             if (string.IsNullOrEmpty(
115                 identityProviderCertificateFileName))
116                     throw new ArgumentNullException(
117                     nameof(
118                         identityProviderCertificateFileName));
119
120             var serviceProviderCertificateFullFileName =
121                 !Path.IsPathRooted(
122                     serviceProviderCertificateFileName)
123                 ? Path.Combine(Directory.GetCurrentDirectory(),
124                 serviceProviderCertificateFileName)
125                 : serviceProviderCertificateFileName;
126             var serviceProviderCertificate =
127                 new X509Certificate2(
128                     serviceProviderCertificateFullFileName);
129
130             var identityProviderCertificateFullFileName =
131                 !Path.IsPathRooted(
132                     identityProviderCertificateFileName)
133                 ? Path.Combine(Directory.GetCurrentDirectory(),
134                 identityProviderCertificateFileName)
135                 : identityProviderCertificateFileName;
136             var identityProviderCertificate =
137                 new X509Certificate2(
138                     identityProviderCertificateFullFileName);
139
140             return AddSigningCertificates(
141                                 services,
142                                 new SigningCertificate(
143                                 identityProviderCertificate,
```

```
144                                 serviceProviderCertificate));
145 }
146
147         private static X509Certificate2 GetCertificate(
148             string findValue,
149             X509FindType findType,
150             StoreName storeName = StoreName.My,
151             StoreLocation storeLocation =
                ↪ StoreLocation.LocalMachine,
152             bool validOnly = false)
153         {
154             var store = new X509Store(storeName, storeLocation);
155             try
156             {
157                 store.Open(OpenFlags.ReadOnly);
158                 var found = store.Certificates.Find(
159                                                     findType,
160                                                     findValue,
161                                                     validOnly);
162                 if (found.Count == 0)
163                 {
164                     var searchDescriptor = SearchDescriptor(
165                                                 findValue,
166                                                 findType,
167                                                 storeName,
168                                                 storeLocation,
169                                                 validOnly);
170                     var msg =
171                         $"A configured certificate could not be
                            ↪ found
172                         in the certificate
                            ↪ store.{searchDescriptor}";
173
174                         throw new Exception(msg);
175                 }
176                 if (found.Count > 1)
177                 {
178                     var searchDescriptor = SearchDescriptor(
179                                                 findValue,
180                                                 findType,
```

```
                                        storeName,
                                        storeLocation,
                                        validOnly);
                var msg =
                    $"Found more than one certificate in the
                     ↪  certificate store.
                    Make sure you don't have duplicate
                     ↪  certificates installed.
                    {searchDescriptor}";

                    throw new Exception(msg);
                }
                return found[0];
            }
            finally
            {
                store.Close();
            }
        }

        private static string SearchDescriptor(
            string findValue,
            X509FindType findType,
            StoreName storeName,
            StoreLocation storeLocation,
            bool validOnly)
        {
        var message =
            $"The certificate was searched for in
             ↪  {storeLocation}/{storeName},
            {findType}='{findValue}', validOnly={validOnly}.";

        if (findType == X509FindType.FindByThumbprint
            && findValue?.Length > 0 && findValue[0] == 0x200E)

        message =
        "The configuration for the certificate searches by
         ↪  thumbprint but has an invalid character in the
         ↪  thumbprint string." + message;

```

```
216             return message;
217         }
218
219         private static SigningCertificate GetSigningCertificates(
220             X509FindType findType,
221             string serviceProviderCertificateName,
222             string identityProviderCertificateName)
223         {
224             if (string.IsNullOrEmpty(
225                 serviceProviderCertificateName))
226                     throw new ArgumentNullException(
227                         nameof(serviceProviderCertificateName));
228
229             if (string.IsNullOrEmpty(
230                 identityProviderCertificateName))
231                     throw new ArgumentNullException(
232                         nameof(identityProviderCertificateName));
233
234             var serviceProviderCertificate =
235                     GetCertificate(
236                         serviceProviderCertificateName,
                            ↪ findType);
237             var identityProviderCertificate =
238                     GetCertificate(
239                         identityProviderCertificateName,
                            ↪ findType);
240             var certificates = new SigningCertificate(
241                         identityProviderCertificate,
                            ↪ serviceProviderCertificate);
242
243             return certificates;
244         }
245
246         private static void
            ↪ CheckServiceProviderCetificatePrivateKey(
247             SigningCertificate signingCertificates)
248         {
249             if (signingCertificates == null)
250                 throw new ArgumentNullException(
251                     nameof(signingCertificates));
```

```
252
253             if (!signingCertificates
254                     .ServiceProvider.HasPrivateKey)
255                 throw new InvalidOperationException(
256                     "Certificate does not have a private key.");
257         }
258
259         private static IServiceCollection AddSigningCertificates(
260             IServiceCollection services,
261             SigningCertificate signingCertificates)
262         {
263             CheckServiceProviderCetificatePrivateKey(
264                 signingCertificates);
265                 services.AddSingleton<ICertificateProvider>(
266                     new CertificateProvider(
267                         signingCertificates));
268             return services;
269         }
270     }
271 }
```

D Saml2Extensions

```
using System;
using Microsoft.AspNetCore.Authentication;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Options;
using Saml2.Authentication.Core.Authentication;
using Saml2.Authentication.Core.Options;

namespace Microsoft.Extensions.DependencyInjection
{
    public static class Saml2Extensions
    {
        public static AuthenticationBuilder AddSaml(
            this AuthenticationBuilder builder)
                => builder.AddSaml(
                    Saml2Defaults.AuthenticationScheme, _ => {
                    ↪ });

        public static AuthenticationBuilder AddSaml(
            this AuthenticationBuilder builder,
            Action<Saml2Options> configureOptions)
                => builder.AddSaml(
                    Saml2Defaults.AuthenticationScheme,
                    ↪ configureOptions);

        public static AuthenticationBuilder AddSaml(
            this AuthenticationBuilder builder,
            string authenticationScheme,
            Action<Saml2Options> configureOptions)
                => builder.AddSaml(
                    authenticationScheme,
                     Saml2Defaults
                        .AuthenticationSchemeDisplayName,
                            configureOptions);
```

```
32
33          public static AuthenticationBuilder AddSaml(
34              this AuthenticationBuilder builder,
35              string authenticationScheme,
36              string displayName,
37              Action<Saml2Options> configureOptions)
38          {
39              builder.Services.TryAddEnumerable(
40              ServiceDescriptor.Singleton<
41                  IPostConfigureOptions<Saml2Options>,
42                      Saml2PostConfigureOptions>());
43
44              return builder.AddScheme<Saml2Options, Saml2Handler>(
45                      authenticationScheme, displayName,
                        ↪  configureOptions);
46          }
47      }
48  }
```

E Saml2PostConfigureOptions

```
using System.Text;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.Extensions.Options;
using Saml2.Authentication.Core.Authentication;

namespace Saml2.Authentication.Core.Options
{
    public class Saml2PostConfigureOptions :
     ↪ IPostConfigureOptions<Saml2Options>
    {
        private readonly IDataProtectionProvider
         ↪ _dataProtectionProvider;

        public Saml2PostConfigureOptions(
            IDataProtectionProvider dataProtectionProvider)
        {
            _dataProtectionProvider = dataProtectionProvider;
        }

        public void PostConfigure(string name, Saml2Options
         ↪ options)
        {
            options.DataProtectionProvider =
             ↪ options.DataProtectionProvider
                ?? _dataProtectionProvider;

            if (string.IsNullOrEmpty(options.SignOutScheme))
                options.SignOutScheme = options.SignInScheme;

            if (options.StateDataFormat == null)
            {
                var dataProtector =
                 ↪ options.DataProtectionProvider
```

```
30                  .CreateProtector(typeof(Saml2Handler).FullName);
31                  options.StateDataFormat =
32                      new PropertiesDataFormat(dataProtector);
33              }
34
35              if (options.StringDataFormat == null)
36              {
37                  var dataProtector =
38                      options.DataProtectionProvider
39                      .CreateProtector(
40                          typeof(Saml2Handler).FullName,
41                              typeof(string).FullName);
42
43                  options.StringDataFormat = new
                    ↪ SecureDataFormat<string>(
44                                      new StringSerializer(),
                                        ↪ dataProtector);
45              }
46          }
47      }
48
49      internal class StringSerializer : IDataSerializer<string>
50      {
51          public string Deserialize(byte[] data)
52          {
53              return Encoding.UTF8.GetString(data);
54          }
55
56          public byte[] Serialize(string model)
57          {
58              return Encoding.UTF8.GetBytes(model);
59          }
60      }
61  }
```

F Saml2Options

```
using System;
using System.Security.Authentication;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Internal;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.AspNetCore.Http;
using Saml2.Authentication.Core.Authentication;

namespace Saml2.Authentication.Core.Options
{
    public class Saml2Options : AuthenticationSchemeOptions
    {
        private CookieBuilder _requestIdCookie;

        public Saml2Options()
        {
            AssertionConsumerServiceUrl =
            ↪ "Saml2/AssertionConsumerService";
            SingleLogoutServiceUrl = "Saml2/SingleLogoutService";
            DefaultRedirectUrl = "/";
            SignInScheme = Saml2Defaults.SignInScheme;
            AuthenticationScheme =
            ↪ Saml2Defaults.AuthenticationScheme;
            RequestIdCookieLifetime = TimeSpan.FromMinutes(10);

            _requestIdCookie = new RequestIdCookieBuilder(this)
            {
                Name = $"{Saml2Defaults.RequestIdCookiePrefix}
                                        .{Guid.NewGuid():N}",
                HttpOnly = true,
                SameSite = SameSiteMode.None,
                SecurePolicy = CookieSecurePolicy.SameAsRequest
            };
```

```
32          }
33
34          public string AssertionConsumerServiceUrl { get; set; }
35
36          public string SingleLogoutServiceUrl { get; set; }
37
38          public string DefaultRedirectUrl { get; set; }
39
40          public string SignInScheme { get; set; }
41
42          public string SignOutScheme { get; set; }
43
44          public string AuthenticationScheme { get; set; }
45
46          public ISecureDataFormat<AuthenticationProperties>
47                                                  StateDataFormat {
                                                     ↪ get; set; }
48
49          public IDataProtectionProvider DataProtectionProvider {
             ↪ get; set; }
50
51          public TimeSpan RequestIdCookieLifetime { get; set; }
52
53          public CookieBuilder RequestIdCookie
54          {
55              get => _requestIdCookie;
56              set => _requestIdCookie = value
57                  ?? throw new
                     ↪ AuthenticationException(nameof(value));
58          }
59
60          public ISecureDataFormat<string> StringDataFormat { get;
             ↪ set; }
61      }
62
63      internal class RequestIdCookieBuilder :
         ↪ RequestPathBaseCookieBuilder
64      {
65          private readonly Saml2Options _options;
66
```

```
67          public RequestIdCookieBuilder(Saml2Options options)
68          {
69              _options = options;
70          }
71
72          public override CookieOptions Build(
73              HttpContext context,
74              DateTimeOffset expiresFrom)
75          {
76              var cookieOptions = base.Build(context, expiresFrom);
77
78              if (!Expiration.HasValue ||
                ↪  !cookieOptions.Expires.HasValue)
79                  cookieOptions.Expires =
80                      expiresFrom.Add(
81                          _options.RequestIdCookieLifetime);
82
83              return cookieOptions;
84          }
85      }
86  }
```

G Saml2Configuration

```
namespace Saml2.Authentication.Core.Options
{
    public class Saml2Configuration
    {
        public bool ForceAuth { get; set; }

        public bool IsPassive { get; set; }

        public bool SignAuthnRequest { get; set; }

        public string AuthnContextComparisonType { get; set; }

        public string[] AuthnContextComparisonItems { get; set; }

        public bool? AllowCreate { get; set; }

        public bool OmitAssertionSignatureCheck { get; set; }

        public IdentityProviderConfiguration
                        IdentityProviderConfiguration { get; set;
                        ↪ }

        public ServiceProviderConfiguration
                        ServiceProviderConfiguration { get; set;
                        ↪ }
    }
}
```

H IdentityProviderConfiguration

```
using System.ComponentModel;

namespace Saml2.Authentication.Core.Options
{
    public class IdentityProviderConfiguration
    {
        public string EntityId { get; set; }

        public string Name { get; set; }

        public string IssuerFormat { get; set; }

        [DefaultValue(
        "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified")]
        public string NameIdPolicyFormat { get; set; } =
        "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified";

        [DefaultValue("http://www.w3.org/2001/04/xmlenc#sha256")]
        public string DigestAlgorithm { get; set; } = "SHA256";

        [DefaultValue(
        "http://www.w3.org/2001/04/xmldsig-more#rsa-sha256")]
        public string HashingAlgorithm { get; set; } = "SHA256";

        [DefaultValue(
        "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect")]
        public string ProtocolBinding { get; set; } =
        "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect";

        public string SingleSignOnService { get; set; }

        public string SingleSignOutService { get; set; }

```

```
34          public string ArtifactResolveService { get; set; }
35      }
36  }
```

I ServiceProviderConfiguration

```
1  namespace Saml2.Authentication.Core.Options
2  {
3      public class ServiceProviderConfiguration
4      {
5          public string EntityId { get; set; }
6      }
7  }
```

J SamlService

```
using System;
using System.Text;
using System.Xml;
using dk.nita.saml20;
using Microsoft.AspNetCore.Http;
using Saml2.Authentication.Core.Bindings;
using Saml2.Authentication.Core.Factories;
using Saml2.Authentication.Core.Options;
using Saml2.Authentication.Core.Providers;
using Saml2.Authentication.Core.Validation;
using Saml2LogoutResponse =
 ↪  Saml2.Authentication.Core.Bindings.Saml2LogoutResponse;

namespace Saml2.Authentication.Core.Services
{
    internal class SamlService : ISamlService
    {
        private readonly ICertificateProvider
         ↪  _certificateProvider;
        private readonly IHttpArtifactBinding
         ↪  _httpArtifactBinding;
        private readonly IHttpRedirectBinding
         ↪  _httpRedirectBinding;
        private readonly IdentityProviderConfiguration
                    _identityProviderConfiguration;
        private readonly Saml2Configuration _saml2Configuration;
        private readonly ISaml2MessageFactory
         ↪  _saml2MessageFactory;
        private readonly ISaml2Validator _saml2Validator;
        private readonly ISamlProvider _samlProvider;
        private readonly ServiceProviderConfiguration
                        _serviceProviderConfiguration;

```

```
29          public SamlService(
30              IHttpRedirectBinding httpRedirectBinding,
31              IHttpArtifactBinding httpArtifactBinding,
32              ISaml2MessageFactory saml2MessageFactory,
33              ICertificateProvider certificateProvider,
34              ISamlProvider samlProvider,
35              ISaml2Validator saml2Validator,
36              Saml2Configuration saml2Configuration)
37          {
38              _httpRedirectBinding = httpRedirectBinding;
39              _httpArtifactBinding = httpArtifactBinding;
40              _saml2MessageFactory = saml2MessageFactory;
41              _certificateProvider = certificateProvider;
42              _samlProvider = samlProvider;
43              _saml2Validator = saml2Validator;
44              _saml2Configuration = saml2Configuration;
45              _identityProviderConfiguration =
46                  saml2Configuration.IdentityProviderConfiguration;
47              _serviceProviderConfiguration =
48                  saml2Configuration.ServiceProviderConfiguration;
49          }
50
51          public string GetAuthnRequest(
52              string authnRequestId,
53              string relayState,
54              string assertionConsumerServiceUrl)
55          {
56              var signingCertificate =
                ↪  _certificateProvider.GetCertificate();
57
58              var saml20AuthnRequest =
59                  _saml2MessageFactory
60                      .CreateAuthnRequest(
61                          authnRequestId,
62                              assertionConsumerServiceUrl);
63          // check protocol binding if supporting more than
            ↪  HTTP-REDIRECT
64              return _httpRedirectBinding.BuildAuthnRequestUrl(
65                  saml20AuthnRequest,
66                  signingCertificate.ServiceProvider.PrivateKey,
```

```
67                  _identityProviderConfiguration.HashingAlgorithm,
                     ↪  relayState);
68          }
69
70          public string GetLogoutRequest(
71              string logoutRequestId,
72              string sessionIndex,
73              string subject,
74              string relayState)
75          {
76              var signingCertificate =
                 ↪  _certificateProvider.GetCertificate();
77
78              var logoutRequest = _saml2MessageFactory
79                  .CreateLogoutRequest(logoutRequestId,
                     ↪  sessionIndex, subject);
80
81              return _httpRedirectBinding.BuildLogoutRequestUrl(
82                  logoutRequest,
83                  signingCertificate.ServiceProvider.PrivateKey,
84                  _identityProviderConfiguration.HashingAlgorithm,
85                  relayState);
86          }
87
88          public bool IsLogoutResponseValid(Uri uri, string
             ↪  originalRequestId)
89          {
90              var signingCertificate =
                 ↪  _certificateProvider.GetCertificate();
91
92              var logoutMessage =
93                  _httpRedirectBinding.GetLogoutResponseMessage(
94                  uri,
95                  signingCertificate
96                      .IdentityProvider
97                          .PublicKey.Key);
98
99              var logoutRequest =
                 ↪  _samlProvider.GetLogoutResponse(logoutMessage);
100
```

```
            _saml2Validator.CheckReplayAttack(
                                logoutRequest.InResponseTo,
                                originalRequestId);

            return logoutRequest.Status.StatusCode.Value
                                                ==
                    Saml2Constants.StatusCodes.Success;
        }

        public Saml2LogoutResponse GetLogoutReponse(Uri uri)
        {
            var signingCertificate =
                _certificateProvider.GetCertificate();

            var logoutResponse =
                _httpRedirectBinding.GetLogoutReponse(
                uri,
                signingCertificate
                    .IdentityProvider.PublicKey.Key);

            if (!_saml2Validator.ValidateLogoutRequestIssuer
            (logoutResponse.OriginalLogoutRequest.Issuer.Value,
                _identityProviderConfiguration.EntityId))
            {
                logoutResponse.StatusCode =
                        Saml2Constants.StatusCodes.RequestDenied;
            }

            return logoutResponse;
        }

        public string GetLogoutResponseUrl(
            Saml2LogoutResponse logoutResponse,
            string relayState)
        {
            var signingCertificate =
                _certificateProvider.GetCertificate();

            var response =
                _saml2MessageFactory.CreateLogoutResponse(
```

```
137                 logoutResponse.StatusCode,
138                 logoutResponse.OriginalLogoutRequest.ID);
139
140             return _httpRedirectBinding.BuildLogoutResponseUrl(
141                 response,
142                 signingCertificate.ServiceProvider.PrivateKey,
143                 _identityProviderConfiguration.HashingAlgorithm,
144                 relayState);
145         }
146
147         public Saml2Assertion HandleHttpRedirectResponse(
148             string base64EncodedSamlResponse,
149             string originalSamlRequestId)
150         {
151             var samlResponseDocument =
                ↪ _samlProvider.GetDecodedSamlResponse(
152                         base64EncodedSamlResponse,
                            ↪ Encoding.UTF8);
153             var samlResponseElement =
                ↪ samlResponseDocument.DocumentElement;
154
155             _saml2Validator.CheckReplayAttack(
156                                 samlResponseElement,
                                    ↪ originalSamlRequestId);
157
158             return
                ↪ !_saml2Validator.CheckStatus(samlResponseDocument)
159                 ? null
160                 : GetValidatedAssertion(samlResponseElement);
161         }
162
163
164         public Saml2Assertion
            ↪ HandleHttpArtifactResponse(HttpRequest request)
165         {
166             var signingCertificate =
                ↪ _certificateProvider.GetCertificate();
167
168             var artifact =
                ↪ _httpArtifactBinding.GetArtifact(request);
```

```
169             var stream =
                ↪ _httpArtifactBinding.ResolveArtifact(artifact,
170                 _identityProviderConfiguration
171                     .ArtifactResolveService,
172                 _serviceProviderConfiguration.EntityId,
173                 signingCertificate.ServiceProvider);
174
175             var artifactResponseElement =
176                             _samlProvider
177                                 .GetArtifactResponse(stream);
178
179             return
                ↪ GetValidatedAssertion(artifactResponseElement);
180         }
181
182         private Saml2Assertion GetValidatedAssertion(
183             XmlElement samlResponseElement)
184         {
185             var signingCertificate =
                ↪ _certificateProvider.GetCertificate();
186             var assertionElement =
187                 _samlProvider.GetAssertion(
188                     samlResponseElement,
189                     signingCertificate
190                         .ServiceProvider.PrivateKey);
191
192             return _saml2Validator.GetValidatedAssertion(
193                 assertionElement,
194                 signingCertificate
195                     .IdentityProvider.PublicKey.Key,
196                 _serviceProviderConfiguration.EntityId,
197                 _saml2Configuration.OmitAssertionSignatureCheck);
198         }
199     }
200 }
```

K SamlProvider

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
using System.Xml;
using dk.nita.saml20;
using dk.nita.saml20.Schema.Core;
using dk.nita.saml20.Schema.Protocol;
using dk.nita.saml20.Utils;
using Saml2.Authentication.Core.Bindings;

namespace Saml2.Authentication.Core.Providers
{
    internal class SamlProvider : ISamlProvider
    {
        public XmlDocument GetDecodedSamlResponse(
            string base64SamlResponse,
            Encoding encoding)
        {
            var doc = new XmlDocument
            {
                XmlResolver = null,
                PreserveWhitespace = true
            };
            var samlResponse = encoding.GetString(
                                Convert
                                .FromBase64String(
                                    base64SamlResponse));
            doc.LoadXml(samlResponse);
            return doc;
        }

        public XmlElement GetAssertion(
```

```
34              XmlElement xmlElement,
35              AsymmetricAlgorithm privateKey)
36          {
37              if (IsEncrypted(xmlElement))
38                  return GetDecriptedAssertion(xmlElement,
                    ↪ privateKey);
39
40              var assertionList = xmlElement
41                  .GetElementsByTagName(
42                          Assertion.ELEMENT_NAME,
                            ↪ Saml2Constants.ASSERTION);
43
44              var assertion = (XmlElement) assertionList[0];
45              if (assertion == null)
46                  throw new Saml2Exception("Missing assertion");
47
48              return assertion;
49          }
50
51          public LogoutResponse GetLogoutResponse(string
            ↪ logoutResponseMessage)
52          {
53              var doc = new XmlDocument
54              {
55                  XmlResolver = null,
56                  PreserveWhitespace = true
57              };
58              doc.LoadXml(logoutResponseMessage);
59
60              var logoutResponse =
61                  (XmlElement)doc.GetElementsByTagName(
62                      LogoutResponse.ELEMENT_NAME,
                        ↪ Saml2Constants.PROTOCOL)[0];
63              return Serialization
64              .DeserializeFromXmlString<LogoutResponse>(
65                                      logoutResponse.OuterXml);
66          }
67
68          public XmlElement GetArtifactResponse(Stream stream)
69          {
```

```
70              var parser = new HttpArtifactBindingParser(stream);
71              if (!parser.IsArtifactResponse())
72                  return null;
73
74              var status = parser.ArtifactResponse.Status;
75              if (status.StatusCode.Value !=
                ↪ Saml2Constants.StatusCodes.Success)
76                  throw new Exception(
77                      $"Illegal status: {status.StatusCode}
78                          for ArtifactResponse");
79
80              return parser.ArtifactResponse.Any.LocalName
81                  != Response.ELEMENT_NAME ? null :
                    ↪ parser.ArtifactResponse.Any;
82          }
83
84          public Status GetLogoutResponseStatus(string
            ↪ logoutResponseMessage)
85          {
86              var doc = new XmlDocument
87              {
88                  XmlResolver = null,
89                  PreserveWhitespace = true
90              };
91              doc.LoadXml(logoutResponseMessage);
92
93              var statElem = (XmlElement)doc.GetElementsByTagName(
94                              Status.ELEMENT_NAME,
                                ↪ Saml2Constants.PROTOCOL)[0];
95
96              return Serialization
97                  .DeserializeFromXmlString<Status>(
98                                      statElem.OuterXml);
99          }
100
101         public XmlElement GetDecriptedAssertion(
102             XmlElement xmlElement, AsymmetricAlgorithm
                ↪ privateKey)
103         {
104             var encryptedList =
```

```
105                 xmlElement.GetElementsByTagName(
106                     EncryptedAssertion.ELEMENT_NAME,
                        ↪ Saml2Constants.ASSERTION);
107             var assertion = (XmlElement) encryptedList[0];
108             if (assertion == null)
109                 throw new Saml2Exception("Missing assertion");
110
111             var encryptedAssertion
112                 = new Saml2EncryptedAssertion((RSA) privateKey);
113             encryptedAssertion.LoadXml(assertion);
114             encryptedAssertion.Decrypt();
115
116             return encryptedAssertion.Assertion.DocumentElement;
117         }
118
119         private static bool IsEncrypted(XmlElement element)
120         {
121             var encryptedList = element.GetElementsByTagName(
122             EncryptedAssertion.ELEMENT_NAME,
                ↪ Saml2Constants.ASSERTION);
123             return encryptedList.Count == 1;
124         }
125     }
126 }
```

L Saml2MessageFactory

```
using System;
using System.Collections.Generic;
using dk.nita.saml20;
using dk.nita.saml20.Schema.Core;
using dk.nita.saml20.Schema.Protocol;
using Saml2.Authentication.Core.Extensions;
using Saml2.Authentication.Core.Options;

namespace Saml2.Authentication.Core.Factories
{
    internal class Saml2MessageFactory : ISaml2MessageFactory
    {
        private readonly IdentityProviderConfiguration
                         _identityProviderConfiguration;
        private readonly Saml2Configuration _saml2Configuration;
        private readonly ServiceProviderConfiguration
                         _serviceProviderConfiguration;

        public Saml2MessageFactory(
            Saml2Configuration saml2Configuration)
        {
            _saml2Configuration = saml2Configuration;
            _serviceProviderConfiguration =
                saml2Configuration.ServiceProviderConfiguration;
            _identityProviderConfiguration =
                saml2Configuration.IdentityProviderConfiguration;
        }

        public Saml2AuthnRequest CreateAuthnRequest(
            string authnRequestId,
            string assertionConsumerServiceUrl)
        {
            var request = new Saml2AuthnRequest
```

```
34              {
35                  ID = authnRequestId,
36                  Issuer = _serviceProviderConfiguration.EntityId,
37                  ForceAuthn = _saml2Configuration.ForceAuth,
38                  IsPassive = _saml2Configuration.IsPassive,
39                  Destination =
40                      _identityProviderConfiguration
41                              .SingleSignOnService,
42                  IssuerFormat =
                    ↪ _identityProviderConfiguration.IssuerFormat,
43                  IssueInstant = DateTime.UtcNow,
44                  ProtocolBinding =
                    ↪ _identityProviderConfiguration.ProtocolBinding
45              };
46
47              request.Request.AssertionConsumerServiceURL =
48                                  assertionConsumerServiceUrl;
49
50              var audienceRestrictions = new
                ↪ List<ConditionAbstract>(1);
51              var audienceRestriction =
52                  new AudienceRestriction
53                      {Audience = new List<string>(1)
                        ↪ {_serviceProviderConfiguration.EntityId}};
54              audienceRestrictions.Add(audienceRestriction);
55              request.Request.Conditions =
56                                      new Conditions {Items =
                                        ↪ audienceRestrictions};
57
58              if (_saml2Configuration.AllowCreate.HasValue &&
59                  _identityProviderConfiguration.NameIdPolicyFormat
60                      .IsNotNullOrEmpty())
61                  request.Request.NameIDPolicy = new NameIDPolicy
62                  {
63                      AllowCreate =
                        ↪ _saml2Configuration.AllowCreate,
64                      Format = _identityProviderConfiguration
65                                  .NameIdPolicyFormat
66                  };
67
```

```
68              if (_saml2Configuration.AuthnContextComparisonType
69                      .IsNotNullOrEmpty())
70                  request.Request.RequestedAuthnContext =
71                                  new RequestedAuthnContext
72                  {
73                      Comparison =
74                          Enum.Parse<AuthnContextComparisonType>
75                      (_saml2Configuration
76                          .AuthnContextComparisonType),
77                      ComparisonSpecified = true,
78                      Items = _saml2Configuration
79                                      .AuthnContextComparisonItems
80                  };
81              return request;
82          }
83
84          public Saml2LogoutRequest CreateLogoutRequest(
85              string logoutRequestId,
86              string sessionIndex,
87              string subject)
88          {
89              var request = new Saml2LogoutRequest
90              {
91                  Issuer = _serviceProviderConfiguration.EntityId,
92                  Destination =
93                      _identityProviderConfiguration
94                          .SingleSignOutService,
95                  Reason = Saml2Constants.Reasons.User,
96                  SubjectToLogOut = new NameID()
97              };
98
99              request.Request.ID = logoutRequestId;
100
101             if (sessionIndex.IsNotNullOrEmpty())
102                 request.SessionIndex = sessionIndex;
103
104             if (subject.IsNotNullOrEmpty())
105                 request.SubjectToLogOut.Value = subject;
106
107             return request;
```

```
108         }
109
110         public Saml2LogoutResponse CreateLogoutResponse(
111             string statusCode,
112             string inResponseTo)
113         {
114             return new Saml2LogoutResponse
115             {
116                 StatusCode = statusCode,
117                 Issuer = _serviceProviderConfiguration.EntityId,
118                 InResponseTo = inResponseTo,
119                 Destination =
120                     _identityProviderConfiguration
121                         .SingleSignOutService
122             };
123         }
124     }
125 }
```

M Saml2ClaimFactory

```
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using Saml2.Authentication.Core.Authentication;
using Saml2.Authentication.Core.Extensions;

namespace Saml2.Authentication.Core.Factories
{
    internal class Saml2ClaimFactory : ISaml2ClaimFactory
    {
        public IList<Claim> Create(Saml2Assertion assertion)
        {
            var claims = new List<Claim>();
            if (!string.IsNullOrEmpty(assertion.Subject.Value))
            {
                claims.Add(new Claim(
                    Saml2ClaimTypes.Subject,
                     ↪ assertion.Subject.Value));
                claims.Add(new Claim(
                    Saml2ClaimTypes.Name,
                     ↪ assertion.Subject.Value));
                claims.Add(new Claim(
                    Saml2ClaimTypes.NameIdentifier,
                     ↪ assertion.Subject.Value));
            }

            claims.Add(assertion.SessionIndex.IsNotNullOrEmpty()
                ? new Claim(
                    Saml2ClaimTypes.SessionIndex,
                                assertion.SessionIndex)
                : new Claim(Saml2ClaimTypes.SessionIndex,
                 ↪ assertion.Id));

```

```
30              claims.AddRange(assertion.Attributes.Select(attribute
                ↪  =>
31                  new Claim(
32                      attribute.Name,
33                          attribute.AttributeValue[0]
34                              .ToString())));
35
36              return claims;
37          }
38      }
39  }
```

N CertificateProvider

```
using System.Security.Cryptography.X509Certificates;

namespace Saml2.Authentication.Core.Providers
{
    internal class CertificateProvider : ICertificateProvider
    {
        private readonly SigningCertificate _signingCertificate;

        public CertificateProvider(SigningCertificate
        ↪ signingCertificate)
        {
            _signingCertificate = signingCertificate;
        }

        public SigningCertificate GetCertificate()
        {
            return _signingCertificate;
        }
    }

    public class SigningCertificate
    {
        public SigningCertificate(
            X509Certificate2 identityProvider,
            X509Certificate2 serviceProvider)
        {
            IdentityProvider = identityProvider;
            ServiceProvider = serviceProvider;
        }

        public X509Certificate2 IdentityProvider { get; }

        public X509Certificate2 ServiceProvider { get; }
```

```
33          }
34      }
```

O HttpArtifactBinding

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using System.Xml;
using dk.nita.saml20.Utils;
using Microsoft.AspNetCore.Http;
using Saml2.Authentication.Core.Extensions;

namespace Saml2.Authentication.Core.Bindings
{
    /// <inheritdoc cref="HttpSoapBinding" />
    /// <summary>
    ///     Implementation of the artifact over HTTP SOAP binding.
    /// </summary>
    internal class HttpArtifactBinding : HttpSoapBinding, IHttpArtifactBinding
    {
        public bool IsValid(HttpRequest request)
        {
            return request?.Method == HttpMethods.Get &&
                   !string.IsNullOrEmpty(
                            request?.Query["SAMLart"]);
        }

        public string GetArtifact(HttpRequest request)
        {
            return request?.Query["SAMLart"];
        }

        public string GetRelayState(HttpRequest request)
        {
            var encodedRelayState =
                request?.Query["RelayState"].ToString();
```

```
32              return encodedRelayState.DeflateDecompress();
33          }
34
35          /// <inheritdoc />
36          /// <summary>
37          ///     Resolves an artifact.
38          /// </summary>
39          /// <returns>A stream containing the artifact response
             ↪  from the IdP</returns>
40          public Stream ResolveArtifact(
41              string artifact,
42              string artifactResolveEndpoint,
43              string serviceProviderId,
44              X509Certificate2 cert)
45          {
46              if (artifactResolveEndpoint == null)
47                  throw new InvalidOperationException(
48                          "Received artifact from unknown IDP.");
49
50              var resolve = new Saml2ArtifactResolve
51              {
52                  Issuer = serviceProviderId,
53                  Artifact = artifact
54              };
55
56              var doc = resolve.GetXml();
57              if (doc.FirstChild is XmlDeclaration)
58                  doc.RemoveChild(doc.FirstChild);
59
60              XmlSignatureUtils.SignDocument(doc, resolve.ID,
                 ↪  cert);
61
62              var artifactResolveString = doc.OuterXml;
63              return GetResponse(artifactResolveEndpoint,
                 ↪  artifactResolveString);
64          }
65      }
66  }
```

P HttpRedirectBinding

```
using System;
using System.Security.Cryptography;
using System.Text;
using System.Web;
using dk.nita.saml20;
using dk.nita.saml20.Bindings;
using Microsoft.AspNetCore.Http;
using Saml2.Authentication.Core.Bindings.SignatureProviders;
using Saml2.Authentication.Core.Extensions;

namespace Saml2.Authentication.Core.Bindings
{
    /// <summary>
    ///     Handles the creation of redirect locations when using the HTTP redirect binding, which is outlined in [SAMLBind]
    ///     section 3.4.
    /// </summary>
    internal class HttpRedirectBinding : IHttpRedirectBinding
    {
        private const string SamlResponseQueryKey = "SamlResponse";

        private const string SamlRequestQueryKey = "SAMLRequest";

        private const string SamlRelayStateQueryKey = "RelayState";

        private readonly ISignatureProviderFactory _signatureProviderFactory;

        public HttpRedirectBinding(
            ISignatureProviderFactory signatureProviderFactory)
```

```
29          {
30              _signatureProviderFactory = signatureProviderFactory;
31          }
32
33          public bool IsValid(HttpRequest request)
34          {
35              if (request == null)
36                  return false;
37
38              if (request.Method == HttpMethods.Get)
39                  return
                    ↪  request.Query.ContainsKey(SamlRequestQueryKey)
                    ↪  ||
40                         request.Query.ContainsKey(
41                                              SamlResponseQueryKey);
42
43              if (request.Method != HttpMethods.Post)
44                  return false;
45
46              var form = request.Form;
47              return form != null && form.ContainsKey(
48                                         SamlResponseQueryKey);
49          }
50
51          public bool IsLogoutRequest(HttpRequest request)
52          {
53              if (request == null)
54                  return false;
55
56              if (request.Method == HttpMethods.Get)
57                  return request.Query.ContainsKey(
58                                         SamlRequestQueryKey);
59
60              if (request.Method != HttpMethods.Post)
61                  return false;
62
63              var form = request.Form;
64              return form != null &&
                    ↪  form.ContainsKey(SamlRequestQueryKey);
65          }
```

```
66
67          public Saml2Response GetResponse(HttpRequest request)
68          {
69              if (request.Method == HttpMethods.Get)
70                  return new Saml2Response
71                  {
72                      Response =
                        ↪ request.Query[SamlResponseQueryKey],
73                      RelayState =
                        ↪ request.Query[SamlRelayStateQueryKey]
74                      .ToString()?.DeflateDecompress()
75                  };
76
77              if (request.Method != HttpMethods.Post)
78                  return null;
79
80              var form = request.Form;
81              if (form == null)
82                  return null;
83
84              return new Saml2Response
85              {
86                  Response = form[SamlResponseQueryKey],
87                  RelayState =
88                      form[SamlRelayStateQueryKey].ToString()
89                          ?.DeflateDecompress()
90              };
91          }
92
93          public string GetCompressedRelayState(HttpRequest
             ↪ request)
94          {
95              if (request.Method == HttpMethods.Get)
96                  return request.Query[SamlRelayStateQueryKey]
97                                        .ToString();
98
99              if (request.Method != HttpMethods.Post)
100                 return null;
101
102             var form = request.Form;
```

```
103
104             return form?[SamlRelayStateQueryKey].ToString();
105         }
106
107         public string BuildAuthnRequestUrl(
108             Saml2AuthnRequest saml2AuthnRequest,
109             AsymmetricAlgorithm signingKey,
110             string hashingAlgorithm,
111             string relayState)
112         {
113             var request = saml2AuthnRequest.GetXml().OuterXml;
114             return BuildRequestUrl(
115                 signingKey,
116                     hashingAlgorithm,
117                         relayState,
118                             request,
119                                 saml2AuthnRequest.Destination);
120         }
121
122         public string BuildLogoutRequestUrl(
123             Saml2LogoutRequest saml2LogoutRequest,
124             AsymmetricAlgorithm signingKey,
125             string hashingAlgorithm,
126             string relayState)
127         {
128             var request = saml2LogoutRequest.GetXml().OuterXml;
129             return BuildRequestUrl(
130                 signingKey,
131                     hashingAlgorithm,
132                         relayState,
133                             request,
134                                 saml2LogoutRequest.Destination);
135         }
136
137         public string BuildLogoutResponseUrl(
138             dk.nita.saml20.Saml2LogoutResponse logoutResponse,
139             AsymmetricAlgorithm signingKey,
140             string hashingAlgorithm,
141             string relayState)
142         {
```

```
143             var response = logoutResponse.GetXml().OuterXml;
144             return BuildRequestUrl(
145                 signingKey,
146                     hashingAlgorithm,
147                         relayState,
148                             response,
149                                 logoutResponse.Destination);
150         }
151 
152         public string GetLogoutResponseMessage(Uri uri,
             ↪ AsymmetricAlgorithm key)
153         {
154             var parser = new HttpRedirectBindingParser(uri);
155             if (key == null)
156                 throw new ArgumentNullException(nameof(key));
157 
158             if (!parser.IsSigned)
159                 throw new InvalidOperationException(
160                     "Query is not signed, so there is no
                         ↪ signature to verify.");
161 
162             // Validates the signature using the public part of
                 ↪ the asymmetric key given as parameter.
163             var signatureProvider =
164                 _signatureProviderFactory
165                 .CreateFromAlgorithmUri(
166                     key.GetType(),
167                         parser.SignatureAlgorithm);
168             if (!signatureProvider.VerifySignature(
169                     key,
170                         Encoding.UTF8.GetBytes(
171                             parser.SignedQuery),
172                                 parser.DecodeSignature()))
173 
174                 throw new InvalidOperationException(
175                             "Logout request signature
                                 ↪ verification failed");
176 
177             return parser.Message;
178         }
```

```
179
180         public Saml2LogoutResponse GetLogoutReponse(
181             Uri uri,
182             AsymmetricAlgorithm key)
183         {
184             var response = new Saml2LogoutResponse();
185             var parser = new HttpRedirectBindingParser(uri);
186             if (key == null)
187                 throw new ArgumentNullException(nameof(key));
188
189             response.OriginalLogoutRequest =
                ↪ parser.LogoutRequest;
190
191             if (!parser.IsSigned)
192                 response.StatusCode =
                    ↪ Saml2Constants.StatusCodes.RequestDenied;
193
194             // Validates the signature using the public part of
                ↪ the asymmetric key given as parameter.
195             var signatureProvider =
196                 _signatureProviderFactory
197                     .CreateFromAlgorithmUri(
198                     key.GetType(), parser.SignatureAlgorithm);
199
200             if (!signatureProvider.VerifySignature(
201                     key,
202                         Encoding.UTF8.GetBytes(
203                             parser.SignedQuery),
204                                 parser.DecodeSignature()))
205                 response.StatusCode =
                    ↪ Saml2Constants.StatusCodes.RequestDenied;
206
207             response.StatusCode =
                ↪ Saml2Constants.StatusCodes.Success;
208
209             return response;
210         }
211
212         /// <summary>
213         ///     If an asymmetric key has been specified, sign the
                ↪ request.
```

```
214         /// </summary>
215         private void AddSignature(
216             StringBuilder result,
217             AsymmetricAlgorithm signingKey,
218             ShaHashingAlgorithm hashingAlgorithm)
219         {
220             if (signingKey == null)
221                 return;
222
223             result.Append(
224                 string.Format("&{0}=",
                     ↪ HttpRedirectBindingConstants.SigAlg));
225
226             var signingProvider =
227                 _signatureProviderFactory
228                     .CreateFromAlgorithmName(
229                         signingKey.GetType(), hashingAlgorithm);
230
231             var urlEncoded =
                 ↪ signingProvider.SignatureUri.UrlEncode();
232             result.Append(urlEncoded.UpperCaseUrlEncode());
233
234             // Calculate the signature of the URL as described in
                 ↪ [SAMLBind] section 3.4.4.1.
235             var signature = signingProvider.SignData(
236                     signingKey,
237                         Encoding.UTF8.GetBytes(
238                                 result.ToString()));
239
240             result.AppendFormat(
241                 "&{0}=", HttpRedirectBindingConstants.Signature);
242
243             result.Append(
244                 HttpUtility.UrlEncode(
245                     Convert.ToBase64String(signature)));
246         }
247
248         private string BuildRequestUrl(
249             AsymmetricAlgorithm signingKey,
250             string hashingAlgorithm,
```

```
251             string relayState,
252             string request,
253             string destination)
254         {
255             var shaHashingAlgorithm = signatureProviderFactory
256                             .ValidateShaHashingAlgorithm(
257                                 hashingAlgorithm);
258 
259             // Check if the key is of a supported type.
                ↪  [SAMLBind] sect. 3.4.4.1 specifies this.
260             if (!(signingKey is RSA || signingKey is DSA ||
                ↪  signingKey == null))
261                 throw new ArgumentException(
262                         "Signing key must be an instance of
                            ↪  either RSA or DSA.");
263 
264             var result = new StringBuilder();
265             result.AddMessageParameter(request, null);
266             result.AddRelayState(request, relayState);
267             AddSignature(result, signingKey,
                ↪  shaHashingAlgorithm);
268 
269             return $"{destination}?{result}";
270         }
271     }
272 }
```

Q Saml2StringBuilderExtensions

```
using System;
using System.Text;
using dk.nita.saml20.Bindings;

namespace Saml2.Authentication.Core.Extensions
{
    public static class Saml2StringBuilderExtensions
    {
        /// <summary>
        ///     If the RelayState property has been set, this method adds it to the query string.
        /// </summary>
        /// <param name="result"></param>
        /// <param name="relayState"></param>
        /// <param name="request"></param>
        public static void AddRelayState(
            this StringBuilder result,
            string request,
            string relayState)
        {
            if (relayState == null)
                return;

            result.Append("&RelayState=");
            // Encode the relay state if we're building a request. Otherwise, append unmodified.
            result.Append(request != null
                ? relayState.DeflateEncode().UrlEncode()
                : relayState);
        }

        /// <summary>
        ///     Depending on which one is specified, this method adds the SAMLRequest or SAMLResponse parameter to the URL query.
```

```
32          /// </summary>
33          public static void AddMessageParameter(
34              this StringBuilder result,
35              string request,
36              string response)
37          {
38              if (!(response == null || request == null))
39                  throw new Exception(
40                      "Request or Response property MUST be set.");
41
42              string value;
43              if (request != null)
44              {
45                  result.AppendFormat(
46                          "{0}=", HttpRedirectBindingConstants
47                              .SamlRequest);
48                  value = request;
49              }
50              else
51              {
52                  result.AppendFormat(
53                          "{0}=", HttpRedirectBindingConstants
54                              .SamlResponse);
55                  value = response;
56              }
57
58              var encoded = value.DeflateEncode();
59              var urlEncoded = encoded.UrlEncode();
60              result.Append(urlEncoded.UpperCaseUrlEncode());
61          }
62
63          public static string TrimSpecialCharacters(this string
            ↪ str)
64          {
65              var sb = new StringBuilder();
66              foreach (var c in str)
67                  if (c >= '0' && c <= '9'
68                      || c >= 'A' && c <= 'Z'
69                      || c >= 'a' && c <= 'z'
70                      || c == '.'
```

```
71                      || c == '_')
72                          sb.Append(c);
73              return sb.ToString();
74          }
75      }
76  }
```

R Saml2StringExtensions

```
using System;
using System.IO;
using System.IO.Compression;
using System.Text;
using System.Web;

namespace Saml2.Authentication.Core.Extensions
{
    public static class Saml2StringExtensions
    {
        public static bool IsNotNullOrEmpty(this string value)
        {
            return !string.IsNullOrEmpty(value);
        }

        public static bool IsNullOrEmpty(this string value)
        {
            return string.IsNullOrEmpty(value);
        }

        public static string UrlEncode(this string value)
        {
            return HttpUtility.UrlEncode(value);
        }

        public static string UrlDecode(this string value)
        {
            return HttpUtility.UrlDecode(value);
        }

        /// <summary>
        ///     Uses DEFLATE compression to compress the input value. Returns the result as a Base64 encoded string.
        /// </summary>
```

```
34          public static string DeflateEncode(this string value)
35          {
36              var memoryStream = new MemoryStream();
37              using (var writer = new StreamWriter(
38                      new DeflateStream(memoryStream,
39                          CompressionMode.Compress, true),
40                  new UTF8Encoding(false)))
41              {
42                  writer.Write(value);
43                  writer.Close();
44
45                  return Convert.ToBase64String(
46                          memoryStream.GetBuffer(), 0, (int)
47                              memoryStream.Length,
48                      Base64FormattingOptions.None);
49              }
50          }
51
52          /// <summary>
53          ///     Take a Base64-encoded string, decompress the
            ↪  result using the DEFLATE algorithm and return the
            ↪  resulting
54          ///     string.
55          /// </summary>
56          public static string DeflateDecompress(this string value)
57          {
58              var encoded = Convert.FromBase64String(value);
59              var memoryStream = new MemoryStream(encoded);
60
61              var result = new StringBuilder();
62              using (var stream = new DeflateStream(
63                          memoryStream,
                             ↪  CompressionMode.Decompress))
64              {
65                  var testStream = new StreamReader(
66                                  new BufferedStream(stream),
                                     ↪  Encoding.UTF8);
67                  // It seems we need to "peek" on the StreamReader
                     ↪  to get it started. If we don't do this, the
                     ↪  first call to
```

```
68                  // ReadToEnd() will return string.empty.
69                  var peek = testStream.Peek();
70                  result.Append(testStream.ReadToEnd());
71
72                  stream.Close();
73              }
74              return result.ToString();
75          }
76
77          /// <summary>
78          ///     Uppercase the URL-encoded parts of the string.
            ↪  Needed because Ping does not seem to be able to
            ↪  handle lower-cased
79          ///     URL-encodings.
80          /// </summary>
81          public static string UpperCaseUrlEncode(this string
            ↪  value)
82          {
83              var result = new StringBuilder(value);
84              for (var i = 0; i < result.Length; i++)
85                  if (result[i] == '%')
86                  {
87                      result[++i] = char.ToUpper(result[i]);
88                      result[++i] = char.ToUpper(result[i]);
89                  }
90              return result.ToString();
91          }
92      }
93  }
```

S Identity provider metadata

```
<md:EntityDescriptor
 ↪  xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
 ↪  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 ↪  entityID="https://saml2login.my.salesforce.com"
 ↪  validUntil="2028-05-05T09:46:30.439Z">
<md:IDPSSODescriptor
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
<md:KeyDescriptor use="signing">
<ds:KeyInfo>
<ds:X509Data>
<ds:X509Certificate>
MIIErDCCA5SgAwIBAgIOAWMp4yXaAAAAAAcFp9YwDQYJKoZIhvcNAQELBQAwgZAx
KDAmBgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAxOF8wNjQyMjIxGDAWBgNV
BAsMDzAwRDFyMDAwMDAyQ3YzcjEXMBUGA1UECgwOU2FsZXNmb3JjZS5jb20xFjAU
BgNVBAcMDVNhbiBGcmFuY2lzY28xCzAJBgNVBAgMAkNBMQwwCgYDVQQGEwNVU0Ew
HhcNMTgwNTA0MDYOMjIyWhcNMTkwNTA0MDAwMDAwWjCBkDEoMCYGA1UEAwwfU2Vs
ZlNpZ25lZENlcnRfMDRNYXkyMDE4XzA2NDIyMjEYMBYGA1UECwwPMDBEMXIw
MDAwMDJDdjNyMRcwFQYDVQQKDA5TYWxlc2ZvcmNlLmNvbTEWMBQGA1UEBwwNU2F
uIEZyYW5jaXNjbzELMAkGA1UECAwCQ0ExDDAKBgNVBAYTA1VTQTCCASIwDQYJKo
ZIhvcNAQEBBQADggEPADCCAQoCggEBAKIF3ftUDKu5JKhFxkhRwCg9jZaPqJyXQ
0Q9vuaQ+CDLn+Zya4DZWaOEMIVPaL5b6ArvrT9W0QGbDNWRdyoHeL2iaWHuKU+
kFe+pvwe1H4sSiqioRRnK8KY4xxmDiajIPISEgcPUnONNd39zWvar6Q1/IwytZ
QvOQr9YvH5nvGmgcjeqXbNuAhOUfaCn/Qt+ssiCuMY+WRr+iDsYOfMyZbj75+2
Eb03N+HZ3C+xJ5YsseNzgPr5zj6ktH93llOhcO5C/OMl/l7HdFmFgkNMKGDKYA
4qlNMIlMqDrRrSC8bhcxnGJ/7L40kdw8CWg+XqKTRwVtu7377K+uo8AMuBCCEE
CAwEAAaOCAQAwgf0wHQYDVR0OBBYEFPIqUh1nbrZhKxSjg5z2LyiGMVqgMA8GA
1UdEwEB/wQFMAMBAf8wgcoGA1UdIwSBwjCBv4AU8ipSHWdutmErFKODnPYvKIY
xWqChgZakgZMwgZAxKDAmBgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAx
OF8wNjQyMjIxGDAWBgNVBAsMDzAwRDFyMDAwMDAyQ3YzcjEXMBUGA1UECgwOU
2FsZXNmb3JjZS5jb20xFjAUBgNVBAcMDVNhbiBGcmFuY2lzY28xCzAJBgNVBA
gMAkNBMQwwCgYDVQQGEwNVU0GCDgFjKeMl2gAAAAAHBafWMA0GCSqGSIb3DQE
BCwUAA4IBAQCeTzfDa72aXQ60wrWCYuEzB75Kj/3BaAAHSjb22jbcNVQPncEP
phoP+baMEHDlUf5XNDZx5octD3hrKwGe14wcmr8LaBFrRDwwomVepP84wuz0
```

```
Hoqi+qrvTpFZyhDsvyyHEKTaNDNeQtkzbc3VCqsUW5KBYIXukys26h1pmQDm
HcMmkjMErY1yYAZu7mhM965ZDT90W1x8spWT/p82p/Ak6mjW/GcxPQnM4j9Nv
0JucOf1SfNjTAIDTgonPNf2PoNUmgLgXcan6S38bZBCRWfdC4f2GohfiOuScD
O+4pMFJ9yv/zm2ldh/j0cFe2fhknbCu2zXeWCnFgx31kE5
ZzSj
</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</md:KeyDescriptor>
<md:SingleLogoutService
 ↪  Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location=
"https://saml2login.my.salesforce.com/../logout"/>
<md:SingleLogoutService
 ↪  Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
Location=
"https://saml2login.my.salesforce.com/../logout"/>
<md:NameIDFormat>
urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified
</md:NameIDFormat>
<md:SingleSignOnService
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
 ↪  Location="https://saml2login.my.salesforce.com/../HttpPost"/>
<md:SingleSignOnService
Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
Location=
"https://saml2login.my.salesforce.com/../HttpRedirect"/>
</md:IDPSSODescriptor>
</md:EntityDescriptor>
```

T Base64 encoded SAMLResponse

```
PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz48c2FtbHA6bWVzc2FnZSB4bWxuczpzYW1scD0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnByb3RvY29sIiBEZXN0aW5hdGlvbj0iaHR0cHM6Ly9sb2NhbGhvc3Q6NDQzMDEvQWNjb3VudC9Bc3NlcnRpb25Db25zdW1lciIgSUQ9Il9hNmVlYjI2Yi1kMTI3LTQ0ZDAtOWM5ZS1jZDQ3MDQwOWY5ZiIgSW5SZXNwb25zZVRvPSJfNjE1YjQ5ZS1mMGQzLTRlOTUtOWI4Ni1iYjY1ZjE3NjQwMDkiIElzc3VlSW5zdGFudD0iMjAyMS0wNC0yOFQyMjo0MDo1NC4yMjVaIiBWZXJzaW9uPSIyLjAiPjxzYW1sOklzc3VlciB4bWxuczpzYW1sPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoyLjA6YXNzZXJ0aW9uIiBGb3JtYXQ9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDpuYW1laWQtZm9ybWF0OmVudGl0eSI+aHR0cHM6Ly9sb2NhbGhvc3Q6NDQzMDEvQWNjb3VudC9Bc3NlcnRpb25Db25zdW1lcjwvc2FtbDpJc3N1ZXI+PHNhbWxwOlN0YXR1cyB4bWxuczpzYW1scD0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnByb3RvY29sIj48c2FtbHA6U3RhdHVzQ29kZSBWYWx1ZT0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnN0YXR1czpTdWNjZXNzIi8+PC9zYW1scDpTdGF0dXM+PHNhbWw6RW5jcnlwdGVkQXNzZXJ0aW9uIHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDphc3NlcnRpb24iPjx4ZW5jOkVuY3J5cHRlZERhdGEgeG1sbnM6eGVuYz0iaHR0cDovL3d3dy53My5vcmcvMjAwMS8wNC94bWxlbmMjIiBUeXBlPSJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGVuYyNFbGVtZW50Ij48eGVuYzpFbmNyeXB0aW9uTWV0aG9kIEFsZ29yaXRobT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS8wNC94bWxlbmMjYWVzMjU2LWNiYyIgLz48S2V5SW5mbyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC8wOS94bWxkc2lnIyI+PGVuY0tleSBSZWNpcGllbnQ9Imh0dHBzOi8vc2FtbC5vbmVsb2dpbi5jb20vc2FtbDIvc2VydmljZV9wcm92aWRlci5waHAiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGVuYyMiPjxlbmM6RW5jcnlwdGlvbk1ldGhvZCBBbGdvcml0aG09Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvMDQveG1sZW5jI3JzYS1vYWVwLW1nZjFwIiAvPjxlbmM6Q2lwaGVyRGF0YSB4bWxuczplbmM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvMDQveG1sZW5jIyI+PGVuYzpDaXBoZXJWYWx1ZT5ZMlRqZ2ZqZ2ZqZ2Z
```

U Decoded SAMLResponse

```
<?xml version="1.0" encoding="UTF-8"?>
<samlp:Response
 ↪ xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
Destination=
"https://localhost:44344/Saml2/AssertionConsumerService"
ID="_a2f0a6ec822e51674785ac43a63c5eab1525535780073"
InResponseTo=
"b3dc5923fff09491d9f66ec6000c"
IssueInstant="2018-05-05T15:56:20.073Z" Version="2.0">
<saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 ↪ Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
https://saml2login.my.salesforce.com</saml:Issuer>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod
 ↪ Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<ds:SignatureMethod
Algorithm=
"http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference
 ↪ URI="#_a2f0a6ec822e51674785ac43a63c5eab1525535780073">
<ds:Transforms>
<ds:Transform
Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
<ds:Transform
 ↪ Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ec:InclusiveNamespaces
 ↪ xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"
 ↪ PrefixList="ds saml samlp xs xsi"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod
 ↪ Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```
<ds:DigestValue>
rMTl5dDORsJGVaXbhWyoYp49HDc=
</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>heKcp+99s0dYWOhiJXu1WYZ3Blh4NNFg1Rx/hHuM
/kOmY6vsjeaENO3m64PwBDhqVVxfzkJMf6Eh
e7lE2ALX/cVaDT4i6HlZkvcf2WEO9NNOEX5hPV/YT4AL6RYRL/
96AusIjFS5LstJ8XMGrmu4sOWnpmG3pBTBXNj4WE/+KTMnOjGp
cGhYgLaMc61h1hScTYfXLBSz8RMLLyQIGG1Faxj07PWlSCiovO6q
x6IokwCwlINI1nLyPwsJuS0toTDiYPzPT3zbi5FUzD/TnBkeShT
2LY1ZLGLqFoskhOD+/PBUeqQ7bFy62xtbcbJsMl1OVXUafwgmgi
/sobTENl1JFg==</ds:SignatureValue>
<ds:KeyInfo>
<ds:X509Data>
<ds:X509Certificate>
MIIErDCCA5SgAwIBAgIOAWMp4yXaAAAAAAcFp9YwDQYJKoZIhvcNAQELBQAwgZAx
KDAmBgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAxOF8wNjQyMjIxGDAWBgNV
BAsMDzAwRDFyMDAwMDAyQ3YzcjEXMBUGA1UECgwOU2FsZXNmb3JjZS5jb20xFjAU
BgNVBAcMDVNhbiBGcmFuY2lzY28xCzAJBgNVBAgMAkNBMQwwCgYDVQQGEwNVU0Ew
HhcNMTgwNTA0MDY0MjIyWhcNMTkwNTA0MDAwMDAwWjCBkDEoMCYGA1UEAwwfU2Vs
ZlNpZ25lZENlcnRfMDRNYXkyMDE4XzA2NDIyMjEYMBYGA1UECwwPMDBEMXIw
MDAwMDJDdjNyMRcwFQYDVQQKDA5TYWxlc2ZvcmNlLmNvbTEWMBQGA1UEBwwNU2F
uIEZyYW5jaXNjbzELMAkGA1UECAwCQ0ExDDAKBgNVBAYTA1VTQTCCASIwDQYJKo
ZIhvcNAQEBBQADggEPADCCAQoCggEBAKIF3ftUDKu5JKhFxkhRwCg9jZaPqJyXQ
0Q9vuaQ+CDLn+Zya4DZWaOEMIVPaL5b6ArvrT9W0QGbDNWRdyoHeL2iaWHuKU+
kFe+pvwe1H4sSiqioRRnK8KY4xxmDiajIPISEgcPUnONNd39zWvar6Q1/Iwytz
QvOQr9YvH5nvGmgcjeqXbNuAhOUfaCn/Qt+ssiCuMY+WRr+iDsYOfMyZbj75+2
Eb03N+HZ3C+xJ5YsseNzgPr5zj6ktH93llOhcO5C/0Ml/l7HdFmFgkNMKGDKYA
4qlNMIlMqDrRrSC8bhcxnGJ/7L40kdw8CWg+XqKTRwVtu7377K+uo8AMuBCCEE
CAwEAAaOCAQAwgf0wHQYDVR0OBBYEFPIqUh1nbrZhKxSjg5z2LyiGMVqgMA8GA
1UdEwEB/wQFMAMBAf8wgcoGA1UdIwSBwjCBv4AU8ipSHWdutmErFKODnPYvKIY
xWqChgZakgZMwgZAxKDAmBgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAx
OF8wNjQyMjIxGDAWBgNVBAsMDzAwRDFyMDAwMDAyQ3YzcjEXMBUGA1UECgwOU
2FsZXNmb3JjZS5jb20xFjAUBgNVBAcMDVNhbiBGcmFuY2lzY28xCzAJBgNVBA
gMAkNBMQwwCgYDVQQGEwNVU0GCDgFjKeMl2gAAAAAHBafWMA0GCSqGSIb3DQE
BCwUAA4IBAQCeTzfDa72aXQ60wrWCYuEzB75Kj/3BaAAHSjb22jbcNVQPncEP
phoP+baMEHDlUf5XNDZx5octD3hrKwGe14wcmr8LaBFrRDwwomVepP84wuz0
Hoqi+qrvTpFZyhDsvyyHEKTaNDNeQtkzbc3VCqsUW5KBYIXukys26h1pmQDm
HcMmkjMErY1yYAZu7mhM965ZDT90W1x8spWT/p82p/Ak6mjW/GcxPQnM4j9Nv
```

```
0JucOf1SfNjTAIDTgonPNf2PoNUmgLgXcan6S38bZBCRWfdC4f2GohfiOuScD
O+4pMFJ9yv/zm2ldh/j0cFe2fhknbCu2zXeWCnFgx31kE5
ZzSj</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</ds:Signature>
<samlp:Status>
<samlp:StatusCode
 ↪  Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<saml:Assertion
 ↪  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 ↪  ID="_f580b191d80f081b766398a5738360aa1525535780073"
 ↪  IssueInstant="2018-05-05T15:56:20.073Z" Version="2.0">
<saml:Issuer
 ↪  Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
https://saml2login.my.salesforce.com</saml:Issuer>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod
 ↪  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<ds:SignatureMethod
 ↪  Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference
 ↪  URI="#_f580b191d80f081b766398a5738360aa1525535780073">
<ds:Transforms>
<ds:Transform Algorithm=
"http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
<ds:Transform
 ↪  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
<ec:InclusiveNamespaces
 ↪  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"
 ↪  PrefixList="ds saml xs xsi"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod
 ↪  Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>YKJC6DmbJUociDCUODISdHw9RO8=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
```

```
<ds:SignatureValue>
HlSy0lGGSBys3VGdpj7iNmO+PZpN90wN/
vhPA4wZse5ItWdXF+V4mnLrK8L+RXBHYwg3wWfzuY9Z
JdyX7nwZwI9jTMWEvREkbQS4SIvCFa89pNMZs7qi0fLfj
Rfz9B9OLeFdeNabw3A67ouVsC8Ss4KABkPNz1ZNydcBSq
Sv2AKEUKPLueGzRBBKhDoDV7LvYczZ/uNkRE1QPPzU926
spRyU4ASBdj1bDS1bhIB9e7XePE7IIsGF9ZC7ASC7SVoG
ibRsRLk8wlZzn+yiFkFU9K2YeeN79Grh5XJZ4geo7EtHK
v5RAAe3+ObU/scQAZLispNmY/5ky20fwSVosArpvw==
</ds:SignatureValue>
<ds:KeyInfo>
        <ds:X509Data>
      <ds:X509Certificate>
MIIErDCCA5SgAwIBAgIOAWMp4yXaAAAAAAcFp9YwDQYJKoZIhvcNAQELBQAwgZAx
KDAmBgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAxOF8wNjQyMjIxGDAWBgNV
BAsMDzAwRDFyMDAwMDAyQ3YzcjEXMBUGA1UECgwOU2FsZXNmb3JjZS5jb20xFjAU
BgNVBAcMDVNhbiBGcmFuY2lzY28xCzAJBgNVBAgMAkNBMQwwCgYDVQQGEwNVU0Ew
HhcNMTgwNTA0MDY0MjIyWhcNMTkwNTA0MDAwMDAwWjCBkDEoMCYGA1UEAwwfU2Vs
ZlNpZ25lZENlcnRfMDRNYXkyMDE4XzA2NDIyMjEYMBYGA1UECwwPMDBEMXIw
MDAwMDJDdjNyMRcwFQYDVQQKDA5TYWxlc2ZvcmNlLmNvbTEWMBQGA1UEBwwNU2F
uIEZyYW5jaXNjbzELMAkGA1UECAwCQ0ExDDAKBgNVBAYTA1VTQTCCASIwDQYJKo
ZIhvcNAQEBBQADggEPADCCAQoCggEBAKIF3ftUDKu5JKhFxkhRwCg9jZaPqJyXQ
0Q9vuaQ+CDLn+Zya4DZWaOEMIVPaL5b6ArvrT9W0QGbDNWRdyoHeL2iaWHuKU+
kFe+pvwe1H4sSiqioRRnK8KY4xxmDiajIPISEgcPUnONNd39zWvar6Q1/Iwytz
QvOQr9YvH5nvGmgcjeqXbNuAhOUfaCn/Qt+ssiCuMY+WRr+iDsYOfMyZbj75+2
Eb03N+HZ3C+xJ5YsseNzgPr5zj6ktH93llOhcO5C/0Ml/l7HdFmFgkNMKGDKYA
4qlNMIlMqDrRrSC8bhcxnGJ/7L40kdw8CWg+XqKTRwVtu7377K+uo8AMuBCCEE
CAwEAAaOCAQAwgf0wHQYDVR0OBBYEFPIqUh1nbrZhKxSjg5z2LyiGMVqgMA8GA
1UdEwEB/wQFMAMBAf8wgcoGA1UdIwSBwjCBv4AU8ipSHWdutmErFKODnPYvKIY
xWqChgZakgZMwgZAxKDAmBgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAx
OF8wNjQyMjIxGDAWBgNVBAsMDzAwRDFyMDAwMDAyQ3YzcjEXMBUGA1UECgwOU
2FsZXNmb3JjZS5jb20xFjAUBgNVBAcMDVNhbiBGcmFuY2lzY28xCzAJBgNVBA
gMAkNBMQwwCgYDVQQGEwNVU0GCDgFjKeMl2gAAAAAHBafWMA0GCSqGSIb3DQE
BCwUAA4IBAQCeTzfDa72aXQ60wrWCYuEzB75Kj/3BaAAHSjb22jbcNVQPncEP
phoP+baMEHDlUf5XNDZx5octD3hrKwGe14wcmr8LaBFrRDwwomVepP84wuz0
Hoqi+qrvTpFZyhDsvyyHEKTaNDNeQtkzbc3VCqsUW5KBYIXukys26h1pmQDm
HcMmkjMErY1yYAZu7mhM965ZDT90W1x8spWT/p82p/Ak6mjW/GcxPQnM4j9Nv
0JucOf1SfNjTAIDTgonPNf2PoNUmgLgXcan6S38bZBCRWfdC4f2GohfiOuScD
O+4pMFJ9yv/zm2ldh/j0cFe2fhknbCu2zXeWCnFgx31kE5
ZzSj
```

```
</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</ds:Signature>
<saml:Subject>
<saml:NameID
Format=
"urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
jkmu@xxxxxxxx.com</saml:NameID>
<saml:SubjectConfirmation
 ↪  Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
<saml:SubjectConfirmationData
InResponseTo=
"b3dc5923fff09491d9f66ec6000c"
 ↪  NotOnOrAfter="2018-05-05T16:01:20.073Z"
Recipient=
"https://localhost:44344/Saml2/AssertionConsumerService"/>
</saml:SubjectConfirmation>
</saml:Subject>
<saml:Conditions NotBefore="2018-05-05T15:55:50.073Z"
 ↪  NotOnOrAfter="2018-05-05T16:01:20.073Z">
<saml:AudienceRestriction>
<saml:Audience>https://localhost:44344</saml:Audience>
</saml:AudienceRestriction>
</saml:Conditions>
<saml:AuthnStatement AuthnInstant="2018-05-05T15:56:20.073Z">
<saml:AuthnContext>
<saml:AuthnContextClassRef>
urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified
</saml:AuthnContextClassRef>
</saml:AuthnContext>
</saml:AuthnStatement>
<saml:AttributeStatement>
<saml:Attribute Name="userId"
NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
<saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:anyType">0051r0000089ZM2
</saml:AttributeValue>
```

```
</saml:Attribute>
<saml:Attribute Name="username"
NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
<saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 ↪  xsi:type="xs:anyType">jkmu@xxxxxxxx.com</saml:AttributeValue>
</saml:Attribute>
<saml:Attribute Name="email"
NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
<saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 ↪  xsi:type="xs:anyType">jkmu@xxxxxxxx.com</saml:AttributeValue>
</saml:Attribute>
<saml:Attribute Name="is_portal_user"
NameFormat=
"urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
<saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 ↪  xsi:type="xs:anyType">
false
</saml:AttributeValue>
</saml:Attribute>
</saml:AttributeStatement>
  </saml:Assertion>
</samlp:Response>
```

V Class diagram

pkg Saml2.Authentication.Core
Saml2Constants
Saml2Exception
HttpRedirectBindingConstants
HttpRedirectBindingParser
Saml2Assertion
ServiceProviderConfiguration
Saml2LogoutResponse
HttpSoapBindingParser
SigningCertificate
Saml2Defaults
EncryptionProperty
ConditionAbstract
ProxyRestriction
DependencyInjection
DSAKeyValue
Transform
AssertionIDRequest
LogoutRequest
RequestedAttribute
ISamlService
Authentication
HttpRequestExtensions
Saml2ExtensionsContactType
Microsoft
dk
Saml2
Providers
Options
Factories
Extensions
Saml2ClaimTypes
SignatureProviders
ISignatureProviderFactory
ISignatureProvider
ClaimsPrincipalExtensions
Saml2StringExtensions
ItemsChoiceType
ShaHashingAlgorithm
Saml2Configuration
Saml2FormatException
Saml2LogoutResponse
Saml2AssertionValidator
Saml2KeyInfoValidator
Saml2ArtifactResolve
IdentityProviderConfiguration
Saml2Options
SimpleBodyWriter
Schema
Utils
Validation
Bindings
Saml2StatementValidator
Saml2NameIDValidator
AudienceRestriction
AuthnContext
AuthnStatement
AuthzDecisionStatement
Subject
SubjectLocality
Evidence
KeyInfoConfirmationData
ObjectType
CipherReference
CipherData
PgPData
NameID
OneTimeUse
AgreementMethod
Reference
SPKIData
SignedInfo
SignatureValue
Terminate
ArtifactResolve
ArtifactResponse
RequestedAuthnContext
Response
Scoping
KeyDescriptor
IDPEntry
IDPList
SignatureProviderFactory
SignatureProviders
Services
KeyTypes
Saml2ServiceCollectionExtensions
HttpResponseExtensions
Saml2StringBuilderExtensions
ISaml2SubjectConfirmationDataValidator
AuthnContextComparisonType
ISaml2AssertionValidator
ISaml2Validator
IHttpArtifactBinding
IHttpRedirectBinding
ICertificateProvider
DecisionType
Saml2SubjectConfirmationValidator
Saml2SubjectValidator
Saml2NameIDValidator
Saml2XmlAnyAttributeValidator
Saml2Utils
Saml2ClaimFactory
HttpArtifactBinding
SoapConstants
Saml2MessageFactory
Saml2Validator
Saml2Handler
ReferenceList
SignedXMLWithIdResolvement
HttpArtifactBindingConstants
Saml2AuthnRequest
StringSerializer
Saml2Response
CertificateProvider
BaseIDAbstract
Authentication
Saml2XmlAnyAttributeValidator
Saml2Utils
Extensions
SamlService
HttpArtifactBinding
HttpRedirectBinding
Saml2SubjectConfirmationValidator
Saml2NameIDValidator
Serialization
TimeRestrictionValidation
Saml2PostConfigureOptions
Saml2SubjectConfirmationDataValidator
Saml2InResponseToValidator
Saml2LogoutRequest
SAMLAction
RequestIdCookieBuilder
Saml2SubjectConfirmationValidator
AssertionProfile
Encrypted
SubjectQueryAbstract
RequestAbstract
Saml2AttributeValidator
Saml2EncryptedElementValidator
XmlSignatureUtils
ArtifactUtil
saml20
nita
Saml2EncryptedAssertion
Saml2StatementValidator
Saml2SubjectConfirmationDataValidator
Saml2ArtifactResolve
SAMLAction
Saml2SubjectConfirmationValidator
HttpSoapBinding
HttpRedirectBinding
HttpArtifactBindingParser
Saml2SubjectConfirmationValidator
Saml2SubjectValidator
Saml2NameIDValidator
AssertionProfile
Saml2AttributeValidator
Assertion
AttributeStatement
EncryptedData
EncryptedKey
EncryptionMethod
SubjectConfirmation
SubjectConfirmationData
StatementAbstract
Core
CanonicalizationMethod
DigestMethod
KeyValue
Manifest
EncryptedElement
EncryptedAssertion
AuthzDecisionQuery
AuthnRequest
AuthnQuery
Signature
SignatureMethod
X509IssuerSerial
SignatureProperties
SignatureProperty
RetrievalMethod
StatusResponse
NameIDPolicy
XEnc
XmlDSig
Protocol
Metadata
Core
KeyInfo
X509Data
Transforms
AttributeQuery
RsaSha256SignatureProvider
RsaSha512SignatureProvider
DsaSha1SignatureProvider
RsaSha1SignatureProvider
IHttpRedirectBinding
IHttpArtifactBinding
ISamlProvider
ISaml2Validator
ISaml2AssertionValidator
ISaml2StringExtensions
Extensions
Factories
Options
Providers
ISaml2ClaimFactory
ISaml2MessageFactory
ManageNameIDRequest
NameIDMappingRequest
NameIDMappingResponse
LogoutResponse
StatusCode
StatusDetail
Status
Action
Advice
Conditions
EncryptionProperties
Saml2InResponseToValidator
Saml2SubjectValidator
Saml2SubjectConfirmationValidator
Saml2NameIDValidator
Saml2SubjectConfirmationDataValidator
Saml2ServiceCollectionExtensions
HttpResponseExtensions
Saml2StringBuilderExtensions
ISaml2SubjectConfirmationDataValidator
AuthnContextComparisonType
KeyTypes
Services
Validation
ICertificateProvider
SignatureProviderFactory
SignatureProviders
Generated by UMLtoXaml
www.altova.com