Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Eirik Hemstad
Torstein Holmberget

SSO Solution for mobile apps from DIPS

Bachelor's project in Computer Engineering
Supervisor: Ali Alsam
May 2021

NTNU
Kunnskap for en bedre verden

Eirik Hemstad
Torstein Holmberget

SSO Solution for mobile apps from DIPS

Bachelor's project in Computer Engineering
Supervisor: Ali Alsam
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Preface

This project belongs to the Department of Computer Science (IDI) under the Faculty of Information Technology and Electrical Engineering (IE) at the Norwegian University of Science and Technology (NTNU). Research was conducted for DIPS ASA, a leading eHealth provider in Norway.

DIPS had a problem they needed researched, and since one of our team members was a part-time employee there we got the task as our bachelor assignment. This problem revolved around finding a way to implement Apple's new enterprise single sign-on solution into their mobile development for hospital applications. The task presented many challenges both due to the lack of documentation since the solution was so new, and our lack of experience with the technologies involved. In the end the task gave us the opportunity to use many of the skills we had acquired through our education, including system development, testing, security and research. And it gave us the opportunity to learn many new skills within mobile development and iOS development specifically, as well as a greater understanding of the complexities of research.

Acknowledgements

We would like to thank DIPS ASA for giving us the opportunity to work with this assignment. Big thanks goes out to Tore Mørkved at DIPS for providing the assignment and giving us insight into the relevance and importance of our work. We would also like to thank several employees at DIPS: Erling Moxnes Kristiansen for aiding us in all things regarding mobile development and practices at DIPS, and Christer Brinchmann for helping us with server configurations and issues. We would also like to express our warm gratitude towards our supervisor Ali Alsam for providing great feedback and support, along with discussions regarding the scientific aspects of our report. Lastly, a big thanks to Kjell Arild Sandvik at Knowit for letting us debrief him about his experiences with the problem domain.

Trondheim, 20.05.2021

Eirik Hemstad Torstein Holmberget

Assignment Description

Original Assignment

DIPS ASA wants an SSO solution for our native mobile applications for clinics. Extensible Enterprise SSO that was part of iOS 13 makes it possible to implement this for our apps so that users only need to log on once, even if they are using several apps from different providers.

Priority 1:

- Research how this can be implemented and configured using iOS Enterprise Extensions for iOS apps alongside MDM.
- Research how to support 2-factor authentication with username and password, and a key card using NFC technology.
- Create a prototype

Priority 2 (bonus):

- Research similar solutions for native apps on Android

Updated Assignment

After clarification from product owner, it was decided that 2-factor authentication with a key card using NFC technology would be moved to priority 3 because of the scope of the assignment. Neither priority 2 or 3 was required for the completion of the assignment. The new assignment listing became:

Priority 1:

- Research how this can be implemented and configured using iOS Enterprise Extensions for iOS apps alongside MDM.
- Create a prototype

Priority 2 (bonus):

- Research similar solutions for native apps on Android

Priority 3 (bonus):

- Research how to support 2-factor authentication with username and password, and a key card using NFC technology.

Summary

In a time where smartphones provide new ways to digitalize work tasks and administration, a lot of users face the need to use more and more mobile applications for their work. As enterprise applications will require the user to be authenticated and authorized before use, users can risk spending an unreasonable amount of time logging into every application at the start of the workday. The introduction of Single Sign-On in both web browsers and smartphones aim to reduce the impact authentication can have on a user's workflow.

DIPS ASA wanted us to figure out a way for them to implement Single Sign-On into their applications using Apple's new Extensible Enterprise Single Sign-On solution.

The goal was to research this new solution and try to build a working prototype for DIPS to use in future implementations of this technology into their applications for the hospital sector specifically. We also wanted to research the viability of existing SSO solutions based upon this technology.

We developed several prototypes, two of which used existing solutions made with Extensible Enterprise SSO, and one attempt at creating a custom SSO extension. The prototypes taught us how the technology could be utilized, and gave us a lot of experience with dealing with related technology and its issues. Completing a functioning prototype for a custom SSO extension was unfortunately hindered by technical issues and insufficient documentation.

Our research shows us that Extensible Enterprise Single Sign-On is designed to abstract the complexity of handling authentication and authorization flows away from individual applications, into central platform-based repositories that can serve all incoming and outgoing identity traffic. Such SSO Extensions are available to be developed by any provider, but should not be used only as proprietary solutions for the companies that develop them.

Report structure

Chapter 1 contains the background for the assignment and our understanding of the assignment, as well as the research questions we derived from the task.
Chapter 2 contains all the theory behind our work as well as some background on the various technologies we would be working with.
Chapter 3 is a runthrough of all the technologies we used, our research and development methodologies and the tools we used to accomplish our assignment.
Chapter 4 contains the results of our research, the implementation of the extension and our process and administrative results.
Chapter 5 will be a discussion of those results and of our progress through the project.
Chapter 6 is the final chapter where we draw the conclusions of our research and what can be done of further work on this.

Appendices

Appendix A contains the full Assignment as given to us at the start of the project, without the changes agreed to during the project as described in chapter 1.

Appendix B contains the Vision Document we set up at the start of the project as a guide to how and what we were trying to achieve.

Appendix C contains the bug report to Apple regarding the problems we ran into during development.

Appendix D is the raw minutes of our interview with Knowit.

Contents

Introduction and relevance

1.1 Background

Usage of smartphones in everyday life is increasing quickly. These handy devices are becoming more and more integrated in modern society, both as a means of entertainment and a necessary tool for performing everyday tasks. Every day, we use a plethora of unique applications for communicating, buying bus tickets and handling bank transactions. This usage of mobile applications have also extended into many work environments, becoming an important part of the working day for many different professions. Most of these apps will require access to some data over the internet, which often can be user-specific and sensitive. Therefore, users will have to authenticate with their personal credentials before they can use the application, proving who they are to the service provider.

But authenticating through every new app you open can be cumbersome, and managing many different user accounts can quickly lead to unsafe or repeatedly used passwords. To try and solve this, many service providers let you authenticate through pre-existing accounts, like Google, Apple ID or Facebook. Android devices and iPhones can even store authenticated user sessions in the operating system, using Google accounts or Apple ID's respectively. When you open a new application on your iPhone, there's a chance that the application lets you log in using your Apple ID. If you choose to do so, you won't need to enter your credentials, as your account is already authenticated through the operating system. The application will simply get the user session from the operating system, and use it when it needs to access guarded resources. This method of storing authenticated user sessions where applications can access them is known as Single Sign-On (SSO).

An increase in use of smartphones in hospitals and clinics by medical personnel to perform various work tasks has lead to a situation where they often need to use several different applications from different providers. Logging in on each of these apps every day can be obstructive to their workflow. One login at the start of the work shift to access all their required tools would be preferable, and would improve the quality of the user-experience and contribute to a more efficient work process.

1.2 The Problem Domain

DIPS ASA is one of Norway's leading providers of electronic patient journals. Their main product is currently DIPS Arena, a desktop application designed to be used by health personnel in hospitals. DIPS Arena provide functionality for keeping track of treatment plans, hospital-admitted patients, appointments and other related features. DIPS has also recently launched a pilot project involving two smartphone applications that can be used on hospital premises. These two applications are DIPS Visit and DIPS Tasks. DIPS Visit lets personnel quickly access a patient's journal while making rounds at the hospital, with support for adding further notes and documentation. DIPS Tasks acts as a checklist where personnel can see work-related tasks which needs to be completed, with support for viewing documents and transferring tasks to other personnel. Both of these apps are designed to be used in a hectic environment, so usability and efficiency is of high importance. It is expected that users will frequently jump between these two apps, among other third-party apps that might be included in their workflow. Authentication prompts popping up while switching between two apps can be disruptive to the user's workflow and cause a lot of annoyance. But proper authentication for each app is still very important. In order to provide the desired functionality to the user, the apps need to access and present sensitive data from patient journals. The security requirements for storage and access of such data are strict and regulated by law, meaning that the authentication process cannot be compromised or simplified for ease of use. This provides a challenge, as users must be authenticated and authorized properly while still providing a seamless and productive experience when jumping between applications.

When DIPS ASA started developing for the mobile platform, they wanted to know whether they should focus on creating one large app, or rather focus on multiple smaller apps that provide specific sets of functionality. In 2019, university students from NTNU wrote a bachelor thesis for DIPS ASA, involving researching the practicality and efficiency of dividing DIPS mobile services into smaller apps, and how this would compare to one, monolithic app CITE BACHELOR. The thesis led to the students developing a Single Sign-On solution that could be used to streamline the authentication between the apps developed by DIPS. The finished product was an authentication library that could be integrated into every app they develop. This solution was programmed specifically to interact with DIPS identity servers. This solution was functional for a while, but newer versions of iOS rendered the Single Sign-On aspect of the library obsolete. The library used the local storage of the native Safari browser to share session data between apps. This method is no longer functional due to a change in iOS 13.

1.3 The Assignment

The assignment that created the basis of this research report was provided to us by DIPS ASA. The assignment was to research the possibilities of implementing Apple's new Extensible Enterprise Single Sign-On (EESSO) framework into their products as a way of handling user authentication cross-application on iOS. EESSO is a system integrated in iOS, that lets developers create App extensions that allow for SSO capabilities in any application that implement them. We expand more upon EESSO in chapter 3.

DIPS provided multiple reasons for integrating EESSO into their mobile platform:

- EESSO allows for Single Sign-On capabilities between any application provider that implements the same SSO extension
- DIPS' current SSO solution is deprecated for newer versions of iOS
- EESSO is the proper standard for SSO on iOS

In addition to this research, we were to develop a prototype that demonstrated how an SSO extension could be implemented towards DIPS' identity providers.

The original assignment text can be found in Appendix A - Assignment Text

1.4 Research Questions

Initial discussion around the assignment inspired various related questions that we wanted to take a closer look at.

1.4.1 Are there advantages to developing a proprietary SSO extension, or is it better to use an existing maintained extension?

When aiming to implement Single Sign-On to applications in a specific system, the provider has to make a choice on how to create their SSO extension to work with apps from other providers in the same usage area. This means that they either have to use an existing extension from a third-party provider that is set as the standard for their user area, or they can create their own proprietary extension that leaves open the possibility of other services connecting into their system to get a true universal Single Sign-On for the end-user. But which of these solutions are the best ones both for security and usability?

There are already two available SSO extensions that have been developed with EESSO. If multiple, independent application vendors use the same SSO extension for their applications, they could

achieve cross-vender Single Sign-On. These are meant for specific identity providers, and are closed-source. This means that there is no way for a company to extend the functionality of such an extension, without developing an entirely new extension. If a vendor does not use one of the officially supported identity providers, they may have to develop their own extension. Are the pre-existing extensions flexible enough to be used by an application vendor (granted that the right identity provider is used), or should they rather develop their own extension targeting their identity provider of choice?

1.4.2 How can Apple Extensions be implemented and used with Xamarin?

Applications for any of Apple's mainstream platforms are programmed in Swift. This means that Apple has intended their extensions to be implemented through this technology. DIPS uses the cross-platform framework Xamarin, to efficiently build mobile apps for both Android and iOS. Many developers that develop in Xamarin find themselves in the situation where they need to implement these solutions into a cross-platform system. This means that we had we had to find out how Apple Extensions can be utilized through Xamarin.

1.4.3 How can Single Sign-Out be handled on non-personal mobile devices?

When employees share work devices, it is important that the previous user is logged out before a new user gains access to the device. When using digital tools, it is important that each user is logged in as themselves, as this maintains integrity across the system. This can be hard to enforce, as it's not guaranteed that a user will check that they're properly logged in every time. If one user's authenticated session slides over to the next user, it can cause the wrong user to gain or lose access to sensitive data. Since a Single Sign-On system holds shared authenticated sessions, signing a user out on one app will result in a sign-out across the entire device for that user. This is known as Single Sign-Out. Is there any way to log out users automatically on shared work devices?

Glossary

app extension A form of application add-on that can extend the functionality of an application and make it available while the user is in another app. 3, 6, 32

AppAuth A program library for Swift that lets developers easy set up authentication through OAuth 2.0 or OpenId Connect. 40

AppSSODaemon A system daemon responsible for handling EESSO operations. 33, 41

bundle identifier The unique identifier of an application on the Apple platform. Often in the form of *com.company.appname*. 14, 28, 29, 32

daemon A background process running on a computer system. 33

identity provider A system entity that creates, maintains, and manages identity information for principals and also provides authentication services to relying applications within a federation or distributed network. 23

Intune A cloud-based service that focuses on mobile device management (MDM) and mobile application management (MAM). 17, 25, 26, 28, 30, 32, 33, 42

MDM Payload A set of configuration data that is assigned to devices enrolled in Mobile Device Management.. 13, 41, 42

Microsoft Authenticator An application that provides functionality for keeping track of accounts on passwords. 42

Mobile Device Management A deployment of managed devices and/or applications, often used to enforce corporate policies and configurations. 12, 13, 20, 25, 32

OAuth 2.0 . 17, 30, 42

OpenID Connect An identity layer that sits on top of OAuth 2.0. It is one of the most used protocols in modern authentication. 12, 17, 23, 30, 34, 42, 43, 47

PingFederate A global authentication authority server that allows employees, customers and partners to securely access all the applications they need from any device. 23

protocol In Swift, a protocol is assignable to a class to force it to implement a set of methods to fulfill the functionality expected by some external functionality. 26

realm A subset of a domain that is managed by one or more Kerberos Domain Controllers. 14

REST Representational state transfer (REST) is a software architectural style which uses a subset of HTTP. It is commonly used to create interactive applications that use Web services.. 10

SSO extension An app extension that utilises EESSO to provide SSO functionality to an application. vi, 3, 12–14, 20, 23–27, 30–35, 40–42, 44–50

swcutil A network utility for validating Associated Domains on Apple platforms.. 16, 42

Swift Programming language for creating native applications for the Apple platform. 25, 26, 29, 33, 34, 40, 49

target A sub-project contained within an XCode project, often following a specific template. 32

team identifier The unique identifier of a development team registered as licensed Apple application developers. 14

Universal Links A system on all Apple platforms that lets developers link together applications and websites under their domains. 41, 42, 48

view controller A class that handles logic and input operations for a view or UI element. 26, 29, 45

Xamarin A cross-platform mobile framework developed by Microsoft. vii, 4, 17, 25–27, 29, 40, 42–45, 49

Acronyms

AASA Apple App Site Association. 16, 41, 42

CDN Content Delivery Network. 41, 42

EESSO Extensible Enterprise Single Sign-On. 3, 5, 6, 12, 13, 15, 22, 23, 25, 26, 34, 35, 40, 41, 43, 44, 47–50

IPA iOS App Store Package. 33

MSAL Microsoft Authentication Library. 27, 42

SSL Secure Sockets Layer. 14, 41

SSO Single Sign-On. 1, 3, 6, 24, 25, 30, 41, 44–46

Chapter 2: Theory

This chapter introduces the reader to concepts relevant to our research questions in Chapter 1, as well as the theoretical structures which our research builds upon.

2.1 Authentication

The users have to authenticate themselves with secure credentials only known to the owner of the device, or in the case of shared devices, only the permitted users of the device. This means verifying to the system that they are who they claim to be, either with a biometric system for single-user devices, or a two-factor system for shared devices, this prevents unauthorized access to the system and a means of identifying the user.

2.1.1 Single Sign-On

Single sign-on is an authentication scheme that allows users to log in with a single account to several related but independent systems through an external system that stores credentials upon initial authentication. These credentials can be served to other systems and re-used as long as they are valid.

2.1.2 eIDAS

eIDAS (electronic Identification Authentication and trust Services) is a European regulation. It is designed to ensure a functioning electronic market, along with a proper level of security for electronic identification and trust services. The regulations require all providers of trust services to provide technical and organizational actions for handling potential security risks. In relevance to this report, eIDAS has certain requirements for the login process when accessing resources from government organizations. For example, eIDAS require that an authorized login key must at least contain two factors of authentication. [1]

2.1.3 Federated Security

In information technology a federated identity is a link between a person electronic identity and attributes, stored across multiple distinct identity management systems.
Federated identity in relation to Single Sign-On is where a users authentication ticket or token is trusted across multiple IT systems or organizations.
Federated identity management describes the technologies, standards and use-cases which serve to enable to portability of identity information across otherwise autonomous security domains, with the goal of enabling users of one domain to securely access data or systems from another domain seamlessly without the need for redundant user administration.[2]

2.1.4 Biometric Security

Biometric security are systems that rely on unique biometric identifiers like facial recognition, fingerprints, retina scans, etc. to authenticate a person. The usage of biometrics is very application dependant. Certain biometrics are better then others depending on required levels of convenience and security, as well as the constraints of the hardware it is running on.

2.1.5 Two Factor Authentication

Two factor authentication (2FA) requires a user to perform two steps of authentication before they are logged in. This is in theory more secure than one single step, as it challenges the user with providing more information before they are authenticated. Two factor requires the user to provide two out of three types of credentials before they can be logged in. The three credential types are:

- Something you know, like a PIN or combination of username and password.
- Something you have, like a personal device or card.
- Something you are, like biometric fingerprints or facial recognition.

These unique factors makes it harder for attackers to compromise an identity, as they would at least have to gain access to a physical object belonging to the user to be able to authenticate. [3]

2.1.6 Token or Ticket Authentication

Authentication of the token or ticket being passed from the Federation server has to be done through a external certified authentication provider like for instance BuyPass to make sure that the token or ticket provided is correct and authentic.

2.2 Authorization

Authorization is the process of granting a user or automated process a level of access or privilege in a computing system. This means controlling who can access what, and to what degree they are able to read or manipulate the accessed data. Access control in computer systems rely on access policies. The access control process can be divided into the following phases: policy definition phase where access is authorized, and policy enforcement phase where access requests are approved or disapproved.

2.2.1 OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization. OAuth provides clients a secure delegated access to server resources on behalf of a resource owner [4].

2.2.2 OpenID Connect

OpenID Connect is a simple identity layer on top of the OAuth 2.0 protocol. It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an inter-operable and REST-like manner [5].

2.2.3 ADFS - Active Directory Federation Services

ADFS is a Microsoft developed system to provide SSO capabilities to Windows Servers and is part of Active Directory Services. It uses a claims-based access-control authorization model to maintain application security and to implement federated identity [6].

2.2.4 MDM(Mobile Device management)

MDM is typically a deployment of a combination of on-device applications and configurations, corporate policies and certificates, and backend infrastructure. Some of the core functions of MDM include

- Ensuring that diverse user equipment is configured to a consistent standard
- Ensuring that sensitive equipment complies with required policies, thus improving security and damage control
- Updating equipment, applications, functions, or policies in a scalable manner

- Ensuring that users use applications in a consistent and supportable manner
- Monitoring and tracking equipment
- Being able to efficiently diagnose, troubleshoot and manage equipment remotely

Typical solutions include a server component which sends out the management commands to the mobile devices, and a client component which runs on the managed device and receives and implements the management commands.

Over-the-air programming is considered the main component of operator and enterprise-grade MDM. This includes the ability to remotely configure a single mobile device, all the devices affected by the server, or only certain devices decided by IT.

Chapter 3: Technology and Methodology

3.1 Technology

This section will be a summary of the technology we used for the prototype solution to the project, as well as the subjects of the research of EESSO.

3.1.1 Extensible Enterprise Single Sign-On

Apple's Extensible Enterprise Single Sign-On is a built-in feature introduced in iOS 13, iPadOS 13 and MacOS Catalina. This feature lets app or web developers use the operating system as a means of persisting authenticated user sessions between both native apps and websites through Safari. This functionality is meant to be implemented through app extensions, where the developers themselves have to program the desired authentication flow [7].

3.1.1.1 SSO Extensions

EESSO is utilized through specific SSO extensions. An SSO extension is a form of program library for Apple platform that implements a set of abstract methods that are called by the authentication framework used by the Swift platform. This means that the extension developers have to program exactly how the authentication procedure should work. For example, developers can choose to implement client code for an OpenID Connect setup in the extension. When the host app makes a request for authentication through one of the built-in frameworks, EESSO will call the SSO extension, which again will process the request further to the implemented endpoints. The extension must be deployed through Mobile Device Management. As the extension functions as a single application per device, authorization data can be stored and accessed by all permitted applications. Apple provides two types of SSO extensions [7].

3.1.1.2 Redirect Extension

A Redirect Extension is a type of SSO extension. It is based on redirecting requests from normal applications. The application can either use an authentication framework to start a web request, or trigger a special authentication operation. The extension will intercept the request or operation. The extension receives the request with the URL, HTTP headers and body, and the extension developer is responsible for using this data to complete the authentication process with the identity provider. When the authentication process is complete, the extension can append the resulting authorization headers to the original request, which will proceed to the identity provider again. The identity provider will not need to challenge the request again, as it's already authorized through the extension [8]. The extension can choose to store already acquired authorization headers inside its own state, thus enabling Single Sign-On between separate applications [7].

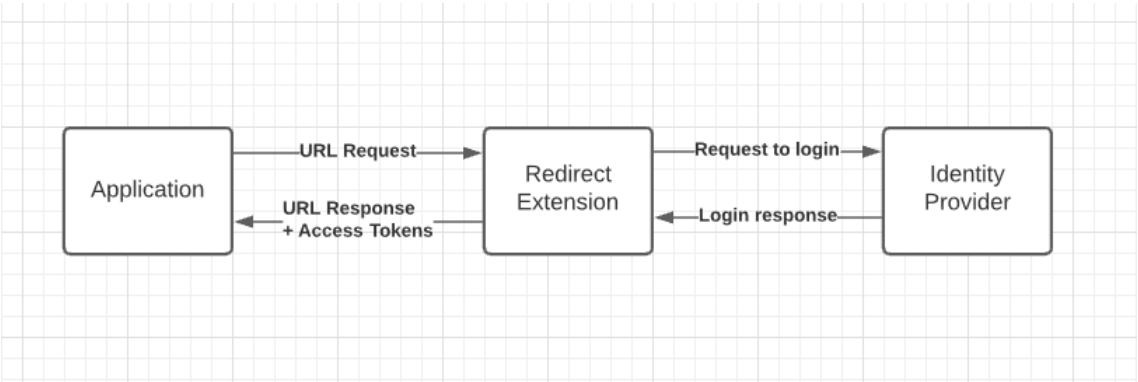


Figure 3.1: The flow of a Redirect Extension

3.1.1.3 Credential Extension

A Credential Extension is the second type of SSO extension. It is based on a web server returning a HTTP challenge when an application attempts to contact the web server. The challenge is directed to the extension, which will complete the authentication request with the authentication server. When the request is completed, the authorization headers will be sent to the web server, which again will use this data to send a response to the application (figure 3.2) [7].

3.1.1.4 Prerequisites

As EESSO is a feature of the operating system, only applications running on iOS 13, iPadOS 13 or MacOS Catalina can utilize this behaviour. In addition, applications have to be deployed through an Mobile Device Management solution, as EESSO has to be enabled and configured through a specific MDM Payload. This payload is configured to be deployed to devices that are enrolled in the Mobile Device Management environment, and has to include the following data[9]:

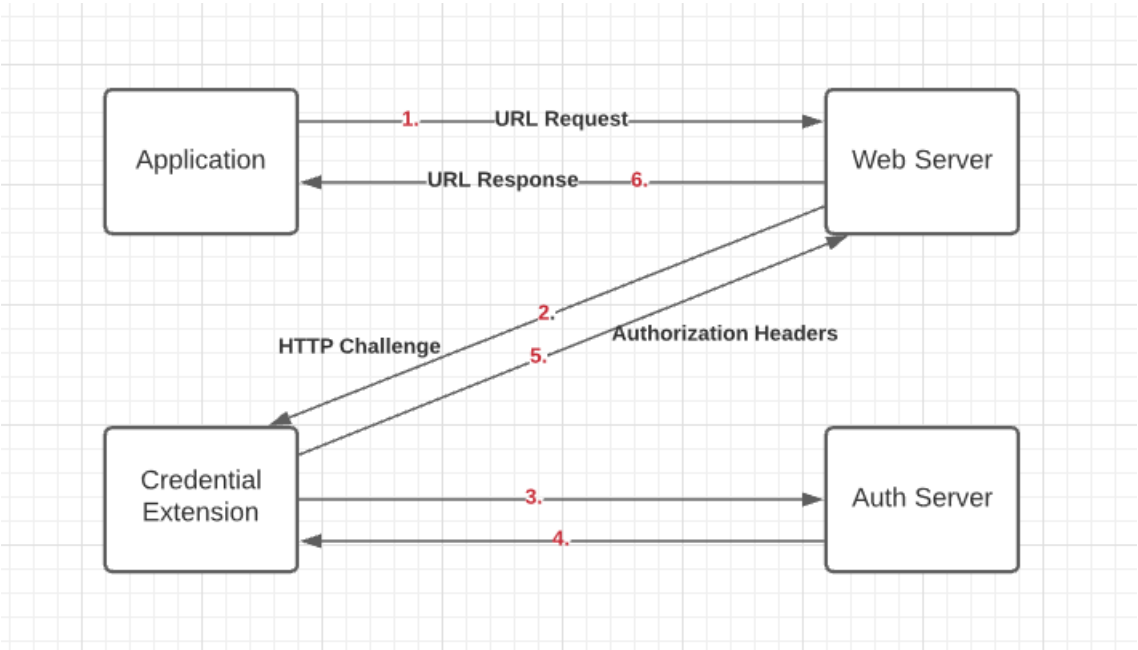


Figure 3.2: The flow of a Credential Extension

- The type of extension (Redirect or Credential)
- The bundle identifier of the SSO extension (including the team identifier of the SSO extension developer)
- The realm of the Kerberos environment where the user resides (Credential only)
- A list of domains that are approved for the extension (Credential only)
- A list of URL prefixes that should be intercepted by the extension (Redirect only)

When deploying a Redirect Extension, the extension also requires that the application lists the target identity server through Associated Domains. This is a list of domain names that are associated with the app. This can be configured in the app directly, or deployed through MDM App Configuration. The identity server also has to serve an Apple App Site Association file, which contains a list of all the applications that are authorized to use the server for authorization. The server must have a valid SSL certificate [7].

3.1.2 Kerberos

Kerberos is the default system used by Apple's SSO solution. It is an authentication protocol that allows nodes to communicate over a non-secure network to prove their identity to one another in a secure matter using tickets.
Kerberos builds on symmetric key cryptography and requires a trusted third party for authentication and is the default protocol used by Azure AD and Active Directory. It works across platforms,

uses encryption, and has protections against replay attacks [10].

3.1.3 Microsoft Enterprise SSO plug-in

Microsoft Enterprise SSO plug-in is Microsofts solution for enabling SSO to Azure AD accounts on all applications that support Apple's Enterprise Single Sign-on on iOS, macOS and iPadOS devices. It is usable by all MDM solutions and extends SSO to applications that use OAuth 2, OpenID Connect and SAML. There are some requirements for it to operate successfully, the device needs to support and have installed an app that has the MSESSO plug-in such as Microsoft Authenticator on iOS or iPadOS devices and Intune Company Portal on macOS, The device needs to be enrolled in an MDM system, and the configuration must be pushed to the device through the MDM system.

3.1.4 Universal Links

Universal Links is a feature of the Apple Ecosystem that lets developers and system architects link together apps and websites. This means that apps and websites can be associated to trust each other and communicate certain data. Servers can be declared as to provide certain services to other applications, and applications can be told to use these services [11].

3.1.4.1 Associated Domains

Associated Domains lets a developer register an associated service with an app. For example, a developer can specify that the domain name *auth.example.com* is to be recognized as the application's authorization server. Associated Domains must be listed in the application's entitlement file, with a special prefix for each domain name that specifies what type of service the domain serves. EESSO requires that the authorization server is prefixed with the *authsrv* service type, which is short for *Authentication Services* [11].

3.1.4.2 Apple App Site Association File

The Apple App Site Association File is a .json file that should be located on every server that interacts with Apple applications. It is required for all servers that are listed as one or more application's associated domain. When a network framework in the application tries to contact a server, the application's device retrieves this file, and ensures that it is listed and approved for access. This file must be available at the URL */.well-known/apple-app-site-association*, relative to the server's root URL [11].

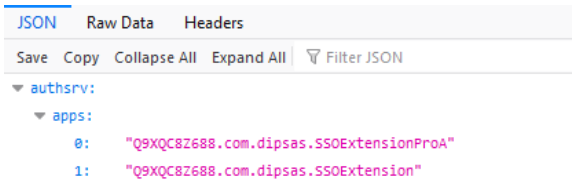


Figure 3.3: An example of the Apple App Site Association file viewed in a web browser

3.1.4.3 swcutil

swcutil is a utility available on most devices running Apple operating systems. This is a utility that runs validations on Universal Links. It can be used to retrieve and validate the Apple App Site Association (AASA) file from an Associated Domain. Although swcutil is a tool made by Apple, there is no official documentation describing it, outside the instructions provided by the command line utility [8].

```
swcutil [--leaks] [--verbose] <command> [options] [command [options] ...]
  --leaks
    Run leaks before termination.

  --vmmap
    Run vmmap before termination.

  --heap
    Run heap before termination.

  --verbose
    Increase verbosity of output.

  dl -d <domain> [-t <timeout>]
    Downloads an A-A-S-A file from a domain.
    If specified, the timeout is in seconds.

  get [-s <service>] [-a <app-id>] [-d <domain>]
    Gets info about apps and/or domains.

  openul -u <url> [-r <referrer-url>]
    Open a URL as a universal link.

  show
    Show the current state.

  verify [-d <domain>] -j <JSON-path> [-u <url>]
    Verify apple-app-site-association files. Use -u to match a URL.

  reset
    Reset the database and restart swcd.

  watch
    Watch the system log for SWC logging.
    Specify --verbose to enable debug-level logging.

  match -u <url> -j <json>
    Test a pattern-matching dictionary against a URL.

  developer-mode -e <enabled>
    Enable or disable developer mode.
```

Figure 3.4: Instructions on how to use swcutil

3.1.5 XCode

XCode is Apple's IDE for macOS and the default software for creation apps for macOS, iOS etc. It is primarily used to program in the Swift language, which is Apple's own developed programming

language. Since XCode is created specifically for the Swift Language and macOS it is an easy and safe way to implement the various features of the language.

3.1.6 Swift

Swift is a compiled programming language developed by Apple and the open-source community developed as a replacement to Object-C. On Apple platforms it uses the Objective-C runtime library which allows C, Objective-C, C++ and Swift code to run within one program. Swift supports the core concepts associated with Objective-C like dynamic dispatch and extensible programming but in a "safer" way that makes it easier to catch software bugs. It also contains features to addresss common programming errors like null pointer dereferencing. It also supports the concept of protocol extensibility, a system that can be applied to types, structs and classes which creates a change to protocol-oriented programming over object-oriented programming.

3.1.6.1 AppAuth

AppAuth is a Software Development Kit for Android, iOS, MacOS and tvOS for communicating with OAuth 2.0 and OpenID Connect providers.

3.1.7 Xamarin

Xamarin is an open source framework developed by Microsoft to build cross-platform applications for iOS, Android and Windows with .NET from a single shared C# codebase.

3.1.8 Intune (Microsoft)

Microsoft Intune is a cloud-based service that focuses on MDM (Mobile Device Management) and MAM (Mobile Application Management). Intune integrates with Azure Active Directory, and lets the administrator manage all mobile devices that have joined the domain under Azure AD.

3.2 Research Methodology

At the start of the project, we were not particularly familiar with the hospital domain in which the solution would be used. This knowledge would be vital in defining which problems had to be taken into account when developing our prototype. There were a multiple subsections of the domain that had to be considered. These were as followed:

- How does the hospital handle their mobile device logistics?

- Would mobile devices switch users often?
- If above is true, how is switching of user sessions handled?

The answers to these questions will naturally vary between different hospitals. Still, we wanted to base most of our understanding on a particular environment that could serve as an example.

3.2.1 Qualitative Research

In qualitative research, the goal is to analyze detailed, non-numerical data to understand a concept, experience or opinion. Such data is gathered through reading and understanding scientific articles or conducting a detailed interview with a party that has relevant opinions, knowledge or experience about the data you wish to gather.

Qualitative research involves collecting and analyzing non-numerical data (e.g., text, video, or audio) to understand concepts, opinions, or experiences. It can be used to gather in-depth insights into a problem or generate new ideas for research. Qualitative research is the opposite of quantitative research, which involves collecting and analyzing numerical data for statistical analysis. Qualitative research is commonly used in the humanities and social sciences, in subjects such as anthropology, sociology, education, health sciences, history, etc.[12]

3.2.1.1 Goal Determination

The first step in a qualitative interview is to define the goal of the interview. What knowledge do we wish to gain from the interview, and what overarching questions do we need to be answered?

3.2.1.2 Target interview object

The next step is to determine which person should be interviewed. The individual's role, experience and knowledge should be considered when choosing the right candidates.

3.2.1.3 Plan and design questionnaire

How should the interview questions be structured and defined? Should the questions follow a strict and specific structure, or should the questions be open and allow the individual to speak more freely? The questions should be formed in such a way that they cover the entirety of what the interviewer seeks to find out. They should also be clear and easy to understand by the subject.

3.2.1.4 Perform the interview

Perform the interview as planned with the prepared questionnaire and the relevant parties. Make sure to take notes or record the interview. The subject must be informed of what is recorded.

3.2.1.5 Process the data

Go through your notes or the recording, and organize what questions were answered, and what was left unanswered.

3.3 Development Method

Our team chose at the start of the project to use a thinned out version of Scrum, with sprints, sprint-reviews and daily scrum meetings. We chose this because scrum works well in a development process with a lot of changes underway in the project. We decided not to go with any of the scrum roles due to the small size of the team, and rather have a flat hierarchy.

3.3.1 Sprints

In scrum, sprints are iterations with a specific time period decided in advance, usually between 1 and 4 weeks. Sprints start with sprint planning that establishes the goal for the particular sprint and what backlog items are to be included, then ends with a sprint review.

3.3.1.1 Sprint planning

First day of a sprint the team had a sprint planning event where they discuss the sprint goal and selects the product backlog items that are to be completed in this sprint, and what priority each task has.

3.3.1.2 Sprint review

When the sprint is done the sprint review is a walk-through of the results this sprint where the team demonstrates current progress to the customer and decide if the results are accepted or not in regards to where to go next in the process.

3.3.2 Pair Programming

Since we were only two people working at separate locations it was an ideal environment for us to use pair-programming, where one person actively codes while the other watches and makes comments and suggestions along the way to get both points of view into the same code. Only Eirik was able to actually program and test the prototypes, as he was the one with access to DIPS' internal systems and the Mobile Device Management environment. Since we for the most part worked at separate locations, pair programming was a necessity when development and testing of the SSO extension started.

3.4 Tools

To achieve a good workflow when working digitally due to the global pandemic we used various tools to optimize workflow by improving communication, sharing of documents, maintaining a workboard and keep track of time spent on different tasks.

3.4.1 DIPS' Internal Tools

Because of strict guidelines regarding access to intellectual property, we had limited access to DIPS' internal system. Since Eirik already had a work account through a part-time employment at DIPS, he was responsible for any task that required access to the internal system.

3.4.1.1 Slack

Slack was used to communicate with the stakeholders at DIPS, as the Trondheim offices were largely empty during the span of the project.

3.4.2 Microsoft Teams

Microsoft Teams is a free to use communication platform offering workspace chat, file storage, calendar and video conferencing. We chose Microsoft Teams as our primary tool for document sharing and as a meeting platform for video chats with our guidance councilor and the client due to its familiarity and popularity, also securing that any necessary outside interviews would go smoothly.

3.4.3 Discord

As a secondary communication tool we chose Discord. We wanted this as a tool for informal day to day communication internally.

3.4.4 Microsoft Word

For smaller independent documents we chose to use Microsoft Word for its integration with Microsoft Teams and the team's familiarity with the software.

3.4.5 Overleaf

For bigger documents and the main report we chose to use Overleaf, an online LaTex editor with live collaboration abilities. It is also a tool the team has used previously and are familiar with.

3.4.6 Azure DevOps

Azure DevOps is the tool we used for version control during the development and as a work-board to keep track of tasks and objectives during the sprints and the overall project progress. We chose this tool since it's a tool we are familiar with and its a tool that's free to use for students at NTNU.

3.4.7 Clockify

For keeping track of how much time each team member spent on different tasks we used a web site called Clockify, a free time-tracking tool with the possibility to categorize time usage and see statistics of how much time is spent on each item.

Chapter 4: Results

Our employer DIPS ASA wanted us to research and test Apple's new Extensible Enterprise Single Sign-on as a Single Sign-On solution for their mobile suite on iOS. They had already decided on using this solution into their system due to their old way of dealing with this being deprecated, but had not yet had the opportunity to figure out how it works and how to implement this properly. This meant we had to research how it works and create a working prototype that demonstrates its capabilities and usage.

4.1 Research Results

4.1.1 Experience Interview

As stated in 3.2, we wanted to gain some insight into the relevance of our research in the hospital setting. Our employer was unfortunately quite busy with a product delivery, reducing the resources they had to give us this insight. Initially, we had not planned to perform any surveys or formal interviews, as we figured that DIPS would be the sole providers of knowledge in the relevant field. Due to the fact that DIPS had limited experience with iOS and the EESSO technology, they reached out to an external System Architecture consultant who had gone through a similar process of building a prototype using EESSO in a hospital setting. The consultant was kind enough to grant us a meeting where we could debrief him about his experiences with the project. This was early in the process, and we figured that this would be very helpful, as we could get a rundown of how suitable EESSO was for the hospital environment.

As this was an external consultant who wanted to aid us out of goodwill, we wanted to make sure that we got what we needed from the meeting, instead of having to hassle them with follow-up questions at a later time. This made us look into qualitative interviews, which is a form of interview that focuses on gaining insight into the subject's experience and knowledge. This seemed fitting for the meeting, so we wanted to plan the interview following the guidelines for qualitative interview.

The minute of the interview can be found in Appendix D - Interview.

These were the goals that we wanted to achieve through the interview:

- Understand the subject's project as a whole
- Understand how an SSO extension is implemented and what the activity flow is like
- Find out how they plan to handle user session switching on work devices
- Discuss the best authentication method for the environment
- If they had any knowledge about extension implementation through Xamarin.Forms.

The following sections describe the main points that we learned from the interview.

4.1.1.1 Overview

The subject of our interview had played a big part in an inquiry into the use of digital tools in a hectic hospital environment. Their observations led them to discover that health personnel spent a surprisingly large amount of time authenticating their devices to access these tools. This was of a big annoyance to the employees. This led the consultants to develop a prototype using EESSO. They started by testing out the Kerberos Extension, which was very simple to implement with a few lines of code. Though, the hospital environment in question were using an identity server with PingFederate, which meant that they had to develop an SSO extension that targeted PingFederate as the identity provider. DIPS uses a custom configuration of OpenID Connect, so the exact implementation of the SSO extension was not relevant to our assignment.

4.1.1.2 Authentication Method

The prototype they developed utilized a two-factor authentication using a physical smart card and a PIN code. Employees can tap their cards against the phone, and are then prompted for a PIN code. Requiring a PIN code was chosen over username and password, as they have a lot of drawbacks, both by its impracticality and how insecure personal passwords can be.[13]

4.1.1.3 Single Sign-Out

When discussing the possibilities of Single Sign-Out on shared work devices, the subject told from experience that this was complicated.

The only solution to automatically log out a user, is to revoke their access tokens at a set time. Though, applications can prefer varying expiration times for their user's credentials. Revoking an access token will result in the entire device losing access, resulting in the user needing to authenticate again. This is not necessarily a big issue, but it is not an elegant solution. In

addition, knowing exactly when a user's session should expire can be complicated. Lining it up with a user's work shift is a possible solution, but there can be discrepancies of exactly when an employee goes off-duty. If a device changes owner before the access token is revoked, the new user may use the previous user's session without knowing.

There can also be applications involved in the SSO environment that should never be completely logged out, especially in a hospital setting. Consider an application that alerts health personnel whenever a patient calls for assistance from their room. This assistance could be vital to the patient. If a user's session expires, they won't know that they need to re-authenticate before bringing up the application on their phone. If a patient happens to call for assistance before they re-authenticate, the alert may never be pushed to the employee's phone due to them not being authenticated.

It is recommended that automatic Single Sign-Out is completely emitted as a solution to user switching. The subject speculated an alternative solution. When a user authenticates for the first time, you get an access token that is valid for e.g 24 hours. But when an application becomes idle, either by being closed or pushed to the background, the employee has to validate through a single credential, like a PIN code or an access card. If the employee fails to validate, they won't be able to access the application. Though, this will have to be implemented directly in the application. This is because an SSO extension won't be able to know when an application has entered a state where it has to be validated again.

The subject also speculated a solution using facial recognition technology. Adopting this as a way of authenticating the user would allow the user to stay authenticated as long as the user keeps looking at the device screen. For example, if the user looks away for 10 seconds, they could be logged out automatically. As soon as they look back at the screen, their face will be recognised again, and they get authenticated. This solution would be considered very secure, as it authenticates users on the fly and uses an authentication method that's hard to break. This is not a solution that has been tested out in any environment, but would also require implementation directly in the application.

4.1.1.4 Authenticating using iPhone as a device

Newer versions of iPhone are approved FIDO2 devices. This can in theory provide support for using the device itself as a form of identity. Further, two-factor authentication can be achieved simply from an employee tapping their card against the device.

4.1.1.5 Using Xamarin to trigger SSO extension

Like DIPS, the subject also mainly used Xamarin for mobile development, and had therefore performed a test triggering their SSO extension from a Xamarin application. This seemed to work fine. Xamarin makes sure to have a working implementation of all the native built-in libraries that are available through Swift. Using the built-in authentication services from Xamarin will trigger the same background operations that the same services would do in Swift. The subject said that their Xamarin developer would advise against using Xamarin to develop the actual extension. This is because the SSO extension would almost exclusively consist of native code, as there are native abstract classes that needs to be implemented. Using Xamarin for this would cause a lot of overhead, both in performance and development. They advised to just use Swift for the SSO extension itself.

4.1.2 Shared Devices

Looking into Shared Devices we can see that there is within Microsoft's SSO extension for Apple devices an option for supporting MDM and SSO through the Microsoft Authenticator app in devices that are in "Shared Device Mode", specifically created for frontline workers that rely on sharing single devices for accessing sensitive information. This entails the user logging in to the actual device with a unique identifier, and by that enabling them to use that account to access all the necessary applications. This also gives the safety of knowing that when the user logs out of the device, cookies are cleared so everything gets logged out properly and you ensure that no traces of user state are left behind. Properly setting up this sort of Shared Device allows for much greater control in terms of security, especially when dealing with sensitive data [14].

4.2 SSO Extension Implementation

As there are already two SSO extensions free on the market, we wanted to compare these extensions to DIPS developing their own extension that they could use for their mobile platform.

4.2.1 General Implementation

4.2.1.1 MDM Enrollment

One of the main requirements for using EESSO is that the host device has to be enrolled in an Mobile Device Management solution. This is because the SSO extension and its configuration has to be deployed through Mobile Device Management. DIPS uses Microsoft Intune as an Mobile Device Management provider, so this was the solution we had to use. Intune has built-in support

for EESSO, so configuring a profile containing the required configuration was easy.

When a profile is configured, we had to make sure that our iPhones got enrolled in the right device group in Intune. On the device, we had to install Microsoft's Company Portal application, which functions as a client towards Intune. After logging in with a company account, we could see the device in Intune and assign it to the device group. Once Intune indicated that enrollment was successful, we were able to see the applied profile on our iPhones, under:

Settings >General >Profiles & Device Management >Management Profile >More Details >Single Sign On Extension.

4.2.1.2 Triggering an extension

An SSO extension can be triggered in two ways; either through native Swift networking libraries, or using the class *ASAuthorizationSingleSignOnProvider* from the *Authentication Services* library. Note that all of these libraries are available in both Swift and Xamarin.

When using a native library, Redirect extensions will automatically intercept the outgoing request on any configured URL's, while Credential extensions will handle all HTTP challenges returned from the configured hosts. When the extension has finished handling the request, it will append the resulting authorization headers to the request and resume the original procedure. The calling application will then receive a fully handled response, as the authentication already has been processed. If no extension is available on the device, the native library method will process as it usually would.

When using the *ASAuthorizationSingleSignOnProvider* class, you only have to specify the URL for the targeted identity provider / server. You can add additional authorization query options to the request. You then have to create an instance of the *ASAuthorizationController* class, which will handle the presentation of the authentication request. The request can then be triggered by setting the *requestedOperation* (for example "Login") and calling the *performRequests()* method on the controller (see figure 4.5 for code example). The calling application can then handle the response by implementing the protocol *ASAuthorizationControllerDelegate* and its callback methods *authorizationController(...)*. An object implementing the protocol has to be set as the delegate on the *ASAuthorizationController*. The controller will also need a view controller implementing the *ASAuthorizationControllerPresentationContextProviding* protocol. The drawback for this method is that the application will be dependant on an SSO extension for authentication.

```swift
func signIn() {
    // ...
    let authController = ASAuthorizationController(authorizationRequests: [request])
    authController.delegate = AuthorizationController()
    // ...

    authController.performRequests()

}

class AuthorizationController : NSObject, ASAuthorizationControllerDelegate {
    func authorizationController(controller: ASAuthorizationController,
                                 didCompleteWithError error: Error) {

    }

    func authorizationController(controller: ASAuthorizationController,
                                 didCompleteWithAuthorization authorization: ASAuthorization) {

    }
}

```

Figure 4.1: The delegate methods that can be implemented to handle operation results

4.2.2 Prototype using Azure Active Directory SSO Extension

4.2.2.1 Client Application

We made a simple prototype application that targeted the Azure AD SSO extension by Microsoft for authentication. This prototype was implemented in Xamarin, and used MSAL for communicating with the identity provider. This library is heavily integrated with Azure AD, so authentication could be implemented through few lines of code (figure 4.2) [15]. When triggering the login, we get presented with a login screen. If we have logged in earlier, we can see that Microsoft Authenticator provides an overview of users that are already logging in. Selecting a user lets us log in without having to re-authenticate (figure 4.3). Unfortunately, we were not able to test this further due to running out of time.

```
0 references
protected override void OnStart()
{
    PublicClientApp = PublicClientApplicationBuilder.Create(ClientId)
        .WithRedirectUri("https://login.microsoftonline.com/common/oauth2/nativeclient")
        .WithAuthority(AzureCloudInstance.AzurePublic, Tenant)
        .Build();
}
```

```
    0 references
    private async void OnClick(object sender, EventArgs e)
    {
        var authResult = await App.PublicClientApp.AcquireTokenInteractive(_scopes)
                                .WithParentActivityOrWindow(Activity.Main)
                                .ExecuteAsync();
    }
}
```

Figure 4.2: The code blocks we mainly used to trigger the authentication through MSAL

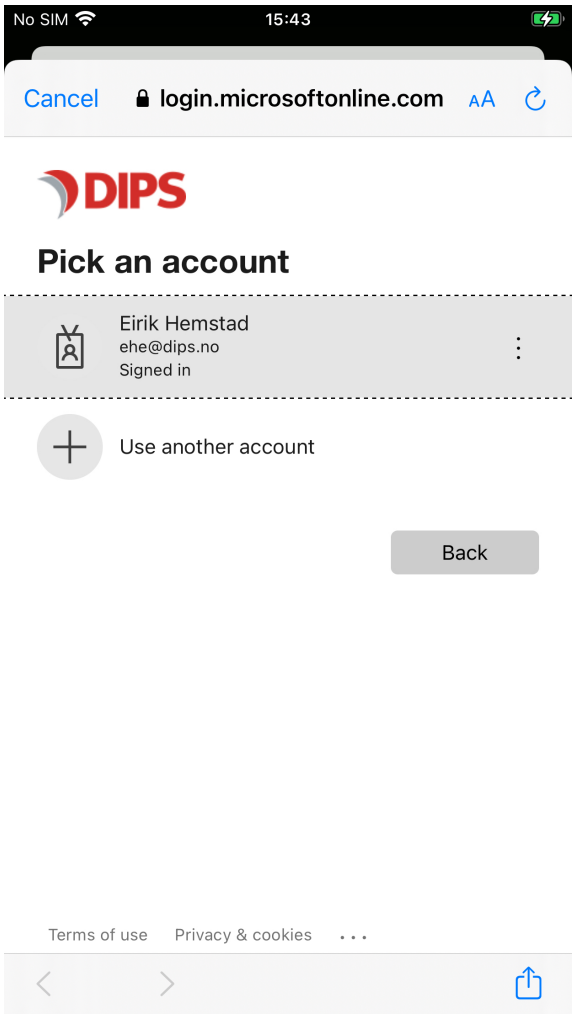


Figure 4.3: The Azure AD extension shows all users that are logged in through Microsoft Authenticator

4.2.2.2 MDM Configuration

In Microsoft Intune, we added a new configuration profile, of the type *Device Feature*. Intune provided a template called Single Sign-On App Extension. Here, we could set the extension type to *Microsoft Azure AD* and add the bundle identifiers for the applications that should use the extension [15].

4.2.3 Prototype using Apple's Kerberos Extension

Configuring an application to use the Kerberos Extension was very simple. Just a few lines of code and a valid MDM Payload containing the extension configuration was enough to successfully trigger the extension from a host application.

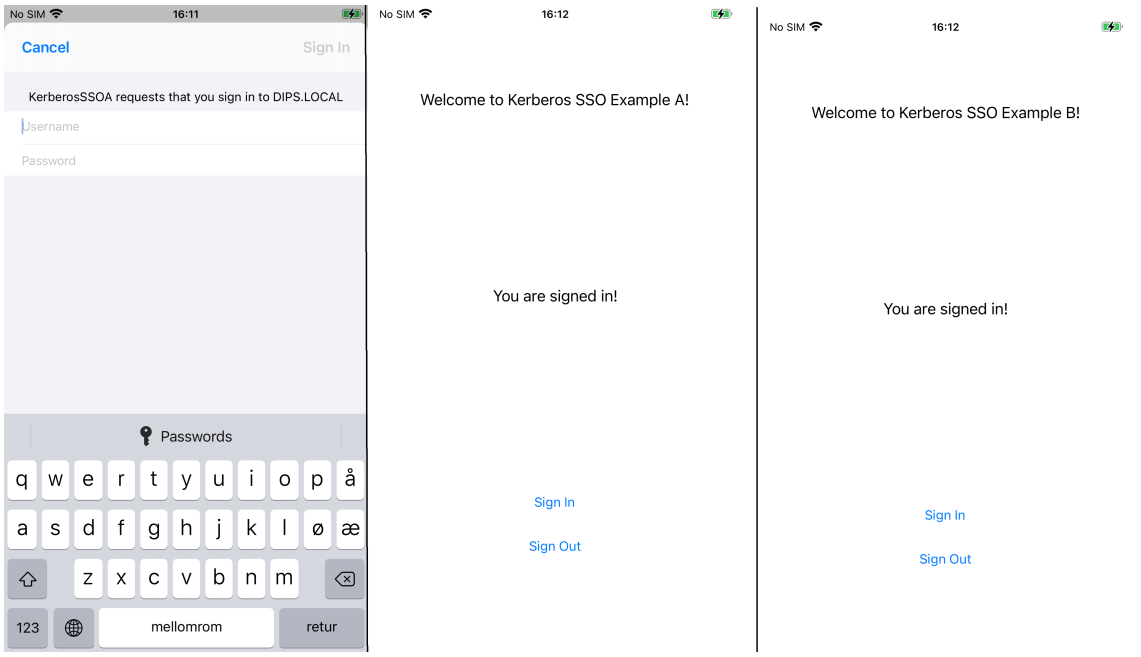


Figure 4.4: The different views of the Kerberos prototype, depicting how the extension authenticates the user with only one initial prompt

4.2.3.1 Client Application

When testing the capabilities of Apple's Kerberos Extension, we created a simple prototype in Xamarin and Swift. This was an iterative prototype. Once we got the extension triggered in Swift, we switched over to Xamarin to see how easily the code could be migrated. We ended up with a simple prototype that would trigger login on application start. This would trigger the extension, prompting the user to log in with their domain username and password. We used DIPS' internal domain for testing. If the user already was authenticated, the extension would not prompt for login. A label in the middle of the application indicates whether the extension has registered the user as authenticated or not.

We deployed two identical applications with different bundle identifiers, to ensure that they would work separately. Logging into the first application authenticates the user, and the extension registers the user as being logged in. When restarting the application, or opening the other application, the extension would be requested to login again. But since the user already is authenticated through the extension, the login prompt did not trigger, and both applications show that the user is logged in.

We got some pointers on how to trigger the extension from the WWDC 2019 talk [7]. This code could easily be translated into C#. Note that the *sender* object must be a view controller that implements the *IASAuthorizationControllerPresentationContextProviding* interface. The *DidComplete(...)* delegate method would not be called if this object was null.

```
0 references
public AuthenticationService()
{
    m_AuthProvider = ASAuthorizationSingleSignOnProvider.CreateProvider(new NSUrl("realm://dips.local"));
}

3 references
public void SignIn(object sender)
{

    if (m_AuthProvider.CanPerformAuthorization)
    {
        var request = m_AuthProvider.CreateRequest();
        request.RequestedOperation = ASAuthorizationOperation.Login;
        m_AuthController = new ASAuthorizationController(new[] { request });
        m_AuthController.Delegate = this;
        m_AuthController.PresentationContextProvider = sender as IASAuthorizationControllerPresentationContextProviding;
        m_AuthController.PerformRequests();
    }
}

0 references
public override void DidComplete(ASAuthorizationController controller, ASAuthorization authorization)
{
    var request = controller.AuthorizationRequests[0] as ASAuthorizationSingleSignOnRequest;

    if (request.RequestedOperation == ASAuthorizationOperation.Login)
    {
        var credential = authorization.GetCredential<ASAuthorizationSingleSignOnCredential>();
        var response = credential.AuthenticatedResponse;

        if (response.StatusCode == 200)
        {
            IsSignedIn = true;
        }
        else
        {
            IsSignedIn = false;
        }

        OnSignedInChanged?.Invoke(this, null);
    }
    else if (request.RequestedOperation == ASAuthorizationOperation.Logout)
    {
        IsSignedIn = false;

        OnSignedInChanged?.Invoke(this, null);
    }
}
```

Figure 4.5: An example of how we triggered the extension and handled the response

4.2.3.2 MDM Configuration

As with the Azure AD SSO extension, Intune had official support for Kerberos Extensions. Intune provided the extension type *Kerberos* which let us easily configure the profile (figure 4.6).

4.2.4 Custom SSO Extension Prototype

DIPS are currently using OpenID Connect to authenticate through their mobile platform, using their proprietary SSO solution. Therefore, we wanted to research how DIPS could implement their own SSO extension, as there is no available extension for OpenID Connect at this time.

4.2.4.1 Extension type

We chose to develop a Redirect Extension, as Apple has stated that Redirect is suited for modern authentication flows like OpenID Connect or OAuth 2.0 [7].

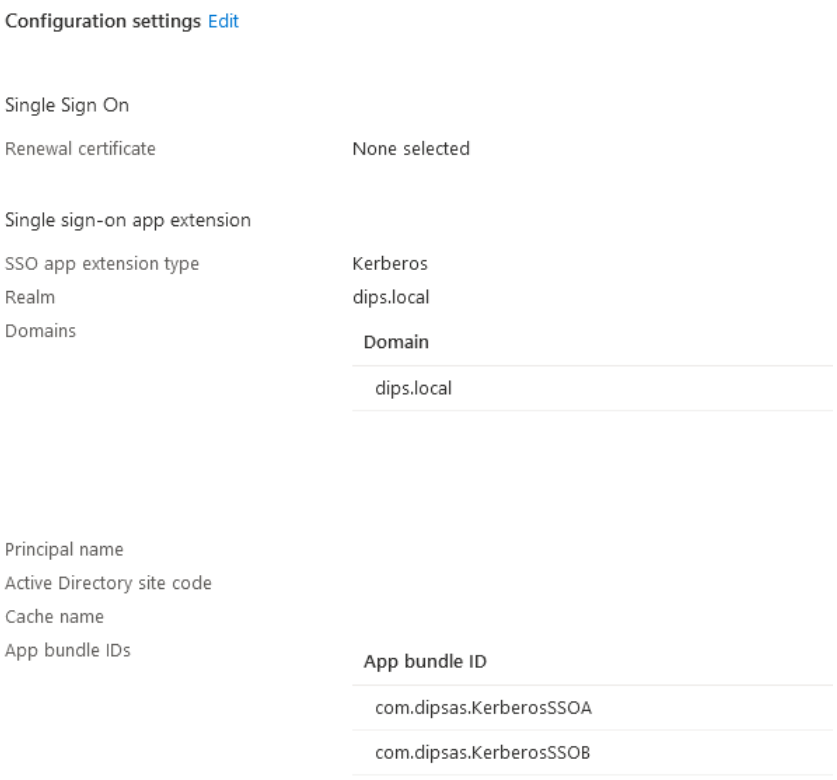


Figure 4.6: The MDM payload for the Kerberos Extension

4.2.4.2 Associated Domains

Redirect Extensions requires a certain configuration for Universal Links to function. Since our SSO extension has to communicate with an authorization server, the operating system wants to ensure that the extension is actually allowed to communicate with the server. For this to work, we had to declare the authorization server as an associated domain to the app. We did this by adding an *Associated Domains* entitlement to the application, and add the Fully Qualified Domain Name of the server to the list with the "authsrv" service type (figure 4.7). We had to enable alternate mode for the operating system to be able to directly access the server. More about this setting in [SECTION]

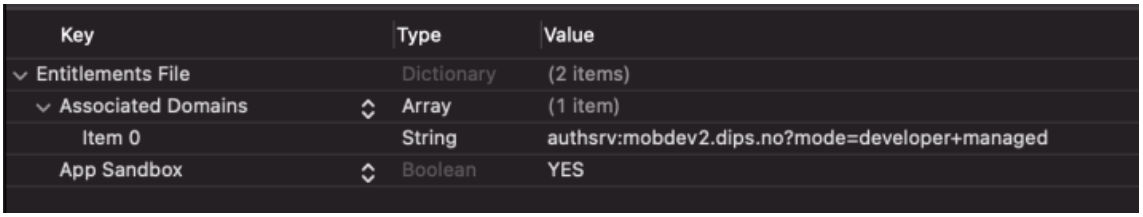


Figure 4.7: The entitlement file with the "authsrv" associated domain

4.2.4.3 Apple App Site Association

Another requirement for Associated Domains is that every associated domain has an *apple-app-site-association* file. This is a JSON file that needs to list every application that is approved to communicate with the server, and the associated service type. This file **must** be located at *https://<server-name>/.well-known/apple-app-site-assocation*.

4.2.4.4 MDM Configuration

When configuring a custom SSO extension, we have to specify a type of either Redirect or Credential. We selected Redirect, and added the bundle identifier for the extension, along with the URL's which the extension should intercept (figure 4.8).

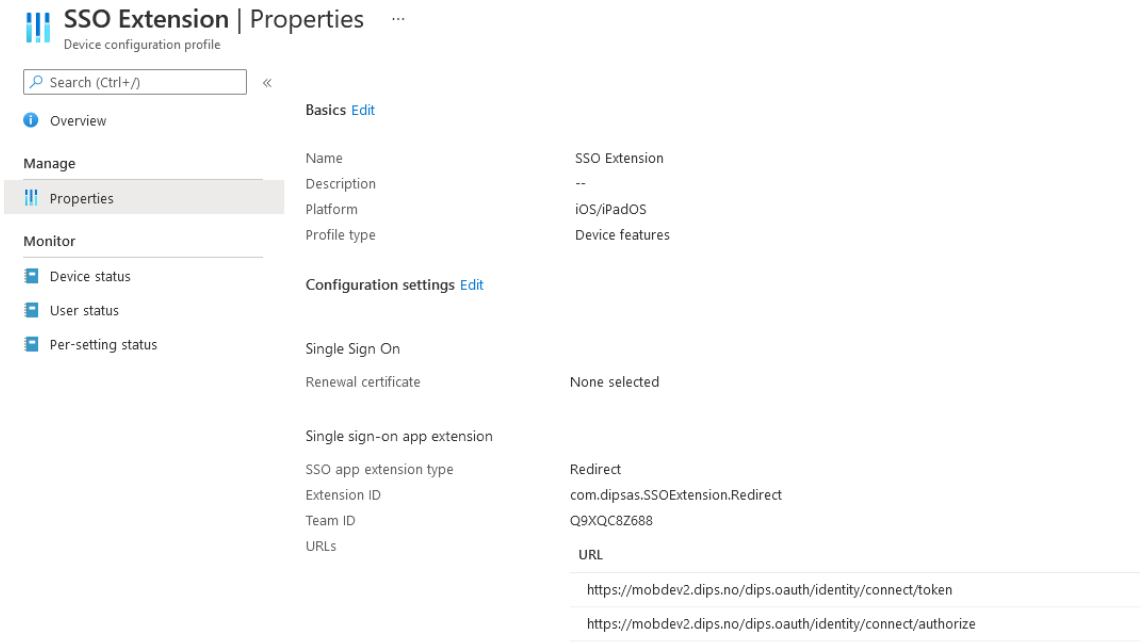


Figure 4.8: The profile configuration we set up in Intune

4.2.4.5 Creating an SSO extension

App extensions in Swift must be created by adding a target to an existing application. This application may only serve as a container for the extension, but may also be extended with other functionality such as password reset. When creating a target, the developer has to select a template. SSO extensions are created using the Authentication Services template. This gives us a template that provides some empty methods that needs to be implemented. These methods will be called whenever the extension is triggered, and must return by calling pre-defined methods for a successful or failed authorization (figure 4.9).

```swift
import UIKit
import AuthenticationServices

class AuthenticationViewController: UIViewController {

    var authorizationRequest: ASAuthorizationProviderExtensionAuthorizationRequest?

    override func loadView() {
        super.loadView()
        // Do any additional setup after loading the view.
    }

    override var nibName: String? {
        return "AuthenticationViewController"
    }
}

extension AuthenticationViewController:
    ASAuthorizationProviderExtensionAuthorizationRequestHandler {

    public func beginAuthorization(with request:
        ASAuthorizationProviderExtensionAuthorizationRequest) {
        self.authorizationRequest = request

        // Call this to indicate immediate authorization succeeded.
        let authorizationHeaders = [String: String]() // TODO: Fill in appropriate
            authorization headers.
        request.complete(httpAuthorizationHeaders: authorizationHeaders)

        // Or present authorization view and call self.authorizationRequest.complete() later
            after handling interactive authorization.
        //request.presentAuthorizationViewController(completion: { (success, error) in
        //     if error != nil {
        //         request.complete(error: error!)
        //     }
        //})
    }

    public func cancelAuthorization(with request:
        ASAuthorizationProviderExtensionAuthorizationRequest) {
        self.authorizationRequest = request

        request.cancel()
    }
}
```

Figure 4.9: The template for creating a new SSO extension

4.2.4.6 Deploying and loading a custom SSO extension

When we wanted to deploy our SSO extension, we had to upload the extension's host application to Microsoft Intune. We chose to upload a 'line-of-business' application, which let us upload the IPA file directly. No further configuration for the application was needed.

When the application is started on the test device, we can see from the device logs that a background process called AppSSODaemon gets triggered. The daemon logs whether the extension was loaded successfully or not. We used the daemon logs to deduce how the extension is loaded, as there is no proper documentation of this process. This deduction is visualised in figure 4.10.

4.2.4.7 Prototype Stages

4.2.4.7.1 Early prototyping

Before we actually started work on the prototypes for the final solution, we wanted to get some experience in actually programming in Swift. Therefore, we did a bit of individual programming where we tried to implement a simple iOS application that could be deployed in a simulator or on an actual device. This way, we gained some experience with the Swift programming language,

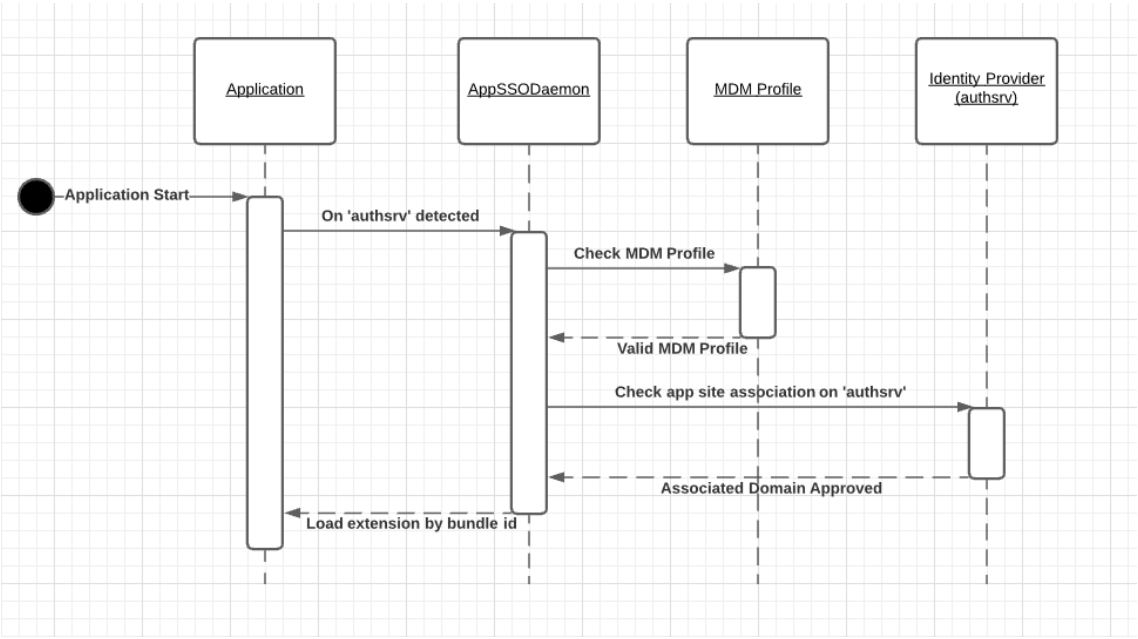


Figure 4.10: Sequence diagram of how we picture the process of an extension being loaded. 'authsrv' is the Associated Domain with 'authsrv' type.

and how we could use XCode to actually develop a simple application. Apple has a very unique ecosystem, so this was a necessary step in understanding how we would go about starting the actual prototypes. [FIGURE]

4.2.4.7.2 First Prototype

We decided that our first actual prototype should allow us to see how we could interact with DIPS' identity server. As this step was crucial in actual getting a working authorization flow, we figured that this would be a natural first step. The prototype ended up being a simple iOS application that allowed the user to log in to DIPS' identity server, running on OpenID Connect. We used the Swift library AppAuth for simplicity's sake, as it's always wise to use existing libraries when available. When the user had logged in, restricted data would be fetched from the resource server and logged to the console, proving that the authorization process has succeeded. Using AppAuth proved to be a good solution for implementing OpenID Connect. The library streamlines a lot of the steps that are involved in authorizing through OIDC, and made it easy to work with the resulting authorization keys.

4.2.4.7.3 Second Prototype

For our second prototype, we wanted to explore exactly how an App Extension works, and how EESSO can be utilized to trigger an SSO extension. This proved to be one of the most challenging steps in prototyping, as we had to properly configure an EESSO MDM profile, set up Associated

Domains and implement a very limited set of example code. When things did not work as expected, debugging what went wrong was very challenging, as EESSO is dependent on work from background processes in the operating system. Apple provided a set of debugging tools specifically for EESSO, but the output that we logged was poorly documented and not particularly helpful. More details on these issues is provided in chapter 5.

4.2.4.7.4 Third Prototype

For the third prototype, we wanted to implement AppAuth into an actual extension, to get closer to a final result. Since there was a lot of waiting involved in getting help from Apple technical support, we wanted to use this time to make some progress on the actual functionality of the extension. We started by migrating our AppAuth implementation from Prototype 1 into a new extension, so that the extension actually could handle authorization requests. This shift of focus helped us to easier understand the connection that had to be made to trigger an SSO extension. We found a solution to the problems introduced in Prototype 2, and were finally able to approve the authorization server as an associated domain.

4.2.4.8 Final Result

Unfortunately, we were barely able to develop functional prototype for a custom Redirect Extension. This was due to a large number of issues that was never resolved, despite continuous dialogue with Apple technical support. We touch more upon these issues in chapter 5.

4.3 Engineering Process Results

4.3.1 Scrum

The project started out with a scrum process adapted for smaller teams but it soon became apparent that due to the nature of the work and our unfamiliarity with the language and process that we needed to go over to more of an iterative method where we worked our way step by step through learning the language, learning to implement extensions with it and work our way through prototypes for an authentication system. This mode of work was not very suited for a Scrum process since we had to finish each prototype, see how it worked, then start the process all over again.

4.3.2 Working environment

Due to the ongoing pandemic we had a lot of challenges when it came to our work process, since we had to adapt to all our collaboration digitally which lead to some challenges compared to how we were used to working together on projects. We had little physical encounters with employees at DIPS, which made it more difficult to be integrated into the systems and infrastructure of the company. This also created an unfortunate gap in how the team members could relate to the company.

The mobile development team at DIPS was unfortunately very busy with a product delivery. This made it harder to maintain a continuous dialogue between the team members and the product owner.

4.3.3 Communication with supervisor

Communication with our supervisor was scheduled as a weekly meeting on Teams to update him on our progress and get feedback on the progress of the report and any questions we might have at the time.

4.4 Administrative Results

4.4.1 Progress plan

At the beginning of the project we set up a simple Gantt-diagram of how we expected our progress to roughly pan out.

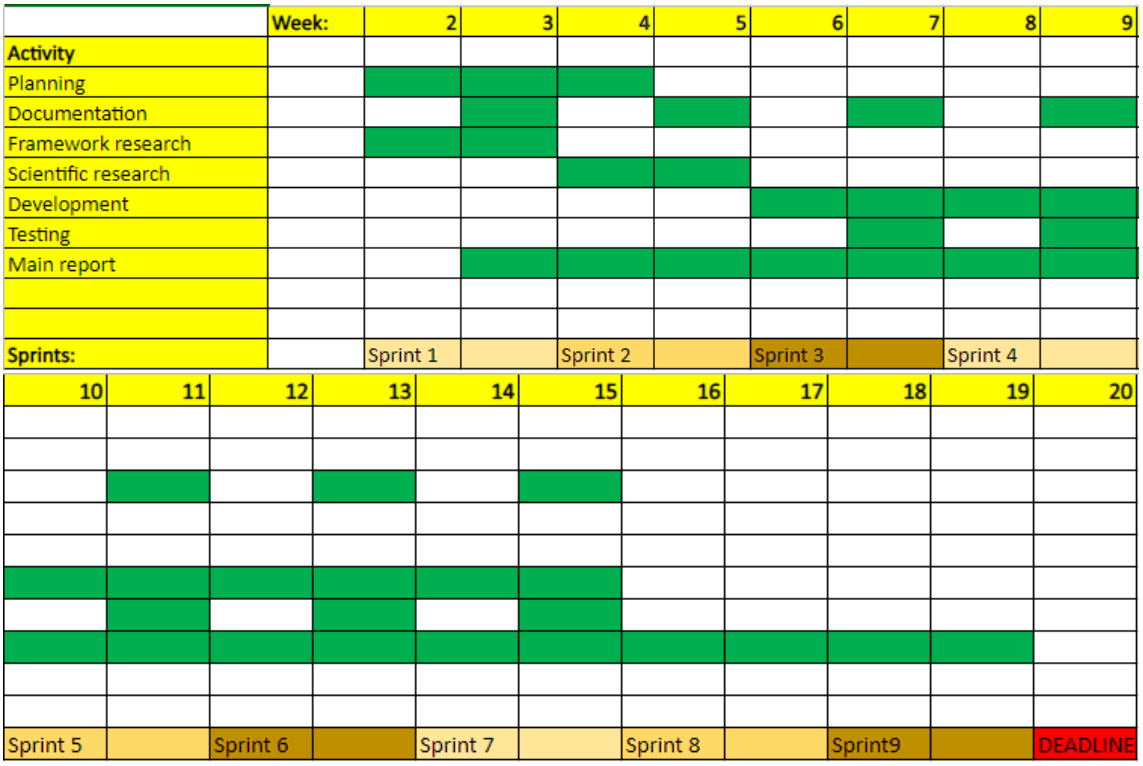


Figure 4.11: Progress plan at the beginning of the project

As you can see in this figure, we expected to run sprints were we spent the first two sprints researching and planning, then the remaining sprints developing and documenting our work as we went along, but this approach only lasted for a few weeks until we realized that we needed much more knowledge and create several prototypes to test out in order to get where we wanted.
So the end results was more along the lines of this figure:

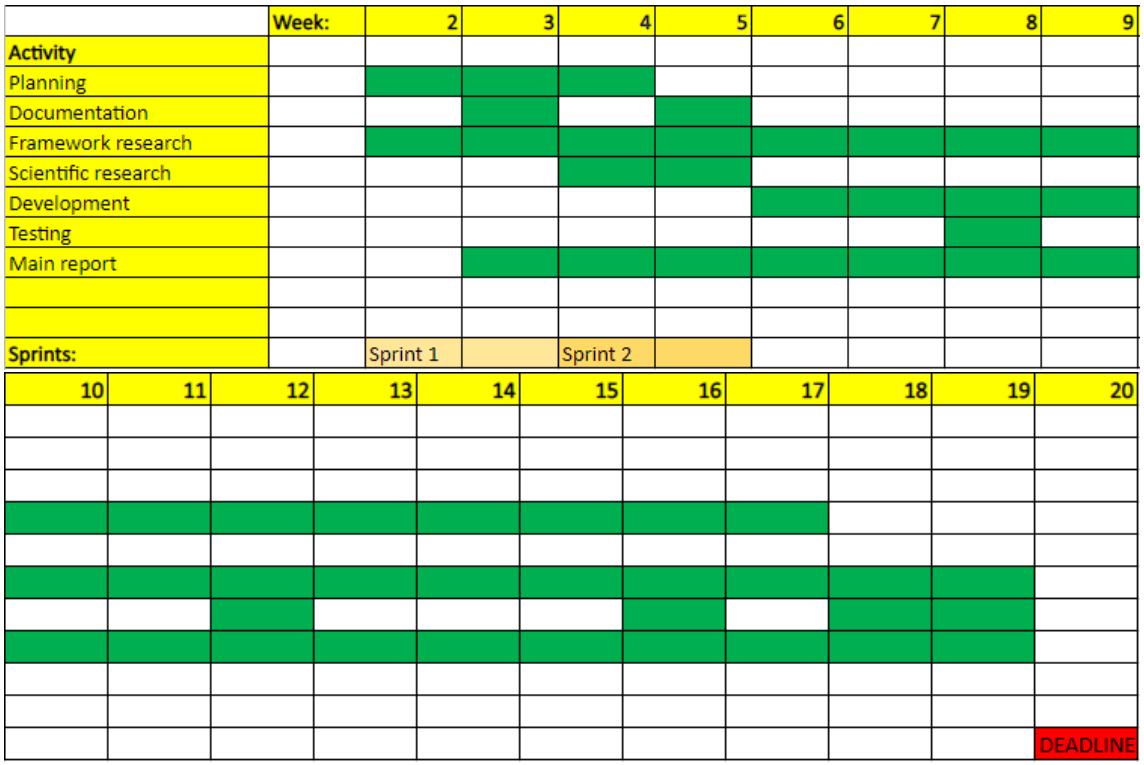


Figure 4.12: Progress plan at the end of the project

As is visible here the approach changed into a more iterative process without sprints, with a lot more research and a continued development cycle where we created a new prototype and tested it regularly. This also lead to less documentation along the way since we were spending so much of our time elsewhere. Due to this need for extra research there was what we felt was a lot of delays and unnecessary lost time waiting for responses from Apple when we ran into bugs and difficulties with our implementation of the extension. We also struggled to progress since we chose a difficult path when we elected to start by trying to create our own extension from the start instead of working our way up by testing the already existing solutions first to learn from how they worked.

4.4.2 Summary of hours

Throughout the project we logged hours spent on different categories, expecting to reach in the area of 500 hours spent each. With the most time spent on the report, and research and development as the next two major factors.

Activity	Hours
Documentation	101
Report	397.5
Other	70.5
Planning	8
Research	144
Develoment	167.5
Testing	8
Total	896.5

Figure 4.13: Total hours by category

As you can see from this chart we ended up with a little over 800 hours total, not quite as much as we wanted, but we felt we put what time we had available into the project.

Chapter 5: Discussion

5.1 Product

5.1.1 Research Process

The knowledge we gathered and the prototypes we produced were a result of continuous research and experimenting. From the very start of the project, we experimented extensively with Xamarin, Swift and EESSO implementations specifically.

Since none of us had any experience with mobile development in neither Xamarin or Swift, we wanted to get familiar with the workflow, architecture and relevant programming languages. En employee at DIPS gave us an informal introductory course to mobile development in Xamarin to get acquainted with how DIPS develop their applications, and also how we could get started making some simple prototypes. This was valuable in getting insight in how DIPS ultimately should implement SSO extensions into their mobile suite.

After we started looking into the technical details of EESSO, we had trouble finding any documentation on implementing it in an application made with Xamarin. We realized that it would be wise to attempt an implementation in Swift before we potentially added the extra complexity of trying to get it to work in Xamarin. Therefore, we spent some time learning to create simple applications in Swift, so that we could understand the language and how we could produce simple applications for iOS.

5.1.2 Custom SSO Extension

When we felt comfortable using Swift, we wanted to get working on the prototype for the custom SSO extension. We started by producing a simple prototype application for communicating with the DIPS identity server. This process made us familiar with how DIPS uses OpenID Connect to authenticate and authorize their application users, and how this can be implemented using iOS libraries like AppAuth.

At this point, we started prototyping a custom Redirect Extension using EESSO (as described in section 4.2.4). This proved to be the most time consuming and troublesome part of the process, as Universal Links had a lot of strict requirements. We spent a lot more time on these issues than we had anticipated in our original time schedule. See also section 5.3.2. We filed a bug report to Apple, which can be seen under Appendix C - Apple Bug Report.

5.1.2.1 Redirecting calls for Apple App Site Association file

We initially set up the AASA file to be available at a virtual directory on the server, meaning that this URL would redirect to the actual location of the file instead. This configuration was the easiest solution and was expected to work fine. We eventually discovered that this could cause complications when trying to download the file, as not all requests would work properly with redirection. Therefore, we made sure to place the file at the proper location instead of relying on a redirect.

5.1.2.2 Associated Domain not getting approved

The authorization server that we used for testing was meant for a test environment, and was not signed with a proper SSL certificate for production use. Universal Links has strict requirements for such certificates, and does not allow self-signed certificates. Therefore, we had trouble getting the system to approve the server as an associated domain. We also discovered that newer versions of iOS used an online Content Delivery Network (CDN) for keeping track of all Apple App Site Association files [16]. This requirement meant that our server had to be available online for it to be discovered by the system. We could bypass this by appending *?mode=developer+managed* to the Associated Domain. This setting also allowed the application to bypass the Secure Sockets Layer (SSL) requirement, as long as the application was signed with a Development Provisioning Profile. Still, we discovered that the system did not accept the server properly. When discussing this with Apple Support, we eventually found out that the Universal Links approval process would not accept *.local* domains. We fixed this by getting a proper domain name and SSL certificate for the server.

5.1.2.3 Not AppSSO URL

As described in section 3.1.1.2, Redirect extensions are triggered either by intercepting outgoing requests towards specific URL's, or through special SSO-specific operations. When attempting to use either of these methods to trigger our custom Redirect extension, we got the error message *Not AppSSO URL*. This indicates that the AppSSODaemon does not recognize the request URL to be a URL meant to be handled by SSO extensions. We had made sure to include the right URL's in the MDM Payload. When reporting this issue to Apple Support, they had no information on

what could cause this error. They urged us to file a bug report describing the issue. This issue has unfortunately not been resolved at the time of writing this report.

5.1.2.4 Unable to retrieve Apple App Site Association file

While debugging issues with the *Not AppSSO URL* error, we discovered swcutil, which could be used to manually validate Associated Domains on MacOS. We tried using the command *swcutil dl -d mobdev2.dips.no* to manually download the file from the identity server. This did not succeed, and we got the error message *SWCErrorDomain error 7*. When reporting this error to Apple Support, they replied that the error indicated that the AASA download was interrupted by the CDN used by Universal Links. As stated in section 5.1.2.2, we had configured our application to bypass the CDN by using alternate mode. The reason for this is unclear, and we had to include this in a bug report. This issue has not been resolved at the time of writing this report.

5.1.3 Azure AD SSO Extension

In section 4.2.2, we described how we set up Microsoft's SSO extension for Azure Active Directory. In this case, the documentation provided by Microsoft was sufficient to smoothly implement a prototype that triggers the extension using MSAL, a library which provides authentication with Azure AD regardless of the presence of the associated extension. The extension is bundled together with Microsoft Authenticator on iOS. Microsoft Authenticator acts as a password manager that also holds information of which users are currently logged in. When MSAL is triggered, the extension will plug into Microsoft Authenticator and presents the user with either a login form, or the option to select an already authenticated user. Selecting an authenticated user won't require the user to provide any more credentials.

In our prototype, the authentication with the extension is working, but more time would be needed to get the authorization to work. The solution using Microsoft Authentication Library (MSAL) should be looked at further, as it supports both OpenID Connect and OAuth 2.0, (as long as the user belongs to Azure AD). The simplicity and flexibility of its implementation may make it a viable form of authentication for DIPS.

5.1.4 Kerberos Extension

As shown in section 4.2.3, implementing a Kerberos Extension in Xamarin was efficient, required few code lines, and Intune provided a simple way of configuring the required MDM Payload. Getting the extension to log the user in and out was straightforward, and the workflow shows that *ASAuthorizationSingleSignOnProvider* is the best method for triggering any SSO extension and handling the response. Here we would like to specify that we didn't test the authorization capab-

ilities of the Kerberos Extension properly as the test server we were using would only authorize tickets granted with OpenID Connect, and Kerberos does not support OpenID Connect at this time.

5.2 Scientific Results

5.2.1 Apple Technology

Apple Inc., the proprietor of the technology stack, has a rather unique set of practices, and understanding the product environment from the perspective of both users and developers proved to be a challenge.

We quickly realized that there was a lot of research necessary for understanding the platform in which the solution should be implemented. Although DIPS has been developing on the mobile platform for some time, they have mainly focused on cross-platform development with Xamarin. Therefore, there has not been much focus on the Apple platform specifically, and DIPS does not have much internal expertise or experience with Apple technology.

Apple technology is largely built around high-level programming, where the operating system provides a large collection of functionality that the application has to plug into. The API's are rather limited in favor of platform-specific automation of generic tasks and processes. Further, the platform introduces a lot of concepts and standards that aren't particularly applicable to other leading platforms.

In addition to this, Apple are notorious for not providing sufficient documentation and code examples [17]. The only official documentation for Extensible Enterprise Single Sign-On was a recording of a tech talk from WWDC 2020 [7]. This talk goes in-depth into what EESSO has to offer, but provides few code examples and leaves out key points about its implementation. The lack of documentation proved to be a large issue in developing a proper prototype for a custom extension. Investigating error messages and other faults was challenging, as we often were unable to find any information on the errors. There was no official documentation found for troubleshooting EESSO, and all of the relevant forum posts we found were left unanswered. We managed to come into contact with official Apple Technical support, but unfortunately found their answers to be vague and lacking technical insight of the issues we were experiencing. The communication with Apple took place via e-mail with representatives from a different time zone. When we sent questions early in the day we would receive replies in the evening at the earliest, which made the communication slow and tedious, especially when there were miscommunication that needed clarification. Dialogue with technical support ended with them urging us to open a bug report, as they were unfamiliar with the issues we were having. Opening a bug report, we were promised,

would let the developers of EESSO take a look at the issues directly.

5.2.2 Research Questions

5.2.2.1 How can Apple Extensions be implemented and used with Xamarin

App Extensions that are developed using Authentication Services, will intercept any network requests that are triggered through the standard networking libraries available in Swift. They can also be triggered using specific operations.

A mobile application project in Xamarin is structured in such a way that platform-specific code (usually I/O operations or specific UI components) for iOS and Android are completely separate and unaware of eachothers existence. The iOS and Android libraries are only dependent on the platform agnostic core library, while the core library is unaware of the platform libraries (see figure 5.1). If a Xamarin application is to support EESSO, then each platform library needs to implement its own authentication process, as EESSO can only be triggered through native iOS functions or classes. This can lead to complex solutions when developing applications, as every application needs to define two different implementations of authentication.

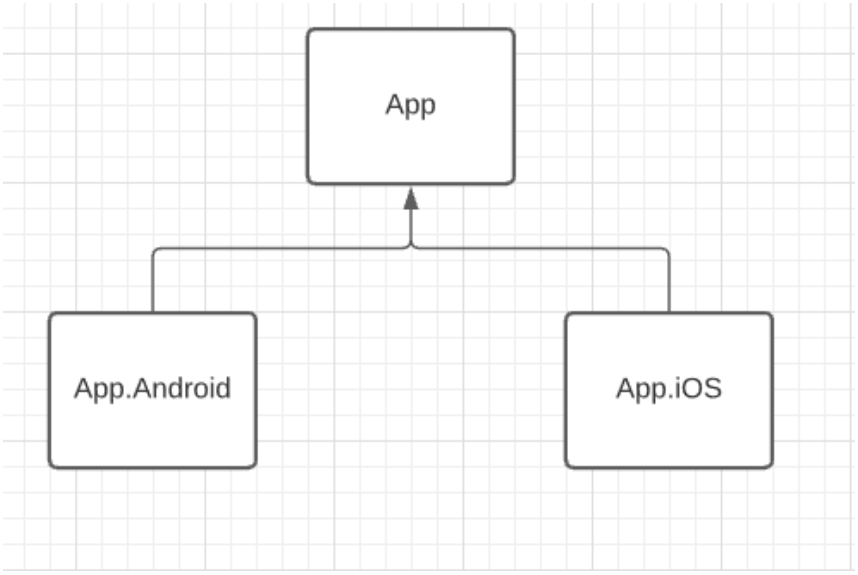


Figure 5.1: Depiction of the dependency flow between the modules of a Xamarin project

DIPS already has a program library that can be integrated to provide sign-in support. As stated in Chapter 1, the SSO capabilities of this library are not functional on iOS 13 or above. The library can still be used in iOS, although SSO will not work, and the user will have to sign in on every application separately. Therefore, the existing library can be used as a fallback solution for authentication when an SSO extension is not available. The native iOS class *ASAuthorizationSingleSignOnProvider* provides us with the boolean property *canPerformAuthorization*. This method can be called to check whether an extension is loaded on the operating system, and whether

the specified identity provider is available for authorization. The latter property can be used to determine whether the iOS-specific SSO extension can be called, or whether the application should use the default sign-in library, at the expense of losing SSO. Such a choice may also be implemented directly in the sign-in library. This method decreases the amount of refactoring needed to use an SSO extension in a Xamarin project.

As noted in 4.2.1.2, triggering an extension using *ASAuthorizationSingleSignOnProvider*, requires the authorization controller to have a presentation context, which implements the protocol *ASAuthorizationControllerPresentationContextProvider*, i.e the authentication view controller in any application needs an iOS-specific implementation that implements this protocol.

5.2.2.2 How can Single Sign-Out be handled on non-personal mobile devices?

As part of the interview with the external consultant we talked about how handling Single Sign-Out could be done, and considered many approaches, but didn't come to any concrete solution due to the different needs of each department since different departments might have different duration shifts. This might then be handled either in user configurations or in the device configuration where it considers on login the current time and sets a session expiration time according to which shift is logging in. A potential issue with this solution is that some apps shouldn't be logged out automatically due to the nature of their tasks, so a full single sign-out solution might be impractical in some situations. Other solutions to consider might be partial single sign-out where a subset of apps share a sign-out while the most important apps have their own sign-out either manually or on a timer. Although this could lead to these apps not being properly exited by the users at the end of their shift. The most practical solution would probably be to tie the entire log-out process to the device, so when the user logs out of the device it disables the current tokens on all applications automatically, this way it would be cleaned out and ready for the next user of the device to log in without any interference or overlap from the previous user. This option is not currently viable due to the devices only having a shared login and the users only having separate log-ins for the applications. Single Sign-Out can possibly be implemented later along with an NFC solution for logging in to the device and the applications, where the employees use their identification badges as authentication to the device.

5.2.2.3 Are there advantages to developing a proprietary SSO extension, or is it better to use an existing maintained extension?

There are several advantages to each solution, depending on field of use and compatibility needs. If you are developing for your own apps, a proprietary extension would be better as it gives you greater control over all the data and how everything connects together, which allows you to include more specialised features. If you are developing for use in a field with several application providers

you might want to use an existing SSO extension to aim for cross-provider SSO. This can improve ease of use for the end-user by making it possible to use the same solution as other providers. In this case, users only need a single authentication to access all the applications they use regardless of provider. Although you would still be depending on other providers to implement the same solution, we have seen that existing SSO extensions are designed to be easy to integrate into existing systems, given that the relevant identity providers are in place.

When looking at the SSO extensions that have been developed so far, it seems that Apple has intended this technology to mainly be utilized by identity providers. Ideally, each identity provider has their own SSO extension that all their clients can use for cross-vendor SSO, instead of a large amount of companies using their own proprietary extensions internally. In reality, there are few solutions on the market, and waiting for an identity provider to release their own extension can take an unknown amount of time. We have not been able to find any indications that there are more SSO extension in development for public consumption.

5.3 Administrative Process

5.3.1 Progress plan evaluation

The progress plan ended up with several discrepancies, some of these very early in the project due to several reasons.

- We had to spend more time than anticipated on research in the beginning.
- We had to get the equipment needed to work on iOS and MacOS while being contained to a work from home situation.
- We spent more time than planned trying to get the custom SSO extension to work and contacting Apple tech support due to a lack of documentation.

These items were the most important points of delay for us on the project. Other issues took more time then planned but they were minor problems

5.3.2 Development Process

The development process turned out differently than what we would have wanted, since we ran into challenges working with things we had never used before and trying to implement an extension that had no previously documented usage and little to no actual documentation from Apple. We had to do a lot of research right from the beginning and figure out by ourselves how everything worked.

This process caused us to start over several times while trying to figure out how to approach the problem, and how best to create a solution that would fit the criteria of our assignment.

Our initial goal was mainly to create a functioning prototype of a custom SSO extension. As stated earlier, this was based on the fact that DIPS relied on the OpenID Connect protocol for authorization, which no existing extension supported. We quickly encountered technical issues that were relatively small and seemed easy to overcome. But as the communication with Apple was slow and only yielded little progress, the problem started eating up a lot of time. Unfortunately, we were so caught up in solving these issues and getting the custom extension to trigger that we almost forgot about the existing extensions using EESSO. These solutions weren't directly suitable for DIPS' environment, so we were quick to write them off.

5.3.3 Teamwork

The team knew each other well before the project started, and had worked together several times previously, so we didn't need to focus on team building at the start of the project. At the start of the project we created a work contract to agree on details regarding goals and expectations.

Chapter 6: Conclusion and further work

6.1 Assignment Conclusion

6.1.1 Research implementation of Extensible Enterprise Single Sign-On

We have researched EESSO extensively. We have documented how EESSO operates, how a developer can utilize it for authentication in their own applications, and how a developer can create their own SSO extension.

In addition to this, we have looked at the two SSO extensions that have already been developed, and what type of environment they require.

6.1.2 SSO Extension Prototypes

We iteratively created and improved three different prototypes. Two of these were applications that could use pre-existing SSO extensions, while the third was an attempt at a custom SSO extension using OpenID Connect.

The prototype using the Kerberos extension proved to be easy to implement, and required very little configuration and code to work properly in designated environments.

The custom SSO extension for OpenID Connect was unfortunately unsuccessful, despite spending most of the total development time getting it to work. Getting a Redirect Extension to trigger was the first step, which we never got past due to problems with Universal Links. We were mostly hindered by the lack of documentation on Apple systems like EESSO and Universal Links, and the fact that these issues were too technical for Apple Technical Support. We also decided to make the custom extension the first prototype we made, which resulted in us getting caught up in getting it to work. We either should have started with prototyping the existing extensions, or moved on sooner.

6.1.3 Bonus Priorities

We were unfortunately unable to complete any of the bonus priorities. These tasks were:

- Research similar solutions [to EESSO] for native apps on Android
- Research how to support 2-factor authentication with username and password, and a keycard using NFC technology.

6.2 Research Conclusion

6.2.1 Are there advantages to developing a proprietary SSO extension, or is it better to use an existing maintained extension?

From our work we find that there is no clear answer to this question, but rather it is entirely dependant on what the usage is, and how much reliance there is on compatibility with apps from other providers, as well as what kind of authorization services and Active Directory system the provider uses.

6.2.2 How can Apple Extensions be implemented and used with Xamarin?

We have seen that SSO extensions can be implemented in Xamarin, as all relevant methods and classes from Swift are available through the *AuthenticationServices* package in Xamarin. The code is easily translatable, and both triggering extensions and handling their responses works fine. DIPS will have to create different methods for authentication for Android and iOS, and iOS should still be able to authenticate if an extension for some reason won't trigger.

6.2.3 How can Single Sign-Out be handled on non-personal mobile devices?

The most practical solution so far is to have a timer for the tokens expiration to make sure that devices are automatically logged out after a set time, in case the user forgets to log out themselves when they are done. Later implementations with NFC can more reasonably implement a solution where all the apps are logged out when the user logs out of the device at the end of the shift. Further work in this field should look closer into Shared Device policies on iOS and Android to find a solution not requiring a shared account on the device with the same pin for everyone due to security concerns.

6.3 Further Work

6.3.1 OpenID Connect Redirect Extension

Our attempt at creating a custom Redirect SSO extension for OpenID Connect was not successful, as the issues we were facing were unknown to Apple Technical Support, and would require more time for us to submit a bug report and get any response. The best solution for DIPS is to create their own SSO extension for OpenID Connect, and potentially distribute it to other application vendors that may use their identity providers.

The *Not AppSSO URL* and *SWCErrorDomain error 7* errors (see sections **??** and **??**) are not yet resolved, but have been collectively reported in a bug report to Apple. Hopefully, this bug report will be picked up by the developers of EESSO, eventually resulting in a solution.

Once developers at DIPS has managed to trigger a Redirect extension, development of a functioning prototype can be resumed. Further investigation into how OpenID Connect is best implemented through an extension will need to be done, as there are no proper code examples or documentation showcasing what an SSO extension needs to implement to function properly.

References

[1] EU. *Discover eIDAS*. URL: https://digital-strategy.ec.europa.eu/en/policies/discover-eidas (visited on 27th Apr. 2021).

[2] Microsoft. *Definition of Federated Security*. 2017. URL: https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/federation.

[3] J. CIPRIANI S. ROSENBLATT. *Two-factor authentication: What you need to know (FAQ)*. 2015. URL: https://www.cnet.com/news/two-factor-authentication-what-you-need-to-know-faq/ (visited on 5th Mar. 2021).

[4] IETF. *The OAuth 2.0 Authorization Framework*. 2012. URL: https://tools.ietf.org/html/rfc6749 (visited on 5th Mar. 2021).

[5] OpenID. *Welcome to OpenID Connect*. URL: https://openid.net/connect/ (visited on 27th Apr. 2021).

[6] Microsoft. *Active Directory Federation Services*. URL: https://docs.microsoft.com/en-us/windows-server/identity/active-directory-federation-services (visited on 27th Apr. 2021).

[7] Apple. *Introducing Extensible Enterprise SSO*. 2019. URL: https://developer.apple.com/videos/play/tech-talks/301/.

[8] MacSysAdmin. *Single Sign-On Extensions*. 2019. URL: http://docs.macsysadmin.se/2019/video/Day2Session2.mp4 (visited on 11th May 2021).

[9] Apple. *Extensible Single Sign-On MDM payload settings for Apple devices*. 2020. URL: https://support.apple.com/guide/mdm/extensible-single-sign-on-payload-settings-mdmfd9cdf845/web (visited on 5th May 2021).

[10] Apple. *Intro to Kerberos Single sign-on with Apple devices*. 2020. URL: https://support.apple.com/guide/deployment-reference-macos/intro-to-kerberos-single-sign-on-apdf5b35aad2/web (visited on 5th Mar. 2021).

[11] Apple. *Support Universal links*. 2018. URL: https://developer.apple.com/library/archive/documentation/General/Conceptual/AppSearch/UniversalLinks.html (visited on 7th May 2021).

[12] P. BHANDARI. *An introduction to qualitative research*. 2020. URL: https://www.scribbr.com/methodology/qualitative-research/ (visited on 3rd May 2021).

[13] R. RAFAELI. *Passwords Are Scarily Insecure. Here Are a Few Safer Alternatives.* URL: https://www.entrepreneur.com/article/309054 (visited on 4th May 2021).

[14] Microsoft. *Shared Device mode for iOS Devices.* 2020. URL: https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-ios-shared-devices (visited on 18th May 2021).

[15] Microsoft. *Microsoft Enterprise SSO plug-in for Apple devices.* 2020. URL: https://docs.microsoft.com/en-us/azure/active-directory/develop/apple-sso-plugin (visited on 12th May 2021).

[16] Apple. *Associated Domains Entitlement.* URL: https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_developer_associated-domains (visited on 27th Apr. 2021).

[17] K. KRYCHO. *Apple, Your Documentation Is ...Missing.* 2019. URL: https://v4.chriskrycho.com/2019/apple-your-developer-documentation-is-garbage.html (visited on 4th May 2021).

Appendix

Appendix A - Assignment Text

Arbeidstittel:	SSO til mobile apper fra DIPS

Hensikten med oppgaven:
DIPS ønsker en SSO løsning til våre native mobile apper for klinikere. Extensible Enterprise SSO som kom i iOS13 gjør det mulig å implementere dette for våre apper, slik at brukere trenger bare å logge på 1 gang selv om de hopper mellom apper fra flere leverandører.

Kort beskrivelse av oppgaveforslag:
Vi ønsker en SSO løsning for våre native kliniker apper i DIPS.
Prioritet 1:
- Undersøke hvordan dette implementeres og konfigureres ved hjelp av iOS SSO Enterprise Extensions for apper på iOS sammen med MDM.
- Undersøke hvordan vi kan støtte 2-faktor autentisering med brukernavn+passord OG nøkkelkort vha NFC.
- Lag en prototype

Prioritet 2 (bonus):
- Undersøke lignende løsning for native apper på Android

Oppgaven passer for (kryss av de(t) som passer og skriv evt. en kommentar til oss):	- Bacheloroppgave
Skal oppgaven utføres av bestemte studenter? (der avtalt) Fyll i så fall inn studentenes navn	Eirik Hemstad og Torstein Holmberget Sundfær
Kan oppgavestiller stille arbeidsplass med nødvendig utstyr og programvare:	ja
Oppgaven passer best for, antall studenter:	- 1 - 2
Opplysninger om oppgavestiller	
Er du fra bedrift/virksomhet eller er du student med en egendefinert/selvlaget oppgave?	- Bedrift/virksomhet
Navn på bedrift/organisasjon/student:	DIPS AS
Addresse	Beddingen 10
Postnummer	7014
Poststed	Trondheim
Navn på kontaktperson/veileder:	DIPS ASA
Telefon:	45477035
Epost:	rri@dips.no

Utfyllende kommentarer til hva oppgaven gjelder:
Oppgaven har blitt diskutert direkte med student. Derfor kommer den litt senere enn vanlig frist for innsendte oppgaver fra bedrift.

Appendix B - Vision Document

119-TDAT3001
Implementation of Apple Extensible Enterprise SSO
Vision Document

Version 1.0

Eirik Hemstad & Torstein Holmberget

Revisions

Date	Version	Description	Authored by
23/02/2021	0.1	Created document	Torstein Holmberget
25/02/2021	1.0	Preliminary version	Eirik Hemstad, Torstein Holmberget

Table of Contents

1. Introduction

The increasing use of smartphone applications by personnel at hospitals demands a carefully crafted blend of usability and good security. The solution to this is *Single Sign-On*, which lets a user stay authenticated to access a suite of applications, through one single login. This way, developers can enforce a strict and thorough authentication procedure without sacrificing much of usability.

This document is written with the intention of presenting a new solution for *Single-Sign-On*, to be used by DIPS ASA's mobile applications on iOS. The solution is to be implemented through Apple's *Extensible Enterprise SSO*, which is a new feature introduced in iOS version 13 [1]. The solution will be implemented through an iOS extension, which can be used with any native iOS app. The main purpose of the solution is to replace DIPS' existing, now deprecated by Apple, solution for SSO.

1.1 Abbreviations

ADFS – Active Directory Federation Services
EESSO – Apple's Extensible Enterprise SSO
SSO – Single Sign-On

1.2 Glossary

Xamarin – Cross-Platform mobile development platform by Microsoft
iOS Extension – A library that extends abstract functionality defined in native applications

1.3 References

2. Summary of problem and product

2.1 Problem Summary

Problem with	That the current system does not function properly since Apple has removed support for the previous solution in iOS version 13
impacts	DIPS and their customers.
As a result of this	An SSO solution cannot be implemented without complications.
A successful solution will	Give DIPS mobile applications better compatibility with other applications.

2.2 Product Summary

For	DIPS
which	Need a system that can accommodate Single Sign-On in DIPS native mobile applications for simplicity and better security.
Product named	Is a functional prototype in the form of an iOS extension
that	Can guide developers at DIPS on how to use EESSO as an SSO solution, and in best case provide a library that can be integrated in their applications
Which unlike	DIPS' in-house SSO solution
Our product has	A stronger technological foothold, along with compatibility with other applications targeting the same extension

3. Overview of stakeholders and users

3.1 Summary Stakeholders

Name	Description	Role during development
Tore Mørkved	Product owner, mobile team in DIPS ASA	Giver of assignment and formal contact person at DIPS
DIPS ASA	The company in need of the solution.	Source of domain-specific information that's vital to the implementation of the solution
Eirik & Torstein	Actors of the assignment	Develops and researches what's necessary to complete the solution
NTNU	The university	Guides the students through the process, and gives a final evaluation of the solution

3.2 User Summary

Name	Description	Role during development	Represented by
DIPS	Developers and maintainers of the applications meant to use the solution	Technical insight and documentation of the existing mobile applications and SSO solution, along with providing feedback of the usability of the prototype	Tore Mørkved
Nurses & Doctors	The users of the applications that targets the SSO extension		

3.3 User Environment

The system consists of several mobile applications from DIPS that are in use at hospitals and clinics. Today's user environment is iOS and Android devices in use by health personnel to access patient journals and test results as well as making notes on patients quickly without having to access a computer terminal.

This solution is for iOS, as there is currently a working solution for Android today. Our SSO-solution should be able to provide an opportunity to log on to all the systems the user needs with a single login and keep these logged in for a given timeframe according to client policy.
For units that are in the sole possession of one user this can be solved by biometric login options on the device so that only the owner of the device can access the device and the patient records on it.
Units that are shared between users need another solution, like connecting it to the user logging on to the device at the start of their shift and logging out before handoff to ensure data safety.

Each hospital setting has its own internal network, with its own set of authentication / authorization servers and policies enforced by these. The solution must use this existing infrastructure to communicate with the authentication services on the premises, and properly communicate authorization data to the application.

3.4 Summary of user's needs

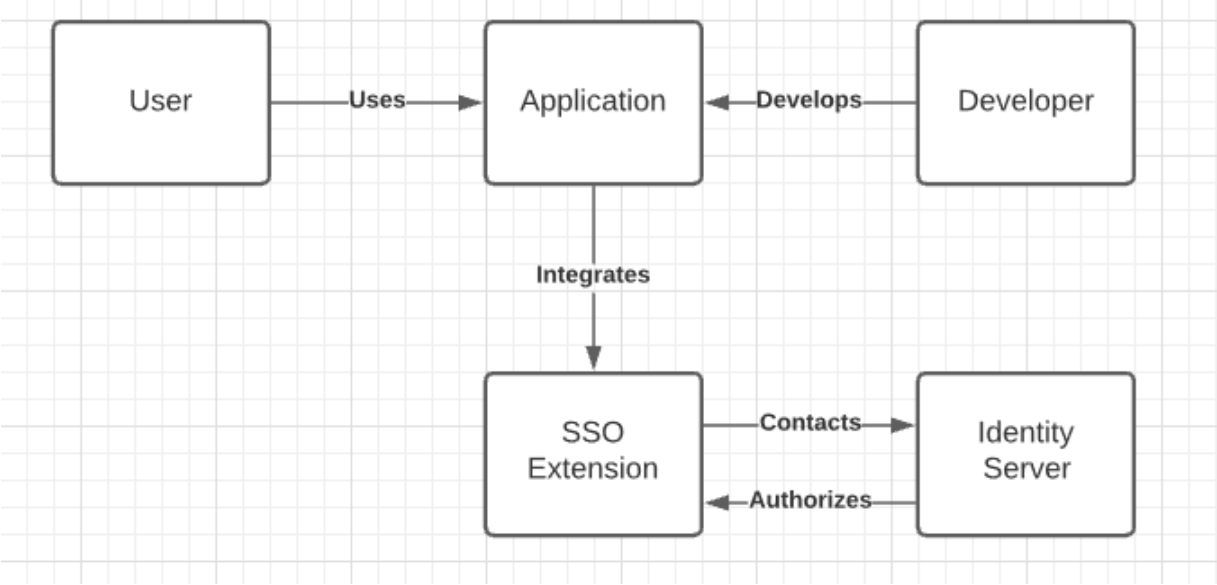
Needs	Priority	Affects	Today's solution	Proposed solution
Implementation of Apple Enterprise SSO Extension	High	DIPS	None	Make a prototype demonstrating the implementation of Extensible Enterprise SSO
Keep authenticated session between applications	High	DIPS	No longer works on iOS	Prototype multiple apps that implement the same SSO Extension
Login page bundled with library	High	DIPS	Unknown	Login page is integrated into library
User is logged out at the end of their shift	High	DIPS	Unknown	Find a way to ensure that users are logged out of their session when no longer using their device
Integration with Active Directory Federation Services	Medium	Nurses & Doctors	Yes	EESSO supports integration with ADFS
Ease of use and portability in using the extension on existing apps	Medium	DIPS	Using a library that's developed in-house	New library that implements EESSO
2-factor authentication using smart cards	Low	Nurses & Doctors	No such solution	User must authenticate with username/password, along with their key card
Compatibility with Xamarin	Low	DIPS	Developed in C# and compatible with Xamarin	Link to Swift library that implements the extension

3.5 Alternatives to our product

As of today, there are no official or better alternatives for implementing an SSO solution on iOS. EESSO is an official feature developed and maintained by Apple, and is therefore the most solid solution. The only alternative is DIPS' existing in-house solution, which is no longer functional after iOS version 13.

4. Product Summary

4.1 The product's role in the user environment



This figure shows how the SSO solution is meant to be integrated into the existing environment, that of the user, application and developer.

4.2 Conditions and dependencies

The most important dependency for this solution to work is Apples continued support of the EESSO implementation in future iOS updates. The users' network configuration must also be compatible with the authentication methods included in EESSO.

Conditions for the product is that the solution being implemented meets the rigorous safety demands of dealing with confidential patient records and test data, any major changes to either the backbone of the product or the safety demands of the clients will alter the products viability.

5. Product features

- Implements EESSO
- Authentication with username and password
- Keep user authenticated on all integrated apps
- 2-factor authorization with key card
- Works with Active Directory Federation Services

Appendix C - Apple Bug Report

EH **You**
May 20, 2021 at 2:31 PM – FB9112396

ASSIGNEE

Problem triggering a custom Redirect Extension with Extensible Enterprise SSO

Recent Similar Reports: None
Resolution: Open

Basic information

Please provide a descriptive title for your feedback:
Problem triggering a custom Redirect Extension with Extensible Enterprise SSO

Which area are you seeing an issue with?
Something else not on this list

What type of feedback are you reporting?
Incorrect/Unexpected Behavior

Description

Please describe the issue:
We are trying to trigger a custom SSO Redirect Extension on iOS, but are seemingly having issues with validating associated domains and routing the traffic to the extension. We were advised to create a bug report after a long dialogue with technical support.
There are two errors we are receiving.
First, we are trying to use swcutil to download the apple app site association file from our identity server. We are using the command "swcutil -d <authserver>" command. We are getting the error "SWCErrorDomain error 7" for unknown reasons. We can easily download the file through cURL. See steps to reproduce for the server setup we have.
Second, we are getting the error "Not AppSSO URL" when we are trying to trigger the Redirect extension from our application. As stated below, we are going towards the "https://<authserver>/dips.oauth/identity/connect/authorize" endpoint, which is listed in the MDM payload.
We have included all relevant files and logs below.

Please list the steps you took to reproduce the issue:
We have a server running OIDC which we want to use as identity provider. The authority endpoint is located at "https://<authserver>/dips.oauth/identity/connect/authorize". The server has a valid SSL certificate, but it is only available through our internal network. The server has an Apple App Site Association file located at "https://<authserver>/.well-known/apple-app-site-association". We are able to fetch this file with cURL.
We are using Microsoft Intune as MDM provider. Our test device is an iPhone 6 Plus running iOS 14.5, and is managed through MDM.
We have two applications. Both applications has the associated domain "authsrv:<authserver>?mode=managed" listed in its entitlements file. Alternate mode is used because the server is not available through the internet, so your CDN can't access it. One of the applications are using sample code from "Introducing Extensible Enterprise SSO" (from WWDC 2020) to try and trigger the extension. We have provided the authority endpoint of the <authserver> as the identity provider to ASAuthorizationSingleSignOnProvider. The other application contains a target with the default Authentication Services template. There is no code really written in this project. Both of these application's bundle ID's are listed in the apple app site association file, under the "authsrv" service type. The MDM has an SSO Extension profile of type Redirect, where we have listed the bundle identifier for the extension (not the host app of the extension), and both the authority and token endpoints. Both application's IPA files are uploaded to Intune.

What did you expect to happen?
For the extension to be triggered, triggering a simple alert that would indicate successful trigger.

What actually happened?
The extension is not being triggered, and the didCompleteWithError delegate is being called, with error com.apple.AuthenticationServices.AuthorizationError error 1000. The "Not AppSSO URL" error is shown in console logs.

File Uploads

SSOPrototypeC.entitlements
314 Bytes May 20, 2021 at 2:30 PM

sysdiagnose_2021.05.20_14-12-14+0200_iPhone-OS_iPhone_18D70.tar.gz
186.99 MB May 20, 2021 at 2:30 PM

AuthManager.swift
2.28 KB May 20, 2021 at 2:30 PM

apple-app-site-association
221 Bytes May 20, 2021 at 2:30 PM

swcutil_d.png
18.43 KB May 20, 2021 at 2:30 PM

SSOExtension.entitlements
366 Bytes May 20, 2021 at 2:30 PM

Console.txt
150.13 KB May 20, 2021 at 2:30 PM

mdm_payload.png
14.72 KB May 20, 2021 at 2:30 PM

Appendix D - Interview

Spørsmål til erfaringsintervju

TDAT3001-119

Parter

Intervjueholdere v/ studenter, NTNU

Eirik Hemstad
Torstein Holmberget

Intervjuobjekt v/ Sykehuspartner

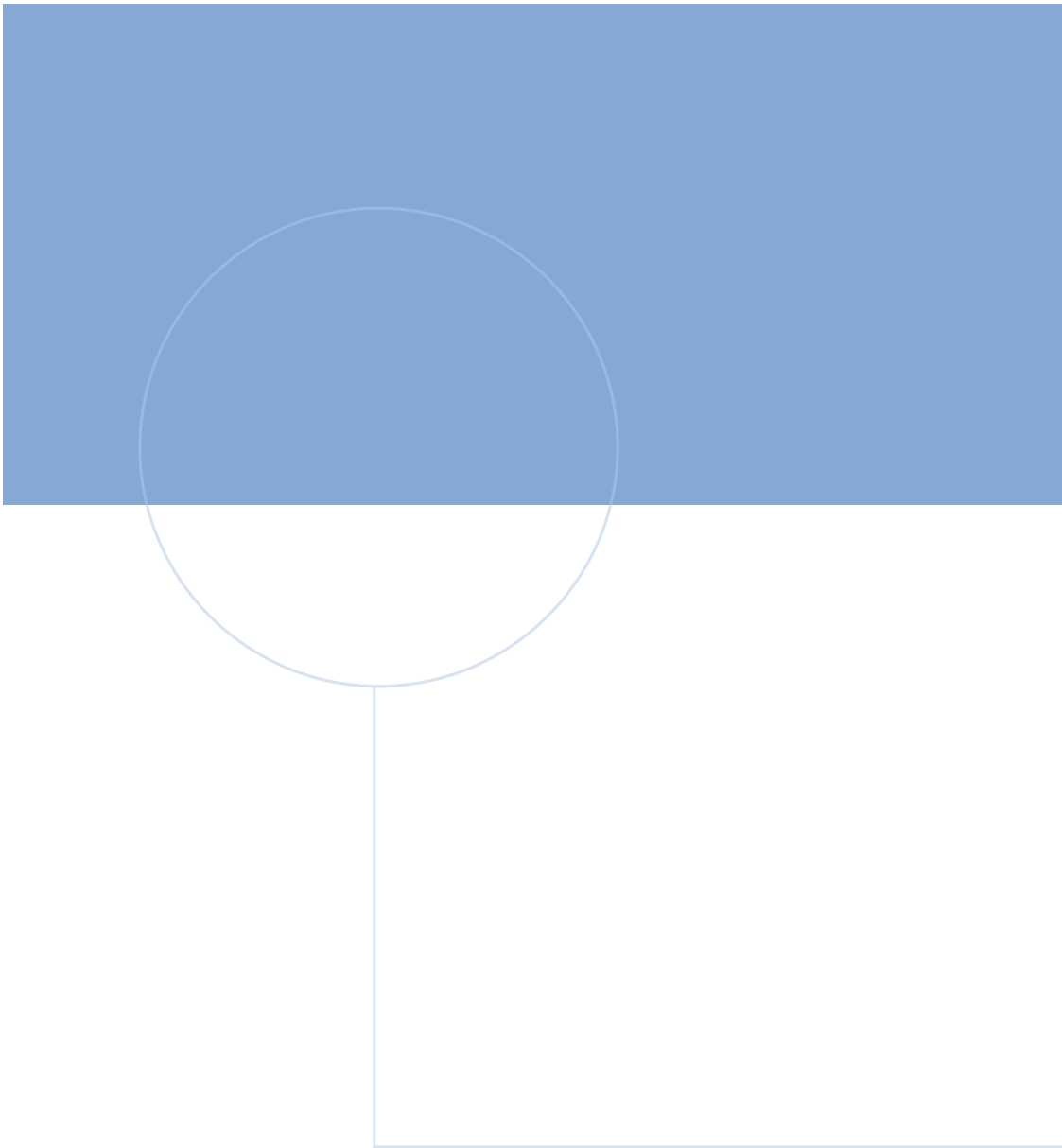
Kjell Arild Sandvik, systemarkitekt

Alle parter samtykket til opptak av intervju

Spørsmålsliste

- Kan du fortelle kort om prosjektet som har blitt utført?
    - Prototype prosjekt kjørt både med Kerberos versjon og med egendesignet versjon mot PingFederate
- Hva slags løsning for MDM har inngått i utviklingen av prototypen?
    - MDM inngår gjerne under Enterprise Mobility Management (EMM) eller Unified Endpoint Management (UEM). MDM inngår ofte som komponent i disse løsningene
    - Det er forutsett at apper som er en del av SSO-miljøet rulles ut med en MDM-løsning og at de har en spesifikk payload
    - De mest utbredte løsningene er VMWarew (Workspace One), IBM (MaaS360), Microsoft (Intune) og Citrix (Unified Endpoint Manager)
- Hvordan bruker dere PingFederate i deres løsning?
    - Egen versjon med PingFederate, må lage egen hostapp, starter app eller nettside, dette kommuniserer til host appen at du trenger autentisering, som henter denne fra pingfederate som sender dette via BuyPass
    - BuyPass brukes for at man må ha en godkjent provider for dette, ikke noe man kan gjøre selv. Dette kontrollerer om token kan sendes ut og gir den til hostappen. Neste app får da bare direkte fra hostappen i en snarvei
    -
- Hva slags MDM-konfigurasjon blir spesielt brukt i produksjon?
    - Ikke interessant
- Hvilken type SSO extension utvikler dere mot PingFederate?
    - Redirect Extension
- Hvilken teknologi implementeres deres SSO-løsning på?
    - Native iOS applikasjoner programmert med Swift
- Hva er fordeler og ulemper med egenutviklet løsning fremfor Kerberos?
    - Kerberos kan brukes rett ut av boksen hvis man bruker det

    - Kerberos veldig enkelt å få til å fungere, par linjer kode for å implementere så bare fungerer det.
    - I dette tilfellet behøvdes støtte mot PingFederate, som det ikke finnes noen utbredt løsning for. Derfor er behovet for å lage en egen løsning.
- Hvilke autentiseringsmetoder blir benyttet i deres prototype?
    - Tar utgangspunkt i eIDAS, regelverk for innloggingssystemer innført i norsk lov.
    - Bruker OpenID COnnect, som er bedre enn OAuth2 pga. flere felter og muligheter, Har AD autentisering i back-end, slik at den kan hente en AD-autentisering via PingFederate.
    - Større problem med Single-Sign-Out, må finne en løsning der en app kan logge ut uten å automatisk få ny token neste gang den åpnes uten å må logge ut alle apper ved å drepe token.
      Får problemer med apper som absolutt ikke skal logges ut i løpet av et skift, som for eksempel apper som varsler om pasienttilstand
      evt. Løses med egen innlogging på disse appene slik at de ikke blir påvirket av de andre appene.
    -
- Har dere noen konkret plan for implementering av 2FA-autentisering med adgangskort?
    - 2FA med NFC kort og pin-kode. FIDO 2
    - IOS 14.2+ for å få med implementeringen av FIDO2, med protokoll ctap2
    - NFC ved adgangskortet blir lest av operativsystemet som videresender informasjonen til applikasjonen, dette blir gjort ved at appen sender en forespørsel til operativsystemet om å hente data fra NFC kortet, som deretter kan låses opp med PIN koden
      Må undersøke hvordan nøyaktig appen gjør denne forespørselen, om dette er en nativ metode i xcode eller noe annet.
    - Utfordringer med delte devicer, må ha eget system for dette med egen systemkonto
    - Personlige devicer er noe annet, siden Apple sine devicer er godkjent som FIDO2 device av FIDO alliance, noe som gjør at man kan lagre sikkerhetsnøkler på iphonen, så de som bruker personlig device slipper kort som faktor 2, slik at faktor 1 blir telefonen, og faktor 2 er biometrisk innlogging.
    - Kombinert kort med smartkortfunksjon for å bruke smartkort id'n til å verifisere brukeren ved inaktiv app.
    - Yubikey som har flere sikkerhetsløsninger som kan brukes for å hindre
- Andre bemerkelser
    - Hvilket identitetssystem bruker DIPS i sitt system? Viktig for utviklingen
    - DIPS sitt system må støtte federert sikkerhet i backend for å sikre tokens mot felles system. Så lenge man bruker den standariserte xcode innloggingsmetoden så trenger ikke appen gjøre noe selv, det blir automatisk tatt hånd om i bakgrunnen.
    - Vanskeligere å implementere med Xamarin, må implementere native kode, og skrive den native koden inn i Xamarin, dette fører til mye overhead. Må ha et mellomledd mellom appen og en native extension, bruker da bare Xamarin som en proxy mellom. Godt valg med en større app der innloggingen er en liten del, men når innloggingen er hovedsaken så blir det lettere å lage og vedlikeholde i ren native kode.
      AFDS som er Microsoft sitt alternativ kan brukes som alternativ til PingFederate ved å bruke Azure AD som autentiseringsmekanisme siden den er et federert system.



NTNU

Kunnskap for en bedre verden