Master's Thesis

Investigating Open Source Alternatives for an Electronic Identity System

Per Ahlbom
Martin Richter

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2016.

Investigating Open Source Alternatives for an Electronic Identity System

Per Ahlbom & Martin Richter
ada10pah@student.lu.se, adi10mri@student.lu.se

Department of Electrical and Information Technology
Lund University

Advisor: Martin Hell

May 18, 2016

Printed in Sweden
E-huset, Lund, 2016

Abstract

Electronic IDs enable people, companies and organizations to sign documents and authenticate online. Considering the potential losses, the security in an eID system is crucial. The eID system in Sweden today, BankID, is closed source and uses proprietary standards. In our thesis we have investigated if open standard and open source can be an alternative. First we reviewed the research about security in open source contra closed source. The research was not conclusive and one can not conclude that either of them provide more security. We show that using open source is a possibility, by implementing a proof-of-concept eID solution utilizing the framework SAML 2.0 and the protocol FIDO U2F. They are both open standards and there are several open implementations of SAML 2.0 and libraries for FIDO U2F to use. To verify that FIDO is a suitable protocol we looked at other possible two factor authentication solutions, such as OATH-HOTP and OATH-TOTP. The thesis also reviews some potential attacks against our system and we discuss how to mitigate them.

Acknowledgements

We would like to thank our supervisor Martin Hell for guiding us with content and giving us new ideas when we had none. We would also like to thank our reviewers for going through the thesis and helping us improve it. Last but not least, we thank Ludivine for her help improving the language.

Table of Contents

1 Introduction 1
- 1.1 Purpose and goals . . . 1
- 1.2 Delimitations . . . 2
- 1.3 Outline . . . 2

2 Background 3
- 2.1 Electronic identity . . . 3
- 2.2 Federated Identity . . . 3
- 2.3 Linus' law vs. security through obscurity . . . 4
- 2.4 Two factor authentication . . . 6
- 2.5 SAML 2.0 . . . 7
- 2.6 FIDO . . . 9
- 2.7 OATH-HOTP and OATH-TOTP . . . 14

3 Electronic ID systems 19
- 3.1 BankID . . . 19
- 3.2 Swedish eID federation . . . 20

4 Our implementation 23
- 4.1 Infrastructure . . . 23
- 4.2 Distribution and issuing . . . 26
- 4.3 Authentication . . . 28

5 Attacks on the implementation 31
- 5.1 Phishing . . . 31
- 5.2 Man-in-the-browser . . . 32
- 5.3 Denial of service . . . 35
- 5.4 SQL injection . . . 37

6 Analysis 41
- 6.1 Open Source secure enough? . . . 41
- 6.2 FIDO . . . 42
- 6.3 OATH analysis . . . 44
- 6.4 Comparison FIDO & OATH . . . 45

6.5 SAML 2.0 analysis 47

7 Discussion **49**
7.1 Goal: evaluate the security of open source protocols 49
7.2 Goal: implement a proof-of-concept 49
7.3 Goal: evaluate security of our implementation 50
7.4 Future work 51

8 Conclusion **53**

List of Figures

1.1 Overview of the new federation 2

2.1 Overview of SAML Web Browser SSO POST 8
2.2 Overview of FIDO in the federation. Used with permission from FIDO Alliance. 10
2.3 Overview of the HOTP algorithm 16
2.4 Overview of the OATH–HOTP proposed protocol 17

3.1 Federation infrastructure 21

4.1 Our infrastructure 24
4.2 FIDO bridge 25
4.3 Registration process 27
4.4 Authentication process 29

Acronyms and Abbreviations

2FA Two Factor Authentication

CSRF Cross Site Request Forgery

eID Electronic Identity

FIDO Fast Identity Online

HOTP HMAC-Based One-Time Password

IdP Identity Provider

LDAP Lightweight Directory Access Protocol

MITB Man-in-the-Browser

MITM Man-in-the-Middle

OATH Initiative for Open Authentication

OTP One-Time Password

SP Service Provider

SSL Secure Sockets Layer

SSO Single Sign On

TLS Transport Layer Security

TOTP Time-Based One-Time Password

U2F Universal 2nd Factor

UAF Universal Authentication Framework

WAR Web application ARchive

DoS Denial of service

DDoS Distributed denial of service

CDN Content delivery network

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

Chapter 1
Introduction

More and more services that require users to prove their true identity and to sign contracts are transitioning to the Internet. The different types of services include banking, tax and legal services. As these services involve legal contracts, the authenticity of the electronic identity (eID) is crucial. The services that require authentication need to be certain that the identity service is secure and cannot be tampered with. It is also important that no user's eID falls into the hands of a third party. As of today in Sweden, BankID almost has the monopoly of the eID service. BankID runs on proprietary code which can be a problem because one can not look into how the security features are implemented. The Swedish eID board is launching a new eID federation where this problem will be partly addressed. The communication between the different parties in the federation will be based on the open framework SAML 2.0. Compared to traditional authentication, once a user joins a federation, it can authenticate towards several Service Providers without creating several accounts. This thesis evaluates different open source protocols for authentication to be used in the new federation. By using open source for the authentication, the whole system would be open source. The protocols we will investigate for authentication are FIDO U2F, FIDO UAF, OATH-HOTP and OATH-TOTP. Furthermore, we will implement a proof-of-concept application using FIDO for the authentication and Shibboleth for the implementation of the SAML 2.0 protocol. Figure 1.1 shows a scheme of the different parties in the federation and how authentication is done. When a user wants to access a resource at the Service Providers, it authenticates to the Identity Provider that communicates with the Service Provider and confirms that the user is authenticated. This thesis will focus on the authentication at the Identity Provider when using open source protocols.

1.1 Purpose and goals

- Evaluate the security of open source protocols for potential use in the new Swedish eID federation.
- Implement proof-of-concept application using Shibboleth and FIDO.
- Evaluate the security of our implementation.

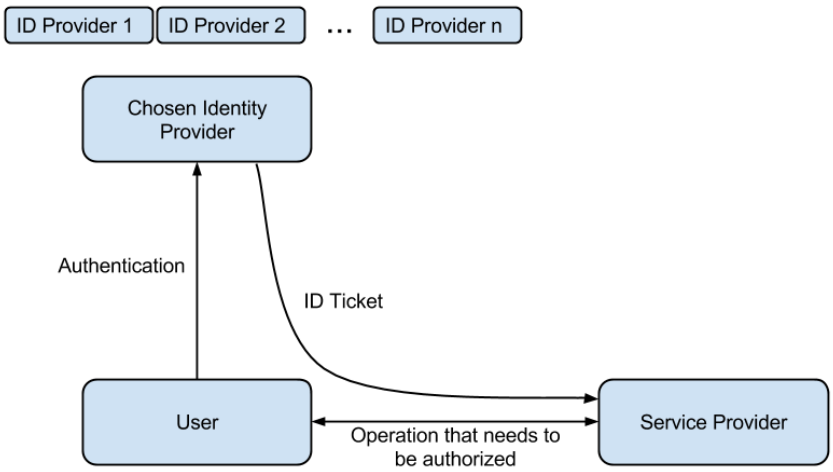

Figure 1.1: Overview of the new federation

1.2 Delimitations

This thesis will focus on investigating the security of a few selected open source protocols as well as investigating their suitability to use in the new Swedish eID federation. The infrastructure of the federation will be reviewed briefly but the focus of this thesis is the protocols/techniques concerning authentication. The analysis will focus on a few critical attacks and how our proof-of-concept and chosen protocols handles these attacks rather than a brief analysis of several attacks.

1.3 Outline

There are 8 chapters in this thesis, which are:

1. Introduction: contains goals and a brief background.
2. Background: introduces and defines the different protocols, frameworks and concepts relevant for the thesis.
3. Electronic ID systems: reviews existing eID solutions and the proposed Swedish eID federation.
4. Our implementation: presents our solution and how it integrates with other software used.
5. Attacks on the implementation: evaluates different attacks on our system and how we protect against them.
6. Analysis: goes through the results and draws conclusion about open source and the different protocols.
7. Discussion: reviews our goals and how they were met.
8. Conclusion: a short summary of the thesis and the results.

Chapter 2
Background

This chapter will describe different techniques, concepts and protocols that are relevant for this thesis. Concepts and definitions, such as eID, federated identity and two factor authentication are explained. Previous research on the topic of open source security is reviewed and summarized in the section "Linus' law". The frameworks/protocols SAML, FIDO & OATH are described in this chapter and later investigated from a security perspective in the analysis.

2.1 Electronic identity

The information society of the European commission [1] defines an electronic identity as "a means for people to prove electronically that they are who they say they are and thus gain access to services. The identity allows an entity (citizen, business, administration) to be distinguished from any other". In this report we use the term electronic identity or eID for the concept of authenticating a real person or company online.

2.2 Federated Identity

Federated Identity is defined by Madsen [2] as "linking a person's electronic identity and attributes, stored across multiple distinct identity management systems". Shibboleth uses a similar definition and states that [3] "when a group of Identity and Service Providers agree to work together, this group is called a federation". Federated identity management involves having a common set of practices for a group of Service Providers (SPs) and Identity Providers (IdPs). These practices include trusting each other, technical frameworks, rules, regulations et cetera. For a SP, using a federation means that they can outsource the identity management to an IdP.

This is related to Single Sign On (SSO), in which a single user's authentication or token is trusted by different services or organizations. SSO is a subset of federated identity management, as it is an implementation of a federated identity system. Often, there is a central authority that can admit new SPs or IdPs into the federation and also exclude them. Later on, we will explain how the Swedish eID board has chosen to set up their eID federation. Using a federation or SSO solution

does not only make it easier for the user but can also improve security. A study by Herley [4] showed that having multiple SPs with isolated identity management leads to users choosing weak and/or reusing passwords. If a user has one or few IdPs instead, they are more prone to choose stronger passwords.

There are several frameworks that can be used to implement identity federations, for example SAML, Liberty Identity Federation Framework and OAuth. SAML 2.0 will be explained more in detail in Section 2.5. SSO is an implementation of federated identity. SSO is usually deployed within an organization, while an identity federation typically spans over several organizations. In both cases the technology is the same. Some examples of SSO are Kerberos, Microsoft Outlook account and Facebook connect. Shibboleth is a SSO framework that implements the SAML 2.0 framework. This will be explained further in this chapter and the security will be reviewed in the analysis.

2.3 Linus' law vs. security through obscurity

In this report we use the definition of a vulnerability as "a bug that can be used by an attacker to gain access to the system". The definition comes from the "Common Vulnerability and Exposure" (CVE) [5], which is "a dictionary of publicly known information security vulnerabilities and exposures", provided by the MITRE Corporation. The National Institute of Standards and Technology (NIST) adapts the CVE standard in the National Vulnerability Database [6]. NIST is an American governmental organization under the Department of Commerce. The not-for-profit MITRE also provides the Common Weakness Enumeration that we use for defining attacks. Most research on software security use the CVE standard as well as the same definition of a vulnerability.

This section summarizes research that has been done on the topic of open source security. It covers empirical studies of vulnerabilities in open source software, compared to vulnerabilities in closed source software. Some arguments for and against the security open source are raised and later discussed in the analysis.

Linus' law states that [7] "given enough eyeballs, all bugs are shallow". The law is named after Linus Torvalds, the creator of the Linux kernel. This law advocates the use of open source software. On the other hand, another argument is that "keeping the source code closed prevents the attacker from having easy access to information that may be helpful to successfully launch an attack" [8]. Hiding the source code and having a secret design is commonly referred to as security through obscurity. Opinions on open source are often highly biased. Companies using proprietary code claim that hiding the code is important to prevent attackers. Meanwhile, open source advocates estimate that bugs are patched much faster when several people can review the code [9]. This section will go through research on the topic of open source security. In the analysis we will discuss how this affects our proposed system and why it is an important aspect.

In order to find out if open source systems are more or less secure than closed source systems, one has to define what security is. This is very difficult to define, as one has to consider the amount of vulnerabilities in a software, their severity, as well as how fast they are patched. Schryen & Kudo [10] propose a method on how to measure security in open and closed source software. They propose weighting vulnerabilities. The weighting takes into account the number of vulnerabilities in a software, how easily they are exploited, the severity of the vulnerabilities, time between detection and deletion of vulnerability and how many bugs are unpatched.

In an empirical study, Schryen [11] compares published vulnerabilities in 8 open source and 9 closed source software packages, using the metrics from previous research. They analyzed e-mail clients, web browsers, web servers, office software, operating systems and database management systems. The results show that the mean time between vulnerability disclosures, i.e. how often a bug is found, was lower for the open source e-mail clients & web browsers. For the other software systems, there was no statistically significant difference in mean time between vulnerability disclosure. A low mean time between vulnerability disclosures indicates that there are more bugs in the system. In another study by Schryen [12], he makes a comprehensive study on the patching behaviour for open and closed source software. This study takes into account the severity of the vulnerabilities using the common vulnerability scoring system (CVSS) [13], as well as whether if a patch is provided. The study does not show any significant statistical difference in overall patching behaviour for open source and closed source. However, it does indicate that closed source vendors prioritize severe vulnerabilities higher. The study also indicates that there is a greater spread for closed source code vendors, which means that the best as well as the worst patching behaviour was found among closed source software.

In a large scale exploratory analysis of software vulnerability life cycles [14], Shahzad, Shafiq & Lio analyze vulnerability data for 8 vendors: Microsoft, Apple, Sun Oracle, Linux, Mozilla, Redhat and Google. The methodology for this study and how data was chosen is very similar to the research of Schryen, but it uses a larger set of data. The result of the research also differs as Shahzad et.al. conclude that open source vendors are slower at providing patches. It also criticises Schryen's study for using a smaller set of data and not considering the time between disclosure and vendor providing a patch. Schryen argues that the time between public disclosure and when a patch is provided is not reliable. This is because the time gap between the actual disclosure of a vulnerability (on the web or in mailing lists, for example) and its consideration in the "Assigned" phase of the MITRE CVE workflow is unreliable.

Hoepman & Jacobs [8] argue in a less statistical study that when a bug is found on closed source code, the attacker will manage to exploit the vulnerability longer, as patching is done faster for open source. They also point out that having the source code open will result in a higher exposure initially, but that over time the security will increase, as more people are able to find vulnerabilities and either report or patch them.

Another study by Schryen [15] does not prove that the security is significantly better or worse in open source software or closed source software. He argues that it is rather the policy of the vendor that will influence the software's security. This theory is also supported by Anderson [9] who has looked at the asymmetry between open source and closed source software. For example, if a new type of theoretic attack is found, it is easy to scan open source code to see if it is vulnerable or not. It is more difficult to check whether or not the attack applies to a closed source software, which puts more responsibility on the vendor. Anderson's paper concludes that there is no security difference in theory. He states that "the attack and the defence are helped equally. Whether systems are open or closed makes no difference in the long run."

2.4 Two factor authentication

Password based authentication is the most common authentication method online. There are several possible attacks to password based systems, such as brute-force attacks, dictionary attacks, as well as offline attacks such as rainbow attacks. A study by Burnett [16] shows how poorly users choose their passwords. For example, "qwerty" and "123456" were shown to be among the most common passwords. Florencio & Herley [17] studied the password habits of half a million users. They looked at how many passwords an average user has, how often they type them and how often a user re-uses a password on different sites. The results showed that users often use the same password for multiple accounts. Having the same password for different accounts leads to a single point of failure, as one account being compromised would allow an attacker to gain access to all of a user's accounts. It is possible to set a minimum length and require a mix of characters, which will defend against the basic attacks. However, this will not protect against phishing or keylogging attacks. Having a password based authentication system, requires that the service properly stores and manages the passwords as well as the user.

Considering the problems with password based authentication systems, using only a password does not provide adequate security for the purpose of electronic ID. An alternative to username password authentication is two factor authentication (2FA). This is an authentication mechanism where the user is required to provide two credentials to prove their identity. These credentials can be of three different kinds: knowledge factor (what a user knows), ownership factor (what a user owns) and inherence factor (what a user is) [18]. An example of knowledge factor is a password. An example of ownership factor is a USB key or a credit card and an example of inherence factor is biometric information such as a fingerprint. By using two credentials, the potential security threat of a weak password diminishes [19]. For an eID system, the security is more important than for example an account on a forum. If a forum account is hacked, an attacker can post as the user. However, if a person's eID is hacked it can lead to devastating financial losses and legal problems. All eID systems (that we found) use two factor authentication. In the report "Authentication in an internet banking environment", the Federal

Financial Institutions Examination Council considers single-factor authentication to not provide adequate security for financial transaction, and recommends using multi-factor authentication instead [20]. Since 2FA provides more security than a simple username password system, using 2FA is a requirement in our IdP solution.

2.5 SAML 2.0

Security Assertion Markup Language (SAML) is a framework for sending authentication and authorization data between parties. It is based on XML to represent the information. SAML 2.0 is the third SAML specification. It was developed by Organization for the Advancement of Structured Information Standard (OASIS). The Framework was developed to provide a SSO solution where users only need to authenticate against one IdP in order to gain access to different SPs.

The authentication can be initiated by an IdP or a SP. The IdP starts the authentication process when the user is redirected from the IdP to the SP. An example of that would be a hotel website redirecting the user to a taxi rental service. In the redirect request, an authorization token is added to the request. When the user wants to access a protected resource, the SP redirects the user to an IdP. The user authenticates themselves at the IdP which generates an access token. It is used as a credential at the SP to get access. After the authentication is finished, the user is redirected to the SP where they supply the access token [21].

Information about the user from the IdP is sent as an assertion. An assertion is a package with security consideration from the IdP. It can contain information of when the user was authenticated, attributes about the user and what resources they can access [22].

The SAML 2.0 framework is large and covers many user scenarios. To address this, SAML 2.0 uses profiles. A profile specifies how the communication between the IdP, user and SP works. Other organisations can set up their own profiles. The new Swedish eID federation uses the Kantara Initiative eGovernment Implementation Profile of SAML 2.0. This profile is based on the Web Browser SSO Profile, which is defined in the SAML 2.0 specification. This profile specifies how the authentication is initiated and how the messages are sent between the SP and the IdP. In this profile, the user's browser works as a relay between the SP and the IdP. TLS is the recommended transport protocol between the different parties. The authentication process is initiated by the SP, which is illustrated in Figure 2.1. UA in Figure 2.1 is an automated script running in the browser which is responsible for redirecting. The script is called when the user wants to access a protected resource, as illustrated in step 1 in Figure 2.1. The SP sends a redirect request to the user, which redirects the user to the IdP, shown as step 2 and 3 in Figure 2.1. To identify the connection the, SP stores a relaystate of authentication request. When the response is returned, the relaystate is used to redirect the user to the right resource. This relaystate is also sent with the request. When the IdP receives the authentication request, it initiates the authentication process. The

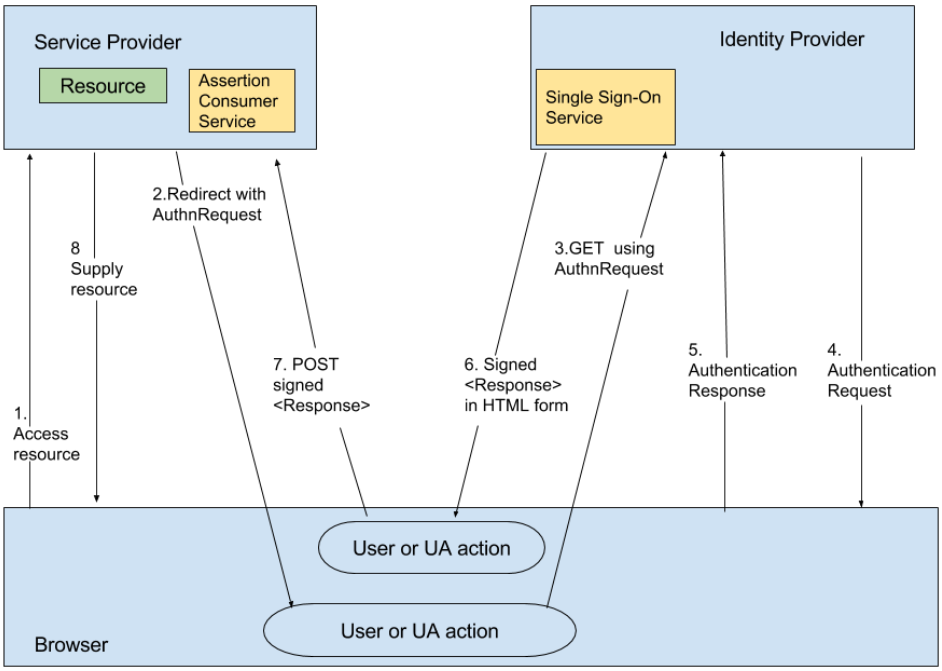

Figure 2.1: Overview of SAML Web Browser SSO POST

user authenticates against the IdP, as shown in steps 4 and 5 in Figure 2.1. If the authentication was a success, the IdP builds the response which is signed with a private key. The IdP builds a HTML form where the response is added as a B64 encoded string, as illustrated in step 6 in Figure 2.1. The form also contains the relaystate from the SP. After the form is created it is returned to browser. The browser can call a JavaScript function, which posts the form to the SP as shown in step 7 in Figure 2.1. When the SP receives the posted form, it checks the response signature and the valid time frame. If these attributes are valid then the response is valid. The SP then restores the state which the relaystate was bound to, and the user is redirected to the requested resource, as demonstrated in the last step in Figure 2.1 [21].

2.5.1 Security in SAML 2.0 Web Browser SSO Profile

To provide assurance that the responses from the IdP are valid, SAML 2.0 needs to ensure confidentiality, integrity and authenticity. This is done with TLS and digital signatures. The SAML 2.0 security consideration states some different attack scenarios [23]:

- Stolen Assertion
- Forged Assertion
- Replay

- Man in the middle

If an assertion is stolen the attacker could impersonate the user and gain access. To protect from this attack it is required that confidentially is provided. This is done with the use of TLS. The assertion is only valid a certain amount time, which should be as short as possible. In a forged assertion attack, the attacker forges or alters the assertion. To avoid this the assertion is signed. If the signature is false then the assertion is invalid. A replay attack is avoided using the same technique as for the stolen assertion. In a man in the middle attack, the attacker impersonates the user at a new SP using a stolen assertion. This attack is mitigated by the SP, which checks that the request for the assertion was made by itself. This information is stored in the assertion which is signed by the IdP.

2.5.2 Shibboleth

Shibboleth is a consortium that develops products made for SAML framework deployments. Their products include SAML Discovery Service, SAML SP, SAML IdP, SAML Metadata aggregator for direct deployment, and low level SAML libraries in C++ & Java. We chose to use the SP and IdP from Shibboleth as they are recommended by the Swedish E-identification Board because of the complexity of the SAML protocols.

Shibboleth SP integrates with popular web servers such as Apache and Microsoft IIS. The service is added to the web server as an add-on. It then provides authentication to the websites and web resources deployed on the web server. This means that the websites can be written in any language. If a web resource wants to utilize the SP, the web server is configured to perform authentication when that resource is accessed. The IdP is written in Java and deployed in a Java servlet container. It supports several password authentication protocols such as LDAP, Kerberos and servlet container. You can also add your own authentication protocols written in Java. The officially supported servlet containers are Apache Tomcat 8 and the Jetty version 9.2 or 9.3. The discovery services can either be installed with the SP or as a standalone system. The discovery services enable the user to choose what IdP they use for identification [3].

2.6 FIDO

FIDO Alliance is a non-profit organization. It is an industry consortium with members such as Google, PayPal, Microsoft, VISA, American Express and Intel. It was founded in order to "address the lack of interoperability among strong authentication devices as well as the problems users face with creating and remembering multiple usernames and passwords" [24]. FIDO is meant to simplify and strengthen authentication online. As of January 2016, there are two published protocols: Universal Authentication Framework (UAF) and Universal 2^{nd} Factor (U2F). There is also a third protocol that is submitted to W3C to become a standard, which is called FIDO 2.0 WEB API.

Figure 2.2 shows where FIDO is used in the overall architecture of an identity system. The federated relying party websites are the SPs (for example banks). The FIDO authenticators can either be embedded in the user device (in UAF) or can be a separate FIDO token that is connected to the user device (in U2F). The FIDO protocols are public key cryptosystems that use a challenge response

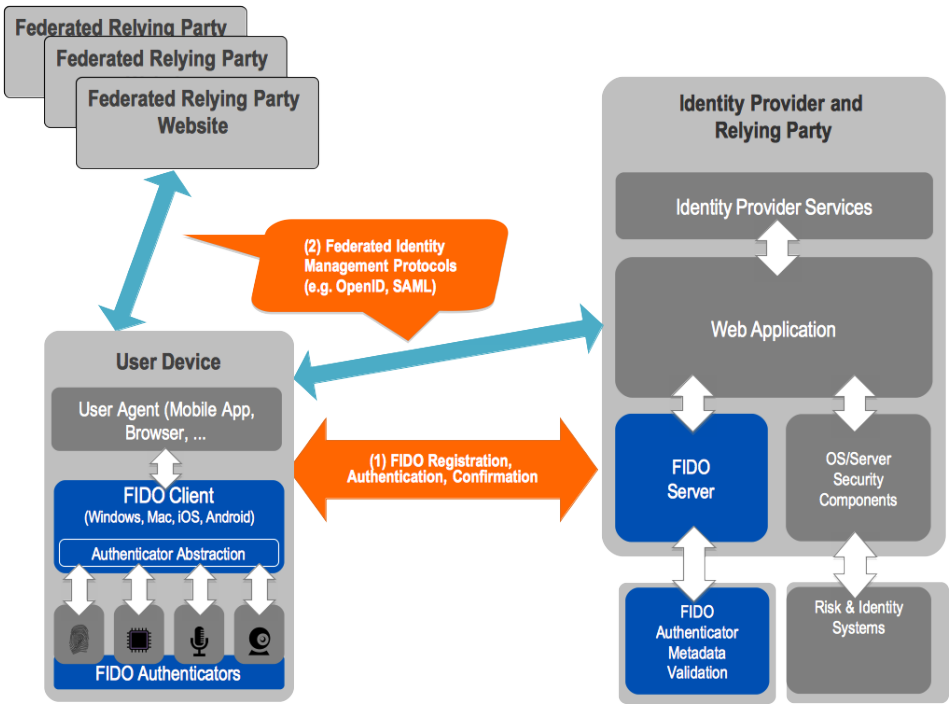

Figure 2.2: Overview of FIDO in the federation. Used with permission from FIDO Alliance.

protocol for authentication. They are open protocols and published implementations and libraries are open source. The key pairs are unique between the device and the server. This provides anonymity to the user, because the different FIDO server can not link the other keys to the device. As FIDO is a relatively new authentication technique, we wanted to investigate the possibilities of using it in an eID system. Another reason FIDO was chosen is because it supports two factor authentication, a requirement for our proposed solution.

2.6.1 Universal 2nd Factor

U2F uses a physical device, also called token. This token is connected to the client device. As of October 2015, most devices are connected by USB, but the protocol supports NFC and Bluetooth as well. In U2F, the user first authenticates using their password to the IdP and is then prompted to authenticate using the token. U2F uses the physical token to strengthen the password authentication. The token is responsible for managing the keys [25]. The FIDO U2F infrastructure contains

three parties, which are the FIDO token, FIDO Client and the relying party. As mentioned before, the FIDO token is a hardware device that signs the challenge from the server. The FIDO Client is usually a web browser which is responsible for handling the request to the FIDO token. The relying party is the server which is responsible for authenticating the user.

To provide protection against man-in-the-middle and phishing attacks, the FIDO Client packages information about the server. This is called client data. When the server receives the response, it checks that the client data is correct and that the request came from the server. The Client data contains:

- Challenge: the challenge from the relying party.
- Origin: the web origin that the FIDO client sees such as transport protocol and the domain names.
- Channel ID (Optional): used if the communication is secured by the Channel ID protocol.

To inform the token what key it should use, the relying party sends a key handle during the authentication request. The server also identifies itself in the request with an application id (AppID). This is the same as the web origin of the server. The steps for registration and authentication using U2F are described in detail below [26].

1. The user first logs on to the web service with username and password, and is redirected to the token registration page.
2. A JavaScript is called to start the registration. The JavaScript requests a RegistrationRequest from the server.
3. The RegistrationRequest contains a challenge and an AppID.
4. The JavaScript then calls the registration process in the FIDO Client with the info from the server.
5. The FIDO Client then computes the Client Data and packages the information. The information is packaged as two hashes: one over the Client Data which becomes the challenge and one over the AppId.
6. A request is then sent to all FIDO compliant devices attached to the user's device with the information.
7. The end user chooses a device by activating it, for example by clicking on it.
8. The activated device signs the challenge and returns the response containing:
   (a) The public key.
   (b) A key handle.
   (c) An Attestation certificate. This validates that the key was produced by an authorized manufacturer.
   (d) The signed challenge.

9. The FIDO Client then packs the response to the server. The response contains the raw response from the FIDO token and the Client data.
10. The package is then returned to the JavaScript which sends it to the server.
11. When the server receives the response, it verifies that it is correct. If it is correct, the key handle and the public key are bound to the user account.

After the user has registered the token, it can be used to generate the signature for authentication. The signature process consists of the following steps:

1. The user first logs on to the service using the original credentials (username and password).
2. The user is then redirected by the web service to the second stage of authentication. A JavaScript on this page calls the server and requests the key handle that the user has registered as well as a challenge.
3. The browser asks if the user wants to authenticate (this option can be remembered).
4. If the user wants to authenticate, the JavaScript calls the FIDO client's authentication function with the parameters from the server.
5. The FIDO client computes two hashes, one over the Client data and one over the AppID. Both parameters are hashed with the SHA-256 hash algorithm.
   (a) The hashed client data is called challenge.
   (b) The hashed AppID is called application data.
6. The FIDO client then sends the two hashes with the key handle to the different tokens.
7. The user activates one of the tokens. The token checks the key handles to see if it can sign the challenge. If the key handle was created by the token, it signs the challenge and the AppID data. The token then returns the signature to the FIDO client.
8. The FIDO client function packs the signature, the client data and the key handle.
9. The FIDO client returns the information to the calling JavaScript.
10. The JavaScript function sends the information to the server.
11. The server validates the data and if it is correct it grants the user access.

2.6.2 Universal Authentication Framework

The UAF registration and authentication process has many similarities to the U2F process for registration and authentication. The biggest difference compared to U2F is that it replaces the password, whereas U2F strengthens password authentication. This means that the user needs a way to authenticate to the device. FIDO proposes several ways to authenticate. Some of the proposed ways are fingerprint, face recognition and pin code. The different ways to authenticate provide different

levels of security. The server can chose which type of authentication it will allow to gain access to the service. This information is sent to the FIDO client during registration and is called polices. The different methods are presented to the user and the user chooses the one they prefer.

Given that UAF devices have a screen, the protocol can be used to sign transaction information. U2F tokens lack this feature as they do not have a screen. In the UAF protocol, the server can send messages to the device which the UAF client can show to the user in a secure manner. The user can then verify the information and sign it by authenticating to the UAF device.

2.6.3 FIDO Security

As mentioned before, FIDO was developed to strengthen the security of authentication and make it simpler. This is done by minimizing the use or importance of passwords. As mentioned in Section 2.4, using only a password does not provide enough protection. To provide security during transport, FIDO requires the use of TLS. The transport technique should provide a way to authenticate the server. This information is then used during authentication [27].

In the FIDO specification, the FIDO alliance mentions different attacks which the new protocols protect against. We have chosen to describe three major attacks against the system: man-in-the-middle, phishing and cloning a token.

Man-in-the-middle-attack

Both FIDO protocols have strong protection against man-in-the-middle attacks. To accomplish this, the FIDO protocols have server verification during the authentication and registration process. During authentication and registration, the server provides its identity two times. The first time is during the request message with the AppID information. The FIDO client then verifies that the web origin of the server is the same as the AppID. If these two values match, no man-in-the-middle is present. This is because the server has been verified during the setup phase of TLS using a certificate. If an attacker is present, they would have to forward the request with an AppID of the legitimate server. This would make the authentication fail as the web origin would be wrong or the attacker could not provide a valid certificate for the origin [26].

A man-in-the-middle attack can be performed on web services that use FIDO but it requires an advanced adversary. To perform the attack the adversary needs to acquire valid certificate with the same web origin as the web service. This certificate must also be signed by a trusted Certificate Authority. It is not the FIDO protocol per se that is bypassed but the TLS infrastructure. The integrity and confidentiality between the server and the user is broken. This have other larger consequences than just FIDO is broken. The attacker can then control all the communications between the browser and the server. This will bypass the protection in FIDO but the bar is high.

Phishing attack

A phishing attack on a FIDO system is difficult because of how the protocols are designed. Phishing is a valid attack when the goal is to obtain static password or credit card information. The information can be used by the attacker until the user makes an active decision to change or block them.

Both FIDO protocols are by design protected against this. This is because during every authentication, the user needs to sign a challenge from the real server. The phishing attack must then be performed in real time and the phisher must provide the origin and a valid certificate of the targeted server.

Cloned authenticator

By cloning a device, the attacker is in possession of a device that can be used to perform authentications. The sever can not distinguish the real authenticator from the cloned one. The specifications of both protocols specify that the keys should be stored in secure hardware [26]. To further protect against consequences of cloning, both UAF and U2F have a counter which is incremented for every authentication. This value is sent to the server with the response. If the counter value sent in the response is smaller than the stored counter value at the server, the device is cloned. The server then informs the user that the device is cloned and blocks the user [27].

2.7 OATH-HOTP and OATH-TOTP

The HOTP (HMAC-Based One-Time Password) and TOTP (Time-Based One-Time Password) algorithms provide methods for generating one time passwords (OTP). The HOTP algorithm was developed to provide an easy way for two factor authentication. As two factor authentication was a requirement, we chose to investigate OATH and its suitability to be used in an eID federation. The developers of OATH argued that previous authentication system had poor adaptation because they were expensive and hard to integrate, which led to low adaptation. HOTP and TOTP want to solve this with an open standard that is cheap and easy to implement in hardware and software. This is done using HMAC-SHA1, which is widely used and easy to implement. This provides a simple way to add two factor authentication to a system. TOTP is used by Facebook and Google to provide two factor authentication. They both use a software implementation of the algorithm. Google has developed its own authenticator app whereas Facebook uses third party apps [28].

OTP can enhance the authentication security in the following way: the user's OTP client generates a passcode. The generator can either be a hardware token or a software running on a device. The code is entered when the user provides their credentials. The OTP can be used as the only secret during authentication or in combination with a regular password. When the server receives the password from the client, it generates its own OTP and verifies that they are the same. If

the verification is successful, the user is authenticated.

The two algorithms use the same formula to calculate the OTP value but different seeds. The algorithm is based on HMAC-SHA1 where the output is truncated to make it easier to read for humans. HOTP was the first version of the algorithm, it uses a counter as the seed to generate OTPs. TOTP uses time steps derived from UNIX time as the seed for the OTP generation. During the development of the algorithm, six different requirements were defined:

1. The algorithm must be sequence- or counter-based.
2. The algorithm should not be costly to implement in hardware.
3. The algorithm must work with tokens that do not support any numeric input, but may also be used with more sophisticated devices such as secure PIN-pads.
4. The value displayed on the token must be easily read by the user. The HOTP value must be at least 6 digits.
5. There must be user-friendly mechanisms available to resynchronize the counter/clock.
6. The algorithm must use a strong shared secret. The length of the shared secret must be at least 128 bits. The recommended length is 160 bits.

To calculate the OTP, these values are needed:

- (HOTP) A counter C that is input to the function, which is incremented for each authentication. This counter must be synchronized between the HOTP generator and the validator.
- (TOTP) A time step T which is calculated in UNIX time.
- A shared secret K between the client and the server. The key needs to be unique for each generator.
- A Digit D which says how may number digits the HOTP value contains.

Figure 2.3 shows the different steps in the algorithm. The two algorithms are not complete protocols but the specification proposes some recommendations for how a protocol could work. The different steps are described in Figure 2.4. In the first step, the user authenticates against the server with something they know, for instance a password. In step two, the OTP value is sent and validated. If it fails, the user needs to send a new value. This is done a certain number of times. If this also fails, the user's account gets locked [29].

2.7.1 OATH–HOTP

This algorithm uses a counter to generate the HOTP value. A user can generate HOTP values that are not used to authenticate the user. This leads to desynchronization between the user token and the server. To solve this, the HOTP has a synchronization parameter called s. If step 3 in Figure 2.4 fails, the server tries to resynchronize s times by increasing the counter [29].

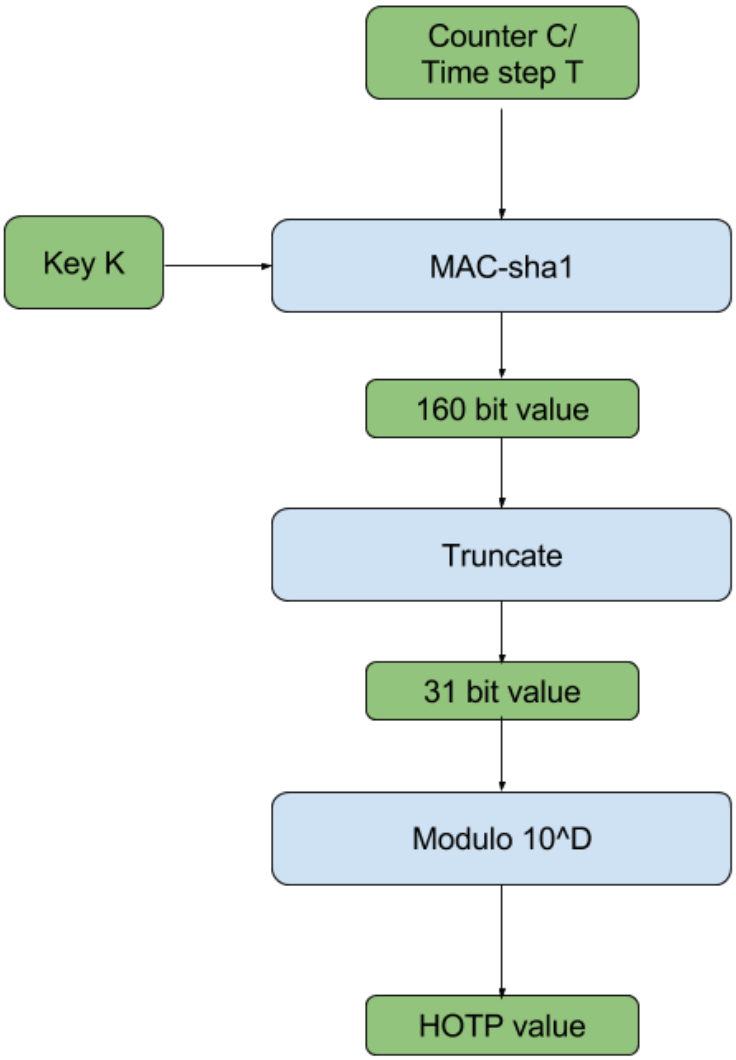

Figure 2.3: Overview of the HOTP algorithm

2.7.2 OATH−TOTP

The Algorithm is based on the HOTP algorithm but uses a time based counter to calculate the value. It can also use the more secure algorithms HMAC-SHA-256 or HMAC-SHA-512. As the algorithm uses time steps as generator value, it limits the time when the OTP can be used. The parameters to calculate the time step are:

- X which determines the step size. 30 is the default value.
- T_0 Is the start value in UNIX time when the generation of values started. 0 is the default value meaning that the number generation started at Unix

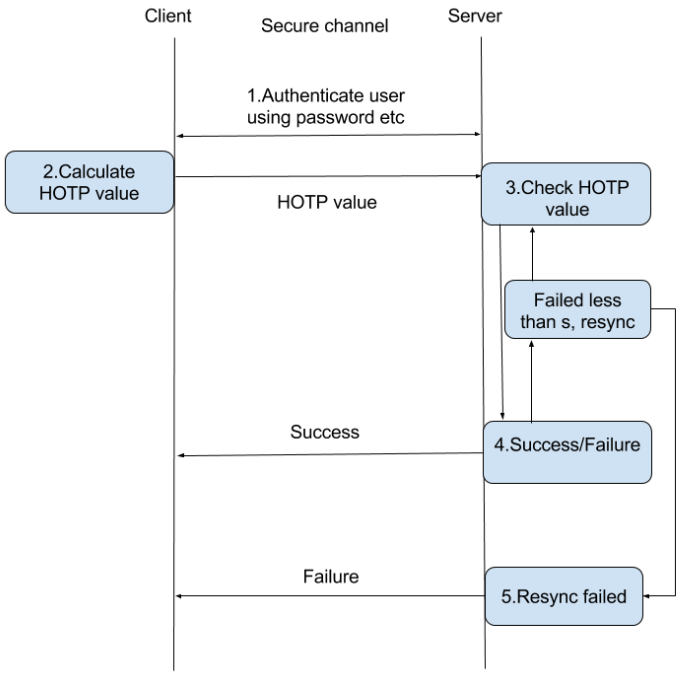

Figure 2.4: Overview of the OATH−HOTP proposed protocol

time zero.

The formula to calculate the time step is:

$$T = \left\lfloor \frac{(CurrentUnixtime - T0)}{X} \right\rfloor \tag{2.1}$$

T is used instead of C, like in the HOTP algorithm. To compensate for time fluctuation in the different devices, the algorithm supports a window around the time step. The window should be as short as possible. The specification recommends to only allow the next time step [30].

Chapter 3
Electronic ID systems

As mentioned before, BankID has a near monopoly on electronic IDs in Sweden, with a market share of 87% for all potential digital signatures in Sweden as of December 2015 [31]. The remaining 13% consists of mostly SP specific authentication and signature methods. The only other non-SP specific eID provider, Telia, has stopped issuing new eIDs for individuals [32]. This chapter describes how BankID and the new eID federation work.

3.1 BankID

BankID is developed and maintained by Finansiell ID-Teknik BID AB, which is owned by seven of the largest banks in Sweden [33]. It is a closed source code software, which makes it difficult to find out exactly how the protocol works and how the system is built. There are three forms of BankID: card BankID, desktop BankID and mobile BankID. Mobile BankID accounts for the vast majority of all signatures and authentications, and will therefore be the focus of this report. The local authentication differs slightly, e.g. when using BankID on a card, the user needs to connect it to the computer using a USB card reader. With mobile BankID it is enough to have the smartphone or tablet connected to the internet. Some reverse engineering has been done on the application mobile BankID [34]. A description of how some parts of BankID work can also be derived from the the BankID Relying Party guidelines [35].

BankID works like a federation but with only one IdP. Different SPs, such as banks, governmental agencies and other services can join the federation. BankID is issued by most banks in Sweden, including SEB, Swedbank, Handelsbanken and Nordea. The exact issuing process varies. For mobile BankID, a one time code is obtained either from a bank office or through a bank's online portal. The one time code is used to install the certificate on to the device (smartphone or tablet). When installing the certificate, the user chooses a PIN code of at least 6 digits. After installing the certificate, it is not possible to move it to another device or to change the PIN code. If a user wants a new PIN code, they have to uninstall the certificate and re-install it, going through the issuing process again. As it is not possible to move it to another device, the user will have to install a new certificate. This indicates that the private key is hard-coded into the device during setup.

When authenticating with mobile BankID, the user can access the SP either through the device with BankID or through another device. When asked to log on, the user types in their personal identity number. The SP then sends an authentication request to BankID. The SP prompts the user to open the BankID application. When the user opens the application, the application retrieves the request from BankID. This request contains information about which SP the user is authenticating themselves to. After the user authenticates using the PIN code, the device then sends an authentication confirmation (token of some sort) to BankID that in turn forwards it to the SP. After the SP has received the confirmation, the user is authenticated and logged in. Signing transactions follow the same procedure as authentication with the addition that the SP can choose a text that the user sees in the application before signing. The text displayed can for example be "2 transactions from SEB, total amount 1 000 SEK". This text prevents man-in-the-middle-attacks, as a potential attacker can't make a user sign another transaction.

It is possible to revoke a certificate without having access to it, by contacting the issuing bank. This security measure is similar to how credit cards that have been lost can be blocked. This is an efficient way to prevent ID thefts if the device is lost or compromised. If more than one authentication/signature is requested, all requests will be canceled. The application also has a maximum of four wrong PIN codes entered within a certain time limit, which prevents brute force attacks.

BankID is updated regularly but no bugs detected are posted publically. In our e-mail correspondence with BankID, they confirmed that they do regular penetration tests and security analysis. Furthermore, they have independent, external security analysis done at lease once a year and with every major deployment. However, these reports are classified for security reasons.

There is a Norwegian BankID, which is a different system sharing the same name. More success has been made reverse engineering the Norwegian version [36] than the Swedish one. There is also a published paper on weaknesses [37] for the Norwegian system. As this report focuses on existing and new eID systems in Sweden, no analysis is done for the Norwegian BankID.

3.2 Swedish eID federation

As mentioned, the Swedish eID board is in progress of launching a new infrastructure for electronic identification and signing (as of July 2015). Up until now, the private sector has been responsible for providing eID services, granted that they followed the guidelines and security levels. With the new infrastructure, an eID federation is created. This federation, under the control of the Swedish eID board, is responsible for rules, regulations as well as some technical infrastructure [38]. The infrastructure is open source and the communication is based on the open standard SAML 2.0. The federation has a central XML file with information about all the parties and their certificates. This XML file is signed by the eID

board's certificate. Having a central authority managing who can join the federation will make it easier for new IdPs to connect, as they only need to connect to the federation and not to each individual SP. As seen in Figure 3.1, a user connects

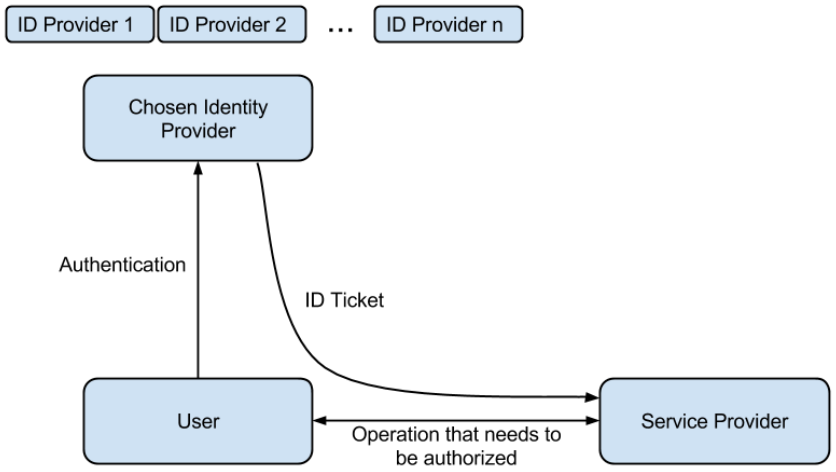

Figure 3.1: Federation infrastructure

to the SP and gets redirected to the chosen IdP using the Discovery Service (DS). The IdP then authenticates the user and sends an identity assertion to the SP, proving to the SP that the user is authenticated. The Swedish eID board provides a DS so the user can choose which IdP to use, no matter what SP it is connecting to [39]. To become an IdP, the organisation needs a way to register and authenticate users. These two processes needs to be evaluated by the Swedish eID board. If the processes are approved the organisation connects to the federation and can start generating ID certificate. IdPs are not responsible for providing signatures. Instead there is a Signing Authority that provide this service. When a user wants to sign a document, the Signing Authority redirects to IdP where the user authenticates. The Signing Authority then signs the document. With this solution, the IdPs do not need to be certificate-based.

Furthermore, there will be an Attribute Authority (AA). The AA can be used if a SP wants more information about a user, for example their phone number, provided that this information is stored with the corresponding eID. The communication between parties in the federation is based on the two SAML 2.0 profiles "Kantara Initiative eGovernment Implementation Profile of SAML V2.0" and "Deployment Profile for the Swedish eID Framework". By having the infrastructure open source, it is possible for people to review the code and settings used. However, how the IdP chooses to authenticate the user is up to each IdP and there is no requirement that this should be done using open source solutions. How the user is registered to the IdP is not defined by the Swedish eID federation. Each method of registration is evaluated when the IdP applies for IdP status. The registration process must ensure the authenticity of the user. If this cannot be granted, then

ID theft is possible.

Chapter 4
Our implementation

This chapter addresses our application both from a technical perspective and a functional perspective. The section about the infrastructure describes how the application is implemented, the different parts and how they interact. This chapter includes technical description, issuing, registration of a device, authentication and signing. This chapter also explains how a user interacts with the system, the communication in the system and what parties are involved. The authentication feature is fully implemented while the others are simulated in order to get a full cycle, from an issued eID until it is made invalid (either because of expiration or revocation).

In this chapter one can find the different system parts explained. The security analysis for different attacks can be found in Chapter 5. Finally, the implementation is discussed and reviewed in Chapter 7.

4.1 Infrastructure

To implement our system we have used several tools and libraries. The SP and the IdP run on a Linux server with the CentOS as operating system. For the web server, we use the Apache HTTP server. The Apache server is responsible for all communication with the user, as described in Figure 4.1. The communication with the user is done over TLS. Shibboleth's SP integrates natively with the Apache web server as a plug-in. The integrated SP provides access management to our protected resources in the proof-of-concept implementation.

The IdP from Shibboleth is written in Java and can be deployed in a web container as a Web Application Archive (WAR) file. We chose to use the web container Tomcat from the Apache foundation. A web container is responsible for managing Java servlets. Java servlets is a Java technology which provides dynamic web service. The dynamic content is generated by the Java classes and methods. Web containers are responsible for redirecting the web request to the right servlet. It is also responsible for managing the servlet's life cycle and the access rights to it. Java servlets are packed in a WAR. These WARs contain all information that is needed to run the web application. We chose to use the Tomcat container because it is officially supported by the Shibboleth IdP.

Tomcat also integrates easily with the Apache HTTP server. The Tomcat server talks with the Apache server over the AJP protocol. AJP stands for Apache JServ Protocol and is a binary protocol which provides a proxy between the Apache HTTP server and an application server. The application server is in our case the Tomcat server. Requests to the Tomcat server are redirected by the Apache server from the rest of the Internet. Any response from Tomcat is returned to the Apache server, which sends the response back to the user who requested the web resource. The Tomcat runs on the same server and the communication between the Apache server is done over localhost. Figure 4.1 illustrates the connection between the Tomcat server and the Apache server.

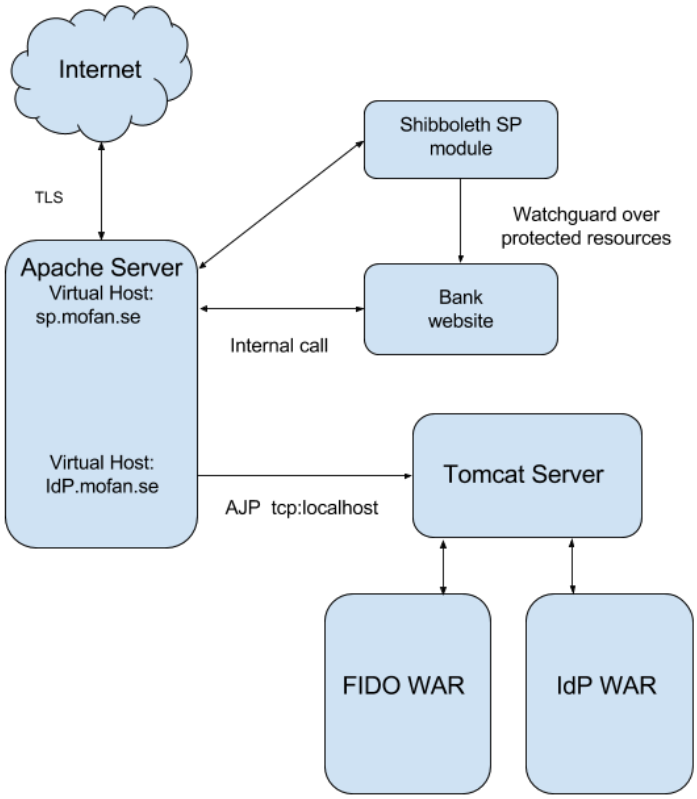

Figure 4.1: Our infrastructure

To store information about our user we use MariaDB, a SQL database based on MySQL. The database was forked from MySQL and is developed by developers from MySQL who quit after Oracle acquired Sun in 2009 [40]. MariaDB is compatible with many MySQL commands. In the database we store the user credentials. The information we store are the users' personal identity number, password and the registered FIDO keys. To protect the passwords, we use a unique salt for each password and compute a hash. The computed hash is then stored in the database.

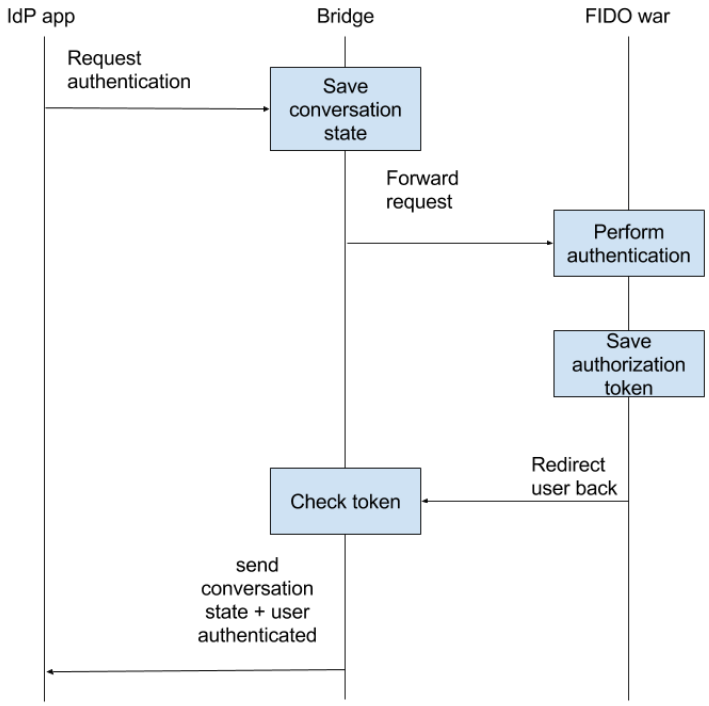

Figure 4.2: FIDO bridge

Our IdP system is based on two web applications, the FIDO WAR and the IdP WAR, as shown in Figure 4.1. First of all, we use the IdP from Shibboleth that handles all the authentication requests from the SP. The IdP is configured to support the authentication from our FIDO implementation. To connect the IdP and the FIDO application, we have a service running inside the IdP application that acts as a bridge. This bridge is responsible for handling the authentication request between the FIDO application and the IdP application.

Figure 4.2 shows how the user is redirected between the different parties: IdP application, bridge and FIDO application. When the IdP application wants to authenticate a user, they are redirected to the bridge which saves a conversation state. It is used by the IdP application to identify that the user has been authenticated. After the conversation state is set, the user is redirected to the FIDO application. The FIDO application then authenticates the user. This process is described in Section 4.3. When the FIDO application has authenticated the user it puts a token in the database and redirects the user back to the bridge. This token contains the conversation state and the personal identity number of the user. The bridge verifies that authentication has occurred by verifying the saved token in the database. If the token exists and is correct, the bridge notifies the IdP application that the authentication has occurred. To generate FIDO authentica-

tion and registration requests we use a library written by Yubicon. This library is also responsible for verifying the responses from the user. Yubicon is one of the organisations that created the U2F protocol. It also manufactures U2F compliant tokens.

4.2 Distribution and issuing

As a complement to our technical implementation, we propose a scheme for how the distribution, issuing and revoking FIDO devices could work. One idea with FIDO is that the user should be able to choose which vendor and device to use. Therefore, the distribution process proposed does not include the acquiring of the FIDO device.

The distribution of the eID will only concern how the user registers a FIDO device. As long as a FIDO device is approved by the alliance, it is considered safe, the user can therefore get the device from the vendor of their choice. The steps before a user can authenticate using FIDO are explained below.

1. The user applies at the IdP with personal identity number.
2. A one time code valid for a limited time is sent to the closest post office.
3. The user gets a note in the mailbox (to the address associated with the personal identity number) that they need to visit the closest post office. At the post office, the user identifies themselves using ID and receives the letter with the code.
4. The user goes to the IdP's website and registers their FIDO token using the one time code. It will only be possible to use the code to register a FIDO device with the correct personal identity number. This process is described in Section 4.2.1: registration of a device.
5. The user can now use the FIDO token to authenticate at the IdP. The IdP then grants the user access to all SPs in the federation.

With this system, a potential attacker would need to have the ID as well as access to the user's physical mailbox. The IdP will be responsible for storing the public key together with the personal identity number. In case of a lost key, it is possible to revoke a token by contacting the IdP. If a U2F token is lost and found by someone else, they will not be able to link the token to a personal identity number.

The eID credentials are valid a limited time, for example five years. Before expiration, the user must renew the key pair. This is done at the IdP where a new FIDO registration process occurs. The user can use the old device but the key pair of the account has to be regenerated.

4.2.1 Registration of a device

To register at the IdP, the user first performs the steps described in Section 4.2. When the user has received the registration code, they can register at the registra-

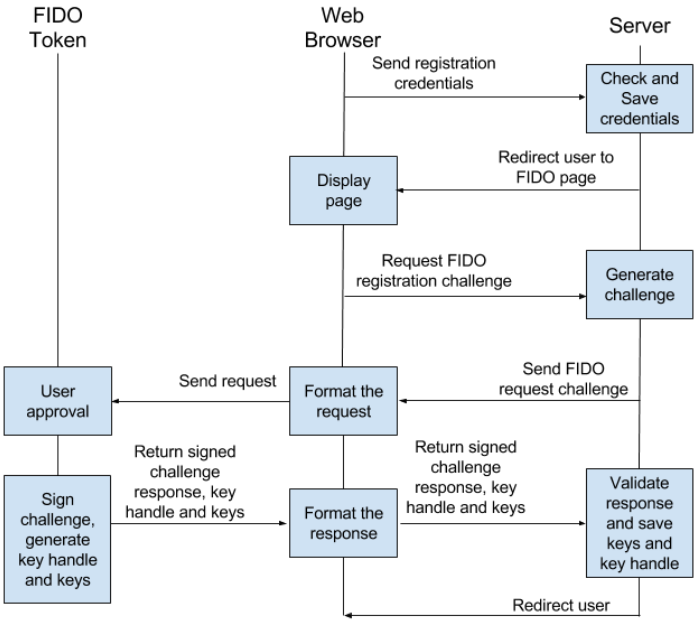

Figure 4.3: Registration process

tion page. Figure 4.3 illustrates the different parties in the registration process and their communication. The user performs the following steps during registration:

1. First the user enters their personal identity number, password and the one time registration code.
2. The information is sent to the server, which verifies that it is correct.
3. If it is correct, a new user account is created and the information is saved in the database.
4. The user is redirected to the FIDO token registration web page.
5. The browser sends a registrations request to the FIDO service.
6. The server responds with a registration request containing a challenge.
7. When the browser receives the request it starts the FIDO registration process, which is explained in detail in Section 2.6.1.
8. The response from the FIDO token is then returned to the FIDO service.
9. The FIDO service verifies the response. If it is correct, the FIDO token is registered to the user account.
10. The user can now use the key to authenticate at the IdP.

4.3 Authentication

The authentication in the FIDO application is done in two stages. Figure 4.4 describes how the different parties communicate. To authenticate against the FIDO service the user follows the following steps:

1. The user gets redirected to FIDO service from the bridge between the FIDO service and the IdP service.
2. The redirect contains the conversation state which is used to identify the authentication.
3. The user enters their personal identity number and the password into to the browser.
4. The browser then sends the conversation state, the personal identity number and the password to the FIDO service.
5. The FIDO service saves the personal identity number and conversation state and verifies the password.
6. If the password is correct, the user is redirected to the FIDO authentication page.
7. The browser requests to authenticate and the FIDO server responds with an authentication request.
8. When the browser receives the request, it starts the authentication process, which is explained in detail in Section 2.6.1.
9. The response from the FIDO key is then returned to the FIDO service.
10. The FIDO service verifies the response. If it is correct, the user is authenticated.
11. The FIDO service saves the personal identity number and conversation as a token in the database. The user is then redirected back to the bridge.
12. The bridge verifies that the user is authenticated by checking the token.

4.3.1 Signing

As described in Section 3.2, the signing process is done by the signing authority. The signing process have the following steps:

1. The user is redirected from the SP with the document it wants to sign to the signing authority.
2. The signing authority present the information the user should sign.
3. If the user approves it is redirected to the IdP.
4. At the IdP the user performs the authentication and the attestation is generated by the IdP.
5. The user is returned to the signing authority that verifies the attestation.

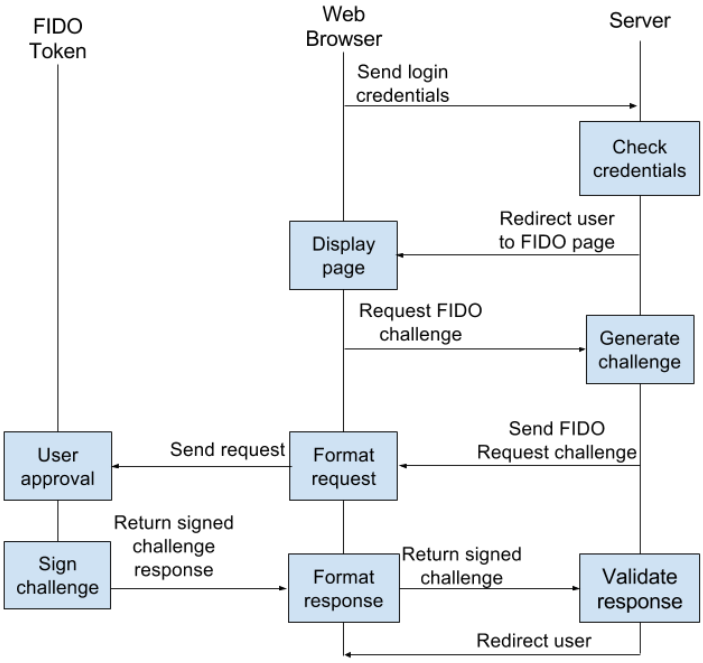

Figure 4.4: Authentication process

6. If the attestation is correct the document is signed.
7. The user is then redirected back to SP with the signed document.

Chapter 5

Attacks on the implementation

This chapter describes four attacks that could be carried out on our proof of concept implementation for how the Swedish eID federation could work. Each attack is described in general, then how it could be applied to our system and lastly how our system mitigates the attack. The attacks were chosen based on how common they are and how severe the damages would be to our system if successful. To asses this, we looked at the Open Web Application Security Project's (OWASP) top 10 [41] and the CWE's list of 25 most common attacks [42]. This lead us to choose the following attacks: phishing, Man-in-the-Browser, SQL injection and Denial of Service.

5.1 Phishing

Phishing is an attack where the user is tricked into providing confidential information about themselves. The confidential information is usually a password, credit card number or personal identity number. To phish the confidential information, the fraudster mimics trustworthy websites and emails that are sent to the user. Jakobsson [43] describes three stages of a phishing attack:

1. In the first step, called the lure, the fraudster spams the user with emails. In these emails there is a link to the fake website that the phisher controls. The emails state some facts that encourage the user to follow the URL. A simple example would be the need for the user to verify that their credit card is still valid.

2. The second step is a mock website that mimics the original website. The mimic website should be so alike the original that it fools the user to enter its confidential information.

3. In the last step, the phisher uses the information that has been phished to perform some malicious activity. An example would be that the attacker uses credit card details to make a purchase.

The security software company Kaspersky Lab publishes yearly reports about phishing attacks against their users. These reports give a comprehensive overview of which businesses that are attacked, where the attacks come from and the targeted countries. In 2015, the primary targets were online finance companies with

34 percent of all recorded attacks, followed by Internet portals with 32 percent. Internet portals are companies like Google and Yahoo, where the attacker gains access to many different features from only one account. Russian citizens are the most targeted with 17 percent of all attacks targeting them [44]. In 2014 the main targets were Internet portals with 41 percent of all phishing attacks while 29 percent of the attacks targeted online finance companies [44]. Phishing is a very serious attack with many successful attacks. In a white paper by Kaspersky Lab, they show that during 2012 37.3 million users were affected by phishing attacks [45].

5.1.1 Applied to our system

An attacker can have different goals with a phishing attack. The most basic attack would be phishing the user for their personal identity number. This number can be used to perform other types of fraud, such as identity theft. Because of FIDO's design, a successful phishing attack, meaning getting hold of login credentials, would not give access to the victim's account. In an advanced phishing attack, the attacker would have the possibility to perform a real time attack and get access to the account once. To perform this attack, the phishing must be combined with a man-in-the-browser. This set up is described further in Section 5.2, when describing the man-in-the-browser attack.

5.1.2 Mitigation

As stated before, there can be different goals with a phishing attack. The case when the attacker only wants to phish the credentials is a standard phishing attack. Such an attack is mitigated with blacklisting of sites and educating users about the risks of phishing. Other solutions have been proposed by Rachna Dhamija [46] in a research paper. One of them is to move the authentication from the original web site to a trusted login window. To make this window hard to impersonate for the phisher, the login window displays a picture unique to each user. The user only enters their credentials when the window with the picture is shown. This makes it hard for the phisher to recreate a login screen which fools the user to enter their credentials by mistake. FIDO is by design protected from phishing in the second step of the authentication. This is explained more in detail in Section 2.6.3 about security in FIDO.

5.2 Man-in-the-browser

Man-in-the-browser (MITB) is an attack where the attackers inserts a Trojan in the user's browser. This is a Man-in-the-Middle (MITM) attack where the attacker resides in the browser. The Trojan intercepts, reads and modifies data in the browser. Since the data is decrypted before it reaches the Trojan, TLS does not offer any protection for the data. To perform its operations, the Trojan exploits external features like browser plugins, add-ons and user scripts [47]. The add-ons are installed either through malware or in rare cases through the browser add-on distribution center. Add-ons run with high privilege and can access many of the

browser's features. One of the privileges is access to the DOM tree. The Trojan uses this feature to modify which elements the user sees and which elements gets posted using the HTTP command POST. An example would be a user accessing a bank website to perform a bank transaction. The Trojan then modifies the DOM tree so that the fraudster's account information is used instead. The information that the user entered is stored to be used when the server responds. When the server responds with the confirmation, the Trojan alters the response back to the information entered by the user. This gives the illusion that the right transaction occurred.

Another threat would be that the Trojan attacks the AJAX functions in the web browser. To achieve this, the Trojan changes the XMLHttpRequest object in the browser. This is possible as objects are created by cloning other objects. By altering the XMLHttpRequest send function through adding malicious code, all future XMLHttpRequests will run the malicious code inserted [48]. An example of how the code is changed is shown below. The Trojan can also get all the passwords and usernames that are entered in the browser as it has access to the DOM tree [47].

```
XMLHttpRequest.prototype.originalSend = XMLHttpRequest.
    prototype.send;
var evilSend = function(data) {
  // Modify the data here
  this.originalSend(data);
};
XMLHttpRequest.prototype.send = evilSend;
```

An example of a MITB Trojan is Zeus [49]. This malware is capable of performing many different tasks. It was designed to attack banking sites. When the Trojan has infected the system, it intercepts and modifies information that is sent to and from the browser. The codebase for the Trojan is published online. This has resulted in the publication of many different software based on Trojan code [49]. The Trojan has a low detection rate making it hard to find for virus protection software [50]. In 2009, Zeus and Zeus based trojans were estimated to have infected 3.5 million devices [51]. With the use of machine learning techniques, Mohaisen and Alrawi propose different techniques to identify the Zeus Trojan [49].

5.2.1 Applied to our system

Our system and the FIDO protocol can be vulnerable to this type of attack. This is because the FIDO protocol relies on the browser being secure. If the attacker can install an add-on in the web browser that controls the communication between the browser and the FIDO token, it can bypass the origin check. Without the origin check, U2F's MITM protection is removed. A MITB attack on its own will not damage our system, it is rather used as a tool to perform other attacks. A successful MITB attack will enable an attacker to perform real time phishing attacks against the system. In a real time phishing attack, the intention is not to obtain the credentials but to get the user to authenticate and give the attacker access to the account during the attack.

To perform the MITB attack on our system, the attacker would need to install a malicious add-on that can send sign requests with an alternative origin to the FIDO token. When the malicious site wants the user to sign a request the add-on is called to perform a signing. The add-on would then set the origin to whatever the attacker wants it to be. The add-on would be installed like any other type of malware. The attack would be divided in several steps. In the first step, the malware is downloaded. It then collects all the information needed to perform the attack and sends an e-mail to the intended target. The e-mail contains a link to a malicious site where the user is prompted to authenticate. A JavaScript on the mock site would use the malicious add-on to authenticate. The following list describes the attack in more detail:

1. The user visits a site that is infected with malware.
   (a) A script on the website identifies an application that can be exploited.
   (b) The script exploits the application to download and install a malicious add-on.
2. The malicious add-on monitors if the user authenticates to the IdP using a U2F token.
3. If the user authenticates the malicious add-on starts to record e-mail address that can be used to send phishing mails.
4. The attacker sends a phishing e-mail encouraging the user to authenticate with the FIDO token on the phishing site.
5. On the phishing site, the user is prompted to authenticate. The server that runs the phishing site requests a login on the site it wants access to. It forwards the FIDO request to the user.
6. The malicious add-on captures the request and forwards its own code that performs the communication with the token. This code sets the origin of the intended target so the MITM protection is bypassed.
7. When the malicious add-on returns the signed U2F request, the browser forwards it to the malicious server. The malicious server then uses the request to get access to the the targeted server and can perform its task on it.
8. If the malicious server needs the user to perform another task, it prompts the user to authenticate once more. When the malicious server is done, it redirects the user to a site where it confirms that the authentication was successful.

5.2.2 Mitigation

The attack above is difficult to mitigate from if Channel ID is not used and the browser is susceptible to MITB attacks. To secure the browser from this kind of attack, the add-ons used in the browser must be tested extensively so they do not

perform any malicious activities. The browser needs to verify that add-ons are safe at any given time. Modern browser have this feature as the add-ons are checked by the browser vendor and signed. When the add-on is loaded, its signature is verified to make sure it is safe. The user must also explicitly approve of the add-on when it is installed. This solution provides good protection against the MITB attack. Even though the protection is good, it is not one hundred percent effective. There have been cases where malicious add-ons have been published in the Chrome Web Store [52]. These add-ons passed Google's inspection and were installed from Chrome Web Store on user's computers. One of the add-ons targeted Facebook, where the malicious add-on could perform many actions. A common action was to post information from the user's account, but it could also message the user's friends through the Facebook messaging function.

Another solution is to use a hardened web browser. The hardened browser is statically compiled, cannot run any add-ons and its binaries are protected. This provides more protection than the first example because it is nearly impossible to run malicious code in the browser. There are some hardened browsers available today but these are not widely used. As the hardened browser needs to be statically compiled, this feature cannot be turned on and off on command.

In the documentation, the FIDO alliance proposes a solution where the communication with the FIDO token is done in the OS kernel [53]. The OS can then make sure that the communication with the FIDO token goes through the OS, which disables all other direct communication. Since the communication is done through the OS, all attacks that alter the origin are stopped.

5.3 Denial of service

In a denial of service (DoS) attack, the attacker's goal is to disrupt the service so that other users can't access it. There are many different types of DoS attacks:

- Attack on the Network device: the device software or hardware is attacked resulting in a crash in the device.
- Attack on the OS: the attacker exploits how the OS implements protocols to crash the system.
- Attack on the Application: the attacker either exploits a bug in the application to crash it or triggers extensive calculations in the application making it inaccessible.
- Data flooding attack: the network is flooded with dummy packages to congest the network.
- Attack on protocol features: the attacker exploits weaknesses in the protocol to perform a DoS attack.

Flooding the system is the most common DoS attack. The flooding either uses all the bandwidth or allocates all the resources on the device, making it impossible to connect to the device. Flooding attacks are often in a distributed form with

many attackers. This is called a distributed denial of service attack or DDoS. In a DDoS attack, the attacker utilizes many unknowing zombie computers to perform the attack. This is accomplished with the use of a Trojan that infects the user's computer and take orders from the command and control server. The attacker uses the botnet to flood the target with bogus requests [54]. These botnets can contain thousands of attackers. Recent real world attacks have been on the Sony PlayStation network [55] and targeting the largest newspapers in Sweden [56]. In the PlayStation network case the attack lasted several days. In the attacks on newspapers, legitimate users could not access the website and could not use the service.

5.3.1 Applied to our system

Our system is as weak as any other system to DoS attacks. No web service is safe against DoS attacks. The most suitable DoS attack against our system would be a DDoS. A DDoS attack that knocks out the IdP server would have severe consequence, as many users would not be able to use their eIDs. Two examples that can be used against our system are DDoS flooding and amplification attacks. These attacks would congest the network and utilize too much processing power at the server, making the network drop legitimate packets. This will stop the legitimate users form accessing SPs.

5.3.2 Mitigation

As mentioned previously, protecting the system against DoS attacks is difficult. This is because an attacker can use many different techniques to perform a DoS attack. Douligeris [54] proposes many different techniques to advert DoS attacks. The first step is to disable non-essential services that are running on the server which are connected to the Internet, as these services can be exploited to perform the attack. Other techniques that are proposed include detection and filtering where the systems detects the attack sources and block them. Filtering techniques like Ingress filtering and Egress filtering check that only valid IP addresses enter and leave the network. This protects against spoofed addresses.

One can also use a multiple server solution. In this scenario the several slave servers share the same content and a master that delegates the workload. This minimizes the load on each station. This makes it harder for an attacker to bring down the service as they will have to bring down several servers. The network can also increase the bandwidth of critical systems. This limits the effect if the network is flooded. To increase the protection one gets from load balancing one can use content delivery networks (CDN). These networks are designed to distribute the content of the web service on different places to reduce latency and increase capacity under load. These features can be used to protect against DDoS attacks. Zakaria Al-Qudah proposes a way to utilize CDN to protect against DDoS attacks [57]. In March 2013, the CDN CloudFlare mitigated an attack which generated a mean traffic of 75Gbit using DNS amplification [58]. To protect our system from DDoS is difficult but there are techniques that mitigate the attacks as explained.

A deployment of an authentication system should consider every possible solution to protect against DDoS attacks. Our system does not have any DoS mitigation implemented as it is only a proof-of-concept.

5.4 SQL injection

SQL injection is a class of attacks that create malicious SQL statements which perform operations on a database. CWE's list of the most dangerous software errors in 2011 ranks SQL injection on top [42]. There has been many incidents where the consequences were severe. In an attack in 2010, the company Neo Beat based in Japan and their business partners were attacked. The hackers downloaded credit card information for 30,000 customers [59]. Another major attack was an attack against Yahoo in 2012. In this attack nearly 443,000 accounts where compromised including the passwords which were stored in plain text [60]. The website RockYou also suffered an attack in 2009 where 30 million plain text passwords and usernames were leaked [61]. In an attack against the Swedish site bloggtoppen in 2011, 94000 hashed password and usernames were leaked[62].

Halfond [63] describes 10 goals for SQL injection attacks. The most important goals for us include retrieving information about the database in order to perform other SQL injection attacks, extracting data, modifying data, adding data and bypassing authentication. In the case of extracting data, the attacker is often interested in information concerning the users. This information can be e-mail addresses, credit card information, passwords and/or other personal information.

An attacker can use different ways to inject code on the SQL server. Halfond [63] describes four different ways to inject the SQL statements:

1. Injection through user input fields.
2. Injection through cookies. Only possible if cookie information is used to build the statements.
3. Injection through server variables stored in communication headers. These variables are used when traffic is logged.
4. Second-order injection. The attacker stores the injection in the database. For example in the user name column. An example of input would be `admin '--`. When the database executes another statement that uses the stored information, the injection attack is performed.

There are many different types of SQL injection attacks. Some attack types are tautologies, union queries, illegal/logically incorrect queries and piggy-backed queries. In a tautology attack, the attacker changes the database call to always evaluate to true. This attack can be used to bypass authentication in order to extract data. In the example below, a statement is built using the input from the input fields "username" and "password" to authenticate a user. The statement is then sent to the database which fetches information stored in the table "USERS". If the username and the password match the information stored in a row, that row is returned and the authentication process is complete.

```
--The statement is built with this string concatenation.
--This pseudocode describes hows server
--builds the request to the SQL database
SELECT * FROM users WHERE
        login="+user_input_name+"  AND pass="+
            user_input_password+"
--In a normal case the variables' value would be
user_input_name = admin
user_input_password = password1234

--Resulting in a string that is sent to server:
SELECT * FROM users WHERE
        login="admin"  AND pass="password1234"

--This would return a result set with information
--from the admin row in the user table
```

Instead of entering a password, the attacker performs a tautology injection in the password input field. This will cause the statement to always evaluate to true and return that row.

```
-- attacker input would be
user_input_name = admin
user_input_password = " OR 1=1 --

--this is the executed statement where from the attacker
SELECT accounts FROM users WHERE
        login="admin"  AND pass="" OR 1=1 --
--This would render the same result as the proper
--statement without the attacker knowing the password.
```

Since the statement evaluates to true even though the password is not correct, the admin row is returned. As the row is returned, the authentication is successful. This injection always returns a row if the entered username exists. In other words, the attack will be successful.

When attackers perform illegal/logically incorrect queries attacks, the goal is to extract information about the tables and the SQL server. This will give the attacker insight on the way the tables are constructed. Using this information, the attacker can launch other attacks, for instance to extract data. With piggybacking, the attacker adds extra commands to the request to perform many different types of attacks. These attacks are dependent on the server supporting several statements in the same query. An example of this attack would be a denial of service where the attacker drops one of the tables.

```
--The statement is built whit this string concatenation
SELECT accounts FROM users WHERE
        login="+user_input_name+"  AND pass="+
            user_input_password+"
-- attacker input
user_input_name = doe
```

```
user_input_password = "; drop table users --"
--the executed statement
SELECT accounts FROM users WHERE
        login="doe"  AND pass=""; DROP table users --"
```

When the server receives this request, it executes the first request before the query delimiter (;) and then the DROP table statement. The consequence of this injection attack is that the table "users" is dropped. This causes denial of service to the rest of the users. If the table does not have a back up the data is lost forever.

5.4.1 Applied to our system and mitigation

Applied and mitigation is together in this attack because it is simpler to explain the connection between them in this attack. Since our code base is small and the SQL queries are simple, it is easy to mitigate SQL injection attacks. Our requests are protected in two ways: by prepared statements and by logic on the web server. In the following naive implementation the code uses no protection against SQL injections.

```
public boolean checkUser(User u, Password p) {
  Connection con = null;
  con = db.getConnection();
  Statement stmt = con.createStatement()
  String query = "select * from Users where pNbr = "+u.getPnbr
      +";"
  ResultSet rs = stmt.executeQuery(query);
  if(rs.next()){
    String salt = rs.getString("salt");
    stmt = "select * from Users where pNbr = "+u.getPnbr+"and
    passWord="+Utilities.generateHash(salt, p)+";"
    rs = stmt.executeQuery(query);
    if(rs.next()){
      return true;
    }else{
      return false;
    }
  }else{
    return false;
  }
```

The implementation would be vulnerable to a simple input that is saved in `User u` attribute `pNbr`. The input requires a valid personal identity number that is registered in the database for instance `0000001111`. The injection value would then look like this `0000001111";--`. This would return true from the function and the attacker would bypass the first step in the authentication process. In this case the only thing that the attacker needs is the FIDO key to complete the process.

The following implementation is used in our system, and offers security against attacks on the authentication process.

```
public boolean checkUser(User u, Password p) {
  Connection con = null;
  con = db.getConnection();
  String query = "select * from Users where pNbr = ?";
  PreparedStatement stmt = con.prepareStatement(query);
  stmt.setString(1, u.getPnbr());
  ResultSet rs = stmt.executeQuery();
  if(rs.next()){
    String salt = rs.getString("salt");
    String username = rs.getString("pNbr");
    String storedPassword = rs.getString("passWord");
    return storedPassword.equals(Utilities.generateHash(salt,
        p))
                    && username.equals(u.getPnbr());
  }else {
    return false;
  }
```

By checking the username and the password from the returned statement, any SQL injection problem is resolved. The system also uses prepared statements which protect against SQL injections. Using prepared statements means that they are sent to the server first and are compiled without the data. This is used to speed up requests that use the same query statement. But this can also be used to secure the request to the database. As the query is already compiled, the data can't change the logic of the query. Without these protections, a SQL injection would be possible.

Chapter 6
Analysis

This chapter aims at analyzing the security of the protocols and frameworks used in our prototype. The security is analyzed for each protocol and framework individually, by reviewing relevant security research and published attacks. Apart from security reviews on the used techniques, a security analysis on the OATH protocols is also included. This analysis is then used when comparing the security between OATH and FIDO and motivating why FIDO was chosen. The chapter also contains an analysis of the way using open source affects the software security. How using open source affects an eID system and the non-security related questions will be discussed in Chapter 7.

6.1 Open Source secure enough?

BankID runs on proprietary code and exactly how the protocol works is secret. This makes it difficult for someone without access to the source code to make a comprehensive security analysis of it. For that reason, this thesis looks into the security of open source software in general in order to find out if open source could be secure enough for an eID system. Previous research comparing the security between open and closed source has mostly concerned operating systems, office software, e-mail clients, web servers, web browsers and database management systems. Even if there is no specific research on authentication systems, the results are coherent over different systems. Therefore it is a reasonable assumption that a deep analysis of authentication systems would yield the same result.

The research on the issue of open source security does not provide a definite answer on what is more secure. As demonstrated in Section 2.3, some research shows that closed source yields better security and other research shows that there is no difference in security. There has been limited research done on the topic and few vendors and software have been deeply analyzed, which makes it hard to say that one is more secure than the other. As some research indicates, it is rather the policy of the vendor and how they work with bugs, development processes and so on that influences how secure the software is. As the level of security depends on the policies of the vendor, it is important to use open source properly if one decides to use it. Good policies, from a security perspective, is a vendor who has regular security reviews, a well functioning bug reporting system and regularly

updates the software to fix bugs. If the code is not reviewed by more than a few people, there is a risk that potential attackers will have an advantage.

Already in 1883, Kerchkhoff's principle was formed where he argued that secure military systems [64]: "must not require secrecy and can be stolen by the enemy without causing trouble". This is widely recognized in academia for cryptography algorithms and we believe this should be applied to software systems as well. The National Institute of Standards and Technology (NIST) has the same opinion and has openness as one of its main principles for software security. They state [65]: "system security should not depend on the secrecy of the implementation or its components". Open source should always be considered an option and the results of previous research [11] [15] [9] combined with above reflections shows that open source code should not be considered less secure than closed source code.

One can also argue that having the software open source is not just a question of security. When having a system for the citizens, it should be as open and transparent as possible. Hoepman & Jacobs [8] compare the trust issue with a locksmith. Would you rather trust a locksmith that shows how he works or one who does it all in secret? If one can see the process, the security relies exclusively on the complexity of the key, which is how it should be. It is important that users trust the federation, and having it open source makes it more trustworthy. The topic of trusting the federation is reviewed further in Chapter 7.

6.2 FIDO

This section will analyze the security of FIDO. It will look into: a trust attack on FIDO, the security reference of FIDO, security differences between UAF & U2F and why we chose U2F.

In a study from 2015, Loutfi & Jøsang [66] look at the trust requirements that FIDO deals with. Trust is defined by McKnight [67] as: "the extent to which one party is willing to depend on the other party in a given situation with a feeling of relative security, even though negative consequences are possible". When a SP has its own authentication service, it is called isolated identity management. Then there is a trust requirement that the SP handles the privacy with care and that the user takes adequate precautions with their credentials. Some trust requirements when using FIDO include the user having to trust the FIDO device and that the manufacturer will not intentionally or unintentionally break the unlinkability property.

The paper by Loutfi & Jøsang points out that using FIDO introduces many new trust requirements. However, several of the requirements introduced are not unique to FIDO and will be present in any federated identity service. By using FIDO in a federated identity service, the trust requirements between the SP and the user are not related to FIDO, as the SP will not implement the FIDO protocol and only forward the user to the IdP. The trust requirements towards the manufacturer

are important as U2F uses a physical device that can be supplied from different vendors. However, as all FIDO tokens are identified to which batch they belong, it is possible for the protocol to make the compromised tokens invalid, which would counter any security breach in hardware. A compromised hardware would then lead to a user being forced to get a new token.

Another feature that will be available with the federation is that one can decide how reliable a mode of authentication is and give the privileges thereafter. This will enable a simple authentication process (for example a fingerprint) to give a user access to check the account balance. A more complicated authentication process (for example a fingerprint and password) would be required in order to sign transactions. This will not replace the trust requirements but will make it easier for the federation and/or user to decide what devices are to be trusted and to which extent.

Loutfi & Jøsang raise some valid concerns that FIDO has received little challenge from the security community. They also conclude that FIDO does not eliminate trust requirements and rather move them or introduce new ones between different parties, such as trust requirements for the hardware manufacturers and the FIDO consortium. By using FIDO in a federated identity service, there will inherently be many trust requirements. However, these will not be unique to FIDO and will be present no matter what technique or protocol that is used for authentication.

FIDO has published a security analysis [27], called "security reference" that covers both UAF and U2F. It is edited by security experts from PayPal and Nok Nok Labs. The analysis has also been reviewed and contributed to by the independent security company iSECPartners. The security reference first states 15 security goals, such as security goal 2 (SG-2): "Credential Guessing Resilience: Provide robust protection against eavesdroppers". It also contains security measures, for example security measure 2 (SM-2): "Unique Authentication Keys: Cryptographic authentication key is specific and unique to the tuple of (FIDO Authenticator, User, Relying Party)". Finally, the report makes security assumptions, for example that the used algorithms and key sizes are adequate and not broken.

There is a threat analysis covering several attacks, including phishing and three kinds of MITM-attacks. By looking at the security measures, the reference shows how FIDO mitigates against the attacks. For example a real time MITM attack is not possible as the protocol uses channel binding (SM-12) and will detect the MITM in the TLS channel by comparing the channel binding information provided by the client and the channel binding information retrieved locally by the server. The security reference makes a comprehensive analysis of how FIDO is affected or not by several attacks. In Chapter 5, some of the attacks in the FIDO security reference are also reviewed but with regards to the way they impact our system.

Both UAF and U2F fulfill the requirements for two factor authentication. They both rely on the ownership factor, as the user either needs to have the device with UAF embedded or the U2F token. For the second factor, U2F uses the knowledge

factor (with a password), while UAF has not defined which factor to use. With U2F, it would be possible to add the inherence factor by making the tokens biometric scanners. Instead of clicking on the device like today, the user would scan their fingerprint in order to unlock the device. This would make the authentication very secure as an attacker would need to get access to the device, know the password and be able to clone the fingerprint. Another difference between UAF and U2F is how they can sign transactions. As mentioned in Chapter 2.6, UAF is able to sign transactions while U2F is not due to the lack of screen. However, for our solution, this is not a problem as all signatures within the federation will be handled by the Signing Authority, as described in Section 3.2.

As the security differences between UAF and U2F are minor, we chose to use U2F in our implementation because of its availability. For a national eID system to work, it is important that a user can easily join and use it. U2F only requires a user to buy a U2F token and register it, while UAF requires a FIDO compliant device. [26]

One of the biggest drawbacks of FIDO is its relative novelty. It has not yet undergone deep security analysis from neither the open source community nor from academia. However, the published security analysis shows protection against many attacks. By using it in a federated identity system, one can add features such as blocking devices. We propose a solution for blocking a device in Chapter 4. It is also worth noting that FIDO is gaining support from big companies, such as Google and PayPal. Wide support as well as companies that have incentives for keeping the protocol secure will ensure that the protocol will be maintained. In Chapter 6.4, we analyze the security differences between FIDO and OATH and motivate why FIDO was chosen.

6.3 OATH analysis

The two different algorithms OATH-HOTP and OATH-TOTP, described in Section 2.7, propose methods to generate secure one time passwords. The two different RFCs define the specifications and how to implement a secure authentication protocol using OTPs. RFC 4226 contains a security analysis of the algorithm that generates the OTP. This analysis concludes that the truncation of the output from the algorithm does not provide any information that could be used to restore the secret key. Furthermore, it shows that the most effective way to perform an attack against the algorithm is to use brute force [29]. As mentioned before, two factor authentication is recommended to use for authentication for financial and governmental services. The two different algorithms can be used to achieve two factor authentication. Often, two factor authentication protocols use the possession factor and the knowledge factor as described in Section 2.4. Facebook and Google use OATH-TOTP with these two factors [28]. They are easy to develop both in hardware and software. The possession property comes from owning the device and only the owner having access to it. It would be possible to have a biometric authentication in order to access the token or software. A biometric authentica-

tion would most likely be more expensive than just the knowledge factor, such as a password. The software may also run on a device that has the possibility to perform biometric authentication.

OTP offers better protection against phishing attacks than static passwords [68], which can be used until they are changed. In a scenario where the attacker phishes passwords and the HOTP algorithm is used, they could authenticate once. With TOTP, the OTP can only be used during the valid time slot [69]. In a study, Schneier [70] shows that a time limited OTP is not enough to protect against phishing attacks. In a scenario where the attacker sets up an operation which uses the password in real-time, the phishing protection can be bypassed [71].

One way to combat this is to use CAPTCHA, which stands for Completely Automated Public Turing test to tell Computers and Humans Apart [72]. It is used to protect against computers performing automated operations. Leung proposes a solution called Extended CAPTCHA Input System [71]. Extended CAPTCHA can be used for stronger phishing protection as well as MITM protection. In this protocol, the OTP code is entered using a CAPTCHA input system. The input system is downloaded from the site with a preshared key. This key will be used to encrypt the input from the user. The input system is an application running inside the web browser and can be written for Flash. It provides graphical windows where the user can click on graphic numbers. The numbers are difficult for a computer to identify but easy for humans. After the application has loaded, the user inputs their OTP by clicking on the numbers with the mouse. Each click is recorded as a coordinate and time stamp. When the user has entered all the numbers, they are encrypted using the preshared key and sent to the server. To prevent the application from being relayed, the frame that contains the numbers is moved randomly when the application starts. This will record the coordinates in the user's browser, making them invalid for the attacker when they try to authenticate. To perform an attack, the attacker must stream their own application as a video. This requires large amount of bandwidth and computing power, making it an unfeasible attack. This protection may not hold forever because of the increasing power in modern computers.

As stated in RFC 6238, a short lived OTP is more secure than non-time based OTP [30]. In HOTP, a password is valid until it is used or any password generated after it is used. Given that the TOTP is only valid a short time, it provides greater phishing attack protection than HOTP. TOTP does require a more advanced OTP generator as it has to be synchronized. This feature also makes HOTP the more secure algorithm of the two. In Section 6.4, the OATH algorithms will be compared with FIDO both from availability and security perspectives.

6.4 Comparison FIDO & OATH

This section analyses the strengths and weaknesses of FIDO compared to OATH. This comparison also serves as the motivation why FIDO was chosen for our im-

plementation. First the criteria that we focused on are described, followed by the comparison and a conclusion.

Both protocols provide two factor authentication and are built on open standards without license fees. These two features were requirements for the authentication system we wanted to use. The OATH algorithm has been used over ten years and has not been broken yet. FIDO UF2 has existed two years. To evaluate the best solution for our system, we chose the following criteria:

- Strong protection against replay attacks.
- Strong protection against phishing attacks.
- Strong protection against man-in-the-middle attacks.
- Availability and deployment.

6.4.1 Comparison

Both FIDO and OATH have built-in protection against replay attacks. They solve the problem differently. In OATH the protection comes from the fact that the OTP is only valid for one authentication. After the password is used, the OTP becomes invalid. In a FIDO scenario the challenge provides the protection. A signed challenge is only valid until it has been received by the server. In this case there is no difference between FIDO and OATH.

As mentioned in Section 6.3, the phishing protection in OATH is good but it can be bypassed with a real time phishing attack. This can be mitigated with the use of Extended CAPTCHA Input System. FIDO has a strong protection against phishing built into the protocol. To accomplish this, FIDO verifies the server during authentication. This also provides protection against real time phishing attacks. As FIDO has stronger protection against phishing than OATH without the use of other systems, we conclude that FIDO is better.

FIDO also has the strongest MITM protection. This is because of the server verification during authentication. OATH does not have this feature. This could be mitigated with the use of Extended CAPTCHA Input System to receive similar protection. As with phishing attacks, we think FIDO has the strongest protection as the protocol by design protects against them.

The criteria availability and deployment is not about security but rather about usability and convenience for the user. OATH is a well-tested technology which is widely used to provide two factor authentication. Deploying a system that utilizing OATH is easy as there exists several server side solutions providing OATH authentication. As mentioned previously, one can use existing solutions for hardware and software tokens. The easiest solution to deploy is a software based system, where the key is generated on the user's device. Often in these systems, the key generation process does not take place in tamper resistant environments, making an attack on the device possible. Using a hardware solution is more secure but more

costly to implement as one needs to buy tokens for every user. In the FIDO case, the user must buy a U2F token. Deployment for FIDO is as costly as OATH with hardware tokens.

The tokens for both FIDO and OATH can be bought online. There are many libraries that can be used to integrate FIDO authentication on the server side. As of April 2016, U2F is only supported in Chrome but support in Firefox is under development. OATH on the other hand is supported everywhere a password can be inputted. Both FIDO and OATH are easy to deploy but FIDO is easier for the end user. FIDO still suffers from limited browser support which makes OATH better from availability perspective. However, as the use of FIDO spreads, the availability will increase.

6.4.2 Conclusion

Both FIDO and OATH provide strong authentication processes that are difficult to break. One strong argument for using FIDO is that companies using OATH today are founders of the FIDO Alliance. This indicates that they do not think OATH is good enough for authentication. FIDO also provides stronger protection against phishing and MITM attacks by design. OATH can add this protection by using Extended CAPTCHA input system. This will introduce other attack vectors, as it relies on the use of third party software running in the browser to execute its code. Attackers could exploit this software to gain access to the system. Based on what we found, the security of Extended CAPTCHA input system is only evaluated in the author's paper. It also seems like there are no real world implementation of the process. OATH's greatest strength is that it is well tested and easy to implement. There are no published attacks against the OATH key generation techniques. FIDO's biggest weakness is that it is rather new. The specification was finalised and published on December 9th 2014. Because of it being relatively new, the protocol has not been reviewed by many academic security researchers. We think that FIDO is more suitable for authentication in an eID federation. This is based on all the security features that are built-in the FIDO protocol as well as the ease of using it.

6.5 SAML 2.0 analysis

SAML 2.0 is an extensive framework, which enables a developer to make several design decisions within the specification. This puts a great responsibility on the developer as they need to know what implications the different choices have on the security. Because of the complexity, there are several published attacks on different implementations. In a paper that evaluates the security of SSO in general and SAML in particular, the authors discovered flaws in the implementation of SAML [73]. In the attack, the authors showed that it was possible to steal the users' cookies and get access to the user's Google resources. This attack exploits the fact that the relaystate often is an URL, which makes is possible to inject malicious code in. As the relaystate is not sanitized, the attacker can perform a Cross-Site Scripting attack. This is used to steal the cookie of the user, which grants greater

access than what the service was authenticated for. To evade the attack the solution was to sanitize the input. This removes the possibility to perform a Cross-Site Scripting attack.

In another paper [74], the authors investigate the implementation of XML signature method that is used in SAML 2.0. The XML signature method is more complex than older techniques like PKCS#7 [74]. In PKCS#7, the signed hash over the document is added to the end of the document. With a XML signature, the position of the signature is not specified, instead it can be placed differently inside the XML structure. This leads to that different versions of a document being evaluated to the same thing. In the study, the authors found that they could bypass the security of many SAML 2.0 implementations and authenticate as any user at the SP. The attacker utilizes a flaw for how the signature of the assertion is checked. In the attack, an old valid signature for another assertion is used to fool the software that the assertion is valid. The assertion gives the attacker full access to the target's resources. These two attacks show that SAML 2.0 can be vulnerable to attacks because of its complexity. This threat has been resolved when the authors contacted the developers and helped them solve the problem.

The federation has chosen to base the system on SAML 2.0, which is widely used. Keeping a technical infrastructure up to date is not the main purpose of the Swedish eID board and using a well spread technique is a good idea. By doing so, the board will only have to keep it updated and not create the updates themselves. In order to keep the system secure, it is crucial that patches are applied when they are available. It is important to either provide incentives and/or requirements so the SPs and IdPs update their systems. This can be done with legal requirements where the SP and IdP will take the financial risk if an attack leads to losses due to lack of updating. Another solution could be that the system can force the updates in order to stay functional. The first method is very efficient and can be compared to how credit card companies like VISA handle this problem. If a merchant does not support the use of the latest security (e.g. chip), they will be responsible for any losses due to fraud.

To conclude, SAML is an extensive framework with many configuration options. Having many options makes it customizable but also introduces many potential errors. These potential errors have introduced bugs and it is important that the federation updates the framework regularly.

Chapter 7
Discussion

This chapter focuses on our goals and discusses how they were achieved. We also look at future research possibilities within our and similar topics.

7.1 Goal: evaluate the security of open source protocols

The first goal of the thesis was to evaluate if it would be suitable to use open source software in a federated identity service. This goal was reached by looking at research on the security of open source in Section 2.3 and analyzing the results in Section 6.1. The research shows no conclusive answer to if open source or closed source code provides the best security. As the topic of software security is rather complex, it would be beneficial to use more methods, rather than just counting the bugs, the severity and the patching behaviour.

Another important aspect is the democracy aspect for an eID system. Whether to use open source software or not, should not just depend on the security itself but also who needs to trust the system. For a system that is to be deployed nationally and to be used by many citizens and governmental organizations, having it open source makes it more trustworthy. Even though most people will not review the code, it is important that they are able to do so if wanted. This can be compared to how decisions are made by governments. Few people will follow the debates and how the parliament votes in every question, but if needed, it would be possible for an external stakeholder to examine that everything is done correctly.

It is worth noting that the launch of the Swedish eID federation has been postponed due to security concerns from governmental organizations [75], for example the National Defence Radio Establishment (FRA in Swedish). These security reports have not been published, which shows that even when a system is open, not everything is open. We advocate having the whole system open to the public in order to enable external parties and users to analyze the security.

7.2 Goal: implement a proof-of-concept

We have succeeded in creating a proof-of-concept that implements SAML and FIDO. The proof-of-concept served as a testing environment where we could get

hands-on experience with the security in the protocols and frameworks, for example by implementing SQL injection protection. There are limitations in the implementation as it is only a proof-of-concept, with possible improvements being:

- Better user interface.
- Better integration with the IdP.
- Fully implemented issuing process.

These features would make the authentication system fully functional. In order to use the existing implementation, one needs to understand how it works to perform an authentication. A better user interface would guide the user through the authentication, which would increase security. The FIDO service could also be better integrated with the IdP service. As of now, the IdP and FIDO services are written using two different libraries for HTTP requests. If the FIDO service would instead utilize the same library as the IdP, the FIDO service could run inside the IdP service. This would simplify the communication between the implementation and the Shibboleth IdP.

Our proposed solution for distribution and issuing, as described in Section 4.2, is one example of how it can be done. Another solution could be that one applies at the tax office and receives the FIDO token in person. We think that our solution will be easier for the users. It will have a fair balance between convenience and security.

To verify our idea that open source can be used in an eID federation, we used different open source software and open libraries in our implementation. This makes the development easier as one can look at the library code during development. Most of our problems with open source arose around its documentation, especially the Shibboleth IdP documentation. The information in Shibboleth IdP documentation was often scarce and we had to turn to the Shibboleth mailing lists for help. This makes configuring the IdP software more difficult than it should be. As we did not work with any closed source code we can not comment whether it would be easier or not to work with. One would think that closed source is better documented but this may not be the case. This gives open source an advantage because one can look in the source code and draw conclusions from it.

7.3 Goal: evaluate security of our implementation

We reached the goal, which was to evaluate how well our implementation handles different attacks. The analysis is informal and covers the attacks we consider most critical to our system. If our system would fail to protect against these attacks, the consequences would be severe.

As the implementation is a proof-of-concept implementation, further development is needed before a complete security analysis can be done. This analysis should not only cover our implementation but the whole IdP solution. An analysis of

the whole solution is needed to verify the security when the different subsystems interact. This analysis should be done by a third party to certify that the implementation is secure. A third party analysis would also have more weight than an analysis of our own.

7.4 Future work

This thesis has focused on the authentication part of an electronic ID system. We have done a brief analysis of SAML 2.0 but it would be interesting to analyze this framework further. One could also do a comparison of the security in SAML 2.0 compared to OpenID Connect, another open framework.

Regarding the implementation, one possible next step could be a deeper integration towards the federation, for example by implementing the connection to the signing authority. This would enable a deeper analysis of the security differences between a certificate based signing process and a solution with a signing authority. Another possible improvement for the application could be to add support for UAF as it is a more flexible solution for the user. With an implementation of both UAF and U2F, it would be possible to do a comprehensive study on security differences.

We have not done any security analysis on the current solution, BankID, as it is closed source. But in order to draw a conclusion whether open source is suitable for an eID system, one needs to compare the security of BankID and open source solutions.

Another interesting topic that we have not reviewed is the security of open source software that are delivered by a for-profit vendor. Some of our findings suggest that the vendor has a great influence on the security. Would it be possible to get the best of two worlds by combining the benefits of an open source community and a strong vendor?

Chapter 8
Conclusion

The goal of this thesis was to evaluate the potential use of open source for the authentication in the new Swedish eID federation. By looking at different open source frameworks and protocols, we could investigate their security. Furthermore, we wanted to implement a proof-of-concept application for authentication. This gave us hands-on experience working with the open protocols and enabled us to do a more thorough security analysis.

The proof-of-concept application was implemented in Java using FIDO and SAML libraries. It supports registering a FIDO U2F device as well as authentication. The security of the application has been analyzed by looking at four potential attacks. In the analysis we look at how the application is affected by the attacks and how to mitigate them. The thesis also contains security analysis for SAML 2.0, FIDO, OATH and open source software in general. There is no conclusive answer to if open source software is more secure than closed source software. Further investigation is needed in order to get a more definitive answer. However, our application shows that it is possible to use open source in the Swedish eID federation. The thesis also raises the issue of trust and how open source can provide openness and trustworthiness, which are two important aspects of a national eID system.

Bibliography

[1] E. Commission. (2007). Electronic identities - a brief introduction, [Online]. Available: http://ec.europa.eu/information_society/activities/ict_psp/documents/eid_introduction.pdf (visited on 08/19/2015).

[2] P Madsen, "Liberty alliance project white paper: Liberty id-wsf people service-federated social identity, retrieved march 13, 2012 from http://www. projectliberty. org/liberty/content/download/387/2720/file," *Liberty_ Federated_ Social_ Identity. pdf*, 2005.

[3] Shibboleth. (2015). How shibboleth works, [Online]. Available: https://shibboleth.net/about/basic.html (visited on 07/21/2015).

[4] C. Herley, "More is not the answer," *IEEE Security & Privacy*, no. 1, pp. 14–19, 2014. [Online]. Available: http://research.microsoft.com/pubs/208503/MoreIsNotTheAnswer.pdf.

[5] *Common vulnerability and exposure*. [Online]. Available: http://cve.mitre.org/ (visited on 01/19/2016).

[6] *National institute of technology*. [Online]. Available: https://nvd.nist.gov/ (visited on 01/19/2016).

[7] E. Raymond, *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Media, 2001, ISBN: 9780596553968. [Online]. Available: https://books.google.se/books?id=F6qgFtLwpJgC.

[8] J.-H. Hoepman and B. Jacobs, "Increased security through open source," *Commun. ACM*, vol. 50, no. 1, pp. 79–83, Jan. 2007, ISSN: 0001-0782. DOI: 10.1145/1188913.1188921. [Online]. Available: http://doi.acm.org/10.1145/1188913.1188921.

[9] R. Anderson, "Security in open versus closed systems, the dance of boltzmann, coase and moore," *At Open Source Software Economics*, 2002. [Online]. Available: http://www.cl.cam.ac.uk/~rja14/Papers/toulouse.pdf.

[10] G. Schryen and R. Kadura, "Open source vs. closed source software: Towards measuring security," in *Proceedings of the 2009 ACM Symposium on Applied Computing*, ser. SAC '09, Honolulu, Hawaii: ACM, 2009, pp. 2016–2023, ISBN: 978-1-60558-166-8. DOI: 10.1145/1529282.1529731. [Online]. Available: http://doi.acm.org/10.1145/1529282.1529731.

[11] G. Schryen, "Security of open source and closed source software: An empirical comparison of published vulnerabilities," in *15th Americas Conference on Information Systems*, 2009. [Online]. Available: http://epub.uni-regensburg.de/21296/.

[12] G. Schryen, "A comprehensive and comparative analysis of the patching behavior of open source and closed source software vendors," in *IT Security Incident Management and IT Forensics, 2009. IMF'09. Fifth International Conference on*, IEEE, 2009, pp. 153–168.

[13] K. Scarfone and P. Mell, "An analysis of cvss version 2 vulnerability scoring," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09, Washington, DC, USA: IEEE Computer Society, 2009, pp. 516–525, ISBN: 978-1-4244-4842-5. DOI: 10.1109/ESEM.2009.5314220. [Online]. Available: http://dx.doi.org/10.1109/ESEM.2009.5314220.

[14] M. Shahzad, M. Z. Shafiq, and A. X. Liu, "A large scale exploratory analysis of software vulnerability life cycles," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12, Zurich, Switzerland: IEEE Press, 2012, pp. 771–781, ISBN: 978-1-4673-1067-3. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337314.

[15] G. Schryen, "Is open source security a myth?" *Commun. ACM*, vol. 54, no. 5, pp. 130–140, May 2011, ISSN: 0001-0782. DOI: 10.1145/1941487.1941516. [Online]. Available: http://doi.acm.org/10.1145/1941487.1941516.

[16] M. Burnett, *Perfect Password: Selection, Protection, Authentication*. Elsevier Science, 2006, ISBN: 9780080489513. [Online]. Available: http://books.google.se/books?id=18PMr6ra0UQC.

[17] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th international conference on World Wide Web*, ACM, 2007, pp. 657–666. [Online]. Available: http://www.ra.ethz.ch/CDStore/www2007/www2007.org/papers/paper620.pdf.

[18] B. Lakshmiraghavan, "Two-factor authentication," English, in *Pro ASP.NET Web API Security*, Apress, 2013, pp. 319–343, ISBN: 978-1-4302-5782-0. DOI: 10.1007/978-1-4302-5783-7_14. [Online]. Available: http://dx.doi.org/10.1007/978-1-4302-5783-7_14.

[19] M. Mannan and P. C. van Oorschot, "Using a personal device to strengthen password authentication from an untrusted computer," in *Financial Cryptography and Data Security*, Springer, 2007, pp. 88–103.

[20] F. F. I. E. Council, "Authentication in an internet banking environment," Tech. Rep., 2005. [Online]. Available: http://www.ffiec.gov/pdf/authentication_guidance.pdf.

[21] "Security assertion markup language (saml) v2.0 technical overview," Mar. 2008. [Online]. Available: https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf.

[22] "Assertions and protocols for the oasis security assertio markup language (saml) v2.0," Dec. 2009. [Online]. Available: https://www.oasis-open.org/committees/download.php/35711/sstc-saml-core-errata-2.0-wd-06-diff.pdf.

[23] "Security and privacy considerations forthe oasis security assertion markup language (saml) v2.0," Mar. 2005. [Online]. Available: https://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf.

[24] FIDO. (2015). About the fido alliance, [Online]. Available: https://fidoalliance.org/about/overview/ (visited on 05/12/2015).

[25] T. F. Alliance, "Fido 1.0 final specification have arrived," Tech. Rep., Dec. 2014.

[26] FIDO, *Universal 2nd factor (u2f) overview*, 1.0, 2015-10-09, FIDO Alliance, Oct. 2015. [Online]. Available: https://fidoalliance.org/specs/fido-u2f-v1.0-ps-20141009/fido-u2f-overview-ps-20141009.html.

[27] ——, (2015). Fido security reference, [Online]. Available: https://fidoalliance.org/specs/fido-u2f-v1.0-nfc-bt-amendment-20150514/fido-security-ref.html (visited on 04/05/2016).

[28] A. Wawro. (2013). How to set up two-factor authentication for facebook, google, microsoft, and more, [Online]. Available: http://www.pcworld.com/article/2036252/how-to-set-up-two-factor-authentication-for-facebook-google-microsoft-and-more.html (visited on 03/29/2016).

[29] B. M, F Hoornaert, D Naccache, and O Ranen, "Hotp: An hmac-based one-time password algorithm," Dec. 2005, RFC 4226. [Online]. Available: https://tools.ietf.org/html/rfc4226.

[30] D M'Raih, S Machani, M Pei, and J Rydell, "Totp: Time-based one-time password algorithm," May 2011, RFC 6238. [Online]. Available: https://tools.ietf.org/html/rfc6238.

[31] F. ID-teknik. (2015). Bankid statistik, [Online]. Available: https://www.bankid.com/om-oss/statistik (visited on 01/17/2016).

[32] Telia. (2015). Telia e-legitimation, [Online]. Available: https://www.telia.se/privat/bredband/tjanster/produkt/e-legitimation (visited on 01/16/2016).

[33] BankID. (2015). Bankid about us, [Online]. Available: https://www.bankid.com/om-oss (visited on 01/16/2016).

[34] S. Borell. (Aug. 2015). Bank-id-cava, [Online]. Available: http://wiki.fribid.se/sidor/BankID-Cava.

[35] BankID. (Sep. 2015). Bankid relying party guide, [Online]. Available: https://www.bankid.com/assets/bankid/rp/bankid-relying-party-guidelines-v2.9.pdf (visited on 01/16/2016).

[36] M. Holm, *Project title*, https://github.com/judofyr/bankid-api, Apr. 2013.

[37] K. Gjøsteen, "Public key infrastructure: 5th european pki workshop: Theory and practice," in. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. Weaknesses in BankID, a PKI-Substitute Deployed by Norwegian Banks, pp. 196–206, ISBN: 978-3-540-69485-4. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69485-4_14.

[38] S. eID board. (2015). Advertisement of a new free system, [Online]. Available: http://www.elegnamnden.se/svenskelegitimation/ansokantillvalfrihetssystem.4.10cbb69314111c2d94ba5be.html (visited on 07/16/2015).

[39] ——, "Technical framework for swedish e-identification," Oct. 2015. [Online]. Available: http://www.elegnamnden.se/download/18.77dbcb041438070e039d236/1432125895897/ELN-0600+-+Tekniskt+ramverk+f\%C3\%B6r+Svensk+e-legitimation.pdf (visited on 01/19/2016).

[40] R. Pearce, "Dead database walking: Mysql's creator on why the future belongs to mariadb," Mar. 2013. [Online]. Available: http://www.computerworld.com.au/article/457551/dead_database_walking_mysql_creator_why_future_belongs_mariadb/ (visited on 04/11/2016).

[41] OWASP. (2015). Owasp top 10, [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (visited on 04/10/2016).

[42] S. Christey. (2011). Sans top 25 most dangerous software errors, [Online]. Available: http://cwe.mitre.org/top25/ (visited on 01/11/2016).

[43] M. Jakobsson and S. Myers, *Phishing and countermeasures: Understanding the increasing problem of electronic identity theft*. John Wiley & Sons, 2006.

[44] N. D. D. G. Maria Vergelis Tatyana Shcherbakova. (2016). Kaspersky security bulletin. spam and phishing in 2015, [Online]. Available: https://securelist.com/analysis/kaspersky-security-bulletin/73591/kaspersky-security-bulletin-spam-and-phishing-in-2015/ (visited on 05/03/2016).

[45] K. Lab. (2013). The evolution of phishing attacks: 2011-2013, [Online]. Available: http://media.kaspersky.com/pdf/kaspersky_lab_ksn_report_the_evolution_of_phishing_attacks_2011-2013.pdf (visited on 05/03/2016).

[46] R. Dhamija and J. D. Tygar, "The battle against phishing: Dynamic security skins," in *Proceedings of the 2005 symposium on Usable privacy and security*, ACM, 2005, pp. 77–88.

[47] P. Gühring, *Concepts against man-in-the-browser attacks*, 2006.

[48] S. Rauti and V. Leppänen, "Browser extension-based man-in-the-browser attacks against ajax applications with countermeasures," in *Proceedings of the 13th International Conference on Computer Systems and Technologies*, ser. CompSysTech '12, Ruse, Bulgaria: ACM, 2012, pp. 251–258, ISBN: 978-1-4503-1193-9. DOI: 10.1145/2383276.2383314. [Online]. Available: http://doi.acm.org/10.1145/2383276.2383314.

[49] A. Mohaisen and O. Alrawi, "Unveiling zeus: Automated classification of malware samples," in *Proceedings of the 22nd international conference on World Wide Web companion*, International World Wide Web Conferences Steering Committee, 2013, pp. 829–832.

[50] D. J. Kevin Stevens. (2010). Zeus banking trojan report, [Online]. Available: https://www.secureworks.com/research/zeus (visited on 04/20/2016).

[51] D. Lawrence. (2015). The hunt for the financial industry's most-wanted hacker, [Online]. Available: http://www.bloomberg.com/news/features/2015-06-18/the-hunt-for-the-financial-industry-s-most-wanted-hacker (visited on 04/20/2016).

[52] F. Mercês. (2014). Uncovering malicious browser extensions in chrome web store, [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/uncovering-malicious-browser-extensions-in-chrome-web-store/ (visited on 05/04/2016).

[53] FIDO. (2015). Specifications overview, [Online]. Available: https://fidoalliance.org/specifications/overview/ (visited on 05/28/2015).

[54] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: Classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.

[55] J. S. Davis, "Sony psn downed; hacking group claims ddos attack," Jan. 2016. [Online]. Available: http://www.scmagazine.com/sony-psn-downed-hacking-group-claims-ddos-attack/article/463065/ (visited on 04/18/2016).

[56] P. Kudo, "Ny ddos-attack mot svenska medier," Mar. 2016. [Online]. Available: www.svd.se/just-nu-misstankt-dos-attack-mot-svenska-medier (visited on 04/18/2016).

[57] Z. Al-Qudah, B. Al-Duwairi, and O. Al-Khaleel, "Ddos protection as a service: Hiding behind the giants," *International Journal of Computational Science and Engineering*, vol. 9, no. 4, pp. 292–300, 2014.

[58] M. Prince. (2013). The ddos that knocked spamhaus offline (and how we mitigated it), [Online]. Available: https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho/ (visited on 04/08/2016).

[59] T. J. Times. (2010). Hackers tap online supermarket databases, steal customer info, [Online]. Available: http://www.japantimes.co.jp/news/2010/08/15/national/hackers-tap-online-supermarket-databases-steal-customer-info/#.VthHyNBF141 (visited on 03/03/2016).

[60] D. Goldman. (2012). Yahoo's password hack shows that it failed security 101, [Online]. Available: http://money.cnn.com/2012/07/12/technology/yahoo-hack/ (visited on 03/03/2016).

[61] J. Vijayan. (2009). Rockyou hack exposes names, passwords of 30m accounts, [Online]. Available: http://www.computerworld.com/article/2522045/security0/rockyou-hack-exposes-names--passwords-of-30m-accounts.html (visited on 05/02/2016).

[62] L. Larsson. (2011). Lösenorden kommer från bloggtoppen.se, [Online]. Available: http://computersweden.idg.se/2.2683/1.412262/losenorden-kommer-fran-bloggtoppense (visited on 05/02/2016).

[63] W. Halfond, J. Viegas, and A. Orso, "A classification of sql-injection attacks and countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, IEEE, vol. 1, 2006, pp. 13–15.

[64] A Kerckhoffs, "La cryptographie militaire (military cryptography), j," *Sciences Militaires (J. Military Science, in French)*, 1883.

[65] K. Scarfone, *Guide to General Server Security: Recommendations of the National Institute of Standards and Technology*. DIANE Publishing Company, 2009, ISBN: 9781437913507. [Online]. Available: https://books.google.se/books?id=EimzA3Fn12UC.

[66] I. Loutfi and A. Jøsang, "Fido trust requirements," in *Secure IT Systems*, Springer, 2015, pp. 139–155.

[67] D. H. Mcknight and N. L. Chervany, "The meanings of trust," Tech. Rep., 1996.

[68] X. Ruan, "Intel identity protection technology: The robust, convenient, and cost-effective way to deter identity theft," in *Platform Embedded Security Technology Revealed*, Springer, 2014, pp. 211–226.

[69] C.-Y. Huang, S.-P. Ma, and K.-T. Chen, "Using one-time passwords to prevent password phishing attacks," Apr. 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804511000427.

[70] B. Schneier, "Two-factor authentication: Too little, too late.," *Commun. ACM*, vol. 48, no. 4, p. 136, 2005.

[71] L. C. Ming, "Captcha in security ecis: Depress phishing by captcha with otp," Jun. 2011. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/1023697X.2011.10668241.

[72] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, in. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ch. CAPTCHA: Using Hard AI Problems for Security, pp. 294–311, ISBN: 978-3-540-39200-2. [Online]. Available: http://dx.doi.org/10.1007/3-540-39200-9_18.

[73] A. Armando, R. Carbone, L. Compagna, J. Cuellar, G. Pellegrino, and A. Sorniotti, "From multiple credentials to browser-based single sign-on: Are we more secure?" In *Future Challenges in Security and Privacy for Academia and Industry*, Springer, 2011, pp. 68–79.

[74] J. Somorovsky, A. Mayer, J. Schwenk, M. Kampmann, and M. Jensen, "On breaking saml: Be whoever you want to be," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 397–412.

[75] P. Mattsson and T. Larsson. (2015). Fra condemns eid federation, [Online]. Available: http://www.svt.se/nyheter/inrikes/fra-domer-ut-gemensam-e-legitimation (visited on 04/28/2016).

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2016-499

http://www.eit.lth.se