Multi-factor Authentication using Secure Elements
Enhancing the Usability with new Web APIs

Lars Tiedemann Thorsen
Master’s Thesis Spring 2016
Abstract

The number of Secure Elements (SEs) embedded in devices such as mobile phones, smart watches and physical security tokens are increasing. The cryptographic services these tamper-resistant micro-controller chips provide are excellent for use in multi-factor authentication mechanisms. However, there are known limitations with the current SE solutions in Web Applications. This thesis presents an analysis of the current solutions that explains some of their limitations, in addition to an analysis of some of the proposed Web APIs and protocols with the goal of increasing the overall usability of SE solutions on the Web. A Proof of Concept (PoC) Web Application was created for this Master’s project, using Fido Alliance’s U2F protocol for multi-factor authentication. The PoC provides a high degree of usability, without the need to install any additional software components when used with Google Chrome, as it is the first major Web Browser that has built in support for the U2F Web API. In addition, the thesis discusses how the PoC and its authentication mechanism ranks on a set of recognized security assurance level frameworks for entity authentication.
Acknowledgements

I would like to thank my supervisor Audun Jøsang for excellent guidance and feedback, especially during the final phases of this project.

A big thanks goes to Anders Fongen and Federico Mancini at FFI, for the great help and guidance they provided in the beginning of this project.

I would also like to thank my part-time employer EVRY for the great opportunity to work on a project related to topics brought up during this Master’s project.

A great thanks also to my fellow students Chris, Robin, Tormod and Bjørn for excellent support during this period and the opportunity to exchange knowledge, thoughts and experiences.

Finally, the biggest thanks goes to my family and wonderful girlfriend for supporting me through a stressful and hectic period, while also giving me the motivation to finish this thesis!
## Contents

1 Introduction ................................. 1
   1.1 Motivation ................................ 2
   1.2 Research Questions ....................... 3
   1.3 Method .................................. 3
   1.4 Related Work ............................. 4
   1.5 Structure ................................ 5

2 Background ................................. 6
   2.1 Secure Elements ........................... 6
      2.1.1 Form Factors ......................... 7
      2.1.2 Applications .......................... 7
      2.1.3 Protocols and Technical Specifications .... 9
      2.1.4 Available Programming Interfaces ........ 10
      2.1.5 SE Threat Model and Security .......... 11
   2.2 Multi-factor Authentication ............... 13
      2.2.1 Identification, Authentication and Authorization .................. 13
      2.2.2 Multiple factors of Authentication ........ 13
      2.2.3 Advantages over Single-factor ........... 14
   2.3 Security Tokens ........................... 15
      2.3.1 OTP devices ........................... 16
      2.3.2 USB “dongles” .......................... 16
      2.3.3 Smart Cards ............................. 16
   2.4 Asymmetric Cryptography .................. 16
      2.4.1 Applications ........................... 18
      2.4.2 Utilizing Digital Signatures in Challenge-Response Authentication ........................... 18
      2.4.3 Certificates ............................. 19
      2.4.4 Public Key Infrastructure ............... 19
   2.5 Web Application Security .................. 19
      2.5.1 Web Technologies ....................... 20
      2.5.2 The Web Browser ....................... 20
      2.5.3 Same Origin Policy ..................... 21
      2.5.4 Authenticated Sessions and Cookies ........ 21
      2.5.5 Threat Model ............................ 22

3 Analysis of Current Solutions ............. 26
   3.1 TLS Client Authentication ............... 26
      3.1.1 PKCS#11 ............................... 27
3.1.2 Browser User Interface ........................................ 28
3.1.3 Limitations .................................................. 28
3.2 Java Applets ..................................................... 30
  3.2.1 Java Smart Card I/O API .................................. 30
  3.2.2 Browser User Interface ................................... 31
  3.2.3 Limitations ................................................ 31
3.3 Browser Extensions .............................................. 32
  3.3.1 Example - SConnect ....................................... 32
  3.3.2 Limitations and Security Issues ......................... 34
3.4 Other Middleware Solutions .................................... 34
  3.4.1 Example - SCProxy ....................................... 34
  3.4.2 Example - Out-of-Band Solutions ....................... 36
3.5 Summary ....................................................... 38

4 Analysis of Future Solutions .................................... 39
  4.1 W3C Drafts .................................................... 39
    4.1.1 Secure Element API .................................... 39
    4.1.2 Web NFC API ........................................... 44
    4.1.3 Web Cryptography API ................................ 47
  4.2 FIDO U2F (Universal 2nd Factor) ........................... 49
    4.2.1 The FIDO Alliance .................................... 49
    4.2.2 Introduction to U2F .................................... 49
    4.2.3 Registration Process ................................... 51
    4.2.4 Authentication Process ................................ 52
    4.2.5 Cryptography and the Key Handle ...................... 53
    4.2.6 JavaScript Web API .................................... 54
    4.2.7 Achieving Driverless Experience with USB as Transport 54
    4.2.8 Mitigations against MITM and other Threats .......... 57
    4.2.9 FIDO 2.0 and Progress ................................. 58
  4.3 Summary ....................................................... 58

5 Design and Development of U2F PoC ............................ 60
  5.1 Functionality ................................................ 60
    5.1.1 Welcome Page .......................................... 60
    5.1.2 Login Page ............................................. 61
    5.1.3 User Registration Page ................................. 62
    5.1.4 U2F Device Registration Page ......................... 63
    5.1.5 U2F Device Authentication Page ....................... 64
    5.1.6 Members-only Page .................................... 65
  5.2 Architectural Overview ....................................... 66
  5.3 System Components .......................................... 66
    5.3.1 Backend ............................................... 66
    5.3.2 Client .................................................. 70
  5.4 Process View ................................................ 71
  5.5 Security and Limitations ................................... 71
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Different SE Form Factors</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Different types of Security Tokens</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Alice and Bob. Adopted from [128]</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>TLS Client Authentication Sequence</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Client Certificate Prompt (Firefox). Adopted from [26]</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of launching a smart card Java applet. Adopted from [116]</td>
<td>31</td>
</tr>
<tr>
<td>3.4</td>
<td>Gemalto (SConnect) PIV Card Sample Application</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>PIV Sample Application code snippet (VERIFY PIN)</td>
<td>33</td>
</tr>
<tr>
<td>3.6</td>
<td>Sending of an APDU command to SCProxy</td>
<td>35</td>
</tr>
<tr>
<td>3.7</td>
<td>APDU response returned from SCProxy</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>SE API - Layered view. From [99, slide 4]</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Code snippet from the SE API draft[104]</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Code snippet from a demo using the Firefox OS NFC API</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>Function declaration of u2f.register, as defined in [53]</td>
<td>55</td>
</tr>
<tr>
<td>4.5</td>
<td>Function declaration of u2f.sign, as defined in [53]</td>
<td>55</td>
</tr>
<tr>
<td>4.6</td>
<td>The life cycle of the different data formats in U2F</td>
<td>56</td>
</tr>
<tr>
<td>4.7</td>
<td>Format of a registration response message. Adopted from [34]</td>
<td>56</td>
</tr>
<tr>
<td>5.1</td>
<td>The Welcome Page</td>
<td>61</td>
</tr>
<tr>
<td>5.2</td>
<td>The Login Page</td>
<td>62</td>
</tr>
<tr>
<td>5.3</td>
<td>The Register Page</td>
<td>62</td>
</tr>
<tr>
<td>5.4</td>
<td>U2F Device Registration Page</td>
<td>64</td>
</tr>
<tr>
<td>5.5</td>
<td>U2F Device Authentication Page</td>
<td>65</td>
</tr>
<tr>
<td>5.6</td>
<td>The Members-only Page</td>
<td>65</td>
</tr>
<tr>
<td>5.7</td>
<td>Simplified Architectural Overview</td>
<td>66</td>
</tr>
<tr>
<td>5.8</td>
<td>Architectural Overview (including subcomponents)</td>
<td>67</td>
</tr>
<tr>
<td>5.9</td>
<td>Sequence of events during U2F device authentication</td>
<td>72</td>
</tr>
<tr>
<td>5.10</td>
<td>Technologies involved in each component</td>
<td>78</td>
</tr>
<tr>
<td>6.1</td>
<td>Result of comparison</td>
<td>88</td>
</tr>
</tbody>
</table>
List of Tables

2.1 APDU Command ........................................... 9
2.2 APDU Response ............................................ 9

4.1 U2F APDU Request Command ......................... 57
4.2 U2F APDU Request Command ......................... 57

5.1 The User Database table (with user example) .......... 67

6.1 Support for the major Operating Systems ............. 84
6.2 My caption .................................................... 85
6.3 Support for driverless experience ..................... 86
6.4 Need for third-party middleware ..................... 86
6.5 Support for mobile devices ............................. 87
1 Introduction

A press release from Eurosmart, “the Voice of the Smart Security Industry”, stated that the worldwide shipment of Secure Elements (SEs) was projected to be over 9 billion units in 2015 [36]. These micro-controller chips have been present in the form of smart cards for many years, but now the number of embedded Secure Elements in devices such as mobile phones, tablets and wearables (e.g. smart watches) are also growing considerably. Financial services is a distinctive contributor to the growth in the SE market with their use of SEs in millions of credit and debit cards, as well as the increasing number of mobile payment solutions such as Apple Pay[7] and Samsung Pay[102]. However, the press release from Eurosmart also describes a notable growth also in eGovernment and eHealth sectors, indicating an increased usage of Secure Elements for electronic Identity (eID) purposes, such as user authentication.

As Secure Elements can provide several cryptographic services like digital signatures, hashing and encryption/decryption, they are very suitable for physical security tokens used in multi-factor authentication mechanisms. Online services have utilized the capabilities of SE-enabled tokens in authentication on the Web for quite some time. However, the current solutions that provide this SE utilization to Web Applications often use technology that are increasingly discouraged or have known limitations.

This Master’s thesis presents an analysis of these current solutions for SE interaction with Web Applications, as well as analysis of some of the proposed and future solutions. It introduces the Proof of Concept (PoC) Web Application that was created to explore one of the future solutions analyzed, FIDO Alliance’s U2F (Universal Second Factor) protocol. The Web Application enables End Users to register U2F-enabled security tokens to their accounts, which can then be used together with knowledge-based factors such as passwords and numerical PINs to provide strong multi-factor user authentication. The goal of the PoC was to experiment with how future Web APIs can provide a pleasant user experience and still be considered to provide a high level of security. Selected usability characteristics of the PoC has been compared to those of several existing solutions in the Norwegian eID market. In addition, this thesis discusses how the authentication mechanism in the PoC complies with a couple of recognized security assurance level frameworks for entity authentication.
1.1 Motivation

Credentials consisting of usernames and passwords are still dominant in authentication mechanisms used on the World Wide Web. A major disadvantage with this type of knowledge-based authentication is the amount of separate services and systems that End Users need to remember their credentials for. This issue, combined with the many policies that e.g. force the users to change their password regularly, tend to make a lot of people reuse credentials or write them down [2]. The combination of weak passwords and password reuse has led to several account takeovers in the recent years [75, 61].

Many system owners and Online service providers try to increase the security with OTP (One Time Password) solutions. However, these solutions can also be vulnerable to attacks such as phishing [100] and have some usability issues. Another approach to securing services with second-factors, is to use security tokens such as smart cards that have a Secure Element chip. These kind of solutions does however have some well known usability limitations when it comes to usage with Web Applications, as previously mentioned. They often require third-party hardware and software components to be installed before first-time use, and often depended on discouraged technologies such as Java Applets to be used with Web Applications. To utilize the huge potential of Secure Elements in multi-factor authentication mechanisms, the usability of the current solutions must be improved or new approaches must be introduced.

The usability of a security mechanism is the key for customer acceptance, as stated in [83]. An article by Knight [68] also states that poor usability of a security measure leads to reduced usage. End Users can in addition be influenced by external de-motivators such as access to a competing solution that they perceive to have a higher degree of usability, but may be less secure [65]. This means that multi-factor solutions involving Secure Elements ideally should have an equal or higher degree than e.g. solutions based on single-factors, or pure software implemented security tokens not using tamper-resistant hardware.

There is a need for new, standardized Web APIs that gives Web browsers access to Secure Elements, and thus eliminates the need of Java applets, browser extensions or other types of middleware to utilize SEs. These APIs can be used to increase the usability of external tokens on Desktop computers, but is also very relevant for mobile devices which can include several SEs in various forms, e.g. SIM cards, microSD and embedded. By increasing the overall usability of the authentication mechanisms, hopefully more End Users and services will take advantage of these powerful micro-controllers for security purposes.
1.2 Research Questions

The following are the research questions defined for this Master’s project.

- How can the standardization of future Web APIs enhance the usability of Multi-factor authentication mechanisms that use SE-enabled security tokens?
- What are the main limitations of the current solutions for SE utilization on the Web?
- Can solutions based on future Web APIs contain the same level of security assurance as existing solutions and simultaneously provide a higher degree of usability?

1.3 Method

The research conducted in this project can be divided into the following phases:

- Theoretical study of Secure Elements and related topics
- Analysis of the current solutions for SE utilization in Web Applications
- Analysis of several proposed solutions for SE utilization in Web Applications
- Implementation of a Proof of Concept using one of the proposed solutions, FIDO U2F
- An evaluation comparing the PoC against similar solutions and security assurance level frameworks

To be able to really comprehend the topics introduced in this thesis, a theoretical study of relevant material was needed. Much information has been gathered from research articles and technical specifications, as well as from Online resources produced by e.g. vendors from the commercial and public sector. In addition, knowledge about fundamental security principles and applications have been obtained from textbooks and other learning resources used in various information security courses.

The analysis of current solutions for SE integration into Web Applications, has been conducted by reading through technical specifications and relevant learning material from the vendors involved. Several of the these solutions have in addition been tested “hands-on” to better understand the user
experience. Analysis has in addition been performed by inspecting traffic, events and source code in the Web browser.

As for the future solutions, the technical specifications for the different technologies and their corresponding APIs has been analyzed. It is worth noticing that since the start of this project, the development of some of the different implementations of the APIs has been ongoing. Since most of the APIs still have the status as drafts and/or have not yet been sufficiently standardized, time has been spent reading through message boards, Wiki pages and email-lists in addition to the available specifications. Some minor experiments where conducted, in addition to the implementation of the PoC that was based on one of the solutions analyzed.

After analyzing the FIDO U2F protocol, it was discovered that the process of standardization had gone far enough to be able to implement a working Proof of Concept. The development went through several iteration stages, and changes were made to take advantage of new possibilities that opened up during the process.

In the final phase of the research, an evaluation of the PoC was performed. The first part of the evaluation consisted of a comparison with similar multi-factor authentication solutions against a set of selected categories measuring End Device support. The second part of the evaluation compared the requirements of the authentication mechanism used in the PoC to those set in a couple of recognized security assurance level frameworks.

1.4 Related Work

Earlier this year, researchers at Google published a report from a two year study of their roll-out of Security Keys to all employees[72]. Security Keys is the same type of device that is used for the PoC in this thesis, and it uses the FIDO U2F protocol as well. Their report touches several of the topics that is brought up during this thesis, such as the status of current solutions and an analysis of the U2F protocol. However, the report only focuses on FIDO as a future solution, while this thesis will analyze several other of the potential future solutions in addition to the FIDO protocols.

RFID Authentication for the World Wide Web [73], describes a system using two-factor authentication for the World Wide Web based a RFID (Radio Frequency Identification) card. However, it does not take advantage of any functionality on the embedded microchip other than storing an UID on the tag to identify the different users. The thesis was written a decade ago, which explains the usage of a memory based microchip instead of a smart card with a Secure Element. The NFC technology that was in addition still in its early stages at the time of writing. The system was also depending
on locally installed middleware to be used with Web Applications, and thus moves away from the motivation for this thesis.

1.5 Structure

The thesis is divided into 7 chapters, where each chapter presents a separate topic.

The first, and current chapter is the introduction. It presents the research questions and describes the motivation for writing this thesis.

The second chapter introduces the theory that is relevant for getting a deeper understanding of the topics discussed in this thesis.

The third chapter presents the analysis performed on some of the current solutions for Secure Element integration using Web technologies.

The fourth chapter presents the analysis performed on some of the proposed, future solutions for Secure Element integration using Web technologies.

The fifth chapter describes the design and development process of the Proof of Concept implementation that was conducted for this project.

The sixth chapter presents and evaluation of the PoC based on a comparison with existing solutions in terms of End Device compatibility and support, as well as an evaluation of the PoC against a set of recognized security assurance level frameworks.

The seventh, and final chapter gives a brief summary of this thesis, and explains what conclusions can be drawn from the research. It then looks at how future research can expand the research presented in this thesis, and discuss what could have been differently.
2 Background

This chapter provides the necessary background material that enables the reader to have a better understanding of the topics brought up during this thesis. It begins by introducing Secure Elements, including a description of possible applications, communication protocols, interfaces and the Threat Model. Further, the concept of multi-factor Authentication will be described and some of the different types of Security Tokens will be presented. Then, a section is devoted to Web Application Security, before the final part of the chapter will introduce the field of research on the security and usability trade-off that is exist.

2.1 Secure Elements

There is commonly agreed definition of a Secure Element, but GlobalPlatform describes its core functionalities quite well[45].

“A Secure Element (SE) is a tamper-resistant platform (typically a one chip secure microcontroller) capable of securely hosting applications and their confidential and cryptographic data (e.g. key management) in accordance with the rules and security requirements set forth by a set of well-identified trusted authorities.”

The perhaps most commonly known type of SE is the chip found in smart cards such as credit and debit cards, which have been around for over three decades at this point. In fact, the term Secure Element and smart card chip can be easily confused as they both describe the same type of micro-controller chip. The term smart card is mostly used when talking about pocket-sized plastic cards or SIM cards used in mobile phones, while the Secure Element is perhaps more associated with the increasing number of secure micro-controller chips embedded to the motherboard of other devices.

In this thesis, Secure Element (SE) will be used as a common term for all of the different form factors of these chips, while smart card will be used when talking about the commonly known plastic cards containing a tamper-proof chip. The term chip will sometimes be used for the same purpose as Secure Element.
2.1.1 Form Factors

As mentioned, Secure Elements come in several shapes and sizes, also known as form factors.

Some popular form factors are:

- **UICCs (Universal Integrated Circuit Cards)** is more commonly known as the SIM cards we use with mobile devices today. Actually, the SIM (Subscriber Identity Module) is the application running on the chip and securely storing the unique subscriber identity number (IMSI), while UICC is the term used to describe the physical chip itself. UICCs often host other applications in addition to the SIM.

- **USB tokens** are devices with a USB interface that host a Secure Element internally. Some of these devices even have a removable chip [46], for instance in the form of a UICC. The USB device used with the PoC in this thesis, has an internal, non-removable chip and is also an example of a USB device that also supports NFC.

- **MicroSD cards** can contain Secure Elements, and are then often called *smartSDs*. Some of the newer card models have a NFC interface in combination with the standard SD (Secure Digital) interface [50], enabling them to add a Secure Element to for instance a mobile device.

- **Embedded SEs** is a term used to describe Secure Elements that are soldered to the motherboard of other devices. It has become an increasing trend to include Secure Elements in mobile phones[7][54] or even smart watches[7][55].

- **Smart cards**, here used to describe the typically pocket-sized plastic cards, represent one of the original applications for Secure Elements. They are for instance used for public transportation, physical access, gift cards, electronic IDs and credit/debit cards. Smart cards can have contact or contactless interfaces. There are even cards with both, called dual-interface cards, which utilizes the same Secure Element.

2.1.2 Applications

Secure Element chips have their own CPU, ROM, RAM, EEPROM and more. Some chips even have crypto-accelerators which enables them to perform strong cryptographic functions such as digital signatures and encryption. They run small Card Operating Systems (COS), which can host up to several applications in parallel. These operating systems can have their own directories and files, or share these with other applications. Some of the most famous operating systems for smart cards are MULTOS[80] and
JavaCard\[60\]. Chips with the JavaCard operating system, actually runs applets in a Java Card Virtual Machine (JCVM) on the chip.

There are numerous application software created for Secure Elements today. Each application is identified by there AID (Application Identifier). Some examples of applications running on Secure Elements are:

- Payment applications such as EMV (Europay Mastercard Visa), are perhaps the most widely used today. They provide both credit and debit transactions for customers using a chip embedded in a bank card in combination with a secret PIN. USA has for a long time been lagging in their usage of EMV cards, and the use of magnetic stripes is still dominant over embedded chips. After the so called fraud liability shift that took place in October 2015 we can expect increased usage of EMV ahead. This shift means that many stores are required to have Point-Of-Sales (POS) terminals that are EMV compliant.

In the recent years there has also been a rise of mobile wallet applications that utilize Secure Elements, like Apple Pay\[7\] and now recently Samsung Pay\[102\]. These wallet solutions are available today in both mobile phones and smart watches.

- Electronic identification (eID) applications are used to provide services like encryption/decryption, authentication and electronic signatures to End Users. Electronic identity cards are used in both commercial and governmental services. In Norway there are solutions where the SE resides on a typical smart card\[13\], as well as mobile solutions where the SE resides in the SIM (UICC)\[9\]. Several countries have started creating National eIDs \[101][12\], were one of the goals is to be able to securely identify citizens digitally, which can reduce identify theft and fraud.

- Other application types include ticketing (e.g. public transport,
concerts and events), coupons and employee badges.

2.1.3 Protocols and Technical Specifications

Common for most Secure Elements, regardless of their form-factor, is that their micro-controller chip is complying with the standard ISO/IEC 7816-4[59] which defines commands for interchange. These command-response pairs are commonly known as APDUs (Application Protocol Data Units) and have multiple fields of bytes that each have a distinctive meaning. Every APDU command must be followed by the corresponding APDU response before another command is to be sent.

Table 2.1: APDU Command

<table>
<thead>
<tr>
<th>Header (required)</th>
<th>Body (optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>INS</td>
</tr>
</tbody>
</table>

Where
- **CLA** (1 byte) is the instruction class
- **INS** (1 byte) is the instruction code
- **P1** (1 byte) is the first parameter
- **P2** (1 byte) is the second parameter
- **Le** (1 or 3 bytes) is the encoded length (Nc) of the data payload
- **Data** (Nc bytes) is the data payload to be sent
- **Lc** (1, 2 or 3 bytes) is the encoded length (Ne) of expected response data

Table 2.2: APDU Response

<table>
<thead>
<tr>
<th>Body (optional)</th>
<th>Trailer (required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data field</td>
<td>SW1</td>
</tr>
</tbody>
</table>

Where
- **Data field** (Ne bytes) is the response data
- **SW1** (1 byte) is the first status byte
- **SW2** (1 byte) is the second byte

Every session with the SE starts with an electrical reset of the card and the following ATR (Answer To Reset) response which gives some info regarding the card’s state, proposed communication parameters and more. After the ATR, the terminal typically send an APDU to select an application on the card identified by its AID. Then it can send APDUs to perform different operations that either are publicly open and available, or protected by for
instance a PIN credential. This is enforced by the chip’s own access control scheme.

### 2.1.4 Available Programming Interfaces

Secure Elements have different programming interfaces that can be used by applications to establish connection with them, based on the SE form factor and type of End Device.

**Desktop**

For desktop computers, the most common way to access a SE is via a *smart card reader* which is either built into the device or is an external reader. When the reader is built in, it is common that it has the correct drivers installed as factory default. External readers often need a manufacturer proprietary device driver to be recognized by the computer as a smart card reader. There are however exceptions that can enable “driver-less” experiences on some Operating Systems, like using a CCID class driver[115].

All major Operating Systems come with default support for the PC/SC (Personal Computer/Smart Card) programming interface. This programming interface can be used directly by applications to establish channels to the SE chip, or be used by higher layer programming interfaces such as PKCS#11 and 15 (see Section 4.1) supported by most OS, BaseCSP and MiniDriver (Windows) or *tokend* MAC OSX. These interfaces can be used by the OS specific *key stores*, and can enable applications to access key material directly from them instead of integrating with lower layer interfaces.

**Mobile**

On mobile devices, there are no standard that fits all Operating Systems. There are several different approaches to connect with the SE depending on the device type and SE form factor. External devices with with embedded Secure Elements such as smart cards, can be accessed by both native and installed Applications with OS and device specific APIs.

For embedded SE and removable UICCs like the SIM, there are also different approaches. The newer models of Apple’s iPhone and Apple Watch includes a SE that is used for their Apple Pay solution. Apple used a proprietary, native API to access the SE[7] which is not available to developers. The same goes for their support for NFC.
SIM cards can be accessed using a cellular network protocol called OTA (Over the Air programming), which actually sends hidden SMS messages to the phone[43]. This approach can be used to create out-of-band eID solutions that enables authentication and digital signatures using the SE (SIM). These solutions are discussed further in Section 4.3.

For other types of embedded or removable SEs in mobile phones, there is the Open Mobile API (OMAPI) from SIMAlliance[106], which has been standardized for Android. This API lets Applications interact with SEs available in the mobile device and send APDUs.

2.1.5 SE Threat Model and Security

Physical Attacks

Early generations of smart card chips had some security issues and were vulnerable for several different techniques for extracting protected data[69]. This included functions such as manual microprobing, glitch attacks, laser cutting and power analysis. These attacks are often referred to as side-channel attacks, as their cryptoanalysis is based on the physical implementation of a cryptosystem rather then using brute-force. As low-cost mitigations were presented, the security of the chips grew stronger and reached the point were the SE micro-controllers today are described as tamper resistant.

Passive Attacks

The communication channel can be exposed to eavesdroppers when using the contacts or contactless interfaces of a SE device. To mitigate this possible threat, the APDUs can be encrypted and by applied integrity checks using the concept of secure messaging, explained shortly.

Active Attacks

There is a possibility of an attacker actively trying to interface with the SE by for instance trying to performs DoS (Denial of Service) attacks. An example is an attacker trying to brute-force the PIN on a smart card with multiple VERIFY PIN commands to the chip. To mitigate these threats, most SEs implement an incremental PIN-try counter, that when exceeded blocks all access to the card. This mechanism could however be used by an attacker as a DoS, disabling a user from using his card before unlocking it. A possible mitigation against these kind of attacks can be performed by using a
PIN suspend mechanism specified for instance in the EAC (Extended Access Control) introduced in the BSI TR-03110 series[11].

Chip Security

The chips and card operating systems are normally certified according to the Common Criteria (CC) standards, and are evaluated according to their Evaluation Assurance Level (EAL). SEs normally rank between EAL 4 and EAL 5[114]. Some chip and COS manufacturers have an even higher focus on the security, and have reached the highest CC certification level of EAL 6+ for their products [47]. These products are normally intended for eID applications with very high focus on security and privacy of the chip holders.

To protect the messages (APDUs) transmitted to and from the card, the ISO7816 standards has defined the concept of Secure Messaging (SM). It provided this by ensuring both data confidentiality and data origin authentication. To use secure messaging, one or more security mechanisms has to be applied. The security mechanism includes the following [19]:

- an algorithm
- a key
- a security argument
- (often) initial data

The security attribute of a ISO7816 compliant Secure Element defines the allowed actions, as well as the procedures necessary to complete such actions. The security attributes can be associated to files and objects on the SE.

The security status represents the current state of the chip after either an answer to reset (ATR) or the execution of a single or several commands performing authentication procedures. These procedures can be

- proving the knowledge of a credential (e.g. with the VERIFY PIN command)
- proving the knowledge of a key (e.g. using GET CHALLENGE or EXTERNAL AUTHENTICATE)
- by using secure messaging features like message authentication

These security statuses can either be global, file-specific or command-specific.

Access control schemes such as BAC (Basic Access Control) and the earlier mentioned EAC are commonly used by National eID solutions. These
mechanisms where originally developed for the SE chip that resides in newer Passports in Europe [11].

2.2 Multi-factor Authentication

In the context of information security, authentication is to provide some form of proof that an entity is what it claims to be[62]. In addition to entity authentication, there is data authentication which provides the proof of the origin of the data. The term authentication is often confused with the terms identification and authorization, and some can have difficulty telling them apart. However, they have clearly distinct meanings.

2.2.1 Identification, Authentication and Authorization

Identification is to claim that you are someone. A real life example of this is when you are talking to somebody on the phone and you tell them what your name is. In theory, anyone can claim to be you, but hopefully only you can successfully authenticate yourself. Another option is that there can be a third party that determines the identity of a subject.

If you were to provide a secret passphrase to the person on the other side of the phone, and they could be sure that your were the only one knew this passphrase, they could trust that you were who you claimed to be. You would then have been authenticated.

Authorization is to specify the policy for making access control decisions. After creating the policy with authorization, access control is the process of deciding what you are allowed to do after you have been successfully authenticated. Access control takes place after identification and authentication. If the person on the other end of the phone in the example above were a bank employee, they could provide you with some information about your own account, but you would normally not be authorized to receive information about other members of that bank. If you requested for this information about another account, you would then be rejected.

2.2.2 Multiple factors of Authentication

The term multi-factor authentication is used to describe an authentication scheme were the End Users needs to present two or more separate pieces of proof to an authentication mechanism. These pieces are known to be of different factors.
There are three types of authentication factors that are commonly used:

- **Knowledge factors** are still the most commonly used form of authentication. This requires the user to provide something they know to the authentication mechanism. Passwords are perhaps the most used type of knowledge-based factor. Other types of knowledge factors are for instance **secret questions**, like “What is your mother’s maiden name?”. Basically anything that can be memorized can be used as a knowledge factor.

- **Possession factors** are something you have, or as the names suggests, something you possess. This type of authentication has been around for ages. The prime example of a type of possesion-based factor, is a key used with a lock. Only the ones that have the correct key to for instance a house, can open the door and enter. In information security, there are similar objects referred to as **security tokens**, which will be covered in more detail later.

- **Inherence factors** are something associated with a person. Another popular word used in this context is **biometrics** (metrics that are related to human characteristics). Fingerprints are perhaps the most known type of biometrics, but other modalities include retina scanners, facial recognition and voice recognition.

### 2.2.3 Advantages over Single-factor

By combining at least two of the factors mentioned above, we get multi-factor authentication. Single-factor authentication, is as the name implies, an authentication mechanism that only consist of one factor. On the Web, single-factor authentication has dominated in the form of the combination of a *username* and *password* for years.

The major advantage of multiple factors, is that an attacker trying to impersonate you needs more than on factor to succeed. For instance, if they manage to steal your password, they still need to get your physical security token or your fingerprints. If they manage to steal your physical token, they would still need to know your password, get your fingerprints, or a combination of both (three-factor authentication). This harmonizes well with the information security strategy *defence in depth*, where there are multiple layers of security that an attacker must break.

Two-factor is perhaps the most used of the multi-factor authentication mechanisms as of today, but there is no limit to how many, or which, factors that can be combined. A combination of *something you know* and *something you possess* is one of the most widely used combinations, and is also the combination that is the most relevant in the context of this thesis. There are
however for instance examples of combinations of inheritance and possession based factors, such as the Norwegian developed Zwipe credit card[131]. This is a credit card that replaces PIN codes with fingerprints that are read from an embedded reader on the card itself.

2.3 Security Tokens

A security token, in the context of this thesis, is a device that can be used to authenticate authorized users. Other well used names for such a device are hardware tokens, cryptographic tokens, software tokens and security keys. This thesis will use the term security token consistently to avoid confusion.

![Security Tokens](image.png)

(a) Yubico Neo Security Token [129]  
(b) OTP Device [113]

Figure 2.2: Different types of Security Tokens

Security tokens can roughly be separated into two categories

- **Disconnected tokens**, which are pre-programmed to output information that can be synchronized with for instance a software based replica counterpart on a validation server. The most common types of disconnected tokens are OTP (One Time Password) devices.

- **Connected tokens**, which can connect with another entity to exchange information and often process information received through an interface. Examples of connected tokens are smart cards, USB dongles, types of SD cards or software based tokens, implemented for instance as an App on a mobile device.

The security tokens can be used to provide multi-factor authentication. They exist in several form factors that are both hardware and software based. Some of the main types of tokens will be presented in the subsections below.
2.3.1 OTP devices

One Time Password (OTP) tokens, produce a fixed number of characters (often digits) that can be provided as part of a two-factor authentication mechanism. The sequence of characters must be hard to predict and the generation is thusly often based on cryptography. The party that validates the OTP output must be synchronized with the token. With physical OTP tokens, this synchronization with the validating party is often solved with a either a counter based or clock based solution [62].

OTP tokens can be implemented in a number of ways. Examples of existing implementations are:

- SMS-based OTP, where codes are sent via SMS to End User
- Mobile Apps, where code can be presented and/or generated
- Offline devices, that can have a display to output the characters

2.3.2 USB “dongles”

USB devices is also a popular form factor of Security Tokens. These devices often contain a Secure Element with an authentication application, but there are devices that can be used for tasks such as automatically deliver OTP codes to forms in Web pages.

2.3.3 Smart Cards

Smart Cards have already been mentioned several times during this thesis. They can perform strong cryptographic functions that can be used to sign authentication challenges. They depend on the End Devices having either a contact or contactless reader. These smart card readers can either be built in to the device or be external.

2.4 Asymmetric Cryptography

This part of the chapter will give an introduction to the concept of asymmetric cryptography and describe applications that are relevant for concepts brought up during this thesis, such as challenge-response authentication mechanisms.

The concept of asymmetric cryptography was first introduced to the world by Diffie and Hellman in 1976[32]. Asymmetric cryptography is a
security mechanism involving two distinctive keys, in contrast to symmetrical cryptography, which only operates with a single key [107, p. 94]. Asymmetric cryptography is also commonly known as *public key cryptography*. The concept is based around the concept of two types of keys; a *private key* and a *public key*. The private key is only known to the owner and must be protected from disclosure to others, while the public key is known and can be distributed to others. The two keys are algorithmically related, but the private key can not be derived from its public counterpart. The private key is used for decryption purposes, while the public key is used for encryption.

![Figure 2.3: Alice and Bob. Adopted from [128]](image)

The perhaps most common example is that of two people, Alice and Bob, wanting to communicate securely (shown in Figure 2.3). In this example, Alice has given her public key to Bob and is in possession of the corresponding private key. Bob can then use Alice’s public key to encrypt a message before sending the outputted ciphertext back to Alice. Alice will now be able to use the private key to decrypt the message from Bob, and read the plain text message intended for her. The two keys are related in a way that ensures that the private key is the only key that can decrypt data encrypted by its public key counterpart. The public key can not be used to decrypt its own messages, which means that if someone else is in possession of Alice’s public key, they can not simply decrypt messages that others have encrypted with it.
2.4.1 Applications

The concept of public-key cryptography has several applications. Depending on the type of application, there is involvement of either the transmitter’s private key, the receiver’s public key or both, which are used to perform a type of cryptographic function [107, p. 96].

Described below, are three common types of asymmetric cryptography applications:

- **Encryption and decryption** is an application which has already been mentioned. Here the sender encrypts the plain-text message with the receiver’s public key. The message can then be decrypted by the receiver using their private key.

- **Digital signature** is an application where the sender digitally signs the message using their private key. The signing is performed by applying a cryptographic algorithm to the whole message itself, or parts of it. This signature can be validated by others using the public key of the signer.

- **Key exchange** is a process where two sides cooperate to exchange a shared, secret key. There are several approaches to achieve this. They usually involve one or both private keys of the participating parties.

Next, we will see how the digital signatures can be used in a challenge-response authentication mechanism.

2.4.2 Utilizing Digital Signatures in Challenge-Response Authentication

*Challenge-response authentication* is a security mechanism where a party supplies a challenge, and the recipient party must provide a valid response. One of its simplest forms is password authentication. Here the challenge is a request for a valid password, and the response is the value of said password.

Another approach, is when a party generates some random data to be sent to another party. The originating party then expects a response that in some way involves the transmitted challenge. This is where the application of digital signatures can come in hand.

If the recipient signs the entire, or parts of the challenge data with their private key, the issuer of the challenge can verify the signature of the response using the recipient’s known public key. The public key must be known to the party generating the challenge, and it must be trusted that the owner of the key is who they claim to be.
2.4.3 Certificates

To be able to verify the identity of the key holders with a higher level of trust, digital certificates where introduced. These certificates includes information about the owner of the key in addition to the public key of the user itself. X509 is perhaps the most common certificate format today. These certificates can be self-signed or be signed by a trusted party called a CA (Certificate Authority), which are introduced in the next subsection.

This principle is used today in HTTPS, where a CA trusted by the Web browser issues a certificate. This certificate guarantees that the Web site you are visiting is the legit owner of the public key used to establish a secure TLS channel with your Web browser.

2.4.4 Public Key Infrastructure

PKI (Public Key Infrastructure) is defined in RFC 4949 [105]. It is the set of hardware, software, people, policies and procedures needed to create, store, distribute and revoke digital certificates based on asymmetric cryptography [107, p. 139]. The main components in a PKI system is the CA, Registration Authority and some type of lookup service to check the validity of certificates (CRL and/or OCSP).

PKI is often used in electronic ID (eID) schemes. Private and public key pairs are generated on the Secure Element, or alternatively outside of the chip and imported later on. The public key(s) are sent to the CA to get a signed certificate, often using a CSR (Certificate Signing Request). The signed certificate(s) are then imported to the SE. The SE can then perform cryptographic operations on the SE and present a certificate, which includes the public key, to parties that want to validate their identity.

2.5 Web Application Security

Strong user authentication offers little benefit if the Web Application accessed is vulnerable to attacks where the authenticated session can be stolen by an attacker.

This part of the chapter introduces the Web security model and presents common threats for Web Applications. Chapter 18 in Computer Security by Gollman [48] has a good introduction in to Web security, as well as resources from OWASP [92].
2.5.1 Web Technologies

The dominant protocol on the Web is HTTP (HyperText Transfer Protocol), which is used to send requests and responses between the Web Server and the Client (Browser). HTTP includes several methods, such as POST, GET, PUT and DELETE. The two most dominant methods that are used in Web pages, are POST and GET.

GET indicates a request for some resource on a Web Server. This resource is addressed with a host field (e.g. newspaper.com) and a request-URI (Uniform Resource Identifier) (e.g. /sport/2016/article.htm). The POST method can be used in a similar manner, but was primarily intended for posting messages and sending larger data in the body of the request. The Web pages that are returned in these HTTP responses are written in HTML.

HyperText Markup Language (HTML) is a language that is used to create and structure Web pages. HTML consists of several elements, which uses tags to indicate what type of objects are represented in the Web page document. These elements can say something about the structure of the document, such as the body and section, while other elements indicate the type of content present, such as img (image) and script.

In addition to HTML, JavaScript and CSS are known to be the most essential Web technologies. CSS are primarily used to describe the layout and design of the Web page, while JavaScript is a programming language that can be used to create more dynamic and interactive Web Pages.

The term Web Application is used quite often in this thesis, and refers to Web sites that can have quite complex and interactive user interfaces, often similar to a classic desktop application. This is in contrast to the older Web model, often referred to as Web 1.0, which consisted of very static Web pages with limited functionality.

2.5.2 The Web Browser

The primary objective of the Web Browser is to display Web pages. These Web pages are represented internally in the browser as the DOM (Document Object Model). The HTML pages that come in responses from the server are parsed into the DOM. The result is objects such as document.location, document.origin and document.cookie. These are objects that can then be manipulated by Web Technologies like JavaScript. If you for instance want to change the background color of the HTML body, you can manipulate the element in the DOM by writing e.g. document.body.style.backgroundColor = 'blue';.

Browsers performs access control in many cases, like when scripts are
executed on a page. They also try to enforce the principle of *Same Origin Policies* (described in the next subsection). In addition, browsers includes and manages *restrictions*, that for instance limits what APIs that can be used without the End User’s consent. An example is when users enter fullscreen video mode and most browsers trigger a pop-up asking the user if the Web page are allowed to do this. Another example is if a Web page is requesting to use *geolocation* to approximate your current location and you are asked if you allow this action.

Web browsers also manages *sessions* and *cookies*, which are introduced further in Section 2.4.4.

### 2.5.3 Same Origin Policy

The Same Origin Policy(s) (SOP) is a term used for policies enforced by browsers to protect application data, such as session identifiers and cookies, from outside attackers. A Web page or Web Application is identified by what the browser sees as the *domain* of the Web server.

The *origin* is defined by a *scheme (or protocol)*, *host* and *port* of a URL (Uniform Resource Locator)\[119\]. Documents retrieved from *distinct* origins are said to be isolated from each other. An example on how this policy is executed, is if a document received from *http://domain.org/page1* tries to access the DOM of a document from *https://domain.org/page2*. The browser will in this example refuse connections, since the two protocols/schemes (HTTP and HTTPS) does not match.

Other “rules” states that scripts may only connect back to the domain they originated from and that cookies are only included in requests to the Web server that created them.

The general intention of the SOP, is a result of the Threat Model of the Web; it is assumed that End Users may visit malicious pages. It is thus important to protect active sessions with legitimate/trusted Web pages from being accessed or interfered with by malicious/untrusted Web pages.

### 2.5.4 Authenticated Sessions and Cookies

The TCP protocol that is used to establish a communication session between the Web server and the browser does not provide the necessary authentication. It is often important that the End User at the client is properly authenticated as the originator of requests to the server. To achieve this, the principle of *authenticated sessions* was introduced \[48, p. 342\]. There are several approaches on how to establish authenticated sessions.
Authenticated sessions can be established at the transport layer using mutual TLS. Mutual TLS lets users in possession of a certificate and the corresponding private key authenticate to the server, and is described in detail Section 3.1.

Session identifiers (SID) can be created by the server and sent to the client to establish an authenticated session on the application layer. These session identifiers are included in every request to server, and sessions are said to be authenticated if the request includes the correct identifier. The following lists some of the separate methods for transferring the sessions identifiers between server and client:

- The SID can be sent using a cookie. The cookie is sent to the client via the Set-Cookie header field in a HTTP response. The browser then stores the cookie in the document.cookie field of the DOM. All succeeding requests to the server includes the cookie in the Cookie field of the header. The cookies are stored on the computer of the client, but are subject to the SOP of the browser, meaning that Web sites from one origin can not access cookies from a Web sites in another origin.

- The SID can be included as part of the URI in requests for resources on the server.

- The SID can in addition be included in a hidden field in an HTML form and sent as a POST parameter.

Cookies are perhaps the most used method for SIDs today. Malicious End Users and outside attackers may try to elevate their privileges by modifying cookies, also known as cookie poisoning. This can be avoided by making the sessions identifiers unpredictable and storing them in a safe place. The server can in addition encrypt the cookies and/or add message authentication codes to the sessions identifiers to protect the integrity.

Another well known attack regarding cookies is cookie stealing. If an attacker manages to steal the cookie (including the session identifier) of an already authenticated user, they may be able to take over the session. As mentioned in the beginning of this section, this means that a very strong user authentication using multi-factor mechanisms can be of little use if an attacker manages to take over the active session at a later stage. The stealing of the cookie can be performed in numerous ways, such as with XSS and XSRF attacks, explained shortly.

### 2.5.5 Threat Model

This subsection will introduce some of the well known threats on the Web today. OWASP (Open Web Application Security Project) regularly issues
a Top 10 list to increase the awareness of important security flows in Web Applications[39]. As of 2013, this list is the following:

1. Injection
2. Broken Authentication and Session Management
3. Cross-site Scripting (XSS)
4. Insecure Direct Objective References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-site Request Forgery XSRF
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

A selection of these threats, as well as some that are not directly mentioned, will be described in the next subsections, mainly based on their relevance for topics later in this thesis.

**TLS MITM**

Man in the Middle (MITM) is a term commonly used for an attack, where the attacker positions themselves between two entities that are communicating. The MITM can then intercept, eavesdrop or alter the communication as they choose. Many believe that such an attack is completely mitigated when using TLS to secure the communication, but there is still a possibility that an attacker manages to apply such an attack, here refereed to as a **TLS MITM attack**.

If e.g. an attacker manages to lure an End User to accept their certificate, they can set up two TLS *tunnels*; one between the attacker and End User, and one between the attacker and the legitimate Web server. When the Web server asks for credentials, the attacker redirects this request to the End User. The End User may not discover the MITM and will respond with their credentials to what they believe is the legitimate Web server, but which in reality is the attacker. The attacker can then continue to “proxy” the traffic between the two entities as they please.
XSS

Cross-site scripting (XSS) is a type of injection attack where a malicious script is injected into otherwise trusted Web sites[94]. It can be defined as an *elevation of privileges* attack, where the trust in the server is exploited [48, p. 347]. There are three common types of XSS attacks:

- **Stored XSS Attacks**, are attacks where the injected script is permanently stored on the Web server[94]. These type of attacks are common to see in e.g. *message boards* and comment sections of Web pages. The attacks often occur if a script is posted to a text field without proper validation and sanitation of the input. If the attack is successful, the Web browser will think that the message is a JavaScript and execute it every time an End User accesses the page.

- **Reflected XSS Attacks**, are attacks where the injected script is reflected off the Web server via another route. This route may for instance be a *phishing email* or an URL from a malicious Web site.

  A common approach is to exploit a vulnerability in an custom made error page, such as “404 - Not Found”, where the name of resource not found is presented to the End User[96] (e.g. “The page /article/page.html was not found on this server”).

  If an attacker created the URL https://www.Website.com/</script>malicious script...</script>, and the Web server did not properly sanitize the request-URL before displaying it in the browser, the script </script>malicious script...</script> could execute.

  The attacker can use social engineering to trick End Users to click on this type of link, and the script would automatically execute if the Web page is vulnerable to reflected XSS. The malicious script could for instance steal the cookie and transmit it to the attacker.

- **DOM Based XSS Attacks**, are a special kind of XSS attacks where the attacker tries to modify the DOM environment of the End User with an attack payload[95]. Contrary to the two other types of attacks, the response from the Web server does not contain the attack payload. The payload embodies itself in the browser runtime and can for instance change a parameter to a value that is not expected. For example, the attacker can set a parameter called *default* to their malicious script with https://www.Website.com/index.html?default=</script>malicious script...</script>. If the JavaScript code in the Web Application does not expect this value and fail to filter out this type of malicious content, it will process the </script>malicious script...</script> as it if where normal HTML content, and thus execute the script.
XSRF

Cross-site Request Forgery (XSRF) is an attack where the payload is executed with the permissions of the user [48, p. 350]. Contrary to XSS, the attacker can not see the response to the forged request[93]. The attacker lures targets to execute state-changing requests to Web Applications, like performing financial transfers or changing the user’s email address, by clicking a link$^1$. For these attacks to succeed optimally, the user has to be pre-authenticated and thus making it harder for the Web server to differ the action as coming from the legitimate user or attacker.

The attacker can in theory use both GET and POST method to perform the attack. Using GET, the attacker can for instance lure the user to click an URL that performs an action. If GET messages are blocked for performing user actions, the attacker can create a POST request in a hidden form on their malicious site (or plant this request at the legitimate site) which will be automatically triggered when loading the Web page. If this request is prepared and posted from the malicious site, the Same Origin Policy will in most cases stop this kind of attack$^2$.

---

$^1$The URL of the link can in addition be “obscured” by a popular URL-shortener, making it harder for a User to detect the malicious action

$^2$Web services can use CORS (Cross-origin Resource Sharing to allow specific (or all) origins to access resources using the Access-Control-Allow-Origin header
3 Analysis of Current Solutions

This chapter will introduce and analyze some of the different approaches for interacting with Secure Elements in Web browsers today. A common factor for all of the solutions presented, is that they all need to install some sort of software component on the device that will interact with the SE enabled security token. In addition, external chip readers and the security token itself often require a custom driver if they are to be connected to a desktop computer using USB. These factors can tend to decrease the overall usability, especially when it comes to first-time usage and adoption. The solutions that will be discussed are TLS Client Authentication, Java Applets, Browser Extensions as well as some other approaches.

3.1 TLS Client Authentication

TLS (previously known as SSL) in its normal operational mode, running for instance on a Web site, only authenticates the server by presenting their certificate to the client trying to access the Web site. With client authentication, there can be mutual authentication by also asking the client for its certificate and ask it to prove that it possess the corresponding private key. A security token, enabled with a Secure Element storing keys and certificates can be involved in this process.

Following is a step-by-step description of the sequence shown in Figure 3.1. Some details about the complete TLS protocol has been left out, and it is encouraged to go through the TLS 1.2 specification published by IETF[31] for the exact sequence of steps.

1. Client Hello
2. Server Hello, including the server certificate
3. Client Certificate Request
4. User prompted to choose certificate stored on Security Token.
5. The user may be asked to enter their PIN to access the chip (if enabled)
6. Client Certificate sent back to server, via Client browser
7. Key info, encrypted by the server’s public key (published in certificate received from step 2)
Figure 3.1: TLS Client Authentication Sequence

8. *Certificate Verify*, which includes the previous handshake message digitally signed by the private key of the Client Certificate.

9. Server verifies signature received in 7. Both sides calculate the symmetric key that will encrypt further communication, based on info exchanged in earlier steps.

10. *Finished Message* sent by Client, encrypted with symmetric key.

11. *Finished Message* sent by Server, encrypted with symmetric key. If both parties can decrypt and read this message, mutual authentication has succeeded.

Reading client certificates of Secure Elements, and performing cryptographic operations with corresponding private keys, can be achieved by adding a PKCS#11 module/library to the Web browser.

3.1.1 PKCS#11

The Public-key Cryptography Standards (PKCS) have defined some higher level interfaces and formats for physical tokens, such as the Cryptographic Token Interface Standard (PKCS#11) [70]. PKCS#11, also commonly known as “Cryptoki”, is a higher-level programming interface that provides cryptographic functions to be used with physical tokens, such as requesting and proving ownership of certificates. The libraries, also referred to as modules, provides the necessary mapping of the underlying APDU commands.
to higher-layer functions such as *GenerateKeyPair*, *Encrypt* and *Sign*. This provides cryptographic functions that can be used by applications, without having to consider the majority of different chip types with their distinctive, proprietary APDU commands.

A downside of PKCS#11 is that each COS provider often have different underlying APDU schemes, and thus have to create different PKCS#11 libraries. These libraries often need to be downloaded from the Web and installed per COS or chip manufacturer. Open Source libraries such as OpenSC[85] exist, which covers multiple card and chip types.

Another standard in the PKCS series that is often associated with Secure Elements, is the Cryptographic Token Information Format Standard (PKCS#15) [71]. PKCS#15 defines a standard layout for storing cryptographic keys, certificates and other security objects on the chip, independent from the chip manufacturer or Card OS. This standard is often used in PKI based eID schemes.

### 3.1.2 Browser User Interface

When a PKCS#11 library/module has been successfully configured in a Web browser, the End User can be prompted to present a certificate that resides in their security token.

The user interface depends on the type of Web browser and Operating System that is used. Some browsers integrate with the OS key store application, such as Microsoft Security and Apple OSX Key chain App. In some of these cases, the application delivers the user interface. If the private key is protected, the End User may be prompted to enter their PIN code to complete the cryptographic operations involved in the TLS Client Authentication process.

### 3.1.3 Limitations

The article “Practical Issues with TLS Client Certificate Authentication”, published in 2014, discusses several issues related to the usability and security of TLS Client Authentication[97].

There are known issues when using a more advanced, hierarchical PKI setting with intermediate CAs, when using the very popular web server Apache’s *mod_ssl* module. Authentication, in this setting meaning verification of the client certificate, will be successful even when presented a certificate signed directly by the Root CA or any other intermediate CA intended for other purposes than authentication. Many eID schemes chooses to have dedicated private keys and corresponding certificates for
authentication purposes. This issue could therefore be exploited by an attacker obtaining a certificate from another intermediate CA that shares the same root, and thus getting successfully verified. It is then up to the application to post-verify the certificate, which is not trivial if they for instance only check if the commonName attribute in the X509 certificate matches the one received in a form. The 2014 report recommends that the application in addition verifies the issuerName to check that the certificate indeed was issued by the correct intermediate CA.

Another issue is that the Certificate Message is sent before encryption has been applied. This can be a huge privacy issue if the certificate includes enough information to track an end user. However, this can be mitigated by using renegotiate of an existing TLS connection before Client Authentication is initiated.

It can be problematic that the browser or OS controls the user interface when for instance prompting the user for a PIN or listing available certificates. An usability issue persists if the End User by mistake selects the wrong certificate and the “Remember this decision” option is set. There are currently no standard JavaScript APIs enabling the End Users to clear the Client Certificate choice and TLS cache, leaving it up to the users to manually perform this operation in the browser settings. Another well known usability issue with these user interfaces, is that the End User often have to choose characteristics of the cryptographic protocols to be used themselves, such as the key length and algorithm. This can be difficult and confusing for the average End User.

Since not all browsers support integration with the operation system’s key store, the PKCS#11 modules would also have to be installed per browser.
manually. This is definitively not optimal from a usability perspective.

In addition, many enterprises prefer to put a proxy server in front of their application and back-end servers, which would result in the TLS-channel being terminated before its final destination, and thus breaking the end-to-end authentication and encryption.

3.2 Java Applets

Oracle defines a Java Applet as a “special kind of Java program that a browser enabled with Java technology can download from the internet and run”[88]. It is actually the Java bytecode that is delivered to the end users and is then launched from the Web page while being executed within a Java Virtual Machine on the client End Device. This is actually two separate processes. The Java bytecode includes the instructions for the Java Virtual Machine, and can be produced by other programming languages than Java. As long as the programming language can be compiled to Java bytecode, they can in theory be used to develop Java applets as well. Jython[64] and jruby[63], which are Java written implementations of the Python and Ruby programming languages respectively, support development of Java Applets.

Java applets run at very fast speeds, which was one of the reasons that they for a while was preferred over than standard Web technologies for some special purposes. Before browsers added support for hardware-accelerated graphics using technologies such as HTML canvas and WebGL, Java applets had a much larger advantage in speed. Java applets that are signed can in addition access some hardware on the computer, which can be used to e.g. communicate with a SE.

3.2.1 Java Smart Card I/O API

Signed Java Applets can have a direct access to smart cards or other tokens by use of the Java Smart Card I/O API[90], which lets the applet transmit and send APDUs to the Secure Element in for instance a smart card. The Smart Card API takes advantage of the underlying PC/SC API to discover the available readers on the system and to establish logical channels(see chapter 2.1) with the present security tokens.

Since Java can run on most operating systems today as a result of the concept of the Java Virtual Machine (JVM), Java applets can be considered to have a very wide cross-platform support. This is great in regards of reaching as many End Users as possible. Java is however not included in all
Operations Systems, which means that End Users have to manually install it in the cases.

### 3.2.2 Browser User Interface

The user interface of the applet can be embedded inside Web pages with the `object`, `applet` or `embed` tag. However, since the `applet` has been deprecated[118] and `embed` only works with Mozilla family browsers, `object` is now the recommended tag.

![Figure 3.3: Example of launching a smart card Java applet. Adopted from [116]](image)

As a security mechanism, the End User is always asked if they want to run the Java applet that is embedded in the Web page. The End User is in addition prompted with warning messages if the applet is untrusted. Untrusted applets does not have access to anything on the local computer, and can only access the server that the applet originated from.

### 3.2.3 Limitations

A downside of using Java, is that the user too often have to download and install updates to the JRE (Java Runtime Environment) before being able to run the applets in the Web page. In 2015, there were 7 new update releases[87]. This may not look overwhelming, but some end users may struggle with understanding the consequences of not updating and will keep postponing the installation of new updates. The security settings in the
browser may prohibit the execution of Java applets if not the latest version of JRE is installed, which forces the End Users to update.

As a result of these usability issues, in addition to reports from End Users not feeling safe with vulnerabilities in Java, several of the Norwegian eID providers has chosen to deploy Java-free versions of their solutions\[81, 14\].

The use of Java applets is increasingly discouraged\[86\]. Newer browsers such as Microsoft’s Edge, does not support the use of NPAPI (Netscape Plugin Application Programming Interface) plugins\[91\]. As result, Java applets are not supported because of their dependence on this technology. In a blog post dated January 2016, Oracle announced that they plan to deprecate the Java browser plugin in JDK 9 and will be ultimately removed from the Oracle JDK and JRE in a future Java SE release\[89\]. Developers are now encouraged to use Oracle’s Java Web Start technology. Java Web Start lets you launch full-featured applications outside the Web browser environment by clicking a link in the Web page.

### 3.3 Browser Extensions

Another way of accessing SE chips from the browser environment is by developing browser extensions, also known as plug-ins or add-ons. With browser extensions you can extend the features of the standard Web browser. Such an extension can access hardware components and talk directly to for instance a card reader with a smart card in it. This way you can send and receive APDUs directly from the browser environment and set up whatever process you want. As mentioned earlier, a Java Applet can be defined as a browser extension. This section however focuses on extensions that do not use the Java technology.

#### 3.3.1 Example - SConnect

The digital security company Gemalto’s product SConnect, is an example of such an browser extension. From the Opera add-on store, SConnect is defines as “a browser extension that enables Web Applications to enhance their security by utilizing security devices connected to a computer”\[103\].

Figure 3.4 shows a sample application developed by Gemalto that utilizes the extended functionality provided by SConnect. The sample application lets you interact with a PIV (Personal Identity Card)\[82\] card, and for instance display certificates present on the chip. SConnect enables connection with the lower PC/SC interface layer. The Web browser can then
communicate with smart cards that are present in a reader connected to the computer.

```javascript
// get PIN input value.
var pin = document.getElementById('PINTextBox').value;

// check PIN length validity.
if (pin.length == 0) {
    document.getElementById("auth-result").innerHTML = "<blink><font color='red'>Error</font></blink>";
    return;
}

// convert pin value to hex-string and add padding.
var paddedPin = StringToHex(pin) + "FFFFFFFFFFFFFFFF";
paddedPin = paddedPin.substring(0, 16);

// authenticate with card, send Verify PIN APDU.
pcsc.transmit("00020000" + "92" + "00" + paddedPin, readCallback);
```

Figure 3.5: PIV Sample Application code snippet (VERIFY PIN) [42]

Web Applications can use JavaScript to send commands to the card using APDUs. Figure 3.5 is a code snippet showing how the PIV application takes a PIN inputted via a Web form and constructs a VERIFY PIN command that is sent to the card.
3.3.2 Limitations and Security Issues

The problem with browser extensions is that you have to create a different extension for every type of browser, as there is no standard way of creating extensions that fits all. Every browser uses their own architecture and APIs to create extensions. One of the original APIs for plug-in development NPAPI has been increasingly deprecated by the biggest Web browsers, as mentioned in section 3.2.

The article *The Most Dangerous Code in the Browser* expose and discuss several critical security issues with browser extensions today[51]. A huge liability with browser extensions is their lack of privilege control, and as a result it threatens the privacy of both users and trustworthy Web Applications. It includes examples of how malicious extensions with elevated privileges can exploits trustworthy Web Applications, as well as how trustworthy but vulnerable extensions can be exploited by malicious Web sites. It proposes a new browser extensions security system the is based on mandatory access control.

The Web browsers can potentially limit some of these issues by only allowing extensions that are published through their respective extension stores to be installed.

3.4 Other Middleware Solutions

There are several other creative middleware-solutions to enable communication with SE enabled security tokens to Web browsers. SCProxy, used by the Norwegian eID-provider Buypass, is one of the newer approaches.

3.4.1 Example - SCProxy

SCProxy[14] is a solution where you install an application that runs a simple Web server bound to ‘localhost’ (IP address 127.0.0.1) and that runs silently in the background. Is was created to replace their Java applet variant that enabled the usage of Buypass eID (in the form of a smart card) to authenticate to public and private online services. In addition it can be used to digitally sign documents online.

The SCProxy application exposes a RESTful Web service that accepts, redirects and responds APDUs, and thus effectively works as a proxy between the Web Application and the smart card reader on the computer. This works because the locally installed application exposing the Web service
interface, has access to the smart card readers through the underlying PC/SC interface.

The APDU commands are sent in the body of the HTTP POST requests JSON encoded, as seen in Figure 3.6, and the corresponding APDU responses are returned in the HTTP response message as seen in Figure 3.7.

To be able to comply with the same-origin policy of Web browsers, SCProxy needs to add a self-signed certificate for `localhost` to the operating system key store. This is to be able to use HTTPS between the browser and SCProxy component. Web browsers do not support scripts that are delivered to a Web page using HTTPS to connect with other endpoints using normal HTTP.

In addition, the local Web server component uses CORS (Cross-Origin Resource Sharing). CORS lets the Web server limit what origin requests are accepted. SCProxy uses this feature to only allow requests from Web pages that originates from the Buypass domain.

In theory this will work with all Web browsers, as they are all able to work with standard HTTP requests and responses. This contributes to an advantage over e.g. browser extension solutions, as all the different Web
browser will work with this component. As mentioned earlier, browser extensions need to be developed for each type of browser, and with this solution all browsers can utilize the same component.

Limitations

Note that this solution still needs to install a software component to the local machine. The end user has to go through the installation process of the application, which again is not very optimal from a usability point of view.

3.4.2 Example - Out-of-Band Solutions

*Out-of-band* solutions for authentication, are increasingly popular. The principle of out-of-band authentication is that the actual authentication mechanism is performed on another device\(^1\) and over a separate channel than where the End User wants to log in. This means that the End User can e.g. use the browser on their desktop computer and perform authentication on their mobile device. When the authentication is successfully performed on the mobile, the authentication process will finish and the user is logged in on their desktop. In this solution private keys are in the possession of the owner, in stead of residing in the “cloud” (such as with BankID’s OTP solution).

*BankID for mobile*, is such a solution and is used with one of the primary eID vendors in Norway. It is a PKI based eID solution, with the End Users’ certificates and keys stored on the Secure Element in the SIM card of their mobile phones. When a user wants to authenticate using BankID for mobile, they enter personal data in the browser that is used to look up the device that is currently registered to the specific user. If a device is found, the solution uses OTA messages (described in Section 2.1) to automatically trigger the eID SIM application, where the user is guided through the authentication process. The private keys are protected by a PIN code that must be typed correctly before signing any challenge data. A backend service validates the signed data, and indicates to the active session in the browser that the authentication has succeeded.

The GSMA foundation, most famous for managing the mobile GSM standard, has started a global authentication initiative named *Mobile Connect* based on a very similar approach to that of BankID for mobile. Mobile Connect aims to create a secure universal log in solution, reducing the need for usernames and passwords on the Web\([41]\). In their solution, the

\(^1\)In some cases, such as when the user performs the login in their Mobile browser, the same device can be used for both steps of the process.
keys and certificates also resided in the SIM card. The difference between Mobile Connect and BankID for mobile, is that GSMA’s solution is a global initiative where Mobile Operators can opt in.

Another approach for out-of-band authentication is to create an App for Mobile Devices that can communicate with a NFC-enabled End Device with a Secure Element. This App can be triggered using one of the mobile OS push services or by other means. When the App is launched, APDU commands can be transmitted to the device to perform the necessary cryptographic operations. For devices that do not support NFC (such as Apple iPhone), external readers are available.

**Limitations**

A problem with the SIM based solution of BankID, is that the card is locked to each Network Operator. This means that if the End User wants to switch operator to switch to a more suitable subscription, they need to register this with their BankID-issuer manually in addition to receiving a new SIM card.[109].

The out-of-band solution, with the App reading an external device, still depends on the End User manually installing an App from the mobile OS’ respective App stores. If one of the future APIs presented in the next chapter was sufficiently standardized, it could be possible to eliminate the separate App and communicate with external (and/or internal) SE-enabled devices directly from the mobile browser.
3.5 Summary

Several solutions that enable Web Applications to utilize security services with Secure Elements exist today. Most of the approaches include extending the Web browsers capabilities, by either creating an extension specifically intended for smart card purposes, or utilizing the increased hardware access of Java applets.

Common for all of these solutions, is that they are all dependent on additional software to be installed on the device that are to be used. Several usability security and usability limitations with these solutions have been presented. In addition, several of the current solutions depend on outdated technology that are increasingly discouraged. Because of this fact, it is made clear that the Web could potentially gain from new solutions and approaches to enable Secure Element utilization.
4 Analysis of Future Solutions

This chapter will introduce some of the proposed standards and early specifications aimed to make it easier and more user-friendly to interact with Secure Elements through your favorite and/or native Web browser. It begins with an analysis of certain W3C standards before it introduces FIDO Alliance’s U2F concept. The latter is the one that has been given the most attention in this chapter, as this initiative is much further ahead when it comes to becoming a global standard.

4.1 W3C Drafts

The World Wide Web Consortium (W3C) is a global community that develops open protocols and standards to ensure the long-term growth of the World Wide Web [121]. Most of the specifications presented in this section are drafts published by W3C working groups and unofficial drafts. Unofficial drafts have no official standing with W3C and is not supported by the organization, and is merely public proposals of new potential standard specifications. When there is enough interest around a particular topic, typically after some member submission and workshops, W3C creates either a new Activity or Working group. After technical specifications have been through several cycles of revision and review, they can potentially advance to W3C Recommendation status and eventually be included in the major Web browsers. The full process is documented in W3C Process Document [122].

There are especially two specifications in different stages of the W3C process that are related to applications of Secure Elements. A description of each follows, where the main focus is on the first, as it is perhaps the most relevant in the scope of this project.

4.1.1 Secure Element API

The Secure Element API Unofficial Draft [125] defines a communication interface between a Web Application and a Secure Element. The specification makes no assumption about the Secure Element type, application domain, or physical media. As the term “unofficial draft” implies, it is a work-in-progress specification and has no official standing yet.
The SE API Draft specifies Secure Elements as micro-controller chips that comply with ISO7816-4 (APDU command/response protocol). This definition harmonizes well with the one that has been established earlier in this thesis. It also lists up use cases that are very similar to those mentioned in Section 2.1. These include:

- **Authentication**
  Online services can be protected by a strong security mechanism based on credentials which are stored and processed on a Secure Element. It also mentioned that Web based OSs (e.g. Chrome OS) may use the API to utilize the cryptographic functionality of Secure Elements in for instance VPN and email system applications.

- **Digital signature**
  Applications can use the SE to digitally sign document and other data with keys that resides in it. The cryptographic operations will take place inside the chip, and the SE API will handle the communication flow of commands and responses to and from the SE.

- **Payment**
  Payment applications, such as those based on specifications from EMVCo, can be used to secure online transactions. I can be used with the common smart card form factor used with all credit cards today, or a payment application residing in a SE on a SIM, microSD or embedded SE chip.

- **Credential provisioning**
  The last use case listed in the API specification, describes how it could potentially be used to install, update or remove for instance an entire application or credential material on an SE. Examples mentioned are online updating/renewal of public transportation cards and employee badges including X509 certificates.

**Connection with Underlying Technologies**

The SE API lives in the Web runtime of the Web browser, between the underlying API layers from the OS and the Web Applications (as shown in Figure 4.1). In this way it similar to browser extensions, with the difference that a W3C standardized SE API would be built into Web browsers and thusly gain from tighter revision and control.

On desktop devices, it would provide access to the PS/SC programming interface described in Section 2.1. PS/SC covers all the major desktop operating systems (Windows, Mac OS X and some Linux distributions) and is factory default on most of them. As PS/SC is the industry standard for smart card readers, this would cover a large amount of SE enabled security
tokens.

On mobile devices, there are several options for what could be the underlying layer. It could utilize the OMAPI, also mentioned in Section 2.1, which is supported for some Android devices, to access Secure Elements. Additionally, it could interconnect with devices’ native programming interfaces for NFC to communicate with external security tokens. In general, there is lack of standardized APIs for Secure Elements on mobile devices today except for the OMAPI. The SE API however, does not “see” the underlying layers, and it would be up to the Web browser developers/vendors to implement the API for each device type.

Figure 4.1: SE API - Layered view. From [99, slide 4]

API Interfaces and Connection Sequence

Below are some of the important interfaces and types described in the draft.

- **Navigator** interface: exposes the SE service
- **SecureElementManager** interface: provides access to Secure Element readers
- **Reader** interface: makes readers connected to the device available
- **Session** interface: a connection session to one of the available SEs
- **Channel** interface: opens a basic or logical channel[59] to the SE
- **SECommand** interface: APDU commands to be sent to the SE
• **SEResonse** interface: APDU commands responses received from the SE

• **SecureElementType** enum: the type of SE in the Reader (UICC, smart card, eSE, microSD, other)

• **ConnectivityType** enum: the type of connectivity (embedded, plugged, wireless)

The **Reader** interface is used to communicate with the SE. As smart card readers may be empty and NFC devices may not be in the field of reach, the Reader interface lists the available readers, and then returns if a SE is present or not for each of them. When the Web Application returns a Reader with a SE present, it can open a **Session** with it. After establishing a Session between the Secure Element and the Web Application, it can open a **Channel** with a specific SE application identified by its AID (see Section 2.1.2). The Channel object then includes the methods for sending and receiving APDUs to the card. Figure 4.2 shows the whole connection sequence in the form of a code example.

### Security Considerations

Some of the security concerns for this type of API, is that there needs to be an common access control scheme that fits both Secure Elements and the Web.

This draft suggests to implement the access control defined by GlobalPlatform in [44]. A list of access rules, including the AID of the SE Application and the **Trusted application identifier** of the Web Application running on the device, is stored on the Secure Element. The Trusted application identifier is the SHA-1 digest of the Web Application origin. The access rules are collected by an **Access Control Enforcer (ACE)** residing in the Web Application runtime. The rules are fetched (and usually cached) when a Web Application tries to access the SE. The ACE then checks if the requesting application is allowed to access the specific AID on the SE. The Access Rules can additionally store a filter on APDU commands that are allowed, and/or a simple boolean that switches between all or no authorized communication with the chip.

It is stated in the API draft specification that the GP Access Control can protect legitimate users using non-compromised devices from malicious Web Applications. However, it does not offer protection from compromised devices that does not implement the ACE correctly, and would then rely on the internal SE security mechanisms such as secure messaging and PIN protection described in Chapter 2.1.
The draft also specifies that to be eligible to access the SE API, the Web Application must use HTTPS with a valid, trusted certificate.

Progress

After emailing one of the co-authors of the SE API, useful information about the progress of this API was received.

The *SysApps Working Group* at W3C that was working on this API, has now been closed[120]. The specification is currently being reviewed at
GlobalPlatform under the working group WebApis-for-SE[99, slide 14]. They have recently re-published the API at [84]. In addition, it could also be a topic for the new W3C Working Group HaSec, which has recently been started[117]. It is expected that they also will be involved with Secure Elements and perhaps also this type of API.

One of the few vendors that have implemented an API inspired by this draft, is Firefox OS. Firefox OS claims to be the first truly open mobile platform and is built entirely on Web technologies [76]. As a result of their open attitude towards new initiatives, they tend to implement experimental drafts (such as W3C drafts) before others. Their implementation of SE API currently only defines a SecureElementType for embedded SE and UICC, which makes it difficult to experiment with. The SE API has been included under their NFC API since Firefox v2.2[77]. To demonstrate its potential, a sample mobile wallet app has been developed[29].

Another initiative that has implement a SE API based on Web technologies is Tizen[112]. Tizen OS is an open and flexible operating system under the Linux Foundation that is built with multiple profiles, such as Mobile, TV and Wearable (e.g. smart watches), to serve different industry requirements[111]. Their SE JavaScript API is currently an optional API for the Mobile and Wearable Web.

4.1.2 Web NFC API

The Web NFC API Draft specification [124] defines an API that enables selected use-cases based on NFC technology for Web Applications. It is published by the Web NFC Community Group[123], and is currently not on any W3C Standards Track. The goal of the Web NFC Community Group is to create a NFC API that is browser-friendly and fits well with the Web’s security model. The API will not expose the lower levels of NFC functionality, but concentrate on more of the higher level functionality that is safe for Web pages, at least in the first phases of the process.

Near Field Communication

Near Field Communication (NFC) is a technology that enables wireless communication between two devices at close proximity and operates at 13.56 MHz. Several SE enabled security tokens utilizes NFC for transmission, like the U2F token that was used in the PoC described later in this thesis. An increasing number of mobile devices supports NFC, and can be used to both read data from other devices and to emulate a passive NFC device.

The tokens, or tags, to be read are either passive or active. Passive tags does not have an internal power source. The reader emits a small electric
charge, which then creates a magnetic field. This magnetic field ultimately powers the passive device when it enters, using the principle of induction. A read is always performed when a tag enters the powered field. Before the two entities start transferring data, they first specify which technology they can understand. The technology can be divided roughly into five different types of tags (listed later in this section), three signaling technologies (NFC-A,B,F,V) and three modes of operation (reader/writer, Peer-to-Peer and Card Emulation)[38].

The NFC is based on the already existing RFID (Radio Frequency Identification) standard. Some RFID devices can be read with NFC, but RFID is not a part of the NFC standards. The development of NFC specifications is handled by the NFC Forum, which is a non-profit, collaborative organization. A complete list of their specifications can be found at [37].

There are five types of tags that must be supported by NFC devices to be compliant with the NFC Forum specifications:

- **Type 1 Tags**, are based on the ISO/IOC 14443-3A standard (also known as NFC-A) and can be configured as rewriteable, or read-only.

- **Type 2 Tags**, are also based on ISO/IOC 14443-3A and can be configured with mostly the same specs as Type 1. The main difference is that Type 2 tags support ant-collision when there are multiple tags in the same NFC field.

- **Type 3 Tags**, are based on the JIS (Japanese Industrial Standard) X 6319-4 (also known as FeliCia). These types of tags can have much larger memory than Type 1 and 2, and also supports anti-collision.

- **Type 4 Tags**, are also based on ISO/IOC 14443 but support both the NFC-A and NFC-B communication standards. The tag can also support ISO-DEP (Data Exchange Protocol) as defined by Part 4: Transmission protocol in the ISO 14443 standard series. ISO-DEP can be used to send APDUs over the NFC link, and is thusly well suited for Secure Element enabled tokens.

- **Type 5 Tags**, are based on the ISO/IEC 15693 standard. The specification for this tag was released October 2015 with the goal of enhancing the interoperability with some existing RF (Radio Frequency) technologies.

**Web NFC Functionality**

The Web NFC is as mentioned earlier, centered around higher level NFC functionality. The current version of the specification focuses on reading,
writing and storing data in the NDEF format. NDEF (NFC Data Exchange Format) messages consists of one or several NDEF records. These NDEF records contains a header that describes which type of data it contains, and the payload containing the actual data to be sent. The NDEF format is mainly used to store data such as URLs, business cards and email addresses to tags. When the NDEF message is received, the record headers are typically read to find out what to do with the information received, like automatically open a Web browser with the URL received or initiate a Bluetooth connection with another entity. The API specifies a Web NFC Record (which is part of a Web NFC message), which is a special type of NDEF record that uniquely identifies Web NFC content made for Web Applications from generic NDEF content.

At its current status, the Web NFC API can not be used to utilize Secure Elements. The reason why it is still mentioned in this part of the thesis, is that there are vendors who have implemented similar Web NFC APIs which are related to this specification. These implementation have chosen to include support for ISO-DEP (Type 4 Tag), which can be used to transmit and receive APDU messages from Secure Element enabled devices. Even though this functionality is not part of this W3C draft’s current specification, it can be optionally added in a future release.

The draft lists several possible scenarios where a NFC API can be used. Some of them are:

- Holding a device in close proximity to a passively powered tag (e.g. card) in order to read and/or write data.
- Holding two active devices (e.g. mobile phones) in close proximity to push ‘Web NFC messages’ from one device to the other.
- Holding two active devices in close proximity to initiate a connection using another wireless carrier (e.g. Bluetooth or WiFi)
- Card emulation
  1. With a Secure Element, like payments by holding your phone near a Point-Of-Sale terminal
  2. With host card emulation (HCE), like using your phone as an emulated hotel room key

**Security and Privacy**

A separate report on the security and privacy considerations of the Web NFC API was published January 2016[21]. It states that NFC technology involves multiple levels of security and discusses some of the relevant threats and possible mitigations. Examples of threats listed are:
- **Web page using the Web NFC API collects user data without the consent of the user**
- **Malicious Web page overwrites tags without user consent**

The security mechanisms are centered around the concept of permissions. As mentioned in Section 2.6, the Web security model assumes that End Users will visit malicious Web pages, and that Web pages thus can not be trusted. The same principle is used in this API. Web pages using the NFC API are not trusted, which means that the End User needs to be aware what the Web page is trying to do with NFC. Web pages need permission to use NFC, and several methods are discussed in the report. This can be that the Web Browser prompts the user and describes what the Web page is requesting to do with NFC, or that the origin id in a Web NFC message matches the origin of the Web page trying to read or write to the tag.

**Progress**

As mentioned, some vendors have started implementing NFC APIs to be used with Web technologies. Again, it is Firefox OS and Tizen that have taken the initiative to implement this draft. Both APIs have extended the functionality of the W3C Draft, and support the ISO-DEP technology that enables transmitting and reception of APDUs from Secure Elements.

The Mobile Wallet App for Firefox OS, mentioned in at the end of Section 4.1.1, uses this NFC App together with the Secure Element API to perform HCE (Host Card Emulation). Host Card Emulation is a concept where a device acts as a contactless device and can be read by NFC readers, as if they were for instance a smart card.

The most relevant application, is that these extended NFC APIs can be used to send commands to external security tokens\(^1\) from Web Applications. A code snippet showing how APDU commands are sent using Firefox OS NFC API implementation is shown in Figure 4.3, where the commands sending the APDU are marked in red.

**4.1.3 Web Cryptography API**

The Web Cryptography API\(^[126]\), produced by the Web Cryptography WG (Working Group), is a *Candidate Recommendation* API that lets you perform basic cryptographic operations in Web Applications. These operations include hashing, signature generation and verification, encryption and decryption. In addition to this, it also describes an API for applications

\(^{1}\) Devices that do not have an internal SE.
Figure 4.3: Code snippet from a demo using the Firefox OS NFC API [79]

to generate and/or manage the necessary key material to perform the kind of operations. Several browser have already started their own implementation of this API, including Firefox, Chrome and IE11/Edge.

The API is agnostic to the underlying implementation of key storage, but provides a common set of interfaces that allow Web Applications to perform the operations mentioned above. It also provides interfaces for key generation, key derivation and key import/export.

It has no direct relation to Secure Elements, but deserves a mention in this chapter as it could utilize objects such as key materials that are stored and retrieved from SEs.

The following is an example of a use case where an End User wants to decrypt some data intended for them inside a Web Application, utilizing a SE:

1. The Web Application sends the symmetric key used to encrypt the document with the End Users public key, to the SE.
2. The SE decrypts the symmetric key using the End Users private key.
3. The SE responds with the symmetric key.
4. The received symmetric key is used to decrypt the data using the WebCrypto API, and then displays the deciphered data to the End User.

In this example, the cryptographic operations are performed inside the chip. The encrypted data displayed inside the Web Application, can not be decrypted before a successful interaction with the SE. This example utilizes an SE API in addition to the Web Crypto API.
4.2 FIDO U2F (Universal 2nd Factor)

In this section the FIDO U2F protocol will be introduced and analyzed. The analysis will be mainly based on the *U2F v1.0 Specifications* [6]. During the work with the thesis, the *Fido 2.0 Specifications* were presented as a proposal to W3C. A smaller subsection will be devoted to this at the end.

4.2.1 The FIDO Alliance

The FIDO (Fast Identity Online) Alliance is a non-profit organization formed in 2012, with the goal of addressing the lack of interoperability among strong authentication devices, as well as the problems End Users face with creating/remembering multiple credentials[3].

Their core ideas are:

- ease of use
- privacy/security
- standardization

In addition to the U2F protocol described in this section, they lead the work on a second protocol, UAF (Universal Authentication Framework). UAF is a framework with the goal of a passwordless UX (User Experience). The protocol can replace today’s passwords by combining a registered device with e.g. biometric methods such as swiping a finger or speaking into a microphone. The protocol enables the combination of several different factors, such as PIN + biometrics.

4.2.2 Introduction to U2F

U2F allows Online services to increase the security of their current password infrastructure, by adding a strong second factor to user logins. They claim that the service can use simpler, and thus easier to remember (e.g. 4-digit PIN) passwords without compromising security, by adding a U2F Device (security key).

FIDO specifies that a critical factor for success, is that the U2F devices ‘just work’ with all types of client devices without the need for additional driver and middleware installation. This fits well with the goal of the other APIs that relate to Secure Elements, like the SE API mentioned above.

The U2F device can be embedded in multiple form factors. Examples which have been already specified are standalone USB devices, NFC devices
and Bluetooth LE devices. Theoretically, it can also be implemented inside a SE in the user’s client mobile device, or even as a pure software implementation. Physical tokens with dedicated Secure Elements are recommended, but not mandated.

The protocol includes functionality that gives Online services (referred to as the Relying Parties) the capability of discovering the type of U2F Device being used. They can then accept or deny the different Device types depending on their own policy. This is achieved by using Attestation Certificates. These certificates include information about the Device, such as its manufacturer, and can for instance be signed by a trusted third-party.

Regardless of the form factor, the protocol specifies that there should be a Test of User Presence (TUP). On the USB token, this is often solved by a button that the end user must touch before starting secure exchanges of data. On NFC devices, the TUP is considered to be performed when the End User taps the device against another device that has an active reader. The TUP is optional, and can be implemented based on the Online service’s policy. For example, the policy can state that a TUP is necessary when performing transactions exceeding a certain amount in an Online banking service.

In contrast to many other two-factor devices, U2F enabled devices can be used by multiple users. This means that a family can share a device that is permanently connected to their home computer. One device can in addition be used for an unlimited number of Online services or Applications.

This analysis is mainly focused around development of services and applications on the Web, such as Web Applications, but there are already API implementations for e.g. mobile devices that are running Android OS. In the scenario where someone wants to develop both a Web Application and for instance an Android App, and want to use the same U2F validation server, the concept of applicationIDs and facetIDs were introduced. One applicationID points to a single or several facetIDs. For example, a U2F service can have a facetID of https://login.mycorp.com/ for the Web Application, and the facetID value of android:apk-key-hash:<sha1_hash-of-apk-signing-cert> for an Android App[5].

The U2F specs can be divided into an upper layer and a lower layer. The upper layer specifies the cryptographic core, while the lower layers specifies how the clients will communicate to the U2F device over their respectively transport protocols (e.g. USB, Bluetooth and NFC)[6]. The upper layer remains unchanged regardless of transport type. At the lowest levels, there are exchanges of APDUs, as most tokens contain a Secure Element chip. FIDO has created a reference implementation for a JavaCard type SE, but there are other options as well.

2 Depends on the device’s implementation of key storage. See Section 4.2.5
There are two main events in the U2F protocol; *Registration* and *Authentication*.

The registration process is where you register and create a new key pair tied to an user account for a type of service (e.g. a Website). In the authentication process, you prove that you are in possession of the private key already registered by signing some data from the Relying Party.

The following two subsections will present a deeper dive into these events.

### 4.2.3 Registration Process

There are several steps to the registrations process, but this subsection will focus on the steps after the user has successfully registered in advance and now wants to register a U2F device (Security Token) to their account.

1. The Relying Part (server) generates a random challenge.
2. The Client (Browser) binds the challenge into a *Client Data* structure. This Client Data consists of the challenge itself, the type of request (register/authentication) and optionally the *TLS channel ID* (described in Section 4.2.7).
3. The browser then sends the server’s origin and a hash of the Client Data to the U2F Device.
4. The U2F device then generates a new asymmetric key pair and a corresponding *Key Handle*. The Key Handle is used to recover the private key in the Authentication process, and is stored by the Relying Party. This step will be covered more thoroughly in a separate subsection.
5. The U2F Device associates this key pair with the Web origin received by the browser. It returns the *Attestation Certificate*, *public key*, *Key Handle* and finally a *digital signature* over the Web origin, hash of Client Data, public key and the key handle.
6. The Browser sends the received data back to the Relying Party.
7. The signature is verified and the public key and Key Handle is associated with the account that triggered the registration.

The next section will focus on what happens inside the U2F Device during the registration process.
Inside U2F Device

After the device has generated the key pair, it performs a store operation to store the private key for the application parameters (Client Data) received. This store operation produces the Key Handle. The public key, Key Handle, challenge from the server and the application parameter is then signed by the device’s attestation private key. The output of this is operation is the Registration Message response, and consist of the public key, Key Handle and the device attestation certificate, including the attestation private key’s public counterpart. This message is then sent back to the browser.

4.2.4 Authentication Process

Similarly to the Registration process, this subsection will focus on the steps after the user has been authenticated by other means, and now needs to prove the possession of a previously registered U2F-enabled device.

1. The Relying Party sends one or several Key Handles that have been registered to the pre-authenticated user, together with a challenge to the Client (Browser)

2. The Browser generates the Client Data, like in the Registration process previously. It then sends the hash of the Client Data together with the user’s Key Handle(s) and the Web Origin to the U2F Device.

3. Upon reception, the device will do a check on the Key Handle. The device will reject the request if it does not recognize it or if it is not associated with a previously registered Web origin. Else, it will recover the private key and continue with a signature.

4. The U2F device then provides a signature over the Client Data and some additional attributes, such as an optional TUP (Test of User Presence) and a counter.

5. The Browser forwards the signature, TUP and counter value back to the Relying Party.

6. The Relying party then validates the signature with the public key previously registered to the user. If the validation is successful, the Relying Party can state that it has authenticated the user’s possession-based factor.

As with the registration process, the next part will describe the sequence of events inside the U2F Device.
Inside U2F Device

After receiving the three parameters (application parameter, key handle, challenge), the device performs a retrieve operation. This operation uses the Key Handle to retrieve the private key and the stored application parameter for the particular user. As mentioned, a check is performed on the Key Handle received. If there is an error, the device returns the exact same error message for the two different outcomes (*unknown Key Handle* and *Key Handle does not match application parameter registered*). This is to prohibit an Online service from learning that a Key Handle is valid for some other Online service. It performs the digital signature over a concatenation of the incremented counter value, TUP and the provided challenge. In addition to the signature over all parameters, it returns the TUP and counter separately back to the browser.

4.2.5 Cryptography and the Key Handle

The *Security Keys Report*[72] states that all signing operations are performed with *ECDSA* using the *NIST P-256 curve*. For hashing operations, SHA-256 is used. The choice of signing algorithm, corresponding curve and the hashing algorithm was based on their increasingly wide availability on embedded devices, as well as on the basis of their cryptographic strength.

The Key Handle

There are several, different approaches for dealing with Key Handle introduced earlier. The Key Handle is used to retrieve the private key that performs signatures during the Authentication Process.

The Key Handle can be as simple as an index in a database storing the actual private keys on the U2F Device. There are however both practical and privacy issues with such an approach. Since the keys are usually stored on Secure Element chips, storage capacity is usually limited and the number of private keys that could be present at the same time would be low. In addition, if the index is predictable, it could predict the number of accounts it is used for.

In the U2F reference implementation, *key wrapping* is used for the Key Handles. In this approach, the store operation encrypts private key and application parameter using a secret, symmetrical key only known to a single U2F device. The retrieval of the the private key then only becomes an encryption/decryption (*wrap, unwrap*) operation. The storage of keys then becomes practically unlimited. As an added security mechanism, a different secret key is used to encrypt the application parameter before it
and the private key is encrypted together with the wrapping key. This is to mitigate known plain-text attacks, as the application parameter would always be known and could help someone trying to break the encryption with cryptoanalysis.

However, the wrapping mechanism above does store the private key of the End User on the server, though encrypted. Some may argue that this is not optional from a security and privacy perspective, as the key performing the digital signatures is not in possession of the user. It is however up to the different vendors how they will implement the wrapping mechanism. The U2F Device distributor Yubico, has for instance chosen their own approach to the key wrapping which does not store the private key itself on the server [27].

4.2.6 JavaScript Web API

The registration and authentication operations are available through a Javascript API, which the FIDO Alliance hopes will be built into all the major browsers in the future.

The higher-level functions of this API, intended to be used by developers are:

- **u2f.register()**, accepts a challenge and a list of pre-registered Key Handles. The list is provided to avoid double registration of a device. After discovering and exchanging data with an available U2F device, the method returns the device’s *Registration Method* output in addition to the Client Data mentioned earlier, in a JSON encoded object. This information can then be sent to the server for validation. The function declaration for u2f.register(), as found in the official u2f-api.js, is displayed in Figure 4.4.

- **u2f.sign()**, also accepts a challenge and the already registered Key Handles. After a successful signature operation on one of the available, pre-registered devices, it returns the signature and Client Data which can be sent back to the server for validation. See Figure 4.5 for the function declaration.

4.2.7 Achieving Driverless Experience with USB as Transport

One of the biggest selling points of U2F USB Security Tokens, is that they in theory work without the need to install drivers on all major Operating Systems. To achieve this, the security tokens use the USB HID (Human
The Interface Device class, which is normally used to identify device classes such as keyboards, mice and game controllers. Drivers for such devices are built into Operating Systems such as Windows, Mac OS X and Linux by default, and thus eliminates the need to download and install drivers manually. Programming interfaces for applications to interact and exchange data with these USB devices, are also standard on most Operating Systems. Chrome, and other browsers in the future, use these APIs to automatically discover when the Security Token is connected and start communicating with it.

The format of the data that is finally sent to the tokens, changes several times on its journey from the Relying Party, through the Client (Browser) and over a specific medium such as USB, NFC or Bluetooth. FIDO has specified the formats and encodings from the Client side to the U2F devices,
but leaves it up to implementors how the data is sent from the Relying Party to the Client. The life cycle of the different data formats can be seen in Figure 4.6.

The high-level JavaScript API functions (u2f.sign() and u2f.register()) is given JSON encoded objects. The Client will then prepare and format the data to be sent to the token using the Raw Message Formats, specified in [34]. The specifications uses the term framing for the process where the messages are to be sent to a lower layer. At the heart of this process, are the registration and authentication request and response messages that are encoded into bytes. The Raw Message Formats specification specifies both sequence and number of bytes for each field that is present (see Figure 4.7 for an example message). These messages are then sent as the payload of extended APDU command and are expected to be received in the APDU response, in addition to the status bytes:
Table 4.1: U2F APDU Request Command

<table>
<thead>
<tr>
<th>Header</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLA</td>
<td>INS</td>
</tr>
<tr>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>Lc1 - 3</td>
<td>U2F Raw Request</td>
</tr>
</tbody>
</table>

Where

**CLA** is reserved for underlying layers

**INS** is the U2F command code

**P1, P2** are defined by each U2F command

**LC1 - 3** are the bytes indicating the length of the request data

**U2F Raw Request** is either a registration or authentication request message in the U2F Raw Message Format, sent to the U2F Device

Table 4.2: U2F APDU Request Command

<table>
<thead>
<tr>
<th>Body</th>
<th>Trailer</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2F Raw Response</td>
<td>SW1</td>
</tr>
<tr>
<td></td>
<td>SW2</td>
</tr>
</tbody>
</table>

Where

**U2F Raw Response** is either a registration or authentication response in the U2F Raw Message Format, received from the U2F Device

**SW1, SW2** are the two status bytes

This data can then be “framed” for transport using one of the mediums listed in the specifications, namely USB, NFC and Bluetooth BLE. Their individual specifications includes the description of the protocol needed to comply with U2F.

### 4.2.8 Mitigations against MITM and other Threats

When the **TLS ChannelID** is added to the Client Data in the protocol execution, the server can discover a potential TLS Man In The Middle (MITM) attack. The concept of TLS ChannelID is presented in an IETF draft[8] and enables the server to detect if there are two separate TLS channels, indicating the probability of a MITM attack. Google and U2F are one of the first to have implemented this concept. However, a report from 2014[67] indicated some issues with the present ChannelID approach, and proposed some changes to the implementation.

As mentioned in the beginning of this section, one of the primary ideas of FIDO is security and privacy. A separate FIDO specification has been dedicated to security considerations[74].
The presence of the signature counter mitigate against the possible cloning of hardware based implementations, such as pure software implementations of an U2F Authenticator. If the counter value is detected to be less than the previous registered value, it is highly possible that an attacker may have managed to copy an U2F device/Authenticator.

The attestation certificates mentioned earlier, are proposed to be issued in batches to prohibit potential privacy leaks. Some additional approaches to attestation is mentioned in Section 7.1 of the Security Keys Report.

A U2F device does not have a global identifier that is visible across services or Web Applications, meaning that it can not be pointed at a Web site if found/stolen to see if an account has been registered with this device.

### 4.2.9 FIDO 2.0 and Progress

The registration and authentication operations are now available through a Javascript API in Google Chrome, which the FIDO Alliance hopes will be built in natively in browsers in the future. Just recently, they submitted three of their specifications, including APIs tied to U2F, to W3C as the Fido 2.0 specifications[127]. This could be done because some of the members of FIDO (e.g. Google, Paypal and Microsoft) are also members of W3C.

The new FIDO 2.0 APIs use a different JavaScript API than the U2F JS API presented earlier. The new APIs will include support to replace passwords with biometrics, thus blending the UAF and U2F protocol.

At this time, only Google Chrome has added support for the U2F API to their browser natively. Both Firefox and Microsoft Edge has announced that they plan to add support for FIDO’s protocols in the future [56, 78]. Google, LastPass and GitHub some of the vendors that have implemented support for U2F as a strong two-factor login mechanism. Google has deployed U2F security tokens to 50 000 employees and their two-year study [72] reports of a major increase in usability related to authentication.

### 4.3 Summary

This chapter has presented several solutions, that if in the future are picked up further by the W3C Process, could be built in to all major browsers. This could potentially eliminate the need for middleware on End Devices to interact with Secure Elements in different form factors, and thus increase the usability of multi-factor authentication mechanisms utilizing SEs in Web Applications.
The SE and NFC API has still a long way to go before being standardized by W3C, but can gain from more browser vendors and Web based OSs implementing these APIs.

The U2F protocol is more mature, as it is has already been implemented and built into one of the major browsers. In addition its progress is driven by the FIDO Alliance, which has several of the largest IT companies as contributors and members. The progress of U2F was found to be significant and mature enough to create a PoC (Proof of Concept) for this Master’s project, that utilized this protocol for two-factor authentication.
5 Design and Development of U2F PoC

This chapter will present the Proof of Concept Web Application that was developed for this thesis. First, the functionality provided by the PoC will be presented. Then the architecture and design of the Web Application will be gone through, as well as a more detailed description of the main components. It also includes an explanation of the user experience for both Desktop and Mobile. Several tools and multiple programming languages were used during the development, and thus a presentation of these are also included in this chapter. Finally, the different iterations of the PoC will be described.

5.1 Functionality

The main functionality of the Web Application built for this PoC, is a private, members-only area that is protected by two-factor login using a U2F device as a security token.

It consists of several, independent Web pages or views:

- Welcome page
- Login page
- User registration page
- U2F Device Registration page
- U2F Device Authentication page
- Members-only page

5.1.1 Welcome Page

The welcome page is a very simple Web page where you are asked if you want to see something secret, and are provided with a button that directs you to the members-only page.
5.1.2 Login Page

The Login page is where you are redirected if you try to access the members-only page, but do not have a valid authentication session. It consists of two forms where you are asked to provide your username and password/PIN. The point of having the End User choose between using a password or a PIN is explained in the Registration page subsection. You are provided with a link to the Registration page, where you can register a new user if you have not registered already.

After you have submitted your credentials, there are four different outcomes:

- Your credentials are correct and you are directed to the U2F Device Authentication page.
- Your credentials are correct, and you are directed to the U2F Device Registration page if this is the first time you login
- You get a message saying that your username or password are incorrect.
- You get a message telling you that your account has been locked, and that you need to contact the owner of the page.
5.1.3 User Registration Page

On the Registration page, you can create and register a new user. It consists of four forms, that all need to be filled to be able to submit:
• **Username**, where you enter the username you wish to have in the system.

• **Email**, where you enter your email address.

• **Password or PIN**, where you either enter a password or a numerical PIN to be used for logging in. Whatever you find the easiest to remember. The point of this, is that the two-factor login makes it less important to have a strong password. The End User can thus choose to use a PIN if they prefer this over a password.

• **Re-type password/PIN**, where you re-type the password or PIN you chose in the form above.

After you have submitted the forms, there are three different outcomes:

• *You are successfully registered and directed to the Login page*

• *You get a message saying that the username you provided already exists in the system*

• *You get a message saying that the email you provided already exists in the system for another user*

• *You get a message that saying that the two passwords/PINs you provided do not match*

### 5.1.4 U2F Device Registration Page

The U2F Device Registration page is where you are directed if it is the first time you login. A description on how to proceed with the device registration is presented. The next steps depends on which type of device you are on; **desktop or mobile**.

**Desktop**

If you are on desktop device, you are presented with a figure and a description on how to continue with the device registration using the USB interface of the security token.

**Mobile**

If you are on a mobile device, you are presented with a figure and a description on how to continue with the device registration using the NFC interface of the security token. If you are using an Android device with Chrome and have the Google Authenticator App[52], said App is automatically launched.
After you have held the the security token against the back of your device or touched the button on the security token, there are two outcomes:

- You have either successfully registered the device to your account and are directed back to the login page
- You are prompted with an error message that describes what went wrong

Figure 5.4: U2F Device Registration Page

5.1.5 U2F Device Authentication Page

The U2F Device Authentication Page is where you are directed if you successfully logged in using your credentials and have pre-registered a U2F Device. A description on how to proceed with the device authentication is presented. Again, the next steps depends on which type of device you are on; desktop or mobile.

The description and figures are almost identical to what has been already described in the subchapter U2F Device Registration Page.

Again, after you have held the the security token against the back of your device, there are two outcomes:

- You have either successfully authenticated the device and are directed back to the members-only page
- You are prompted with an error message that describes what went wrong
5.1.6 Members-only Page

The Members-only page is a Web page that can only be reached if the user has a valid session cookie. At the top of the page there is a drop-down menu, where additional, personal pages could be linked to. A good example of a page that could be added, is a *My Devices* page that could show information about which U2F Devices/Security tokens that the user has registered.

A logout button is present in the upper-right corner of the Web page, and directs the End User back to the login page after it has killed the current user session.
5.2 Architectural Overview

In its simplicity, the Web Application can be divided into two components; a backend component and a client component. The Client component is the code that is executed in the Web browser and is what the End User interacts with directly. The Backend component retrieves information from the Client and processes it, returns information back to the Client if requested.

![Simplified Architectural Overview](image)

Figure 5.7: Simplified Architectural Overview

The backend component consists of multiple subcomponents that are interconnected. There is one central component that receives all the incoming requests from the Client, and connects with the other components in the backend if necessary.

On the other side we have the user interface that lives in the Web browser. There is also the Security Token that connects with the Client through the browser. Finally, we have the End User who interacts with both the user interface and the Security token.

5.3 System Components

This section will go into detail about what roles the different components, and their subcomponents have in this PoC system.

5.3.1 Backend

The backend part of the system has multiple roles.

- It keeps track of all the users, registers new ones and store their personal info in a database
- It handles the validation of credentials and manages the authentication sessions
Figure 5.8: Architectural Overview (including subcomponents)

- It validates authentication and registration challenges coming from the security tokens of the users

Below, is a more detailed description of the backend components seen in Figure 5.8.

**User Database**

The user database stores all the info about the users. The exception is the information regarding their security tokens, which is handled by a different service. All the data is stored in a single table that consists of 8 columns.

<table>
<thead>
<tr>
<th>id</th>
<th>username</th>
<th>password</th>
<th>salt</th>
<th>email</th>
<th>hasU2Ftoken</th>
<th>passRetryCounter</th>
<th>accountLocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>user</td>
<td>15877f2a3...</td>
<td>580c7b...</td>
<td>user@localhost</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.1: The User Database table (with user example)

The *id* column, stores a unique identifier for each user in the table, in the form of an integer. It also stores the *index* in the database table. The index in a database helps improve the speed of operations performed on the database table, by not having to search every row in the table each time it is accessed. The value of this column is auto-incremented each time a new user is added to the table.

The *username* column, stores the usernames of the different users in the system. There are currently no strict limits on the values of this column.
other than the fact that it needs to be unique, consist of UTF-8 characters and not be longer than 255 characters total.

The password column, stores a 64 byte long Hex string that is the hashed value of the plain text password and the salt. This makes it much harder for an attacker to get the users password if they somehow managed to get access to the database. A more detailed description on the security aspects of this follows in section 5.3.

The salt column, stores the 8 byte long Hex string value of the salt. The salt is randomly generated for each user, and is hashed together with the password. The main purpose of this process is to slow down brute-force attacks. More on this in section 5.3.

The email column stores the email address of the users. This value is currently not used in the PoC, but is very common and practical to store in a system. This could for instance be used to send out URLs where users could reset their password, if this functionality were to be added in a future extension of the PoC.

The hasU2Ftoken column, stores a boolean value that tells if the user has a registered token or not. This is automatically set to “0”, or false, when a new user is added. The column is queried each time a user successfully logs in to see if they have registered a token. If not, they are directed to the U2F Device Registration Page.

The passRetryCounter column, stores an integer representation of the number of times a user has unsuccessfully authenticated. This value is by default set to zero when a new user is created.

The accountLocked column, stores a boolean value that tells if the user has been locked out of the system. This value is automatically set to true, or “1”, if the value of passRetryCounter exceeds 5. The default value of this column is false.

U2F Validation Service

The U2F Validation Service manages the registration and authentication of U2F security tokens. It is a server component that is interacted with through its REST API.

It generates authentication and registration challenges according to the FIDO U2F technical specifications [4]. The information regarding the security tokens that are registered, is stored in a separate database.

A client needs to be created for each of the applications that are to use the Validation Service. Such a client consists of:
- a client name
- the appID (according to the U2F standard) of the client
- the facetID (according to the U2F standard) of the client

**Central Component**

The *central component* is the component that has a single public facing interface, or API, that all traffic requests from the Client goes through. This component also has connections with all other subcomponents. It is the heart and brain of the backend system.

The API has several URL endpoints that exposes different services for the Client to use.

- **/api/rest/register** is an endpoint that handles data sent using HTTP POST requests. It takes the username, email address and password of a user and creates a new user in the database. It thoroughly checks the received parameters against already registered users in the database to avoid collisions. If a username or email is already in use, it returns an error message to the Client. Else, it returns a message telling that the registration was successful.

- **/api/rest/login** is also an endpoint that handles data sent using HTTP POST requests. It receives the username and password of a user and validates it against the values stored in the User Database. Before validating the provided credentials, a check on the value of accountLocked in the database for the username is performed. If the user has been locked out, an error message regarding this is returned. If the login is ok, it returns a message telling that the login was successful and creates a valid session the current user, that enables them to access the U2F Device Authentication or Registration page.

  If the login was unsuccessful, it increases the passRetryCounter value in the User Database and return an error message in the HTTP POST response.

- **/api/rest/authenticateu2f** utilizes both the HTTP GET and POST method.

  The GET method is used to retrieve an U2F authentication challenge tied to the user that is trying to log in. The Central component uses the U2F Validation Service’s API to fetch an authentication challenge and returns it as a JSON object to the Client as the HTTP response.

  The POST method is used to retrieve and process signed U2F authentication challenges coming from the Client. The Central
component transfers the JSON object with the signed challenge to the U2F Validation service, again using its RESTful API. If the user has posted a valid signed response to the challenge, a valid session for the corresponding user is set up enabling them to reach the Members-only Page.

- **/api/rest/registeru2f** also handles both the HTTP GET and POST method.

   The GET method is used in a very similar way to the endpoint above, but instead of requesting an authentication challenge, the Client is interested in a registration challenge. The Central component connects with the U2F Validation Service, and returns a registration challenge to the Client for the user that wants to register an U2F device.

   The POST method receives the signed registration challenge and checks its validity with the U2F Validation Service. If the registration challenge is successfully validated, the value of the hasU2Ftoken in the user database is set to true for the corresponding user. The End User can now use the registered U2F device in the two-factor login mechanism.

- **/api/rest/members (currently not in use)** is used to return a list of all the members of the Web Application. It accepts GET request from users that have a valid authentication session, and return a JSON encoded list of all the members. This endpoint is currently not adopted in to the PoC Web Application.

- **/api/rest/listdevices (currently not in use)** returns a list of the current logged in End User’s U2F devices.

   As with the endpoint above, this endpoint was not utilized in the current version of the PoC. They were planned to be used to add useful information to the Members-only page, and therefore deserves a mention in this section.

### 5.3.2 Client

The Client holds the User Interface that the End User interacts with through the Web Browser. It is very minimalistic in design, with the main focus of enabling the End User to easily understand how to interact and complete the necessary authentication steps. The layout of the user interface is responsive and changes dynamically depending on the type of device; mobile or desktop.

The User Interface captures and manages the input from the End User, as well as the security token. The Client then transmits the captured
information to the Backend part of the system, and then awaits a response. After receiving a response from the Backend, usually one of the following events are triggered:

- redirection to another page of the Web Application
- an error message is displayed to the End User
- additional capture and transmittal of End User/Security Token input

An important responsibility of the Client is to handle the communication with the Security Token. For this purpose, it uses the U2F JavaScript API. Data is retrieved from the backend and transmitted to the token with the use of the JS API, and vice versa. This functionality and the exploration of it, is the key purpose of the PoC.

5.4 Process View

Figure 5.9 is a diagram that includes the sequence of events during the two-factor login mechanism in this PoC. It presents the flow of events between the components presented earlier.

5.5 Security and Limitations

This section will briefly discuss the security aspects of the PoC and some of the security mechanisms that are implemented. At the end, there is a short subsection dedicated to some of the limitations of the Web Application created in the PoC.

5.5.1 Security Mechanisms

Because of this Web Application being a Proof of Concept, there has not been as much focus on secure development as it would be if the application where to be used in production. Nevertheless, some security mechanisms have been implemented and are described below.

Secure Storage of Passwords

The End User’s passwords that are stored in the User Database, are hashed together with a salt using a strong hashing algorithm (SHA-256). The salt is a unique, random generated value for each user. The presence of the salt in the hashing function together with the password, creates different hash
values for each individual user, even if two users share the same password. This delays the process of dictionary attacks, as well as rainbow table attacks when the attacker tries to crack multiple passwords.
Prepared Statements

Prepared statements can be used to mitigate *SQL injection attacks*. This feature is normally used by database management systems to execute a defined query repeatedly with increasing performance. It can be compared to a type of template. Because the values of the statements are sent at a later time, using a different protocol, they need to be escaped correctly.

Session Cookie

In contrast to a normal cookie, the session cookie only stores a reference to the actually cookie object which is stored on the Web server. This means that the content of the cookie can not be tampered with on the Client side. Although this mitigates tampering of the authentication cookies, the session cookie can still be stolen and used by an attacker.

HTTPS

HTTPS is used to secure the channel between the Web Browser and the Backend service. This ensures that the username and password are transmitted encrypted over the internet to the server hosting the Web Application.

5.5.2 Limitations of the PoC

There are some limitations that restrict the goal of wide user adoption to some degree.

- The U2F registration and authentication functionality is limited to the browser that supports the U2F JS API. Currently, only Google Chrome has built in support for this on their Desktop and Android browsers. Firefox has a browser plugin that enables U2F support.
- Android is currently the only mobile operating system that is supported, and you need to manually install the Google Authenticator App.
- No mechanism for securely registering a new U2F device (if an End User loses their initial, registered device) has been implemented in this PoC.
- There is also no automatic mechanism for updating a password, or recovering the password if the End User forgets it. After an account
has been locked out due to too many failed login attempts, the user is presented a message describing them to contact the site administrator.

5.6 Development Tools and Environment

This part of the chapter will present the most important development tools and the programming languages chosen for the implementation of this Proof of Concept.

5.6.1 Hosting Server

For development of this Proof of Concept, a private server was chosen for hosting the different iterations of the final Web Application. This choice was primarily out of convenience reasons. The server has a static IP address and a connection to the Internet, as well as the most necessary ports for Web Applications to be hosted open (port 80 and 443). This meant that the Web Application could be reached from anywhere and on any device, without the need for VPN clients that are common if I had chosen to host the Web Application on for instance a server owned by the university. There were also no restrictions on what software components that could be installed, which is very convenient for testing.

Since the source code was continuously modified on a publicly reachable Web server during the development (see the Git sub chapter), there was a risk that an attacker could exploit one or several possible vulnerabilities in the application. The server contains no critical personal data, and are mainly used exploring and home projects, so the risk was considered to be low.

Another big advantage of choosing to have the project live on the Web, was that I could have a valid, trusted certificate for HTTPS, contrary to using a self-signed certificate that would have to been imported manually for each Web browser. Since the possibility to test the Web Application using different browsers was important for the research, this was a major convenience.

The server runs a Linux distribution called Debian 8 “Jessie” [28], which is Open Source and thus free to install.

5.6.2 Certificates for HTTPS

Since the U2F protocol out of security precautions only works when HTTPS is used, a valid TLS certificate for the domains chosen for the Web
Application was needed. For this, *Let’s Encrypt* [35] was chosen as a tool.

The objective of Let’s Encrypt is to make it easier to set up an HTTPS by automatically obtaining a trusted certificate without human intervention. Getting a valid certificate for the Web sites, was as easy as typing a single command in the server terminal. A part for the usability, Let’s Encrypt was chosen because the certificates are free of charge. The certificates are also trusted by all the major browsers.

5.6.3 Security Token

The security token that was used during the development of this U2F PoC, was a Yubikey Neo by Yubico [129]. This token has an USB interface and also supports NFC. In addition to U2F, it also supports OTP and has some smart card functionality. The computer recognizes it as a USB keyboard, which makes it practically driver-less, and thus enhances the usability greatly.

5.6.4 Kanbanize, Git and Visio 2013

Kanbanize[66] was used to exercise the Kanban methodology during the development. Kanbanize is visual management software that helped with visualizing the workflow in this project. It is designed for teams working together using *lean principles*[57], but is also very suitable for only one developer. In addition to being a Web Application, Kanbanize is also available as an mobile App, which made it very easy to add new tasks at any time.

*Git* is a very popular version control system. It was primarily used to continuously push changes to the source code of the project to the server that hosted the Web Application. Contrary to using a service to host the repository such as GitHub, a central repository was set up on the hosting server. Having a central repository, opposed to having perhaps multiple local instances, made it easier to work from different computers.

A functionality of Git named *hooks*, was used to automatically copy any changes of the project directly to the hosting folder of the Web server. This meant that a live, up to date version of the source code was instantly available on the Web after pushing changes to the repository.

Microsofts *Visio 2016* was used for architectural sketches, often by digitalizing sketches that were already hand drawn.
5.6.5 Integrated Development Environment (IDE)

The Integrated Development Environment (IDE) is an application that is used for software development. It consists of the necessary development tools, such as a source code editor, debugger and often a compiler.

Eclipse was used as the IDE for the development. This was chosen primarily because of earlier experiences with the software, which meant that I was familiar with most of its functionality.

It supports syntaxes for all the different programming languages that were used, which makes it easier to spot typing mishaps, while also offering autocomplete functionality. Eclipse also integrates well with Git, which made it easy to push the changes to the remote repositories with keyboard shortcuts during the development.

5.6.6 MySQL and u2fval

MySQL was chosen as the relational database management systems (RDBMS) for this project. It is both free to use and easy to setup on a server. The tool phpMyAdmin [98] was used to administrate the MySQL database through an easy-to-use Web interface.

u2fval (U2F Validation Server) is standalone server that offers U2F registration and authentication services through a REST API [130]. It is built using Python, and was installed on the hosting server using pip[40], a tool for installing Python packages.

The U2F Validation Server needs a database to store information about the registered tokens and corresponding users. The default database option, SQLite, was changed to a PostgreSQL instance running on the server, mainly because of the added security features.

5.6.7 Programming Languages

A number of programming languages were used for the development of the Proof of Concept.

PHP

PHP is a scripting language that is much used for Web development, but also serves as a general-purpose programming language. The PHP code is processed by a PHP interpreter in the Web server. The final result that is displayed to the End User may be a combination of the interpreted code and
generated HTML and/or dynamic Javascript content. This feature was used to have a short PHP code snippet on the top of all the protected Web pages, to check if the user had a valid browser session.

PHP can also be used to build pure server-side components. In this project it was used to build a REST API, that among others received usernames and passwords for validation and returned authentication challenges.

**JavaScript, jQuery and AngularJS**

JavaScript, jQuery and AngularJS have all been used during the development for different purposes.

*JavaScript* is a high-level programming language most often used the development of Web Application. It is considered, together with HTML and CSS, as the three essential technologies for the production of World Wide Web content. JavaScript was mainly used to used together with the U2F API, to connect and exchange data with the security token.

*jQuery* is a very popular JavaScript library. It was used when there was a need to receive and send authentication challenges using HTTP GET and POST to the backend API. For this purpose, jQuery’s excellent support for *Ajax*, which can send and retrieve data asynchronously without refreshing the Web page, was used. jQuery was used when there was a need for simple interaction with the backend, such as requesting and returning a single challenge.

*AngularJS* was used to build the login page and the page where you can register a new user. AngularJS is a JavaScript framework that is very useful to build Web Application that are responsive to user actions, by extending the HTML syntax [49]. It is often used to build single-page applications (SPAs), that focuses on fluid user experiences. AngularJS is great for pages with forms that are filled by the End User and sent to the backed server. You can easily prompt the user with server responses without a page reload. Examples on how this was used in this PoC, are checks to see if a user already exists in the system or if the password provided is incorrect.

**SQL**

SQL is a programming language especially designed to be used for managing data held in relational database management systems (RDBMS), such as the MySQL database used in this project. It was used in combination with the components written in PHP to query, create and update data in the user database.
5.7 Iterations

Throughout the development of this PoC, there has been several changes to the code as well as the architecture. New ideas was formed during the progress, which led to new functionality and security enhancements. This subsection of the chapter will describe how the application evolved during the development through three iterations.

5.7.1 Iteration 1

The development of the PoC began with building a simple Web Application with a authentication protected members-only area. The first iteration was built entirely using PHP together with a MySQL database.

It had the following Web pages:
• **login.php**, with two simple Web forms for username and password login

• **private.php**, the members-only Web page

• **logout.php**, a simple PHP script for login out correctly

• **memberlist.php**, included a list of all the registered members and their public info

• **register.php**, with multiple forms intended to register new users

• **edit_account.php**, used to edit and existing account

• **common.php**, used for the session management and connection to the database storing the users

The primary focus of the first iteration was to gain knowledge on how to build a basic username/password protected Web Application. The goal was to build a basic, but secure login system using best practices to mitigate common and possible attacks.

The starting point of the Web Application was a very informative code example with a lot of descriptive comments throughout the source code[30]. The tutorial was especially centered around four security topics:

- Secure storage of passwords in databases
- Preventing XSS attacks
- Securely redirecting End Users to other pages
- Preventing SQL injection

The first iteration, in addition to begin a good introduction to secure Web Application development in PHP, formed the basis for later iterations.

### 5.7.2 Iteration 2

The second iteration changed the architecture of the Web Application quite drastically. The aim was to split the responsibilities into different components and thus making the application more modular. In addition, the functionality of registering U2F security tokens and enabling to use them in a two-factor authentication mechanism was added. The design and layout of the Web pages were also refreshed by introducing other Web technologies for this purpose, namely CSS and HTML5. Some effort were put in to make the different Web pages have a similar “look”.

Being familiar with Angular from earlier experiences, I chose to rebuild the login and registration page using it. Since Angular works great with JSON, the Backend was chosen to be built with a public facing RESTful
API that consumes and returns said format. Deciding to build the Central Component and the REST API using PHP, meant that much of the code from the first iteration could be reused.

To be able to use U2F, the U2F Validation Service component was added to the architecture. Two new pages for U2F device registration and authentication was created. To allow the use of U2F JS API `sign` and `register` functions, a JavaScript file name “u2f-api.js” was added to the project. jQuery was chosen to fetch and post both the U2F registration and authentication challenges that were transmitted to and from the security token.

This iteration was fully functional as a Proof of Concept, and worked well with the USB Security Token that were purchased for this thesis. Yet, a new software update to the Google Authenticator Android App, opened up the possibility of a new and final iteration.

### 5.7.3 Iteration 3

The third iteration became centered around the fact that Google added U2F support to their Authenticator App for Android. With this new update, it was now possible to connect with security tokens that have implemented U2F NFC support. This, combined with the fact the App would be automatically launched when visiting a page using the U2F JS API, made it clear that the PoC should be adapted for Mobile use.

To make the authentication and registration processes more user-friendly, two separate figures (mobile and desktop) were made to show the user how to proceed. The two figures where initially camera photos of someone interacting correctly with the security tokens, and were later digitally transformed to look like they were drawn by hand. Both figures had descriptive captions that match the images. The concept of feature detection was used to display the correct image to the End User depending on which platform they were on. The feature detection were based on a few lines of JavaScript code that investigated the User-Agent string of the HTTP requests, and decided which figure to display based on a list of known type of mobile devices. Some work were also put in to make sure that the design was responsive and adapted correctly to the typically smaller screens on mobile phones. For this purpose, the Bootstrap framework was used.

There were surprisingly few changes needed to make Web Application function correctly on Android devices. Some problems with the `facetID` of the U2F protocol were discovered when inspecting the JSON data returned to the Mobile from the Security Token. There were different values of this parameter depending on if the USB or NFC interface was used. The problem
was solved by adding multiple facetIDs to the list of accepted values on the U2F Validation Service.

As a result of the changes made from the second iteration, the third iteration worked successfully on both desktop and Android mobile devices.

5.8 Demonstration Video

A demonstration video showing how the PoC is used in action can be seen at [110]. The video shows how usable the authentication and registration processes in the PoC are. Both processes are performed on the desktop device, while the video only shows the authentication process on the mobile device.
6 Evaluation of PoC

This chapter will start out with an evaluation of the Proof of Concept developed for this master project by comparing it to similar multi-factor solutions. The second part of the evaluation will take at look at how the PoC and its authentication mechanism complies with requirements introduced in two, recognized assurance level frameworks for entity authentication.

6.1 Comparison with other Multi-factor Solutions

As part of the evaluation of this PoC, a comparison with similar existing solutions was performed based on a set of categories. These categories are mainly related to compatibility and support for End Devices, Operating Systems and the major Web browsers. Other categories that will be included in the comparison are the types of dependencies needed, such as the need for drivers or third-party middleware\(^1\) to be installed on the End Devices for the solution to function properly.

The context of this comparison is set to multi-factor authentication even though some of the solutions can be used for digital signatures and encryption as well.

Solutions from Norwegian eID vendors Commfides and Buypass was chosen to be a part of this comparison. Both use security tokens that are hardware based and perform cryptographic operations inside a Secure Element embedded in the device. In addition, they use USB as the main interface against the computer and are thusly comparable to the security token used in the PoC.

The Buypass solution uses a smart card as form factor, while the Commfides solution is a USB device/dongle. Despite the different form factors, all three solutions can be used for two-factor login on the Web using something you know (PIN or password) and something you have (the token itself). The subsections below gives a brief introduction to each of the solutions that are part of the comparison, including the PoC.

\(^1\)The need to install additional software on the End Devices, such as Browser Extensions, Java Applets or Desktop applications
6.1.1 PoC

The PoC is the solution that was described in Chapter 5. It uses an USB security token that in addition supports NFC. The solution uses the FIDO U2F protocol which uses different private keys per user and per Online service in contrast to normal PKI based solutions where often the same certificates and keys are used for multiple services.

6.1.2 Commfides

Commfides is an eID vendor that uses a USB stick with a Secure Element inside it. It is a PKI solution where the End User’s private keys and personal certificates are stored on the SE. In addition to being used for strong, two-factor authentication, it can be used used for digital signatures and encryption of emails. It is approved for Security Level 4 for usage with public services by Difi (Norwegian Agency for Public Management and eGovernment). Commfides uses a Java Applet to communicate with the USB stick from the browser.

Data used to evaluate the solution in this comparison are collected from Commfides’ support pages on the Web[22, 23].

6.1.3 Buypass

Buypass eID is a smart card based solution. Similar to Commfides, the solution is based on PKI and supports digital signatures and encryption in addition to authentication. It is also approved for Security Level 4 by Difi and can thusly be used by all levels of public services. For this comparison, two versions of their solution has been selected, as both are still in use:

- **Buypass**, is the solution that uses Java Applets to communicate with the card.

- **Buypass (Java-free)**, is a newer solutions that uses a component called SCProxy (see Section 3.4.1) instead of Java.

Data used in this comparison is collected from instructions manuals and support pages on the Web[17, 16, 18, 15]. This solution has also been tested “hands-on” during the work in this project.

The next five sections will be devoted to comparisons in the different categories before Section 6.7 provides a summary. Each category will consist of a matrix that show the different solutions level of support. Below each matrix, comments about each of the solutions are presented.
6.2 Comparison - Operating System Support

Table 6.1 shows how the different solutions support the three major desktop Operating Systems.

<table>
<thead>
<tr>
<th></th>
<th>PoC</th>
<th>Commfides</th>
<th>Buypass</th>
<th>Buypass (Java-free)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mac OSX</td>
<td>Yes</td>
<td>Partly</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Linux</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

PoC

The PoC supports all Operating Systems.

Commfides

The Commfides solutions mainly supports Windows Operating Systems. There are however mention of a third-party Mac OSX driver on their support pages[23], and thus Mac OSX support was set to Partly.

Buypass

Both Buypass solutions support all the major Operating Systems.

6.3 Comparison - Support for the major Web browsers

Table 6.2 shows how the different solutions support the four major Web browser\(^2\).

\(^2\)Based on statistics from October-December 2015 provided by StatCounter[108]
PoC

The PoC has as mentioned earlier in this thesis, currently only built-in support for Google Chrome browsers. A browser extension can be installed in Firefox to enable support[20]. It is worth noticing that statistics show that Chrome almost has a 50% share of the global Web browser market[108], and that the PoC thus can be considered to reach out to a considerable amount of End Users despite it only having built-in support for a single browser to this date.

Commfides

Commfides supports the Microsoft browsers IE and Edge, in addition to Firefox. Because Commfides uses Java, Chrome is no longer supported[86].

Buypass

The Java based version of Buypass supports all the browsers, except Chrome which no longer support Java applets.

Buypass (Java-free) supports all the major browsers. This is because of the architecture of their middleware component SCProxy, described in Section 3.4.1. In addition to the four browsers listed, it also support Web browsers such as Opera.

6.4 Comparison - Driverless experience

Table 6.3 show how the different solutions supports a driverless experience, meaning that there is no need to manually install device drivers on the major Operating Systems in Section 6.2.
Table 6.3: Support for driverless experience

<table>
<thead>
<tr>
<th></th>
<th>PoC</th>
<th>Commfides</th>
<th>Buypass</th>
<th>Buypass (Java-free)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driverless experience</td>
<td>Yes</td>
<td>No</td>
<td>Partly</td>
<td>Partly</td>
</tr>
</tbody>
</table>

PoC

The security token used in the PoC is a USB device of the *HID class* and has built in driver support on all major Operating Systems.

Commfides

The Commfides solution requires the End User to install the necessary driver manually.

Buypass

Both Buypass solutions is marked as *partly* because the smart card readers used can have different driver support. Some smart card readers use drivers that are built in to most Operating Systems, while some manufacturers use their own proprietary drivers. In addition, Linux requires multiple packages to be installed before first time use[15].

6.5 Comparison - Installation of third-party Middleware

Table 6.4 shows how the selected solutions has the ability to be used without any kind of third-party middleware to be installed on the computer.

Table 6.4: Need for third-party middleware

<table>
<thead>
<tr>
<th></th>
<th>PoC</th>
<th>Commfides</th>
<th>Buypass</th>
<th>Buypass (Java-free)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need for middleware</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
PoC

The PoC does not need any additional installation of third-party middleware, as the U2F Web API is built into the Chrome Web browser.

Commfides

The Commfides solution depends on a Java applet to be used for Online authentication. Java often needs to be manually installed on the computer and regularly updated.

Buypass

The regular Buypass solution also uses Java in the browser.

The Java-free version of the Buypass solution, requires the End User to install the SCProxy component on their computer.

6.6 Comparison - Mobile Device Support

Table 6.5 shows if the selected solutions can be used with a Mobile device.

<table>
<thead>
<tr>
<th></th>
<th>PoC</th>
<th>Commfides</th>
<th>Buypass</th>
<th>Buypass (Java-free)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for mobile devices</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

PoC

The PoC can be used with Android mobile devices that have support for NFC. Note that in this case, users currently have to manually install the Google Authenticator App.

Commfides

Commfides has currently no support for Mobile device usage.
None of the Buypass alternatives selected for this comparison has currently support for Mobile device usage.

### 6.7 Summary and Result of Comparison

The results from the comparisons show that the PoC matches or surpasses the other solutions in several categories.

Figure 6.1 shows a bar chart where the x-axis shows the summarized score of the comparisons performed in the Tables in earlier subsections. The score is calculated in following way:

- Each column with value of **No**, result in a score of 0.
- Each column with value of **Partly**, result in a score of 1.
- Each column with value of **Yes**, result in a score of 2.

The exception is the column in Table 6.4, where the score is calculated in reverse order (**No** is a positive outcome in this setting).

From the graph, we can see that the PoC scores the same as the Buypass solution that is Java-free. The categories where the PoC has the best scores compared to the others, are the categories that are typically related to *first-time usage*. The PoC outscores the other solutions in the categories *Support for driverless experience* and *Installation of third-party Middleware*. It is also the only solution that currently supports authentication using mobile devices. Some of the Buypass smart cards has a contactless interface, but this is not yet utilized in a mobile solution.
The primary reason why the PoC scores the same as Buypass (Java-free), is because of the category devoted to browser support. The Buypass Java-free component (SCProxy) in theory supports all browsers because it communicates with the Web browsers via standard HTTP requests.

The U2F Web API that the PoC uses is currently only built into the Chrome browser, but is planned to be implemented in several of the other major browsers over time. It is thus a possibility that the PoC could have the highest score overall, if the same comparison is executed at a later stage.

6.8 Evaluating against Assurance Level Frameworks

This section will discuss how the PoC would rank on security assurance level frameworks from the Norwegian governmental agency Difi and from ISO/IEC.

6.8.1 Difi Security Assurance Levels

The Norwegian Agency for Public Management and eGovernment (Difi) has created an assurance level framework for electronic Identity solutions which consist of four different security assurance levels [33]. It is up to owners of the Online service to decide which level of security is required. This decision process is determined by the type of service being offered. It is primarily based on the consequences that an unfortunate incident could have for both the End User and the agency itself. The PoC will only be evaluated against the two highest levels (3 and 4), as Level 1 and 2 only require single-factor authentication solutions and is thus “automatically” fulfilled.

- **Security Level 3** offers medium-high level of security. Security Level 3 must use two-factor authentication and should consist of the username in form of the Norwegian nation identity number and a personal password, in combination with a second, possession based factor.

- **Security Level 4** offers the highest level of security. As with Security Level 3, this level requires a two-factor authentication mechanism that consist of a knowledge based and a possession based factor. The difference from Level 3, is that the e-ID is handed over in a meeting with the person after their identity has been established [33].
Evaluating the PoC

The authentication mechanism used in the PoC meets the requirements for Security Level 3, as it uses a two-factor authentication mechanism consisting of knowledge-based credentials in addition to the possession-based factor in the form of the security token. Even though it is not a requirement in the PoC Web Application, the username could be the national identity number.

Whether the authentication mechanism in the PoC meets the criteria of Security Level 4, is a subject of discussion. As the PoC is based on the Fido U2F protocol, there is no ownership tied to the tokens in use. This means that if the token was delivered to the End User in a meeting with the person as required by Level 4, this token could still be used by other users. The token could in addition be registered for use on any other Online service than the service it was registered to, governmental or not. Still, if the security token was pre-registered to the specific service when handed to the User, Fido U2F ensures that nobody else can use this token on behalf of the user to authenticate, without knowing the registered knowledge-based credentials.

A registration process for a Fido U2F enabled security token as used in the PoC that fits within Difi’s framework, could occur in several ways.

- End Users could identify themselves “over the counter” when ordering the token, which would enable a registration officer to register this token to the user. The security token could then be ordered, and would be delivered to the End User already pre-registered.

- Another option is to use an existing, approved Level 4 eID to register the new token. An Online service, which required Security Level 4 authentication, could be created to successfully identify an End User and order a token. The End User could then be required to meet in person when retrieving the token, which in this case also would be pre-registered for the public service.

Alternatively, the End User could register an approved security token that they already possess in the Online service, after successfully authenticating with an existing Level 4 eID.

6.8.2 ISO/IEC 29115

The Entity Authentication Assurance Framework standardized in ISO/IEC 29115 [58], specifies four levels of entity authentication assurance. It states that the selection of the appropriate assurance level should be based on a risk assessment of the services that entities will authenticate to. The standard also includes a table to help system owners to select the appropriate
level of assurance. As with the Difi security levels, the PoC automatically fulfills the requirements of Level 1 and 2, as they only require single-factor solutions. The following discussion regarding the PoC’s fulfillment will thus only concern the two highest levels.

- **Level of assurance 3 (LoA3)**, delivers a *high confidence* in the asserted identity. It must use a multi-factor authentication mechanism. Secret information exchanged in the authentication protocols should be cryptographically protected. There are no special requirements regarding generation or storage of credentials.

- **Level of assurance 4 (LoA4)**, delivers a *very high confidence* in the asserted identity and must also use multi-factor authentication. The difference from LoA3, is that the identity proofing requires physical presence. In addition, it must use tamper-resistant hardware devices for storing the cryptographic private keys.

### Evaluating the PoC

The authentication mechanism used in the PoC complies with the first requirement in LoA3, as it is a multi-factor solution. In addition, it uses the FIDO U2F protocol which provides cryptographic protection of all sensitive data, as described throughout Section 4.2.

As for LoA4, the PoC uses tamper-resistant hardware for storing the private keys, and thus fulfills this requirement.

The previous section discussed how a U2F based solution such as the PoC, could potentially fulfill requirements regarding physical appearance in the identity proofing process. Even though the PoC Web Application does not require physical present during registration of the security token, it could be implemented.

It can thus be concluded that the current iteration of the PoC does not fully comply with LoA4, but that is possible to fulfill all requirements with some changes to the enrollment process.

### 6.9 Summary

This evaluation has shown that compared to existing solutions in the Norwegian eID market, the PoC has a equal or greater level of usability based on the categories set in the comparison performed. The PoC falls short in the category regarding Web browser support, but can increase the score significantly in this category in the future, as several of the major browsers already has planned to implement support for the U2F Web API.
The evaluation has also shown that the PoC, and its authentication mechanism, complies with at least Level 3 of the Difi and ISO/IEC 29115 security assurance levels for entity authentication. With some modifications to the enrollment process in the future, the solution could also comply with the highest assurance levels for both frameworks.
7 Conclusions

The study in this Master’s project was set out to explore how new Web APIs can increase the usability of multi-factor authentication mechanisms utilizing Secure Elements. This thesis has analyzed the existing solutions enabling SE utilization with Web Applications, and presented known usability and security limitations with these. Due to the limitations of the state-of-the-art solutions, this study searched for new solutions to the problem and analyzed several of the proposed Web APIs and protocols that could potentially increase the usability of multi-factor authentication involving Secure Elements. The research sought out to answer these three research questions:

- How can the standardization of future Web APIs enhance the usability of multi-factor authentication mechanisms that use SE-enabled security tokens?
- What are the main limitations of the current solutions for SE utilization on the Web?
- Can solutions based on future Web APIs contain the same level of security assurance as existing solutions and simultaneously provide a higher degree of usability?

The most significant findings are chapter specific, and were summarized at the end of the empirical chapters; Analysis of Current Solutions, Analysis of Future Solutions and Evaluation of PoC. The next part of this section will take a look at how the findings presented throughout this thesis answer the project’s research questions.

Regarding the first research question, the analysis of the future solutions showed that their proposed Web APIs plan to make Web browser able to directly communicate with the lower interface layers on End Devices, thus eliminating the need to install additional software components. The usability enhancements provided by these APIs are however depended on proper standardization by the W3C.

The PoC built for this project, shows how usable multi-factor authentication mechanisms can be when these Web APIs are built into the major browsers. The comparison performed in the evaluation resulted in a score equal to one of the current solutions, but this was due to the other solution vastly outscoring the PoC in the browser support category and that the ongoing W3C standardization process may have an impact on this score (favoring the PoC) at a later stage. It can be concluded that proper standardization
of the future Web APIs presented in this thesis, can in fact enhance the usability of multi-factor authentication with SE-enabled tokens on the Web, as proven by the success of the Fido U2F initiative.

The analysis of the current solutions performed during this study have produced findings that can answer the second research question. The analysis shows that some of the current solutions are depending on increasingly discouraged technologies such as Java applets. The solutions are also suffering from the lack of interoperability between the Web browser’s different development APIs for extensions, resulting in developers having to create different versions of their extensions per browser type. Research presented from other studies expose several security issues and vulnerabilities with current solutions. A conclusions however, is that the common limitation of almost all of the current solutions is the need to install additional software, ultimately decreasing the goal of high usability of security measures to increase user adoption.

To answer the third research question, findings from the evaluation of the PoC built for this Master’s project can be used, as it utilizes one of these new Web APIs. The results from the usability comparison in this study show that the PoC has the same score as one of the current solutions, with the main drawback being the lack of browser support. In the evaluation rating the PoC against the security assurance level frameworks, it was considered to reach Level 3 on both frameworks with the possibility to reach Level 4 with some changes to the enrollment process. Solutions from the usability comparison were all considered to Security Level 4 in Difi’s assurance level framework. It can thus be concluded that it is plausible for the future solutions to have the same security level assurance, but that the lack of browsers supporting these new solutions at this point of time constrain the goal of higher usability.

An interesting discovery in this research is how some of the future APIs could benefit from each other, such as the example in Section 4.1.3 shows. Both the SE and NFC Web API could potentially provide a direct channel to Secure Elements that the U2F Web API could utilize.

Findings regarding the usability and security of Fido U2F-enabled tokens in this study fits well with the results from the more extensive evaluation performed in the recent Security Key Report by researchers at Google [72].

The Proof of Concept developed for this Master’s project has shown how new Web APIs can be used to create Web Applications that utilize SEs for strong multi-factor authentication with a high degree of usability, given sufficient Web browser support. As a final conclusion, the success of these future Web APIs very much depend on the involvement of W3C ahead. As of today, both the SE and NFC APIs have had a smaller progression towards becoming a standard recommendation approved by W3C. By comparison
the recent Fido 2.0 API specifications that has been picked up by W3C [127] show how far these new solutions can get with enough support from major corporations and governmental agencies.

7.1 Future Work and Improvements

The research presented in this thesis points out some areas that can be enhanced with future work.

The PoC could be extended with more functionality and some of the limitations mentioned in Chapter 5 could be fixed. An interesting improvement would be to add support for adding multiple U2F devices to already registered users in a secure manner. In addition, the back-end could be developed with another, more up-to-date server-side languages than PHP, such as Node.js.

In this thesis, only the U2F APIs were implemented in a PoC. Future research could develop a PoC based on available implementations of either the SE API or NFC API. Both Firefox OS and Tizen OS has supported versions of these Web APIs. The PoC could for instance use a SE enabled UICC or Smart Card and develop a two-factor login in a Web Application developed for one of these Operating Systems.

In hindsight, there are some choices that could have been made differently and phases of the research that could be improved. For instance, the evaluation of the PoC could have been performed on a larger scale and with actual observation of End Users. Due to limited time available after the completion of the PoC, a different, less time consuming approach to the evaluation was performed.

An ideal approach, could be to observe how users performed two-factor authentication with some of the different, already existing solutions in addition to the PoC developed in this thesis. It would be interesting to set up a computer that was completely factory default for each user, and observe how they evaluated the usability of first-time use when equipped with nothing more than the publicly available user guides for help.
Bibliography


[26] csn.arstechnica.net. URL: http://cdn.arstechnica.net/tlscert_firefox_accept_cert.jpg.


[95] OWASP. *DOM Based XSS*. URL: https://www.owasp.org/index.php/DOM_Based_XSS.


