BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

CLOUD SERVICE ACCESS CONTROL USING SMART CARDS

ŘÍZENÍ PŘÍSTUPU KE CLOUDOVÉ SLUŽBĚ POMOCÍ ČIPOVÝCH KARET

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR AUTOR PRÁCE	Bc. Petr Muzikant
SUPERVISOR VEDOUCÍ PRÁCE	doc. Ing. Jan Hajný, Ph.D.

BRNO 2022

Master's Thesis

Master's study program **Information Security**

Department of Telecommunications

Student:	Bc. Petr Muzikant	***ID:***	200517
Year of study:	2	***Academic year:***	2021/22

TITLE OF THESIS:

Cloud Service Access Control using Smart Cards

INSTRUCTION:

This work focuses on the design and implementation of advanced user authentication methods for accessing the Nextcloud open-source cloud storage. The aim of the thesis is to study the issues of access control, authentication and structure of the Nextcloud project. The actual contribution of the thesis is the analysis and exploration of advanced Nextcloud login options and subsequent implementation of an authentication method based on smart cards. This solution should be tested in practice using internal faculty facilities. The work will also include the implementation of advanced authentication protocol specified by the supervisor. The resulting solution should provide advanced, secure and error-free authentication with respect to user-friendliness and confidentially transmit the result of the operation to the Nextcloud server which will provide access to the cloud storage.

RECOMMENDED LITERATURE:

[1] Menezes, Alfred, Van Oorschot, Paul C. a VANSTONE, Scott A.. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] MULTOS Developer's Guide [online]. , 96 [cit. 2021-09-09]. Dostupné z: https://multos.com/wp-content/uploads/2020/09/MDG.pdf.

Date of project specification:	7.2.2022	***Deadline for submission:***	8.8.2022

Supervisor: doc. Ing. Jan Hajný, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
Chair of study program board

ABSTRACT

Using personal smart cards for authenticating into web services (like cloud storage) is not trivial due to the lack of open-source implementations, standardisations and documentation. This thesis explores the possibilities of such operation and provides a solution based on the Estonian Web-eID tool-set. Its main purpose is to enable state-issued electronic ID cards to communicate with web services, but thanks to its extensibility and being an open-source solution, it is suitable even for custom-build web applications and smart cards. Thesis outcomes include an extension of this repository in form of a highly requested new authentication token validation library for PHP language, installable Nextcloud application enabling Web-eID 2-factor authentication and also a new JavaCard applet fully compatible with the Web-eID solution. Implementation results with the developed open-source JavaCard applet are presented as well as evaluation of the potential Czech electronic ID card usage with the Web-eID system for easier and faster deployment in the future. Contributions are also aimed to help developers implement this strong-authentication method in their web application services using a high-level solution without being relied on actual state-issued or proprietary solution.

KEYWORDS

Web-eID, Smart card, Web application, Cloud storage, Authentication

ABSTRAKT

Využití osobních čipových karet pro autentizaci do webových služeb – jako je cloudové úložiště – není triviální kvůli nedostatku open-source implementací, standardizací a dokumentace. Tato práce zkoumá možnosti takové operace a nabízí řešení založené na estonské sadě nástrojů Web-eID. Hlavním účelem tohoto projektu je umožnit komunikaci státem vydávaných elektronických průkazů totožnosti s webovými službami, ale díky své rozšiřitelnosti a open-source licenci je vhodný i pro obecné webové aplikace a vlastní, personalizované čipové karty. Výstupem této diplomové práce je rozšíření projektu v podobě velmi žádané nové knihovny pro validaci autentizačních tokenů pro jazyk PHP, dále instalovatelná Nextcloud aplikace umožňující dvoufaktorovou autentizaci s Web-eID, a také nový JavaCard applet, plně kompatibilní s Web-eID. V práci jsou rovněž prezentovány výsledky implementace s vyvinutým open-source JavaCard appletem a vyhodnocení potenciálu využití českých elektronických občanských průkazů se systémem Web-eID pro snadnější a rychlejší nasazení v budoucnu. Dodatečným cílem práce je také výpomoc vývojářům implementovat tuto silnou autentizační metodu do jejich webových aplikačních služeb, aniž by museli být odkázáni na státní, proprietární či nemoderní řešení.

KLÍČOVÁ SLOVA

Web-eID, čipová karta, webová aplikace, cloudové úložiště, autentizace

ROZŠÍŘENÝ ABSTRAKT

Úvod

Cloudové služby v dnešní době zaznamenávají značný rozmach ve všech informačních oblastech od soukromých aplikací pro jednotlivé uživatele po sdílený výpočetní výkon pro multi-korporátní operace. Zabezpečení dat a autentizace entit proto čím dál tím více hraje podstatnou roli při využívání těchto služeb.

Jednou z možností moderní autentizace je využití čipových (chytrých) karet. Jak již z názvu vyplývá, tyto karty obsahují zpravidla méně výkonný, nenáročný mikročip, pro který je možné programovat a instalovat jednoduché aplikace v podobě sad instrukcí. Výrobě těchto karet předcházelo vytvoření standardů, kterými čipové karty komunikují se svoji čtečkou, resp. s připojeným zařízením. Tyto standardy se však primárně zaměřují na komunikaci, které probíhá lokálně (např. přes USB rozhraní), což značně komplikuje využití karet při autentizaci do cloudových služeb, ke kterým uživatel běžně přistupuje pouze přes webový prohlížeč.

Důvodem jsou dnešní – poměrně striktní – bezpečnostní politiky webových prohlížečů, co se komunikace vzdálených serverů s periferiemi počítače týče. Praxe je proto taková, že se využívá tzv. middleware v podobě dodatečně nainstalovaného software, který zprostředkovává komunikaci mezi lokálně připojenou čipovou kartou a serverem (většinou s pomocí nově vytvořeného zabezpečeného kanálu). Například pokud chce uživatel přistupovat do internetového bankovnictví s pomocí čipové karty, je nutné si nejdříve nainstalovat bankou poskytnutou aplikaci.

Tato práce se věnuje primárně výzkumu možností pro přihlašování uživatelů pomocí komunikace čipové karty s cloudovou službou. V případě lokálně nainstalovaného middleware je požadováno, aby řešení bylo open-source, snadno instalovatelné, podporované a do jisté míry bezpečné. Práce se také věnuje rozšířenými možnostmi autentizace do cloudového úložiště Nextcloud, které finálně zvládne akceptovat výsledek autentizace s využitím čipových karet. Nakonec se práce věnuje výzkumu potenciálního využití českých elektronických občanských průkazů.

Popis řešení

Po teoretické části práce – jejíchž součástí je i výběr cloudového úložiště Nextcloud – následuje zhodnocení aktuálního stavu dostupného software pro využití čipových karet s webovými službami. Závěrem tohoto hodnocení je, že možnosti pro efektivní a moderní komunikaci čipové karty s webovým serverem nejsou triviální z důvodu nedostatku open-source implementací, standardizací a dokumentace. Aktuální řešení ve veřejném (např. státní elektronická identita) a soukromém (např. internetové bankovnictví) si základá na proprietárním a closed-source systému na míru.

Výjimkou předchozího tvrzení je analyzované řešení Web-eID, vyvíjené Estonskou Informační Autoritou (oficiální zkratka RIA) pro estonské veřejné i soukromé internetové služby. Jedná se o sadu nástrojů, které dohromady zajišťují autentizaci a digitální podpis pomocí estonského elektronického občanského průkazu s využitím asymetrické kryptografie. Díky modulárnímu designu, flexibilitě a open-source licenci je možné Web-eID použít i pro personalizované čipové karty s vlastní certifikační infrastrukturou. Zároveň je systém vyvíjen jako nástupce (nejpozději do konce roku 2023) aktuálně používaného řešení *Open Electronic Identity* se záměrem předejít novým technickým překážkám, zmodernizovat, zabezpečit a zpřístupnit nové řešení široké veřejnosti (včetně dalších členských států Evropské Unie). Jedná se tedy o perfektní řešení pro tuto diplomovou práci.

Pro využití Web-eID k účelům, ke kterým nebylo původně zamýšleno, je třeba provést jeho konkrétní modifikaci a rozšíření. Tato diplomová práce tedy prezentuje výsledky těchto úprav, popisuje vývoj appletu čipové karty (tj. zkompilovaný program nainstalovaný na kartě) nejprve pro MultOS kartu, posléze pro JavaCard, a nakonec uvádí novou, instalovatelnou Nextcloud aplikaci, která zpřístupňuje Web-eID autentizaci jako druhý autentizační faktor pro přihlášení uživatele do cloudového úložiště.

Shrnutí

V rámci diplomové práce jsem provedl krátkou analýzu cloudových úložišť, analýzu současného stavu využívání čipových karet pro webové služby a také jsem představil velmi jednoduchý Proof-of-Concept s MultOS kartou a komunikací přes internetové sockety. Samotnými výsledky je však několik aplikací a nástrojů, sloužících k zpřístupnění velmi slibného řešení na potencionálně multi-státní úrovni pro open-source cloudové úložiště Nextcloud.

První aplikací je instalovatelný 2FA modul pro Nextcloud. Využívá Web-eID JavaScript API pro komunikaci s upravenou (viz dále) Web-eID nativní aplikací a zároveň je součástí vnitřního Nextcloud frameworku, díky kterému umožňuje administrátorovi u jakéhokoliv uživatele zaktivovat přihlášení s pomocí čipové karty jako druhý autentizační faktor. Tuto aplikaci jsem naprogramoval v jazycích PHP, JS, HTML a CSS.

Dále jsem vytvořil JavaCard applet, který z prázdné Java karty vytvoří nástroj pro autentizaci a digitální podpis, plně kompatibilní s upravenou verzí Web-eID. Jako inspiraci jsem využil open-source applety *FakeEstEID* (imitace funkcí estonského občanského průkazu) a *IsoApplet* (naprogramovaná podle relevantních ISO standardů). Jedná se tedy o moderní a funkční aplikaci čipové karty, jejichž operace jsou zabezpečené nutností zadat příslušný PIN, který se ověřuje přímo na kartě. Aby mohl administrátor s prázdnými Java kartami manipulovat, vytvořil jsem také management konzoli umožňující jejich inicializaci (tj. generace klíčových párů, nastavení PIN hodnot a vytvoření, podepsání a uložení certifikátů veřejných klíčů). Applet jsem naprogramoval v jazyce Java pomocí developerské sady JavaCard SDK a konzoli jsem naprogramoval v jazyce Python.

Aby mohla Web-eID nativní aplikace (tj. middleware nainstalovaný na počítači uživatele) komunikovat s novým JavaCard appletem, bylo nutné aplikaci modifikovat a rozšířit. Konkrétně jsem implementoval vnitřní rozhraní *ElectronicID*, ve kterém jsem specifikoval, jaké konkrétní řetězce bytů musí aplikace na kartu odeslat, aby dostala zpět požadovaný výsledek (např. prefixy pro ověření PIN, řetězce pro výběr a stažení certifikátu na kartě). Takto modifikovaná aplikace poté nahradí tu oficiální a umožní tím uživateli používat vlastní Java kartu namísto oficiálního estonského občanského průkazu. Rozhraní i další modifikace (definice ATR pro nové rozhraní) jsem provedl v jazyce C++.

Největším přínosem pro obecnou developerskou komunitu je nová Web-eID validační knihovna v jazyce PHP. Aktuálně totiž součástí projektu existují pouze validační knihovny pro webové aplikace napsané v Javě nebo C#. Jelikož back-end Nextcloud využívá jazyk PHP, dohodl jsem se s vývojáři Web-eID, že v rámci diplomové práce také vytvořím údajně žádanou validační knihovnu v PHP. Tyto knihovny mají za úkol generovat a bezpečně uchovávat autentizační výzvu a poté validovat výsledek celého autentizačního procesu (tj. ověřit digitální podpis a certifikát veřejného klíče, který přísluší podepisujícímu privátnímu klíči na kartě). Tuto knihovnu jsem vytvořil podle existující knihovny v Javě tak, aby prováděla téměř identické operace. V rámci tohoto jsem se musel potýkat převážně s knihovnami, které (narozdíl od Javy) v jazyce PHP chybí a naprogramovat tak dané funkce sám (např. OCSP klient). Setkal jsem se tedy s jazyky PHP, Java a C#, ale také jsem si osvojil zkušenosti a znalosti v oblastech certifikační architektury, digitálního podpisu, a především ASN1, BER, DER kódování.

Zhodnocení výsledků

Výsledkem práce je efektivní a bezpečné řešení pro využití čipových karet k autentizaci uživatelů do služby Nextcloud. Z pohledu uživatele je průběh autentizace (včetně instalace) velmi intuitivní a díky nástroji Web-eID dostává dynamicky informace o tom, co má pro úspěšnou autentizaci udělat (zobrazují se mu instrukční hlášky k připojení čtečky, zasunutí karty do čtečky, zadání PINu, do které služby se přihlašuje, atd.). Na druhé straně administrátor má k dispozici zkompilovaný applet a management konzoli, díky které dokáže rychle, formou dialogu a automaticky personalizovat novou Java kartu pro daného uživatele. Díky Nextcloud management konzoli pak jedním příkazem dokáže aktivovat druhý autentizační faktor s využitím Web-eID pro stejného uživatele. Časové měření ukázalo, že přivedení autentizačního postupu na základě digitálních podpisu a certifikátů do procesu přihlašování přináší (mimo neovlivnitelné vstupy uživatele) zpoždění v řádech setin až desítek sekund.

MUZIKANT, Petr. *Cloud Service Access Control using Smart Cards*. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2022, 81 p. Master's Thesis. Advised by doc. Ing. Jan Hajný, Ph.D.

Author's Declaration

Author:	Bc. Petr Muzikant
Author's ID:	200517
Paper type:	Master's Thesis
Academic year:	2021/22
Topic:	Cloud Service Access Control using Smart Cards

[*]The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to thank my thesis supervisor, doc. Ing. Jan Hajný Ph.D. for the professional guidance, support, and patience. I would also like to thank my Estonian consultants Arnis Paršovs and Mart Sõmermaa for sharing their high expertise in smart cards and authentication applications with me as well as providing helpful and encouraging answers to my questions. Finally, I would like to thank my girlfriend for her support and tenacity during my stay in Estonia and my parents for their overall support.

Contents

Introduction **23**

1 Theoretical Introduction **25**

1.1 Cloud Service 25

1.2 Cloud Storage 26

1.3 Authentication, Authorization, and Access Control 28

1.4 Smart Cards 30

2 Current State and Existing Technologies for Connecting Smart Cards and Web Services **35**

2.1 State of the Art 35

2.2 Problem Definition 38

2.3 Proof-of-Concept 39

2.4 Web-eID 42

3 Practical Results: Nextcloud Access Control System and Its Implementation **47**

3.1 Nextcloud Authentication Solution 47

3.2 InfinitEID: Fully Web-eID Compatible Card Colution 51

3.3 Web-eID Extensions and Modifications 55

4 Overview and Verification of Results **59**

4.1 Administrator's Point of View 59

4.2 User's Point of View 61

4.3 Authentication Duration Measurements 64

5 Next Steps: Czech eID **67**

Conclusions **69**

Bibliography **71**

A Code listings **77**

B Content of the electronic attachment **81**

List of Figures

2.1 The result of the script used to test the functions of the MultOS smart card 40
2.2 Terminal application used to control the MultOS smart card 41
2.3 Card-Connector logs 41
2.4 Web-eID full overview by Mart Sõmermaa 44
2.5 Web-eID authentication communication diagram by Mart Sõmermaa 46
4.1 General overview of Nextcloud–Web-eid architecture with highlighted contributions 59
4.2 InifinitEID applet build and reload process 59
4.3 InfinitEID management console 60
4.4 InfinitEID initialization finished 60
4.5 Web-eID compatibility unit tests result 61
4.6 Enabling Web-eID second-factor authentication for ncadmin user . . . 61
4.7 Web-eID web-browser extension installed 61
4.8 Replacing extended and built version of Web-eID native application with the original one 62
4.9 Nextcloud login page 62
4.10 Web-eID second-factor authentication page 63
4.11 Web-eID authentication PIN prompt 63
4.12 Nextcloud web-interface example 64
4.13 On-card digital signature of pre-hashed authentication value measurement 65
4.14 PHP back-end validation of Web-eID authentication token measurement 66

Listings

3.1 build.xml file for building CAP file 53
A.1 Example session-backed challenge nonce store implementation 77
A.2 Challenge nonce generator builder usage example 78
A.3 Authentication token validator configuration example 78
A.4 Example of providing challenge nonce to the client 79
A.5 Front-end code for obtaining authentication token and sending it back for validation 79
A.6 Nextcloud authentication result configuration 80

Introduction

Cloud services are experiencing significant growth in all information domains, from private applications for individual users to shared computing power for multi-corporate operations. Therefore, data security and entity authentication are increasingly essential in using these services.

One modern authentication option is the use of smart cards. As the name implies, these cards generally contain a low-power microchip for which simple applications in the form of instruction sets can be programmed and installed. Production of these cards was preceded by the creation of standards by which smart cards communicate with their reader or connected device. However, these standards primarily focus on communication that takes place locally (e.g., via a USB interface), which significantly complicates the use of the cards for authentication to cloud services generally accessed by the user via a web browser.

This complication is due to today's - rather strict - security policies of web browsers regarding the communication of remote servers with computer peripherals. Therefore, the practice is to use middleware as retrofitted software that mediates communication between the locally connected smart card and the server (usually using a newly created secure channel). For example, if a user wants to access internet banking using a smart card, it is necessary first to install an application provided by the bank.

This thesis investigates the possibilities for users to log in using smart card directly with the cloud service without the need to use locally installed software or with minimal and efficient install process. It also explores the possibilities of more advanced authentication protocols on this new system.

The thesis contains a theoretical overview chapter 1 describing cloud services, cloud storages, authentication, and smart cards in general. In the next chapter 2, I focus on the current state of connecting smart cards and web services, define the problem this thesis is trying to solve, implement simple proof of concept and describe chosen solution Web-eID in more detail. Chapter 3 then describes practical results in the form of an installable Nextcloud module for second-factor authentication, Web-eID compatible JavaCard applet, a new Web-eID authentication token validation library for PHP and Web-eID extensions. Chapter 4 provides a reader with an overview of results from the user's and administrator's point of view and verifies its usability by measuring authentication durations. In chapter 5, I explore the possibilities of using Czech eID cards in combination with Web-eID technology. The last chapter 5 concludes this master thesis.

1 Theoretical Introduction

This chapter describes cloud services, authentication in general, and smart cards. The acquired knowledge is further used in the practical part, in which the identification of options, their selection, and implementation is carried out.

1.1 Cloud Service

Cloud service or cloud computing is defined as cheap, easy to access, immediate, and remote access to computing resources or resources in various forms that are operated and managed by a third party. This system is then made available to the user themselves via an Internet connection without the need to install additional software [1].

The data flow is specific to the cloud service. The data usually travels from the client part (i.e., from the users through the web interface) through the network to the server part (i.e., the internal system of the service provider), where the necessary operations are performed locally. The operations' results then travel back to the user, whereby the user can invoke further operations.

The advantage of cloud services is primarily the *easy and fast access to modern technologies* in computing, analytical, database, and other areas. Furthermore, their *elasticity* in the form of scalable use of computing resources - users will always use the exact amount of resources corresponding to their requirements. The last advantage is the relatively *low cost* [2]. More information about cloud services is available at [3] and [4].

1.1.1 Categorization of Cloud Services

,

These services can be divided into two metrics: what they provide and who uses them. In the first case, we divide cloud services in descending order of abstractness into:

- *Infrastructure-as-a-Service (IaaS)*: provides basic computing, network, and database resources with the most significant degree of flexibility, control, and management.
- *Platform-as-a-Service (PaaS)*: provides pre-built and controlled platforms on which one can develop own system with applications. Users do not have to worry about the resources themselves but only about how they use them.

- *Software-as-a-Service (SaaS)*: provides the product, application, service, or function itself. The end-user should not be able to recognize that the application is not running on their machine but remotely.

The second division is based on accessibility to the general public:

- *public*: services are for those willing to pay for them, regardless of the user's occupation, background, or interest. At the same time, the user has no control over where their data travels for processing.
- *Private*: services provided only within one or more closed networks, e.g., in a specific organization or company. User data is processed on the internal infrastructure of the object.
- *Hybrid*: combines public and private cloud services, where the more sensitive data is processed within the internal network, and the other data is processed on the public network. This solution is more cost-effective than private cloud computing. However, it brings particular implementation challenges [5].

1.2 Cloud Storage

Cloud storage is one of the implementations of cloud services. Its purpose is clear from its name: it is used to store data on a remote server easily and provides users with *easy manipulation*, *instant accessibility*, and *sharing capabilities*. Therefore, they are very often used in organizations and companies where groups and teams of employees access data. The security of the data depends on the security of the storage server itself (or multiple storage servers) [6].

In the most uncomplicated design, a single server is connected to the network. Users on this network access it through a web interface and upload copies of data of their choice. At any time in the future, they can then access and manipulate the data through the same web interface or download copies back to their device. Superstructures and additional functionality are then built on top of this system: e.g., versioning, data storage in multiple locations in case of hardware failure, access control for users and groups, and a tagging system [5].

1.2.1 Why Cloud Storage?

Results from this thesis will contribute to a multi-academical project focused on developing a secure system for *storing and handling electronic evidence for law enforcement*. Therefore, I am researching the usage of smart cards for authenticating into the web interface of cloud storage, which will store this electronic evidence. This system's requirements are listed in the following subsection 1.2.2, where I also pick a suitable cloud storage solution.

1.2.2 Overview of Existing Cloud Storages

This section is used to select suitable cloud storage for this thesis. The main requirements for such an implementation are:

- *open-source*: openness and transparency of the code allow implementation of custom infrastructure and additional features. For a proprietary system, it would be problematic to change and adapt the system source code in the case of developing smart card authentication.
- *Self-hosting* is the ability to download the solution to one's device and install, configure, run and test it locally.
- *Native options for system development and custom features*: cloud storage that is natively programmed with a modular design is preferred. That is, the solution developers anticipate adding new features in the future and appropriately prepare the system for it.
- *Active development*: to maintain the relevance of the results of this work, it is crucial that the system is currently being developed with good prospects for future development (regular updates, active development team, active community, population).
- *Access control*: the solution should have an existing implementation of access control.
- Finally, other general requirements related to security (ensuring confidentiality, integrity, and availability of data), server- and client-side encryption, a low level of discovered security vulnerabilities, and the relevance of the cryptographic algorithms and keys used are welcomed.

Out of the original 31 solutions analyzed (complete analysis available in [7]), I have selected the final four solutions with the most optimal compliance with the requirements. These solutions are listed in the following subsections.

Nextcloud It is the most widely used open-source cloud storage on the market and meets all specified requirements. Proof of its popularity is also the fact that foreign governments actively use this system [8]. Another advantage is the great emphasis on security and the well-managed documentation of security services. All features are already included in the free, open-source version. In the paid Enterprise version, only user support, expert provision, and priority vulnerability reporting are included [9].

OwnCloud The previous Nextcloud solution was created in April 2016 by the separation of several developers and founders of the ownCloud cloud storage. Therefore, the two solutions are not too different from each other, but the development of both continued along their paths. The most significant difference is the licensing policy, in which, for a fee, ownCloud offers features that are not in the open-source version [10]. OwnCloud fulfills all the requirements. However, only the web interface can be used for encryption on the side, not the installed native client.

SeaFile The SeaFile cloud solution meets all requirements except one: client-side encryption does not encrypt metadata, which poses some security risk [11] – specifically, the directory list, file names and sizes, and edit history. However, this is not a serious enough deficiency for the solution sought to exclude the solution from the selection.

MinIO All the solutions mentioned so far have been cloud storage solutions. MinIO, however, is a "distributed object storage system". The data is spread across multiple locations (separately or as duplicates), providing a data distribution. The object storage system then stores the data in object databases. MinIO's licensing policy is similar to that of Nextcloud. Thus, for a fee, it only offers support, an offer of expertise, and no additional features. Unfortunately, unlike the others, MinIO does not provide client-side encryption.

1.2.3 Choosing Optimal Solution

To meet the objectives of this thesis, I decided to use a cloud service in the form of an open-source cloud storage service Nextcloud based on existing solutions and their requirements. This decision is mainly due to the participation in a prototype for a faculty project, therefore, existing experience. Furthermore, the market's popularity, the documentation's quality, and the large, active community. The whole system is also licensed under AGPLv3 [12]. Hence the system can be modified and distributed provided the license, changes, and source code are included.

1.3 Authentication, Authorization, and Access Control

This subsection describes the authentication process, which is an integral part of this work in connection with smart cards. By this process, we commonly refer to *verifying the identity of a given subject* using specific data or procedures that uniquely identify that subject. The goal of authentication is to detect a false identity (pretending to be someone or something they are not) and prevent them from accessing resources or assets they should not have access to [13].

Naturally, successful authentication creates trust in the authenticated entity. The authentication process is much more applicable in the digital world, where creating a false identity is relatively easy due to the physical anonymity of the network. We distinguish between authentication of an entity (i.e., a person, a user, but also, e.g., software performing an action) and authentication of a message (i.e., its sender).

Authentication Factors Authentication factors are methods of authenticating a subject (specifically a user) based on his:

- *Knowledge factors*: e.g., knowledge of a password, a pre-shared secret, a correct answer to a challenge or control question.
- *Ownership factors*: the user demonstrates physical ownership of, e.g., a hardware key, smart card, one-time, time-based password, or phone number.
- *Inherence factors*: the user proves something that is part of him or herself. These are mainly biometric properties such as a fingerprint, retinal image, DNA sequence, or handwritten signature.

We call these divisions *factor classes*. We also distinguish the level of security (trustworthiness of the authentication result) depending on how many factors are used.

Single-factor authentication uses only one specific factor from any factor class. However, this solution is often not considered sufficient and is commonly resorted to only for financial or time reasons. As soon as possible, to improve security, authentication should be switched to *multi-factor authentication*, which combines two or more factors from different factor classes (e.g., password knowledge and smart card ownership). This method does not only serve to strengthen authentication once but also to re-authenticate an already authenticated user. For example, to log in to the system, the user had to provide two different factors, but for a more sensitive operation in the system, the user has to provide a third factor (e.g., a one-time password) [13].

Authorization After successful authentication, we can then perform the authorization process. This time it is about *verifying the level of access*. The authorization policy specifies what the subject is authorized to do – for example, manipulate resources and assets or perform certain operations.

A single entity can have multiple permissions assigned to it, and it can also belong to a group of entities with one common permission. The most usual subject of authorization are rights to operations (read, write, delete, change file owner) and access rights to various services and resources (e.g., an authenticated administrator will be assigned more rights in the system to a given process than an authenticated ordinary user) [14].

Access Control The authorization serves only to define the individual access rights of the subjects. Therefore, access control physically or logically takes care of enforcing these privileges. Thus, it is sector-specific implementations in which assets must be protected from unauthorized entities [15][16].

According to [5], [15] and [16], there are several basic access control models:

- *DAC (discretionary access control)* is one of the first access control models. Each object in the system is assigned a list of all subjects with associated rights for the object. Essentially, each owner of a resource or asset individually assigns access rights to all others. This approach has the advantage of reducing the burden on the system administrator. The disadvantage of the model is the uncontrollable flow of data. A user with the right to read data may pass it on to those who do not have that right.
- *MAC (mandatory access control)* introduces security levels into the system for both subjects and objects. These levels are then managed by the administrator/central authority. For a subject to successfully interact with an object, its security level must be equal or higher.
- *RBAC (role-based access control)* is the most widely used model today, assigning roles to individual subjects or groups of subjects representing certain hierarchical functions. Thus, users can only perform operations their roles allow them to perform. At the same time, access rights to objects are defined based on individual roles.

1.4 Smart Cards

This chapter focuses on a specific means of authentication, namely smart cards. They can be defined as technical devices providing authentication and security functions in the form of key storage, shared secrets, and cryptographic calculations. These plastic cards, which usually incorporate a small chip (also an antenna in the case of contactless cards), fall into the *ownership factor* class and are often used in multi-factor authentication.

The concept of cards has been commonly used for over a century, e.g., to prove membership or a valid transport ticket. In the 1970s, it also became massively widespread in the likes of credit and debit cards. Plastic cards continued to be improved to provide unquestionable authentication (e.g., engraving of names on the card, magnetic strips). However, these physical modifications were not enough to stop the creation of counterfeits.

A decade later, the first combinations of plastic cards and integrated circuits (i.e., microchips) began to take hold in the market. These cards are nowadays called *IC cards (integrated circuit cards)* or *smart cards* and are widely used in many areas (banking, employee or patient identification, and transportation) [17].

1.4.1 Categorization of Smart Cards

As the previous text indicates, smart cards can be divided into *contact* and *contactless cards* (or a *hybrid card*, a combination of both). The advantage of a contactless card is more significant user-friendliness, longer lifetime, and reliability. On the other hand, however, they come with certain security risks in the form of easier eavesdropping.

Another division of smart cards lies in the design of the chip itself concerning its use. *Single-application* smart cards are designed to perform only one specific operation, which cannot or is difficult to change from the moment of manufacture. These cards have access only to static memory, and for the most basic ones, there is no ability to combine instructions to create an executable program. For more complex single-application cards, these programs can be created, but only once, at production. Therefore, the use of these cards can vary enormously - e.g., only reading a stored identifier or pre-shared secret, computing a response to a challenge, or a simple (a)symmetric cryptographic function.

Then there are *multi-application* smart cards, which are fully programmable and allow one or more pre-compiled programs to be loaded into memory (when manipulating the card, one always selects at the beginning which application one wants to work with). Consequently, they have access not only to static memory but also to dynamic memory in which each application's data is stored while running. These cards are much more popular on the market due to their multi-functionality – an employee can use one physical card to authenticate access to the company building, pay for lunch, and identify himself in the printer. However, the greater complexity of the chip brings higher costs and software development requirements.

1.4.2 Communication with Smart Cards

A smart card reader is required to communicate with the smart card. This device powers the card's integrated circuit via a contact plate and sends a reference signal for data transmission: the clock signal. For the data transport itself, two transport protocols are defined in the ISO 7816-3 standard [18]:

- *T=0*: this is an older, byte-oriented protocol (the basic transmission unit is one byte). The reader (acting as a client) always initiates communication, and the card (acting as a server) responds to these commands. The protocol also

defines the form of transmitted messages in the form of APDU (Application Protocol Data Unit) commands and responses with a specific header format. More about APDUs is described in 1.4.2.
- *T=1*: unlike the first protocol, T=1 is block-oriented, which means that the reader and card exchange whole blocks of bytes. The advantage is higher data throughput and, overall, faster data transfer. The disadvantage, however, is the higher memory requirements of the smart card.

Answer To Reset (ATR) The power supply is initialized when the card is plugged into the reader. At this point (or also later on the command), the card sends a message. ATR is a string of bytes defined by ISO 7816-10 [19] that provides information about supported transport protocols, the clock pulse frequency (used to synchronize the signal between the reader and the card), and also the *historical characters*. These contain information about the card itself, such as manufacturer and serial number, but also application data specified by the programmer [17].

APDU Requests and Responses APDUs are logical units defined by ISO 7816-4 [20] and are transferred between the reader and the card. They are divided into commands sent by the reader and responses sent by the card. Both types of messages have a fixed structure which is described in the following paragraphs.

An *APDU command* contains a mandatory header and an optional part containing application data. In the header, the following fields must always be specified (each field is of size 1 Byte):
- *CLA (Class)* specifies the class of the command used to identify its type.
- *INS (Instruction)* specifies the instruction code used to identify a particular application on the card.
- *P1* and *P2 (Parameter)* indicate the input parameters. They are used to identify the functions of a given application further.

The optional part located immediately after the header may contain the *Lc (length command)* field indicating the size of the following data to be sent. The *Le field* can be added at the end of the command to indicate the expected data size in the response.

The *APDU response*, on the other hand, consists of an optional part and a mandatory field. In the first part, the message can contain the actual data to be returned by the card based on the sent *Le field*. Furthermore, the footer always contains two fields, *SW1* and *SW2 (Status Word)*, which specify the result of the operation performed on the card. These return codes are usually defined in the chip card operating system. Traditionally, the code `0x9000` is interpreted as a successful request processing. On the other hand, codes beginning with `0x6XXX` indicate an error specified in the omitted hexadecimal digits.

ISO Cases Finally, due to the optional part in both APDU commands and responses, the already mentioned ISO 7816-4 standard [20] defines four possible cases (called *ISO Cases*) that can occur during communication. These cases define all possible command combinations and associate the correct response combinations to them. The ISO Cases can be seen in Table 1.1.

Tab. 1.1: Possible APDU message structures

ISO CASE	**APDU request**	**APDU response**
1	`CLA INS P1 P2`	`SW1 SW2`
2	`CLA INS P1 P2 Le`	`[data] SW1 SW2`
3	`CLA INS P1 P2 Lc [data]`	`SW1 SW2`
4	`CLA INS P1 P2 Lc [data] Le`	`[data] SW1 SW2`

Information from this chapter and other information can be found, for example, in the MultOS developer documentation [17] and academic papers [21], [22] and [23].

2 Current State and Existing Technologies for Connecting Smart Cards and Web Services

In this chapter, I focus on using smart cards in web services and applications. First, I provide state-of-the-art research and then describe the problem I am trying to solve in this thesis. The following section describes my early attempts to solve this problem using MultOS smart card and socket connector. However, this technique is only a proof-of-concept since it contains significant security deficiencies. The last section of this chapter introduces Web-eID, which I have chosen for the final implementation.

2.1 State of the Art

The following subsections provide a list of existing technologies which are/were used to provide a communication channel between the smart card and the web server. The main focus is on the availability of the technology for the broad developer community[1].

Netscape Plugin Application Programming Interface (NPAPI) NPAPI is a very outdated technology in the form of an application interface that allows various plugins to run in a web browser. However, support is no longer available in 2022. For example, it was discontinued in Google Chrome in January 2014 [24].

Java Applets Since local communication with the card on the client-side using retrofitted software is relatively standard, Java Applet seems to be the ideal solution. It is a code (usually in Java) that has been compiled into Java bytecode and inserted into the source code of a web page. This applet is invoked and executed by the local JVM (Java Virtual Machine) when the page is loaded [25]. This approach thus eliminated the need to install middleware, as the JVM was until recently a standard tool on most PCs. Java even includes native libraries for communicating with the smart card[2].

Unfortunately, this solution ceased to be functional at the same time as support for the NPAPI technology – on which the applets ran – was ending. Applets were

[1] I.e., how easy it is to integrate smartcard usage into a newly developed general web application.

[2] The library is described in detail at `https://docs.oracle.com/javase/6/docs/jre/api/security/smartcardio/spec/javax/smartcardio/package-summary.html`.

also no longer supported in the Java Development Kit (JDK) [26]. Similarly, modern web browsers have also stopped supporting them.

WebUSB API The WebUSB API[3] was created to allow a web server to access non-standard USB devices for which it is common to install additional software that is platform-specific and developed only by the device manufacturers. Thanks to this API, these vendors can create multi-platform SDKs (Software Development Kit) for their devices [27].

However, this solution is also no longer functional, and in 2018, Chrome developers stopped supporting and started blocking smart cards (along with other selected devices) [28].

Smart Card Connector Smart Card Connector[4] is an extension for the Chrome web browser (or as an application for the ChromeOS operating system) that enables communication with a smart card. The disadvantages are low availability, minimal documentation, and the need to install middleware for the smart card in addition to the extension [29].

However, according to [30], Google plans to stop supporting all Chrome Apps, which also applies to this extension. In a December 2020 support tab on the Chrome Web Store, the developers expressed that they are working on migrating the product, but without any further updates or news on the development.

Webcard It is an open-source add-on[5] for Google Chrome on macOS and Windows operating systems. It links the connected smart card directly to a web page using a JavaScript library. However, according to consultations with supervisors of existing implementations [31] and [32], this solution is not ideal due to inactive development and the need for periodic configuration at the operating system level and enabling security features.

SConnect The SConnect solution is the result of work [33] from 2009. However, it is not easy to find any up-to-date documentation. The original solution website is already redirected to another company. I was able to find already compiled solutions for different browsers[6]. However, decompiling them would be beyond the scope and time of this thesis. I also tried contacting the original authors, but most did not work for the company anymore, and the others did not answer.

[3]Documentation is available at `https://wicg.github.io/webusb`.

[4]Source code is available at `https://github.com/GoogleChromeLabs/chromeos_smart_card_connector`.

[5]Source code is available at `https://github.com/cardid/webcard`.

[6]At `https://www.sconnect.com/extensions/`.

WebAuthn standard Since May 2019, the World Wide Web Consortium (W3C) has been developing a new standard to create a unified interface for web user authentication via asymmetric cryptography [34]. This solution is already supported in standard web browsers such as Chrome, Firefox, and Edge by the spring of 2022. However, I could not find any implementations of it for smart cards. Supported devices include mobile phones, smartwatches, and proprietary USB keys such as FIDO2 and UbiKey.

According to Mart Oruaas from Norwegian cyber-security company Cybernetica, this technology is theoretically possible to use for smart cards. However, its implementation would be very complicated. WebAuthn considers having a separate private key for every authentication endpoint, and this kind of key management would require a deep security analysis of the developed smart card applet.

Create Custom Middleware The last option is to program an additional software that can communicate with the smart card and the web server. This solution is the easiest to implement but at the cost of low user-friendliness and the need to perform extensive security analysis beyond the scope of a single person. I have implemented my middleware in 2.3.

Other Technologies/Tools Finally, I will list here other names of technologies for which some indications of the possibility of linking the smart card with a web server were found during the research. However, there was not enough documentation or sources to support the theory, so it was impossible to look into these terms in more detail: *SignalR*[7] and *Websocket-Smart-Card-Signer*[8].

Proprietary Solutions There are also proprietary and closed-source solutions like *signer-digital*[9], *Fujitsu mPollux DigiSign Client*[10] used in Finland, *AusweisApp2* used in Germany, and *IDEMIA Smartcard Web Connector*[11] [35].

[7] According to `https://stackoverflow.com/a/48799171` the implementation of smart cards is reportedly possible.

[8] Source code available at `https://github.com/damianofalcioni/Websocket-Smart-Card-Signer`.

[9] Available at `https://signer.digital/SignerDigitalBrowserExtensions`.

[10] `https://dvv.fi/documents/16079645/17667177/Fujitsu+mPollux+DigiSign+Technical+References.pdf/a6630863-c268-b56a-992c-b51dfce44617/Fujitsu+mPollux+DigiSign+Technical+References.pdf?version=1.0&t=1595588194569`

[11] `https://chrome.google.com/webstore/detail/idemia-smartcard-web-conn/pmcjdgnohppjkhnadihaonfmdponlcbi`

Web-eID This solution is an open-source project currently being developed by the Estonian Information Authority. The project's main aim is to improve the current system for using state-issued ID cards in the public and private sectors.

Web-eID is designed to be highly extensible, is produced at a high-quality state level, and has a promising relevance in the future. Therefore it is my choice for integrating smart cards into web applications. Alongside my further reasoning, it is explained in more detail in 2.4.

2.2 Problem Definition

The previous sections show that the *current situation with personal smart cards used for authenticating into web services is not trivial, but quite complicated* due to the lack of open-source implementations, standardizations, and documentation.

The common practice in the private sector (e.g., internet banking services) is to use a closed-source, proprietary middleware. Public sector implementations (e.g., national electronic ID cards) usually use closed-source solutions as well (e.g., in the Czech Republic, Finland, and Germany). *Unfortunately, these solutions are usually unavailable to the general development community [35].*

2.2.1 Requirements for Solution Connecting Smart Cards and Web Services

The following list contains identified requirements for a solution, which I will use in this thesis [35]:

- *open-source*: the solution should have an open-source license, so it is possible to adapt it for custom systems.
- *Portability*: user (law enforcement in this case) might frequently access the cloud storage from a new computer, where smart card middleware is not installed. The solution must have no middleware or be installed and initialized quickly and efficiently.
- *Web-focused*: the solution needs to work seamlessly with most modern web browsers and with intended behavior.
- *OS support*: the solution needs to work seamlessly on most modern operating systems and with intended behavior.
- *Security*: authentication results cannot be maliciously intercepted, changed, or crafted by third parties.
- *Speed and integrity*: the solution should be responsive and resilient and should handle tasks in an unnoticeable time from the user's point of view.

- *Modularity*: this feature is welcomed for future non-smart card authentication, e.g., security token integration.

2.3 Proof-of-Concept

As mentioned in 2.1, creating custom middleware seems easy to implement. This section provides such implementation as an elementary proof of concept. For this implementation, I used MultOS card and socket programming.

2.3.1 MultOS Card Programming

At first, the faculty provided a *MultOS Card*[12] and the *Gemalto PC Twin Reader*. However, since programming an authentication protocol for the smart card is not a priority in this proof-of-concept, I have implemented a simple challenge-response authentication protocol.

There are many tools for programming smart cards. However, the simplest way is to use the *SmartDeck* framework (a library providing an API to machine commands on the card and compiling programs written in C) in combination with the *MUtil* tool (uploading copied applications to cards)[13].

In C, I then programmed the smart card to accept four commands (INS = instruction) using *SmartDeck* documentation and previous existing faculty projects:

1. *INS_GET_CURRENT_KEY*: the card returns the active key used to generate the response. This command is for development purposes only.
2. *INS_SET_KEY*: the card receives a new 12B key and loads it into static memory to the location where the active key is stored. This command is for development purposes only.
3. *INS_RESET_KEY*: the card resets the active key and substitutes the default key, which is statically stored in memory and cannot be changed. This command is used for development purposes only.
4. *INS_AUTHENTICATE*: the card receives a 52B challenge, attaches the active key, and computes the SHA1 hash. Unfortunately, the provided card does not support other hashing functions. The SHA1 function has been considered obsolete in [36], [37] and [38].

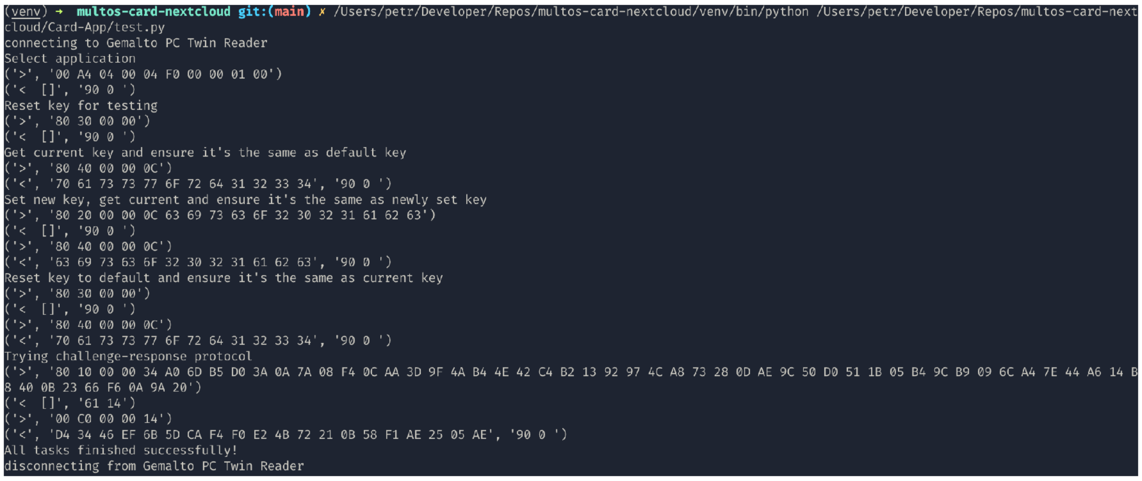
For communicating with the card, it is possible to use the Python package *pyscard*[14]. With its help, I created two scripts designed to test the programmed functions on the card.

[12]A specific card model shall remain nameless due to the non-disclosure agreement.
[13]documentation and free download links available at `https://multos.com/support/`
[14]available from `https://pypi.org/project/pyscard/`

The first script takes the form of a unit test, meaning that if it finishes without errors, all functions work correctly. The script first attempts to connect through the reader to the card and performs the following actions sequentially:

1. it calls the application selection command.
2. Calls the command to reset the active key.
3. Calls the command to obtain the active key and verify that it matches the statically defined default key.
4. Calls the new key set command, calls the key gain command and verifies that it matches the set key.
5. Calls the command to reset the active key.
6. Creates a challenge, calls the authentication command, obtains the response, and verifies that it matches its own computation using a statically defined pre-shared secret.

The result of the test can be seen in Figure 2.1.



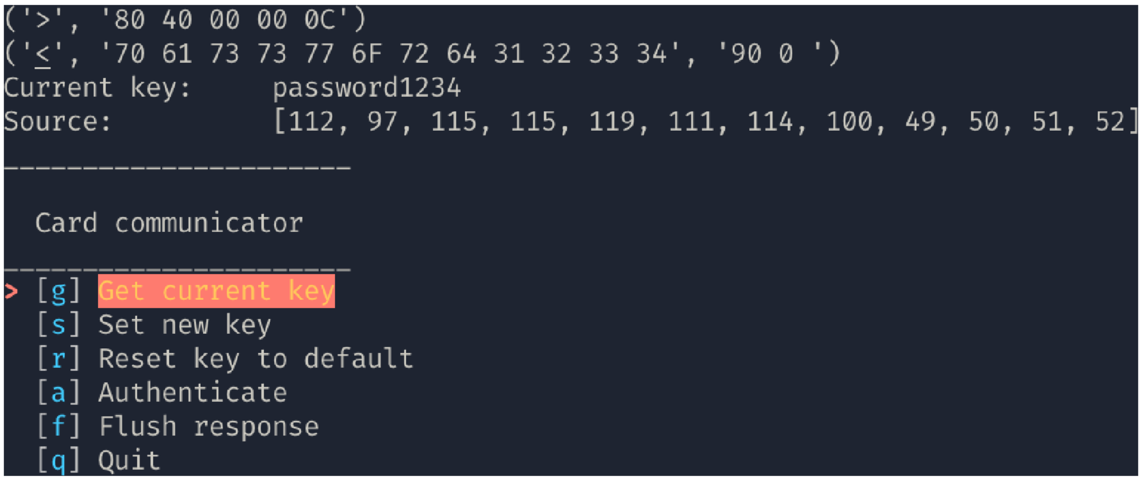
```
(venv) → multos-card-nextcloud git:(main) ✗ /Users/petr/Developer/Repos/multos-card-nextcloud/venv/bin/python /Users/petr/Developer/Repos/multos-card-next
cloud/Card-App/test.py
connecting to Gemalto PC Twin Reader
Select application
('>', '00 A4 04 00 04 F0 00 00 01 00')
('<  []', '90 0 ')
Reset key for testing
('>', '80 30 00 00')
('<  []', '90 0 ')
Get current key and ensure it's the same as default key
('>', '80 40 00 00 0C')
('<', '70 61 73 73 77 6F 72 64 31 32 33 34', '90 0 ')
Set new key, get current and ensure it's the same as newly set key
('>', '80 20 00 00 0C 63 69 73 63 6F 32 30 32 31 61 62 63')
('<  []', '90 0 ')
('>', '80 40 00 00 0C')
('<', '63 69 73 63 6F 32 30 32 31 61 62 63', '90 0 ')
Reset key to default and ensure it's the same as current key
('>', '80 30 00 00')
('<  []', '90 0 ')
('>', '80 40 00 00 0C')
('<', '70 61 73 73 77 6F 72 64 31 32 33 34', '90 0 ')
Trying challenge-response protocol
('>', '80 10 00 00 34 A0 6D B5 D0 3A 0A 7A 08 F4 0C AA 3D 9F 4A B4 4E 42 C4 B2 13 92 97 4C A8 73 28 0D AE 9C 50 D0 51 1B 05 B4 9C B9 09 6C A4 7E 44 A6 14 B
8 40 0B 23 66 F6 0A 9A 20')
('<  []', '61 14')
('>', '00 C0 00 00 14')
('<', 'D4 34 46 EF 6B 5D CA F4 F0 E2 4B 72 21 0B 58 F1 AE 25 05 AE', '90 0 ')
All tasks finished successfully!
disconnecting from Gemalto PC Twin Reader
```

Fig. 2.1: The result of the script used to test the functions of the MultOS smart card

The second script is used for more dynamic testing of the card functions. It takes the form of an interactive menu through which the user can arbitrarily and repeatedly choose which command to send to the card. The results of the instructions are then displayed on the screen. An example of this menu can be seen in Figure 2.2.

2.3.2 Socket Connector

The implemented challenge-response authentication protocol based on pre-shared secrets requires two data messages to be forwarded over the network. The nature of the protocol may not preserve the confidentiality of these messages, so I implemented it using socket programming, i.e., using the socket API to establish a reliable connection between two nodes in the network [39]. The advantage is that creating



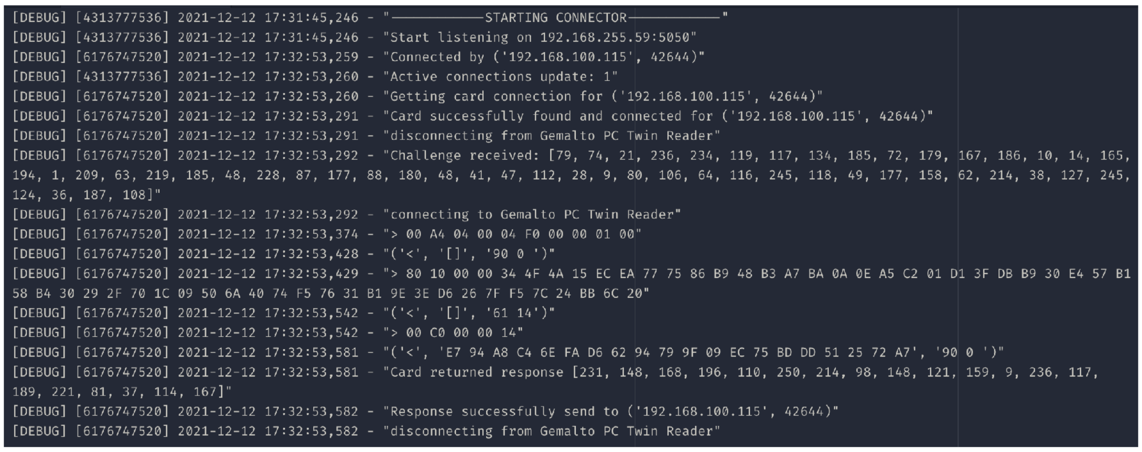
```
('>', '80 40 00 00 0C')
('<', '70 61 73 73 77 6F 72 64 31 32 33 34', '90 0 ')
Current key:     password1234
Source:          [112, 97, 115, 115, 119, 111, 114, 100, 49, 50, 51, 52]
----------------------

  Card communicator

----------------------
> [g] Get current key
  [s] Set new key
  [r] Reset key to default
  [a] Authenticate
  [f] Flush response
  [q] Quit
```

Fig. 2.2: Terminal application used to control the MultOS smart card

sockets and sending and receiving data through these APIs is implemented in almost every programming language and operating system.

I have therefore created two more programs in Python. They are considered client and server in terms of communication between these programs. The server part runs as a process on the user's computer that wants to log in with a smart card. A web server then performs the role of the client with Nextcloud installed, which sends a prompt when the user logs in.

The server listens on a predetermined socket and creates a new thread to serve the request when a new connection is made to the client. It reads the challenge through the open socket and uses the already implemented communication with the card from 2.3.1 to obtain a response, which it then sends back. In Figure 2.3, the operations are shown as program entries.



```
[DEBUG] [4313777536] 2021-12-12 17:31:45,246 - "————————————STARTING CONNECTOR————————————"
[DEBUG] [4313777536] 2021-12-12 17:31:45,246 - "Start listening on 192.168.255.59:5050"
[DEBUG] [6176747520] 2021-12-12 17:32:53,259 - "Connected by ('192.168.100.115', 42644)"
[DEBUG] [4313777536] 2021-12-12 17:32:53,260 - "Active connections update: 1"
[DEBUG] [6176747520] 2021-12-12 17:32:53,260 - "Getting card connection for ('192.168.100.115', 42644)"
[DEBUG] [6176747520] 2021-12-12 17:32:53,291 - "Card successfully found and connected for ('192.168.100.115', 42644)"
[DEBUG] [6176747520] 2021-12-12 17:32:53,291 - "disconnecting from Gemalto PC Twin Reader"
[DEBUG] [6176747520] 2021-12-12 17:32:53,292 - "Challenge received: [79, 74, 21, 236, 234, 119, 117, 134, 185, 72, 179, 167, 186, 10, 14, 165,
194, 1, 209, 63, 219, 185, 48, 228, 87, 177, 88, 180, 48, 41, 47, 112, 28, 9, 80, 106, 64, 116, 245, 118, 49, 177, 158, 62, 214, 38, 127, 245,
124, 36, 187, 108]"
[DEBUG] [6176747520] 2021-12-12 17:32:53,292 - "connecting to Gemalto PC Twin Reader"
[DEBUG] [6176747520] 2021-12-12 17:32:53,374 - "> 00 A4 04 00 04 F0 00 00 01 00"
[DEBUG] [6176747520] 2021-12-12 17:32:53,428 - "('<', '[]', '90 0 ')"
[DEBUG] [6176747520] 2021-12-12 17:32:53,429 - "> 80 10 00 00 34 4F 4A 15 EC EA 77 75 86 B9 48 B3 A7 BA 0A 0E A5 C2 01 D1 3F DB B9 30 E4 57 B1
58 B4 30 29 2F 70 1C 09 50 6A 40 74 F5 76 31 B1 9E 3E D6 26 7F F5 7C 24 BB 6C 20"
[DEBUG] [6176747520] 2021-12-12 17:32:53,542 - "('<', '[]', '61 14')"
[DEBUG] [6176747520] 2021-12-12 17:32:53,542 - "> 00 C0 00 00 14"
[DEBUG] [6176747520] 2021-12-12 17:32:53,581 - "('<', 'E7 94 A8 C4 6E FA D6 62 94 79 9F 09 EC 75 BD DD 51 25 72 A7', '90 0 ')"
[DEBUG] [6176747520] 2021-12-12 17:32:53,581 - "Card returned response [231, 148, 168, 196, 110, 250, 214, 98, 148, 121, 159, 9, 236, 117,
189, 221, 81, 37, 114, 167]"
[DEBUG] [6176747520] 2021-12-12 17:32:53,582 - "Response successfully send to ('192.168.100.115', 42644)"
[DEBUG] [6176747520] 2021-12-12 17:32:53,582 - "disconnecting from Gemalto PC Twin Reader"
```

Fig. 2.3: Card-Connector logs

The second script, in the role of a client, is for testing purposes only and does the exact opposite, i.e., it first attempts to connect to an open server socket, sends a randomly generated challenge, and compares the returned response with its own computation from knowledge of the challenge and the pre-shared secret.

The Nextcloud cloud storage is primarily programmed in PHP, so I also created a client equivalent in PHP. I also created an installable module for Nextcloud, which implements this proof-of-concept in the form of two-factor authentication (see 3.1.2).

2.3.3 PoC Conclusions

Thanks to this proof-of-concept, I have become familiar with smart card programming and Nextcloud authentication module implementations. Since the implemented challenge-response protocol with SHA1 function serves as a demonstration – due to obsoleteness of hash function and no message authentication – the rest of this thesis is mainly focused on improving the communication layer between the smart card and the server.

2.4 Web-eID

Based on the current state-of-the-art situation, I have decided to analyze and use Web-eID. In this section, I describe my reasoning in more detail and introduce the project itself.

Web-eID is an *open-source repository of multiple applications and tools, which together enable the usage of authentication and digital-signing smart card functions on the wide web using public-key cryptography.* Its primary purpose is to enable state-issued electronic ID cards. However, thanks to its extensibility, flexibility, and open-source license, it is suitable even for custom-build web applications with custom smart cards, thus being a perfect, high-level solution for this thesis [35].

It is being developed by the Estonian Information System Authority (RIA in short) to be a reliable, user-convenient solution. Most importantly, it tries to solve the technical challenges of its predecessor, the Open Electronic Identity (OpenID), which is currently used in Estonia for connecting public, but also private web services with electronic, state-issued ID-cards [40].

It is heavily focused on web use cases with web browser platforms, *supporting recent versions of standard operating systems and browsers.* It is also being dissociated from operating systems' inner smart card technologies, using a cross-browser development system and consistent visual elements across all platforms, ultimately allowing *cross-platform, cross-browser, without-restart availability and with no need to install any third-party software.* Authentication and digital signature principles using *public-key cryptography*, *on-card PIN verification*, and *HTTPS-only principle* ensure day-to-day function security. Lastly, it is *designed to be highly extensible* to easily support other security tokens and smart cards [35].

I have chosen this project as it almost perfectly fulfills the requirements listed in subsection 2.2.1:

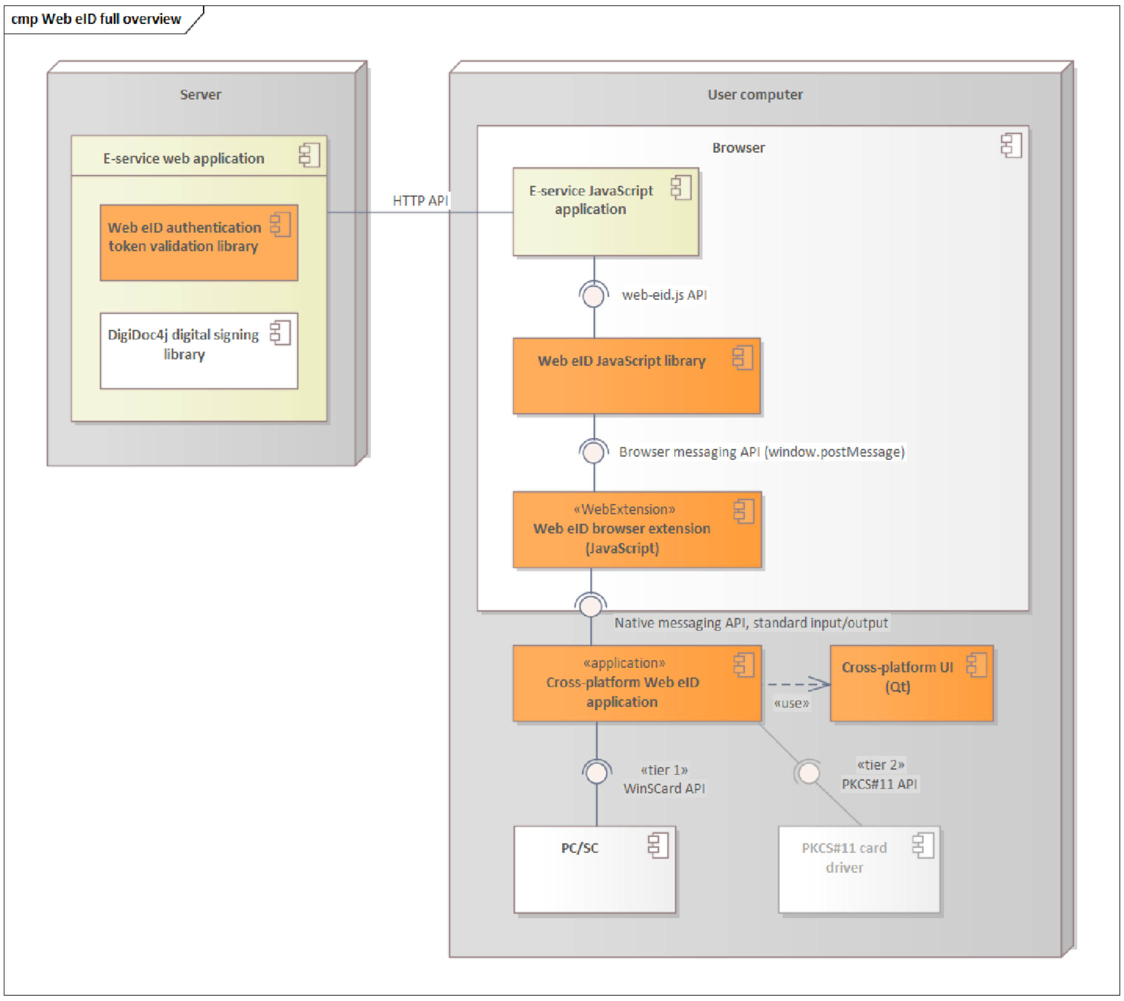
- *open-source*: all of its components are licensed under MIT Licence.
- *Portability*: it can be installed without the need to restart the browser or OS, supporting the immediate launch. It uses standard PS/SC available on most OS' to communicate with smart cards directly.
- *Web-focused*: it works on Windows, Linux, and macOS.
- *OS support*: it works on Chrome, Edge, Firefox, and Safari web browsers.
- *Security*: it is implemented according to Web Authentication API and requires HTTPS protocol to be used (see next paragraph for more details).
- *Speed and integrity*: it seems to be well optimized. There are no noticeable delays and no crashes. Visuals are consistent through all processes.
- *Modularity*: supports an extension for PC/SC, PKCS#11, and other tokens supporting public-key cryptography, such as Yubikey.

2.4.1 Web-eID Architecture

According to [40], main components of Web-eID are:

- *Cross-platform Web-eID Native Application*: running native on Ubuntu Linux, macOS, and Windows, this software communicates with a smart card connected to the computer, provides the user with PIN prompts and information about current operations, and prepares authentication token for the validation library. This component directly communicates with Web-eID browser extensions via *Native messaging API*.
- *Web-eID Browser Extensions*: bundled with the previous application, the cross-browser extension is seamlessly installed and enabled in standard web browsers (Firefox, Chrome, Safari, and Edge). Its task is to directly transfer messages between the native application and the Web-eID JavaScript Library via *Browser messaging API*.
- *Web-eID JavaScript Library*: this library provides an asynchronous interface for invoking Web-eID functions and operations from the web UI through E-service JavaScript code. It poses as a front-end of the server application.
- *Web-eID Authentication Token Validation Library*: as a back-end part of the server application, this library handles generating cryptographic nonce at the start of the process and verification of signed token and attached certificate at the end of the process, ultimately providing validated authentication result.

Graphical interpretation of the connections between these components by the authors of Web-eID is in Figure 2.4.



cmp Web eID full overview
Server
E-service web application
Web eID authentication token validation library
DigiDoc4j digital signing library
HTTP API
User computer
Browser
E-service JavaScript application
web-eid.js API
Web eID JavaScript library
Browser messaging API (window.postMessage)
«WebExtension» Web eID browser extension (JavaScript)
Native messaging API, standard input/output
«application» Cross-platform Web eID application
«use»
Cross-platform UI (Qt)
«tier 1» WinSCard API
«tier 2» PKCS#11 API
PC/SC
PKCS#11 card driver

Fig. 2.4: Web-eID full overview by Mart Sõmermaa [41]

2.4.2 Authentication with Web-eID

The solution utilizes a *digital signature on the smart cards alongside stored digital certificates* for authentication. It uses the same principles as TLS Client Certificate Authentication (TLS CCA) and Web Authentication API (also known as WebAuthn already described in subsection 2.1). The following enumeration provides a shortened and slightly simplified workflow of mentioned principles, also depicted in Figure 2.5:

1. The user tries to log in. His browser requests the server for a cryptographic challenge nonce (session-dependent, only-once value safely generated and stored by the server).

2. The client generates the authentication token using a private key on his smart card.
    (a) Web extension sends the challenge to the native application, which asks the user for consent in the form of PIN verification and obtains a certificate of the public key corresponding to the private key, which is used to create the signature.
    (b) The native application then creates the authentication token, including a to-be-signed authentication value.
    (c) The to-be-signed authentication value is sent to the smart card using APDU commands described in subsection 1.4.2.
    (d) A signed authentication value is inserted into the token, then sent back to the web extensions, which transfer it to the server.
3. The server verifies the signature and the public key certificate to generate the authentication result.
    (a) The signature is verified by reconstructing the authentication value using server origin, and the challenge nonce using the public key from the certificate in the token.
    (b) The certificate in the token is verified using standard certificate verification practices (in this case, with *Online Certificate Status Protocol*).

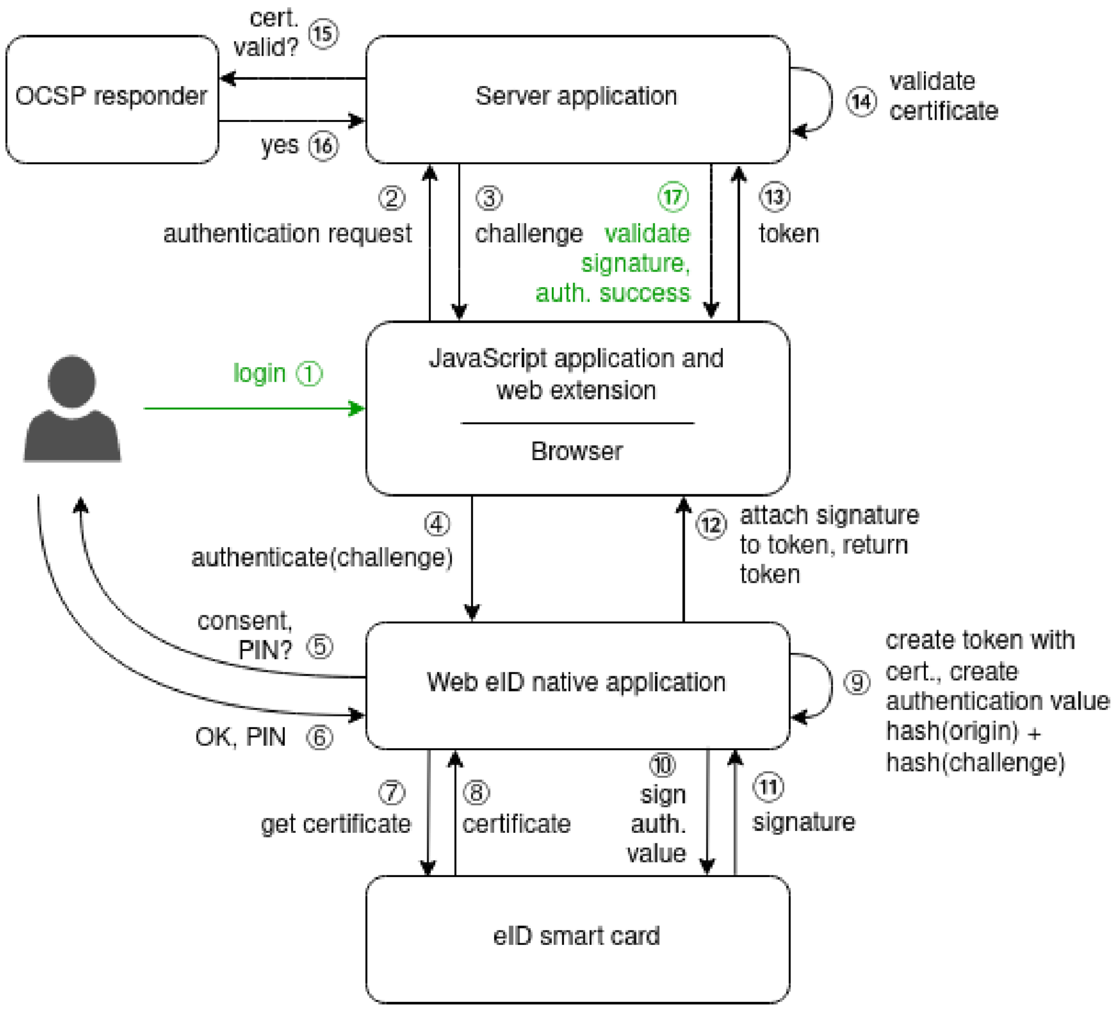
The main difference from the TLS CCA is that this workflow is implemented on the application layer and uses a smart card for client operations. User details and his identity (e.g., username for login or name, surname, and birthname for personal identity) can be obtained from the certificate public subject field.

2.4.3 Security Analysis

A few security analyses have already been done, with some security issues emerging. However, the current assumption of the project's future is set to be positive as it is still in development [35].

First, there is analysis [43]. It introduces relevant technologies, describes similarities and compatibility with TLS CCA, analyses protection against man-in-the-middle attacks, and analyzes threat scenarios with certain assumptions. In the executive summary, the authors mention two possible vulnerable points (simplified – more details are available in the analysis itself):

1. certain scenario/configuration might introduce man-in-the-middle vulnerability into the solution. For example, corporate TLS proxies used for monitoring the traffic of its employees become a point of a single failure.
2. Another difference from TLS CCA is that there need to be some outside countermeasures to prevent session hijacking. The web service, therefore, needs to



cert. valid? ⑮
OCSP responder
Server application
yes ⑯
⑭ validate certificate
② authentication request
③ challenge
⑰ validate signature, auth. success
⑬ token
login ①
JavaScript application and web extension
Browser
④ authenticate(challenge)
⑫ attach signature to token, return token
consent, PIN? ⑤
Web eID native application
OK, PIN ⑥
⑨ create token with cert., create authentication value hash(origin) + hash(challenge)
⑦ get certificate
⑧ certificate
⑩ sign auth. value
⑪ signature
eID smart card

Fig. 2.5: Web-eID authentication communication diagram by Mart Sõmermaa [42]

implement these by itself, for example, by using CSRF tokens.

Second, there is analysis [44], which questions designers' choices to use JSON Web Tokens to carry the authentication proof. According to the article's author, this choice introduces security risks into the system. The article also suggests several improvements to the Web-eID solutions that do not decrease user experience. I plan to observe whether these improvements have been implemented in the Web-eID and if not, I will try to contact the authors to do so (or at least consider the article's suggestions). Lastly, few resources describe the security and protection against specific attacks in [45] and [46].

In conclusion, Web-eID was designed with security in mind from the start. There are some minor issues that can still be solved during the ongoing development. No critical vulnerabilities were found in the Web-eID project. It is expected to be a safe and reliable way of utilizing Estonian (and other countries') state-issued electronic identities in the future.

3 Practical Results: Nextcloud Access Control System and Its Implementation

In this chapter, I am describing the practical results of this thesis and how it solves the problem defined in 2.2. First, I have enhanced Nextcloud's authentication flow so it can accept smart card usage. After that, I created a new applet (application on the smart card) for another type of smart card: JavaCard. Lastly, I describe how I have integrated Web-eID project into this solution and what needed to be done in order to achieve that.

3.1 Nextcloud Authentication Solution

As mentioned in section 1.2.3, I chose Nextcloud cloud storage for my solution. In the following subsections, I explore the natively supported methods of authenticating individual users. The primary purpose of this identification is to find and implement a solution that will be able to integrate smart card data into its authentication process. Therefore, I omit basic authentication options such as name and password combinations.

3.1.1 Overview of Possible Solutions

This subsection lists the solutions that I managed to find during my thesis. For all of them, I give a short description, advantages, disadvantages, and, if necessary, why the solution is insufficient. Unless otherwise stated, the source for the individual solutions is mainly the official Nextcloud documentation for administrators [47] and developers [48]. In the case of "apps for Nextcloud", this refers to the app available on the official Nextcloud App Store [49].

OAuth2 When first searching the admin documentation, one might see the possibility of using OAuth2 technology. However, according to [50] and [51], OAuth2.0 is an *authorization* standard, not an *authentication* standard. This subsection only acknowledges that we are looking for authentication options that precede authorization.

Two-factor Authentication Multi-factor authentication has already been discussed in subsection 1.3. Nextcloud provides information for creating custom second-factor modules in the administration documentation and directly in the developer documentation. It is possible to create, program, and even make it mandatory for specific users or groups.

The module is created as an application, the development of which is described in detail in the development documentation. Some applications for two-factor authentication are already publicly available: e.g., U2F (Universal 2nd Factor, log in with USB or NFC technology), TOTP (Time-based one-time password, log in with an external application with time-varying codes), e-counts and SMS messages.

The advantage of this solution is the possibility of a completely customized solution - Nextcloud only requires the final implementation of the IProvider interface - leading to higher elasticity and better options for connecting to smart cards. On the other hand, the disadvantage is that authentication cannot be entirely replaced by smart cards. In this case, users will still have to log in with a name and password first.

External Authentication Another option is to use the existing Nextcloud External user authentication application, which allows users to log in using services other than the Nextcloud server. Specifically, authentication via IMAP, SMB, FTP, WebDAV, HTTP BasicAuth, SSH, and XMPP. An external server must be set up for each service.

At the same time, the source code of the server and other applications[1] shows that it is possible to program and implement the service separately according to one's wishes.

This option is worth considering, as while we eliminate the possibility of logging in via username and password (thus increasing user-friendliness), we, unfortunately, reduce the level of security by not using an additional factor. The additional external service could run on the same machine where Nextcloud runs.

LDAP Protocol The administrator documentation provides information on using LDAP to log on users through existing organizations, domains, or user lists (such as Active Directory from Microsoft). This option is inappropriate for this work, as it only allows you to set up a new, remote authentication server that supports LDAP, not to change the authentication means (users still log in with a username and password).

OpenID Standard Nextcloud's *OpenID Connect SSO by Gluu* app allows you to authenticate users to any OpenID provider. It is an open standard that enables single sign-on to multiple services at public and private providers [52]. The standard contains many libraries and implementations for OpenID providers. For this application, it is necessary to use the open-source authentication server Gluu.

[1]for example `https://apps.nextcloud.com/apps/user_backend_sql_raw` or `https://github.com/bcecchinato/user_unix_script`

Third-party ID Providers There are also apps on the Nextcloud App Store [49] that allow you to sign in using third-party dedicated apps. These are specifically *Orcid*, *SecSign*, or *miniOrange*. When logging into the repository, a new window will appear and redirect to the service. There, authentication takes place (e.g., using the SecSign mobile app), and the result is returned to the Nextcloud server. This solution is not suitable for this work.

WebAuthn Standard Another option for external login to Nextcloud storage is the Two-Factor Webauthn application. This application directly implements a device that supports the WebAuthn standard, discussed in 2.1. However, as already mentioned, this standard does not support smart cards out-of-the-box.

eID Authentication The last way of logging in is the *eID-login* application[2], which is used to log in directly to the German electronic identity system. The national eID card is used for this purpose. The developers responded to my inquiry about this project by saying that the eID-login solution is unsuitable for sending custom APDU messages or implementing custom smart cards.

However, they referred to the open-source framework *Open eCard*[3] and the proprietary *ChipGateway Protocol*[4]. However, both solutions seem too complicated for this work.

3.1.2 Two-factor Authentication Implementation

For this work, I have chosen a custom two-factor authentication solution. In the following subsections, I describe its development.

Nextcloud Application Skeleton Nextcloud provides the generation of the basic skeleton of the application[5], which contains all the necessary files to install, enable and use it. After downloading the generated package, you need to extract it and put it in the */apps* directory (in my case */var/www/nextcloud/apps*). You can then start editing, adding, and deleting the source files that provide the logic and user interface of the application.

The following is a list and description of the essential files and directories for a basic implementation of two-factor authentication via smart cards (the file paths have the application directory prefix installed in the previous paragraph):

[2]`https://apps.nextcloud.com/apps/eidlogin`

[3]`https://github.com/ecsec/open-ecard`

[4]`https://www.oasis-open.org/committees/download.php/60049/ChipGateway-Specification-OASIS.pdf`

[5]Generation available at `https://apps.nextcloud.com/developer/apps/generate`.

- *appinfo/info.xml*: contains basic information about the application, author, required server version, and web pages. It also identifies special features of the application, such as activity logging, custom sections in settings, and also *providing two-factor authentication.*
- *appinfo/routes.php*: contains an array with special syntax that maps the given application URL to individual functions in *Controller* objects. This file is the primary application disclosure. In my case, this field is empty because, for now, the implemented application does not need any user interaction via URL. It only provides two-factor authentication.
- *css/xxx.css*: files in the CSS directory are used to define HTML styles. They are used in *templates* using the PHP `style()` function.
- *img/app.img*: defines a primary application icon used, for example, when selecting two-factor authentication during login.
- *lib*: the most important application directory that contains all logical operations:
  - *lib/Provider/WebEidProvider.php*: this file contains a class that inherits attributes and functions from the previously mentioned *IProvider*. Therefore, it provides the server with a straightforward interface for using this two-factor authentication method.
  - *lib/Service/SmartCardService.php*: to reduce code redundancy, the *Service* layer is used, which can be inserted into the constructor of all other objects in the application. It contains all helper functions used in *WebEidProvider.php*.
  - *lib/Activity/Provider.php|Settings.php*: these files can log application activity, which can be viewed in the integrated *Activity* application.
  - *lib/AppInfo/Application.php* - contains any code that must be run before using the application. It represents the initialization of the application.
  - *lib/Controller/xxxController.php*: contains the application entry point via the URL specified in *routes.php*.
  - *lib/Db/Entity.php|Mapper.php*: the object inheriting from *Entity* represents the object's definition corresponding to the database record. The object inheriting from *Mapper* contains functions and queries for working with entities and databases.
  - *lib/Event/xxx.php* and *lib/Listener/xxx.php*: provide the application with the ability to listen for and possibly respond to various events on the server.
  - *lib/Migration/VersionxxxDatexxx.php*: objects inheriting from *Migration* define changes in the database, such as creating a new table, changing columns in a table, changing data types in the database, etc.

  - *lib/Settings/Personal.php/Admin.php*: contains options for user or administrator configuration of the application through the web interface settings.
- *templates*: contains PHP scripts for rendering the user interface. In my case, it provides instructions for inserting a tab and a button for initializing authentication with a smart card.
- *src*: contains auxiliary JavaScript files to perform client-side operations. They are used in *templates* using the PHP `script()` function.
- *vendor*: contains PHP packages of third-party applications and frameworks.

Exact Web-eID implementation for Nextcloud is described in section 3.3.3.

3.2 InfinitEID: Fully Web-eID Compatible Card Colution

Web-eID is currently being built primarily for state-issued electronic ID cards. Support for Estonian, Latvian, Finnish, and Croatian eID smart cards is planned for the project launch in the 4th quarter of 2022[6] (other countries are supposed to be supported in the phases). However, I aimed to implement my own "custom self-issued electronic ID card" for this thesis. Since the current smart card solutions are utilizing JavaCards, I have decided to switch from MultOS (used in section 2.3) card to JavaCard as well.

For this thesis, I have borrowed two different JavaCards:

1. *JavaCard SLJ52GCA150 (jTOP SLE78 Estonian ID card platform)*[7][8] from the University of Tartu, Applied Cyber Security Group.
2. *JavaCard NXP JCOP3 J3H145*[9] from the Brno University of Technology, Department of Telecommunications.

In this section, I describe the JavaCard programming in general and introduce my created JavaCard applet and its corresponding management console used for card initialization and self-issuing the certificates saved on the card.

[6] And transition is set to be finished sometime in 2023 according to the official schedule: `https://www.id.ee/en/article/web-eid/#schedule`.

[7] ATR: 3B FE 18 00 00 80 31 FE 45 80 31 80 66 40 90 A5 10 2E 10 83 01 90 00 F2

[8] The card's ATR also points to another card model: Infineon CJTOP 80K INF SLJ 52GLA080AL M8.4.

[9] ATR: 3B D5 18 FF 81 91 FE 1F C3 80 73 C8 21 10 0A

3.2.1 JavaCard Programming

In this subsection, I introduce the JavaCards in general, as well as describe my development environment and how to install an applet on the card.

Introduction to JavaCards JavaCard is a smart card capable of running the *Java Card Virtual Machine*. It executes bytecode, manages classes and objects, and enables secure data sharing between different applications [53]. According to its name, it supports code written in Java with specific *javacard* packages to provide different smart card-specific functions (e.g., handling APDU messages).

However, JavaCard VM is not capable of running vanilla Java code, and there are few specifications or differences:

- *threads*: thread programming is not supported due to smart cards' low-performance CPUs.
- *Garbage Collector:* JavaCard VM does not support garbage collection. Therefore methods like `finalize()` are not supported. The developers need to do their own memory management to prevent writing dynamic values into static memory, ultimately leading to locking the card by having no memory left for other operations.
- *Primitive Types:* JavaCard VM only supports *byte*, *short* and *boolean* primitive types. The *Integer* type is supported only on specific platforms and should not be relied on.

More detailed and technical details about JavaCard programming can be found in [53]. General guidelines and best practices (including memory management mentioned before) are listed in [54].

Development Requirements and Applet Installation In order to create a JavaCard applet (i.e., an application package capable of running on JavaCard) and load it onto a smart card, I had to download a few files and repositories:

- *JavaCard SDK binaries* are necessary to recognize all JavaCard-related keywords in the code and build the applet itself. I have used [55], which provides a list of SDK directories for different versions of JavaCard (version 3.0.4 in my case; therefore, I have used *jc304_kit*).
- *ant-javacard.jar* file from [56] is used to build a CAP file (which can be loaded on JavaCard) from the JavaCard applet source code.
- *gp.jar*: lastly, the file from [57] is used for operations regarding the JavaCard itself, such as reading information from the card, uploading, installing, and deleting CAP files as applets.

First, I had to create a `build.xml` (see Listing 3.1) file with specific information for `ant` command[10]. In `<taskdef>` tags, I have specified which library to use for building (*ant-javacard.jar*). After that, I specified which SDK binary to use, how the CAP file should be named, and the applet's source code. Specified AID values define the unique application ID on the card.

Listing 3.1: build.xml file for building CAP file

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir=".">
    <taskdef name="javacard" classname="pro.javacard.ant.
       JavaCard" classpath="ant-javacard.jar" />
    <javacard jckit="sdks/jc304_kit">
        <cap output="InfinitEID-applet.cap" sources="src/
           InfinitEID" aid="0102030405">
            <applet class="InfinitEID.InfinitEIDApplet"
               aid="0102030405060708" />
        </cap>
    </javacard>
</project>
```

After that, I run these commands in sequence in order to fully build and install my applet on the JavaCard (the result can be seen in Figure 4.2):

1. build the CAP file: `ant -f build.xml`.
2. Uninstall the existing applet from the card, if it exists:
   `java -jar ./gp.jar --uninstall ./InfinitEID-applet.cap`.
3. Install the applet to the card:
   `java -jar ./gp.jar --install ./InfinitEID-applet.cap`.

3.2.2 InfinitEID: JavaCard Applet

As mentioned before, I have decided to create a new, open-source JavaCard applet, which is fully compatible with custom Web-eID solution (more details in section 3.3) and according to the modern electronic ID standards (e.g., ISO 7816 and [58]). During the development, I was inspired primarily by *FakeEstEID* [59] and *IsoApplet* [60])[11]. I have also named a created applet the *InfinitEID*.

The following list summarizes InfinitEID's functionality[12]:

[10] Apache Ant is commonly used for building Java applications.

[11] All inspirations are listed in the source code.

[12] For increased readability, the subject in the following items is abstract. It is either the applet itself (in the form of a source code), the installed application on the card or the card itself (especially when talking about storing values in the memory).

- can handle APDU messages longer than 256B by using:
  - logical chaining of APDU messages in T0 communication protocol or
  - extended APDU protocol[13] in the T1 communication protocol.
- Contains two key-pair objects (containing keys for Elliptic-curve operations over large prime fields) and handles their generation during the card initialization phase (more on that in the following subsection 3.2.2):
  - for authentication and
  - for digital signature[14].
- Contains two X509 certificates over the public keys from key-pair objects, which were generated and signed by a trusted authority during the card initialization phase.
- Supports verifying, handling, changing, and resetting three PIN values: `auth`, `sign`, and `admin`. Their relationships are the following:
  - `auth` PIN is for on-card verification of the authentication operation,
  - `sing` PIN is for on-card verification of the digital signature operation,
  - `admin` PIN is for on-card verification of (re)setting other two PIN values, (re)generating key-pairs, uploading certificates,
  - all PIN values have minimum and maximum length specified,
  - all PIN values have a maximum retries counter, after which the PIN is blocked and cannot be verified anymore (until resetting with the `admin` PIN).
- Supports reading and writing an arbitrary amount of binary data from/to the card. It is currently used only for certificates, but it can be easily expanded when more data is needed to be stored on the card.
- Currently, it uses ES384[15] algorithm for creating signature values. Elliptic curve domain parameters were taken from [61].
- After verifying appropriate PIN value, it allows running `authenticate()` or `performSignature()`. Both operations perform a digital signature algorithm but over different pre-hashed data. The authentication value from Web-eID (built from challenge nonce and server origin) is signed during authentication. During the regular signature operation, an arbitrary amount of data is signed.

InfinitEID also supports some common *Status Word* responses (described in subsection 1.4.2) from [62] and [63]. For more details or while debugging, check the

[13]Built-in protocol in the SDK to automatically handle longer messages. It is not supported by every card chip.

[14]Please note, that although during authentication the card also produces digital signature, the common practice is to have different key pairs for different card operations.

[15]JSON Web Algorithm equivalent of ECDSA algorithm using elliptic curve P-384 (also called "secp384r1") and hash function SHA-384.

InfinitEID source code itself.

Management Console for InfinitEID To safely initialize and manage the card after applet installation (by following instructions from 3.2.1), I have also created an InfinitEID management console (see Figure 4.3).

First, the system administrator needs to create his own root private key and root certificate, which will be used to sign public keys generated on the card by running commands specified and described in the source code. Second, he needs to adjust values in the configuration file `config.yaml`. He can also review or change APDU requests used by the console in `apdulist.yaml`.

The administrator can then run the main Python script, after which he is presented with several options:

- *Initialize currently connected card*: sets the PIN values from the configuration file, requests key-pairs on-card generation, obtains public keys from the card, creates certificate using root private key, and stores these certificates on the card.
- *Select main applet*: this command sends the applet selection request to the card. It is useful for debugging or troubleshooting.
- *Obtain public key from card*: prints the out selected public key in PEM format.
- *Obtain certificate from card*: prints the out selected certificate in PEM format.
- *Set PIN*: asks for a new value of a selected PIN and tries to (re)set it on the card. It is primarily used by the administrator to unblock a locked card.
- *Verify PIN*: asks for a value of a selected PIN and tries to verify it on the card. It is primarily used to authorize other operations and by the administrator to verify his `admin` PIN to set other PINs.
- *Run specific command*: allows to send chosen command from `apdulist.yaml` file. Useful for debugging and troubleshooting.
- *Toggle APDU logging (current: False)*: defines whether or not the specific bytes of APDU messages are shown in the console.

Lastly, as an addition, I have created a second *Python script containing 4 unit tests*. Together, they are supposed to *determine the compatibility of the currently connected card with the custom Web-eID solution*. Results can be seen in Figure 4.5.

3.3 Web-eID Extensions and Modifications

In this section, I describe my changes and additions to the Web-eID solution in order for it to accept InfinitEID and ultimately allow the user to log in to Nextcloud using a custom smart card. The actual research and implementation of these changes have been consulted with Arnis Paršovs (Research Fellow in Cyber Security at the

University of Tartu, Estonia) and Mart Sömermaa (Web-eID developer and maintainer) [35].

3.3.1 Web-eID Validation Library in PHP

As mentioned in subsection 2.4.1, validation libraries are used for securely creating cryptographic challenges and verifying the response and the certificates associated with it. Currently, only two of these libraries are available in the Web-eID repository: Java (finished) and .NET (being developed).

Since the Nextcloud back-end is written primarily in PHP, I have decided to develop and deploy a new PHP validation library from scratch. This implementation is open-source, provides the same functionalities as existing libraries, and can be easily deployed in modern PHP applications. It required studying and researching ASN1 notation, X.509 certificate architecture [64] and OCSP protocol [65]. Together with the original Web-eID JavaScript library, the solution provides on-demand nonce generation and authentication token validation according to the Web-eID standard.

It can be installed and deployed with Composer in any PHP project. Great emphasis was placed on mimicking as much existing Java code-base as possible, although there were some obstacles due to programming language differences. *It should also help the Web-eID community since Mart Sömermaa expressed great interest in its future evaluation and possible integration into the official Web-eID repository* [35].

OCSP Client Library During the PHP validation library development, I also had to create an OCSP (Online Certificate Status Protocol) client in PHP. It allows building an OCSP request and sending it to an OCSP server to obtain a validity check of a certificate (whether or not it has been revoked). In other programming frameworks (Java and .NET), it is already a part of built-in or installable packages. However, it is missing in PHP. Currently, this client is part of said PHP validation library, but I plan to separate it into a standalone PHP library in the future for the development community to use in their own web applications. Mart Sõmermaa well supports this idea.

3.3.2 Web-eID Native Application Extension

For Web-eID to support InfinitEID, I had to also create a native application extension by implementing the pre-defined interface *ElectronicID* in C++ programming language. In this interface, I had to define the following variables and functions:

- APDU message to select application AID,
- APDU messages for selecting authentication or digital signature certificates,
- function for verifying PIN values,

- function for obtaining the number of PIN retries left,
- supported digital signature algorithms (only ES384 at the moment), and
- functions for authenticating and digital signature.

In general, the Web-eID native application uses these definitions to know which sequence of bytes to send to the card to obtain high-level results of specific operations.

3.3.3 Authentication Token Validation in Nextcloud

As mentioned in subsection 3.1.2, I have created an installable Nextcloud App to integrate Web-eID authentication as a second authentication factor. I will further describe how to install the PHP validation library into this application and configure necessary objects (responsible for creating challenge nonce and authentication token validation) to enable users to log in with their initialized InfinitEID.

Installation First, the system administrator must install this application into Nextcloud by copying/cloning the source code into the `nextcloud-path/apps` directory. Web-eID validation token library for PHP is already defined in the Composer installation file, so the administrator needs to issue the command `composer install` to download and install it automatically.

Challenge Nonce Store Configuration After the installation, the administrator also needs to configure a challenge nonce store to guarantee that the authentication token is received from the same browser to which the corresponding challenge nonce was issued. The most probable solution is a session-dependent challenge nonce store.

In Nextcloud, a session variable is available to all classes in the framework. The Listing A.1 provides an example implementation using the *ISession* object from the Nextcloud framework. Note that the class extends a *ChallengeNonceStore*, part of the validation token library.

Challenge Nonce Generator Configuration Challenge nonce generator can be implemented by extending the *ChallengeNonceGenerator* interface. However, a *ChallengeNonceGeneratorBuilder* is available in the library, which can be used to create a default, built-in implementation with specific values. In the Listing A.2, an example usage is provided.

Authentication Token Validator Configuration Lastly, the server administrator needs to configure the validator itself. Mandatory parameters are the website origin and trusted certificate authorities (i.e., loaded PEM or DER-encoded files using *CertificateLoader* class from the library). The administrator can specify a designated OCSP server location or turn off OCSP checking completely. In the Listing A.3, an example configuration is provided.

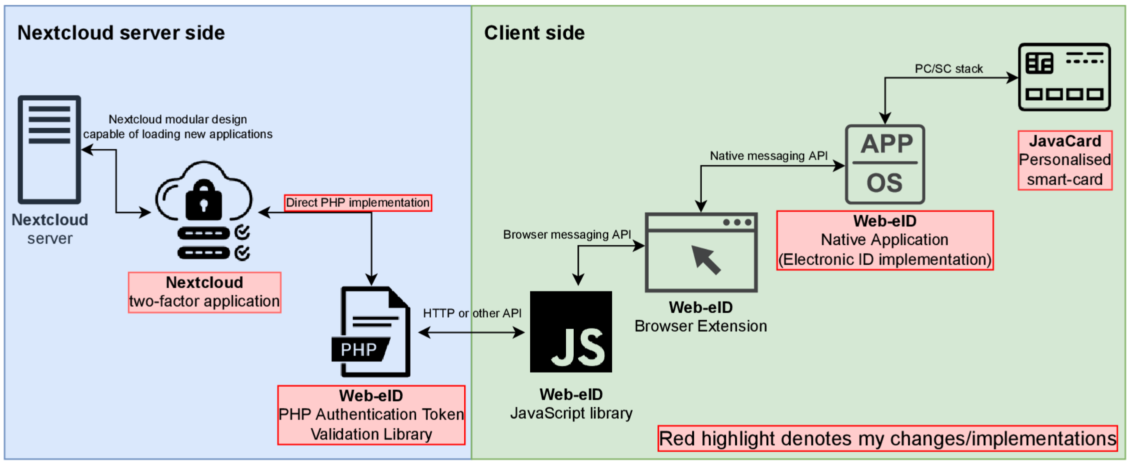
Obtaining Authentication Result During the authentication attempt, the server administrator needs to ensure that:

- challenge nonce is generated, stored, strictly tied with user session in the backend, and is sent to the client:
  - by including it in the rendered template on the authentication page (example in the Listing A.4) or
  - by providing it via REST API point, which the client accesses during authentication initialization.
- The client page tries to provide an authentication token by utilizing the Web-eID JavaScript library, ultimately invoking communication with the smart card. An example is provided in the Listing A.5: the authentication token returned from the card is inserted into the Nextcloud second-factor authentication form, which is afterward submitted.

Lastly, the administrator should also check the function `lib/Service/WebEidService.php:authenticate()` and optionally implement a different authentication mechanism after the Web-eID successfully verifies electronic identity by its own workflow. Currently, the program checks whether the verified certificate from the card contains the same username in its subject field CommonName as provided by the Nextcloud framework (see Listing A.6).

4 Overview and Verification of Results

In this chapter, I present the results of this thesis in sequences of screenshots from the administrator's and user's points of view. I also present measurement results for on-card digital signature operation and duration of token validation by the PHP validation library. Figure 4.1 presents a general overview of this thesis' solution, with my contribution highlighted in red color.

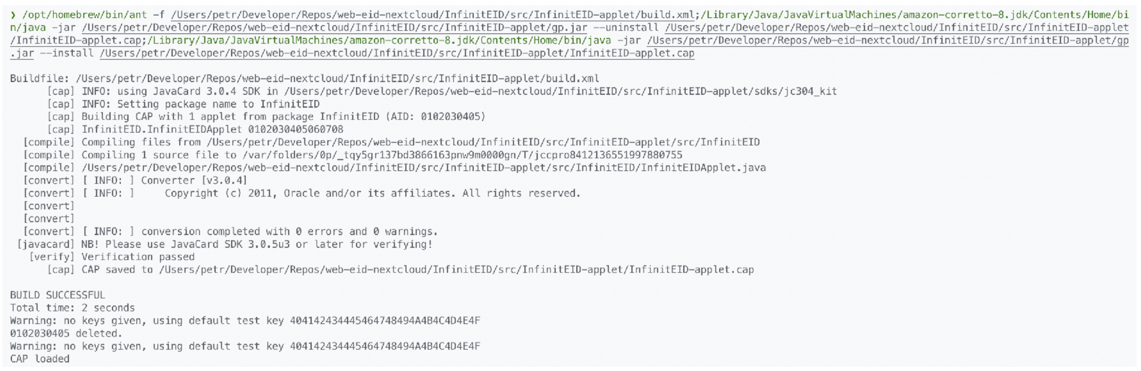


Nextcloud server side
Nextcloud modular design capable of loading new applications
Nextcloud server
Direct PHP implementation
Nextcloud two-factor application
PHP
Web-eID PHP Authentication Token Validation Library
Client side
PC/SC stack
JavaCard Personalised smart-card
Native messaging API
APP
OS
Web-eID Native Application (Electronic ID implementation)
Browser messaging API
Web-eID Browser Extension
HTTP or other API
JS
Web-eID JavaScript library
Red highlight denotes my changes/implementations

Fig. 4.1: General overview of Nextcloud–Web-eid architecture with highlighted contributions

4.1 Administrator's Point of View

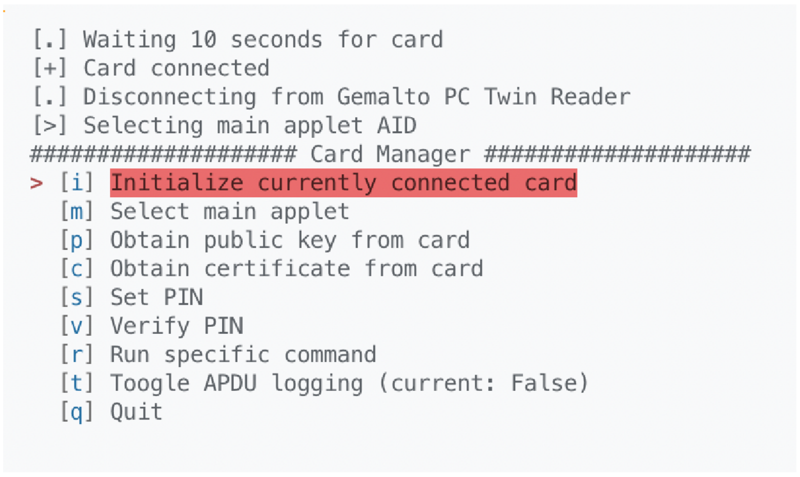
Assuming trusted certificate authorities have already been established and certificates issued (e.g., a self-signed certificate has been created), the system administrator can start personalizing blank JavaCards by building the latest InfinitEID applet and installing it on the card (see Figure 4.2).



```
❯ /opt/homebrew/bin/ant -f /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/build.xml;/Library/Java/JavaVirtualMachines/amazon-corretto-8.jdk/Contents/Home/bi
n/java -jar /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/gp.jar --uninstall /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet
/InfinitEID-applet.cap;/Library/Java/JavaVirtualMachines/amazon-corretto-8.jdk/Contents/Home/bin/java -jar /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/gp
.jar --install /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/InfinitEID-applet.cap

Buildfile: /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/build.xml
      [cap] INFO: using JavaCard 3.0.4 SDK in /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/sdks/jc304_kit
      [cap] INFO: Setting package name to InfinitEID
      [cap] Building CAP with 1 applet from package InfinitEID (AID: 0102030405)
      [cap] InfinitEID.InfinitEIDApplet 0102030405060708
  [compile] Compiling files from /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/src/InfinitEID
  [compile] Compiling 1 source file to /var/folders/0p/_tqy5gr137bd3866163pnw9m0000gn/T/jccpro8412136551997880755
  [compile] /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/src/InfinitEID/InfinitEIDApplet.java
  [convert] [ INFO: ] Converter [v3.0.4]
  [convert] [ INFO: ]     Copyright (c) 2011, Oracle and/or its affiliates. All rights reserved.
  [convert]
  [convert]
  [convert] [ INFO: ] conversion completed with 0 errors and 0 warnings.
 [javacard] NB! Please use JavaCard SDK 3.0.5u3 or later for verifying!
   [verify] Verification passed
      [cap] CAP saved to /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID/src/InfinitEID-applet/InfinitEID-applet.cap

BUILD SUCCESSFUL
Total time: 2 seconds
Warning: no keys given, using default test key 404142434445464748494A4B4C4D4E4F
0102030405 deleted.
Warning: no keys given, using default test key 404142434445464748494A4B4C4D4E4F
CAP loaded
```

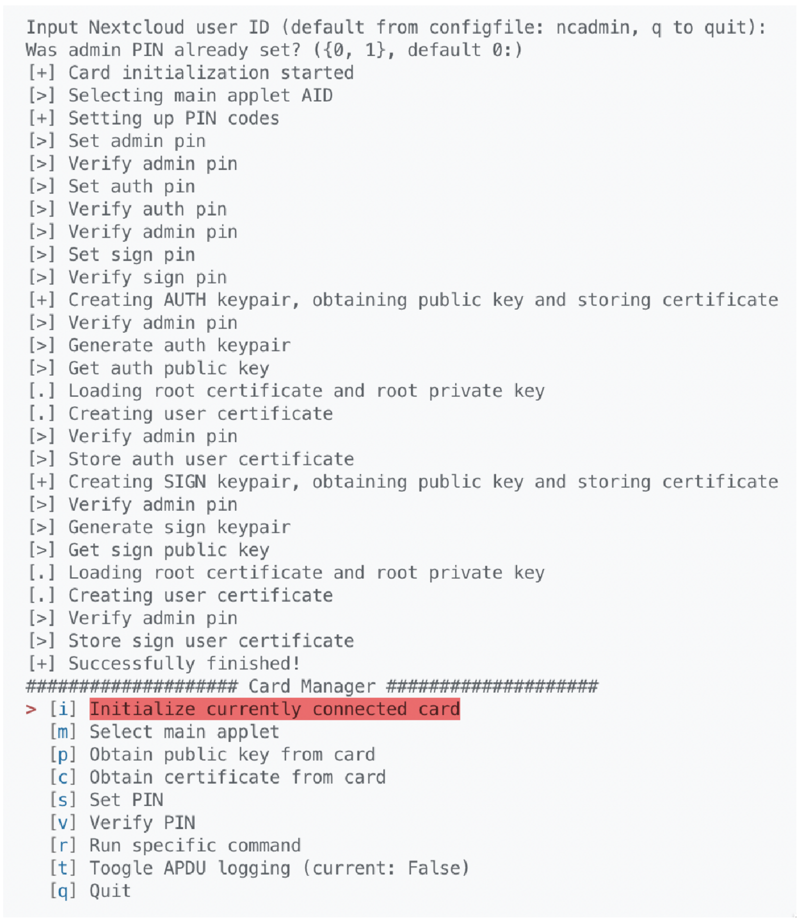
Fig. 4.2: InifinitEID applet build and reload process

He then runs the InfinitEID management console (see Figure 4.3), selects the first *Initialize currently connected card* option, and inserts the Nextcloud username, for which he wishes to activate the Web-eID second-factor authentication (see initialization for user *ncadmin* in Figure 4.4).



```
[.] Waiting 10 seconds for card
[+] Card connected
[.] Disconnecting from Gemalto PC Twin Reader
[>] Selecting main applet AID
#################### Card Manager ####################
> [i] Initialize currently connected card
  [m] Select main applet
  [p] Obtain public key from card
  [c] Obtain certificate from card
  [s] Set PIN
  [v] Verify PIN
  [r] Run specific command
  [t] Toogle APDU logging (current: False)
  [q] Quit
```

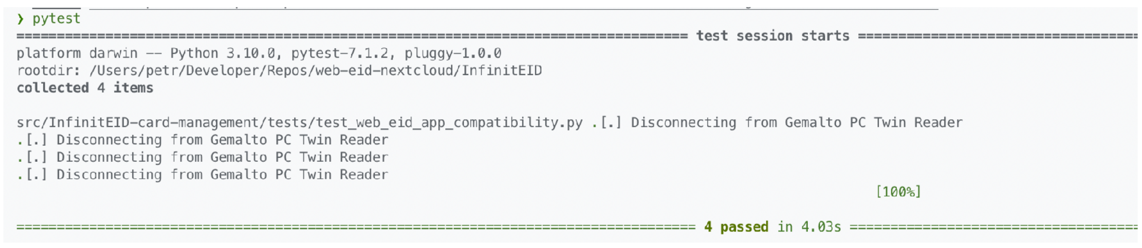
Fig. 4.3: InfinitEID management console



```
Input Nextcloud user ID (default from configfile: ncadmin, q to quit):
Was admin PIN already set? ({0, 1}, default 0:)
[+] Card initialization started
[>] Selecting main applet AID
[+] Setting up PIN codes
[>] Set admin pin
[>] Verify admin pin
[>] Set auth pin
[>] Verify auth pin
[>] Verify admin pin
[>] Set sign pin
[>] Verify sign pin
[+] Creating AUTH keypair, obtaining public key and storing certificate
[>] Verify admin pin
[>] Generate auth keypair
[>] Get auth public key
[.] Loading root certificate and root private key
[.] Creating user certificate
[>] Verify admin pin
[>] Store auth user certificate
[+] Creating SIGN keypair, obtaining public key and storing certificate
[>] Verify admin pin
[>] Generate sign keypair
[>] Get sign public key
[.] Loading root certificate and root private key
[.] Creating user certificate
[>] Verify admin pin
[>] Store sign user certificate
[+] Successfully finished!
#################### Card Manager ####################
> [i] Initialize currently connected card
  [m] Select main applet
  [p] Obtain public key from card
  [c] Obtain certificate from card
  [s] Set PIN
  [v] Verify PIN
  [r] Run specific command
  [t] Toogle APDU logging (current: False)
  [q] Quit
```

Fig. 4.4: InfinitEID initialization finished

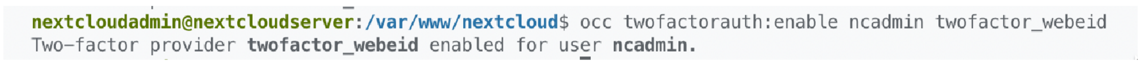
Before activating the second factor for the specific user, the administrator optionally runs unit tests that are part of a management console to test Web-eID compatibility (whether the card responds correctly to specific requests). See Figure 4.5 for successful unit tests run.



```
❯ pytest
================================================================================ test session starts ====================================
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/petr/Developer/Repos/web-eid-nextcloud/InfinitEID
collected 4 items

src/InfinitEID-card-management/tests/test_web_eid_app_compatibility.py .[.] Disconnecting from Gemalto PC Twin Reader
.[.] Disconnecting from Gemalto PC Twin Reader
.[.] Disconnecting from Gemalto PC Twin Reader
.[.] Disconnecting from Gemalto PC Twin Reader
                                                                                                                     [100%]

================================================================================ 4 passed in 4.03s ====================================
```

Fig. 4.5: Web-eID compatibility unit tests result

Lastly, the administrator uses *the OCC* console (built-in management console for Nextcloud) to activate the Web-eID second factor for the specific user.



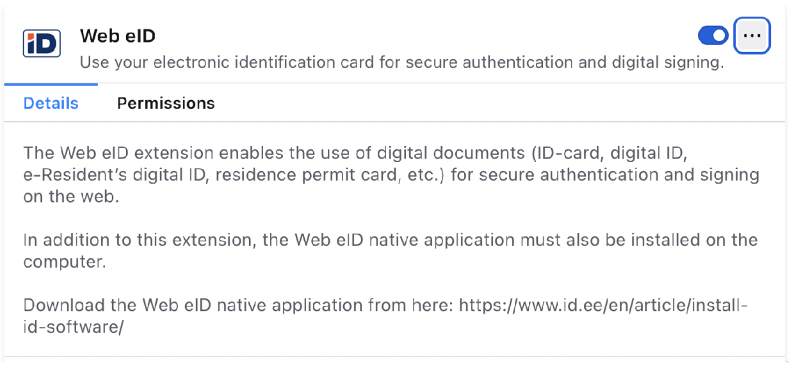
```
nextcloudadmin@nextcloudserver:/var/www/nextcloud$ occ twofactorauth:enable ncadmin twofactor_webeid
Two-factor provider twofactor_webeid enabled for user ncadmin.
```

Fig. 4.6: Enabling Web-eID second-factor authentication for ncadmin user

4.2 User's Point of View

In this subsection, an opposite point of view is described. The following text and pictures depict the final login workflow.

First, the user must install the official version of Web-eID specifically for his operating system and web browser of choice (see Figure 4.7). In order for his instance to accept InfinitEID, he needs to build his own, containing the extension described in subsection 3.3.2)[1] and replace it with the original one (see Figure 4.8).



Web eID

Use your electronic identification card for secure authentication and digital signing.

Details Permissions

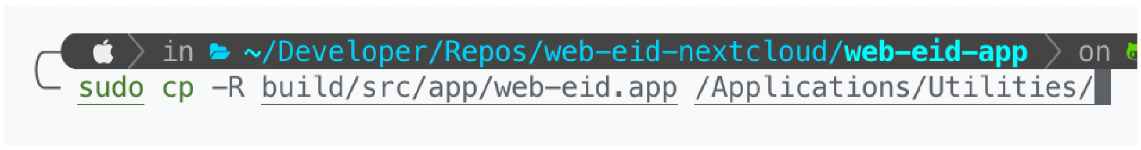
The Web eID extension enables the use of digital documents (ID-card, digital ID, e-Resident's digital ID, residence permit card, etc.) for secure authentication and signing on the web.

In addition to this extension, the Web eID native application must also be installed on the computer.

Download the Web eID native application from here: https://www.id.ee/en/article/install-id-software/

Fig. 4.7: Web-eID web-browser extension installed

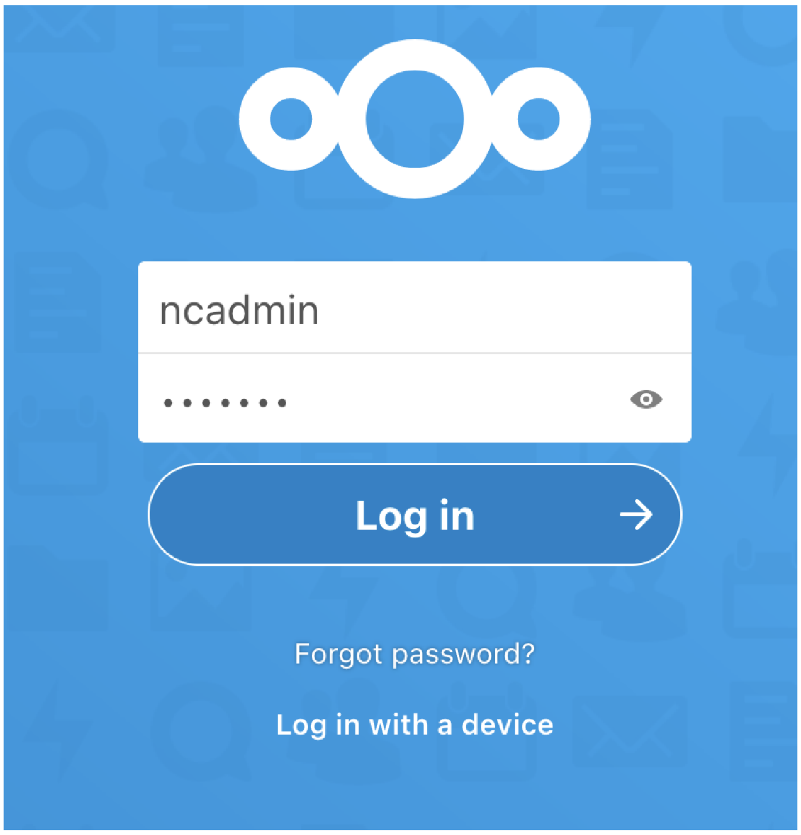
[1]Providing already extended and builded version directly to the users is in the scope for the future.



```
 in ~/Developer/Repos/web-eid-nextcloud/web-eid-app  on
sudo cp -R build/src/app/web-eid.app /Applications/Utilities/
```

Fig. 4.8: Replacing extended and built version of Web-eID native application with the original one

Afterward, the user attempts to log in to Nextcloud (see Figure 4.9) and is presented with a page dedicated to a Web-eID smart card authentication (see Figure 4.10). After clicking on the *Authenticate* button, a regular Web-eID authentication workflow is executed. Thus the user is immediately asked for an authentication PIN (see Figure 4.11).



ncadmin

•••••••

Fig. 4.9: Nextcloud login page

Modified Web-eID native application exchanges data with the user's InfinitEID card and returns an authentication token, which the PHP validation library validates. When successful, the user is authenticated and granted access to the cloud storage (see Figure 4.12).

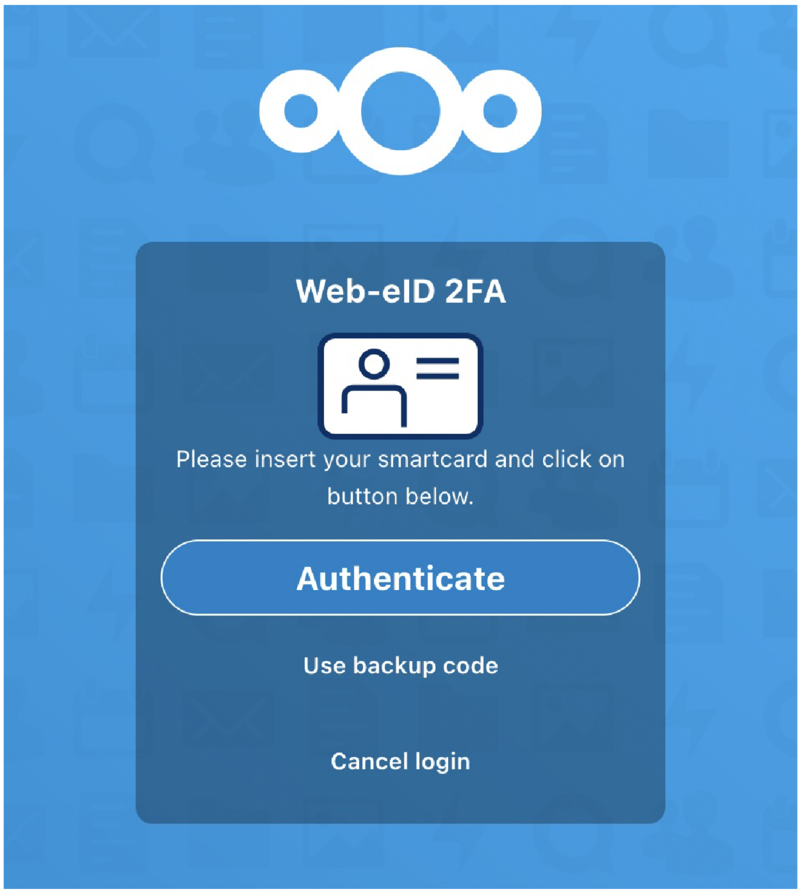
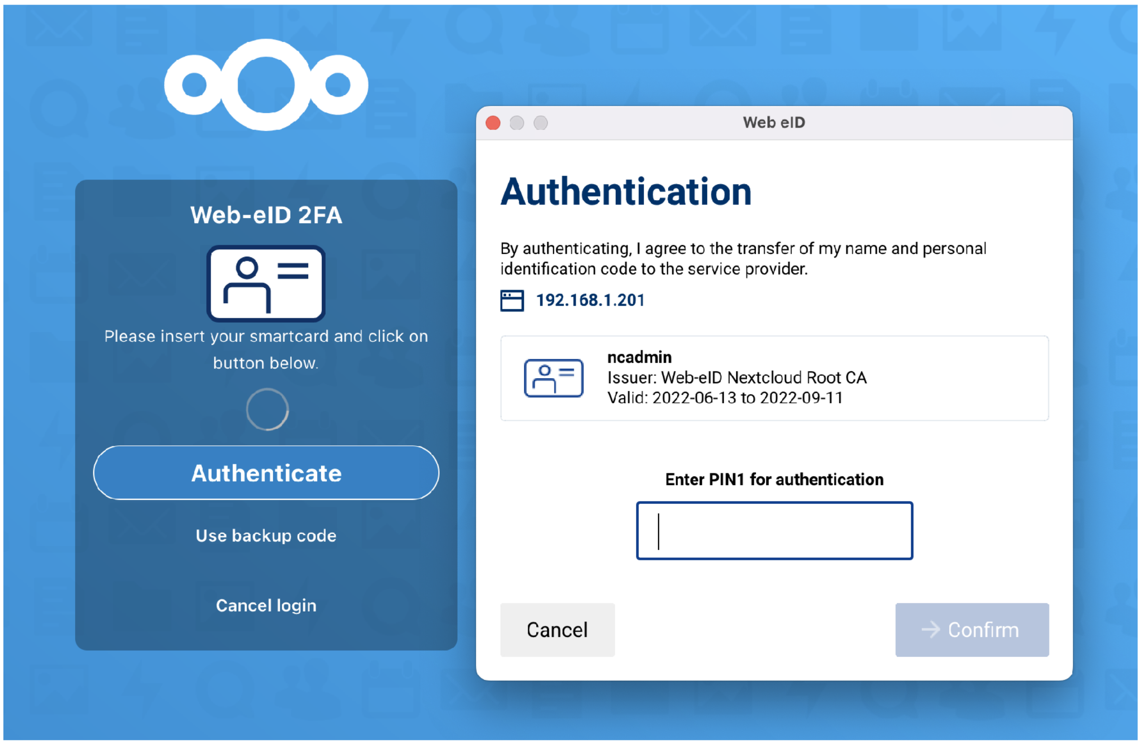


Fig. 4.10: Web-eID second-factor authentication page



Authentication

By authenticating, I agree to the transfer of my name and personal identification code to the service provider.

192.168.1.201

ncadmin
Issuer: Web-eID Nextcloud Root CA
Valid: 2022-06-13 to 2022-09-11

Enter PIN1 for authentication

Fig. 4.11: Web-eID authentication PIN prompt

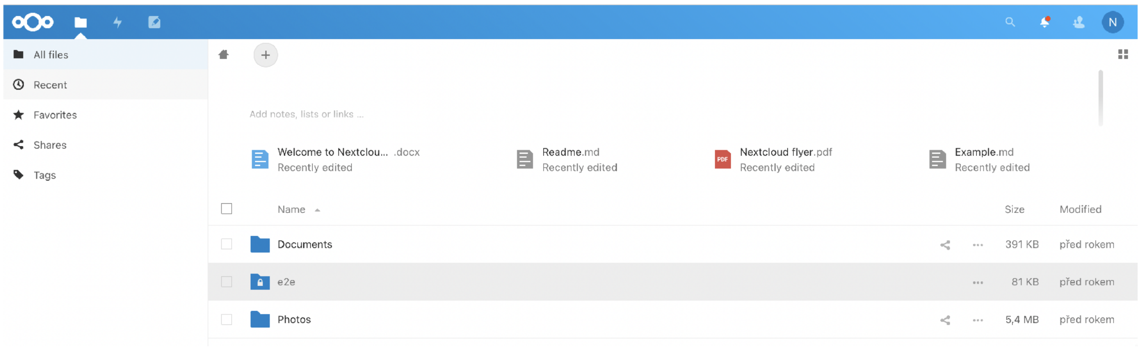


Fig. 4.12: Nextcloud web-interface example

4.3 Authentication Duration Measurements

In this subsection, I provide two measurements: *the on-card digital signature of pre-hashed authentication value*[2] and *PHP back-end validation of Web-eID authentication token*.

During the first measurement, I created a Python script that generates a simulated Web-eID authentication value, sends it to the card *one hundred times*, and measures how long it took to obtain a response. As mentioned in subsection 3.2.1, I was working with two different JavaCards. I have run this script *three times for each card*, so in total, I have done *six measurements with 100 retries each*. The results are depicted in Figure 4.13, and the average values are in Table 4.1. From these, we can conclude that the speed of creating a digital signature of authentication value heavily depends on the exact card model. However, the slower model is still usable and user-friendly. Please note that APDU communication and on-card request processing are also part of a measured duration. Compared to the values from Petr Švenda's *JavaCard Algorithm Test* [66], my measurements are 16–30 % higher. The mentioned actions might cause this delay.

Tab. 4.1: Average duration values for on-card digital signature

Card model:	***JavaCard SLJ52GCA150 (jTOP SLE78 Estonian ID card platform)***	***JavaCard NXP JCOP3 J3H145***
Average from measurement 1 [s]:	0,262262263	0,069637427
Average from measurement 2 [s]:	0,256840816	0,073954861
Average from measurement 3 [s]:	0,264224105	0,074868116

[2]Using already mentioned ES384.

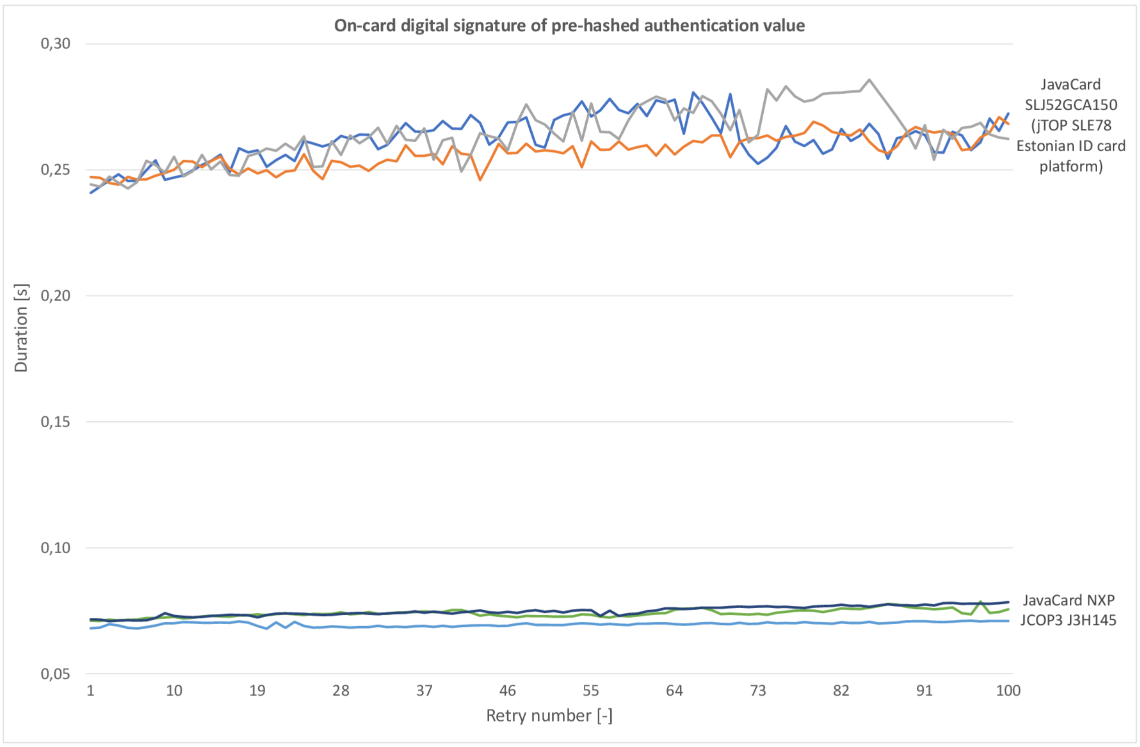


Fig. 4.13: On-card digital signature of pre-hashed authentication value measurement

For the second measurement, I have modified the Nextcloud App for Web-eID second-factor authentication. Instead of providing authentication results to the Nextcloud framework, it also runs the validation process *one hundred times* and measures the duration of this process. I then ran this code *five times*. Results of validation without OCSP service (which might introduce random latency) are depicted in Figure 4.14 and Table 4.2. The first values of each run are slightly higher and are more critical (since the user will probably always try to log in only once a time), so I have excluded them from the average calculation. We can conclude that the pure validation process is fast enough for the user not to notice any significant delay in the authentication process.

Tab. 4.2: Average durations of back-end validation

	First validation [s]	**Average of the 2nd–100th retry [s]**
Measurement 1:	0,152329	0,144614
Measurement 2:	0,153286	0,144802
Measurement 3:	0,153549	0,144638
Measurement 4:	0,153389	0,144777
Measurement 5:	0.153431	0,145102

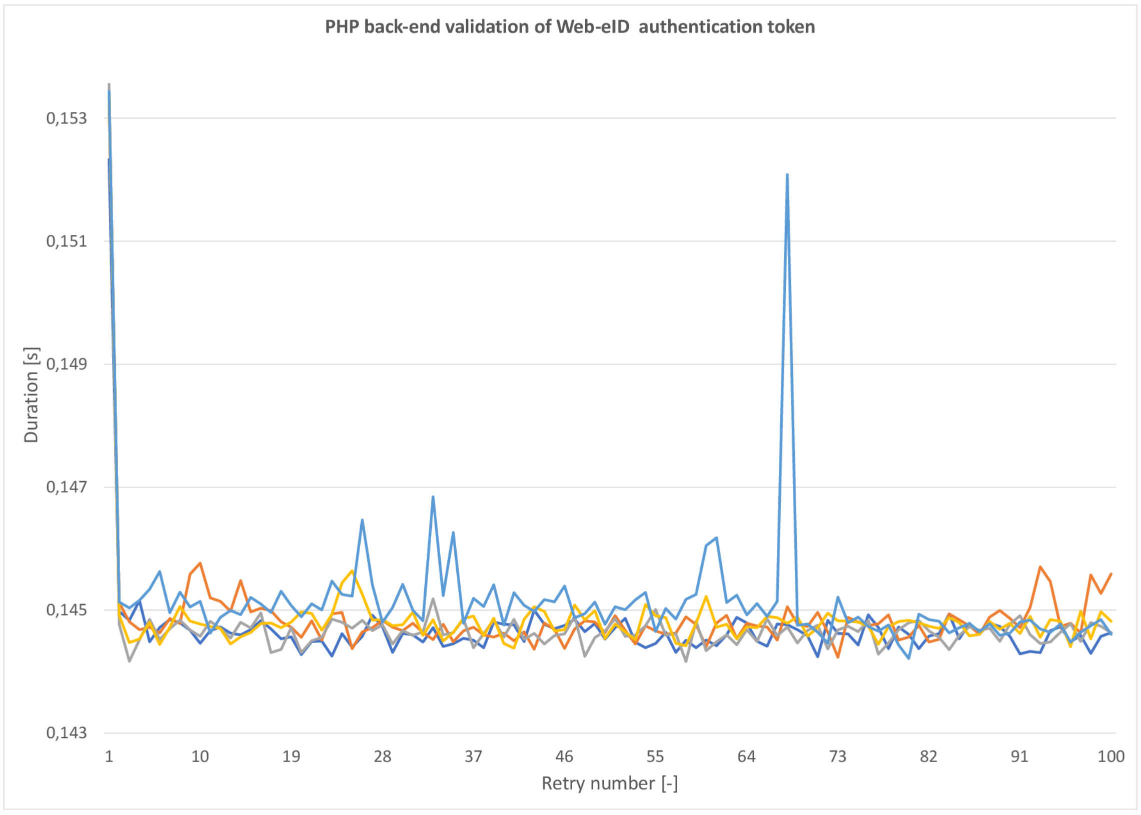


Fig. 4.14: PHP back-end validation of Web-eID authentication token measurement

5 Next Steps: Czech eID

One of Web-eID's main goals is an expansion to other European Union countries. Based on the communication with both Estonian consultants, preparing Web-eID for the Czech ID card would significantly contribute to the project overall. Out of scope of this thesis, I have decided to research and try to implement a new *ElectronicID* interface for the Czech electronic identity card. These cards are being distributed to all fifteen years old citizens or whoever inquires and contain a chip with a JavaCard applet that should serve a similar purpose as Estonian eID.

As soon as I started, I encountered several critical problems. First, there is no official documentation for the Czech eID applet (apparently by choice)[1]. I have tried contacting the official support of *Identita Občana* (the official portal for Czech electronic identity) and asked for an essential public API structure (which APDU commands to send to obtain specific responses). I have received a one-sentence response saying that the Ministry of Interior is not interested in integrating my application. However, a *leader of the development group Smart Security from MONET+[2], a.s.* reached out to me later, saying that although the Ministry of Interior indeed can not provide me with Czech eID documentation, they are interested in the results of this thesis and want to co-operate in the future.

My next option was to use real Czech eID with genuine and legitimate software and trace APDU using standards tools (for example, `pcscd` tool with "-a" flag or running specific command[3] in the macOS terminal and reading APDU messages from the console). For specific operations, I could read which specific bytes were sent to the card before that operation and define them in the Web-eID *ElectronicID* interface. However, from conversations with said MONET+ employee, I have learned that in the case of Czech eID complementary software, *it is forbidden to perform any decompilation, transformation, reverse-engineering, or other similar operations except for the situation described or listed explicitly by law.* Chances are, the same restrictions also apply to the Czech eID JavaCard applet.

According to the description and several articles referenced in [67], the Czech Ministry of Interior has decided *not to make public on-card certificates public*[4], ultimately denying broad public community in developing new applications utilizing new electronic IDs and independent identity verification. ParalelniPolis also provides a way to compute these public keys from card certificates. This way of obtaining

[1]In opposition with other state-issued ID cards. Estonian ID card is openly specified in `https://www.id.ee/en/article/id-card-documentation-2/` and `https://www.id.ee/wp-content/uploads/2021/08/td-id1-chip-app-4.pdf`.

[2]Company, which is developing electronic identity cards for Czech government.

[3]sudo defaults write /Library/Preferences/com.apple.security.smartcard Logging -bool yes

[4]It refuses to publish public keys required to verify these on-card certificates, to be more specific.

public keys required to verify the on-card public key (with the corresponding private key used for digital signature) is in the scope for future work from this thesis (assuming all required steps will be proven legal).

Conclusions

During the work on this thesis, I have focused on cloud services, cloud storages, authentication, and smart cards in general. I have also researched existing technologies that connect smart cards and web services by enabling web servers to utilize personal smart cards for authenticating users into their services.

The state-of-the-art shows that most solutions are either not supported anymore, not documented enough, or do not meet our criteria for secure authentication via smart cards into cloud storage for handling electronic evidence. As a proof-of-concept, I created a simple solution using MultOS smart card and socket programming during the first half of the thesis work.

I have also discovered an up-and-coming Estonian solution called Web-eID – currently being developed collection of tools that safely authenticate a user to the web application using a smart card. It is set to be used in all Estonian e-services in the next few years (even spreading to other European countries as a new standard) and is primarily focused on state-issued electronic ID cards. However, due to its open-source license, high extensibility, and my Erasmus+ stay in Estonia (enabling close contacts and consultations with two Estonian experts on Web-eID and smart cards in general), I have managed to use it with the custom-built and personalized smart card.

I have created an InfinitEID JavaCard applet, which is fully compatible with a modified and extended version of Web-eID. It follows the same authentication and digital signature principles and workflows as the original Estonian ID card (ensuring the solution is a modern and possibly safe option for this thesis outcome).

Then, I created a new Web-eID authentication token validation library in PHP language, which might be part of the official repository someday (since existing Web-eID validation libraries are currently only for Java and .NET applications with more languages to come).

The subsequent outcome is an installable Nextcloud (one of the most popular open-source cloud storage) module that enables Web-eID for a second authentication factor by utilizing this PHP validation library. In the end, I have also measured the duration of authentication with InfinitEID and the duration of back-end token validation.

I have also researched (noted by the two mentioned experts as "highly requested" for Web-eID expansion) the option of using Czech electronic identity cards with Web-eID. This area, however, still needs some time and possible involvement in closed-source development as the choice of the Czech Ministry of Interior (as the opposite of the Slovak or Estonian government) is not to publicize public keys used for on-card certificates verification. This decision prevents the broad community from using Czech state-issued electronic identities in their applications.

Overall, this thesis work and its practical outcomes – besides providing the solution to authenticate users into cloud storage with electronic evidence safely – is supposed to guide/help future developers with effective and secure integration of strong-authentication methods into their web applications on every expertise level.

Bibliography

[1] RedHat. What are cloud services?, December 2019. URL: `https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services`.

[2] Amazon Web Services. What is cloud computing, 2021. URL: `https://aws.amazon.com/what-is-cloud-computing/`.

[3] L. Wang, G. von Laszewski, A. Younge, Xi He, M. Kunze, J. Tao, and Ch. Fu. Cloud computing: a perspective study - new generation computing, 2010. URL: `https://doi.org/10.1007/s00354-008-0081-5`.

[4] Nick Antonopoulos and Lee Gillam, 2017. URL: `https://doi.org/10.1007/978-3-319-54645-2`.

[5] Erik Chovanec. Řízení přístupu k datům v cloudu. Master's thesis, Brno University of Technology, 2021.

[6] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, and Jianqing Fu. Cloud storage as the infrastructure of cloud computing. In *Cloud Storage as the Infrastructure of Cloud Computing*, pages 380–383. IEEE, June 2010.

[7] Tým VUT v Brně. Identifikace požadavků a návrh technického řešení systému pro bezpečnou správu el. důkazů. Technical report, Brno University of Technology, Department of Telecommunications, 2022.

[8] Government - nextcloud. URL: `https://nextcloud.com/industries/government/`.

[9] Enterprise - nextcloud. URL: `https://nextcloud.com/enterprise/`.

[10] owncloud vs nextcloud, October 2020. URL: `https://owncloud.com/owncloud-vs-nextcloud/`.

[11] seafile. Security features - seafile admin manual. URL: `https://manual.seafile.com/security/security_features/`.

[12] Inc. Free Software Foundation. Gnu affero general public license - gnu project - free software foundation (fsf), November 2007. URL: `https://www.gnu.org/licenses/agpl-3.0-standalone.html`.

[13] Dawn M. Turner. Digital authentication - the basics, August 2016. URL: `https://www.cryptomathic.com/news-events/blog/digital-authentication-the-basics`.

[14] Dave Piscitello. What is authorization and access control?, December 2015. URL: https://www.icann.org/en/blogs/details/what-is-authorization-and-access-control-2-12-2015-en.

[15] Faouzi Jaidi and Jaydip Sen. *Advances in Security in Computing and Communications*. InTech, July 2017.

[16] R.S. Sandhu and P. Samarati. Access control: principle and practice. *IEEE Communications Magazine*, 32(9):40–48, September 1994.

[17] MULTOS Limited. Multos developer's guide, 2021. URL: https://multos.com/wp-content/uploads/2020/09/MDG.pdf.

[18] Iso. *ISO/IEC 7816-3:2006*, November 2006.

[19] Iso. *ISO/IEC 7816-10:1999.*, 1999.

[20] Iso. *ISO/IEC 7816-4:2020*, 2020.

[21] Petr Vančo. Kryptografická podpora současných programovatelných čipových karet. Master's thesis, Brno University of Technology, 2019.

[22] Michal Kočíř. Michal použití smart-karet v moderní kryptografii. Master's thesis, Brno University of Technology, 2013.

[23] Alexandr Kuckir. Programování v operačním systému multos. Master's thesis, Brno University of Technology, 2009.

[24] Chromium. Saying goodbye to our old friend npapi, September 2013. URL: https://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html?hl=hu.

[25] Java - applet basics. URL: https://www.tutorialspoint.com/java/java_applet_basics.htm.

[26] Oracle. Jdk 9 release notes, 2021. URL: http://www.oracle.com/technetwork/java/javase/9-deprecated-features-3745636.html.

[27] François Beaufort. Access usb devices on the web, 2016. URL: https://web.dev/usb/.

[28] Reilly Grant. Intent to implement and ship: Webusb interface class filtering, March 2018. URL: https://groups.google.com/a/chromium.org/g/blink-dev/c/LZXocaeCwDw/m/GLfAffGLAAAJ.

[29] Chrome Enterprise and Education Help. Use smart cards on chrome os - chrome enterprise and education help. URL: `https://support.google.com/chrome/a/answer/7014689?hl=en&ref_topic=7015274`.

[30] Chromium Blog. Changes to the chrome app support timeline, August 2020. URL: `https://blog.chromium.org/2020/08/changes-to-chrome-app-support-timeline.html`.

[31] Šárka Chwastková. Webová vizualizace a demonstrátor anonymních pověření. Master's thesis, Brno University of Technology, 2021.

[32] Ondřej Malík. Kryptografie a ochrana soukromí. Master's thesis, Brno University of Technology, 2021.

[33] Kapil Sachdeva, Karen Lu, and Ksheerabdhi Krishna. A browser-based approach to smart card connectivity, 2019. URL: `http://www.ieee-security.org/TC/W2SP/2009/papers/s4p4.pdf`.

[34] W3C. Web authentication: An api for accessing public key credentials - level 2, April 2021. URL: `https://www.w3.org/TR/webauthn/`.

[35] Petr Muzikant and Jan Hajný. Integrating smart-card authentication to web applications. *ICUMT 2022*, June 2022.

[36] ACSC. Australian government information security manual: Guidelines for cryptography, 2021. URL: `https://www.cyber.gov.au/acsc/view-all-content/advice/guidelines-cryptography`.

[37] European Pauments Council. Guidelines on cryptographic algorithms usage and key management., 2021. URL: `https://www.europeanpaymentscouncil.eu/document-library/guidance-documents/guidelines-cryptographic-algorithms-usage-and-key-management`.

[38] Gaëtan Leurent and Thomas Peyrin. Sha-1 is a shambles*. *Cryptology ePrint Archive*, 2021.

[39] IBM. Socket programming, 2012. URL: `https://www.ibm.com/docs/en/i/7.1?topic=communications-socket-programming`.

[40] Mart Sõmermaa. web-eid/web-eid-system-architecture-doc: The web eid project enables usage of european union electronic identity smart cards for secure authentication and digital signing of documents on the web using public-key cryptography, 2022. URL: `https://github.com/web-eid/web-eid-system-architecture-doc`.

[41] Mart Sõmermaa. Web-eid component model, 2021. URL: https://github.com/web-eid/web-eid-system-architecture-doc/blob/master/diagrams/Web-eID-Component-Model.png.

[42] Mart Sõmermaa. Web-eid authentication communication diagram, 2021. URL: https://github.com/web-eid/web-eid-system-architecture-doc/blob/master/diagrams/Web-eID-authentication-communication-diagram.png.

[43] Tõnis Reimo, Sandhra-Mirella Valdma, Kristjan Krips, Oruuas Mart, Pankova Alisa, and Jan Willemson. Analysis of planned architectural changes in open-eid, December 2020. URL: https://web-eid.github.io/web-eid-cybernetica-analysis/webextensions-main.pdf.

[44] Arnis Paršovs. On the format of the authentication proof used by ria's web eid solution, October 2021. URL: https://cybersec.ee/storage/webeid_auth_proof.pdf.

[45] Mart Sõmermaa. Github - web-eid/web-eid-system-architecture-doc: The web eid project enables usage of european union electronic identity smart cards for secure authentication and digital signing of documents on the web using public-key cryptography: Protection against man-in-the-middle attacks during authentication with origin validation, 2022. URL: https://github.com/web-eid/web-eid-system-architecture-doc#protection-against-man-in-the-middle-attacks-during-authentication-with-origin-validation.

[46] The risk of session hijacking and man-in-the-middle attacks in web eid - id.ee. URL: https://www.id.ee/en/article/the-risk-of-session-hijacking-and-man-in-the-middle-attacks-in-web-eid/.

[47] Nextcloud. Nextcloud latest administration manual, 2022. URL: https://docs.nextcloud.com/server/latest/admin_manual/index.html.

[48] Nextcloud. Nextcloud developer documentation &#x2014; nextcloud latest developer manual latest documentation, 2022. URL: https://docs.nextcloud.com/server/latest/developer_manual/index.html.

[49] The Nextcloud Community. All apps - app store - nextcloud, 2022. URL: https://apps.nextcloud.com.

[50] D Hardt. The oauth 2.0 authorization framework. Technical report, Internet Engineering Task Force (IETF), October 2012.

[51] Auth0. What is oauth 2.0 and what does it do for you? - auth0. URL: https://auth0.com/intro-to-iam/what-is-oauth-2/.

[52] Andreas Leicher, Andreas U. Schmidt, and Yogendra Shah. *Smart OpenID: A Smart Card Based OpenID Protocol*, pages 75–86. Springer Berlin Heidelberg, 2012.

[53] Inc Sun Microsystems. Java card applet developer's guide, August 1998. URL: https://ftp.icm.edu.pl/packages/javasoft-docs/javacard/AppletDevelopersGuide.pdf.

[54] Ruim Tools. Ruimtools: Javacard best programming guidelines, 2018. URL: http://ruimtools.com/doc.php?doc=jc_best.

[55] Martin Paljak. Github - martinpaljak/oracle_javacard_sdks: Oracle javacard classic sdk-s for using as a git submodule for ant-javacard projects, April 2022. URL: https://github.com/martinpaljak/oracle_javacard_sdks.

[56] Martin Paljak. Github - martinpaljak/ant-javacard: Easy to use ant task for building javacard classic applets (2.1.1 to 3.1.0), February 2022. URL: https://github.com/martinpaljak/ant-javacard.

[57] Martin Paljak. Github - martinpaljak/globalplatformpro: Manage applets and keys on javacard-s like a pro (via command line or from your java project), June 2022. URL: https://github.com/martinpaljak/GlobalPlatformPro.

[58] Inc. GlobalPlatform. Card specification - iso framework, March 2014. URL: https://globalplatform.org/wp-content/uploads/2014/03/GPC_ISO_Framework_v1.0.pdf.

[59] Martin Paljak. esteid-applets/fakeesteid.md at master · martinpaljak/esteid-applets, June 2017. URL: https://github.com/martinpaljak/esteid-applets/blob/master/docs/FakeEstEID.md.

[60] Philip Wendland. Github - philipwendland/isoapplet: A java card pki applet aiming to be iso 7816 compliant, December 2019. URL: https://github.com/philipWendland/IsoApplet.

[61] Certicom Corp. Sec 2: Recommended elliptic curve domain parameters, September 2000. URL: https://www.secg.org/SEC2-Ver-1.0.pdf.

[62] EFT Lab. Complete list of apdu responses - eftlab - breakthrough payment technologies. URL: https://www.eftlab.com/knowledge-base/complete-list-of-apdu-responses/.

[63] Cheef. Cheef&apos;s personal site., June 2009. URL: http://web.archive.org/web/20090623030155/http://cheef.ru/docs/HowTo/SW1SW2.info.

[64] D. Cooper, S. Santesson, S. Farrel, S. Boeyen, R. Housley, and W. Polk. Rfc 5280 - internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, May 2008. URL: `https://datatracker.ietf.org/doc/html/rfc5280`.

[65] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. Rfc 6960 - x.509 internet public key infrastructure online certificate status protocol - ocsp, June 2013. URL: `https://datatracker.ietf.org/doc/html/rfc6960`.

[66] Petr Švenda. Javacard algorithm test, 2022. URL: `https://www.fi.muni.cz/~xsvenda/jcalgtest/`.

[67] ParalelniPolis. Github - paralelnipolis/obcanka-public: Tool to generate unofficial issuer's certificate for eobcanka, January 2019. URL: `https://github.com/ParalelniPolis/obcanka-public`.

A Code listings

Listing A.1: Example session-backed challenge nonce store implementation

```
class SessionBackedChallengeNonceStore extends
   ChallengeNonceStore {
 private const CHALLENGE_NONCE_KEY = 'web-eid-challenge-nonce
    ';
 private $session;

 public function __construct(ISession $session) {
  $this->session = $session;
 }

 public function put(ChallengeNonce $challengeNonce): void {
  $this->session[self::CHALLENGE_NONCE_KEY] = serialize(
     $challengeNonce);
 }

 protected function getAndRemoveImpl(): ?ChallengeNonce {
  if (!$this->session[self::CHALLENGE_NONCE_KEY]) {
   return null;
  }

  $challengeNonce = unserialize($this->session[self::
     CHALLENGE_NONCE_KEY], array(
   'allowed_classes' => array(ChallengeNonce::class, DateTime
      ::class),
  ));

  if (!$challengeNonce) {
   return null;
  }

  unset($this->session[self::CHALLENGE_NONCE_KEY]);

  return $challengeNonce;
 }
}
```

Listing A.2: Challenge nonce generator builder usage example

```
use muzosh\web_eid_authtoken_validation_php\challenge\
   ChallengeNonceGenerator;
use muzosh\web_eid_authtoken_validation_php\challenge\
   ChallengeNonceGeneratorBuilder;
use muzosh\web_eid_authtoken_validation_php\challenge\
   ChallengeNonceStore;

...
public function getGenerator(ChallengeNonceStore
   $challengeNonceStore): ChallengeNonceGenerator {
  return (new ChallengeNonceGeneratorBuilder())
   ->withNonceTtl($this->config['CHALLENGE_NONCE_TTL_SECONDS'
      ]) // 300
   ->withChallengeNonceStore($challengeNonceStore)
   ->build()
  ;
 }
...
```

Listing A.3: Authentication token validator configuration example

```
use GuzzleHttp\Psr7\Uri;
use muzosh\web_eid_authtoken_validation_php\validator\
   AuthTokenValidator;
use muzosh\web_eid_authtoken_validation_php\validator\
   AuthTokenValidatorBuilder;

...
public function getValidator(): AuthTokenValidator {
  return (new AuthTokenValidatorBuilder())
   ->withSiteOrigin(new Uri($this->config['ORIGIN'])) // '
      https://'.$_SERVER['SERVER_ADDR']
   ->withTrustedCertificateAuthorities(...self::
      loadTrustedCACertificatesFromCertFiles())
   ->withoutUserCertificateRevocationCheckWithOcsp()
   ->build()
  ;
 }
...
```

Listing A.4: Example of providing challenge nonce to the client

```
/**
  * Get the template for rending the 2FA provider view.
  */
 public function getTemplate(IUser $user): Template {
  $generator = $this->webEidService->getGenerator(
   $this->webEidService->getSessionBasedChallengeNonceStore()
  );
  $challengeNonce = $generator->generateAndStoreNonce();

  $template = new Template(Application::APP_NAME, '
     WebEidChallenge');
  $template->append('nonce', $challengeNonce->
     getBase64EncodedNonce()); // challenge is appended to
     HTML hidden input

  return $template;
 }
```

Listing A.5: Front-end code for obtaining authentication token and sending it back for validation

```
submitButton.addEventListener("click", async () => {
  try {
   showSpinner();
   hideError();
   const authToken = await webeid.authenticate(nonce, lang);
   document.querySelector("#webeid-token").value =
    JSON.stringify(authToken);
   form.submit();
  } catch (error) {
   ...
  }
```

Listing A.6: Nextcloud authentication result configuration

```php
public function authenticate(X509 $cert, IUser $user): bool {
  $certCN = CertificateData::getSubjectCN($cert);

  if ($user->getUID() == $certCN) {
    return true;
  }

  $this->logger->error(
    'WebEid authtoken validation successful, but CommonName
       does not match. UserID: '.
    $user->getUID().
    ', CN: '..
    $certCN
  );

  return false;
}
```

B Content of the electronic attachment

The following diagram depicts and describes directories in the electronic attachment. General files have been excluded for readability. Please note that included `README.md` files might provide a better source of information.

```
/................................................root of the attached archive
├── github-links.txt.........................URL links to up-to-date source code
├── InfinitEID
│   ├── README.md
│   └── src
│       ├── InfinitEID-applet ........................ JavaCard applet source code
│       │   └── README.md
│       └── InfinitEID-card-management.....initialization, management and testing
│           └── README.md
├── libelectronic-id-with-InfinitEID .... Web-eID Native Application extension
│   └── README.md
├── nextcloud_twofactor_webeid...................installable Nextcloud module
│   └── README.md
└── web-eid-authtoken-validation-php . Web-eID token validation library for PHP
    └── README.md
```