P. J. Safarik University

Faculty of Science

# REPLACING HANDWRITTEN SIGNATURES WITH OPEN ELECTRONIC SIGNATURE SOFTWARE

## BACHELOR'S THESIS

| | |
|---|---|
| **Field of Study:** | **Applied Informatics** |
| **Institute:** | **Institute of Computer Science** |
| **Tutor:** | **RNDr. Viliam Kačala** |

**Košice 2020**

**Jakub Ďuraš**

## Acknowledgments

I want to thank my supervisor RNDr. Viliam Kačala for his continuous feedback and patience with my questions and late submissions. I would also like to acknowledge help received from RNDr. JUDr. Pavol Sokol, PhD. and prof. RNDr. Gabriel Semanišin, PhD. Finally, I would like to thank my family and friends who contributed in various ways to my thesis.


P. J. Šafárik University in Košice
Faculty of Science

# THESIS ASSIGNMENT

| | |
|---|---|
| **Name and Surname:** | Jakub Ďuraš |
| **Study programme:** | Applied Informatics (Single degree study, bachelor I. deg., external form) |
| **Field of Study:** | 18. - Computer Science |
| **Type of Thesis:** | Bachelor thesis |
| **Language of Thesis:** | English |
| **Secondary language:** | Slovak |

**Title:** Replacing handwritten signatures with open electronic signature software

**Title SK:** Náhrada vlastnoručných podpisov otvoreným softvérom na elektronické podpisovanie

**Aims:** 1. Explore the principles and legal status of electronic signatures.
2. Review existing software and propose and develop open-source, cross-platform, and user-friendly platform for electronic document signing.
3. Provide information and a way to create signatures compliant with eIDAS Regulation (Regulation No 910⁄2014 ).

**References:** [1] MASON, Stephen, 2017. Electronic Signatures in Law: Fourth Edition. London: University of London. ISBN 978-1-911507-01-7. Available at: https://humanities-digital-library.org/index.php/hdl/catalog/view/electronicsignatures/1/86-1
[2] PAAR, Christof and Pelzl, Jan, 2009. Understanding Cryptography - A Textbook for Students and Practitioner. Berlin: Springer. ISBN 978-3-642-04100-6
[3] SOMMERVILLE, Ian, 2016. Software Engineering, 10th Edition. Harlow: Pearson Education. ISBN 978-0-13-394303-0.

**Keywords:** electronic signature, electronic seal, qualified, open-source, XAdES, PAdES, CAdES, desktop software

**Supervisor:** RNDr. Viliam Kačala
**Institutes :** ÚINF - Institute of Computer Science
**Head of Institute:** RNDr. Ondrej Krídlo, PhD.
**Electronic version available:** unlimited

**Approved:** 10.05.2020 Prof. RNDr. Viliam Geffert, DrSc.
riaditeľ ústavu

# Abstrakt

So zmenami v právnom statuse elektronických podpisov vo svete sa vytvára príležitosť rozšíriť elektronické podpisovanie ako plnohodnotnú náhradu pre vlastnoručné podpisy. V práci sa venujeme právnym predpokladom relevantným pre EÚ, technologickým predpokladom, prehľadu v súčasnosti dostupného softvéru a návrhu vlastnej softvérovej platformy. Výsledkom je otvorená softvérová platforma pre operačné systémy Windows, macOS a Linux. Pomocou aplikácie je možné, v súlade s nariadením eIDAS, podpísať a overiť podpis ľubovoľného dokumentu, ktorý má rovnaký právny účinok ako osvedčený podpis. Naše riešenie podporuje občianske preukazy Slovenskej Republiky s čipom eID bez akýchkoľvek úprav alebo nastavení, čím sa umožňuje ich využitie mimo použitia vo webových aplikáciách štátu. Aplikácia nie je viazaná iba na slovenské eID, ale je pripravená na ďalšie použitie s rôznymi zariadeniami v rôznych štátoch a ako platforma pre iné druhy podpisov. Súčasťou je aj webová stránka, ktorá okrem možnosti stiahnutia softvéru a stručnej dokumentácie poskytuje všeobecné informácie o elektronických podpisoch. Dôraz je kladený na jednoduchú rozšíriteľnosť programu vo forme jazykových prekladov, dokumentácie, funkcionality a všeobecných informácií. Prácu je možné ďalej rozšíriť nepreskúmanými možnosťami podpisovania či zlepšením flexibility, ktorá nie je na úrovni niektorých zástupcov porovnávaného softvéru.

**Kľúčové slová:** *elektronický podpis, elektronická pečať, kvalifikovaný, open-source, XAdES, PAdES, CAdES, počítačový softvér*

## Abstract

With the recent changes in the legal status of electronic signatures in the world, there is an opportunity to expand the use of electronic signatures as an alternative to handwritten signatures outside the usual areas. In this work, we devote our attention to the exploration of the legal preconditions relevant for the EU, technological preconditions, review of the existing software, and a proposal of our own software platform. The result is an open software platform for operating systems Windows, macOS, and Linux. Using this application, it is possible, following the eIDAS Regulation, to sign and verify the signature of any document with the legal effect of an attested signature. Our solution supports ID cards of the Slovak Republic with the eID chip without any modifications or settings, which enables use outside the web applications of the state. The application is not tied only to the Slovak eID but is ready for further use with various devices in other countries and as a platform for other types of signatures. There is also a website that, in addition to the possibility of downloading software and brief documentation, provides general information about electronic signatures. The emphasis is on the simple extensibility of the program via language translations, documentation, functionality, and general information. The thesis can be further extended with novel ways of signing or by improving flexibility, which is not completely up to par with the compared software.



**Keywords:** *electronic signature, electronic seal, qualified, open-source, XAdES, PAdES, CAdES, desktop software*

# Contents

# Introduction

Electronic signatures have been around for a while. Their practical usability, however, is influenced by their legal status and available infrastructure. With the recent changes to the relevant laws around the world, and in the European Union specifically, they are seeing increased adoption.

While, specifically in the European Union, electronic signatures seem to be used relatively extensively in communication with the government, they do not seem to have been widely adopted in exchanges between individuals or smaller organizations, which still seem to prefer handwritten signatures. In many cases, however, they may be impractical for electronic communication, due to the need of a person's physical presence or the printing and sending of documents.

Our goal is to explore an accessible way of using electronic signatures. In the beginning, we go over the preconditions that make it a viable alternative to handwritten signatures. We cover the legal preconditions relevant to the European Union, and Slovakia in particular. Also, we explain the technical preconditions, focusing on cryptographic concepts like asymmetric cryptography, timestamping, hashing, and standards that comply with the laws of the European Union. Secondly, we do a review of the existing software, where we briefly cover several applications that are meant for basic use. We compare them using several criteria to find out the strengths and weaknesses of the current implementations and identify a potential gap. Finally, we devote our attention to our own software's implementation. We explain the proposed desktop software and go into details on software engineering aspects. Specifically, we cover usability, localization, modularity, architecture, testing, and automation. In the end, we compare it to the existing software and identify possible future improvements. Another result we focus on is a supporting website and how it helps with the use of the software, all in a way that is open for extension by the community.

# 1 Preconditions and theory

There are several preconditions that allow us to consider electronic signatures as an alternative. Firstly, the current state of the law and recent changes relating to the use of electronic signatures. Secondly, well-established concepts in cryptography like asymmetric cryptography or hashing. Lastly, expertise from the field of software engineering - design, implementation, and maintenance of computer software. We will concentrate on the first two in relation to the electronic signatures and where applicable open-source desktop computer software.

## 1.1 Law

Unless stated otherwise, we are considering law applicable locally in the Slovak Republic (SR). Its law is greatly influenced by the European Union (EU), being its member since 2004, and by the rest of the world. We can assume this is at least partially applicable outside of the SR as well. Our overall view on the global legal status is shaped by [6]. Subsection 1.1.3 is largely based on Chapters 1, 2, 3, 6, and 7 of [7].

### 1.1.1 Signatures

Signatures are an essential part of the written legally binding documents like contracts. They are permanently affixed to the document and are supposed to uniquely identify the person and its deliberate, informed consent. As can be seen in the Slovak Civil Code, § 40, a written legal act is valid if signed by the acting person [1]. Our law often explicitly requires signatures and further clarifies their expected use. For example, when selling an enterprise, looking at the Commercial Code, § 476, the contract requires written form and attested signatures of the seller and the buyer [2].

Signature is considered an *attested signature*[1] if it is verified by an authorized third party. This process is referred to as *legalization* in Notary Law, § 56, and its purpose

[1] Translation from "osvedčený podpis" as used in the Slovak law.

is to attest information that could form the basis for the exercise of rights or which could cause legal consequences [3].

Regarding the electronic signatures, law within the EU used to differ, with the law applicable in the SR being now-repealed *Act No. 215/2002 Coll.*. This situation changed with the EU Regulation eIDAS.

### 1.1.2 EU Regulation eIDAS

With intention to stimulate digital growth by building trust, the EU established regulation on electronic identification and trust services for electronic transactions in the internal market (eIDAS). It applies from 1st of July 2016, replaces local law, and regulates, among other things, electronic signatures and its more specific variants.

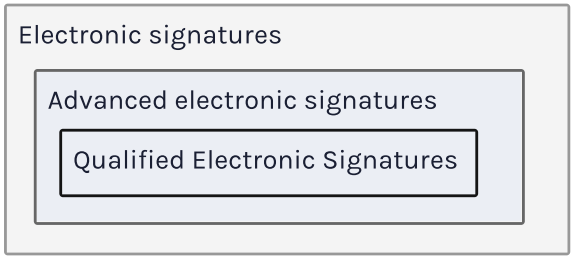


Figure 1.1: Relation between electronic, advanced, and qualified signatures.

In general, electronic signature can be represented in different ways (e.g. as an image, text or other data attached to the document) and they can not be denied in legal proceedings just because they are in the electronic form [4].

*Advanced electronic signatures* are a subset of electronic signatures that have to uniquely link and identify the signature author, be created in a way that is possible only by them and any changes to the signed document have to be detectable [4]. From the technical point of view, standards *PAdES*, *XAdES*, and *CAdES* specified by the European Telecommunications Standards Institute (ETSI) comply with these requirements. We explain why this is the case in Section 1.2.

*Qualified electronic signatures* (QES) are a subset of advanced electronic signatures with more specific requirements that can be found in annex 1 of the eIDAS regulation[2]. In more practical terms, such advanced electronic signatures are created with a qualified device using a qualified certificate and a qualified trust service. Qualified,

[2]Available at `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32014R0910\#d1e32-111-1`.

in this case, means authorized for such use by the legal authorities. These devices, certificates, and trust services are explicitly listed by the EU. In many cases indirectly by listing the organization (and its items) from a specific country.

Furthermore, a qualified electronic signature has the legal effect of a handwritten signature [4] and attested handwritten signatures in the SR.

Mentioned terminology is used when considering a natural person. Legal entities are able to use *electronic seals*, which are, from the technical point of view and for our purposes, identical to electronic signatures.

Overall, electronic signatures are supposed to be, within the EU, interoperable and transparent alternative to handwritten signatures.

### 1.1.3 Software copyright

Computer programs and associated materials are historically protected under copyright as literary works, and protected authors should be able to authorize or prohibit certain acts [5].

Such right can be exercised to license the software under either proprietary or open-source license. Proprietary meaning under the exclusive legal right of the author, typically also confidential and distributed as a paid product. Open-source meaning having its source code freely available, typically also distributed for free and without any liability (the software is provided "as is").

Motivation to license the software under an open-source license can differ, and so do such licenses. While a proprietary license is usually made specifically for that entity and its interests, open-source licenses tend to be reused between different authors. This means that consumers of the software can quickly recognize their rights and responsibilities if they decide to use, modify, or distribute the software. In general, we can classify the open-source licenses into two categories: *permissive* and *copyleft*.

Copyleft license, in general, requires the user to publish the modified work under the compatible (free) license. As a result, it forces them to extend the rights they have received onto others and, in turn, somehow hinder usability in software licensed under a different license. A popular example of such license is the GNU General Public License (GPL) and its derivatives like GNU Lesser GPL (LGPL) or GNU Affero GPL (AGPL).

Permissive licenses, on the other hand, do not have such a requirement in place and are more suitable for potential commercialization. They typically allow commercial

use, modifications, distribution or sublicensing as long as the author is not held liable, and the original copyright and license are distributed with the software. Often used permissive licenses are MIT License, Apache License, or BSD License.

It is not uncommon to see hybrid licensing - which means licensing under more than one license, with one being typically open-source and one proprietary. In this scenario, users can choose which license they want to use based on their needs.

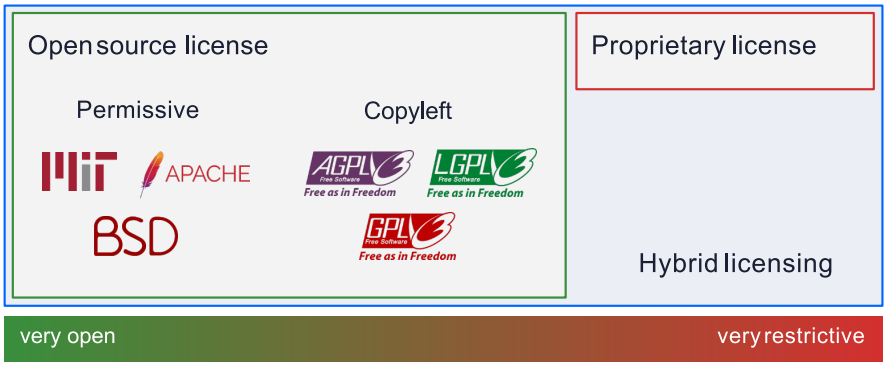


Figure 1.2: Classification and comparison of software licensing models.

As we have already mentioned, software licenses can influence whether and how it can be used as a part of different software. Both when we are the ones using someone else's software and when someone else is using our software. Generally speaking, the more open the license is, the more likely it is it will not hinder adoption for legal reasons. We provide examples of categorized licenses in Figure 1.2.

## 1.2 Cryptography

Previously mentioned standards PAdES, XAdES, and CAdES define how asymmetric cryptography, hashing, or timestamping should be used to comply with the standards. We go over these topics and why they are able to satisfy the requirements for QES mentioned in the Section 1.1.2. We draw our knowledge in chapters about cryptographic primitives from the [8] and [9].

### 1.2.1 Asymmetric cryptography

There are several attributes that make symmetric encryption unsuitable. Let us imagine we are creating a key for each pair of users in our network - imaginary country Isle of Alices and Bobs - for symmetric encryption and decryption of documents shared between each Alice and Bob.

Secure distribution of keys seems to be feasible when we are adding a new user to our network. We require each user to personally come and pick up a physical device that contains pre-generated keys. Changes made after that, however, would mean we either need to require a periodic personal visit to update the saved keys or transmission of these keys over some medium, which can be potentially unsafe. Any new members of the network are unable to participate until other Alices and Bobs get their updates.

Even if we think we have established a reasonably secure way to do that and we are willing to wait, we realize the number of keys is quickly increasing with each new member. With $n$ users, the number of keys is

$$\frac{n \cdot (n-1)}{2}$$

which means that even our little imaginary country with a population of only 70 000 requires roughly 2.5 billion keys. However, since our signature system needs to be portable, we will need to count also with a population of other countries, and so the number of keys grows very quickly.

Either party can also lie about not signing the document. Since both have the key that can be used for encryption, they can sign any document on behalf of the other party, and there is no way to disprove it if we look at it as a third party.

These issues can be addressed with asymmetric cryptography. In general, we assign a pair of keys to each person, one of them used for encryption, known as a *public key*, another used for decryption, known as a *private key*. One party can encrypt, and the other is the only one that can decrypt.

*Digital signatures* are specific asymmetric cryptography algorithms. The signer is the only one who possesses the private key used for signing, and others can use the public key for signature verification. We sign the document, including the other parties' signature.

Practical and widely used implementation of this idea is the *RSA* signature scheme that relies on the integer factorization problem published in 1978 [10].

In Slovakia, the private key is usually handed over to the user personally, saved on the ID card. This ID card, together with a card reader, provides an *Application Programming Inteface* (API) for safe access, management, and cryptoprocessing. Such a device is in general called a *Hardware Security Module* (HSM), and the API relevant

for this thesis is *PKCS #11* [3].

We are left with the problem of public key distribution - even though public-key schemes do not require a *secure channel*, they require *authenticated channels* for the distribution of the public keys [8].

A common solution is the use of *certificates*. Certificates are essentially digital signatures with metadata that can be used to establish the validity of the received public key before it is used. Distribution of such certificates is the responsibility of a *Certification Authority* (CA) - a third party that all users in the network trust. In the case of the Slovak ID cards, it is Disig - SVK eID Accredited CA, issued by the National Security Authority[4]. Such certificates, therefore, reliably uniquely link and identify the public key owners, as can be seen in Figure 1.3[5]. Ability to do that is the first requirement for the advanced electronic signatures mentioned in Section 1.1.2. Commonly used standard for public key certificates is X.509[6].

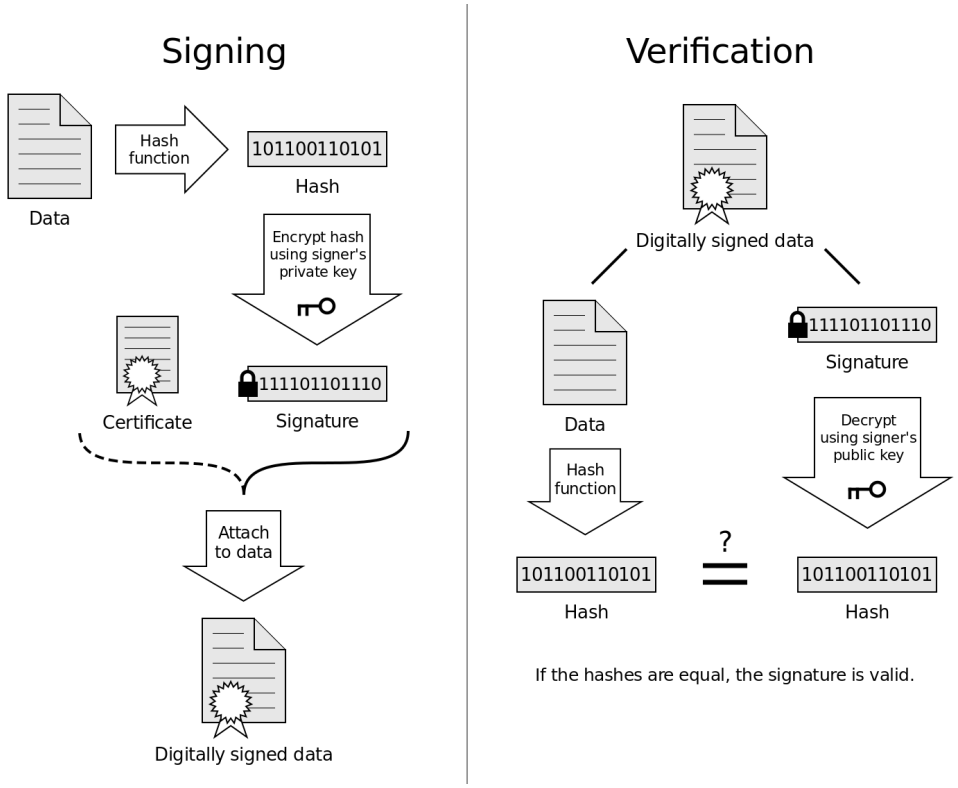


Figure 1.3: Application of digital signature with certificate.

Specific for the EU is online *List of The Lists* (LOTL) that contains a list with

[3]PKCS #11 specification available at `http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html`.

[4]Translation from "Národný bezpečnostný úrad".

[5]Created by an unknown author, licensed under the CC BY-SA 3.0 license.

[6]Specification available under RFC5280 at `https://tools.ietf.org/html/rfc5280`.

trusted service providers for each country. Each member state is obligated to maintain such list, as can be seen in Article 22 of [4]. From the technical point of view, it is an XML document that is signed and can be processed automatically.

### 1.2.2 Hashing and timestamping

It is often useful to map data of arbitrary size to a fixed-length and typically short set of bits called *hash*. Function that can be used for it is called *hash function*.

In cryptography, we are considering only hash functions that can also satisfy these additional requirements:

1. **Deterministic** - same input data are always mapped to the same hash.
2. **One-way** - it should be infeasible to retrieve input data for a given hash.
3. **Resistant to collisions** - it should be infeasible to find two inputs with the same hash.
4. **Avalanche effect** - even small change to the input should lead to significantly different hash.

In the context of digital documents, we can use such hash function to generate a hash for the document. This hash can represent a document of any size and is used during the creation of a digital signature instead of the document itself. Any further changes to the document will, therefore, lead to a significantly different hash, which means the digital signature will no longer be valid, and these changes are detected. This ability is one of the requirements for the advanced electronic signatures from Section 1.1.2. For this purpose, only hash functions that are reasonably fast can be used, so that we can utilize them on documents of variable size.

An example of commonly used hash function family is *Secure Hash Algorithms* (SHA), with hash functions relevant to this thesis being mainly SHA-2[7].

A common part of the paper signed documents is the date when they were signed. Their digital counterpart is PKI-based *trusted timestamping* - a process of digitally signing date and time of document creation or modification. No one should be able to tamper with such timestamp - not even the author of the document. Additionally, we

[7]SHA-2 specification available in FIPS PUB 180-4 at `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`.

should not be able to choose arbitrary date and time ourselves, but it should represent the actual time at the signing.

To achieve this, a trusted third party - *Timestamping Authority* (TSA) - is the one that will sign the hash of the document together with the current date and time. This process can be seen in Figure 1.4[8]. Verifying can be done by decrypting this information using the public key and comparing the hashes. Any attempt at tampering is therefore detected, which helps to cover requirements for the advanced electronic signatures from Section 1.1.2.

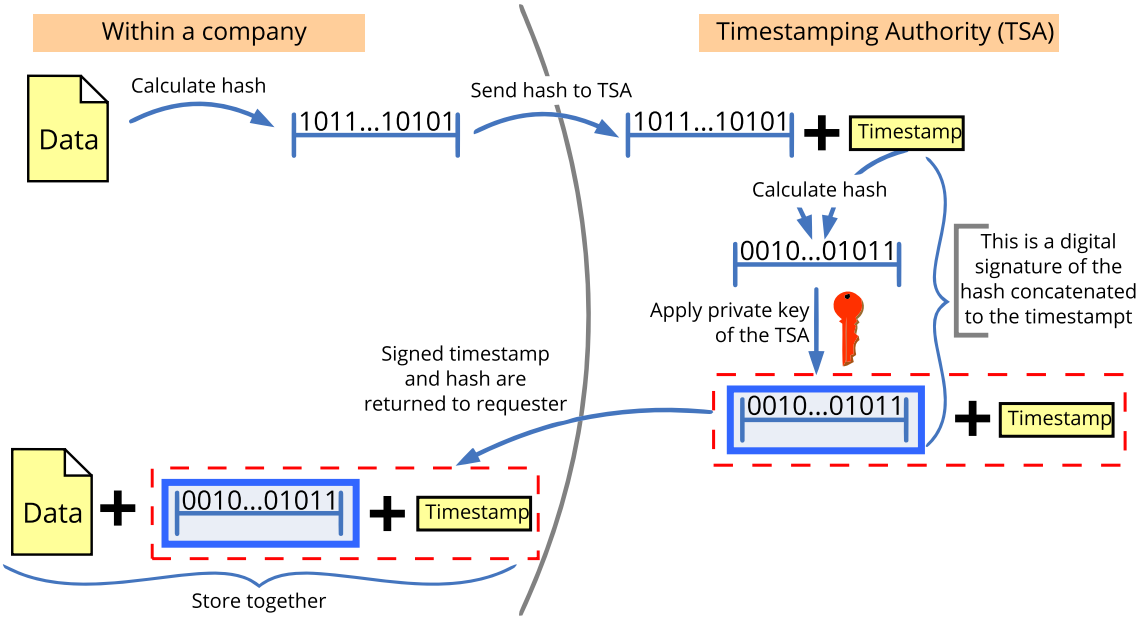


Figure 1.4: Setting a timestamp via a trusted third party.

The commonly used standard for the *Public Key Infrastructure* (PKI) based timestamping is part of the X.509[9].

### 1.2.3 PAdES, XAdES, and CAdES

Extensions for Portable Document Format (PDF) files, eXtensible Markup Language (XML) files, and Cryptographic Message Syntax (CMS) signed data for signing using advanced electronic signatures, as explained in Section 1.1.2, are PAdES, XAdES, and CAdES, respectively. They are defined in the respective specifications by the ETSI in [11], [12], and [13].

When using PAdES, signatures are always part of the PDF file itself. This differs

[8]Schema created by Bart Van den Bosch, licensed under the CC BY-SA 2.0 BE license.
[9]Specification available under RFC3161 at `https://www.ietf.org/rfc/rfc3161.txt`.

from XAdES, where the signature can be either part of the XML or provided as a separate file, or CAdES where the signature is always in the form of binary data, and it is up to the software.

All of them support multiple signatures applied in succession. Implementation of the PAdES is rather portable since any compatible PDF file reader or editor will work with the signatures in a similar way - signature created in one should be easily verifiable in any other. XAdES and CAdES, however, are usually tied to specific implementation or standard that is built on top of them.

| Standard | Usable with | Format | Multi-signature | Appearance |
|---|---|---|---|---|
| PAdES | PDF | embedded | yes, sequential | supported |
| XAdES | XML, any | XML | yes | depends on the usage |
| CAdES | any | binary | yes | depends on the usage |

Table 1.1: Overview of standards PAdES, XAdES, and CAdES.

The ability to keep signed documents valid for a prolonged period of time is covered in these standards under optional *Long-Term Validation* (LTV) that allows archiving of documents for many years, possibly decades. If enabled, a trustworthy timestamp is required as validity is verified for the time the signature (and therefore timestamp) was created. The expiration time is then limited by the validity of the timestamp certificate. Different profiles - levels - for different standards can be seen in Table 1.2.

| Level | XAdES | CAdES | PAdES |
|---|---|---|---|
| Basic | XAdES-B | CAdES-B | AdES-B |
| B with Timestamp | XAdES-T | CAdES-T | PAdES-T |
| T with Long Term Data | XAdES-LT | CAdES-LT | PAdES-LT |
| LT with Archive timestamp | XAdES-LTA | CAdES-LTA | PAdES-LTA |

Table 1.2: Different profiles - levels - for standards PAdES, XAdES, and CAdES.

# 2 Review of existing software

We focus only on simple applications made mostly for the signing of the documents. During our review, we have used several PDF files that we have tried to sign using QES available on the SR eID. We have tried to include software of all types in our review, preferring the most popular, locally available ones, since those should be prepared for the use with the SR eID. Excluded are all complex applications with advanced features and high price tag. We have picked one open-source desktop application, one web application - *Software as a Service* (SaaS), one PDF viewer, one simple paid desktop application, and one commercial desktop application with free version. Criteria we focus on are:

1. **Licensing** - is it freely available - both in terms of money and source code - best case is open-source; worst case is paid proprietary software.

2. **Cross-platform compatibility** - is it available for users of all major desktop platforms - best case is availability on Windows, macOS, and Linux; worst case is availability only on one platform.

3. **Localization** - is it translated and does it display data like dates in local format - best case is availability at least in English and Slovak language; worst case is availability only in one language with no support for localization of dates, etc..

4. **File support** - does it support different file formats and sizes - best case is support for all file formats with sensible size up to 50 MB; worst case is availability only for one file format or strict limits on the file size.

5. **Easy setup** - does it require non-trivial setup to use the SR eID - best case is no setup required; worst case is possibly demanding setup or troubleshooting.

6. **Flexibility** - how much logic can be customized - best case is various options for signature format, signature policy, used algorithms or timestamping configuration; worst case is none or almost none customization.

7. **Built-in verification** - is the verification of signatures available in the same software - best case is ability to verify various signature types; worst case is no support for signature verification.

8. **Extensibility** - is it extensible by the community - best case is it is prepared for easy extensibility in terms of functionality and localization; worst case is no extensibility.

We have summarized our findings in Table 2.1. Value ✓✓ means best-case scenario, no value worst-case scenario, and value ✓ somewhere in between - partially or depends.

| | licensing | cross-platform | localization | file suppport | easy setup | flexibility | built-in verification | extensibility |
|---|---|---|---|---|---|---|---|---|
| JSignPdf | ✓✓ | ✓ | | | | ✓✓ | | ✓ |
| zep.disig.sk | ✓ | ✓✓ | ✓✓ | ✓ | ✓✓ | | ✓✓ | |
| Acrobat DC | ✓ | ✓ | ✓✓ | | ✓ | ✓✓ | ✓✓ | |
| D.PDF Signer | | | ✓ | | ✓✓ | ✓ | | |
| Podpisuj.sk | ✓ | ✓✓ | ✓ | ✓✓ | ✓✓ | ✓ | ✓✓ | |

Table 2.1: Comparison of existing software.

## 2.1 Desktop application JSignPdf

JSignPdf is an open-source Java application that adds digital signatures to PDF documents. It can be used as a standalone application or as an add-on in OpenOffice. We focus only on standalone use.

The application consists only of one window, which makes it very simple at first sight. The very first input - *Keystore type* - can be harder to understand for the basic user since it contains over a dozen options in the form of abbreviations like *CASEEXACTJKS* or *PKCS12-DEF-3DES-40RC2*. The default value is *WINDOWS-MY*, however, which is what is exactly required for the SR eID. Application has a toggle for the *Advanced view*, which is off by default, but it is necessary to turn it on if we want to choose the correct key. On some of the computers, we were not able to see the key from the eID. On others, it took a while and several pressing of a button

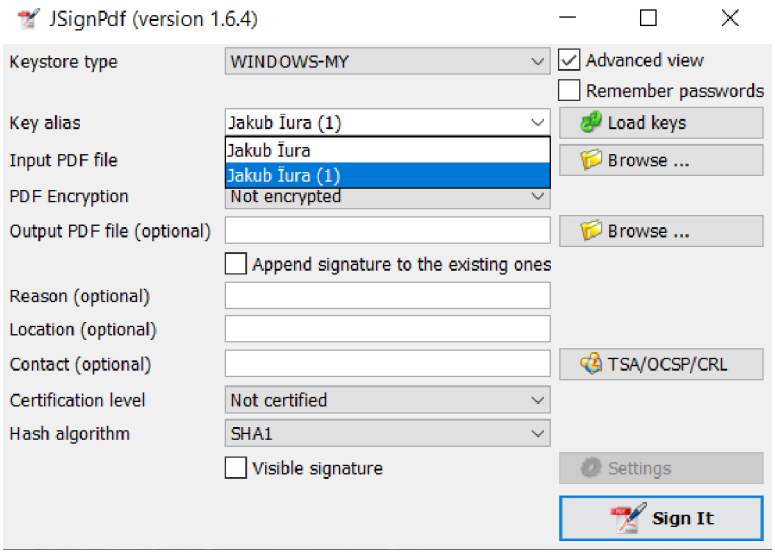


Figure 2.5: Screenshot of the JSignPdf version 1.6.4.

to load them. The correct key was not the first, preselected key in the form of *Name Surname*, but the second *Name Surname (1)*. With everything set up correctly, the application was successfully able to attach a QES. Even though the application looks pretty simple, it is still pretty flexible with its options regarding the used algorithms, TSA, OCSP, or CRL. That being said, it is available only in English, and we have also noticed issues with the diacritics as you can see in Figure 2.5 - "Jakub Ďuraš" is displayed as "Jakub Ďura".

## 2.2 Web application zep.disig.sk

Web application zep.disig.sk is a pro-bono SaaS from company Disig a.s. for user-friendly document signing and validation utilizing the currently bundled software with the SR eID.

The process to sign or validate is very simple and straightforward. There are, however, several potential problems. Since this is a web service, documents have to be uploaded. Any confidential or personal data are, therefore, shared with the company. Additionally, we have no way of knowing "how things work" in the background. File support is pretty extensive, but there is a strict restriction on file size at 4 MB - larger files such as documents scanned at home will possibly not be accepted. The website also provides valuable information, and the company seems to be open to questions from users - all completely for free.


Figure 2.6: Screenshot of the zep.disig.sk.

## 2.3 PDF viewer Adobe Acrobat Reader DC

One of the most popular PDF viewers from the company Adobe - Acrobat Reader DC - can also verify and add signatures to PDF files.

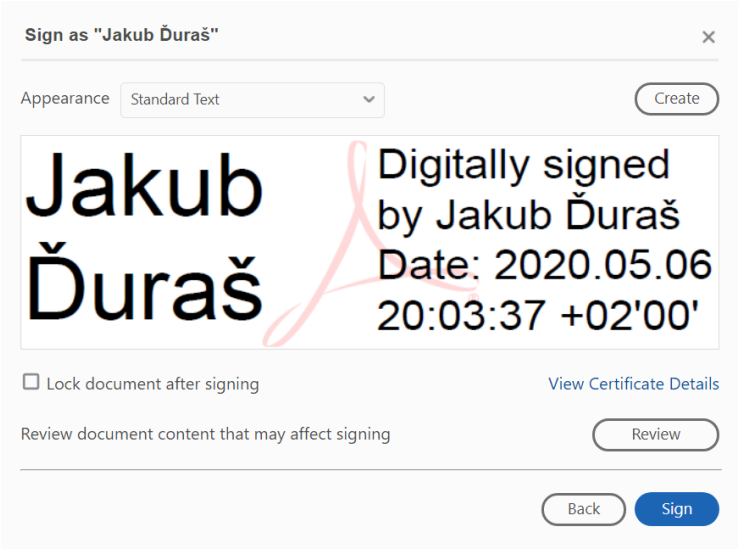


Figure 2.7: Screenshot of the Adobe Acrobat Reader DC.

Using the QES is slightly harder to find in the Adobe Acrobat Reader. It is not available under *Signing*, but *More Tools* where it is called *Certificates*. Setup of the certificate for use with the SR eID can be a hit or miss. On one of our computers, the application would not include a certificate from the SR eID, and we had to go through a process of adding the PKCS #11 library manually. That meant we had to go through several steps and windows and disable the *protected mode* option. The setup of this application may not be straightforward and may require professional help.

On another computer, we were simply able to choose from the list of the available certificates. That being said, everything is correctly localized, and being one of the most used applications of its kind, it is very easy to search for help in situations like this. Outside of that, verification of signatures is part of the application and is very responsive and flexible. The application is proprietary and can not be extended or verified. It is available only on the Windows and macOS.

## 2.4 Commercial application D.PDF Signer

Application D.PDF Signer is a paid proprietary desktop application.

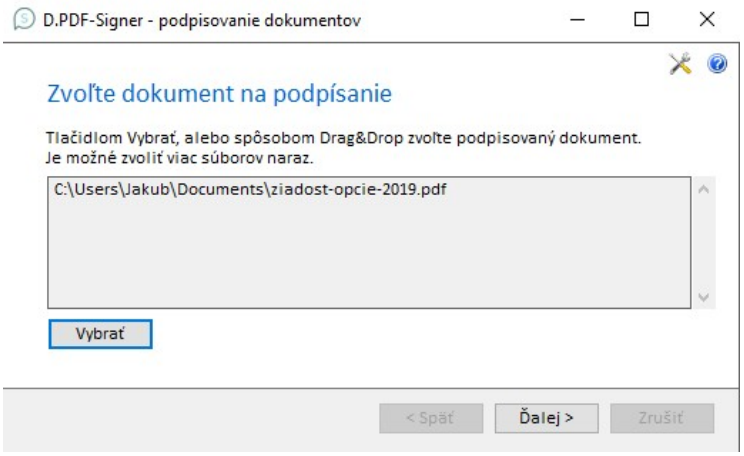


Figure 2.8: Screenshot of the D.PDF Signer.

Just like the zep.digis.sk, D.PDF signer utilizes the currently bundled software with the SR eID. It is straightforward and flexible. On the other hand, it lacks support for platforms other than Windows. Additionally, the application is only partially localized to English. It does not support file formats other than PDF and has no built-in signature verification.

## 2.5 Commercial desktop application Podpisuj.sk

Application Podpisuj.sk is a paid proprietary desktop application with a free version developed locally in Slovakia.

The Podpisuj.sk can be downloaded for all three major desktop platforms. It does not rely on the software bundled with the SR eID. It is relatively easy to use and comes fully preconfigured so it can be used with the SR eID right away. Flexibility

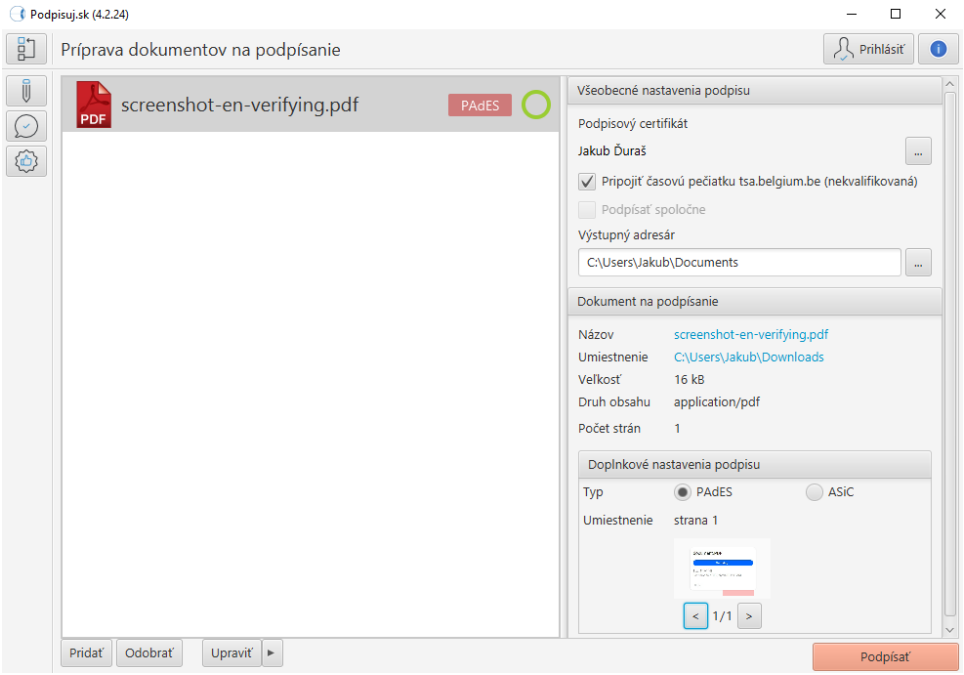


Figure 2.9: Screenshot of the Podpisuj.sk.

is not as great as with some other applications we have reviewed, and some options (like custom TSP) require one of the paid versions. Not everything is fully localized - especially the website, and any documentation we could find was available only in Slovak. Therefore it may be impractical for foreigners. It is not possible to extend it, but it is easy to use different tokens for signing.

# 3 Results

## 3.1 Desktop software

We have developed an open-source, cross-platform desktop software with focus on the modularity, architecture, usability, localization, automated testing with overall automation, and extensibility by the community. Our knowledge on these topics comes from Ian Sommerville's [14]. We try to consider specifics of the open-source software as they are sometimes defined in Chapter 7 of [18].

### 3.1.1 Usability

Our main objective for the usability was to provide the end-users with an easy to use, straightforward UI, and to make as many decisions for them as possible. We have tried to achieve this by using as little controls as possible and by trying to have only one main control that we expect the user to interact with.

We hide all unnecessary information and choose a most likely default without asking users to choose from options that could confuse them, or would require them to study to make informed decisions. We assume this will mean the users will be more confident as there will be fewer steps where they were not sure if they have made the right choice.

Since we are developing this application as a cross-platform desktop application available under a permissive open-source license, it should be usable by as many people as possible.

### 3.1.2 Localization

We have also made the application easily translatable and try to display data like dates in the local format. When the application is started, we try to detect the locale of the user and set it as a default language. If we do not yet support the locale of

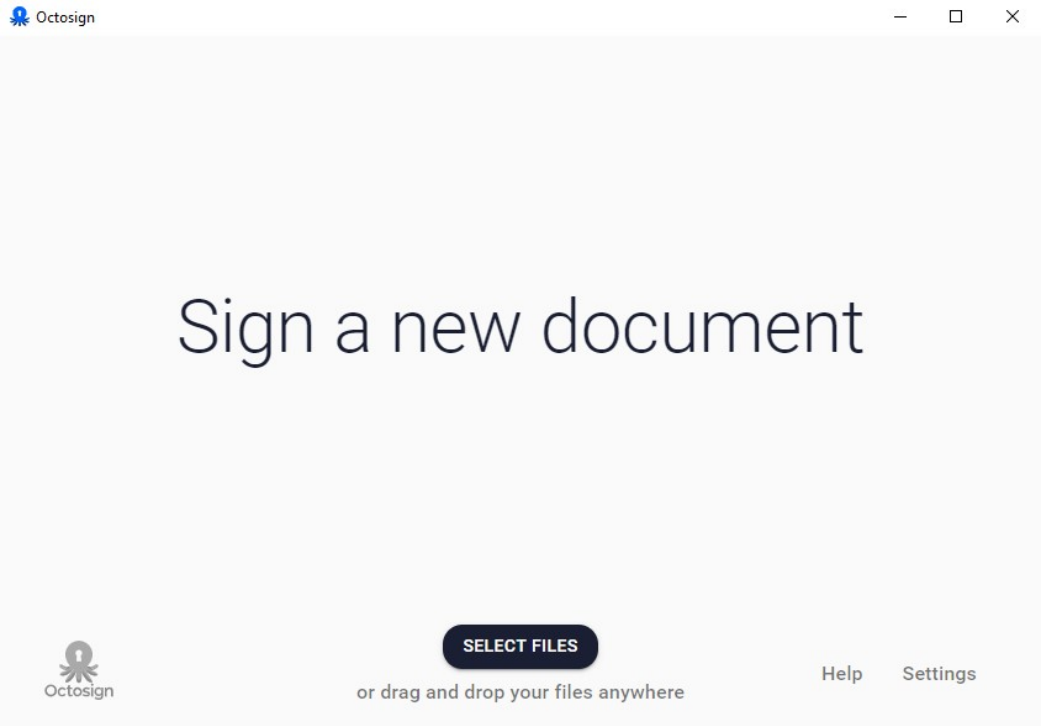


Figure 3.10: Screenshot from the main screen of our application Octosign.

the user, we default to the English language. At the time of writing, our application is fully translated in three languages: English (source language), Slovak, and Czech. Additional languages - German and Spanish - are actively being developed.

The process of content translation is mostly automated. We start with the automated extraction of the source strings from the code of the main module and files provided by the backends. Next, we manually include the extracted source strings in the commit to our versioning control - we have retained manual control over this to prevent a noise that automated commits could make. On push to the hosted repository, a localization platform Transifex[1] automatically pulls and process the updates. These phrases can then be translated by anybody using the *User Interface* (UI) of the Transifex platform to make it easily accessible. Once the changes are made on the Transifex platform, it automatically creates a pull request on our hosted repository with the changes made to that particular language. As we explain in detail in Section 3.1.6, part of the build process is to incorporate these in the application.

[1]Transifex is a SaaS localization platform free for OSS, see `https://www.transifex.com/`.

### 3.1.3 Modularity

Although we focus on the use with the SR eID, we want the application to be extensible by the community and ready for experimentation with new ways of signing. That is why we have designed our application to be a fully modular platform.

The main module is an Electron[2] application written in TypeScript[3]. This module features simple UI and abstraction around modules for signing that we call *backends*. It handles all of the communication with the user for the backend and tries to simplify development of the backends as much as possible.

Signing modules - backends - are separate command line interface (CLI) applications written in any programming language. They are responsible for the actual document manipulation - signing and verification.

The main module loads all available backends at runtime. Once the user decides to verify or sign a file, they are executed as CLI applications.

Communication between the main module and backends is in plaintext over the standard streams (STDIO) - standard input (STDIN), output (STDOUT), and error (STDERR). To make sure we have correctly documented and easily accessible way for backend development, we have prepared a Backend specification that specifies what is required of the backend, what is optionally supported, and the format of the inputs, outputs, and error handling.

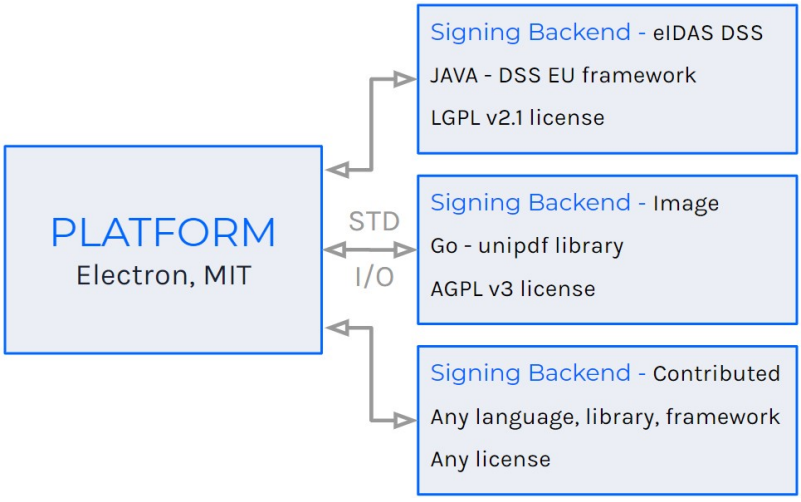


Figure 3.11: Modules of our application and their communication.

There are several advantages to this approach. Firstly, we can write backends in any language that suits us the most. For example, our backend that allows signing

[2]Electron is a framework allowing the development of desktop software using web technologies.
[3]TypeScript is statically typed superset of JavaScript that compiles to JavaScript.

documents with the signatures compliant with the EU eIDAS directive is written in Java because it is based on a library from EU - DSS - also written in Java. We expect most of the languages have APIs for the consumption of the standard streams with excellent portability across different desktop platforms. Secondly, since we are technically working on separate applications, they can be written by different people under a different license. That also means we can easily swap the main module or any of the backends for use in different environments or different bundles. Thirdly, together with the choice of using standard streams, it means that anyone can quickly start the development of a new backend. Writing to the standard output stream in CLI application is one of the first things developers usually learn - so-called "Hello World" applications. It usually should not involve studying any required third-party libraries, and authors can choose the language they are most familiar with.

### 3.1.4 Signing backends

Following our specification, each backend is required to provide a configuration file and follow the communication protocol.

Configuration files are written in YAML[4] and contain basic information about the backend like name, description, repository, version, author, or license. It also contains two properties defining what command should be executed and with what parameters to execute the backend, and build the backend, respectively.

Backend is always called with operation as the first argument, followed by an additional argument that differs between operations. The backend should be stateless - that means it should not be influenced by any state it would save like info about what files it signed. All communication is in UTF-8 encoded plaintext. It needs to implement operations `meta`, `sign`, and `verify` that are run using the configured executable, for example, `./backend-executable meta`, or, `./backend-executable sign /home/path.pdf`.

Operation meta is called to get info about its availability, options, and supported file types. This allows the backend to check the platform, available software of the user, etc. and make the decision based on that.

Operation sign and verify are called to sign or verify the file with the absolute path to the file. The main module never sends anything over the STDIO unless it's asked to do so from the backend. That means the backend acts as a `master`

[4]YAML is a human-readable data serialization language, see `https://yaml.org/`.

and the main module as a `slave`. The UI can end the process if any unexpected situation arises and displays an error. The backend can during those two operations, at any moment: write error to the STDERR that is immediately displayed; prompt for additional information by writing to the STDOUT; get the current value of any option by writing to the STDOUT.

Ending the process with the exit code 0 signals success. Some operations expect a result on the STDOUT before the process exits. If code other than 0 is used, the backend should always send an error message to the STDERR.

Messages exchanged via the STDIN and STDOUT can be sometimes polluted by different libraries used on either part, so they are distinguished from the rest by using the following delimiter:

```
---TYPE---
any information exchanged in the appropriate format
---TYPE---
```

`TYPE` here is replaced by `PROMPT`, `GETOPTION`, or `RESULT` based on the type of the message. Such messages should be written in one chunk and flushed. Communication on the STDERR is accepted and displayed as-is.

Prompt is used to interactively ask for more information from the user. The backend can ask to get an absolute path to the output file, absolute path to opened file, text, password, image (drawn or picked), or position and width of the signature that will be placed in the UI by the user.

Getting an option is used to ask for the current option value that can be set in the UI on the Settings screen. The option is automatically prefixed on the main module for each backend so it can not retrieve options from other backends to prevent collisions and security issues.

Such communication protocol allows using any language that can simply work with the STDIO without a need for knowledge or third-party libraries to work with serialized data in formats like JSON. The backend does not have to use any regular expressions but always simply reads the whole line. It is supposed to be as simple as possible for newcomers.

The current version of the full specification is available at `https://github.com/durasj/octosign/wiki/Backend-specification`.

### DSS eIDAS backend

The backend that we call *advanced electronic signature*, is a Java CLI application that uses *Digital Signature Service* (DSS) framework. The DSS is a project from the Connecting Europe Facility (CEF) Digital, a European Union fund for pan-European infrastructure investment. It is one of the Building Blocks, as they are called on their website, which helps to create and verify electronic signatures in line with European standards. Its documentation, available in [21], also provides general information about the used standards.

Source of the private key is configurable, and we can use PKCS #11 to utilize those in HSMs, PKCS #12 to work with key stores in files, or Microsoft Crypto API that provides an abstraction on different sources. It supports all standards mentioned in Chapter 1.2.3 in various configurations. Since we are trying to simplify it down for the user, when signing, we automatically choose to use PAdES when signing PDF, XAdES when signing XML with enveloped signature, and ASiC with CAdES for all other file formats. This could be further extended in the future, so the user can manually choose the preferred standard, container type, or signature policy.

Signature verification, however, works with all different combinations that are detected automatically. During the verification, we construct a trusted list of root certificates from the LOTL. Different lists for different countries are hosted individually on each countries servers. For better performance and reliability when there is an outage on those servers, we proxy requests through CloudFlare[5] *Content Delivery Network* (CDN) to distribute cached files that are periodically downloaded and stored on the Amazon Web Services[6] (AWS) infrastructure using the AWS Lambda and AWS S3.

The backend currently supports configurable TSP URL, so any TSP following the RFC 3161 can be utilized as long as it does not require authentication. Authentication is usually done using a name and password or using a digital signature but is currently not supported. This could hinder use with qualified TSPs that usually require authentication.

One issue we were facing when working with the DSS framework is the assumption that it is often implemented on the server-side as a service. This meant that some APIs are meant to be used in an environment where they are started once, and then

[5]CloudFlare is a CDN provider, see `https://www.cloudflare.com/`.

[6]Amazon Web Services is a cloud provider with various services, see `https://aws.amazon.com/what-is-aws/`.

they are available or can be run periodically several times a day. This is obviously not the case with our architecture since we always run it only when some operation is required, and the application is stopped again. We assume this issue can come up during the development of other backends as well and may require a certain level of flexibility from the developer to tackle them.

#### Image backend

To provide users with the ability to sign using a simple image saved on the computer, which is probably a scanned signature, or signature drawn directly in our application, we created what we call *simple image signature* backend. It is useful when users do not have the resources required to sign the document using the QES, or in situations when they would rather not do so.

It is written in Go and utilizes a library for manipulation of PDF files. Its purpose is just to ask the main module for the signature file and its position. Therefore, this backend utilizes prompt functionality of the communication protocol with options image and position.

Option to draw the signature in the application is using a splining algorithm with a variable stroke that calculates the time it takes to travel between two points. Such signatures mimic one of the features of the handwritten signatures from [22] that are used to verify the authenticity of handwritten signatures. That being said, we are not aware of any proof that this approach has any practical benefits outside of aesthetics. This approach was outlined on the Square Engineering Blog[7]. There is a JavaScript implementation for the browser called Signature Pad[8].

### 3.1.5 Architecture of the main module

Our main module is the most complex one. Electron applications have a main thread that is running a Node.js[9] and a UI thread running the relevant part of the Chromium[10]. Their communication is done via *inter-process communication* (IPC) and allows access to the Node.js APIs in the UI thread. Considering the JavaScript is

[7]Smoother Signatures article available archived at `https://web.archive.org/web/20150211220342/http://corner.squareup.com/2012/07/smoother-signatures.html`.

[8]Signature Pad library is available at `https://github.com/szimek/signature_pad`.

[9]Node.js is a JavaScript runtime for use outside of the browser with easier access to the OS APIs.

[10]Chromium is a popular open-source browser project, see `https://www.chromium.org/Home`.

interpreted at runtime, we are exposing ourselves to potential attacks (mostly XSS[11]). To reduce the attack surface, as Node.js allows easy access to many APIs like full access to the filesystem, we have fully separated the UI thread and main thread. Our work was influenced by knowledge from [19] and [20].

The main thread is focused only on the communication with the backends and provides the only absolutely necessary functionality for the UI. They are connected by a very thin level of abstraction - a set of functions that are bound on the *window* object during the initialization by the main thread and, when called from the UI thread, automatically communicate via IPC. All of the UI code and main thread code is, therefore, also fully portable and could also be used in a different setup - for example, in a web application or mobile application. We would be mostly required to only change the thin level of abstraction - implement the interface we have created on the *window* object. Furthermore, it allows us to develop the UI in the browser with a mocked main thread and also allows for easy UI testing, which we go over in Section 3.1.6.

### 3.1.6 Testing and automation

When approaching automated testing, we have decided to implement the testing pyramid as defined by Mike Cohn in [15]. One of the implementations of this idea was done by the Ham Vocke in [16]. It is implemented at a scale at Google, which explains their positive experience with it in [17]. Following this approach, we write three types of tests: end-to-end, integration, and unit tests. The ratio of these types of tests should form a pyramid where the heavy bottom is formed by the unit tests, followed by a smaller set of component tests forming the middle of the pyramid, and a small set of end-to-end tests forming the tip of the pyramid. The motivation behind this approach is to write more tests that are faster and break less often due to better isolation, as opposed to tests that are slower and break more often.

In our case, we have decided to write unit tests in Jest testing both the UI and main thread code, component tests in Jest for the main thread and Cypress for the UI thread combined with the Percy, and end-to-end tests as Spectron tests which run and interact with the actual distributables (executable files). We are able to fully isolate each type of tests because of our architecture that allows easy mocking. We explain the architecture in Section 3.1.5.

[11] XSS means Cross-site scripting, see `https://owasp.org/www-community/attacks/xss/`.

Our code is kept in a git repository[12] hosted at Github and is a source for the Continuous Integration (CI) and Continuous Delivery (CD) pipeline. For the CI/CD, we are using Azure Pipelines specifically, that are free for OSS. On each push, apart from setting up the environment, we perform static analysis on the code checking the type safety using TypeScript, code style using the ESLint and Prettier, code quality using the CodeClimate.com, and check for potential vulnerabilities using the Snyk.io. As for the mentioned automated tests, we run the unit and components tests and monitor the coverage using the Codecov.io. In the case coverage would be significantly negatively impacted, the Pull Request (PR) is blocked from being merged. Backends are referenced in the same repository by using the git submodules, which allows us to simplify the development and build process. Build process build individual backends, the main thread, UI thread, and API code, and uploads bundled distributable files for all platforms back to the Github. The whole process runs for each platform - Windows, Linux, and macOS - separately; therefore, there is no need for cross-compiling and running of platform-dependent tools in virtualized environments. Publishing of the new release is then done by saving the release in the Github.

We had to overcome issues we have faced with the performance of the Github relases - mainly apparent bandwidth restrictions. We have deployed an AWS lambda that is called when a new release is published on Github and sends a message to AWS Simple Queue Service (SQS). Another AWS Lambda is triggered on such message, and it downloads all distributable files and creates a metafile with information about the release in AWS S3. These files are then served cached over the CDN from CloudFlare.

### 3.1.7 Comparison with existing software

Using our review from Chapter 2, we have compared our application in Table 2.1. Value ✓✓ means best-case scenario, no value worst-case scenario, and value ✓ somewhere in between - partially or depends.

Our software is available under a permissive license (see Section 1.1.3 for why this matters), supports all major desktop platforms, available in several locales, with support for practically all file types and sizes, with built-in verification and easy extensibility.

In the future, we can improve our application to be easier to set up for more users and be more flexible in the configuration of TSPs or used signature policy. That

[12]Repository is available at `https://github.com/durasj/octosign`.

| | licensing | cross-platform | localization | file suppport | easy setup | flexibility | built-in verification | extensibility |
|---|---|---|---|---|---|---|---|---|
| JSignPdf | ✓✓ | ✓ | | | | ✓✓ | | ✓ |
| zep.disig.sk | ✓ | ✓✓ | ✓✓ | ✓ | ✓✓ | | ✓✓ | |
| Acrobat DC | ✓ | ✓ | ✓✓ | | ✓ | ✓✓ | ✓✓ | |
| D.PDF Signer | | | ✓ | | ✓✓ | ✓ | | |
| Podpisuj.sk | ✓ | ✓✓ | ✓ | ✓✓ | ✓✓ | ✓ | ✓✓ | |
| Our - Octosign | ✓✓ | ✓✓ | ✓✓ | ✓✓ | ✓ | ✓ | ✓✓ | ✓✓ |

Table 3.1: Comparison of existing software with our software.

being said, we need to consider each configurable functionality we add as it increases complexity which could mean our software is harder to understand.

## 3.2 Website

Goal of the website is to both support the application in the form of Download page and Help page and provide a to provide a general information on electronic signatures. The website is, at the time of writing, available in English and Slovak language.

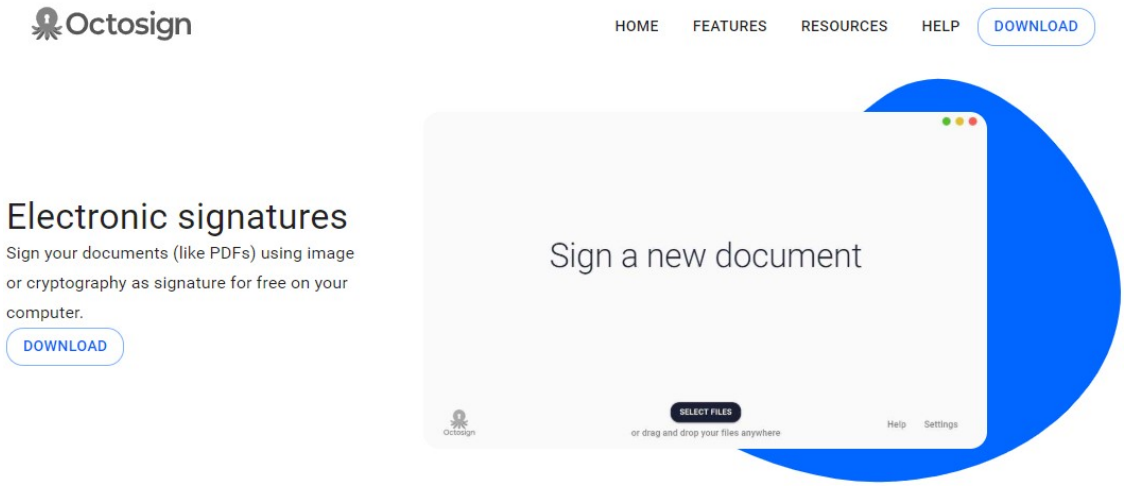


Figure 3.12: Home page of the octosign.com website.

### 3.2.1 Downloads

Downloads are available for each major desktop platform in a format suitable for installation or integration in the operating system. For Windows, we are distributing an `EXE` installer working on all supported 64-bit Windows versions. For Linux, a portable `AppImage` and a `DEB` package for Debian-like distributions. For macOS, a `DMG` disk image.

All of the distributables originate from the Github as assets of a release, where they are uploaded by the CI (see Section 3.1.6). Each release is tied to a tag on the master branch, where each commit has to be signed using a recognized PGP key. In turn, we have some level of confidence source code used for the distributables comes from one of the authors.

To improve on this further, we tried to do a *code-signing* for each release using a trusted way for each platform. On macOS, this means enrolling in the Apple Developer Program that requires an annual fee and installation of signing certificates. Additionally, it is necessary to *notarize* the application for use on macOS. Notarization is an automated process required by Apple to prove that the application does not contain any malicious code and is properly signed and configured. If the application is not notarized, Gatekeeper will warn the user about the potentially unsafe software. On modern versions of Windows, applications that are not code-signed can trigger a warning or, depending on the settings, even completely prevent the user from using software that is not code-signed using a certificate that is cosigned with a trusted root certificate. Certification for individuals on the macOS seems to be fully automated, does not involve lengthy verification, and costs 99 USD a year. Certification on Windows is offered by various trusted third parties like DigiCert.com that we have used and involves lengthy, partially automated verification at a price starting from 60 USD a year. At the time of writing, there does not seem to be any service offering code-signing certificates to OSS for free. We were not able to finish the process on time for the publishing of this thesis - we feel like this process could be improved in the future for OSS.

### 3.2.2 Information and help

Our website provides a simple help page with different use cases that should allow users to start using the application quickly. It also contains a brief explanation of more advanced features of the application like settings.

Another issue we try to solve with the website is a lack of understanding of electronic signatures. Some users do not know if they are a viable alternative, why is that the case, and how they should be used properly. We also plan to provide more detailed information for people who are curious about the specifics of the signatures used in the eIDAS scheme.

These pages are written in markdown - a lightweight markup language, making it more accessible for editing by the community that does not need to understand specific technologies used to display content on the website.

### 3.2.3 Technologies

From the technological point of view, the website is a *Single Page Application* (SPA) using Gatsby written in React. Even though it is a SPA, it is *prerendered* to static HTML and CSS files that are served from CDN, which improves performance and allows for better *Search Engine Optimization* (SEO).

The website is using CI/CD services from Netlify.com that is also responsible for distribution using a CDN. Each push on the master branch triggers a build and deploy on the live website. Each PR has a deploy preview that is basically a unique link that shows proposed changes built and deployed on the web. Such approach allows for easy iteration and allows others to contribute the content in markdown files without a need to build the website locally as they can see results as soon as they open the PR.

Different language mutations are denoted by using a suffix, for example, the home page is written in English and Slovak using file names `home.en.tsx` and `home.sk.tsx`. Same applies for content written in markdown, for example, help page on basic usage is written in files `basic.en.md` and `basic.sk.md`. The URL of the website contains language code of the current language if it is other than English, for example, the download page is available in English on path `/download` and in Slovak on path `/sk/download`. By switching the language using the selector on the upper right, the URL changes to the required format.

# Conclusion

We verified that the main legal and technical preconditions should be met for the use of electronic signatures. We therefore believe they are a viable alternative to handwritten signatures.

We did a review of the existing software, focusing on licensing, cross-platform compatibility, localization, file support, ease of setup, flexibility, verification ability, and extensibility. We identified the positive and negative sides of the current implementations and found a gap in the software for signing using qualified electronic signatures.

Therefore, we proposed a simple open-source software platform that aims to be straightforward for the end-user by choosing the most likely defaults and reducing the amount of information the user has to interact with. Also, we made our application easy to translate using a third-party web UI where anyone from the community can contribute translations. Our application is available in three languages: English, Slovak, and Czech, with German and Spanish in translation. It is also fully modular, dividing the main module that contains the UI and modules performing the document manipulation. We implemented a module for signing using qualified electronic signatures that follow European standards and should be compliant with European law, and a module that allows signing using a scanned or drawn signature. The proposed software is a platform on which others can quickly build new ways of signing documents complying with different laws or basing further research. From a software engineering perspective, the implementation is robust, and we covered testing and automation in connection with Electron applications like ours. On the other hand, there is some room for improvement in flexibility and ease of setup of our software. We also created a supporting website that provides download links for all major desktop platforms, a help page, and general information on electronic signatures. Our project should be open to people from different countries, speaking different languages, and open for contributions from people with various skills - not just from programmers.

# Resumé

Overili sme, že hlavné predpoklady pre použitie elektronických podpisov by mali byť naplnené. Veríme, že sú preto schodnou alternatívou pre vlastnoručné podpisy.

Preskúmali sme existujúci softvér so zameraním na licenciu, kompatibilitu medzi platformami, lokalizáciu, podporu súborov, ľahkosť nastavenia, flexibilitu, schopnosť overiť podpisy a rozšíriteľnosť. Identifikovali sme pozitívne a negatívne stránky súčasných implementácií a našli sme medzeru v softvéri na podpisovanie pomocou kvalifikovaných elektronických podpisov.

Preto sme navrhli jednoduchú otvorenú softvérovú platformu, ktorá sa snaží byť priamočiara pre koncového používateľa výberom pravdepodobných predvolených hodnôt a znížením množstva informácií, s ktorými sa používateľ stretáva. Taktiež sme uľahčili preklad našej aplikácie pomocou webového rozhrania tretej strany, s ktorým môže ktokoľvek z komunity pomôcť prekladmi. Naša aplikácia je dostupná v troch jazykoch: angličtine, slovenčine a češtine s nemčinou a španielčinou v prekladaní. Je tiež plne modulárna, kde hlavný modul obsahuje používateľské rozhranie a niekoľko modulov vykonáva manipuláciu s dokumentmi. Implementovali sme modul na podpisovanie pomocou kvalifikovaného elektronického podpisu, ktorý spĺňa európske štandardy a mal by byť v súlade s európskymi zákonmi a modul, ktorý umožňuje podpisovanie pomocou naskenovaného alebo nakresleného podpisu. Navrhovaný softvér je platforma, na ktorej môžu ostatní rýchlo vybudovať nové spôsoby podpisovania dokumentov v súlade s rôznymi zákonmi alebo založiť ďalší výskum. Venovali sme sa testovaniu a automatizácii vývoja Electron aplikácií, takže implementácia je robustná aj z pohľadu softvérového inžinierstva. Na druhej strane existuje priestor na zlepšenie flexibility a jednoduchosti prvotného nastavenia nášho softvéru. Vytvorili sme tiež podpornú webovú stránku s možnosťou stiahnutia pre hlavné počítačové operačné systémy, užívateľskou príručkou a všeobecnými informáciami o elektronických podpisoch. Náš projekt je otvorený ľuďom z rôznych krajín, rozprávajúcich rôznymi jazkymi a zmenám od ľudí s rôznymi zručnosťami - nie len programátorom.

# Bibliography


[1] *Act No. 40/1964 Coll. Civil Code*

[2] *Act No. 513/1991 Coll. Commercial code*

[3] *Act No. 323/1992 Coll. Notary Law*

[4] *REGULATION (EU) No 910/2014 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL, OJ L 257, 28.8.2014, p. 73–114*

[5] *Council Directive 91/250/EEC of 14 May 1991 on the legal protection of computer programs, OJ L 122, 17.5.1991, p. 42–46*

[6] MASON, Stephen, 2017. *Electronic Signatures in Law: Fourth Edition*. London: University of London. ISBN 978-1-911507-01-7. Available at: `https://humanities-digital-library.org/index.php/hdl/catalog/view/electronicsignatures/1/86-1`

[7] LAURENT, Andrew, 2008. *Understanding Open Source and Free Software Licensing*. Sebastopol: O'Reilly Media. ISBN 978-0596005818.

[8] PAAR, Christof and Pelzl, Jan, 2009. *Understanding Cryptography - A Textbook for Students and Practitioner*. Berlin: Springer. ISBN 978-3-642-04100-6.

[9] ROSULEK, Mike. *The Joy of Cryptography* [online]. Oregon: School of Electrical Engineering & Computer Science, Corvallis, Oregon, USA [cit. 2020-04-18]. Available at: `http://web.engr.oregonstate.edu/rosulekm/crypto/crypto.pdf`

[10] RIVEST, R. L., SHAMIR, A., and ADLEMAN, L., 1978. *A method for obtaining digital signatures and public-key cryptosystems*. In: *Communications of the ACM, 21(2):120–126, February 1978.*

[11] ETSI TS 102 778-1 V1.1.1 2007-07. *PAdES Overview - a framework document for PAdES*. Sophia Antipolis: European Telecommunications Standards Institute. Available at: https://www.etsi.org/deliver/etsi_ts/102700_102799/10277801/01.01.01_60/ts_10277801v010101p.pdf

[12] ETSI TS 101 903 V1.4.2 2010-12. *XML Advanced Electronic Signatures (XAdES)*. Sophia Antipolis: European Telecommunications Standards Institute. Available at: https://www.etsi.org/deliver/etsi_ts/101900_101999/101903/01.04.02_60/ts_101903v010402p.pdf

[13] ETSI TS 101 733 V2.2.1 2013-04. *CMS Advanced Electronic Signatures (CAdES)*. Sophia Antipolis: European Telecommunications Standards Institute. Available at: https://www.etsi.org/deliver/etsi_ts/101700_101799/101733/02.02.01_60/ts_101733v020201p.pdf

[14] SOMMERVILLE, Ian, 2016. *Software Engineering, 10th Edition*. Harlow: Pearson Education. ISBN 978-0-13-394303-0.

[15] Cohn, Mike, 2009. *Succeeding with Agile*. Boston: Addison-Wesley Professional. ISBN 978-0-32-166053-4.

[16] Vocke, Ham, 2018. *The Practical Test Pyramid* [online]. Available at https://martinfowler.com/articles/practical-test-pyramid.html

[17] Wacker, Mike, 2015. *Just Say No to More End-to-End Tests* [online]. Available at https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html

[18] FOGEL, Karl, 2019. *Producing Open Source Software* [online]. [cit. 2020-04-21]. Available at: https://producingoss.com/en/index.html

[19] HAVERBEKE, Marijn, 2018. *Eloquent JavaScript, 3rd Edition*. San Francisco: No Starch Press. ISBN 978-1593279509.

[20] SIMPSON, Kyle, 2020. *You Don't Know JS Yet (book series) - 2nd Edition* [online]. Available at: https://github.com/getify/You-Dont-Know-JS

[21] Vandenbroucke, P. and Beliakov, A. [online]. *Digital Signature Service*. Luxembourg: CEF Digital [cit. 2020-05-16]. Available at: https://ec.europa.eu/cefdigital/DSS/webapp-demo/doc/dss-documentation.html

[22] Hafemann, G. L., Sabourin, R., and Oliveira, S. L., 2017. Offline Handwritten Signature Verification - Literature Review. In: *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pp. 1-8. Montreal: IEEE. ISBN 978-1-5386-1842-4. Available at: `https://arxiv.org/abs/1507.07909`

# Appendices

An up-to-date version of each developed asset is available online on the URL specified in the list. The version available at the time of writing is part of the attached CD, and the user manual is also available as Appendix A.

List of the most important assets available online and on the CD:

- Software Octosign - repository with source code: `https://github.com/durasj/octosign`.
- Software Octosign - released distributable files: `https://github.com/durasj/octosign/releases`.
- Signing backend DSS - repository with source code: `https://github.com/durasj/octosign-dss`.
- Signing backend Image - repository with source code: `https://github.com/durasj/octosign-image`.
- Website octosign.com - repository with source code: `https://github.com/durasj/octosign-website`.
- Documentation - short user manual: `https://octosign.com/help/basics`.
- Documentation - signing backend specification: `https://github.com/durasj/octosign/wiki/Backend-specification`.

# A User manual

## A.1 Downloading and installing the application

Open the subpage *Download* on website `https://octosign.com` where you can download the application by clicking on the button for your operating system. After the download is complete, open the file and follow the instructions.

## A.2 Signing the document

First, it is needed to select a document. To select a document, click on the button *SELECT FILES* or drag and drop your document on the application. The recommended document type is PDF, although other file types are supported as well. After selecting the document, a card with the document name will appear just like in the picture.

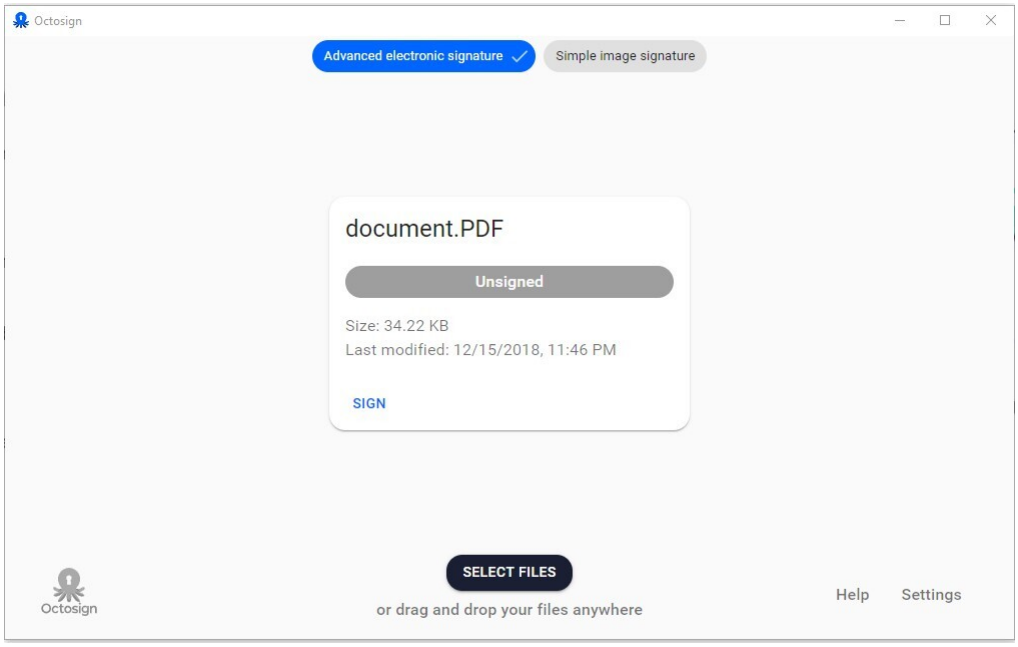


Figure A.13: Screenshot with unsigned document.

Signing is possible using several different types of signatures visible in the upper part of the screen. The default type is *Advanced electronic signature*, if available, else *Simple image signature*. When signing with an image, there is only a drawn or picked signature image placed on the PDF. When signing with an advanced electronic signature, it is possible to use a qualified electronic signature. To sign a document, press the button *SIGN* visible on the document card. Follow the instructions, and after the signing is successful, the document will be marked as *Signed*.

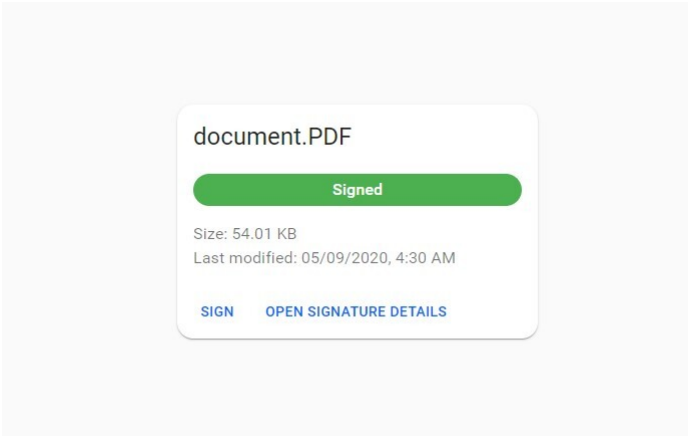


Figure A.14: Card with the signed document.

## A.3 Verifying of document signature

Verification of the document signature is automatically started after the document is selected, and it does not have to be triggered manually.

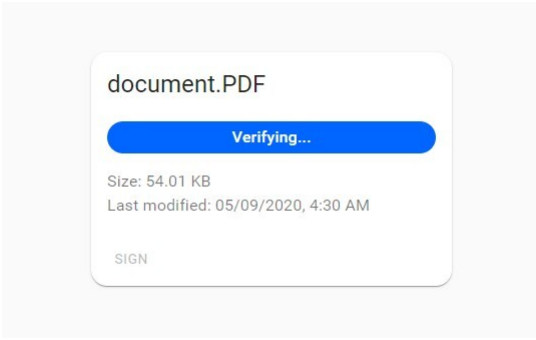


Figure A.15: Card with the document that is being verified.

After the verification is completed, the status of the document will be visible on the card of the document, it can be: *Signed*, *Unsigned*, *Invalid*, *Unknown*, and *Indeterminate*.

If there is at least one signature attached to the document, it is possible to see more information about it by clicking on the button *OPEN SIGNATURE DETAILS*.

## A.4 Settings

Settings can be opened by clicking on the text *Settings* in the lower right corner of the application.

It is possible to set the language and options specific for the type of the signature.

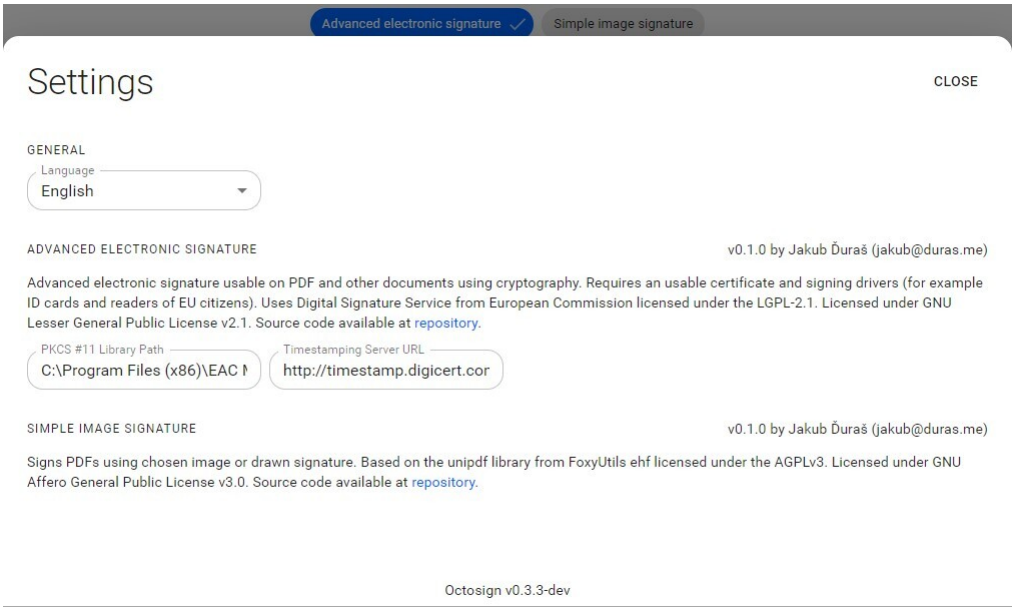


Figure A.16: Screenshot with open Settings.

Options available for the Advanced electronic signature backend are:

**PKCS #11 Library Path** - The path is automatically prefilled if there is a supported known software installed on the computer. If not, it is possible to fill it manually with the path that you can get from the supplier of the software bundled with your device. The path needs to lead to a library compliant with the PKCS #11 standard in a 64-bit version.

**Timestamping Server URL** - The URL is automatically prefilled with value "http://timestamp.digicert.com". In the case you would like to use your own timestamping server, feel free to change it, but please make sure to type in the full URL address, including the protocol ("http://" or "https://").