UNIVERSITY OF TARTU
Institute of Computer Science

Cyber Security Curriculum

Burak Can Kus

USE OF ELECTRONIC IDENTITY DOCUMENTS FOR MULTI-FACTOR AUTHENTICATION

Master's Thesis (24 ECTS)

Supervisor
Arnis Paršovs
PhD

Tartu 2021

Use of Electronic Identity Documents for Multi-Factor Authentication

Abstract:

This work introduces an open-source automated biometric authentication system, "eMRTD Face Access." It uses an Electronic Machine Readable Travel Document (eMRTD) and a facial image to authenticate a person. The solution provides two-factor authentication. The authentication factor "something you have" is implemented by performing cryptographic checks to verify the authenticity of an eMRTD, but the authentication factor "something you are" is verified by comparing the facial image of the person presenting the document with the facial image stored in the document. The solution has been successfully tested on Estonian, Latvian, and Turkish identity documents and is also expected to work on documents issued by other countries.

Keywords:
Machine Readable Travel Documents (MRTDs), Electronic Machine Readable Travel Documents (eMRTDs), ePassports, automated border control, multi-factor authentication, facial recognition, optical character recognition (OCR)

CERCS: P170 – Computer science, numerical analysis, systems, control

Elektrooniliste isikut tõendavate dokumentide kasutamine mitmeteguriliseks autentimiseks

Lühikokkuvõte:

Selles töös tutvustatakse avatud lähtekoodiga automatiseeritud biomeetrilise autentimise süsteemi “eMRTD Face Access”. Süsteem kasutab inimese autentimiseks elektroonilist masinloetavat reisidokumenti (eMRTD) ja näokujutist. Lahendus pakub kaheastmelist autentimist. Autentimistegurit “midagi, mis teil on” rakendatakse krüptograafiliste kontrollide abil, et verifitseerida eMRTD autentsust, ning autentimistegurit “midagi, mis te olete” kontrollitakse dokumenti esitava isiku näokujutise ja dokumenti salvestatud näokujutise võrdlusega. Lahendust on edukalt testitud Eesti, Läti ja Türgi isikut tõendavatel dokumentidel ning eeldatavasti töötab see ka teiste riikide väljastatud dokumentidega.

Võtmesõnad:
Masinloetavad Reisidokumendid, Elektroonilised Masinloetavad Reisidokumendid, Elektroonilised Passid, automatiseeritud piirikontroll, mitmeteguriline autentimine, näotuvastus, optiline märkide tuvastamine

CERCS: P170 – Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

List of Figures

List of Abbreviations and Terms

ABC Automated Border Control.

AID Application Identifier.

APDU Application Protocol Data Unit.

API Application Programming Interface.

ATR Answer to Reset.

BAC Basic Access Control.

BSI German Federal Office for Information Security.

C_{CSCA} Self-Signed CSCA Certificate.

C_{CVCA} Self-Signed CVCA Certificate.

C_{DLS} Deviation List Signer Certificate.

C_{DS} Document Signer Certificate.

C_{DV} Document Verifier Certificate.

C_{MS} Master List Signer Certificate.

CAN Card Access Number.

CNN Convolutional Neural Network.

CRL Certificate Revocation List.

CSCA Country Signing Certification Authority.

CVCA Country Verifying Certification Authority.

DH Diffie–Hellman Key Exchange.

DL Deviation List.

Doc 9303 Document Series 9303.

ECDH Elliptic-Curve Diffie–Hellman.

ECDSA Elliptic-Curve Digital Signature Algorithm.

eMRP Electronic Machine Readable Passport.

eMRTD Electronic Machine Readable Travel Document.

EOL End-of-life.

GNU GNU's Not Unix!.

GPL GNU General Public License.

GUI Graphical User Interface.

HOG Histogram of Oriented Gradients.

ICAO International Civil Aviation Organization.

IED Inter Eye Distance.

JMRTD Java Implementation of Machine Readable Travel Document.

JSON JavaScript Object Notation.

$\mathbf{K_{ENC}}$ Encryption Document Basic Access Key.

$\mathbf{K_{MAC}}$ MAC Document Basic Access Key.

KDF Key Derivation Function.

$\mathbf{KS_{ENC}}$ Encryption Session Key.

$\mathbf{KS_{MAC}}$ MAC Session Key.

LDIF LDAP Data Interchange Format.

LDS Logical Data Structure.

LGPL GNU Lesser General Public License.

LSTM Long Short-Term Memory.

MAC Message Authentication Code.

MIT Massachusetts Institute of Technology.

ML Master List.

MRTD Machine Readable Travel Document.

MRV Machine Readable Visa.

MRZ Machine Readable Zone.

OCR Optical Character Recognition.

OID Object Identifier.

PACE Password Authenticated Connection Establishment.

PKD Public Key Directory.

PKI Public Key Infrastructure.

RFID Radio-Frequency Identification.

RSA Rivest-–Shamir—Adleman.

SHA-1 Secure Hash Algorithm 1.

SM Secure Messaging.

SO_D Document Security Object.

SSC Send Sequence Counter.

TD1 Size 1 Machine Readable Official Travel Document.

TD2 Size 2 Machine Readable Official Travel Document.

TD3 Size 3 Machine Readable Travel Document.

VIZ Visual Inspection Zone.

1. Introduction

As the technology progresses, so does the need for trustworthy authentication means. With the growing population of smart card technologies and electronic identity documents worldwide, solutions offering accurate document and cardholder verification are crucial.

This work introduces and describes an open-source automated biometric system, developed by the author of this thesis, that performs multi-factor authentication. The solution is called "eMRTD Face Access" and is available on GitHub[1]. It uses an Electronic Machine Readable Travel Document (eMRTD) and a facial image to authenticate a person. First, it communicates with the chip contained in a document and reads the facial image. It then takes an image of the individual that presents the document. Next, facial recognition technologies are used to match these two images. Finally, if the document is verified to be valid and not cloned, and the facial images match, the individual is granted access.

This solution has been tested using Estonian *residence permit cards*, Turkish *identity cards* and a *passport*, and a Latvian *identity card* and a *passport*. We expect the solution to work with eMRTDs issued by other countries as well. It can be used as an Automated Border Control (ABC) system by authorities or be deployed on the entrances to buildings or self-checkout machines where strong identity verification is needed.

The thesis is structured as follows. The sections in Chapter 1 introduce the travel document ecosystem and briefly introduce the solution created during this work and related works. The sections in Chapter 2 give information about the implementation details of this solution. Finally, Chapter 3 concludes the work by giving a summary and suggestions for future work.

1.1 Electronic Machine Readable Travel Documents (eMRTDs)

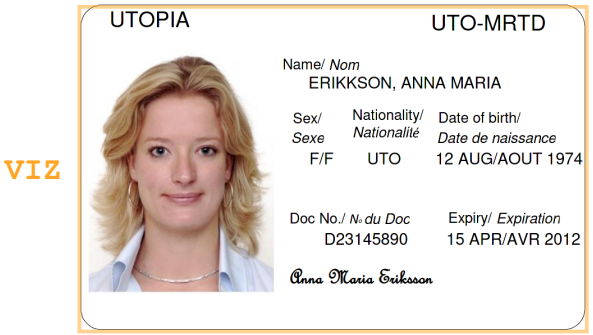
An *identity document* is a document that proves the identity of a person. These documents can be *identity cards*, *passports*, or other types of documents. *Machine Readable Travel Documents (MRTDs)* are official documents issued by a State[2] or an organization and are

[1] `https://github.com/Fethbita/eMRTD_face_access`

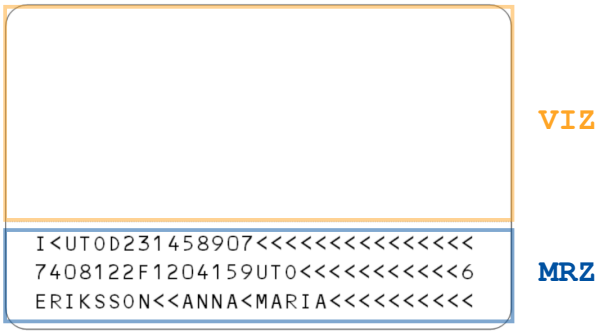
[2] Throughout this work, the capitalized word State refers to countries.

used for travel. They were introduced by the International Civil Aviation Organization (ICAO) to increase the clearance speed of passengers through passport control points [1].

The ICAO specifies three different sized MRTDs. These are *Size 1 Machine Readable Official Travel Documents (TD1)*, *Size 2 Machine Readable Official Travel Documents (TD2)*, and *Size 3 Machine Readable Travel Documents (TD3)* [1]. TD1 and TD2 size documents are usually in the form of a card, and TD3 size documents are usually passport books that contain pages [1]. All three sized MRTDs contain two mandatory zones. The first one is the Visual Inspection Zone (VIZ), which includes information about the document holder and is eye-readable [1]. The other zone is the Machine Readable Zone (MRZ) which summarizes the information contained in VIZ. TD1 size documents and their VIZ and MRZ can be seen in Figures 1a and 1b, front and back, respectively, TD2 size documents and their VIZ and MRZ can be seen in Figure 1c, and TD3 size documents and their VIZ and MRZ can be seen in Figure 1d.

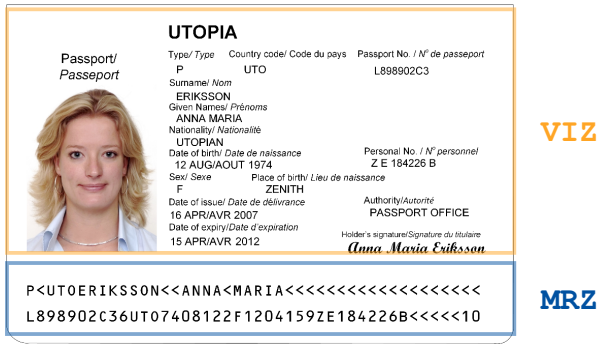


(a) Front side of a TD1 size MRTD (not to scale) [2]



(b) Reverse side of a TD1 size MRTD (not to scale) [2]

(c) Layout of a TD2 Size MRTD (not to scale) [3]



(d) Layout of a TD3 size MRTD data page (not to scale) [4]

Figure 1. Different size MRTDs

In addition, MRZ includes a check digit for data elements such as the document number, date of birth, and date of expiry. The check digits depend on the values of these data elements and determine if the MRZ is scanned correctly. A final check is calculated on the earlier mentioned information with their check digits included and optional data elements

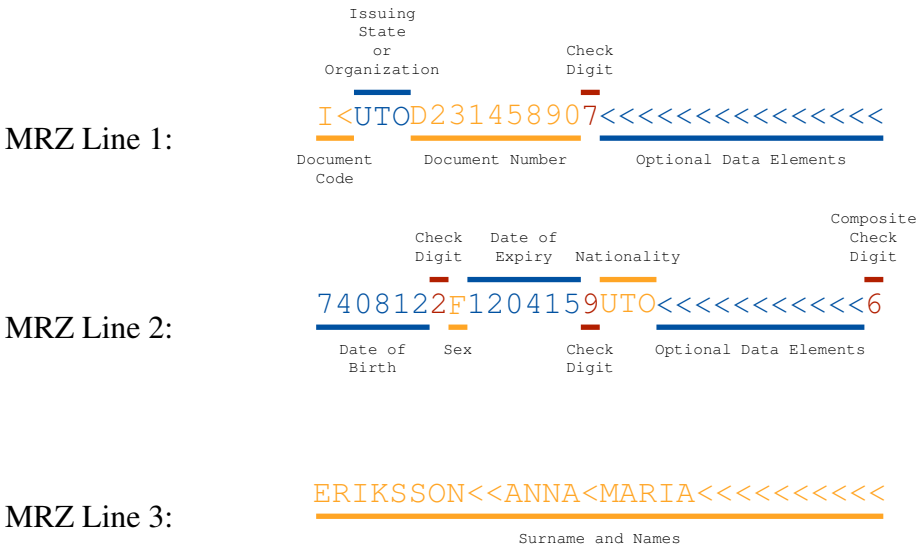


Figure 2. TD1 form Machine Readable Zone elements

and is placed at the end of the MRZ for TD2 and TD3 size documents and at the end of the second line for TD1 size documents. For example, the information in a TD1 size document MRZ is shown in Figure 2.

Electronic Machine Readable Travel Documents (eMRTDs) are Machine Readable Travel Documents (MRTDs) that contain an embedded contactless smart card chip. In 1980, ICAO published the first edition of Document Series 9303 (Doc 9303) to standardize MRTDs [1]. In 2006, the technical reports for eMRTDs were incorporated into Doc 9303 to create an effective biometric identification system [1]. Even though Doc 9303 does not standardize national identity documents, many countries already issue *identity cards* or *residence permit cards* with eMRTD applets. For example, Estonia started issuing *residence permit cards* with eMRTD applets starting from January 1, 2011 [5]. Also, the European Union requires all member states to issue *identity cards* that are valid eMRTDs, starting from August 2, 2021 [6].

The specifications for eMRTDs contain information about the Logical Data Structure (LDS), which describes how the data are stored and formatted in the contactless smart card chip and the Public Key Infrastructure (PKI) used by eMRTDs. Currently, ICAO Doc 9303 part 10 [7] contains the specifications regarding the LDS, and ICAO Doc 9303 part 12 [8] contains the specifications regarding the PKI.

In the PKI used for eMRTDs, a single Certificate Authority called Country Signing Certification Authority (CSCA) is run by each State, and it issues the certificates for that State. In addition, CSCAs have Self-Signed Root CSCA Certificates (C_{CSCA}s) that issue the so-called Document Signer Certificates (C_{DS}s). The C_{DS}s sign the Document Security

Object (SO_D) for each document issued by that State. The signed SO_Ds store the digests[3] of files included in the contactless smart card chips in eMRTDs, in a file called `EF.SOD`.

The LDS contains the data printed on the eMRTD, a facial image, and optionally fingerprints, iris scans, displayed portraits, displayed signature, additional personal details, additional document details, optional details, security options, public keys used for security methods, and name and contact details of the person(s) to notify. In addition, it is digitally signed to prove the authenticity of this data. eMRTDs contain four required files and might contain other optional files. The first required file is the abovementioned `EF.SOD`. The second one is `EF.COM` which contains the LDS version, the Unicode version, and a list of files present on the document. However, the digest of the `EF.COM` file is not included in the `EF.SOD` file. The other two files are called `EF.DG1` and `EF.DG2`. The file `EF.DG1` stores the same information on the MRZ, and the file `EF.DG2` stores facial image(s). These files and other optional files are placed in the documents during the personalization phase. After the personalization of a document, the chip has to be locked, and this means that no further change can be made in these files; no data can be written, modified, or deleted, and a locked chip cannot be unlocked [9].

Verifying the data in eMRTDs is done by first verifying the SO_D using the public key of the C_{DS} and verifying the C_{DS} using the public key of the C_{CSCA}. Next, as the files are read, the digest values should be calculated and must be compared with the digest values stored in the SO_D. This process ensures that the contents of these files are authentic (not modified). This process is called *Passive Authentication*. ICAO Doc 9303 part 11 [10] contains the specifications regarding the security mechanisms used in eMRTDs, including Passive Authentication.

In addition, there are advanced security features that provide other benefits. The first one is called *Chip Access Control*. Without this feature, an attacker can execute two types of attacks. First, they can read the files inside the document just by being nearby, and this type of attack is called skimming, and also, they can eavesdrop on the communication between the chip and an inspection system. This security feature prevents skimming and eavesdropping by requiring information printed on the document from the inspection system to create an encrypted Secure Messaging (SM) channel for communication between the chip and the inspection system. Until the ICAO Doc 9303 7th edition, the specifications used to allow chips that would not implement any access control, but since the 8th edition draft (dated January 8, 2021), chips that do not implement any access control are not allowed [11]. The SM channel is created by establishing two session keys and a Send Sequence Counter (SSC). The first session key, Encryption Session Key (KS_{ENC}), is used to

[3]The term *digest* is used to describe the cryptographic hash value.

encrypt and decrypt the Application Protocol Data Unit (APDU) sent between the chip and the inspection system, and the second session key, MAC Session Key (KS_{MAC}), is used to calculate the Message Authentication Code (MAC) of the APDU. The purpose of the MAC is to verify the integrity and the authenticity of the received APDU. The SSC is a counter that is incremented each time an APDU is sent and is used to prevent APDU reordering and replay attacks. For eMRTDs, there are two access control methods described below.

Basic Access Control (BAC) is used to create two session keys from the information taken from the MRZ using symmetric-key cryptography. However, since symmetric-key cryptography is used to derive the session keys, the entropy of the created keys depends on the input length. As for the input, all three inputs have shortcomings. For example, the document numbers are generally sequential, the date of expiry can be assumed at most ten years, and the date of birth can be estimated or assumed less than a hundred years. This makes the entropy of the session keys low by today's standards[12], and under certain circumstances, these encrypted channels are shown to be cracked in under 30 seconds [13].

Password Authenticated Connection Establishment (PACE) was introduced in 2006 by German Federal Office for Information Security (BSI) to overcome the shortcomings of BAC [14]. PACE uses public-key cryptography to derive the session keys, and the entropy of the session keys does not depend on the input password. Therefore, for the PACE protocol, a six-digit number called Card Access Number (CAN) can be printed on the document and used as a password to establish an SM channel with the eMRTD. An example of this number is shown in Figure 3 on an Estonian *residence permit card* sample. In 2011, it was decided that by December 31, 2014, all European Union Member States have to implement PACE in their electronic passports [15]; however, for global interoperability, States should not implement PACE without implementing BAC until December 31, 2017 [10]. The States may choose only to implement PACE in the eMRTDs they issue starting from January 1, 2018 [10].

Active Authentication and Chip Authentication are two security methods that prevent cloning of the eMRTD chip. For both Active Authentication and Chip Authentication, a private key file resides in the eMRTD chip. However, these private keys are protected and cannot be extracted or retrieved from the chip but are used internally to prove that the chip is not cloned.

For Active Authentication, the inspection system first reads the file `EF.DG15` and extracts

Figure 3. An Estonian *residence permit card* sample issued from 2020-10-01 [16]

the public key corresponding to the private key in the chip. Then, if Elliptic-Curve Digital Signature Algorithm (ECDSA) based Active Authentication is used, it conditionally reads the file `EF.DG14`. The authenticity of the information in these files is provided by the signed digest value of these files. Next, the inspection system sends a challenge to the chip, and the chip creates the signature of this challenge using its private key. This signature is then sent back to the inspection system, and the inspection system verifies that the chip's private key created the signature. Some countries, such as Turkey and Germany, do not support Active Authentication in their passports because of challenge semantics. *Challenge semantics* means that an inspection system could send short messages to the chip such as the current time and location and track the document holder. On the other hand, Chip Authentication uses Diffie–Hellman Key Exchange (DH) protocol to create new session keys and verify that the chip is not cloned. Therefore Chip Authentication is not vulnerable to this type of attack.

Terminal Authentication is a security method that protects sensitive data from unauthorized access. In the context of eMRTDs, the sensitive data is defined as the fingerprint and iris data stored in `EF.DG3` and `EF.DG4` files, respectively [1]. However, this security method requires a new PKI, namely, the Authorization PKI [11]. In this PKI, the States have a Country Verifying Certification Authority (CVCA). This CVCA issues a Self-Signed CVCA Certificate (C_{CVCA}) that in turn issues Document Verifier Certificates (C_{DV}s) for other States. The C_{DV}s issue certificates for inspection systems of this State, and these

inspection systems can obtain access to sensitive data by first sending the certificate chain that begins with the C_{CVCA} public key stored in the chip and ends with the certificate issued for the inspection system to the chip, and next by signing the challenge sent back by the chip using the certificate issued for itself.

1.1.1 Estonian Residence Permit Cards

Contactless smart card chips containing eMRTD applets are also present on Estonian *residence permit cards* issued since 2011 [5]. Since then, there have been three generations of chip platforms used in Estonian *residence permit cards*. These are:

1. `jTOP SLE66` platform chips issued since January 1, 2011;
2. `jTOP SLE78` platform chips issued since October 17, 2014, and
3. `IDEMIA` platform chips issued since December 3, 2018 [17].

Following the EU regulations [18], these platforms provide contactless smart card chips that contain eMRTD applets. The first and second generations of residence permit cards contained two different chips, one contact chip, and one contactless chip. The contact chip can be used to create digital signatures and to authenticate online. This contact chip also contains personal data files that store information about the individual such as name, surname, date of birth, and about the document, such as document number and date of expiry. Today, these data files are used for loyalty programs in certain stores, where the necessary information is read from the card and is used to provide customer benefits [19].

The latest, third-generation cards built on the `IDEMIA` platform only contain one chip. This single dual-interface chip in the third generation cards can communicate with contact and contactless readers. As mentioned earlier, starting from August 2, 2021, *identity cards* issued to Estonian citizens will also contain eMRTD applets [6].

1.2 Multi-Factor Authentication Solution

This work proposes an open-source automated biometric system. A high-level description of the solution is given in the flowchart shown in Figure 4. This solution can perform multi-factor authentication and be used as an access control system. This solution uses two hardware components—a contactless smart card reader and a camera. The camera is used to scan the MRZ of the document and capture a document holder's facial image. The card reader is used to communicate with the chip inside the document. The solution verifies the authenticity of the data stored in the chip by performing necessary cryptographic checks. If

the document supports, cloning detection is also performed. Optionally, a database module we included in this solution can be used. This database module creates a list of documents that can later be checked by the solution to only allow the documents in this list. Lastly, a face recognition module verifies that the document holder is the rightful owner of the document. If the eMRTD passes checks provided by the methods mentioned above and the picture of the individual matches the one read from the chip, the system grants access to the user.

1.2.1 Related Work and Software

Several software solutions try to achieve similar goals as the solution proposed in this thesis. However, we note that none of the solutions described below implement the functionality provided by our solution.

JMRTD: An Open Source Java Implementation of Machine Readable Travel Document is by far the most complete open-source project related to eMRTDs. The first steps of this project were made in 2006 as part of a research project at Radboud University in Nijmegen. Since then, JMRTD has become a Java Application Programming Interface (API) for accessing ICAO-compliant eMRTDs and Electronic Machine Readable Passports (eMRPs). Since version 0.5.x Java Implementation of Machine Readable Travel Document (JMRTD) focuses on providing the low-level building blocks for ICAO compliant communication and does not provide a complete solution for higher functionality (such as Passive Authentication) [20]. JMRTD is licensed under the GNU Lesser General Public License (LGPL) and is available on [20].

InnoValor's ReadID is a closed source commercial product based on JMRTD and provides a complete solution to verify eMRTDs. This mobile application can perform Optical Character Recognition (OCR), BAC, PACE, Active Authentication, Chip Authentication, and Passive Authentication using a static list of C_{CSCA}s; however, it does not support face recognition. This demo application can be used for free. ReadID demo application is available on [21].

RFIDIOt is a collection of Python scripts that allow interaction with Radio-Frequency Identification (RFID) technologies. These scripts are written in Python 2, which has reached End-of-life (EOL) as of January 1, 2020. It provides scripts for communicating with eMRTDs. It can read, write or clone eMRTDs and can parse the files such as

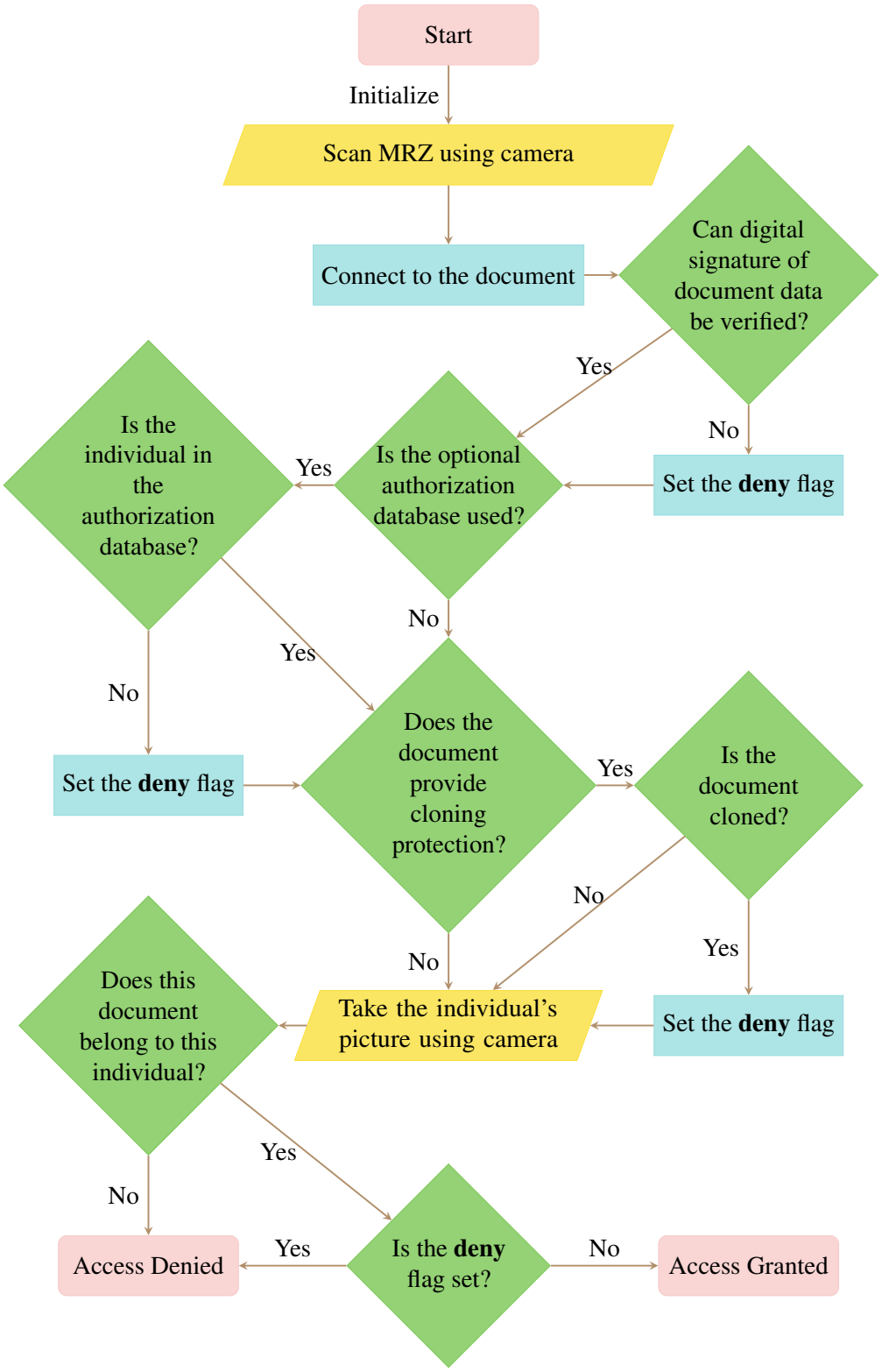


Figure 4. High-level description of "eMRTD Face Access"

`EF.DG2`. It can perform BAC. However, unfortunately, it does not perform OCR, Passive Authentication, Active Authentication, Chip Authentication, or face recognition. It is under GNU General Public License (GPL) v2.0 or later and is available on [22].

OpenPACE is a library that can perform PACE, Terminal Authentication, Chip Authentication, and Passive Authentication. This library is built on top of OpenSSL and is written in C, and has bindings for Python, Ruby, Go, Java and JavaScript. It does not perform OCR, BAC, Active Authentication, or face recognition. OpenPACE is licensed under the GPL v3.0 and is available on [23].

animamea is another library written in Java that can perform PACE, Terminal Authentication, and Chip Authentication and, unlike OpenPACE, can perform BAC. However, it does not perform OCR, Active Authentication, or face recognition. It is licensed under GPL v3.0 and is available on [24].

The pyPassport and ePassport Viewer are two libraries implemented in Python 2 to allow interaction with eMRTDs. They can perform BAC, Active Authentication, and Passive Authentication; however, they do not perform OCR, Chip Authentication, or face recognition. The project was last updated in 2013. It is licensed under GPL v3.0 and is available on [25].

Bachelor's Thesis [26] by Melkus implements an Android application written in Java that uses JMRTD and Tesseract libraries. It can perform OCR, BAC, RSA-based Active Authentication and show the mandatory files inside an eMRTD. However, this project does not perform face recognition, nor does it support Chip Authentication or ECDSA-based Active Authentication. It does not contain any license information and is available on [26].

Master's Thesis [27] by Vošček implements an Android application written in Java that uses JMRTD and Tesseract libraries. It can perform OCR and can show the image from inside an eMRTD, and perform face recognition. It does not support any security methods except for BAC, notably lacking, Passive Authentication, Active Authentication, and Chip Authentication. It does not contain any license information and is available on [27].

Master's Thesis [28] by Grelland implements an Android application written in Java, "MRTD Inspector," using various open-source libraries, including JMRTD and Tesseract. It can perform OCR, BAC, PACE, Chip Authentication, and Terminal Authentication; however, it does not perform Active Authentication or facial recognition. The software solution described in the thesis has not been made publicly available.

Master's Thesis [29] by Heide implements a server-backed proof-of-concept Android application "eMRTDInspectorPoC" and the corresponding server application "emrtdinspectorpocserver" to remotely inspect eMRTDs using the mobile device as a proxy. Both of these applications are written in Kotlin. It can perform OCR, BAC, PACE, Chip Authentication, Terminal Authentication, and Active Authentication; however, it does not perform facial recognition. It uses JMRTD as the back-end, and it incorporates work by Grelland [28]. The software solution described in the thesis has been publicly available, but unfortunately, it is not accessible anymore.

2. Implementation Details

This Chapter provides information regarding the details of this solution. The solution has to perform several tasks. First, the solution has to read or scan the MRZ of a document. Then, the scanned MRZ has to be used to create a communication channel with the chip contained in the document. Next, several security methods implemented in eMRTD chips are used to verify whether the chip is authentic and the files contained are unchanged and whether the chip is cloned. Next, whether the certificate in the document's chip is revoked by the issuing State must be checked. Next, facial recognition technologies that use the facial image in the eMRTD chip and the facial image of the individual that presents the document are used to find whether the individual presenting the document is the rightful owner of the document. Finally, optionally, a list of authorized users is used to determine if this individual has access to the system that this solution protects.

The automated biometric system presented in this work has been implemented in Python 3 and is called "eMRTD Face Access." The source code is licensed under the Massachusetts Institute of Technology (MIT) License and is available on GitHub[1]. To implement its functionality, several open-source libraries are used. The solution also provides a Graphical User Interface (GUI) to show the process of the inspection. The introduced solution uses eMRTDs and verifies a person using two of the three authentication factors. The three authentication factors are "*something you know*," "*something you have*," and "*something you are*" [30]. The authentication factors used are "something you have," which is the eMRTD, and "something you are," which is the facial biometric identifier.

2.1 Machine Readable Zone (MRZ) Scanning Using Optical Character Recognition (OCR)

In this solution, *OCR* technologies had to be used to read the MRZ of the eMRTD to create the necessary session keys for access control methods implemented in eMRTDs. *OCR* uses a set of algorithms that recognize text in images. There are many types of OCR techniques, and for MRZ recognition, we have assessed several solutions to find the best-suited solution.

[1] `https://github.com/Fethbita/eMRTD_face_access`

One of the most popular open-source OCR engines is Tesseract[2]. Tesseract performs very well in complex character recognition tasks [31]. For example, on number plate recognition tasks, Tesseract accuracy is 70%, whereas commercial OCR tool Transym OCR accuracy is 47% [32]. Karasu and Bağtan in [33] compare three open-source OCR engines, Tesseract, CuneiForm, and GOCR, on English and Turkish datasets and find that Tesseract gives the best results. It is important to note that CuneiForm has not been actively maintained since April 9, 2011[3], and GOCR has not been actively maintained since October 15, 2018[4]. Moreover, in recent years Tesseract received a significant update and started using a new OCR engine based on Long Short-Term Memory (LSTM) neural networks since version 4.0 [34], considerably increasing accuracy [35].

We chose to use tesserocr[5], a Python wrapper around Tesseract C++ API, for this solution. Another library that also provides Tesseract API in Python is pytesseract[6]; however, since this library is a wrapper around the command-line utility Tesseract, the performance of this tool was not adequate. By default, Tesseract includes models trained on about 400 000 lines of text containing 4 500 different fonts [36].

In our solution, to locate the MRZs in images, a naive approach was used. The solution separates the Tesseract output, which are the characters found in the image in plain text format, into list items from their newline characters. Then it finds the character count of each line. If three lines contain 30 characters or two lines contain 36 or 44 characters (for TD1, TD2, and TD3 document sizes, respectively), these lines are taken and given to the check digit verifier. We have implemented check digit verifiers for TD1, TD2, TD3, Machine Readable Visa (MRV)-A, and MRV-B documents; but, our solution does not accept MRV-A and MRV-B documents' MRZs because these documents do not contain eMRTD chips and therefore are not usable in our solution. If the MRZ lines have correct check digits, the MRZ is considered correctly read. It is also important to note that older generation Estonian *residence permit cards* had document type character of `R`, whereas, for TD1 documents, only `A`, `C`, and `I` are allowed [2]. We work around this by accepting the document type `R` for TD1 document MRZs.

The characters used in the MRZ fields are printed using the OCR-B font, and only a subset of characters can be used in the MRZs [37]. Therefore we found that a model that was only trained on OCR-B font was needed. To solve this problem, we tested three different models trained only on OCR-B fonts. These were the models from Doubango Tele-

[2] `https://github.com/tesseract-ocr/tesseract`
[3] `https://launchpad.net/cuneiform-linux`
[4] `http://jocr.sourceforge.net/`
[5] `https://github.com/sirfz/tesserocr`
[6] `https://github.com/madmaze/pytesseract`

com's `ultimateMRZ-SDK` software[7], Shreeshrii's `tessdata_ocrb`[8], and DaanVanVugt's `tesseract-mrz`[9]. To test these models, we have handcrafted a small MRZ test dataset that includes MRZs pictures taken in different lighting conditions. This small test dataset contains 708 images of size `640x480` pixels. As a result:

- Doubango Telecom's model found 514 MRZ resembling strings in 152 seconds;
- Shreeshrii's `float` model found 412 MRZ resembling strings out of 708 pictures in 524 seconds;
- Shreeshrii's `int` model found 413 MRZ resembling strings in 155 seconds, and
- DaanVanVugt's model found 92 MRZ resembling strings out of 708 pictures in 80 seconds.

Based on these results, we have decided to use the model from Doubango Telecom's ultimateMRZ-SDK to perform OCR on MRZ images. Another approach could have been to train our model by creating images with OCR-B font and annotating the results; however, this is beyond the scope of this work.

(a) Before adaptive thresholding

(b) After adaptive thresholding

Figure 5. Reverse side of an Estonian *residence permit card* sample issued from 2020-10-01 [16] before and after adaptive thresholding

During the program's execution, the OpenCV[10] library is used to capture the camera feed, and the images from the camera feed are given to an adaptive thresholding function. This function removes the noise around the characters and outputs the image as black and white. This process is shown in Figures 5a and 5b. This image is then used as an input to Tesseract. In the main program, we run the MRZ scan on a different thread to not affect the camera's framerate. After each scan, the output is checked as explained above, and if it is not a valid MRZ, a new scan is started. Optionally the tool saves the picture and the resulting MRZ text in a directory specified by the user.

[7] `https://github.com/DoubangoTelecom/ultimateMRZ-SDK/tree/master/assets/models`
[8] `https://github.com/Shreeshrii/tessdata_ocrb`
[9] `https://github.com/DaanVanVugt/tesseract-mrz/tree/master/lang`
[10] `https://github.com/opencv/opencv`

2.2 Reading the eMRTD Data

To communicate with the smart card chip from within Python, we have chosen to use the pyscard[11] library. To start the reading eMRTD, a communication channel between the inspection system and the eMRTD chip has to be created.

While some eMRTDs may not require any Chip Access Control mechanism to read data from the eMRTD chip, most eMRTDs deployed in practice require Basic Access Control (BAC). Therefore, our solution supports the *BAC* mechanism. We have implemented BAC using the cryptographic primitives provided by the PyCryptodome[12] library.

As mentioned in Section 1.1, another Chip Access Control method, called PACE, also exists. However, in our solution, we do not implement PACE.

We have created an SM class that contains the keys and the communication channel in our solution. The UML diagram of this class is shown in Figure 6.

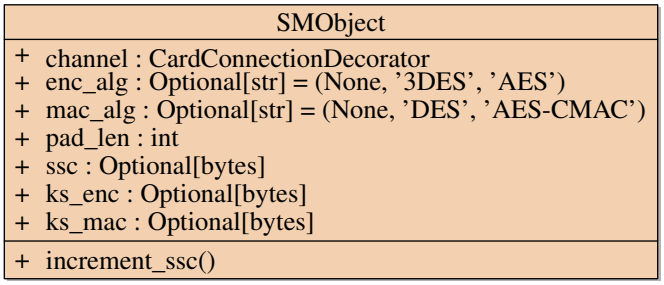


Figure 6. UML class diagram of Secure Messaging Object Class

2.2.1 Deriving the BAC Keys

BAC is the first Chip Access Control system that was standardized for eMRTDs. The BAC mechanism is specified in ICAO Doc 9303 part 11 [10]. BAC creates two session keys using the information from the MRZ and uses symmetric-key cryptography. One of the session keys is then used to encrypt and decrypt the message transmitted between the inspection system and the chip, and the other one is used to calculate the MAC. The MAC is used to find that the data integrity and authenticity of the message are valid. To create an SM channel using BAC, an inspection system first calculates the Secure Hash Algorithm 1 (SHA-1) digest of the [document number | its check digit | date of birth in YYMMDD format | its check digit | date of expiry in YYMMDD format | its check digit]. Check digit calculation is shown in Algorithm 1.

[11] https://github.com/LudovicRousseau/pyscard
[12] https://www.pycryptodome.org/en/latest/

Algorithm 1: Calculate Check Digit

Input: MRZ data
Result: Check digit for the given MRZ data

```
calculate_check_digit (mrz_data)
    values = {"<" : 0, "0" : 0, "1" : 1, "2" : 2, "3" : 3, "4" : 4, "5" : 5, "6" :
     6, "7" : 7, "8" : 8, "9" : 9, "A" : 10, "B" : 11, "C" : 12, "D" : 13, "E" :
     14, "F" : 15, "G" : 16, "H" : 17, "I" : 18, "J" : 19, "K" : 20, "L" : 21, "M" :
     22, "N" : 23, "O" : 24, "P" : 25, "Q" : 26, "R" : 27, "S" : 28, "T" : 29, "U" :
     30, "V" : 31, "W" : 32, "X" : 33, "Y" : 34, "Z" : 35}
    weights = [7, 3, 1]
    total = 0
    for i in range(len (mrz_data)):
        total = total + weights[i%3] × values[mrz_data[i]]
    return total%10
```

The first 16 bytes of this SHA-1 digest make up the key seed. A Key Derivation Function (KDF) is used on this key seed to create the Document Basic Access Keys K_{ENC} and K_{MAC}. Next, the chip and the inspection system exchange 16 randomly generated bytes, and these exchange messages are encrypted using the K_{ENC}. The MAC value is also calculated using K_{MAC} and is verified by the receiver. Next, these 16-byte values are XOR'd and are given to the KDF to generate KS_{ENC} and KS_{MAC}. These session keys are used to secure the communication from now on. During this exchange, an SSC is also established. Each time an APDU is sent between the inspection system and the chip, this SSC is incremented. The SSC is included in the MAC calculation, preventing communication reorder and replay attacks.

2.2.2 Obtaining the BAC Key on Estonian Residence Permit Cards

As explained in Subsection 1.1.1, on the latest generation Estonian *residence permit cards*, only one dual-interface chip is used. Therefore, we can use the contact chip to access both the eMRTD applet and the applet storing the personal data file of the document holder. The information stored in the personal data file can be used to construct the BAC key without the need to perform OCR. However, the default selected applet on the chip is the eMRTD applet. To select the other applet contained in the chip, we must use an Application Identifier (AID)[13]. This AID can be found from the specifications of that applet [38].

However, after the data files are read, selecting the eMRTD application proved a little tricky. One possible solution would have been to reset the card, which would implicitly select the default eMRTD application. Nevertheless, to manually select the eMRTD application, the

[13] 0xA0 00 00 00 77 01 08 00 07 00 00 FE 00 00 01 00

AID of the application that contained the eMRTD application was needed. This AID[14] was found from the IDEMIA LDS Applet Common Criteria [39], and after the selection of this application, the eMRTD application could be selected with its AID[15].

2.2.3 Reading the Data Files

The files contained in eMRTDs are read by first reading the first 4 bytes of a file, parsing the length of the file from these first 4 bytes, and then reading the rest of the file in chunks. The files `EF.DG1` and `EF.DG2` are always read. These two contain the MRZ information and the facial image. `EF.DG3` and `EF.DG4` are skipped because they contain sensitive biometric data (i.e., fingerprint and iris scan) and usually require Terminal Authentication if they exist. The `EF.SOD` file is also always read, and it contains certificates and information necessary for verification of signed data.

At first, we have used the pyasn1[16] library to decode the necessary objects from the `EF.SOD` file; however, it had problems reading these objects. Because of this reason, we have ported asn1-tiny-decoder[17] library written by Jens Getreu from Python 2 to Python 3 and replaced pyasn1 with this library.

2.3 Verification of Digitally Signed eMRTD Data

In eMRTDs, data stored in the files from `EF.DG1` to `EF.DG16` are protected from modification by a mandatory security method called Passive Authentication. Passive Authentication protects the contents of documents from changes and ensures the authenticity of data; however, it does not prevent exact copies of the documents. Therefore, other security methods such as Active Authentication and Chip Authentication should be used to prevent the cloning of eMRTD chips. Passive Authentication used to be the only required security method used in eMRTDs until ICAO Doc 9303 7th edition; however, with the ICAO Doc 9303 8th edition draft, access control is also mandatory [11].

Passive Authentication uses a PKI to verify the authenticity of the data stored in the eMRTD chip. The specifications for this PKI are in ICAO Doc 9303 part 12 [8]. In addition to the certificates introduced in Section 1.1 (C_{CSCA} and C_{DS}), this PKI also includes other types of certificates, such as:

[14] 0xA0 00 00 02 47 10 FF
[15] 0xA0 00 00 02 47 10 01
[16] `https://github.com/etingof/pyasn1`
[17] `https://github.com/getreu/asn1-tiny-decoder`

- *Master List Signer Certificate (C_{MS})*, which is used to sign a Master List (ML) that contains the C_{CSCA}s that are "trusted" by the issuing State;
- *Deviation List Signer Certificate (C_{DLS})*, which is used to issue Deviation Lists (DLs). DLs describe a set of documents with defects; but, it does not apply to individual documents or a small number of documents. In such cases where a small number of documents are defective, the issuing State should recall these documents through other means. DL is issued for thousands of documents if they are all defective, and
- Certificate Revocation Lists (CRLs) are used to revoke any certificate issued by the CSCA. Only the latest C_{CSCA} can issue CRLs, and these CRLs also include the certificates issued by older C_{CSCA}s. This makes the PKI simpler.

ICAO Public Key Directory (PKD) is a PKD that allows States to distribute necessary objects for eMRTD PKI. These include C_{CSCA}s, C_{DS}s, CRLs, C_{MS}s, and MLs, C_{DLS}s, and DLs. Participation in ICAO PKD requires a registration fee and an annual fee. As more countries participate in the ICAO PKD, the price of these fees goes down. As of 2021, the registration fee is 15 900 USD, and the annual fee is 26 282 USD [40]. The downside of not being a participant of ICAO PKD is to do bilateral agreements on a country-to-country basis. As of May 2021, ICAO PKD has 77 member States [41].

Estonia is not a member of ICAO PKD, so Estonian CSCA certificates have to be downloaded from the website of Estonian CSCA[18]. Some of the Estonian C_{CSCA} and CSCA link certificates are in the ICAO PKD because some other States trust these certificates. However, these MLs are not always up to date.

In our solution, we provide an option to use extra C_{CSCA}s placed in the `certs/csca_certs` folder or a folder specified in the `--certs` command-line argument. In addition to the certificates placed in these folders, we create a store of trusted certificates using the MLs from the ICAO PKD. To parse these MLs, we used the NFCPassportReader `extract.py` script[19] written by Andy Qua. However, we have made significant changes to this script. First, we use the python-ldap[20] library to parse the LDAP Data Interchange Format (LDIF) files downloaded from ICAO PKD instead of regex and OpenSSL subprocesses. Second, the outputs of this process are directly loaded into the pyOpenSSL[21] X509 objects instead of running the OpenSSL[22] command-line tool as a subprocess. These changes simplify the script and lower the processing time of MLs from 60.15 seconds to 0.73 seconds on a

[18] `https://pki.politsei.ee/`

[19] `https://github.com/AndyQ/NFCPassportReader/blob/bad7ff8c812595389bb271bd31cedf4a6e44edb9/scripts/extract.py`

[20] `https://github.com/python-ldap/python-ldap/tree/python-ldap-3.3.0`

[21] `https://www.pyopenssl.org/en/stable/`

[22] `https://www.openssl.org/`

personal laptop. These numbers are the average of 100 consecutive runs.

2.3.1 Special Handling of Deviant Estonian eMRTDs

The Estonian Police and Border Guard Board published a DL on June 26, 2018, containing 169 422 document numbers from the first and the second generations of Estonian *residence permit cards* [17], with the reason set to `id-Deviation-LDS-SODSignatureWrong`. We have found out that the authenticity of the data stored in the eMRTDs of the documents listed in this DL cannot be verified because the certification path from a Trust Anchor (C_{CSCA}) to the C_{DS} cannot be built and validated.

We have reached out to the Estonian CSCA about this problem, and we were told that the problem was that the C_{DS}s included in the eMRTD chips, in their `notBefore` and `notAfter` fields had a 12-hour negative offset compared to the original C_{DS} that was signed. We have worked around this by adding 12-hours in the C_{DS} that are stored in these deviant documents. This workaround is applied only if the issuing country is Estonia and the document number is in the published DL.

Another issue with Estonian *residence permit cards* that belong to the `jTOP SLE66` platform was that the `EF.SOD` specifies that the file digest values are calculated using the *SHA-256* function during the verification of signed data, but in practice, the *SHA-1* function is used [17]. In our solution, we work around this by comparing the Answer to Reset (ATR) of the documents with the ATR of this generation of documents. If they match, the *SHA-1* function is used to calculate the file digest values. However, all the `jTOP SLE66` platform documents issued in Estonia have already expired by the end of 2019 [17].

2.3.2 Issues with Estonian CSCA

We have also realized two other issues with the certificates of Estonian CSCA[23], these were:

- Estonian CSCA maintained 1 CRL for each CSCA certificate that was not expired (2012, 2015, 2016, and 2019). However, this is non-compliant with ICAO specifications Doc 9303. A CSCA should only have 1 CRL at any point, signed with the last C_{CSCA} of that CSCA. This CRL must include all other certificates created by that CSCA, including old C_{CSCA} (see Section 4.3 in [8]).
- For C_{CSCA}, a name change is allowed in the context of eMRTD PKI. However, if a

[23] `https://pki.politsei.ee/`

name change is done, the link certificate that conveys the key rollover and CSCA name change should include the `NameChange` extension (`id-icao-mrtd-security-extensions-nameChange(2.23.136.1.1.6.1)`) [8]. However, the Estonian link certificates did not include this extension.

These issues were pointed out to the Estonian CSCA, who have since resolved them.

2.4 Revocation Checks

Our solution performs revocation checks of C_{DS}s contained in the eMRTD chips. These revocation notices are published through the use of CRLs by the issuing States. In our solution, the CRLs are loaded from the ICAO PKD. In addition, the command-line flag `--crls` can also be used to provide additional CRLs.

2.4.1 Online Revocation Check for Estonian Documents

In addition, in Estonia, the Estonian Police and Border Guard Board provides a website[24] where anyone can check if an entered document number belongs to a valid document. Through this service, expired, lost, or stolen documents can be identified. This service has multiple defined reply messages, and these are: "*The document is valid*," "*The document is invalid*," "*The document has not been issued*," and "*The document is a specimen*." This service exists because CRLs can revoke C_{DS}s that are used to sign the data in the chips of documents, but not individual documents. The recommended usage period for C_{DS} private keys is three months or signing 150 000 travel documents, whichever is sooner [42]. For this reason, if C_{DS}s are revoked, they will affect hundreds of thousands of documents.

After successfully verifying the eMRTD chip, our solution uses this service to determine whether a given Estonian document has not expired and has not been declared lost or stolen.

2.5 Clone Detection

ICAO has standardized two security methods that can be implemented by eMRTDs to implement clone prevention. These are Active Authentication and Chip Authentication, and they both use a private key embedded in the chip. These keys cannot be retrieved and are only used by the chip internally to perform some cryptographic operations proving that these private keys are inside the chip.

[24] `https://www2.politsei.ee/en/teenused/inquiries/`

2.5.1 Chip Authentication

If the digests of the EF.DG14 file exist in the SO_D, it might contain SecurityInfos structures that might mean that the Chip Authentication mechanism is supported. If present, the Chip Authentication public key (and optionally elliptic curve domain parameters) are read. The chip might support more than one algorithm for Chip Authentication, and in these cases, the inspection system is free to choose any of the supported. Next, the inspection system creates an ephemeral public key pair in the same domain as the Chip Authentication public key and sends this key to the chip. If the card can successfully calculate the shared secret using the key agreement algorithm (DH or Elliptic-Curve Diffie–Hellman) and use this shared secret to compute new session keys, an OK reply is returned. From here onwards, the new session keys are used. To prove that the Chip Authentication public key is authentic and unchanged, Passive Authentication must be used. As a result of this process, the inspection system can ensure that the chip is not cloned. It also provides strong Session Keys (KS_{ENC} and KS_{MAC}). In our solution, however, the Chip Authentication algorithms that use DH are not supported. For ECDSA-based Chip Authentication, M2Crypto[25] library is used.

2.5.2 Active Authentication

If the digest of EF.DG15 is listed in the SO_D, it signals that Active Authentication is supported. The EF.DG15 file contains Active Authentication public key. For Active Authentication, the chip, using the corresponding private key, signs a challenge sent by the inspection system. The inspection system can later verify this signature using the Active Authentication public key. Since the Active Authentication public key is in EF.DG15 and this file is protected by Passive Authentication, the public key cannot be tampered with. If the chip uses ECDSA-based Active Authentication, the ActiveAuthenticationInfo SecurityInfo structure must be present in EF.DG14 and be used for Active Authentication. This entry contains the Object Identifier (OID) of the hashing algorithm used for the Active Authentication. However, if the document uses RSA-based Active Authentication, the hashing algorithm used is found from the trailer bytes attached in the response. Our solution supports both ECDSA and RSA-based Active Authentication protocols. For ECDSA, the M2Crypto library is used.

[25] https://gitlab.com/m2crypto/m2crypto

2.6 Authorized Users List

We have also created an optional authorization list for this program. The database is a small JSON file located in the db directory. This database consists of the MRZs of authorized documents. That is why each document has to be manually enrolled before usage.

However, instead of the complete MRZs, some information from the MRZ, such as name, surname, date of birth, sex, and nationality, could also be used to identify a person. However, in the unlikely case where two persons with the same name, surname, sex, and nationality are born on the same date, it would allow both of them to be authorized.

Furthermore, the latest generation of Estonian *residence permit cards* also contain the personal ID code in the optional data elements of MRZs [16]. For an Estonian eMRTD only use case, this would be the best option to store in the database since it is unique per person.

If these pieces of information (name, surname, date of birth, sex, and nationality or ID code) are used, a new person could be authorized by entering this data, and it would prevent manually enrolling the document first or the tedious work of typing the MRZ by hand. It would also work for any valid eMRTD that a person possesses.

To build this database, the TinyDB[26] library is used. There is an accompanying script that can be used to add or remove users to this database. However, this script does not verify the signed data and does not perform Active Authentication, Chip Authentication, or other security checks. So we recommend running the main program with the document first to verify the authenticity of the document before adding it to the authorized list. The flowchart of this builder can be seen in Figure 7.

2.7 Face Recognition

In this work, face recognition is used as the second factor for authentication. First, we get the facial image contained in the eMRTD from the `EF.DG2` file and take a picture of the individual using the camera connected to the computer. And then, the face recognition module is used to find the similarity score of the two images. The face recognition module works in the following way:

- First, a bounding box around the face is drawn;

[26] `https://tinydb.readthedocs.io/en/latest/`

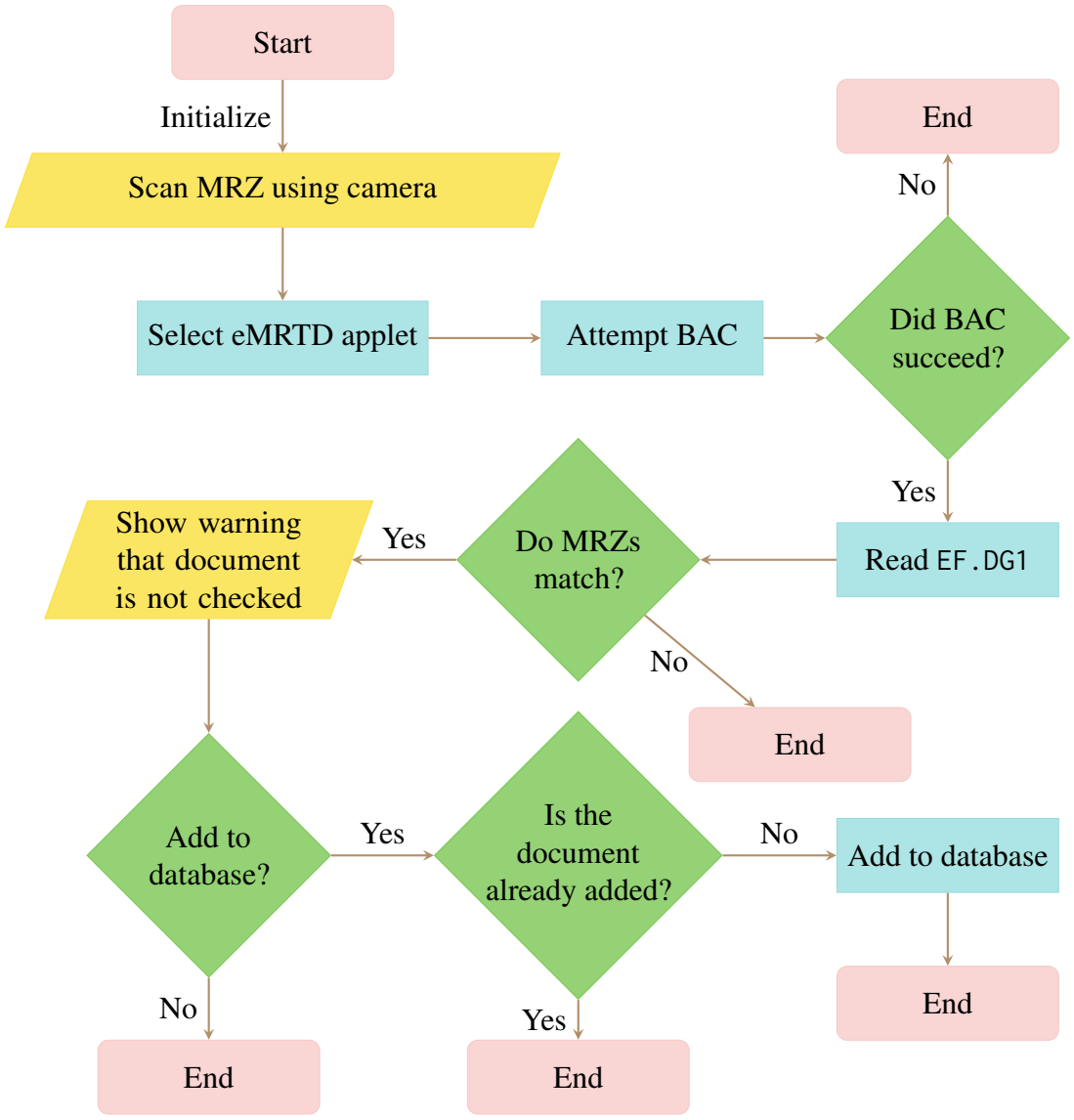


Figure 7. Flowchart of database builder

- Next, encoding vectors corresponding to these faces are obtained, and
- The distance between the vectors is found, and a decision threshold is applied.

A distance of less than or equal to the decision threshold means that the two images are of the same person. However, if the distance is greater than the decision threshold, it is concluded that the images do not belong to the same person.

The images contained in the chips of documents that we tested had images of different sizes. These were:

- `jTOP SLE66` platform chips contained images of size `480x640` pixels. These images had the background portion visibly compressed.

- jTOP SLE78 and IDEMIA platform chips contained images of size 480x640 pixels.
- Turkish *passport* and *identity card* chips both contained an image of size 240x320 pixels.
- Latvian *passport* chips contained an image of size 413x531 pixels.
- Latvian *identity card* chips contained an image of size 238x305 pixels.

As required by ICAO, the facial images stored in the eMRTD chips should have an Inter Eye Distance (IED) of at least 90 pixels [43]. We have found the Latvian *identity card* to have an IED of approximately 69 pixels, the Turkish *identity cards* to have an IED of approximately 62 and 55, and the Turkish passport to have and IED of approximately 63 pixels. The sizes of these images do not conform to the standards. All the other documents we tested had an IED of 125 or more.

2.7.1 Face Detection

To find the bounding boxes around faces on images, we have decided to use the face detector network of the OpenCV[27] library. As explained in [44], the OpenCV face detector model was created with SSD [45] using a ResNet-10 [46] like architecture as a backbone. dlib[28] also implements two face detectors using different methods (Histogram of Oriented Gradients (HOG) face detector and Convolutional Neural Network (CNN) face detector) [47]. However, dlib face detectors are trained on facial images with a minimum size of 80x80 pixels, and they have difficulty detecting faces in images with sizes smaller than that [48]. Because the OpenCV face detector can detect faces in images with small sizes and detect faces in images with non-frontal faces, we found the OpenCV face detector to perform better than the face detectors in the dlib library.

Performance

We have tested the face detection part of the module using all three models mentioned above on a small test dataset the author of this work has created. This test dataset contains 333 facial images of the author of size 640x480 pixels. The pictures vary from very dark images to images where the author looks sideways or wears a mask. This process was repeated five times, and the total time taken was averaged. As a result:

- OpenCV face detector network successfully found faces on all 333 images in 7.3 seconds;
- dlib HOG face detector network found faces on 326 images in 14 seconds, and

[27] https://github.com/opencv/opencv
[28] http://dlib.net/

- dlib CNN face detector network found faces on 323 images in 241 seconds.

dlib HOG model failed on images where the author is wearing a mask or blocking half of his face with his hand or where the author is in motion and the image looks blurry. The dlib CNN model failed on an image where the author's eyes are out of the frame. Since the OpenCV face detector could detect faces in all the images in our dataset and the execution speed was faster than the other models, we decided to use the OpenCV face detector in our solution.

2.7.2 Face Matching

Face matching is a method that compares images that contain faces and finds a similarity/distance score. For this process, we have decided to use dlib's ResNet face recognition model. This model uses 29 convolution layers, trained from scratch on a dataset of about 3 million faces [47], and was inspired by the original ResNet paper [46]. This network transforms the input images into 128-dimensional vectors. The Python face_recognition[29] library provides an easy interface to this dlib function, and that is why it has been chosen to be used in our solution. Because this model was trained to project identities into non-overlapping balls of radius 0.6 [47], the decision threshold is generally chosen as 0.6. However, in our tests, a decision threshold of 0.6 sometimes resulted in false positives, and that is why we have lowered the decision threshold to 0.5 in our solution.

Performance

Using the face detection models mentioned earlier, we have tested the performance of the face matching part with the decision threshold of 0.5. All of the images in the test dataset were compared with the image read from the author's Estonian *residence permit card*. This image can be seen in Figures 9 and 11. This process was repeated five times, and the total time taken was averaged. As a result:

- When used with the OpenCV face detector network, the process took 22.5 seconds on average and found that the images belonged to the same person 315 times and did not belong to 18 times;
- When used with the dlib HOG face detector network, the process took 23.5 seconds on average and found that the images belonged to the same person 314 times and did not belong to 12 times and failed to find a face 7 times, and
- When used with the dlib CNN face detector network, the process took 256.1 seconds on average and found that the images belonged to the same person 313 times and

[29] `https://github.com/ageitgey/face_recognition`

did not belong to 19 times and failed to find a face 1 time.

Therefore we have decided to use the OpenCV face detector network with the ResNet face recognition model.

2.7.3 Liveness Detection

Liveness detection is a method to ensure that biometric identifiers such as facial images, fingerprints, iris scans are taken from a living person, and they are not faked using artificial methods. Since our solution does not perform liveness detection, the program can be tricked by showing a printed image of the rightful document holder or showing the image printed on the document. That is why, where needed, the biometric solution needs to be supervised by a security guard. A live recording of the gate might also be helpful in case of vandalism. In addition, the program can log images taken during the process if it is run with the command-line argument that enables this behavior.

2.8 Graphical User Interface

In creating the Graphical User Interface (GUI), we had several requirements. The GUI had to provide the possibility to see the individual's image read from the chip of the document, to see the captured image of the individual, to see the processes as they happen, and to give information in a clean interface.

To create the GUI, we have decided to use the PySimpleGUI[30] library. The first version of the GUI was implemented by adding smaller GUI loops between the long-running processes, such as file reading. However, even longer running processes such as taking a picture were done by running them in a different thread not to block the GUI. This, however, led to a low-quality code; the process was interrupted multiple times with different loops, and the long-running processes that were not run in a thread (such as reading `EF.DG2`) would block the GUI. Later, the GUI was rewritten, and the main program was run in a different thread. This thread communicates with the main GUI thread by writing events in the window, and these events are polled in the main thread.

To capture the facial image, the user has to press a button (`Enter` or `ESC`) physically, and these keypresses are communicated from the main thread to the child thread using a Queue and thread locks. This process is shown in Figure 8.

[30] `https://github.com/PySimpleGUI/PySimpleGUI`

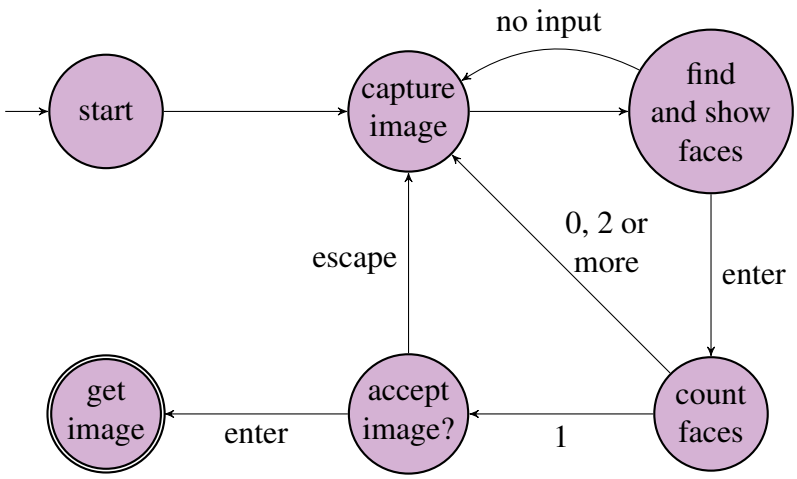


Figure 8. Transition of the camera capturing process

Figure 9 shows the program with every flag enabled (these are explained in Section 2.9). In this process, the Estonian *residence permit card* (issued on 2019-10-20) of the author of this thesis was used.

Figure 9. Program GUI when all flags (`-mrz`, `-online`, `-bio`, `--db db/db.json`) are used

Panel A shows the name and document number of the individual, panel B shows the instructions to run the program. Some examples are "*Please show the Machine Readable Zone (MRZ) of your document to the camera*," or "*Please place your document onto the card reader*." Panel C shows the completed processes or errors that occur during the

inspection of the document.

Panel D shows the debug panel for the program. The completed processes or errors are printed here. A slight side note about this debug panel is that, while it is convenient, there is a noticeable performance penalty if enabled. That is why an optional flag `-no-debug` is present. This flag creates the window without the debugging panel, and instead, the debugging information is printed on the terminal. For example, without the `-no-debug` flag, running the program up to the OCR process takes 9.81 seconds on average, and with the `-no-debug` flag, this time gets down to 3.96 seconds. An example program run using the `-ee` and the `-no-debug` command-line arguments is shown in Figure 10; also, in panel C, note that the line showing "Trying to capture MRZ information" has also disappeared.

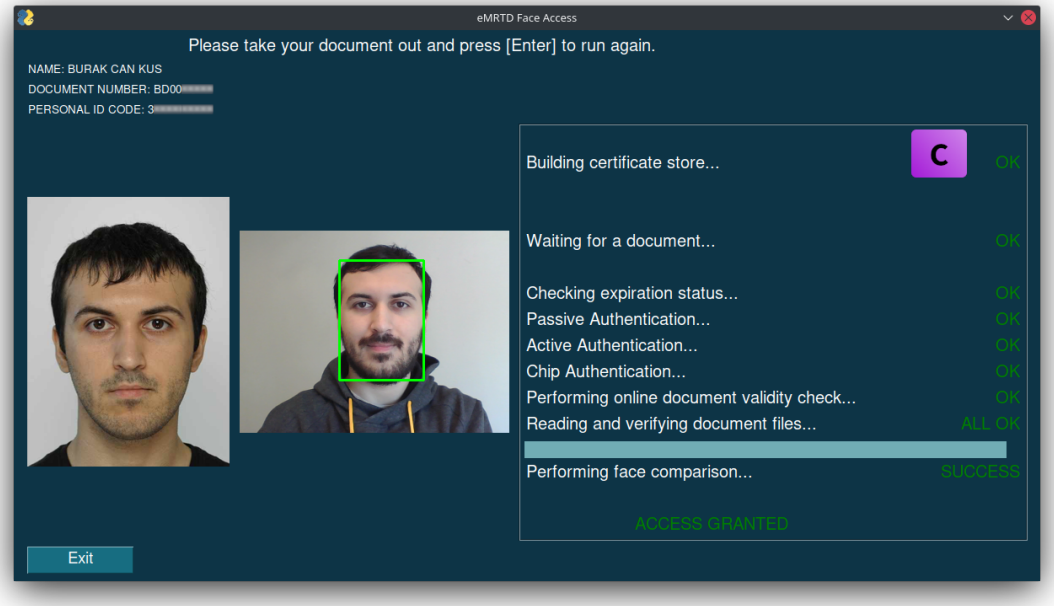


Figure 10. Program GUI when `-ee` and `-no-debug` flags are used

Since not all electronic identity documents support all of these security methods (Active Authentication, Chip Authentication) or if the program was run without a database or biometric part, these fields in panel C disappear. If there are any problems, for example, if the MRZ reading was wrong and the MRZ did not match the `EF.DG1`, these are shown on the right-hand side of panel C. Such an example is shown in Figure 11. This example is taken using the author's Turkish passport. However, we have edited the country field in the MRZ field and printed it on a piece of paper. This paper was then used in the OCR process. That is why panel C in Figure 11 shows an error in the line "Comparing Machine Readable Zone with the DG1." In addition, since the program was run without the optional authorization database, the database check was skipped, and since Turkish Passports do not support Active Authentication, this was also skipped. Lastly, the online document number

Figure 11. Program GUI with some flags and an error

check is only used for Estonian documents, as explained in Section 2.4. If at any point during this process, an unrecoverable error occurs (such as removing the document from the card reader), the program will show an error on panel B and start over.

The bottom of panel C also accompanies a progress bar that shows how the progress of a file read from the chip. This progress bar is visible in Figures 9, 10, and 11 at the bottom of panel C. To show this progress bar, the length of the file is by first reading a chunk of the file and, next, parsing the length of the file. The progress bar is updated each time a new chunk of the file is read. Thus the progress can be nicely shown.

2.9 Modes of Operation

The program has two different primary modes of operation. These are the `MRZ` mode (`-mrz` flag) and the `EE` mode (`-ee` flag). In the `MRZ` mode, the BAC key is constructed by capturing the MRZ with a camera and decoding it using OCR. However, in the `EE` mode Session Keys are created by reading the necessary files from the Estonian document through the contact interface as explained in Subsection 1.1.1, and this way, the OCR step is skipped. In addition, there are other flags or arguments to control the behavior of the program, these are:

1. the `-no-debug` flag disables the debug panel in the GUI and prints the logs in the terminal;
2. the `-online` flag downloads the Estonian CSCA certificates and CRLs from the Estonian CSCA[31] website;
3. the `-bio | -no-bio` flags specify whether the facial recognition process should be used to identify the cardholder;
4. the `--db` argument specifies a TinyDB database file (the creation of this database is explained in Section 2.6) and to perform authorization check;
5. the `--certs` argument specifies a directory to CSCA certificates;
6. the `--crls` argument specifies a directory to the certificate revocation lists folder, and
7. the `--output` argument saves the read document information in the specified directory.

```python
def program_logic(window: sg.Window):
    mrz = capture_mrz(window, -1)
    doc_num, birth, expiry, _, _, _ = parse_mrz_text(mrz)
    mrz_information = other_mrz(doc_num, birth, expiry)
    # Waiting for a document
    sm_object = SMObject(wait_for_card())
    # Select eMRTD Applet
    print("[+] Selecting LDS DF AID: A0000002471001...")
    aid = bytes.fromhex("A0000002471001")
    send(sm_object, APDU(b"\x00", b"\xA4", b"\x04", b"\x0C", Lc=nb(len(aid)),
    ↪  cdata=aid))
    # Secure Messaging
    establish_bac_session_keys(sm_object, mrz_information.encode("utf-8"))
    # Read EF.COM
    efcom = read_data_from_ef(window, sm_object, b"\x01\x1E", "EF.COM")
    ef_com_dg_list = parse_efcom(efcom)
    print(f"[i] DGs specified in EF.COM: {list(ef_com_dg_list.values())}")
```

Listing 1. An example code showing how to perform MRZ OCR, BAC connection and read a file from a document (error handling is not shown to not take up space)

The functionality provided by our solution can be used as a library to perform any of the processes explained in this Chapter. To show how to use the parts of this program, we have created a Small Demo application[32]. A small code snippet from this demo is shown in Listing 1. The first step (not shown) is to create a GUI window. In this small demo,

[31] https://pki.politsei.ee/

[32] https://github.com/Fethbita/eMRTD_face_access/blob/main/emrtd_face_access/small_demo.py

we have only included an image field and a `Multiline` element. However, as discussed earlier, this Multiline has a significant impact on the performance if `STDOUT` capture is done. In the main program loop, we do GUI-related functions; however, in another thread, we run the logic of the program. This main logic of the program that performs OCR, BAC, and reads the `EF.COM` file, is shown in Listing 1.

3. Conclusion

In this thesis, we have created an open-source solution that can be used for multi-factor authentication purposes. This solution is easily deployable on the entrances to buildings or self-checkout machines where strong identity verification is needed. This solution performs OCR to scan the MRZ of the document, performs the necessary cryptographic checks to verify the authenticity of the document, uses a face recognition library to match the individual's image to the image stored in the document's chip, and grants or denies access to the individual depending on the results of these processes.

While developing the solution, we have identified several issues in the Estonian eMRTD ecosystem and contacted the authorities regarding these issues. Furthermore, when possible, we have implemented workarounds to verify the deviant Estonian documents.

Through this research, we have shown that implementing open-source tools for multi-factor authentication is feasible, and since with the new EU regulation [6], the use of these documents will increase, there is a significant opportunity to implement these systems to make use of the security features provided by eMRTDs.

Our solution does not implement every feature of ICAO Doc 9303 (e.g., DH keys for Chip Authentication). However, we have tested this solution with Estonian *residence permit cards*, Turkish *identity cards* and *passports*, Latvian *identity cards* and *passports*, and we expect it to work with documents issued by other countries as well.

Possible future improvements to the biometric functionality would be to add liveness detection checks to verify that the facial image is taken from a living person and is not faked.

Bibliography

[1] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 1: Introduction*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p1_cons_en.pdf.

[2] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 5: Specifications for TD1 Size Machine Readable Official Travel Documents (MROTDs)*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p5_cons_en.pdf.

[3] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 6: Specifications for TD2 Size Machine Readable Official Travel Documents (MROTDs)*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p6_cons_en.pdf.

[4] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 4: Specifications for Machine Readable Passports (MRPs) and other TD3 Size MRTDs*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p4_cons_en.pdf.

[5] Police and Border Guard Board. (2021). "Residence Card", [Online]. Available: https://www2.politsei.ee/en/nouanded/residence-card.dot.

[6] European Parliament, Council of the European Union, *Regulation (EU) 2019/1157 of the European Parliament and of the Council of 20 June 2019 on strengthening the security of identity cards of Union citizens and of residence documents issued to Union citizens and their family members exercising their right of free movement (Text with EEA relevance.)* Legislative Body: CONSIL, EP, July 12, 2019. [Online]. Available: http://data.europa.eu/eli/reg/2019/1157/oj/eng.

[7] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 10: Logical Data Structure (LDS) for Storage of Biometrics and Other Data in the Contactless Integrated Circuit (IC)*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p10_cons_en.pdf.

[8] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 12: Public Key Infrastructure for MRTDs*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p12_cons_en.pdf.

[9] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 9: Deployment of Biometric Identification and Electronic Storage of Data in eMRTDs*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p9_cons_en.pdf.

[10] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 11: Security Mechanisms for MRTDs*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p11_cons_en.pdf.

[11] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. 8th edition (unedited)*, January 8, 2021. [Online]. Available: https://www.icao.int/Security/FAL/TRIP/PublishingImages/Pages/Publications/Doc%209303%20Machine%20Readable%20Travel%20Documents%20%20%28Unedited%208th%20Edition%29.pdf.

[12] ISO/IEC JTC1 SC17 WG3/TF5, "Supplemental Access Control for Machine Readable Travel Documents", International Civil Aviation Organization, Technical Report, April 15, 2014, Release: 1.1, p. 43. [Online]. Available: https://www.icao.int/Security/mrtd/Downloads/Technical%20Reports/NEW%20TRs%20post%20TAG%2022/TR%20-%20Supplemental%20Access%20Control%20V1.1.pdf.

[13] Y. Liu, T. Kasper, K. Lemke-Rust, and C. Paar, "E-Passport: Cracking Basic Access Control Keys", in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, R. Meersman and Z. Tari, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1531–1547, ISBN: 978-3-540-76843-2. DOI: 10.1007/978-3-540-76843-2_30.

[14] Bundesamt für Sicherheit in der Informationstechnik, "Advanced Security Mechanisms forMachine Readable Travel Documents and eIDAS Token - Part 1 - eMRTDs with BAC/PACEv2 and EACv1", 53133 Bonn, Technical Guidelines TR-03110-1, February 26, 2015, Version: 2.20. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/BSI_TR-03110_Part-1_V2-2.pdf?__blob=publicationFile&v=1.

[15] European Commission, *Commission Decision C(2011) 5499 amending Commission Decision C(2006) 2909 laying down the technical specifications on the standards for security features and biometrics in passports and travel documents issued by Member States*, August 4, 2011. [Online]. Available: https://ec.europa.eu/home-affairs/sites/default/files/e-library/docs/comm_native_c_2011_5499_f_en.pdf.

[16] Estonian Police and Border Guard Board. (2020). "2020 - Residence Permit Card Sample", [Online]. Available: https://www.politsei.ee/en/instructions/residence-permit-card-sample.

[17] A. Parsovs, “Estonian Electronic Identity Card and its Security Challenges”, PhD thesis, University of Tartu, 2021, ISBN: 978-9949-03-571-7. [Online]. Available: https://dspace.ut.ee/handle/10062/71481.

[18] Council of the European Union, *Council Regulation (EC) No 380/2008 of 18 April 2008 amending Regulation (EC) No 1030/2002 laying down a uniform format for residence permits for third-country nationals*, Procedure number: 2003/0218/CNS, May 18, 2008. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2008/380/oj.

[19] D. M. Morgan, “Security of Loyalty Cards Used in Estonia”, Master’s Thesis, Tallinn University of Technology, Tallinn, May 29, 2017, 76 pp. [Online]. Available: https://digikogu.taltech.ee/en/Item/5c768e7f-4317-40aa-b38b-913097105f28.

[20] the JMRTD team. (2021). “JMRTD: An Open Source Implementation of Machine Readable Travel Documents”. License: LGPL, [Online]. Available: https://jmrtd.org/.

[21] ReadID. (2021). “ReadID Demo | Mobile Identity Verification”, [Online]. Available: https://readid.com/demo.

[22] A. Laurie. (2021). “python RFID / NFC library & tools”. License: GPL-2.0 or later, [Online]. Available: http://www.rfidiot.org/.

[23] F. Morgner and D. Oepen. (2021). “Cryptographic library for EAC version 2”. License: GPL-3.0, [Online]. Available: https://github.com/frankmorgner/openpace.

[24] T. Senger. (2021). “animamea”. License: GPL-3.0, [Online]. Available: https://github.com/tsenger/animamea.

[25] G. Avoine. (2021). “pyPassport”. License: GPL-3.0, [Online]. Available: https://sites.uclouvain.be/security/epassport.html.

[26] A. Melkus, “Electronic passport reading on OS Android”, Defended: 2014-02-06, Masaryk University, Faculty of Informatics, 2014. [Online]. Available: https://is.muni.cz/th/hop20/?lang=en.

[27] J. Vošček, “Identity verification based on ePassports”, Defended: 2014-06-25, Masaryk University, Faculty of Informatics, 2014. [Online]. Available: https://is.muni.cz/th/325238/fi_m/?lang=en.

[28] H. H. Grelland, “An eMRTD inspection system on Android”, 2016, Accepted: 2016-08-08T22:28:31Z. [Online]. Available: http://urn.nb.no/URN:NBN:no-54538.

[29] H. Heide, “Secure eMRTD Inspection on Android”, 2018, Accepted: 2019-02-07T23:01:16Z. [Online]. Available: http://urn.nb.no/URN:NBN:no-69701.

[30] P. Grassi, M. Garcia, and J. Fenton, *Digital Identity Guidelines*, en, June 22, 2017. DOI: 10.6028/NIST.SP.800-63-3.

[31] M. Shen and H. Lei, "Improving OCR Performance with Background Image Elimination", in *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, August 2015, pp. 1566–1570. DOI: 10.1109/FSKD.2015.7382178.

[32] C. Patel, A. Patel, and D. Patel, "Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study", *International Journal of Computer Applications*, vol. 55, pp. 50–56, 2012. DOI: 10.5120/8794-2784.

[33] K. Karasu and M. Bağtan, "Turkish OCR on Mobile and Scanned Document Images", in *2015 23nd Signal Processing and Communications Applications Conference (SIU)*, May 2015, pp. 2074–2077. DOI: 10.1109/SIU.2015.7130278.

[34] Tesseract Contributors. (2021). "Tesseract User Manual", [Online]. Available: https://tesseract-ocr.github.io/tessdoc/.

[35] R. Smith, "Developing Multilingual OCR at Google", in *International Workshop on Document Analysis Systems DAS 2016*, 2016. [Online]. Available: https://github.com/tesseract-ocr/docs/blob/4d62defbe606a2ae78113e14c374e4e418b2040e/das_tutorial2016/7Building%20a%20Multi-Lingual%20OCR%20Engine.pdf.

[36] Tesseract Contributors, *Fast integer versions of trained LSTM models*, original-date: 2017-09-11T17:11:20Z, May 12, 2021. [Online]. Available: https://github.com/tesseract-ocr/tessdata_fast.

[37] International Civil Aviation Organization, *DOC 9303. Machine Readable Travel Documents. Part 3: Specifications Common to all MRTDs*, 2015. [Online]. Available: https://www.icao.int/publications/Documents/9303_p3_cons_en.pdf.

[38] Estonian Information System Authority. (July 1, 2020). "ID-card documentation Estonia eID 2018 chip specification", [Online]. Available: https://installer.id.ee/media/id2019/TD-ID1-Chip-App.pdf.

[39] S. Mestiri, "LDS Applet V10 in BAC with CA and AA Configuration–Public Security Target Revision 1.0", February 6, 2018, p. 83. [Online]. Available: https://www.commoncriteriaportal.org/files/epfiles/st-2018_03.pdf.

[40] ICAO PKD Board, "PKD Fee Schedule 2021", January 29, 2021. [Online]. Available: https://www.icao.int/Security/FAL/PKD/Documents/PKDFinanceDocuments/B-Fin-191-PKD%20Fee%20Schedule%202021.pdf.

[41] International Civil Aviation Organization. (April 21, 2021). "ICAO PKD Participants", [Online]. Available: https://www.icao.int/Security/FAL/PKD/Pages/ICAO-PKDParticipants.aspx.

[42] International Civil Aviation Organization. (2020). "Document Signer", [Online]. Available: https://www.icao.int/Security/FAL/PKD/BVRT/Pages/DS.aspx.

[43] International Civil Aviation Organization, "Portrait Quality (Reference Facial Images for MRTD)", Technical Guidelines, 2018. [Online]. Available: https://www.icao.int/Security/FAL/TRIP/Documents/TR%20-%20Portrait%20Quality%20v1.0.pdf.

[44] OpenCV Contributors. (2019). "OpenCV Face Detector Training", [Online]. Available: https://github.com/opencv/opencv/blob/7de627c504a3b8aa16cdc3de56efdc358df4b061/samples/dnn/face_detector/how_to_train_face_detector.txt.

[45] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector", in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 21–37, ISBN: 978-3-319-46448-0. DOI: 10.1007/978-3-319-46448-0_2.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[47] D. E. King, *dlib-models*, February 11, 2017. [Online]. Available: https://github.com/davisking/dlib-models.

[48] V. Gupta. (October 22, 2018). "Face detection – OpenCV, dlib and deep learning (c++ / python)", Learn OpenCV, [Online]. Available: https://learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/.

Appendix

I. License

Non-exclusive license to reproduce thesis and make thesis public

I, **Burak Can Kus**,

1. herewith grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,
   Use of Electronic Identity Documents for Multi-Factor Authentication,
   supervised by Arnis Paršovs.
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Burak Can Kus
2021-05-14